Editor: Dana Simian

Proceedings of The International Conference on Applied Informatics

ICDD

IMAGINATION, CREATIVITY, DESIGN, DEVELOPMENT

May 29 - 31, 2025, Sibiu, Romania

Editor DANA SIMIAN

IMAGINATION, CREATIVITY, DESIGN, DEVELOPMENT

Proceedings of the International Conference
on Applied Informatics
ICDD

May 29th – 31th, 2025
Sibiu, Romania

Lucian Blaga University of Sibiu

Lucian Blaga University of Sibiu, 2025

Editor Dana Simian

All papers in this volume were peer review by two independent reviewers

ISSN-L 2069 – 864X

Associated Editor Laura Florentina Stoica

Proceedings of the International Conference on Applied Informatics, ICDD
May 29th – 31th, 2025, Sibiu, Romania

Copyright @ 2025 All rights reserved to editors and authors

Preface

This volume contains refereed papers presented within the International Conference on Applied Informatics, “Imagination, Creativity, Design, Development“ - ICDD 2025, which was held between May 29th – 31th, at the Faculty of Sciences, Lucian Blaga University of Sibiu, Romania.

The conference is mainly addressed to young researchers from all over the world. The conference gives the participants the opportunity to discuss and present their research on informatics and related fields (like computational algebra, numerical calculus, bioinformatics, etc.). The conference welcomes submissions of original papers on all aspects of informatics and related fields ranging from new concepts and theoretical developments to advanced technologies and innovative applications. Specific topics of the conference included but are not restricted to: Algorithms and data structures, Graph theory and applications, Formal languages and compilers, Cryptography, Modelling and simulation, Computer programming, Computer vision, Computer graphics, Game design, Data mining, Distributed computing, Artificial Intelligence, Service oriented applications, Networking, Grid computing, Mobile operating systems, Scientific computing, Software engineering, Bioinformatics, Robotics, Computer Architecture, Evolutionary Computing, Multimedia Systems, Internet Communication and Technologies, Web Applications, Machine Learning.

The conference has brought together participants from 4 countries (Bulgaria, Germany, India, Romania).

We thank all the participants for their interesting talks and discussions. We also thank the members of the scientific committee for their help in reviewing the submitted papers and for their contributions to the scientific success of the conference.

December 2025

Dana Simian
Conference Chair

Motto:

“There are no limits, only your imagination”

Scientific committee

Kiril Alexiev - Bulgarian Academy of Sciences, Bulgaria
Vsevolod Arnaut - Moldova State University, Republic of Moldova
Galina Atanasova - Angel Kanchev University of Rousse
Alina Barbulescu - Ovidius University of Constanta, Romania
Arndt Balzer - Technical University of Applied Sciences Würzburg-Schweinfurt, Germany
Lasse Berntzen - University of South-Eastern Norway
Peter Braun - Technical University of Applied Sciences Würzburg-Schweinfurt, Germany
Amelia Bucur - Lucian Blaga University of Sibiu, Romania
Stelian Ciurea - Lucian Blaga University of Sibiu, Romania
Nicolae Constantinescu - Lucian Blaga University of Sibiu, Romania
Daniela Danciulescu - University of Craiova, Romania
Lyubomyr Demkiv - Lviv National Polytechnic University and Robotics Lead at SoftServe, Ukraine
Oleksandr Dorokhov - Kharkiv National University of Economics, Ukraine
Dmytro Dosyn - Institute of Computer Science and Information Technologies, Lviv Polytechnic National University, Ukraine
George Eleftherakis - The University of Sheffield International Faculty, City College Thessaloniki, Greece
Michael Emmerich - Leiden Institute of Advanced Computer Science, Leiden University
Calin Enachescu -University of Medicine, Pharmacy, Science and Technology of Targu Mures, Romania
Ralf Fabian - Lucian Blaga University of Sibiu, Romania
Tobias Fertig - Technical University of Applied Sciences Würzburg-Schweinfurt, Germany
Stefka Fidanova - Bulgarian Academy of Sciences, Bulgaria
Ulrich Fiedler - Bern University of Applied Science, Switzerland
Adrian Florea - Lucian Blaga University of Sibiu, Romania
Teresa Gonçalves - University of Evora, Portugal
Andrina Granić - University of Split, Croatia
Katalina Grigorova - University of Ruse, Bulgaria
Piroska Haller - University of Medicine, Pharmacy, Science and Technology of Targu Mures, Romania
Daniel Hunyadi - Lucian Blaga University of Sibiu, Romania
Saleema JS - Chris University, Bangalore, India
Milena Lazarova - Technical University of Sofia, Bulgaria
Lixin Liang - Tsinghua University, Beijing, China
Suzana Loskovska - "Ss. Cyril and Methodius" University in Skopje, Republic of Macedonia

Rossitza S. Marinova - Concordia University of Edmonton, Canada
Gabriela Moise - Petroleum-Gas University of Ploiesti, Romania
G.Jose Moses - Raghu Engineering College Visakhapatnam, Andhra Pradesh, India
Mircea Musan - Lucian Blaga University of Sibiu, Romania
Mircea Iosif Neamtu - Lucian Blaga University of Sibiu, Romania
Elena Simona Nicoară - Petroleum-Gas University of Ploiesti
Grażyna Paliwoda-Pękosz - Cracow University of Economics, Poland
Camelia Pintea - Technical University Cluj-Napoca, Romania
Antoniu Pitic - Lucian Blaga University of Sibiu, Romania
Alina Pitic - Lucian Blaga University of Sibiu, Romania
Cristina Popirlan - University of Craiova, Romania
Anca Ralescu - University of Cincinnati, United States of America
Mohammad Rezai - Sheffield Hallam University, United Kingdom
Cosmin Sabo - Technical University of Cluj-Napoca - North University Center Baia Mare
José Saias - University of Evora, Portugal
Abdel-Badeeh M. Salem - Ain Shams University, Cairo, Egypt
Livia Sangeorzan - Transilvania University of Brasov, Romania
Soraya Sedkaoui - Khemis Miliana University, Algeria
Andreas Siebert - University of Applied Sciences Landshut, Germany
Ioan Silea - Politehnica University of Timișoara
Dana Simian - Lucian Blaga University of Sibiu, Romania
Petrica C. Pop Sitar - Technical University Cluj-Napoca, Romania
Lior Solomovich - Kaye Academic College of Education, Israel
Ansgar Steland - RWTH Aachen University, Germany
Florin Stoica - Lucian Blaga University of Sibiu, Romania
Laura Florentina Stoica - Lucian Blaga University of Sibiu, Romania
Detlef Streitferdt - Ilmenau University of Technology, Software Architectures and Product Lines Group, Germany
Grażyna Suchacka - University of Opole, Poland
Jolanta Tańcula - University of Opole, Poland
Milan Tuba - Trinity University, United States of America; Singidunum University of Belgrade, Serbia
Eva Tuba - Trinity University, United States of America; Singidunum University of Belgrade, Serbia
Oana Țicleanu - Lucian Blaga University of Sibiu, Romania
Anca Vasilescu - Transilvania University of Brasov, Romania
Dana Vasiloaica - Atlantic Technological University, Ireland
Sofia Visa - The College of Wooster, United States of America

Contents

Autonomous Navigation Solution Based on Embedded System and Real-Time Object Recognition ………………………… 9
Marian-Daniel Drăghici, Andrei Dăian

EmiNet ………………………… 22
Ștefan Eminovici

User-Centered AI: Improving Workflows through intelligent Chatbots ………… 29
Sophie Geisler, Peter M¨ohle, Marcel Wernisch, Felix Zorn

WebXR-Previs: Low-Cost System for Real-Time Previsualization and Broadcast Graphics………………………… 42
Felix Husac

Implementation and Development of a Rated Voting System………… 54
Giorgiana-Maria Marangoci, Alex-Andrei Rîpan, Stefan-Ioan Istina

“SmartGarden: An Economical IoT-Enabled System for Indoor Plant Surveillance and Irrigation" ………………………… 64
Alexandra Onose

TransitAI: An AI-Powered Conversational Assistant for Public Transportation Information Access ………………………… 79
Eduard-Alexandru Oprea, Elena-Luiza Buzatu, Ioana-Valeria Turcin

AI Platform for Real-Time Cyber Threat Detection ………… 93
Vlad-Matei Poienariu, Rares Muntean, Vlad-Stefan Alexandrescu

Multi-Task Learning vs. Individual Models in Scarce Data Scenarios - Stock Price Prediction ………………………… 107
Andrei Priboi

Game Engine Development: Research, optimization and performance enhancement ………………………… 119
Serkan Sadulov, Mustafa Mustafov

Evaluating Log Messages Using a Big Data Approach ………… 133
Tobias Schneider

Tecky ………………………… 146
Alexandru Emil Sofonea

Web-based System for Dynamic Text Rendering and Interactive Content Management in Full-screen Environments in the Music Industry ………… 156
Kristian P. Spasov, Martin S. Dzhurov, Serkan H. Sadulov

Organ 3D Model Reconstruction using Point Clouds …………………..…….…… 167
Steavu Matei-Cristian

Road Condition Classification and Predictive Maintenance Using (OBD-II) Data … 174
Tanya Teresse, Jarin Justin Victoria

Assessing Visual Tracking in Children with Special Needs: A Tool for Ergotherapists …..…………………………..…………………………….....… 182
Gokul Perumbayil Vijayakrishnan, Anagha Manikathuparambil Baby, Blesson Manjakunnel

List of authors …………………..………..…………..………………..................…… 191

Sponsors …………………………………………………………..…...…………… 196

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Autonomous Navigation Solution Based on Embedded System and Real-Time Object Recognition

Marian-Daniel Drăghici, Andrei Dăian

Abstract

This paper presents the design, implementation and experimental evaluation of a low-cost, embedded autonomous navigation platform that performs real-time traffic-sign recognition and line following. The solution integrates a Raspberry Pi single-board computer, a Hailo-8L neural accelerator and an Arduino microcontroller to split perception, decision and actuation tasks. The perception pipeline combines an optimized YOLOv8 detector for sign recognition with classic image-processing methods for robust line extraction; the two perception channels run in a unified GStreamer pipeline that feeds the decision module. Navigation logic translates visual cues into setpoints communicated to the Arduino by UART; the microcontroller executes a PID controller and enforces safety limits on PWM outputs. The system exposes a remote Tkinter dashboard connected through a Flask + Socket.IO server that provides live video, detection overlays and run-time parameter tuning. Experimental runs on a miniature track demonstrate reliable real-time operation, high detection precision for trained sign classes, and stable line tracking under varied illumination. The main contribution is a practical, reproducible integration of embedded AI acceleration with a vision pipeline and lightweight control logic, appropriate for education, prototyping and further research.

1. Introduction

Autonomous navigation at small scale is a useful research and teaching platform because it forces explicit design choices for perception, timing, resource allocation and safety while remaining affordable and reproducible. Current trends in autonomous systems combine neural perception with classical signal processing and modular control architectures; surveys and reviews highlight the importance of fusing efficient deep detectors with embedded accelerators in constrained environments. Recent literature surveys show the variety of algorithmic solutions and point out the practical value of hybrid systems that balance accuracy and latency for edge devices [3]. The YOLO family of detectors remains a widely used pragmatic choice for embedded detection due to its single-stage architecture and high throughput [1,2].

The present work pursues two concurrent objectives. The first is to demonstrate that a YOLOv8 detector, when properly trained and compiled for an NPU, can be integrated into a real-time pipeline on a Raspberry Pi paired with a Hailo-8L accelerator to achieve sign recognition without overwhelming the host CPU. The second objective is to show how classical image processing (rotation, smoothing, sharpening, color thresholding and morphological processing), tightly coupled to a simple centroid-based error calculation and a PID actuator, can deliver stable line following and intersection detection. The system design, pipeline implementation and tuning were developed and validated on a custom miniature track; important implementation details and parameter choices are described in the following sections. Many architectural and implementation choices summarized here follow the structure and experiments detailed in the bachelor thesis and implementation notes.

The remainder of the paper is organized as follows. Section 2 gives an overall architecture description. Section 3 describes the perception subsystem in depth: acquisition, preprocessing, line segmentation and neural inference for traffic signs. Section 4 explains the motion control, message formats and the integration between Raspberry Pi and Arduino. Section 5 presents the dashboard and network communication. Section 6 discusses test methodology and representative results. Section 7 draws conclusions and outlines future work.

2. System Overview

The system consists of three primary subsystems, all shown in figure 1: perception, control, and user interface. The perception subsystem acquires live video from the onboard Raspberry Pi Camera v2.1, applies preprocessing using GStreamer, and executes the YOLOv8 model on the Hailo-8L NPU. The detection outputs are processed by the control subsystem, which implements decision-making logic for steering and speed regulation via an Arduino Uno and L298N motor driver.

The user interface subsystem provides a Tkinter-based dashboard that enables operators to observe the processed video stream, view detection results, and adjust control parameters such as PID gains in real time.

Communication between the Raspberry Pi and the dashboard is managed through a Flask server and Socket.IO, ensuring low-latency data transfer.

The perception module continuously feeds updated scene interpretations to the control module, which updates motor commands accordingly. The dashboard receives status updates and allows manual override when required.

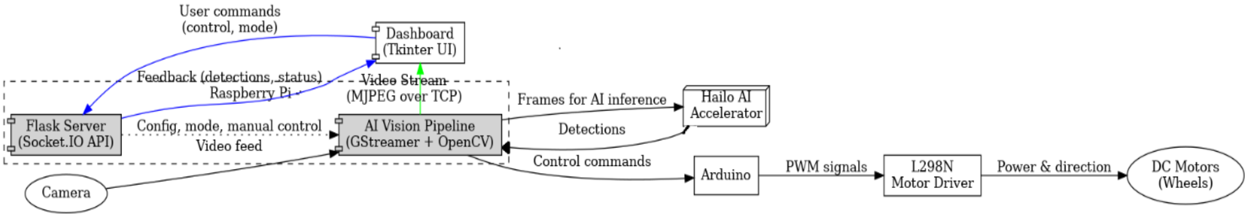


Figure 1: System architecture

3. Perception subsystem — image acquisition and processing

3.1 Acquisition and pipeline orchestration

Frames from the Raspberry Pi Camera Module v2.1 were ingested through a custom GStreamer pipeline specifically designed for embedded real-time inference on the Hailo-8L accelerator.

The source element was implemented with a custom source that's modifiable from the application (appsrc), configured to operate in real-time operation with a max buffer of 1 to prevent latency accumulation by dropping stale frames when downstream processing is delayed.
This ensured deterministic timing and avoided buffer growth, as well as also being the most recommended scenario in these conditions [4].

```
def SOURCE_PIPELINE(video_format="RGB", name="app_source"):
    return (
        f"appsrc name={name} is-live=true leaky-type=downstream max-buffers=1 ! "
        f"video/x-raw,format={video_format} ! "
    )
```

Captured frames were standardized to the 640×640 resolution required by both the Hailo module and the YOLOv8 model.

The stream was then split using the tee element: one branch was routed to the Hailo inference path, while the other was processed with classical vision operations (affine rotation, Gaussian smoothing, sharpening, and threshold-based segmentation) in parallel threads.

On the inference branch, the vendor-provided hailonet element executed the compiled YOLOv8 model (.hef format) directly on the Hailo-8L NPU.
The display branch was responsible for visualization and debugging, having included explicit queue elements to decouple processing stages, videoconvert with multithreaded execution (n-threads=2) for efficient format conversion, and fpsdisplaysink to monitor runtime frame rates.

By design, this branch was isolated from the inference path to ensure that display overhead did not interfere with real-time detection:

```
def DISPLAY_PIPELINE(
    video_sink="autovideosink", sync="false", show_fps="true", name="hailo_display"
):
    display_pipeline = (
        f'{OVERLAY_PIPELINE(name=f"{name}_overlay")} ! '
        f'{QUEUE(name=f"{name}_videoconvert_q")} ! '
        f'videoconvert name={name}_videoconvert n-threads=2 qos=false ! '
        f'{QUEUE(name=f"{name}_q")} ! '
        f'fpsdisplaysink name={name} video-sink={video_sink} sync=false text-        overlay={show_fps} signal-fps-
measurements=true '
    )
    return display_pipeline
```

Pipeline orchestration was managed by a custom Python application built on the GStreamer API, that parses runtime parameters, instantiates the full pipeline string, and attaches the callbacks to handle inference outputs.

The following simplified example illustrates initialization with batch size, resolution, and suppression thresholds:

```
class CustomGStreamerDetectionApp(GStreamerApp):
  def __init__(self, app_callback, user_data):
    parser = get_default_parser()
    parser.add_argument(
      "--labels-json",
      default=None,
      help="Path to custom labels JSON file")
    parser.add_argument(
      "--headless",
      action="store_true",
      help="Run the application in headless mode (no display).")
    parser.add_argument(
      "--stream-address",
      action="store_true",
      help="Streams address of the video source.")
    super().__init__(parser, user_data)
    args = parser.parse_args()
    self.batch_size = 2
    self.video_width=800
    self.video_height=600
    nms_score_threshold = 0.5
    nms_iou_threshold = 0.5
```

This architecture ensured that both perception streams were synchronized in real time and provided unified outputs to the navigation logic. The modular use of queues, controlled buffering, and accelerator-aware configuration allowed the system to sustain 27 frames per second with an average end-to-end latency of ~80 ms. These results demonstrate that combining GStreamer's modular streaming framework with Hailo's dedicated NPU acceleration provides an effective and reproducible approach to embedded real-time perception.

3.2 Color conversion and orientation correction

Captured frames arrive in RGB; OpenCV on the target uses BGR as its working format, so each frame is converted to BGR immediately after capture to guarantee consistent color processing. In the deployed hardware the camera is mounted inverted relative to the vehicle direction; therefore each frame is rotated to correct orientation before any other processing. The orientation correction is implemented by computing an affine rotation matrix about the image center and applying a warp to the frame; this guarantees that subsequent geometric assumptions in the segmentation stage remain valid.

3.3 Noise reduction and detail restoration

Real-world operation introduces high-frequency noise from lighting changes and vehicle vibrations. The pipeline reduces noise with a Gaussian low-pass filter implemented as a 5×5 kernel; this kernel size was empirically selected because it removes transient pixel noise while preserving edges relevant to the line. After smoothing, the pipeline applies a small sharpening convolution to restore local contrast and re-emphasize the line edges that can be attenuated by the blur operation. The sharpening uses a compact 3×3 kernel with a positive center coefficient and negative neighbors; the combined blur-sharpen sequence improves robustness of the later thresholding and contour extraction steps by producing clean but crisp edges. These parameters and ordering were validated in on-track experiments.

3.4 Line segmentation, morphology and contour simplification

Line extraction targets a dark (black) guiding line painted on the track. After preprocessing, color-based thresholding isolates dark pixels into a binary mask. Because real images contain isolated noise and small gaps in the line, morphological operations are applied: an erosion pass to remove tiny spurious blobs, followed by a dilation to recover the intended line thickness and continuity. The combination reduces false positives and yields a connected component representing the primary guidance line in the scene. Contours are extracted from the binary mask and the largest contour by area is selected as the principal line candidate.

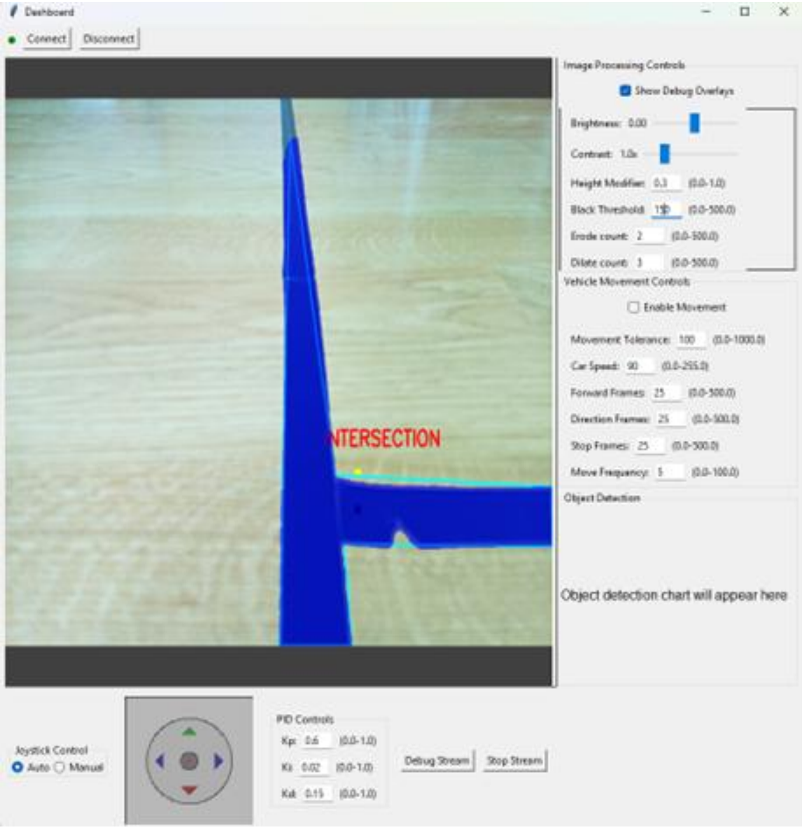


Image Processing Controls
☑ Show Debug Overlays
Brightness: 0.00
Contrast: 1.0x
Height Modifier: 0.3 (0.0-1.0)
Black Threshold: 1[illegible]0 (0.0-500.0)
Erode count: 2 (0.0-500.0)
Dilate count: 3 (0.0-500.0)

Vehicle Movement Controls
☐ Enable Movement
Movement Tolerance: 100 (0.0-1000.0)
Car Speed: 90 (0.0-255.0)
Forward Frames: 25 (0.0-500.0)
Direction Frames: 25 (0.0-500.0)
Stop Frames: 25 (0.0-500.0)
Move Frequency: 5 (0.0-100.0)

Object Detection
Object detection chart will appear here

Joystick Control
● Auto ○ Manual

PID Controls
Kp: 0.6 (0.0-1.0)
Ki: 0.02 (0.0-1.0)
Kd: 0.15 (0.0-1.0)

Figure 2: Intersection polygon

Contour complexity is reduced by approximating the contour with the Ramer–Douglas–Peucker algorithm. This approximation produces a simplified polygon, as seen in Figure 2 highlighted in cyan, that preserves the salient geometric features (corners, bifurcations) while significantly reducing the number of points used to represent the curve. The simplified polygon is then analyzed to detect intersections: multiple polygon vertices with specific angular relations and counts are interpreted as an active intersection; this triggers the sign-detection logic to check for nearby traffic signs before deciding maneuvers. The sequence of image preprocessing, morphological cleanup and polygon simplification was chosen for its computational efficiency and for the clarity of the resulting geometric signatures used by the navigation logic.

3.5 Centroid extraction and error computation

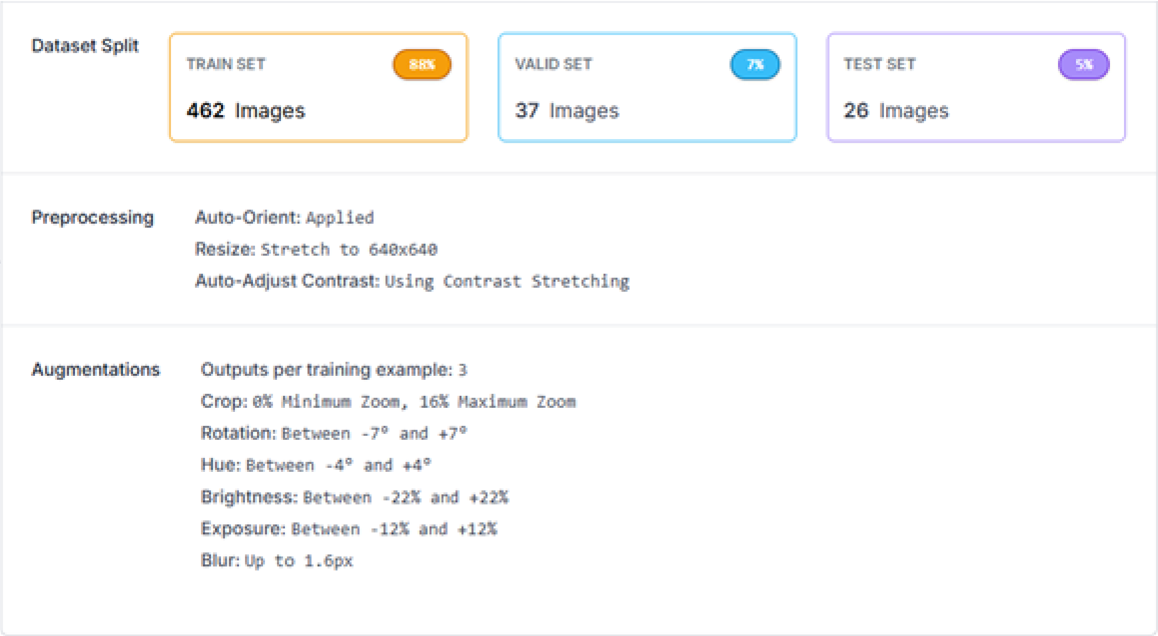
The control rule uses a simple and robust metric: the horizontal offset between the image center and the centroid of the largest line contour. The centroid is computed from raw contour moments; the lateral error is the pixel difference between centroid x coordinate and image center x coordinate. This error is then scaled to a normalized setpoint range and sent to the Arduino for PID control. Empirically, this approach provides good performance on smooth curves and allows the PID controller to correct small deviations without aggressive oscillations. All centroid and error calculations are performed per frame in real time inside the GStreamer application.

3.6 Neural inference for traffic signs (YOLOv8 on Hailo-8L)

Traffic sign detection was based on a YOLOv8 detector trained on a custom dataset prepared with Roboflow. The dataset contained a total of 525 labeled images, divided into 462 training images (88%), 37 validation images (7%), and 26 test images (5%). Preprocessing steps included auto-orientation correction, resizing to 640×640 pixels, and contrast stretching for enhanced visibility of features.

Data augmentation was applied to improve generalization under real-world conditions. Each training image was expanded into three augmented samples through random transformations: brightness variation between –22% and +22%, exposure adjustment between –12% and +12%, and blur up to 1.6 pixels, with aditional augmentations being visible in figure 3.

Following training, the model achieved high precision and recall on the validation set. It was exported to ONNX format and compiled with the Hailo Dataflow Compiler into a hardware-executable file (.hef). This process included quantization and resource allocation steps to optimize inference on the Hailo-8L neural processing unit. Runtime inference outputs underwent non-maximum suppression to yield final bounding boxes and class labels, restricted to the lower portion of the frame to reduce false positives and prioritize relevant signs near the track.



Dataset Split

TRAIN SET 88%
462 Images

VALID SET 7%
37 Images

TEST SET 5%
26 Images

Preprocessing

Auto-Orient: Applied
Resize: Stretch to 640x640
Auto-Adjust Contrast: Using Contrast Stretching

Augmentations

Outputs per training example: 3
Crop: 0% Minimum Zoom, 16% Maximum Zoom
Rotation: Between -7° and +7°
Hue: Between -4° and +4°
Brightness: Between -22% and +22%
Exposure: Between -12% and +12%
Blur: Up to 1.6px

Figure 3: Model dataset

4. Motion control and inter-module communication

4.1 Overview of motion control logic

The navigation logic executes on the Raspberry Pi and chooses actions according to the following modes: manual, autonomous line following, and intersection handling.

In manual mode, the dashboard operator sends direct speed commands which are forwarded to the Arduino; in autonomous mode, the Pi computes the lateral error from the perception pipeline and sends that error periodically to the Arduino for closed-loop control. When a sign is detected and validated by the confidence and location checks (for example, "turn right" with confidence above a configured threshold), the navigation logic executes a short preprogrammed maneuver (advance N frames, apply yaw command for M frames), then returns to line reacquisition.

This behavior can be better understood through the following example where in Figure 2 the Pi is detecting a "stop" sign at an intersection. Once the sign is recognized with sufficient confidence, the navigation logic commands the vehicle to move forward for a few frames, stops for a number of frames, and only then resume following the lane. The entire decision loop is cyclic and runs at the frame rate of the perception pipeline, ensuring continuous adjustment to the environment.

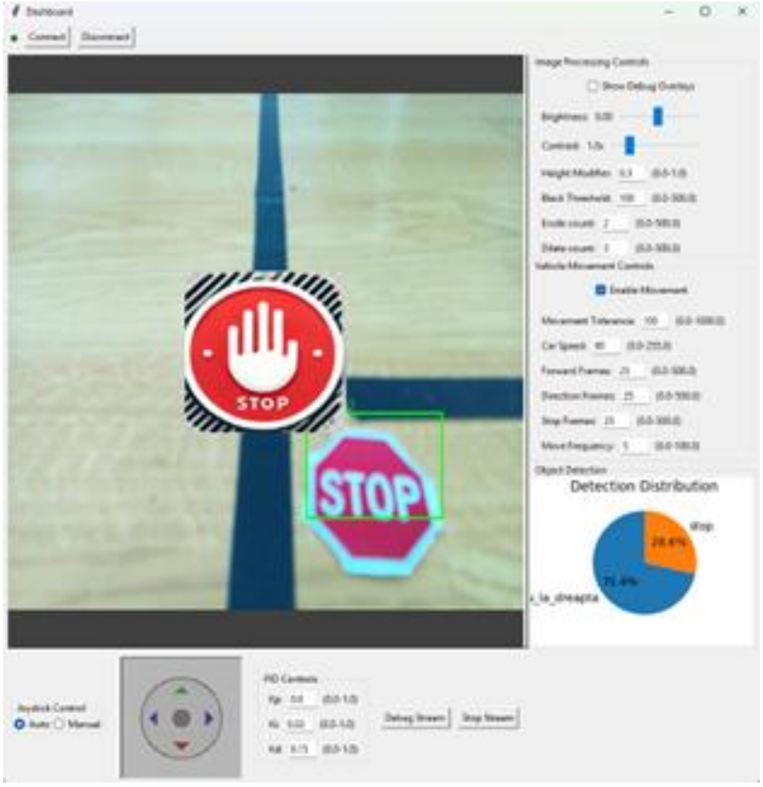


Figure 3: Intersection behavior

4.2 Serial protocol and command formats

Communication between Raspberry Pi and Arduino uses a simple, robust UART textual protocol over the USB serial link. Two message formats are used by convention.

The first format is the actuation command used in manual control: a header letter followed by left and right motor speeds, for example the format "H,left_speed,right_speed" where left_speed and right_speed are integer PWM values in the permitted range.

The second format is the autonomous error update: "E,error_value" where error_value is the signed lateral deviation computed by the perception module. The Arduino treats these messages deterministically: manual actuation messages override the PID mode, while error messages are used by the PID controller when the system is in autonomous tracking mode. Timeouts and validation checks are implemented on both sides to avoid stale commands.

4.3 Arduino controller, PID and safety

The Arduino Uno was responsible for executing low-level motor control through a discrete PID (Proportional–Integral–Derivative) controller. At each cycle, the controller computed the corrective action u(t) based on the lateral error e(t) received from the Raspberry Pi:

$$\boldsymbol{u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}} \tag{1}$$

where K_p, K_i, and K_d represent the proportional, integral, and derivative gains, respectively.

The control signal was translated into PWM values managed by the L298N module that were bounded between a lower safety threshold (to prevent motor stall) and an upper bound corresponding to maximum duty cycle. The PID parameters were tuned through trial-and-error experiments performed on the miniature track, gradually adjusting the gains until oscillations were minimized and smooth trajectory tracking was achieved.

For safety, the Arduino implemented watchdog mechanisms: if no valid message was received within a 200 ms timeout window, the controller forced motor commands to zero, halting the vehicle until communication was re-established. This safeguard was crucial for preventing uncontrolled motion during system testing.

4.4 Intersection maneuvers and realignment

When the perception pipeline signals an intersection and the sign detector confirms an instruction, the Raspberry Pi sequences a short maneuver as noted above. During the maneuver the Pi sends the centroid location to make sure that perform the desired maneuver is executed correctly. After the maneuver, the system enters a realignment subroutine, where the perception pipeline searches for the line again using a sliding-window centroid search and reduces vehicle speed while the centroid returns to the image center. Once the centroid is reacquired, error updates resume and closed-loop PID control continues.

This pragmatic approach avoids complex geometric re-localization and yields robust behavior on structured, repeatable tracks.

5. Dashboard and networking

The communication between the embedded system and the operator interface is mediated by a Flask server extended with Flask-SocketIO, which establishes a persistent WebSocket channel. This allows low-latency, bidirectional data exchange, ensuring that video streams, telemetry, and control commands are synchronized in real time[4].

To avoid blocking the vision pipeline, the server is initialized on a dedicated thread. This ensures that communication tasks are executed independently of image processing:

```
def start(self):
      self.server_thread = thread.Thread(
         target=lambda: socketio.run(
            app, host="0.0.0.0", port=5000, allow_unsafe_werkzeug=True
         ),
         daemon=True,
      )
      self.server_thread.start()
```

The dashboard transmits operator input using the emit function, with each message tagged by an event name. On the server side, these events are matched to functions annotated with the same identifier, creating a direct correspondence between client actions and server responses. In the case of parameter tuning, the dashboard sends updated PID coefficients as an event, which the server receives and applies immediately to the control loop.

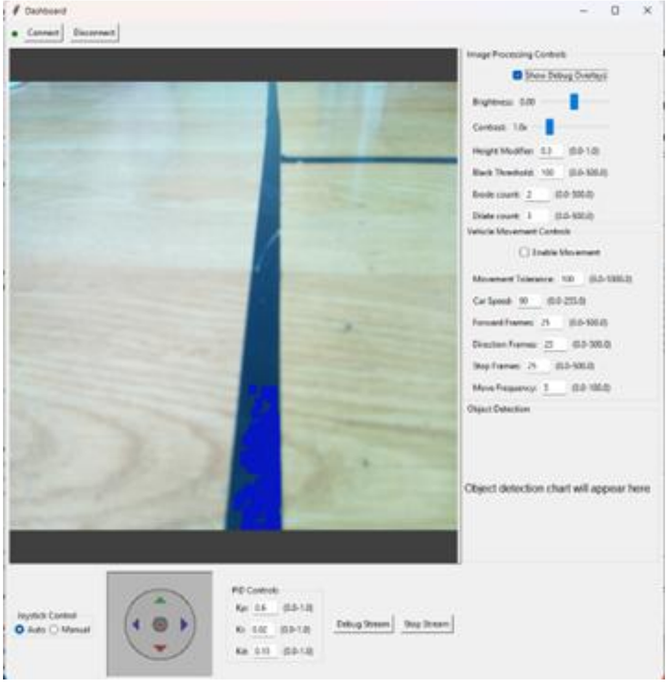
This interaction is illustrated in the following code fragments. First, the client emits the new parameters:

```
sio.emit('set_brightness', float(val))
```

The server then processes the event in the corresponding handler:

```
@socketio.on('set_brightness')
def handle_set_brightness(data):
   value = data.get('value', 0)
   pipeline_server.set_brightness(value)
   emit('brightness_ack', {'value': value}, broadcast=False)
```

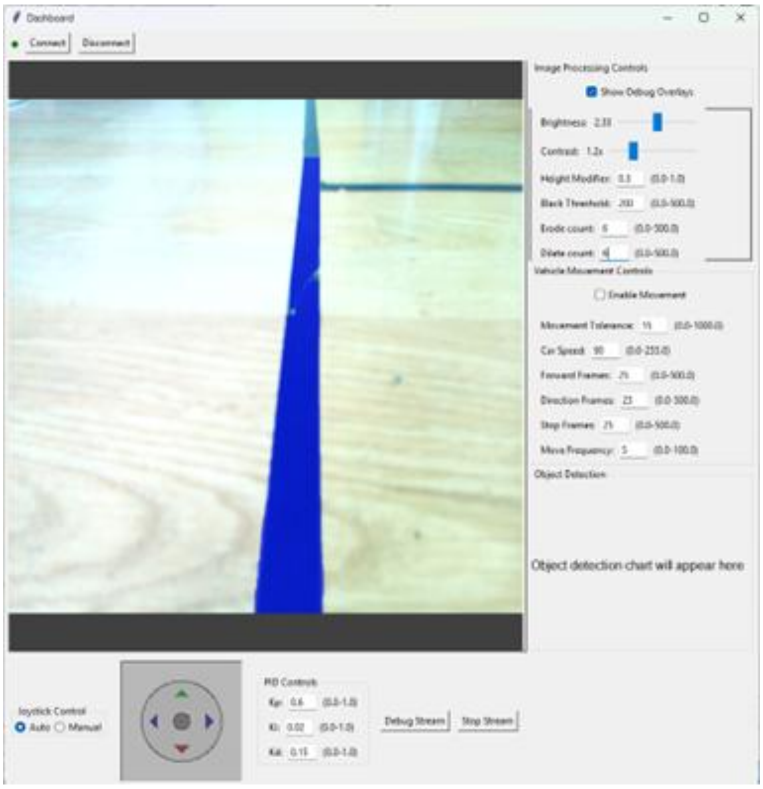
The effect of such runtime adjustments is visible in the live video feed. Figure 4 shows that, without tuning, the guiding line is detected but with unstable and noisy contours.



Brightness: 0.00
Contrast: 1.0x
Height Modifier: 0.3 (0.0-1.0)
Black Threshold: 100 (0.0-500.0)
Erode count: 2 (0.0-500.0)
Dilate count: 1 (0.0-500.0)
Movement Tolerance: 100 (0.0-1000.0)
Car Speed: 90 (0.0-255.0)
Forward Frames: 25 (0.0-500.0)
Direction Frames: 25 (0.0-300.0)
Stop Frames: 25 (0.0-500.0)
Move Frequency: 5 (0.0-100.0)
Kp: 0.6 (0.0-1.0)
Ki: 0.02 (0.0-1.0)
Kd: 0.13 (0.0-1.0)

Figure 4: Dashboard with default parameters

In contrast, Figure 5 illustrates the improvement achieved when brightness, contrast, and filtering settings are adjusted: the detected line is clearly highlighted and more stable.



☑ Show Debug Overlays
Brightness: 2.33
Contrast: 1.2x
Height Modifier: 0.3 (0.0-1.0)
Black Threshold: 200 (0.0-500.0)
Erode count: 6 (0.0-300.0)
Dilate count: 6 (0.0-500.0)

☐ Enable Movement
Movement Tolerance: 15 (0.0-1000.0)
Car Speed: 90 (0.0-255.0)
Forward Frames: 25 (0.0-500.0)
Direction Frames: 23 (0.0-300.0)
Stop Frames: 25 (0.0-500.0)
Move Frequency: 5 (0.0-100.0)

Object detection chart will appear here

● Auto ○ Manual

Kp: 0.6 (0.0-1.0)
Ki: 0.02 (0.0-1.0)
Kd: 0.15 (0.0-1.0)

Figure 5: Dashboard with modified parameters

The comparison between Figures 4 and 5 demonstrates the necessity of runtime parameter tuning to maintain robust perception under varying environmental conditions.

The video stream itself is refreshed continuously within the Tkinter interface, maintaining real-time operator feedback:

```
frame = Image.open(BytesIO(response.content))
photo = ImageTk.PhotoImage(image=frame)
video_label.config(image=photo)
video_label.image = photo
root.after(15, update_video)
```

In addition to parameter tuning, the dashboard enables manual vehicle control through joystick commands. These are emitted by the client and interpreted by the server in the annotated handler:

```
@socketio.on('set_mode')
def handle_set_mode(data):
   mode = data.get('mode', 'auto')
   pipeline_server.set_control_mode(mode)
   emit('mode_ack', {'mode': mode}, broadcast=False)
```

This architecture transforms the dashboard from a simple visualization tool into a control and diagnostic interface, where operator interventions are immediately integrated into the system’s behavior. By combining real-time monitoring, parameter adjustment, and manual override, the Flask–SocketIO communication layer ensures tight coupling between perception, decision-making, and actuation

6. Evaluation

The evaluation of the system was carried out on a miniature autonomous vehicle platform. The car, shown in Figure 6, integrates all of the hardware components specified above, enabling for real-time testing of both perception and control components under realistic conditions.

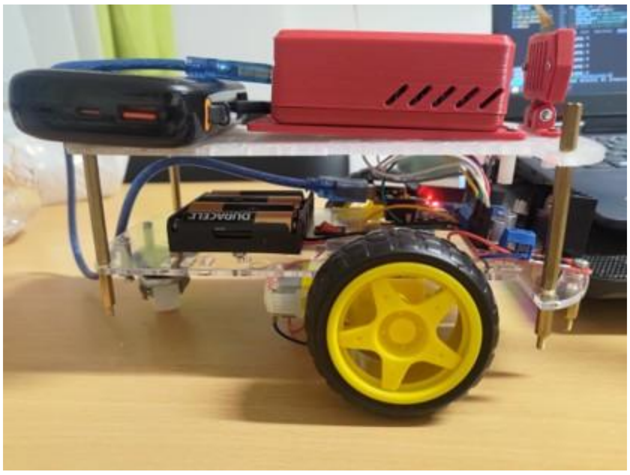


Figure 6: Assembled hardware components

For the traffic-sign recognition task, the YOLOv8 model was trained using a custom dataset prepared on Roboflow[5]. The dataset contains representative traffic signs printed and placed on the track, augmented to cover variations in lighting and orientation. The progression of training and validation metrics is illustrated in Figure 7, which shows steady convergence of the model: classification and localization losses decreased continuously, while precision, recall, and mAP values stabilized at high levels. The detector achieved an mAP50 close to 0.99 and a mAP50-95 above 0.90, confirming that the model generalized well beyond the training set.

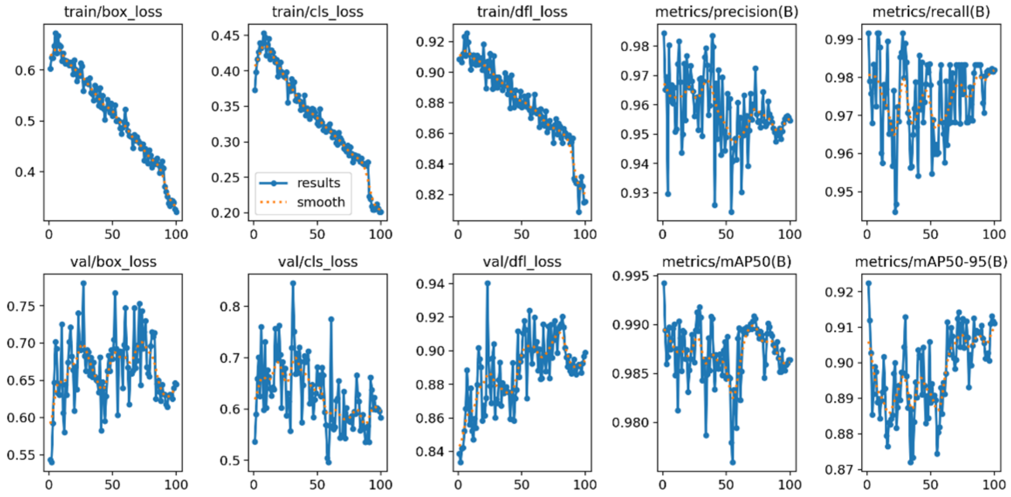


Figure 7: Model training matrics

Qualitative validation was performed by running batch inference on unseen images, with results shown in Figure 8. The model consistently detected relevant signs such as “stop,” “turn right,” “turn left,” and “no entry,” even under challenging illumination or partial occlusions. Bounding boxes and class labels were generated with high confidence, demonstrating robustness for real-time navigation scenarios.

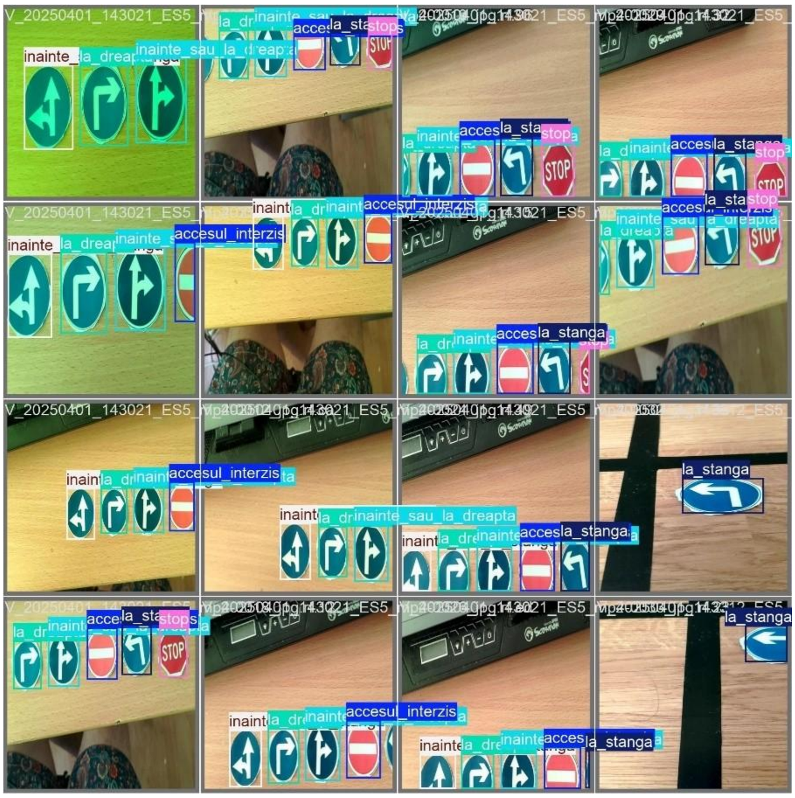


Figure 8: Validation batch

Finally, system-level experiments were conducted on a custom track including straight segments, curves, and intersections marked with printed traffic signs. The perception and control pipelines operated at an average throughout of 27 frames per second, with an end-to-end latency of around 80 ms.
This ensured timely reactions to environmental cues. The line-following algorithm maintained stable trajectories with minimal oscillations, while intersection handling proved reliable: upon detecting a sign, the vehicle executed the appropriate maneuver before resuming the lane.

These results confirm that the proposed integration of classical image processing with YOLOv8 inference accelerated by the Hailo-8L enables a low-cost but effective autonomous navigation solution. The combination of quantitative metrics and qualitative results, as presented in Figures 4–6, highlights both the robustness of the perception subsystem and the stability of the overall navigation performance.

7. Conclusions and Future Work

The presented work demonstrates the feasibility of implementing a fully functional, low-cost autonomous navigation system using embedded AI acceleration and real-time object detection. The combination of the Raspberry Pi, Hailo-8L, and Arduino platforms results in a scalable architecture suitable for both educational activities and experimental research in autonomous driving.

Future developments will focus on enhancing the system's capabilities through the integration of simultaneous localization and mapping (SLAM) techniques combined with LiDAR sensing for dynamic mapping, the addition of obstacle avoidance strategies, the implementation of a mobile-based control interface, and the exploration of reinforcement learning methods for end-to-end driving policy optimization.

Acknowledgement: This work was supervised by Ralf Fabian, PhD, from Lucian Blaga University of Sibiu.

References

[1] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, arXiv:1506.02640.
[2] J. R. Terven and D. M. Cordova-Esparza, *A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS*, arXiv:2304.00501v6.
[3] S. Nahavandi et al., *A Comprehensive Review on Autonomous Navigation*, arXiv:2212.12808.
[4] GStreamer Application Development Manual, https://gstreamer.freedesktop.org/documentation/application-development/index.html?gi-language=c
[5] Flask Server Official Documentation, https://flask.palletsprojects.com/en/stable/
[6] Traffic sign detection dataset, https://app.roboflow.com/daniel-bsc7p/detectie-semne-de-circulatie/11

Marian-Daniel Drăghici
Lucian Blaga University of Sibiu
Faculty of Science
Doctor Ion Ratiu Street 5-7, Sibiu
ROMANIA
E-mail: mariandaniel.draghici@ulbsibiu.ro

Andrei Dăian
Lucian Blaga University of Sibiu
Faculty of Science
Doctor Ion Ratiu Street 5-7, Sibiu
ROMANIA
E-mail: andrei.daian@ulbsibiu.ro

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

EmiNet

Ștefan Eminovici

Abstract

EmiNet is a software application employing a Multi-Layer Perceptron (MLP) neural network to provide preliminary diagnoses based on user-inputted symptoms. The model, trained on a dataset covering approximately 80 common illnesses, utilizes SMOTE (Synthetic Minority Over-sampling Technique) to address class imbalance, and its development involved k-fold cross-validation to ensure robust generalization and almost perfect predictions. It can request additional information (further symptoms) when the confidence level of the initial prediction is low. Natural language symptom processing is performed using advanced techniques, including the NLTK and spaCy libraries. The artificial intelligence component is implemented using TensorFlow and Keras, alongside NumPy for numerical operations. The user interface is developed in Flutter, communicating via a Flask API with the Python-implemented backend. Multilingual support (English/Romanian) ensures broader accessibility.

1 Introduction

Navigating online medical information when you're not feeling well can be overwhelming. EmiNet was born out of a desire to simplify this first step: offering initial guidance based on the symptoms you describe. I wanted to create a helpful tool that would allow people to better understand the possible causes of their health condition before consulting a specialist.

The heart of EmiNet is a neural network – a form of artificial intelligence – trained to analyze symptoms and suggest potential conditions, along with a confidence level. I hope this will help inform users and facilitate subsequent discussion with their doctor. To be useful to as wide an audience as possible, the project works in both English and Romanian. However, it is vital to emphasize: EmiNet is an informational guide, not a replacement for professional medical consultation. Diagnoses are indicative and must always be confirmed by a doctor.

2 Design and User Interface

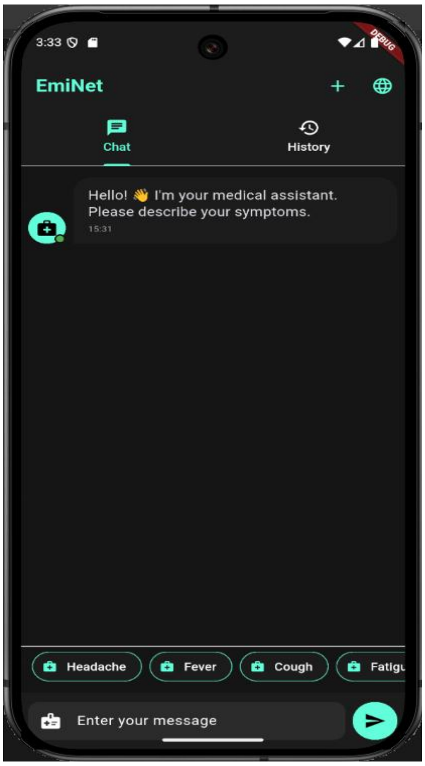
I wanted EmiNet to be easy and pleasant to use, even for someone less familiar with technology or medical terms. We focused on a clean and intuitive design that presents information clearly and directly.

a. Sources of Inspiration

The inspiration behind EmiNet is deeply personal. Conversations with my brother, a medical student, showed me how complex the diagnosis process is and how helpful digital tools can be. But perhaps the most powerful motivation comes from the memory of my mother, a cardiologist. Losing her to COVID-19 painfully underscored the importance of rapid access to medical information and tools. Even though EmiNet cannot replace a doctor's expertise, I hope it serves as a valuable first step, inspired by her dedication.

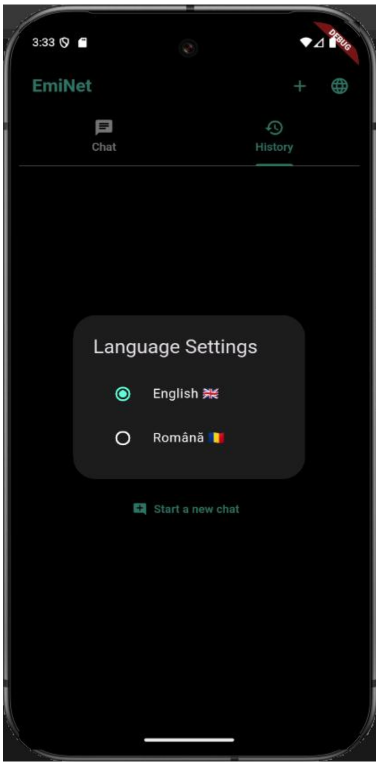
b. Why Flutter for the interface

To bring EmiNet to life on users' screens, we chose Flutter, Google's modern toolkit. Flutter's main advantage is that it lets us build applications that look and perform great on mobile (iOS, Android), web, and desktop—all from the same codebase. This means faster development and a consistent user experience. With Flutter's visual components (widgets), I created a friendly interface where users can input symptoms and view results—preliminary diagnosis, warnings, and possible follow-up questions—clearly and easily. This interface then "talks" to the application's "brain" (Python backend) through a Flask API, sending symptoms and receiving intelligent analysis in return.



Hello! 👋 I'm your medical assistant.
Please describe your symptoms.
15:31

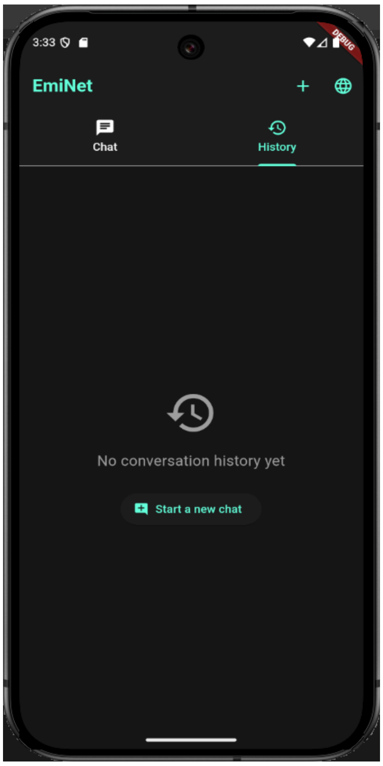
Fig. 1: Home Page



Language Settings

● English 🇬🇧
○ Română 🇷🇴

Fig. 2: Language settings



No conversation history yet

Fig. 3: Conversation history

3 System Architecture and Technologies Used

The application's "brain", which analyzes the symptoms, runs on a Python backend using powerful tools from machine learning and natural language processing (NLP).

3.1 Data Preparation (data_preparation.py)

This step transforms raw data (a list of diseases and associated symptoms in illnesses_and_symptoms.csv) into formats that the neural network can understand. We used pandas for data reading and cleaning. Then, NLP techniques were applied: lemmatization with spaCy (reducing words to their base form), removing stopwords (common unimportant words) using nltk and spaCy. To improve model robustness, we augmented data by replacing words with synonyms (using WordNet from nltk). Since diseases appear with different frequencies, I used SMOTE (from imbalanced-learn) to generate synthetic examples for underrepresented classes, balancing the dataset. The processed text was then transformed into numerical vectors using TF-IDF (from scikit-learn), and disease names were numerically encoded using LabelEncoder. Both the TF-IDF vectorizer (vectorizer.pkl) and the label encoder (label_encoder.pkl) were saved for prediction use.

```python
# data_preparation.py
def preprocess_text(text):
    """Preprocesses text using spacy: lowercasing, lemmatization, removing stopwords and non-alpha tokens."""
    if not isinstance(text, str):
        text = str(text) # Ensure text is string

    doc = nlp(text.lower())
    words = [token.lemma_ for token in doc if not token.is_stop and token.is_alpha]
    return ' '.join(words)

# Reading training set
df = pd.read_csv('illnesses_and_symptoms.csv')
df['Processed_Symptoms'] = df['Symptoms'].apply(preprocess_text)

# Data augmentation with synonyms
def augment_with_synonyms(text, num_augmentations=2):
    words = text.split()
    for _ in range(num_augmentations):
        word_idx_to_replace = random.randint(0, len(words)-1)
        synonyms = get_synonyms(words[word_idx_to_replace])
        if synonyms:
            words[word_idx_to_replace] = random.choice(synonyms)
    return ' '.join(words)

# Applying SMOTE for class balancing
X_resampled, y_resampled = SMOTE().fit_resample(X, y)

```

3.2 Neural Network (model.py)

The core AI component is a Multi-Layer Perceptron (MLP) neural network implemented using TensorFlow and Keras. The network consists of 4 layers, from which 2 hidden layers. The network takes TF-IDF vectors of symptoms and, through dense layers with ReLU activation, learns complex patterns between symptoms and diseases. Dropout and Batch Normalization layers prevent overfitting and stabilize training. Early stopping and reducing the learning rate when encountering a plateau are 2 methods which were also used in training the model. The final layer uses softmax activation to output the probability of each possible illness.

```python
# model.py
def build_model(input_shape, num_classes):
    """Builds the Keras Sequential model."""
    model = Sequential(name="Symptom_Diagnosis_MLP")
    model.add(Input(shape=(input_shape,), name="Input_Layer"))

    # Input Layer -> Dense + BatchNorm + Dropout
    model.add(Dense(256, activation='relu',
                    kernel_regularizer=l1_l2(l1=0.001, l2=0.001),
                    name="Dense_1"))
    model.add(BatchNormalization(name="BatchNorm_1"))
    model.add(Dropout(0.3, name="Dropout_1"))

    # Hidden Layers
    model.add(Dense(128, activation='relu', name="Dense_2"))
    model.add(BatchNormalization(name="BatchNorm_2"))
    model.add(Dropout(0.4, name="Dropout_2"))

    # Output Layer
    model.add(Dense(num_classes, activation='softmax', name="Output_Layer"))

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

```

3.3 Training and Cross-validation (main.py, model.py)

The model was trained on the prepared and SMOTE-balanced data. To evaluate generalization on unseen data, I used stratified cross-validation (StratifiedKFold from scikit-learn). The dataset was split into K sets, training and testing the model K times using a different test set each time. The final saved model (model.keras) is the one that performed best during this process.

```python
# main.py
# Preparation for training with cross-validation
n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)
cv_scores = []

# Cross-validation process
for train_idx, val_idx in skf.split(X_resampled, y_resampled):
    X_train, X_val = X_resampled[train_idx], X_resampled[val_idx]
    y_train, y_val = y_resampled[train_idx], y_resampled[val_idx]

    # Building and training model
    model = build_model(X_train.shape[1], len(np.unique(y_train)))
    history = train_model(model, X_train, y_train, X_val, y_val)

    # Evaluating on test set
    val_loss, val_acc = model.evaluate(X_val, y_val, verbose=0)
    cv_scores.append(val_acc)

# Saving final model for production
# final_model = ... (logic to select the best model based on cv_scores/history)
final_model.save("model.keras")
```

3.4 Performance Evaluation (evaluate.py)

Model performance was evaluated on a separate test set using standard scikit-learn metrics: overall accuracy, detailed classification report (Precision, Recall, F1-Score per illness), and a confusion matrix to visualize misclassifications.

```
# evaluate.py
def evaluate_model(model_path, X_test, y_test, label_encoder):
    """Evaluates model performance on test data."""
    model = load_model(model_path)

    # Prediction & Evaluation
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Computing Metrics
    accuracy = accuracy_score(y_test, y_pred_classes)
    report = classification_report(
        y_test,
        y_pred_classes,
        target_names=label_encoder.classes_
    )

    # Confusion matrix for visualisation
    cm = confusion_matrix(y_test, y_pred_classes)

    return accuracy, report, cm

```

3.5 Application Programming Interface - API (app.py)

The connection between the Flutter interface and the Python backend is handled by a Flask web API. The main endpoint (/predict) receives user symptoms (and language or follow-up answers), loads the trained model and saved artifacts (vectorizer, encoder), preprocesses the new symptoms, requests a prediction, interprets the result (most likely illness, confidence level, whether to ask for more symptoms), translates the final message (via translations.py), and returns a structured JSON response. A simple endpoint (/health) checks API status. Flask-Cors is used to ensure secure communication between frontend and backend.

```
# app.py
@app.route('/predict', methods=['POST'])
def predict():
    """API endpoint to predict illness based on symptoms."""
    # Parsing data
    data = request.get_json()
    symptoms_text = data.get('symptoms', '').strip()
    language = data.get('language', 'en').lower()
    follow_up_response = data.get('follow_up', '').lower().strip()
    # has_follow_up = bool(follow_up_response) # Determine if a follow-up answer was given

    # Preprocessing symptoms
    if language == 'ro':
        symptoms_text = translate_to_english(symptoms_text) # Assume translation needed for model
    processed_symptoms = preprocess_text_for_api(symptoms_text) # API specific preprocessing

    # Vectorizing and predicting
    symptom_vector = vectorizer.transform([processed_symptoms])
    prediction_probabilities = model.predict(symptom_vector)[0]
    predicted_class_index = np.argmax(prediction_probabilities)
    confidence_score = float(prediction_probabilities[predicted_class_index]) * 100

    # Translating and formatting answer
    illness_name_en = label_encoder.classes_[predicted_class_index] # Get English name first
    illness_name = illness_name_en # Default to English
    if language == 'ro':
        illness_name = translate_illness(illness_name_en, 'ro') # Translate if needed

    # Generation of follow-up question if necessary
    follow_up_question = ""
    # Assume has_follow_up logic exists based on follow_up_response check
    # Example logic (replace with actual):
    has_follow_up = len(follow_up_response) > 0
    if confidence_score < 70 and not has_follow_up:
        # Pass English name for consistent question lookup
        follow_up_question = get_follow_up_question(illness_name_en, language)

    # Formatting final answer
    diagnosis_message = get_diagnosis_message(illness_name, confidence_score, language)

    return jsonify({
        'diagnosis_message': diagnosis_message,
        'follow_up_question': follow_up_question
    }), 200
```

Model Saving is addressed by saving the trained model (model.keras), the TF-IDF vectorizer (vectorizer.pkl), and the disease encoder (label_encoder.pkl) to disk, allowing the API to load them quickly at startup without retraining.

Multilingual support & follow-up questions) is implemented through dedicated dictionaries and functions were created for automatic message translation, disclaimer messages, and follow-up questions between English and Romanian, based on user preference.

4 Conclusions and Future Directions

EmiNet demonstrates how artificial intelligence and natural language processing can collaborate to create an initial medical information tool. We developed a full system, from data preprocessing and training a robust MLP model to exposing features through a web API and integrating with a user-friendly interface (Flutter). The ability to understand two languages and request further clarifications increases the app's practical utility.

The main goal of EmiNet is to offer users a quick, informed initial perspective on the possible causes of their symptoms. However, we emphasize once again: EmiNet is an informational guide, not a replacement for professional medical diagnosis. Final medical decisions and treatment plans must be determined exclusively by a specialist. This project demonstrates the potential of AI in medical awareness and paves the way for future developments. Possible improvements include expanding the database with more illnesses, exploring the integration (with strict confidentiality) of other data types (e.g., medical history), and investigating more advanced AI architectures to enhance prediction accuracy.

Acknowledgement: This work was supervised by Professor Delilah Florea, from Samuel von Brukenthal National College Sibiu, Romania.

References

[1] TensorFlow: An end-to-end platform for machine learning, https://www.tensorflow.org/
[2] NLTK Project, Natural Language Toolkit (NLTK). https://www.nltk.org/
[3] Explosion AI, spaCy: Industrial-strength Natural Language Processing, https://spacy.io/
[4] Keras: A superpower for ML developers, https://keras.io/
[5] Scikit-learn: Machine Learning in Python, https://scikit-learn.org/
[6] Flask: A lightweight WSGI web application framework, https://flask.palletsprojects.com/en/
[7] Flutter: Build apps for any screen, https://flutter.dev/
[8] NumPy: Fundamental package for scientific computing with Python, https://numpy.org/
[9] Pandas: Data analysis and manipulation tool, https://pandas.pydata.org/
[10] imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning, https://imbalanced-learn.org/
[11] National Health Service (NHS). Help improve online NHS services by also accepting analytics cookies., https://www.nhs.uk/conditions/

Ștefan Eminovici
„Samuel von Brukenthal" National College Sibiu
Mathemathics-Informatics, intensive Informatics
ROMANIA
E-mail: stefaneminovici@yahoo.com

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

User-Centered AI: Improving Workflows through intelligent Chatbots

Sophie Geisler, Peter Möhle, Marcel Wernisch, Felix Zorn

Abstract

The manual procedure of requesting and scheduling rooms for university classes often causes considerable administrative effort and delays. This paper presents an AI-driven chatbot integrated into a university room booking system to handle first-level support queries (such as room availability checks and booking requests) through natural language. The proposed chatbot leverages a Large Language Model (LLM) to understand user inquiries and generate context-aware responses, thereby automating routine booking interactions. We outline the system's architecture, highlighting the chatbot's central role as an interface between users and the scheduling backend. The chatbot's design incorporates an LLM component for language understanding and generation, guided by system-defined prompts and safeguarded by validation layers to ensure reliable and secure operation. We discuss how user interaction is facilitated via a web frontend and a chat platform integration, and how the AI-driven decision-making enables flexible, context-sensitive dialogue management. In this context, a self-developed backend service was introduced to interface with the existing booking system, ensuring accurate retrieval and synchronization of room availability and booking details. An initial workflow-based chatbot design is described as a conceptual alternative, though our implementation focuses on the User-Centered approach. Preliminary results from the prototype indicate improved efficiency (reduced response times and automated task completion) and high intent-recognition accuracy, while also revealing important considerations regarding system safety and governance. We conclude with insights on the practical implications of deploying LLM-powered chatbots for scheduling support and directions for future work to enhance robustness and scalability.

1 Introduction

Scheduling classrooms and other resources in academic institutions is often a time-consuming process involving numerous manual steps and communications. Instructors typically request room bookings or modifications via emails or phone calls, which administrative staff must process in the scheduling system. This manual workflow can lead to delays, miscommunication, and increased workload for support staff. There is a clear opportunity to streamline this process by introducing automation in the form of an intelligent assistant.

Recent advances in artificial intelligence, particularly in Large Language Models (LLMs) such as GPT-3 and GPT-4, have enabled the development of more sophisticated chatbots capable of understanding and generating human-like language [1]. Such AI-driven chatbots can engage with users in natural language, interpret their requests, and perform complex tasks by interfacing with external systems. Deploying an LLM-powered chatbot for room booking presents an attractive solution to handle routine scheduling inquiries and requests automatically, acting as a first-level support agent that is available 24/7.

To improve efficiency and user experience, we propose a chatbot-based solution that serves as a conversational interface for room and schedule management. Modern Large Language Models enable the chatbot to communicate in natural language, understand user intents, and perform actions, effectively bridging the gap between end-users and the complex scheduling backend. By automating recurring queries and guiding users through structured dialogues, the chatbot (named the Chat-Provider) aims to reduce manual effort and errors while maintaining oversight for critical decisions. This user-centered AI approach treats the chatbot as a “friendly colleague” always available to assist with booking tasks in a straightforward manner.

Our approach emphasizes a modular architecture that integrates seamlessly into the existing system landscape of the university. A web-based frontend and a chat platform interface (e.g., a Mattermost bot) allow users to interact with the Chat-Provider, which in turn communicates with a dedicated management backend service. The management service handles business logic and data persistence, including integration with the legacy Sked (lecture management) database.

A core challenge in designing the Chat-Provider is balancing intelligent autonomy with predictable control. We focus on a modular architecture that combines generative AI capabilities with deterministic workflow logic to ensure reliable operation. In this context, we explore two conceptual architectures for the Chat-Provider: an orchestrated multi-agent approach and a single-agent (chat-oriented) approach. Both solutions are implemented with a strong emphasis on extensibility, security (preventing unwanted or incorrect behavior), and efficiency.

This paper is organized as follows: Section 2 describes the system architecture and the chatbot’s integration into the room booking infrastructure. Section 3 details the chatbot’s functionality, including user interaction flows and the AI-driven decision-making process of the LLM component. Section 4 discusses the implementation and early evaluation of the prototype, highlighting achieved improvements and encountered challenges. Section 5 concludes the paper with a summary of findings and an outlook on future enhancements.

2 Literature Review

Recent research on intelligent conversational systems provides several technical building blocks that directly inform the framework proposed in this paper. We summarise the most relevant strands here.

LLM-based chatbots for administrative tasks. Large language models such as GPT-4 enable chatbots to interpret unstructured, human language requests and translate them into concrete API calls. *HuggingGPT* [3] shows that a single LLM agent can decompose a user goal, select appropriate external tools, and merge partial results into a coherent answer. Our system follows the same principle but is narrowed to the university room-booking domain. Unlike HuggingGPT’s general marketplace approach, we embed domain-specific knowledge (room codes, capacity rules, approval policies) and connect the agent to protected campus databases.

AI-driven workflow automation. Real-world scheduling involves several dependent steps (availability check → booking → confirmation). *WorkflowLLM* [4] fine-tunes a language model on thousands of recorded automation scripts and proves that an LLM can plan API invocations in the correct order and recover from errors. Our framework adopts this idea by exposing a compact set of typed actions (e.g. `book_event_timeslot`) and letting the LLM decide which sequence fulfils a given goal. In contrast to WorkflowLLM’s benchmark setting, we integrate additional guard rails: every irreversible action is wrapped in a confirmation routine or a human-in-the-loop approval.

Modular software architectures. To ensure longevity and testability, the application core is modelled after the *hexagonal architecture* (ports and adapters pattern) defined by Cockburn [5]. Domain logic is completely isolated from infrastructure; concrete technologies such as a web UI, a Mattermost bot, or the room-database connector appear only as interchangeable adapters. Compared to classic layered designs, this separation allows the LLM agent and future auxiliary agents to evolve independently of front-end or storage technology and supports gradual migration toward alternative booking back-ends.

Collectively, the cited work underlines five key insights: (1) a single LLM can map natural language to structured actions, (2) specialised fine-tuning enhances multistep orchestration, (3) multi-agent patterns offer headroom for scaling task complexity, (4) strict validation layers remain imperative for safe deployment, and (5) hexagonal modularity reduces coupling and accelerates integration into legacy landscapes. The proposed chatbot combines these findings by placing a domain-tuned LLM agent at the centre of a hexagonal microservice architecture, surrounded by explicit guard rails and well-defined adapters.

3 System Architecture - Methodology and System Design

The overall architecture of the room booking chatbot system, named `CampusRoom`, follows a microservices approach, separating the user interface components from the backend services. Figure 1 provides an overview of the main components and their interactions. The two primary frontend interfaces — a web application and a chat platform bot — allow users to interact with the chatbot. These frontends communicate with the *Chatbot Service*, which encapsulates the core chatbot logic and LLM capabilities. The Chat-Provider Service in turn interfaces with the *Management System* (scheduling backend) to retrieve and update room booking information.

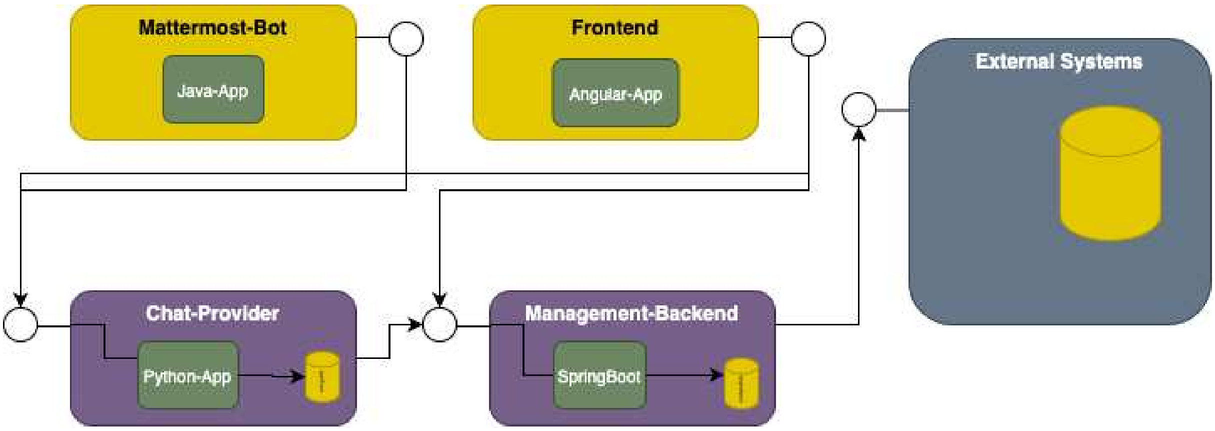
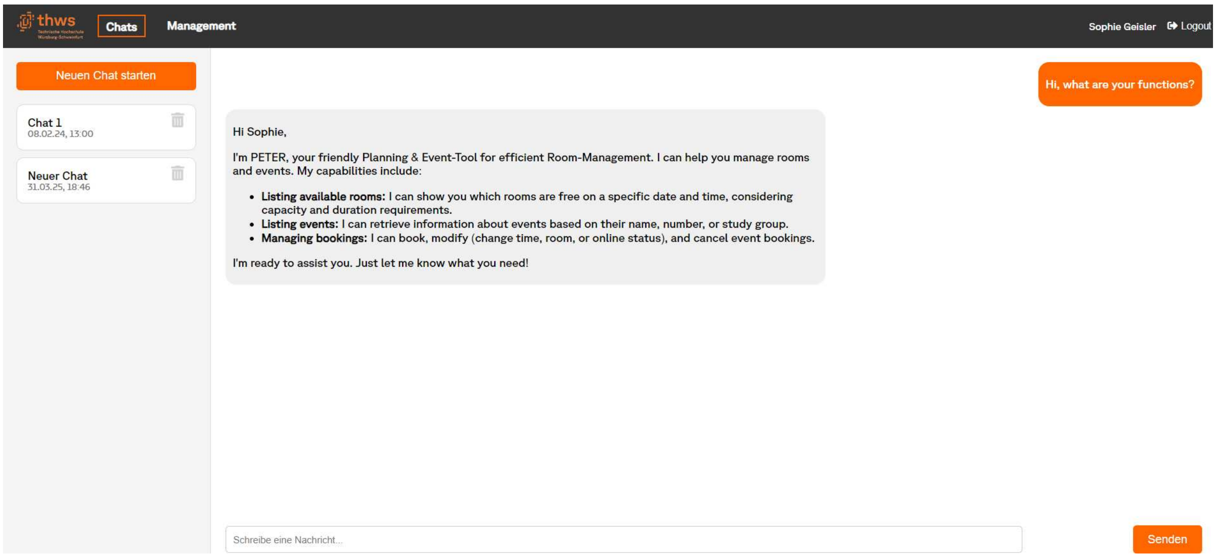


Figure 1: System architecture of the room booking chatbot, showing user interface clients (web app and chat bot) connecting to the Chatbot Service, which interacts with the Management backend system and database.

In the proposed architecture, the Chatbot Service operates as an intermediary between users and the room booking backend. The web frontend is implemented as a single-page application (using Angular) that provides a chat interface embedded in a web page (Figure 2).



Hi, what are your functions?

Hi Sophie,

I'm PETER, your friendly Planning & Event-Tool for efficient Room-Management. I can help you manage rooms and events. My capabilities include:

- **Listing available rooms:** I can show you which rooms are free on a specific date and time, considering capacity and duration requirements.
- **Listing events:** I can retrieve information about events based on their name, number, or study group.
- **Managing bookings:** I can book, modify (change time, room, or online status), and cancel event bookings.

I'm ready to assist you. Just let me know what you need!

Figure 2: User Interface of the web frontend.

The chat platform integration is implemented as a bot user within an institutional messaging platform (Mattermost), enabling users to converse with the chatbot in a channel or direct message. Both frontends send user messages (along with metadata such as user ID and a chat session ID) to the Chatbot Service via REST API calls. The chat session ID allows the service to maintain context across multiple turns of conversation with each user.

The backend consists of the self-developed room scheduling system (which we refer to as the **management system**) and associated databases of the existing integrated scheduling software. A hexagonal architecture (Figure 3) was chosen, providing high flexibility and adaptation for alternative systems.

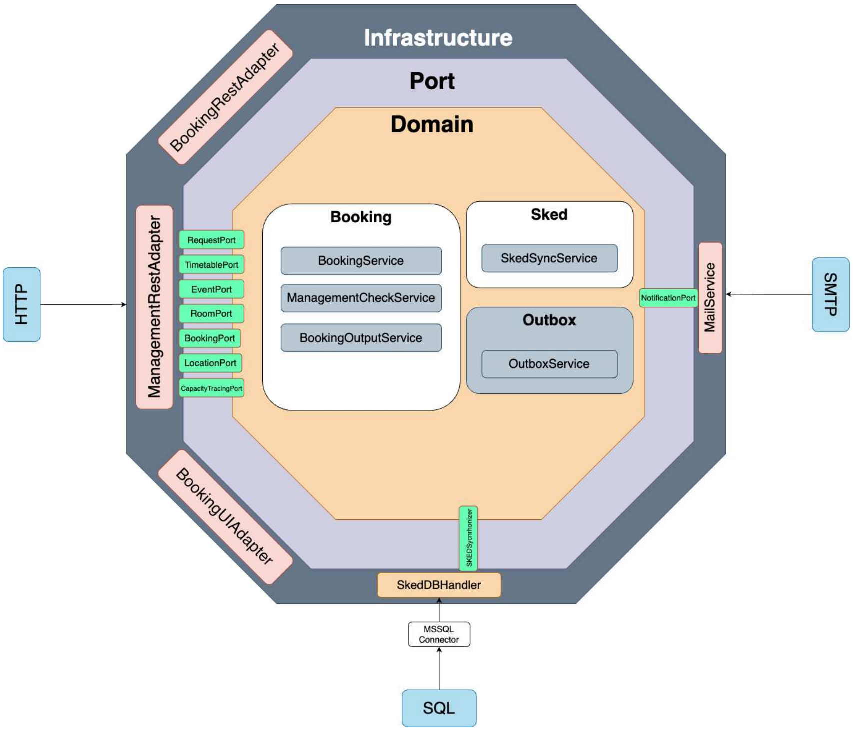


Figure 3: The management-system, providing a hexagonal architecture for higher flexibility.

The **Chat-Provider Service** queries this backend to check room availability, fetch schedules, and create or modify bookings as needed. By using well-defined APIs between the Chat-Provider Service and the Management, the integration remains decoupled: the Chat-Provider can operate without needing direct access to the internal database, and the scheduling system requires minimal changes to support the Chat-Provider's queries. Each microservice, including the Chat-Provider Service and other backend components, manages its own data storage, ensuring loose coupling and easier maintenance.

We considered two conceptual approaches for the chatbot's internal decision-making architecture during development. The first was a *workflow-oriented design* structured as our concept of a multi-agent system, where the task of handling a user query would be broken down into a sequence of specialized steps. In this multi-agent approach, separate AI agents could be responsible for distinct subtasks, such as intent recognition, information retrieval, decision making, and response generation, with a central orchestrator coordinating their actions. This design imposes either a predefined dialogue flow or workflow or alternatively a dynamically created one, based on the system's capabilities and their metadata: for example, when a user wants to book a room, the system would explicitly follow steps to first select or create a workflow, gather required information (date, time, participants, etc.), query the database, and then confirm the booking. Such a structured approach can simplify oversight and ensure each subtask is handled by an expert module, but it may be less flexible in handling unexpected queries or conversational turns.

The second approach, which we finally pursued further, is a *single-agent LLM-driven architecture* as illustrates in 5. In this design, a single large language model agent handles the user's query end-to-end, leveraging its internal reasoning capabilities to decide how to address the request. Rather than following

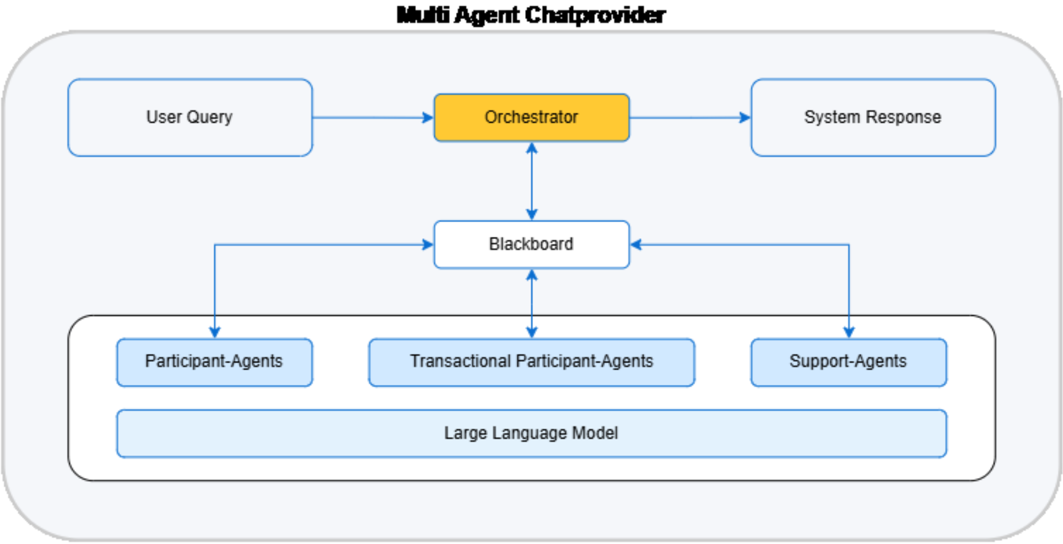


Figure 4: The implemented Workflow-Oriented multi-agent-solution.

or creating a compound workflow, the LLM interprets the user's input and dynamically determines only the next steps (e.g., whether to ask a clarification question, perform a database lookup, or formulate an answer) within the scope of its programming. This approach takes advantage of the LLM's ability to understand context and intent in a flexible manner, allowing a natural, purposeful dialogue. It simplifies the architecture by consolidating the conversational logic within one intelligent component, at the cost of that component needing to be carefully guided and controlled.

To maintain reliability and predictability in the LLM-driven approach, our system incorporates several design principles. First, the architecture remains **modular and extensible** – the Chatbot Service is structured so that underlying components (such as the LLM model, the database connectors, or the agents modules) can be modified or upgraded independently. Second, we enforce **safety through control layers**: the Chat-Provider's outputs are monitored and validated, and certain high-impact actions (like finalizing a booking) require an extra confirmation step (akin to a human-in-the-loop approval) before execution. Third, we strive for **efficiency and scalability**: the LLM is used primarily for understanding language and complex decision-making, whereas simpler operations (e.g., parsing a date or verifying a room code format) are handled with lightweight, rule-based logic or standard library functions. These principles ensure that the system is robust in the face of malicious inputs, remains adaptable to future changes, and uses computational resources wisely.

4 Chatbot Interaction and AI-Driven Decision Making

The user interacts with the system by sending messages in natural language through one of the provided frontends. From the user's perspective, the chatbot behaves like a virtual personal assistant capable of answering questions and performing actions related to room scheduling. For instance, a conversation might begin with the user asking, *"Is there an available classroom for next Monday at 10 AM capable for 30 people?"*. The chatbot will interpret this request, determine what information is being sought, and take appropriate actions to respond.

Under the hood, when a message is received by the Chat-Provider Service, it is processed by the LLM component which serves as the brain of the chatbot. This LLM component performs Natural Language Understanding (NLU) by parsing the input to extract the user's intent (e.g., checking room availability) and key entities such as dates, times, or specific room requirements. In our implementation, the LLM is prompted with the conversation history and a predefined system prompt that outlines its role and

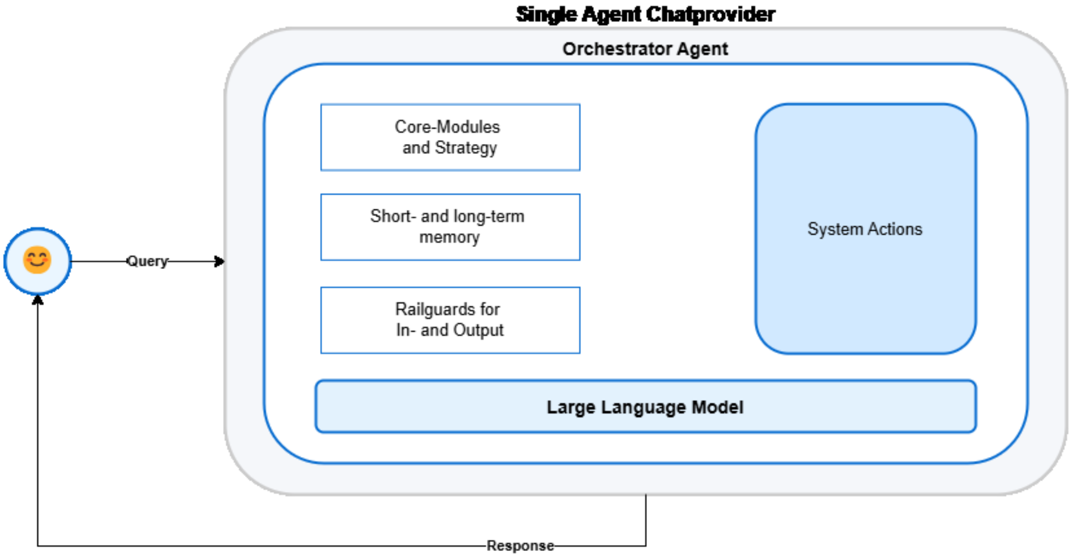

Single Agent Chatprovider
Orchestrator Agent
Core-Modules and Strategy
Short- and long-term memory
Railguards for In- and Output
System Actions
Large Language Model
Query
Response

Figure 5: The structural design of the user-centered single agent implmentation.

constraints. This system prompt acts as a guideline for the AI, specifying the domain of discourse (room booking), the style of interaction (helpful and concise), and any rules it should follow (for example, not to perform actions outside its scope or to ask for confirmation before booking a room on behalf of the user). The response will be interpreted in a predefined JSON structure, guided by the model's system prompt.

Once the user's intent is understood, the chatbot needs to fulfill the request. In many cases, this involves retrieving data from the Management system or updating it. In our LLM-driven design, the Chatbot Service contains logic to perform such operations when required. For example, if the LLM's analysis of the query indicates that a database lookup is needed (implicitly or through the content of the LLM's output), the service will query the scheduling system's API for room availability. We designed the interaction between the LLM and external tools in a manner inspired by the ReAct paradigm [2], where the AI can "think" (reason about what to do) and then "act" (invoke an external tool or API) in a controlled loop. In practice, this means the Chatbot Service might use the LLM to generate an intermediate representation or query plan (e.g., an internal command like "CHECK_AVAILABILITY(date, time, capacity)") which the service then executes against the backend. The result (e.g., a list of available rooms) is then incorporated into the LLM's context for generating the final answer.

Throughout this process, maintaining context across turns is crucial. The system keeps a record of the conversation state associated with the chat session ID. If the user asks a follow-up question like *"Can you book Room H.1.1 for me?"*, the chatbot remembers the earlier discussion about Monday 10 AM and knows the context (the user is likely referring to booking Room H.1.1 at that previously mentioned time). The LLM uses the accumulated dialogue (truncated to fit within its context window as needed) along with the system prompt to generate a relevant response or action for each turn.

Figure 6 illustrates the internal concept of the LLM-driven chatbot component. The LLM is at the core, taking in the user query plus context and system instructions, and outputting either a direct response or an action suggestion. Surrounding it are the supporting mechanisms: the system prompt that governs its behavior, the context memory (chat history and any extracted parameters stored from previous turns), and the interface to external services (such as the scheduling database). This design enables the AI to handle the conversational aspect while delegating concrete operations (like data retrieval) to the appropriate modules.

To ensure the chatbot behaves reliably, we implemented several **safety and validation** measures. All user inputs are first passed through a filter to detect potentially malicious or irrelevant content

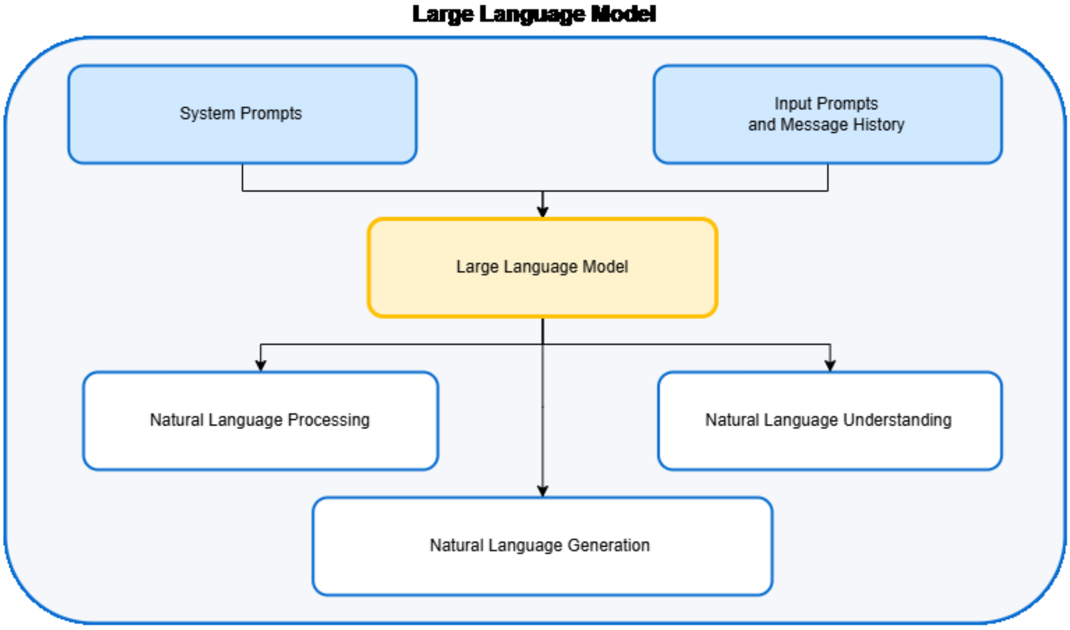

Large Language Model
System Prompts
Input Prompts and Message History
Large Language Model
Natural Language Processing
Natural Language Understanding
Natural Language Generation

Figure 6: Conceptual design of the chatbot's LLM component. The LLM processes user input along with a system prompt (providing guidelines and constraints) and the conversation context. It produces either a conversational response or a formatted action request. The Chatbot Service executes necessary actions (e.g., database queries) and feeds results back into the LLM before finalizing the response to the user.

(for instance, attempts at prompt injection or unrelated queries); suspicious inputs can be sanitized or refused. The system prompt includes explicit instructions to the LLM to refuse requests outside the scope of room booking support or any that would violate usage policies. After the LLM generates a response, the output is checked against expected formats. If the LLM suggests an action like booking a room, the Chatbot Service will require a confirmation step. For example, the chatbot might respond, "Room H.1.1 is available at 10 AM. Would you like to book it?" Only upon the user's explicit confirmation does the service proceed to execute the booking transaction in the backend system. Furthermore, certain critical actions can be flagged for manual review by an administrator (a "human-in-the-loop") if they fall outside predefined safe parameters. This conservative approach helps prevent AI from making unintended changes in the scheduling system and builds trust with users and administrators.

Another aspect of decision-making is handling errors or uncertainties. If the LLM is not confident about user intent (say the request was ambiguous), it is instructed via the prompting strategy to ask clarifying questions rather than guess. For instance, if a user simply says "Schedule a meeting for Room H.1.1," the bot might reply with "Could you provide a date and time for the meeting in Room H.1.1?" To improve user-centered decision-making and guide the user, an action that couldn't be executed will also deliver hints to the LLM on how the user could proceed. This kind of dialogue management is crucial for a good user experience and is made possible by the generative capabilities of the LLM combined with the conversation context memory.

In general, AI-driven decision making in our chatbot balances autonomy with oversight. The LLM provides the flexibility and understanding needed for natural interaction, while the system architecture imposes checks and boundaries to ensure that each decision leads to a valid and safe outcome in the realm of room booking.

5 Prototype Implementation and Evaluation

We developed a prototype of the described chatbot system, integrating it with the existing university scheduling database. The Chatbot Service was implemented in Python, using an API to connect to an LLM (in our case, a GPT-3.5 and Gemini 1.5 series model) for language understanding and generation. The management component was implemented as a Java-based backend system using the Spring framework, facilitating centralized administration and monitoring. The web frontend (Chat UI) was built with Angular and served via a Node.js/NGINX setup, and the Mattermost bot was implemented in Java through Mattermost's bot API. All components were containerized and deployed in a test environment for evaluation. Example chat transcripts illustrating both a successful and an unsuccessful room booking interaction are provided in Appendix A.

In testing the prototype, we focused on common booking scenarios to assess the chatbot's performance.

- **Room availability inquiries:** Users asking if a room is free available at a certain time. The chatbot successfully interpreted various phrasings of such questions (e.g., "I need a lab room next Tuesday afternoon") and returned the relevant availability information after querying the backend.
- **Booking requests:** Users attempting to book a specific room/time. The chatbot could handle multi-turn conversations to gather missing details (if any), confirm availability, and then reserve the room. The enforced confirmation step ensured that no booking was made without user approval, and logs were generated for each booking action for audit purposes.
- **Rescheduling and cancellations:** Users asking to move an existing class or cancel a booking. The bot was able to find the referenced booking (with the help of context or by asking which booking if ambiguous), check alternative options, and carry out the changes. Again, confirmations and permission checks were used for safety.
- **General queries:** Users sometimes asked questions about how the system works or other unrelated queries. The chatbot, guided by its system prompt, provided helpful answers for relevant questions (e.g., "How do I specify the room requirements?") and politely declined or redirected out-of-scope queries and instead informed the user about the chatbots purpose and capabilities.

These tests demonstrated that the LLM-based approach offers a notably **high intent recognition accuracy**. We conducted a structured evaluation of the intent classification across 290 synthetically generated samples, covering three levels of complexity (simple, moderate, and complex). The LLM achieved an overall recognition rate of **89%**. A visual summary of the evaluation results is shown in Figure 7. The scripts for implementation and evaluation are accessible in [6].

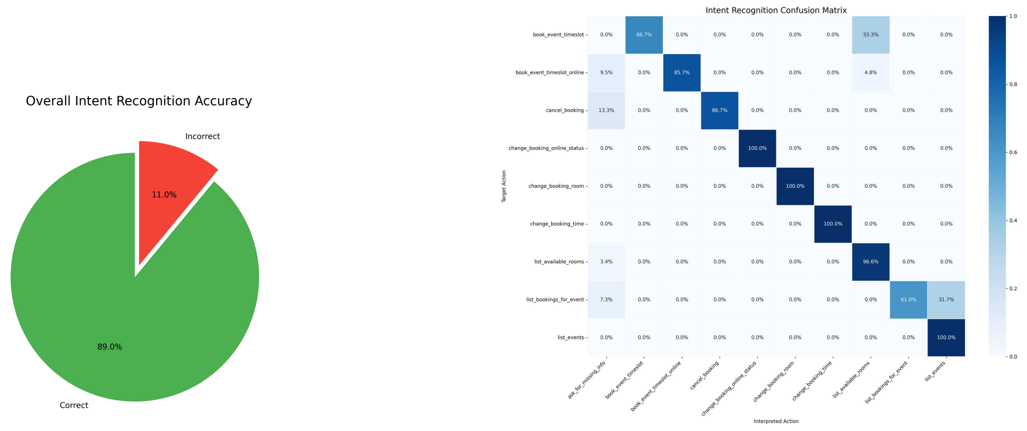


Figure 7: Evaluation summary of intent recognition performance based on 290 synthetic test samples.

It is important to note that these results should be considered as indicative rather than definitive. Since we currently lack real user interaction data and our own tests are inherently biased—due to our *understanding of how to prompt the system to correctly recognize the intent—the evaluation was conducted using synthetically generated samples. While this enables a controlled and systematic analysis, it may not fully capture real-world usage patterns. Therefore, although the results are promising, further validation with real user data is essential to reliably assess the model's performance under authentic conditions.*

The chatbot understood user goals even when they were phrased in colloquial or indirect ways—something that a traditional rule-based system would likely struggle with. As a result, many routine booking tasks could be completed without human intervention, indicating a reduction in both response time and manual effort. For example, what previously involved multiple email exchanges and hours of processing could now be handled by the chatbot in seconds. The list of available intents is provided in Table 1.

Intent Name	**Description**
`list_available_rooms`	Retrieve a list of available rooms based on given time and capacity constraints.
`list_events`	Retrieve all events matching a provided name or number.
`list_bookings_for_event`	Retrieve all timeslots (bookings) associated with a given event.
`book_event_timeslot`	Book a classroom for an event during a specified time slot.
`book_event_timeslot_online`	Book an online event with no physical room assignment.
`change_booking_time`	Reschedule an existing booking to a different time.
`change_booking_room`	Move an existing event booking to another room.
`change_booking_online_status`	Enable or disable online/hybrid status for an event.
`cancel_booking`	Cancel a scheduled event timeslot.
`ask_for_missing_info`	Request missing parameters from user to complete a pending action.
`system_capabilities`	List the full range of supported chatbot actions and commands.
`no_suiting_action`	Fallback if no matching intent can be determined.

Table 1: Mapping of conversational intents to system-level actions within the Chat-Provider architecture. Each action is typed and registered, enabling modular handling and extensibility.

Those actions represent the system-internal operations formally exposed to the chatbot service. However, the actual output presented to the user may differ from the raw retrieved data, as the underlying LLM is capable of performing additional interpretative or generative tasks—such as prioritizing based on constraints, reordering items, or providing context-aware summarizations—during the answer formulation process.

We also compared the two architectural concepts (multi-agent workflow vs. single LLM agent) in terms of development complexity and performance. The workflow-oriented design, had we implemented it fully, would have required a substantial amount of knowledge to maintain due to its complex - over-engineered - architecture. While this solution could enforce strict adherence to procedure (useful for ensuring every required detail is collected), it became apparent that the flexibility of the single-agent design allowed us to easily cover more scenarios.

The single-agent LLM was able to dynamically handle situations that were not explicitly programmed, such as unexpected user questions, turns in conversations, or out-of-order information provision, by relying on its training. However, the multi-agent concept remains a useful mental model, and it could be beneficial in future iterations as soon as the environment is increasing its complexity.

During evaluation, we identified **several challenges and limitations**. One challenge is the LLM's tendency to occasionally generate incorrect or "hallucinated" information, especially if the user query falls outside the narrow domain of room booking. Our safety net of prompts and validation mitigated this to an extent, but it underscores the importance of having fail-safes. Another issue is ensuring data privacy and compliance, as using an AI service may involve sending scheduling data (which could be sensitive) to the LLM provider; deploying a self-hosted LLM model or an on-premises solution might be necessary in the future to address this but it would also require a profound server with good hardware for which we couldn't reach out yet. Performance-wise, the response time was generally acceptable for a prototype (on the order of 1-2 seconds for typical queries), but integrating a large model did come with increased computation costs. We addressed efficiency by avoiding unnecessary LLM calls and integrating multiple steps into a single call.

Overall, the prototype evaluation affirmed that an AI-driven chatbot can substantially improve the efficiency of room booking processes and the realization on how easy it is to build such a system. It also provided insight into the operational considerations for deploying such a system: the need for continuous monitoring, the value of human oversight for critical actions, and the importance of aligning the chatbot's behavior with institutional policies.

6 Conclusion and Future Work

We have presented an AI-driven chatbot designed to facilitate room booking and scheduling through natural language conversations. By integrating a large language model into a room booking system, our prototype can interpret user requests and handle a range of scheduling tasks autonomously, effectively serving as a first-level support agent. This approach has demonstrated clear benefits: routine administrative tasks can be handled in seconds, user satisfaction can increase due to the convenience of a chat-based interface, and support staff can be freed from many repetitive queries, allowing them to focus on more complex tasks.

A key contribution of this work is the system architecture that enables a flexible architecture thanks to a microservice design. The internal services can be exchanged and upgraded with minimal disruption as well as for the existing database as soon as the university would upgrade to another system. The general design of the components is modular, making it feasible to upgrade them (for example, by adding more use-cases).

Our exploration of different design approaches (multi-agent workflow vs. single-agent LLM) provides insights into the trade-offs between structured dialogue management and free-form reasoning. While the single-agent LLM approach proved sufficient and advantageous for our use case, the conceptual exercise of a multi-agent breakdown helped inform our safety strategies and may inspire hybrid approaches in future work. One possible direction is to incorporate a rule-based workflow overlay that triggers the dynamical multi-agent system for unstructured, complex tasks but falls back to a guided single-agent system for critical and simple transaction steps, thus combining predictability with flexibility.

There are several avenues for future improvement. First, enhancing the **robustness and safety** of the chatbot remains paramount. This includes refining the system prompts and constraints to further reduce the chance of inappropriate or incorrect outputs, implementing more sophisticated monitoring of the LLM's decisions, and expanding the library of recognized commands or queries that the chatbot can handle confidently. Second, **scalability and performance optimization** will be important if the system is to be deployed at scale. Techniques such as knowledge distillation or using smaller specialized models for certain subtasks (e.g., a lightweight date parsing model) could help reduce dependence on the large model for every interaction, thereby improving response times and lowering costs. Third, conducting a thorough user study and iterative design based on user feedback would ensure the chatbot's interactions remain user-friendly and useful in a real-world setting as well as it could indicate whether a finetuning of the large language model would be required. Finally, integrating **explainability** features could be valuable – for instance, the chatbot could provide brief justifications for its suggestions (drawing from retrieved data), which would build trust and transparency in an academic environment.

In conclusion, the prototype implementation and evaluation underscore the potential of LLM-powered chatbots to transform scheduling and other administrative processes. By thoughtfully blending AI capabilities with system design and human oversight, we can create intelligent support systems that are not only effective and efficient but also aligned with the requirements for reliability and security in a university context. We believe that this work lays a strong foundation for deploying AI-driven assistants in similar domains and will inspire further research and development in making such systems more robust, accountable, and versatile.

Ethical Considerations

Deploying an AI-based chatbot in university administrative systems does not directly raise disruptive ethical issues that must be proactively addressed. However, there are some aspects we want to highlight when dealing with this topic. One significant concern involves potential biases inherent in the utilized Large Language Model (LLM), which may inadvertently discriminate based on language, origin, or other characteristics [7].

Safeguards must be implemented to prevent misuse, including attempts at AI exploitation (so-called "AI-hacking"), ensuring robust validation through input and output filters and stringent model governance [8].

Social and economic implications of automation, such as job displacement, necessitate careful consideration. Transparent communication and thoughtful management of AI integration are crucial to balance efficiency gains with societal responsibility, ensuring automation remains proportionate and beneficial [9].

Transparency is another critical ethical aspect; users must explicitly be informed that they are interacting with an AI system, especially when the solution involves external or cloud-based components. Compliance with European Union ethical guidelines mandates human oversight for critical or sensitive transactions, reflected in this system through manual approval mechanisms and anomaly detection safeguards [10].

Finally, the environmental impact associated with resource-intensive AI technologies, notably energy consumption by large models, warrants careful consideration. AI deployments should thus be critically evaluated to ensure that the practical benefits justify their environmental costs, supporting responsible innovation aligned with sustainability objectives [11].

Acknowledgement: We thank *Prof. Dr. Peter Braun*, from the Technical University of *Applied Sciences Würzburg-Schweinfurt*, for supervising our project.
AI Acknowledgement: This article was drafted and refined using GPT-4 based on an outline containing related information. The GPT-4 output was reviewed, revised, and enhanced with additional content. It was then edited for improved readability and active tense, partially using Grammarly and DeeplWrite. Diagrams and figures were generated using Deepseek-R1 based on self-created mxgraph models.

References

[1] OpenAI, GPT-4 Technical Report, *arXiv preprint arXiv:2303.08774*, 2023.

[2] S. Yao, et al., ReAct: Synergizing Reasoning and Acting in Language Models, *arXiv preprint arXiv:2210.03629*, 2022.

[3] S. Shen, Z. Liu, Y. Li, et al., HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face, *arXiv preprint arXiv:2303.17580*, 2023.

[4] S. Fan, X. Cong, Y. Fu, et al., WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models, *arXiv preprint arXiv:2411.05451*, 2025.

[5] A. Cockburn, Hexagonal Architecture (Ports and Adapters), *HaT Technical Report 2005.02*, 2005.

[6] Github *Intent Recognition and Evaluation*, 2025. Available at: `https://github.com/PeterMoehle/Intent-Recognition-and-Evaluation`.

[7] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, A. Galstyan, A survey on bias and fairness in machine learning, *ACM Computing Surveys*, 54(6), pp. 1–35, 2021.

[8] W. Luo, S. Dai, X. Liu, S. Banerjee, H. Sun, M. Chen, C. Xiao, AGrail: A Lifelong Agent Guardrail with Effective and Adaptive Safety Detection, *arXiv preprint arXiv:2502.11448*, 2025.

[9] D. Acemoglu, P. Restrepo, Automation and New Tasks: How Technology Displaces and Reinstates Labor, *Journal of Economic Perspectives*, vol. 33, pp. 3–30, 2019.

[10] European Commission, *Ethics guidelines for trustworthy AI*, High-Level Expert Group on Artificial Intelligence, 2019.

[11] E. Strubell, A. Ganesh, A. McCallum, Energy and policy considerations for deep learning in NLP, *arXiv preprint arXiv:1906.02243*, 2019.

Sophie Geisler
Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail: *sophie.geisler@study.thws.de*

Peter Möhle
Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail: *peter.moehle@study.thws.de*

Marcel Wernisch
Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail: *marcel.wernisch@study.thws.de*

Felix Zorn
Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail: *felix.zorn@study.thws.de*

Appendix A: Example Chat Interactions

This appendix presents two chat-based interaction examples: one resulting in a successful room booking, and one that ends without completing the task. These illustrate how the chatbot manages information gathering, confirmation, and fallback strategies.

A.1 Successful Room Booking

User

Hello, I need a room for March 17, 2025, at 10:00 AM for around 80 students.

System

Hi Peter! How long will your event last?

User

From 10:00 to 11:30.

System

The following rooms are available on March 17, 2025, from 10:00 to 11:30:

- H.1.2 (119 seats)
- I.3.24 (32 seats)

Room H.1.2 meets your requirements. Would you like to book it?

User

Yes, please. It's for the course "Programming II [BWI]", event number 5000220.

System

Perfect. Room H.1.2 has been successfully booked for "Programming II [BWI]" (event number 5000220) on March 17, 2025, from 10:00 to 11:30. Can i assist you with further requests?

A.2 Unsuccessful Room Booking

User

I'd like to book room H1.3 on July 1 for 80 people for 120 Minutes. Eventnumber 3651100.

System

I‘m so sorry! I couldn‘t find any available rooms that meet your specification for the booking of Room H1.3 on July 1st, 2025, at 1 AM for 120 minutes with a capacity of 80 people. It seems that there‘s a conflict in the system for that time slot and room size. Could you perhaps consider alternative dates or a different room?

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

WebXR-Previs: Low-Cost System for Real-Time Previsualization and Broadcast Graphics

Felix Husac

Abstract

Cinematographic previsualization (previs) and real-time extended reality (XR) graphics are powerful tools in filmmaking and television broadcasting. These technologies enable the integration of virtual 3D objects into the real world in real time, and in relation to the viewer's perspective. In filmmaking, they are used to explore virtual production sets and plan visual effects sequences. In live broadcasts, they are used to enhance the production value and engagement of a show. Professional video XR systems rely on expensive tracking hardware, dedicated cameras, real time rendering servers and complex software pipelines. Our approach uses readily available consumer hardware: an Android smartphone for spatial input and a desktop PC for rendering. This paper presents the architecture and implementation of a flexible, web-based system for real-time 3D visualization applicable to both cinematographic previsualization and live interactive television graphics. The smartphone camera's pose data (position and orientation) is transmitted in real time via WebSockets to a computer. A Three.js application on the PC receives this data, translating the phone's physical movements into virtual camera control for previs or real-time 3D graphics for live XR applications. This approach, based on standard web technologies, aims to lower the barrier to entry by offering powerful real-time visualization in a cost effective manner, without the need for industry standard equipment.

1 Introduction

Previsualization (Previs) is a technique used across the film, video game and television industries to visually plan and refine a scene before the sets are built and the actual production begins. It allows directors, producers, cinematographers and other creatives to experiment with staging, camera angles, lighting and choreography before live shooting begins. It is mostly used as a cost saving technique, enabling elaborate 3D and motion sequences to be developed and helping to convey the director's artistic vision in the real world. Traditional methods like storyboarding - drawing scenes and camera angles by hand - and mood boards - collages of pictures and videos - while instrumental in the film industry, are inefficient in accurately describing complex camera movements, complex scene setups, visual effects, character movements and they are not interactive, requiring a visual artist to draw each reference frame by hand [13, 21]. Previs overcomes these inefficiencies by offering dynamic, often real-time rough visual templates that can be easily adjusted. Previs scenes are usually created inside dedicated 3D software. Traditional previs tools can be expensive, oftentimes requiring specialized software, hardware and knowledgeable staff to operate them. This makes achieving state-of-the-art results difficult without complex, costly setups that are cumbersome and inconvenient to move or adapt to different environments. For example, techniques such as infrared motion capture for previs is impossible to do outside of a controlled studio volume [9, 26].

The pioneers of previs in film include influential directors such as Peter Jackson, James Cameron, and George Lucas, as well as visual effects companies like Industrial Light & Magic (ILM) founded by George Lucas. Their work helped shape the development and adoption of previs techniques that later expanded into television production as hardware and software technologies advanced. The first 3D computer previs was used for *Star Trek V* (1989) with primitive 3D modeling software [3]. George Lucas described previs as a process that allowed him to "put scenes together without having to shoot them, see how they fit in the movie and then, if they work, I can cut them in and actually go out and shoot them"
[10].

In previs, virtual assets are composited with real actors and backgrounds, allowing directors and camera operators to preview complex, virtual shots in real time, improving creative freedom and production efficiency. Compositing is the process by which two or more graphical elements are combined to create a final image [8].

In this paper, we propose a low cost and minimal setup solution for achieving real time 3D previs, using only an android smartphone and a computer. The solution detailed here is a web based, novel and accessible alternative for real-time 3D previs. It makes use of the smartphone's camera and sensors in order to track its position in the real world, and relays the spatial coordinates in relation to a designated target in real time to the computer, which renders the scene from the real life camera perspective. This allows a real time AR-tracked virtual camera to control the 3D scene viewed on a desktop. Chapter two will briefly discuss the state-of-the-art techniques used in professional motion capture, 3D software and previs. The third chapter will detail the features, design and the user interface of the proposed application and chapter four will focus on the key technologies used to create this solution. The fifth chapter will briefly walk through a normal use scenario, and the last chapter will conclude this paper and describe future development opportunities for the proposed solution.

2 State-of-the-art for real-time 3D graphics

"The Mandalorian," a groundbreaking TV series, integrated Unreal Engine 4, a state-of-the-art game engine, into its StageCraft virtual production system [4, 12]. This enabled the creators to create real-time immersive virtual sets and plan complex camera work. Unreal Engine 4, developed by Epic Games and now on its 5th iteration, offers powerful real-time rendering capabilities and has become an industry standard for virtual production due to its detailed asset management and realistic lighting features [17]. For the film "Thor," the previs work was led by The Third Floor, a specialized previs company that used Autodesk Maya and MotionBuilder software [23]. Their proprietary extensions to these tools advanced the capability of previs by integrating 3D models with camera lens data and green-screen specifications to streamline shoots. "Ford v Ferrari" is another example where cutting-edge previs technology played a key role, leveraging Unreal Engine to create fast, interactive visualizations that helped directors craft high-speed race sequences with accuracy and efficiency [12, 16].

There are several powerful software options currently used for live 3D graphics generation, both for previs and virtual production. Unreal Engine 5 is a free to use 3D rendering game engine, with royalty-based licensing. Its most attractive features are technologies like Lumen and Nanite, features that provide realistic lighting and enable the use of high-polygon count models in complex scenes. It requires specialized game engine knowledge to use [28]. Autodesk Maya, in contrast, is subscription-based and considered expensive, but it is the industry-standard for detailed animation and modeling [1]. Maya has a steeper learning curve due to its complex, professional-grade interface and toolset, but integrates well with live 3D workflows through plugins and pipeline tools. Blender is a fully free and open-source 3D modelling software, used for design, animation and even editing, making it the most financially accessible option. It has a steep learning curve due to its user interface, but it is highly customizable and growing in popularity with independent creators and smaller studios [15]. Its integration into live 3D rendering for previs is improving, but it generally pairs with game engines for final interactive visuals. FrameForge, designed specifically for storyboarding and previs, is a paid specialized tool with a simpler, easy to use interface, designed for filmmakers new to 3D [34]. However, it lacks the advanced real-time rendering and detailed 3D modeling of the other three.

Professional camera tracking solutions like Ncam, Mo-Sys, Vicon, and OptiTrack play a critical role in virtual production by precisely capturing the real camera's position, orientation, and lens metadata. This data is crucial for rendering the virtual elements from the same perspective as the real-world camera. Ncam and Mo-Sys offer sophisticated hybrid tracking systems combining optical and inertial sensors for real-time, low-latency data transmission essential to live LED volume and augmented reality shoots [18, 22]. Vicon and OptiTrack are known for their high-precision motion capture setups, using passive or active markers and multiple cameras to achieve ultra-low latency and accuracy in studio environments [20, 30]. Tracking data collected by these solutions needs to be streamed in real time to the rendering engine. Free-D and VRPN are industry-standard protocols that are used both in movie production and live television [35]. In addition to streaming the camera's position, additional tools are needed for lens calibration and frame syncing.

Not only film productions, but also videogames and live TV productions are also integrating techniques such as extended reality (XR) and augmented reality (AR) into their workflow pipelines, transforming sets and stages into immersive productions. The term virtual reality (VR) encompasses a wider range of techniques and technologies that allow virtual objects to interact or to be displayed on top of real world objects. VR fully immerses the user in a digital environment that blocks the real world. This is often achieved with virtual reality goggles. Augmented reality (AR) overlays digital information or virtual objects onto the user's field of view of the real world, and mixed reality (MR) defines a closer relationship between the real world and the virtual objects, allowing both to interact more deeply.

With the advent of more powerful computers and aided by other technological advancements, such as live camera encoding, 3D tracking, markers and infrared cameras, classical previs techniques found their way into live television. After being perfected for use in live scenarios, these modern technologies found their way back into cinema production, where camera tracking techniques are employed alongside game and rendering engines such as Unreal Engine 5 to create live, in camera CGI in LED volumes, a technique that replaces traditional chroma keying and post production visual effects [12]. Virtual backgrounds are displayed onto led screens and then captured by the camera that also films the actors.

In live television, previs has been integrated into extended reality (XR) and augmented reality (AR) workflows. The studio floor is the 3D canvas where virtual graphics and assets are combined, staged, and then composited on top of the real life video feed. This technology enhances the visual experience of the viewers, makes the broadcast more engaging and reduces costs by bypassing post production. Live 3D graphics are mostly used to better explain infographics, events, sports and weather and political debates [2, 29].

In the recent past, such systems were accessible only to high budget productions. Now, even singular individuals with relatively low budgets can use software such as Unreal Engine 5 and HTC Vive trackers to create their own virtual production stages, but the system requires a lot of tweaking and experience to work [31]. For a professional live environment, such as a broadcast studio, the technological needs are far more advanced, and the cost of a live 3D system rises quickly. A tv studio needs encoder boxes for the cameras that can track the position of the camera in real time. Encoders monitor in real time the FIZ - Focus, Iris, Zoom - and the camera's exact orientation in space. This data is used to render the digital elements from the exact point of view of the real life camera. Rendering servers are needed in order to process the original video feed, position data and graphics in real time, and infrared detectors or AI enhanced stereoscopic cameras for talent tracking in the studio are also employed, to ensure that a 3D virtual object will not be generated behind or on top of a tv presenter or anchor [6, 7, 25].

The proposed app comes as a cheap, simple and effective alternative to the costly studio grade solutions. It integrates 3D graphics and animations inside real time video, using only an android smartphone, a laptop and an internet connection. This system circumvents the use of expensive camera encoders, tracking IR cameras and other expensive legacy hardware, and offers a free, entry level alternative into live previs for film and tv. The application also integrates a simple WYSIWYG 3D staging interface for setting up scenes and environments, based on the three.js javascript library and targeted at users that are inexperienced with traditional 3D modelling and animation software.

3 Application design

The proposed application is split into a phone module and a desktop module, based on the client-server architecture. The android phone acts as a client, and the PC can act as both server and client. The server is responsible for creating the socket connection for the incoming data from the tracker, and establishing subsequent socket connections to multiple viewers and interfaces to broadcast the deserialized and unpacked telemetry (Fig. 1). Its role is as an orchestrator, receiving the broadcasted tracking data, interpreting it and sending it along other socket connections to the viewer interfaces. This ensures that the data is synchronized and interpreted in real time.

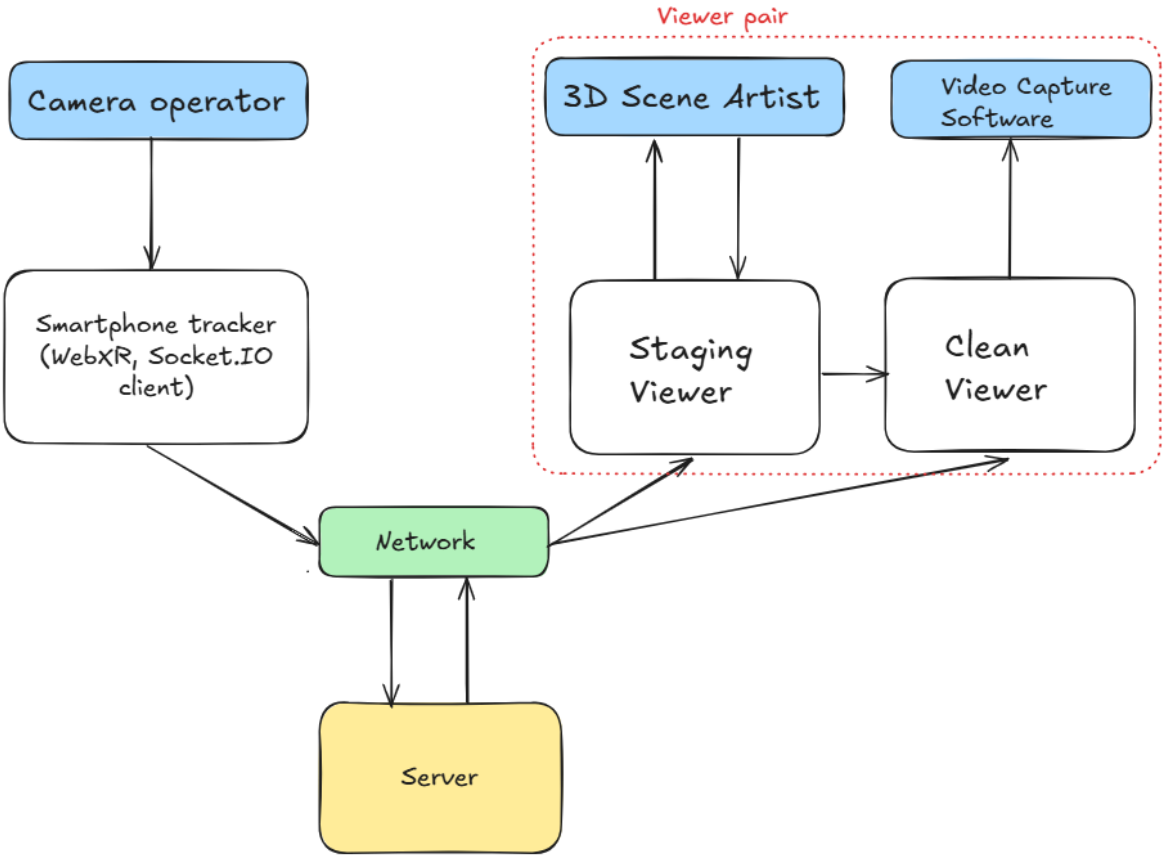


Fig. 1 - A high-level overview of the information flow between app modules

3.1 Phone tracker

The phone module is developed using simple, legacy web technologies - HTML, CSS and JavaScript. The module is powered by the WebXR API, first developed by Google and other collaborators around 2014-2016 [32]. WebXR is traditionally used for creating immersive extended reality experiences. For this application, WebXR is employed to calculate the real-world position of the camera and the phone orientation, and relaying that information back to the server via websocket connection.

The phone module UI is composed of a blank page, that once successfully connected to the server displays a ”Start AR session” button. Once inside the real time session, the camera view is displayed on the screen, and after calibrating the plane detection algorithm by moving the phone around, a target reticle is displayed on top of the camera view. The reticle marks a detected plane surface. The user taps on the screen to place an orange cylinder on the detected plane at the position of the displayed reticle that will act as the virtual world’s origin point.

The phone serves as a motion capture device and real time relay. It calculates relative pose in real time, serializes the data and sends it to the desktop server via websocket to be interpreted, interpolated and applied to the virtual 3D camera. The phone does not receive any data from the server except a successful connection flag. This ensures light weight computation on hardware limited devices. The 3D cylinder placed by the users disappears from the scene in 30 seconds, freeing additional rendering resources on the phone. Data is streamed un-synced to the server so there is less overhead, in case the user has a lower powered device. The movement data is synced to common tv framerate values serverside, using interpolation and smoothing functions in case the phone can't keep up or drops intermediary frames. The device used for the development of this application is only capable of calculating and streaming position data at ~30fps, and the framerate used for the final render is locked at a constant 50fps. Telemetry data, such as relative camera position, orientation and tracking data framerate is displayed in real time in the staging environment on the desktop client.

During use in the studio, the phone is placed directly under the television camera's lens, aligned with the FOV (field of view) of the camera. Three.js offers lens frustum and FOV configuration, so precise lens calibration is possible, if needed.

3.2 Desktop module

For convenience and ease of use, and also due to the separation of concerns principle, the Desktop module of the application is again divided into two main components. The server side, and the desktop client side - composed of two separate interfaces, a staging interface and a clean output interface.

3.2.1 Staging interface

The staging interface is a fully fledged, easy to use WYSIWYG 3D interface that allows the user to create, organize and animate 3D objects in a virtual environment with the help of three.js. Its purpose is setting up the 3D scene and visualizing the live camera moves tracked by the android phone. The staging interface allows the user to add primitive objects, load custom 3D models, environment maps and create display panels for text and still graphics - based on a custom built external canvas editor for HTML elements. The core UI of the interface uses the lil-gui.js library, a well known library used in conjunction with three.js, that helps create menus and controls for object properties and enables the creation of custom menus. The UI is displayed on the left hand side of the screen, while the telemetry data that is received while the phone tracker is active is displayed on a banner on the top right corner of the viewport. Lil-gui has controls for dropdown-selection, sliders, buttons and text boxes. Alongside the usual controls for an object, a custom proprietary waypoint animation control, with multiple motion path options was added. Keyboard controls for camera movement and orbiting, using standard gaming keys for intuitive controls, were also integrated. Another notable feature is GLTF model loading for non-primitive 3D objects sculpted in other 3D software, allowing high quality 3D assets to be imported into the application. A submenu for texturing primitive three.js meshes with still images was also provided. HDRI map support has also been added, for a realistic lighting effect over the whole scene, independent of the lights that three.js already comes with.

For the 3D view, an integrated grid plane serves as the ground of the virtual environment. Hit-testing was also implemented, so that all 3D objects and lights inside the scene are clickable and interactable. Once an object is selected, gizmos for rotation, translation and scale are displayed for that particular object, as well as the side UI updating to display controls for the selected objects. If an object is difficult to select on screen, the user can opt to select it from a dropdown containing all the objects in the scene, by its unique name. The interface allows the user to edit transform controls and material properties, as well as animation controls for rotation and waypoint animations.

The persistence aspect of the desktop client is handled by the browser's localstorage. Scenes are saved, restored and deleted within the browser, making the staging interface setup available to the clean output interface in real time.

The virtual three.js camera displays the 3D scene from the point of view of the tracking device in real time. Both the staging and the clean output interfaces are linked to the same background logic that handles syncing and interpolating the tracking data to deliver a constant framerate output on both interfaces at the same time. Moving the phone in the real world moves the virtual camera on the desktop client interfaces.

3.2.2 Clean output interface

The clean output is a 1-to-1 copy of the staging environment, without the UI and helper overlays and lacking user interaction. This offers a clean, unobstructed view of the 3D scene for previs or recording. It can also load and play animations set in the staging environment automatically. The clean output interface is integrated into the video capture software that will composit the final image. In this particular case, OBS (Open Broadcast Studio), where the clean output page is imported as a browser source on top of the real time video stream from the professional camera in the studio [5].

The proposed solution offers unparalleled flexibility, allowing for multiple client viewer interfaces to be connected to one server on the same network or device. All viewer interfaces run at the same time, in separate staging-clean out pairs, allowing different scene setups while being totally independent of one another, relying only on the same tracking data from the server. It is an all in one application that combines real time tracking, communication, telemetry, synchronization, 3D scene setup and rendering, covering the full previs pipeline in a simple and intuitive manner for inexperienced users. It is cost effective, relying only on an android smartphone and a desktop PC or laptop.

4 Technologies and implementation

The technologies used to implement the application are simple web technologies, already successfully used in other modern web and mobile applications. Node.js and Express.js were used for the server side, while HTML, CSS, and JavaScript were used for the clients. The phone tracker also made use of the WebXR API. This approach was chosen due to the tools being easy to use, having extensive documentation, offering high integrability and adaptability between the components already stated above—being all web technologies—and because of previous experience in developing web applications. The chosen technologies can cover the entire operational pipeline, from backend processing and API handling to frontend user interaction and augmented reality features from pose capture, communication, 3D scene setup, display and integration into an existing video streaming and recording software.

4.1 Client-side

The client side consists of the Desktop viewer interfaces and the phone tracking interface. HTML, CSS and JavaScript were used to create the basic functionality of the modules, creating the underlying structure for the necessary libraries to interact with the components and data of the application. The Socket.io library was used to enable real time bidirectional communication between the tracker and the server, and broadcast communication between the server and the viewers [24]. The three.js library was employed for the core 3D functionality of the app, from rendering 3D scenes, to importing models, lights, materials and virtual camera setup [27]. Lil-gui.js was used as a starting point for creating the menus and submenus related to objects, lights and the 3D scene as a whole [14]. Display panels inside the Staging interface had their own separate design interface, based on the html2canvas library, for rendering HTML content onto a canvas texture that could be loaded into the virtual scene [12]. The WebXR Device API was used to access the phone's camera and motion sensors, and to calculate the position data. An XRSession can be started by the user after the server acknowledges the connection between itself and the phone tracker, starting the camera capture and analyzing the sensor data. It uses this data to make references of the real environment around it and to find solid plane surfaces to place anchors. Reference spaces are coordinate systems that allow the tracker to interpret physical movement differently. The "local-floor" reference space provides a stable coordinate system relative to the ground, allowing virtual content to stay aligned with the physical environment, while the "viewer" reference space tracks the device's position in space for accurate pose data [33]. Through hit-testing, the application performs raycasting from the device into the physical world to detect real surfaces where virtual objects can be anchored. Anchors are then created at these points to ensure that virtual elements remain fixed in place relative to the environment, even as the user moves or the device's pose changes. The reticle functionality briefly presented in the previous chapter makes use of this technique to align the virtual environment to the origin point the user chooses by tapping the phone screen in the desired location on the detected ground plane. The extracted data - position vector and rotation quaternion - is serialized and sent via the websocket to the server as fast as the phone can process the sensor and camera inputs and a new measurement is taken.

4.2 Server-side

Node.js is a widely popular JavaScript runtime environment, used for developing server-side web applications [19]. The Express.js framework is used to serve files and handle HTTP requests made by different modules inside the application [36]. Socket.IO on the server side manages websocket connections and orchestrates data exchanges. The server uses HTTPS to ensure secure encrypted communication because WebXR sessions require a secure context to access sensitive device APIs like the camera and motion sensors. This security requirement prevents unauthorized interception or tampering with data streams during augmented reality sessions. The data sent by the tracker is deserialized, interpreted and sent to the viewer clients who process it in the background, applying interpolation and smoothing functions. After processing, the positioning data is applied to the virtual camera, thus mirroring the real life camera's position, orientation and rotation, and the scene is rendered.

4.3 Challenges faced

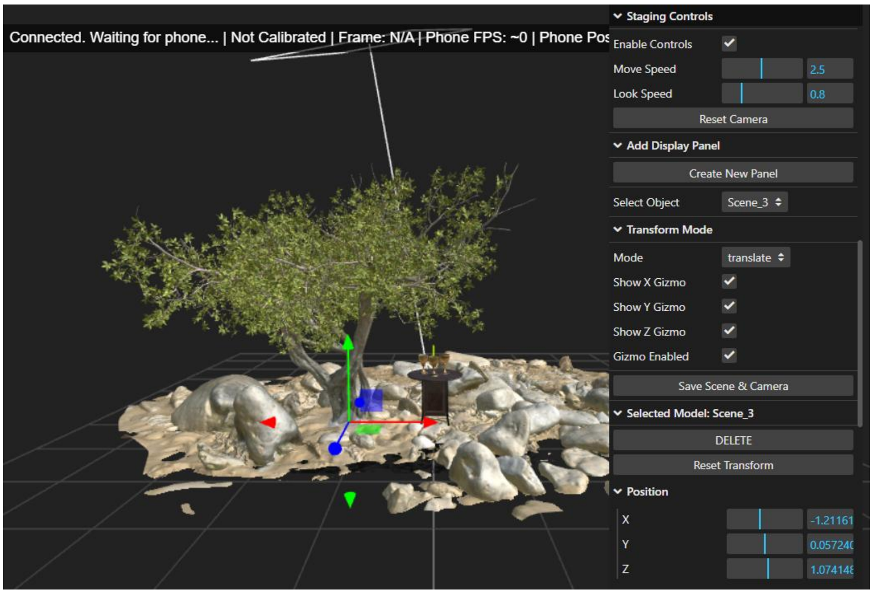
Scene management and persistence in staging mode was a crucial aspect of the implementation. Adding objects, creating unique ids, selecting, modifying object parameters, saving, recalling and synchronizing object behaviour across both staging and clean output interfaces was a difficult and complex part of the development stage.

Complex 3D objects are stored as paths to local files inside LocalStorage, while lights, simple geometries, primitives, lights and animation parameters are stored serialized inside individual JSON wrappers. Events are fired on each create, update and delete operation inside LocalStorage.

Creating display panels locally inside the application was another important milestone. The aim was to simplify the process of creating visual elements and remove the need for the user to create the graphical elements in an external software, export them as a picture and then import them into the scene as a texture. To meet this need, display panels have their own WYSIWYG editor that can be used similar to a presentation slide editor. Users can create text fields, insert images, adjust font families, sizes and colors, and change the z-order of the elements. In its essence, the display panel designer is a div-editor with a JSON output for saving the position and relevant parameters of the graphical elements placed on it. The resulting HTML DIV element is then converted into a canvas with the help of html2canvas library. The canvas texture is then applied to a plane geometry inside the virtual scene, which can then be manipulated just as any other 3D object.

5 User flow

In order to use the application, the following steps must be followed. Firstly, the server must be started. Once the server is running, the tracker and the PC (or PCs if multiple viewers are needed) need to be connected to the same network. The user then opens the Staging viewer interface and starts building the 3D scene. The user can import GLTF 3D models, three.js primitives, he can add textures from pictures, change colors, add lights and shadow casting, and import HDRI maps for realistic environmental reflections and lighting (Fig. 2). After he is happy with the results, the user saves the scene and connects the phone tracker by launching the application in a browser page. After the server acknowledges the connection, the user can start the XR session. The phone needs to be calibrated, so the user will need to move the camera around and tilt the phone in various angles until the target reticle appears on screen. The user then aligns the virtual plane of the reticle with the floor plane, and taps the screen where he would like the virtual scene's origin point to be. Once he does that, pose data starts being sent by the phone to the server, the virtual camera inside the viewer interface starts mimicking the movements of the real world camera, and telemetry is displayed on the staging interface. In 30 seconds, the visual aids on the phone screen are removed, helping unload computational power and enabling faster pose processing and transmission. If a previous scene exists, the user can connect the tracker directly, without needing to first design one. While tracking, the user can keep editing the virtual scene inside the staging view, with all available functionality. The changes are synced on the clean output page, and visible in the final composited view inside the video capture software.



Connected. Waiting for phone... | Not Calibrated | Frame: N/A | Phone FPS: ~0 | Phone Pos

Enable Controls ☑
Move Speed 2.5
Look Speed 0.8
Select Object Scene_3
Mode translate
Show X Gizmo ☑
Show Y Gizmo ☑
Show Z Gizmo ☑
Gizmo Enabled ☑
X -1.21161
Y 0.05724
Z 1.07414

Fig. 2 - Staging interface, showing the directional light beam, the translation gizmo for the selected object (tree) and a part of the lil-gui interface on the left hand side. Top right, the telemetry banner shows the tracker is not yet connected.

6 Conclusions and future developments

The app described in this paper comes as an inexpensive alternative to a complete live 3D pipeline, integrating all features and components necessary for real time previs and television graphics. It supports multiple processes, ranging from position tracking, data transmission, synchronization, virtual camera syncing, scene setup, rendering and composition into a final shot, all by using simple, widely available web technologies. The app is composed of three main components: the tracking module empowered by the WebXR Device API, the simple express.js server, and the desktop viewer interfaces for staging and clean output. Current limitations of this version include no advanced physics besides the waypoint animation system and a basic WYSIWYG editor. Future developments include creating a separate Controller interface for triggering complex animations, spawning objects in and out of the scene remotely and configuring saving a loading of show files, to keep created scenes organized. For advanced animations, integrating a tweening library like GSAP is an important step in making the movement of the objects feel more professional. For more interesting scenes, or where special effects are needed, integrating custom shaders and particle generators, such as smoke, fire and rain are another development requirement. In order for this tool to benefit and positively impact filmmakers, small studios and visual artists, multi-user collaboration support is another essential feature for further development.

Acknowledgement: This work was supervised by Professor *Dana Simian*, from *"Lucian Blaga" University of Sibiu*.

References.

[1] *Autodesk Maya,* Official Product page, accessed on 08.11.2025 https://www.autodesk.com/products/maya/overview

[2] *Augmented Reality Solutions*, Wtvision Official Product page, accessed on 08.11.2025 http://wtvision.com/solutions/augmented-reality

[3] *Become a Previs Artist*, ESMA Blog post, accessed on 08.11.2025 https://www.esma-artistique.com/en/metier/become-a-previs-artist/

[4] *Behind the Scenes of The Mandalorian: The Role of Unreal Engine in Set Creation*, Artemisia College of Art and Design Blog post, accessed on 08.11.2025 https://www.artemisiacollege.com/blog/mandalorian-unreal-engine-set-creation/

[5] *Browser Source*, Open Broadcast Studio Documentation page, accessed on 08.11.2025 https://obsproject.com/kb/browser-source

[6] *Camera Tracking in 3D and Virtual Production*, GarageFarm Blog post, accessed on 08.11.2025 https://garagefarm.net/blog/camera-tracking-in-3d-and-virtual-production

[7] *Choosing the right camera tracking system for virtual production*, EZtrack Studio Blog post, accessed on 08.11.2025 https://eztrack.studio/

[8] *Compositing – Everything You Need To Know*, NFI Blog post, accessed on 08.11.2025 https://www.nfi.edu/compositing/

[9] *Discovering Motion Capture Facilities: A Full Guide for Creators, Studios & Devs*, Animost, accessed on 08.10.2025 https://animost.com/industry-updates/motion-capture-facilities/

[10] *From Celluloid to Ones and Zeros*, Videomaker Blog post, accessed on 08.11.2025 https://www.videomaker.com/article/c18/17043-from-celluloid-to-ones-and-zeros/

[11] *Html2canvas*, Official Documentation page, accessed on 08.11.2025 https://html2canvas.hertzen.com/documentation

[12] Ian Failes, *How Previs Has Gone Real-Time*, VFXV Blog post, accessed on 08.11.2025 https://vfxvoice.com/how-previs-has-gone-real-time/

[13] Kyle Deguzman, *What is Previs — The Art and Process of Previsualization in Film*, StudioBinder, accessed on 08.11.2025 https://www.studiobinder.com/blog/what-is-previs-previsualization-in-film/

[14] *Lil-gui*, Official Documentation page, accessed on 08.11.2025 https://lil-gui.georgealways.com/

[15] *Maya vs. Blender: Which One To Choose?*, Rocket Brush Studio Blog post, accessed on 08.11.2025 https://rocketbrush.com/blog/maya-vs-blender-which-one-to-choose

[16] Mike Seymour, *Ford V Ferrari stylised previs in UE4*, FxGuide Blog post, accessed on 08.11.2025 https://www.fxguide.com/fxfeatured/ford-v-ferrari-stylised-previs-in-ue4/

[17] Mike Seymour, *Improvements to virtual production in Unreal Engine 4*, FxGuide Blog post, accessed on 08.11.2025 https://www.fxguide.com/fxfeatured/improvements-to-virtual-prodn-in-ue4/

[18] *Mo-Sys*, Official Product page, accessed on 08.11.2025 https://www.mo-sys.com/products/startracker/

[19] *Node.js*, Official Documentation page, accessed on 08.11.2025 https://nodejs.org/en

[20] *Optitrack*, Official Product page, accessed on 08.11.2025 https://optitrack3.payloadcms.app/

[21] *Previsualization in Film and Media*, GarageFarm Blog post, accessed on 08.11.2025 https://garagefarm.net/blog/previsualization-in-film-and-media

[22] *Real-Time Camera Tracking On Location*, Ncam Tech Blog post, accessed on 08.11.2025 https://www.ncam-tech.com/real-time-camera-tracking-on-location/

[23] Rick DeMott, *The Third Floor Provides Previs Might For Thor*, AWN, accessed on 08.11.2025 https://www.awn.com/news/third-floor-provides-previs-might-thor

[24] *Socket.IO*, Official Documentation page, accessed on 08.11.2025 https://socket.io/

[25] *Stype*, Official Product page, accessed on 08.11.2025 https://stype.tv/

[26] *The complete guide to professional motion capture*, Rokoko, accessed on 08.11.2025 https://www.rokoko.com/insights/the-complete-guide-to-professional-motion-capture

[27] *Three.js*, Official Documentation page, accessed on 08.11.2025 https://threejs.org/

[28] *Unreal Engine 5*, Official Product page, accessed on 08.11.2025 https://www.unrealengine.com/en-US/unreal-engine-5

[29] *Virtual and Augmented Reality in Television – TV Management*, Fiveable, accessed on 08.11.2025 https://fiveable.me/tv-management/unit-10/virtual-augmented-reality-television/study-guide/rQWyZBJFDSXmJiE7

[30] *Vicon*, Official Product page, accessed on 08.11.2025 https://www.vicon.com/

[31] *Vive Mars Cam-Track*, Official Product page, accessed on 08.11.2025 https://www.mars.vive.com/

[32] *WebXR Device API*, Official Documentation page, accessed on 08.11.2025 https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API

[33] *WebXR Device API: Reference Space and Render Loop*, Educative IO Course page, accessed on 08.11.2025 https://www.educative.io/courses/become-proficient-in-webxr-create-xr-experiences-using-a-frame/webxr-device-api-reference-space-and-render-loop

[34] *What Makes FrameForge so Revolutionary?*, StoryboardSmarter Official Product page, accessed on 08.11.2025 https://www.storyboardsmarter.com/frameforge-features-and-benefits
[35] *What is FreeD and how to use it with your PTZ cameras,* PTZ Optics Blog post, accessed on 08.11.2025 https://ptzoptics.com/freed/
[36] *Express.js*, Official Documentation page, accessed on 08.11.2025 https://expressjs.com/

Felix Husac
"Lucian Blaga" University of Sibiu
Department of Mathematics and
Informatics
5-7 Dr. Ratiu Str, Sibiu 550012
ROMANIA
E-mail: felix.husac@ulbsibiu.ro

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Implementation and Development of a Rated Voting System

Giorgiana-Maria Marangoci, Alex-Andrei Rîpan, Stefan-Ioan Istina

Abstract

A voting system is essential when groups need to make fair, inclusive, and transparent decisions. It promotes order, equality, and cooperation — not just in governments, but in everyday human interactions. Traditional methods such as Plurality voting, Majority voting, Proportional representation may have restrictions that lead to lose of trust and biased fairness on the electoral process. This paper explores a less used method, the Rated Voting System, where voters assign scores to candidates, providing a more refined selection when compared to classical voting systems. We have chosen to highlight the challenges of Plurality and Rating systems, by presenting their characteristics and differences. In order to approach the discussed issues, we have created an application that ranks the candidates using the Plurality voting system and also the Rated Voting System. Our goal was to make use of digital technology in order to cast and count votes when using these two systems.

Keywords: Rated Voting System, Plurality Voting System, e-voting

1 Introduction

At a general level, the importance of a voting system goes beyond politics — it plays a vital role in group decision-making, fair representation, and collective responsibility in many areas of life. For example, in politics, voting is a way for citizens to get involved in the democratic process by having the opportunity to make personal decisions and to affirm their trust in the democratic system [1]. Communities use referendums to vote on local projects, events or laws – bypassing the elected representatives, and therefore directly expressing the power of its members. Organizations use votes to elect board members or to set strategic goals. Universities and schools also use voting systems in order to elect student leaders or to decide on optional classes.

While voting is essential for a functioning democracy, traditional voting systems face numerous challenges. The problems range from technological and logistical difficulties to manipulation of the electoral process. For example, manual counting is prone to errors, which in close elections could affect the outcome; people with disabilities have limited access; ballots transportation may be difficult in conflict areas. Current voting systems also face electoral fraud, as presented in [2]: vote manipulation leads to diminishing the supposed benefits of democratic governance.

Rated Voting System is an electoral model in which participants assign grades or scores to candidates, instead of choosing just one. This system allows for a more refined set of voter preferences, reducing the risk of encountering various events such as strategic voting. This method rooted in the system proposed by the French mathematician and naval engineer Jean-Charles de Borda. He described a voting system related to the total ranking of individual preferences at the time of choice [3], which is one of the ranked methods. Let's consider *n* candidates; the method awards: *n-1* points for the first placed, *n-2* points for the second placed and so on, the last placed receiving *0* points [4]. Even Jean-Charles de Borda confessed that "My system is only intended for honest men", because if the strategic vote were to become a habit, the final results of the voters would no longer reflect the benevolent truth of the citizen [3].

This paper is structured as follows: Section 2 presents several voting systems currently used; Section 3 describes the proposed application for e-voting, and Section 4 concludes the work.

2 Overview of Voting Systems

In this section we present some voting systems currently used.

2.1 Plurality vote

A plurality vote (also called "first-past-the-post") is a voting system where each voter can select only one option, and the option with the most votes wins, even if it does not have a majority (more than 50%). It is a common way to vote, simple and fast, but can lead to strategic vote (people can vote against the option rather than for the desired option). [5]

2.2 Majority vote

A majority vote means an option must receive more than 50% of the total votes to win. If no option reaches a majority, a second round or vote transfer system is often used. This system ensures that the winning option has more than half support, but it may need additional rounds. [5]

2.3 Proportional Vote

The proportional voting system is used when members of multiple groups must make a collective decision when electing a unique board [6]. It is used in corporate governance, academic and professional organizations, NGOs and unions. The voting standards are majority (more than half of board members), supermajority (two-thirds or three-fourths approval), and unanimous consent (everyone must be in favor of the decision). This system allows minority representation, encourages different perspectives in board decisions, and reduces the risk of group control.

2.4 Ranked choice voting

The process that allows voters to rank candidates for a particular position based on their preferences is called ranked choice voting. For example, a voter has the following preferences: option B is in the first position, followed by C, D and A. If none of the options get a majority of the votes in the first round, and option B has the fewest votes, it is automatically eliminated. All the votes of option B are transferred to the second opinion of the voter, in this case being option C. This step is repeated until one of the options obtains a majority of the votes [7]. The Best Picture award from the Academy of Motion Picture Art (a.k.a. the Oscar award) is granted using this type of vote [8].

2.5 Rated Voting System

The rated voting system is a method that allows voters to express their preferences in a more nuanced way compared to traditional voting methods. Without selecting a set of options, the voter expresses personal approval or gives a score to each option [9].

Rated voting system has several variants, each with specific features and mechanisms as voters express their support or preference for options, which offers a more interesting alternative to traditional voting methods. The systems are designed to reflect a wide range of opinions and allow a more accurate, detailed and representative choice, as the voter expresses his or her degree of support.

2.5.1 Score (Range) Voting

Score (Range) voting is a system in which voters evaluate options on a scale, and the option with the highest total score wins. A common scale can be from 0 to 10, where 0 represents total disapproval and 10 indicates total approval. Other used scale is from -10 to +10, in which negative scores express disapproval, and positive scores indicate support [10]. Score voting is usually used in academic peer review panels for grant applications or paper submissions at journals and conferences.

2.5.2 Approval Voting

This system allows voters to express their support for multiple options in the voting process, which means they are not limited to polling a single option or ranking them; the option with the most approvals wins [9]. It is used in online pools in gaming communities, by corporate governance, or by academic and professional organizations for electing board members, committee members, or conference topics (for example, for electing the IEEE boards [11]).

2.5.3 Benefits of the rated voting system

One important advantage is the reduction in the disclosure of votes, because participants can support more options. It allows voters to communicate their different levels of support for multiple variants, providing a more in-depth representation of their preferences. Also, people tend to be more accustomed to assigning scores or grades (like for product reviews) than constructing rigid rankings among a number of options [12].

2.5.4 Disadvantages of the rated voting system

The system can be complicated for the elderly or less technologically-inclined people. Yet, civic training is known to be crucial in ensuring proper understanding and adoption across all levels of age and experience. In terms of risk of strategic inflation of scores (e.g., voters giving uniformly high scores to favored candidates and low scores to all others), some authors propose the median as the aggregation method, rather than the average, to minimize the influence of high or manipulative ratings. This approach makes the system significantly less vulnerable to strategic voting when compared to other methods [13].

3 DualVote: Hybridizing the Voting Process with New Technologies

Like many other current social transformations, the voting process is an area where the technological transition faces challenges. One of the most significant difficulties is to guarantee to anyone fair access to the voting process and outcomes. To solve one of the problems with the current voting system, we created *DualVote*. This app uses both the traditional method of selecting a single option, and a method of assigning scores to several options, which makes it easy and flexible for everyone. This app is not just a technological tool; by comparing its results it can be used in designing new selecting methods in social life, adapted to specific contexts.

This application was designed with the main goal of exploring two different voting systems, and can be used in any voting process, not only political. We have chosen to implement an imaginary political vote mainly because it is a very common situation in the citizens' life, with broad implications at both local and national levels.

3.1 Frontend

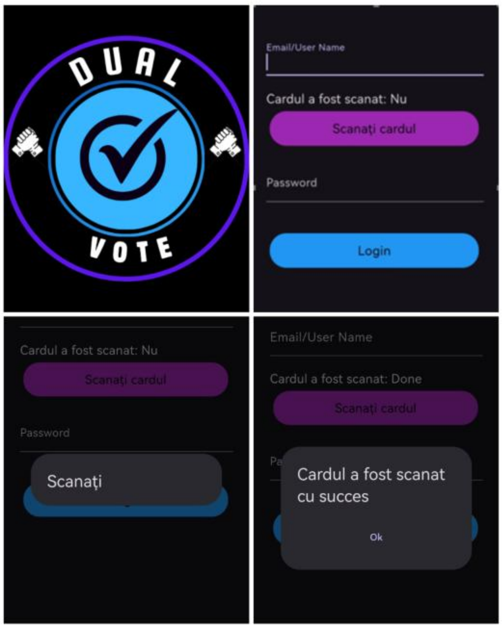
In order to offer a modern and easy-to-use solution, we have designed a voting application that can be used directly from personal mobile devices. The application was implemented with

Flutter framework, based on the Dart programming file. The motivation for choosing this framework comes from several essential factors:

- It allowed us to develop a single application that runs on both the Android and iOS operating systems, without having to separately write the code for each operating system.
- The unified interface and performance gave us a consistent visual and functional experience on both platforms.

In the development of the front-end part, the application was designed to integrate and support the Plurality voting system as well as the Rated voting system, offering to the user a unified and intuitive experience. Regarding the importance of security in the digital electoral process, the authentication mechanism implemented was designed to provide a high degree of protection, reducing the risks associated with unauthorized access and guaranteeing data integrity.

In order to authenticate, users must go through a multi-factor security system that leads to the elimination of the use of accounts by unauthorized persons, combining three essential elements: an official email address that is issued by the managing organization, scanning an NFC card and entering a personal password. For an extra layer of protection, we have also implemented the two-step authentication (2FA) system, requiring the user not only to enter the password but also to scan the NFC card. By performing this action, a unique verification code will be generated, which must be entered for the final confirmation of identity and to combat cyber-attacks, such as phishing. Through these advanced authentication and security measures, the app provides a safe and efficient user environment, ensuring both data protection and compliance with the strict requirements imposed by a modern digital voting system (Fig. 1).



Cardul a fost scanat: Nu

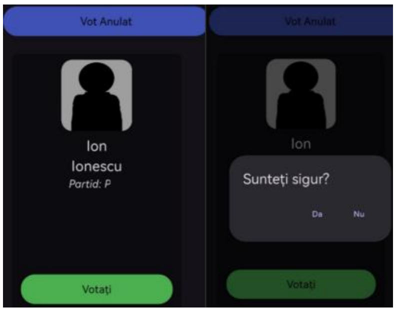
Cardul a fost scanat: Nu

Cardul a fost scanat: Done

Cardul a fost scanat cu succes

Fig. 1 - Authentication system

In the next step, the Plurality system is used, which allows the election of a single option. In the context of this system, candidates are displayed in an intuitive interface, which helps the user to express his vote by selecting the desired candidate. There is also the possibility of canceling the vote: if the user decides not to express his electoral preference, but to prevent accidental cancellations, the system includes a warning mechanism by displaying a message, asking the user to confirm the decision. Only after the explicit acceptance of the annulment, the vote is invalidated and properly recorded (Fig. 2). The difference between a white vote and a canceled vote is that the white vote is validated but does not influence the results of the elections, while the annulled vote is not validated [14].



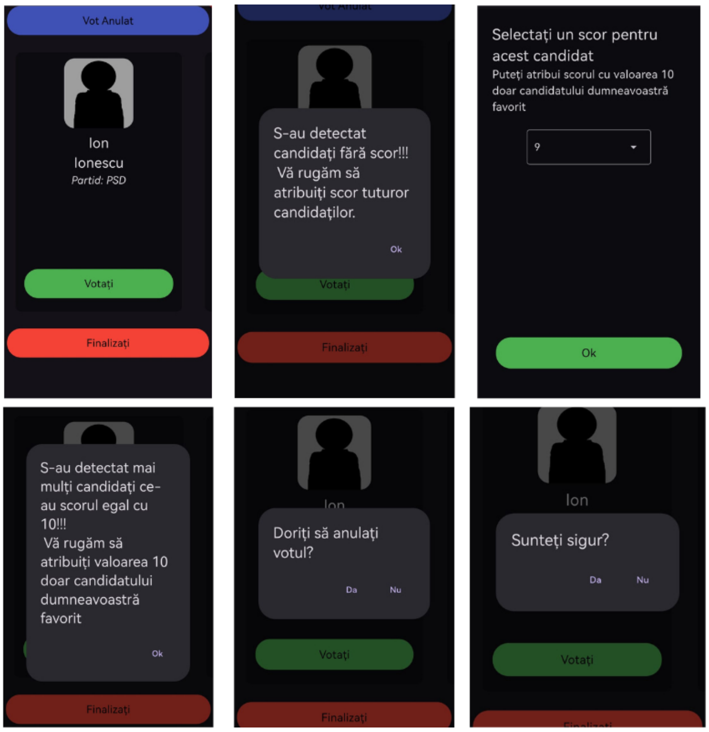
Sunteți sigur?

Fig. 2- Cancelled vote warning message for the candidate Ion Ionescu

The rated voting system, used as an alternative option, allows the citizens to express their preference by giving each candidate a score on a scale from -10 to 10. It allows for a more nuanced evaluation, reflecting both support and opposition. The vote is validated only if some conditions are verified:

- each candidate must receive a score, which guarantees that all participants have been evaluated;
- one and only one of the candidates has to obtain the highest score, which expresses the major preference of the user.

By fulfilling these conditions, the user can confirm the vote, completing the electoral process. To prevent omissions and errors, the application will display warning messages if the user does not assign a score to each candidate, or he wants to cancel the vote, thus generating an accurate, transparent and controlled electoral process (Fig. 3).



S-au detectat candidați fără scor!!! Vă rugăm să atribuiți scor tuturor candidaților.

Selectați un scor pentru acest candidat

Puteți atribui scorul cu valoarea 10 doar candidatului dumneavoastră favorit

9

S-au detectat mai mulți candidați ce-au scorul egal cu 10!!! Vă rugăm să atribuiți valoarea 10 doar candidatului dumneavoastră favorit

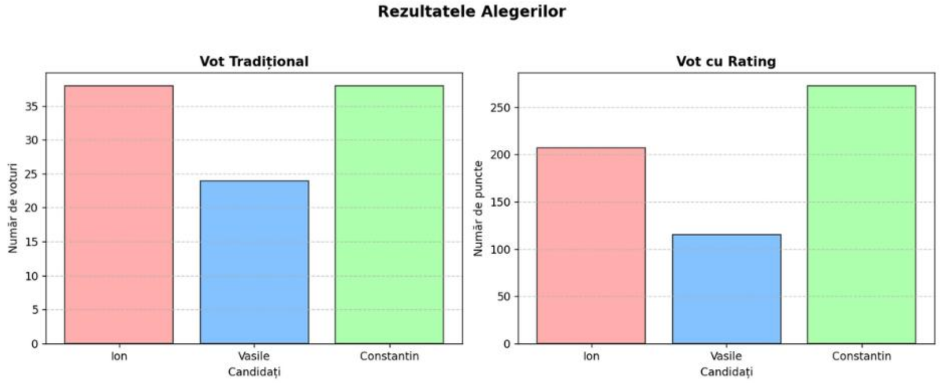
Doriți să anulați votul?

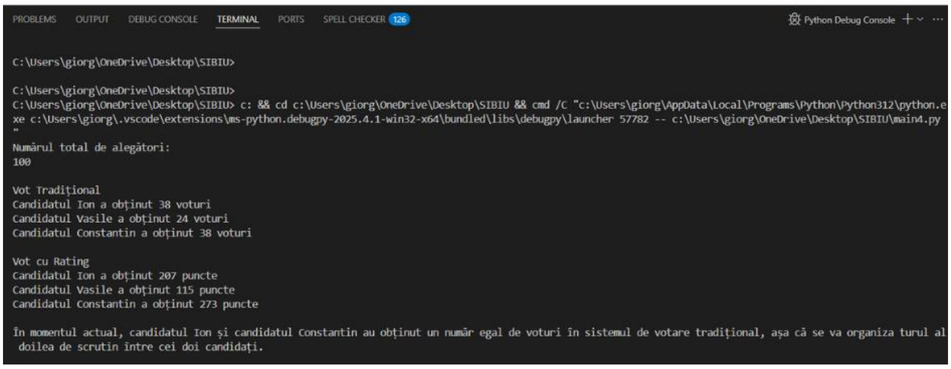
Sunteți sigur?

Fig. 3- Rated voting system interface

In order to simulate the electoral process and present a possible result we included in our voting application a code in the Python programming language. We generated 300 votes on both voting systems (majority and rated). We used the http://random.org/ website to generate the 300 two-types votes, which were saved in a text file. By using the Python program, we read this data from the text file and calculated the results for both voting system, which provided a simulation of a possible outcome of the electoral process. The Python code and the text files are available in the GitHub repository [15].

Fig. 4 presents a first simulation. The results of the votes through the two voting systems highlight the distribution of the scores obtained by each candidate. By processing the text file, we found that two of the candidates obtained an equal highest score through the plurality voting system. Given the situation, a message will be generated in the terminal of the code informing us that a second round will take place between the two candidates with the highest score, in order to determine the final winner, ensuring the transparency of the electoral process. This was an interesting case, showing the more fine-grained candidates' evaluation using the rated voting method. This second voting method did not need a second round.

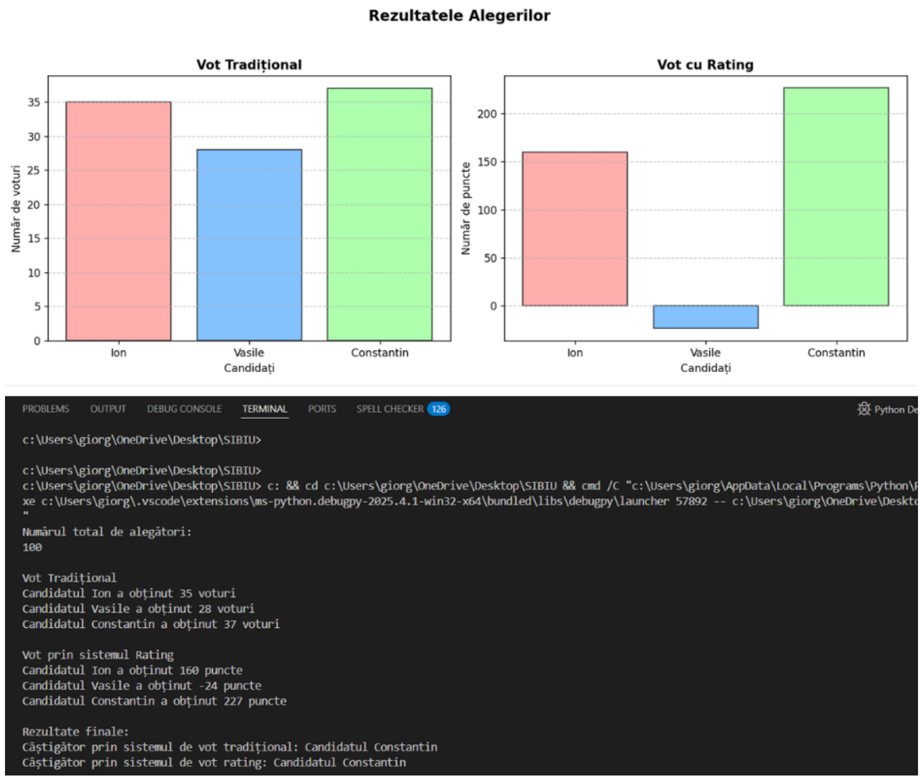




```
C:\Users\giorg\OneDrive\Desktop\SIBIU>

C:\Users\giorg\OneDrive\Desktop\SIBIU>
C:\Users\giorg\OneDrive\Desktop\SIBIU> c: && cd c:\Users\giorg\OneDrive\Desktop\SIBIU && cmd /C "c:\Users\giorg\AppData\Local\Programs\Python\Python312\python.e
xe c:\Users\giorg\.vscode\extensions\ms-python.debugpy-2025.4.1-win32-x64\bundled\libs\debugpy\launcher 57782 -- c:\Users\giorg\OneDrive\Desktop\SIBIU\main4.py
"
Numărul total de alegători:
100

Vot Tradițional
Candidatul Ion a obținut 38 voturi
Candidatul Vasile a obținut 24 voturi
Candidatul Constantin a obținut 38 voturi

Vot cu Rating
Candidatul Ion a obținut 207 puncte
Candidatul Vasile a obținut 115 puncte
Candidatul Constantin a obținut 273 puncte

În momentul actual, candidatul Ion și candidatul Constantin au obținut un număr egal de voturi în sistemul de votare tradițional, așa că se va organiza turul al
 doilea de scrutin între cei doi candidați.
```

Fig. 4 - Status of results with the same score

By processing the results of a second simulation, we obtained clear results from both voting systems, as only one winner was designated. In Fig. 5, the results are represented graphically, but also at the end of the code where the situation is detailed, highlighting the final ranking of the candidates based on the scores obtained. Thus, the results confirm the validity of the electoral process and the correct selection of the winner following the evaluation through both voting systems.



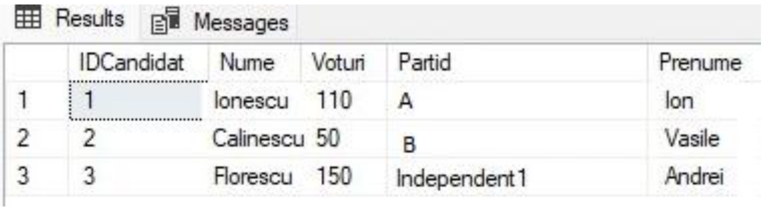
```
c:\Users\giorg\OneDrive\Desktop\SIBIU>

c:\Users\giorg\OneDrive\Desktop\SIBIU>
c:\Users\giorg\OneDrive\Desktop\SIBIU> c: && cd c:\Users\giorg\OneDrive\Desktop\SIBIU && cmd /C "c:\Users\giorg\AppData\Local\Programs\Python\
xe c:\Users\giorg\.vscode\extensions\ms-python.debugpy-2025.4.1-win32-x64\bundled\libs\debugpy\launcher 57892 -- c:\Users\giorg\OneDrive\Deskt
"
Numărul total de alegători:
100

Vot Tradițional
Candidatul Ion a obținut 35 voturi
Candidatul Vasile a obținut 28 voturi
Candidatul Constantin a obținut 37 voturi

Vot prin sistemul Rating
Candidatul Ion a obținut 160 puncte
Candidatul Vasile a obținut -24 puncte
Candidatul Constantin a obținut 227 puncte

Rezultate finale:
Câștigător prin sistemul de vot tradițional: Candidatul Constantin
Câștigător prin sistemul de vot rating: Candidatul Constantin
```

Fig. 5 - *Status of results with sharp and correct results*

3.2 Backend

The Python programming language helped us in the development of the backend, due to its flexibility and efficiency in managing database connections and data manipulation. We used the **pyodbc** module, which allows the application to be connected to the SQL database and linked to the tables that store the candidates' data. With the help of this module, we managed to build and control the SQL cursors that execute the commands *insert*, *update* or *extract* data from tables.

Our database is built using two tables, “Voters” that has the authentication information, and the "Candidates" table that contains the details of the candidates registered in the elections. In *Fig. 6*, it is represented in the form of a tabular structure in a SQL interface. The displayed table is used to analyze the results and to determine the winner.


Results
Messages

	IDCandidat	Nume	Voturi	Partid	Prenume
1	1	Ionescu	110	A	Ion
2	2	Calinescu	50	B	Vasile
3	3	Florescu	150	Independent1	Andrei

Fig. 6 - The "Candidates" table in the SQL database

In order to be able to unify the frontend with the backend, we opted to use Flask, an affordable framework for developing APIs in Python. This framework easily creates a RESTful API, through which the frontend application, made in Flutter, accesses candidates’ data and transmits user votes. The API helps us maintain the connection between the frontend and the backend by managing the application logic and data storage. Through these technologies, the app models a fast, efficient and flexible component able to adapt to the requirements of a modern electronic voting application.

3.3 Numerical hands-on simulations

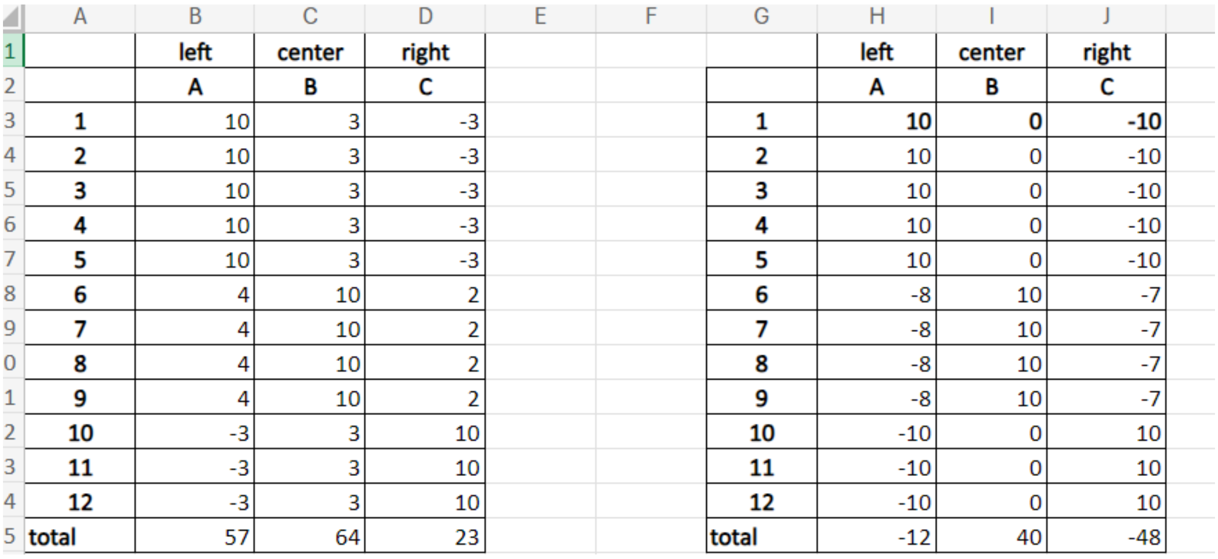
A simple example showing the differences between the two approaches uses a pool of 12 voters, and three candidates: A represents the Left wing, B represents the Center, and C represents the Right wing. We suppose that A has 5 voters, B has 4 voters, and C has 3 voters in the plurality voting system. In the rated system, each voter gives 10 to the preferred candidate, and the following rules apply:

- the voters which prefer A, between B and C always prefer B;
- the voters which prefer B, slightly prefer A;
- the voters which prefer C, between A and B always prefer B.

We imagined two cases:

- the moderate case, where the rates have moderate values,
- the extreme case, where the rates have extreme values.

The results are presented in Fig. 7, showing again the coarse-grained values for the plurality system approach, when compared against the fined-grained results for the rated voting system. In both cases, for the first system, candidate A clearly wins. But, for the second system, in both cases, candidate B wins. When moderate rates are given, there is a very small difference between A and B. But when the voters express a very sharp set of opinions, candidate B wins by far. *In our opinion, this shows how the evaluation of the whole set of candidates could influence the results, not only the first personal choices of the voters.*



	A	B	C	D	E	F	G	H	I	J
1		**left**	**center**	**right**				**left**	**center**	**right**
2		**A**	**B**	**C**				**A**	**B**	**C**
3	**1**	10	3	-3			**1**	**10**	**0**	**-10**
4	**2**	10	3	-3			**2**	10	0	-10
5	**3**	10	3	-3			**3**	10	0	-10
6	**4**	10	3	-3			**4**	10	0	-10
7	**5**	10	3	-3			**5**	10	0	-10
8	**6**	4	10	2			**6**	-8	10	-7
9	**7**	4	10	2			**7**	-8	10	-7
0	**8**	4	10	2			**8**	-8	10	-7
1	**9**	4	10	2			**9**	-8	10	-7
2	**10**	-3	3	10			**10**	-10	0	10
3	**11**	-3	3	10			**11**	-10	0	10
4	**12**	-3	3	10			**12**	-10	0	10
5	**total**	57	64	23			**total**	-12	40	-48

a) moderate rates *b) extreme rates*

Fig. 7 – The results for the rated system, two simulations: a) moderate, b) extreme

3.4 Challenges in implementing e-voting

The difficulties manifested by online voting can be classified in: technical issues, operational issues, privacy concerns, and human-related challenges.

Technical problems could arise from hardware crashes (therefore, the voting process is interrupted), authentication aspects (to verify the voter without breaching the owner's privacy can be a difficult process), or security risks (inherent vulnerabilities when data transfer take place using the internet). Operational problems could manifest when voting platforms are not designed to accommodate people with diverse acting restrictions (for example visual impaired people), or when people do not understand how the voting results are computed (leading to social mistrust). Data protection (identities, votes) is a complex problem, which must be carefully addressed. Traceability of a casted vote may be difficult, as the vote must remain anonymous. Human-related issues refer to uneven access to digital devices, or reliable internet; low level of digital literacy could restrict vote access of large groups of people.

Examples of widely used online voting applications are: ElectionBuddy [16], Polyas [17], Voteer [18], OpaVote [19], Election Runner [20], Simply Voting [21].

4 Conclusions and Future Work

This paper does not intend to rank two voting systems. This is an article that wants to give a spotlight to one less known voting system and bring it to researchers' attention. Depending on the situation, the voting system that is more efficient and gives the best benefits should be used. Each voting system has its own weaknesses and strengths.

Because the application is a prototype that was tested in a controlled environment, the connection between mobile app and server did not require to be secured. Future version of the application will have better encryption modules and the authentication in API will be made using and API key. This facility will ensure that a user cannot vote more than one time. In the case when the login is from a different device, the user will receive a warning email saying that the account was accessed from a different device and she/he will have to confirm the identity.

The mobile application can be further developed by extending its functionality, starting with the authentication system that can be improved by adding different features such as: biometric authentication or fingerprint authentication that would provide more thorough protection to avoid fraud. Inclusiveness is an important aspect: people with disabilities need an accessible solution, to have the chance to participate in elections. By combining technological innovation with transparency and citizenship, we believe that adjusting social systems to current and future social needs will bring fairer and more efficient societies.

Acknowledgement: *This work was supervised by Assoc. Prof. Crișan Gloria Cerasela, from Department of Mathematics & Informatics, Vasile Alecsandri University of Bacău*, *Romania.*

The research was funded by the Ministry of Education, through the National Council for the Financing of Higher Education, Romania, grant number CNFIS-FDI-2025-F-0483: Improving the quality of RDI activities at "Vasile Alecsandri" University of Bacău through national and international cooperation (acronym: Connect-UBc).

References

[1] Douglas E. Schoen - The Power of Voting: Advancing Equity and Championing Democracy, William Morrow, 2007

[2] R. Michael Alvarez, Thad E. Hall, Susan D. Hyde - *Election Fraud: Detecting and Deterring Electoral Manipulation*, Brookings Institution Press, 2009

[3] Emerson, P. - *The original Borda count and partial voting*, Social Choice and Welfare 40, 353–358 (2013).

[4] Cheng, Kung-E, Fadi P. Deek. - *Voting Methods and Information Exchange in Group Support Systems,* Americas Conference on Information Systems (2006).

[5] Encyclopedia Britannica – Plurality and majority systems Encyclopedia Britannica (accessed Oct 2025)

[6] Encyclopedia Britannica - Proportional Representation Voting Systems. https://www.britannica.com/topic/proportional-representation (accessed April 2025)

[7] The University of Chicago, Ranked choice voting https://effectivegov.uchicago.edu/primers/ranked-choice-voting (accessed April 2025)

[8] AMPA - Voting procedures - https://www.oscars.org/oscars/voting (accessed Oct 2025)
[9] Brams, S. J., Fishburn, P. C. - *Approval voting. American* Political Science Review, 72(3), 831–847, (1978)
[10] Smith, W.D. - *The case for score voting*. Constitutional Political Economy 34, 297–309 (2023).
[11] IEEE Annual Election Voting Eligibility - *https://www.ieee.org/about/corporate/election/eligibility* (accessed Oct 2025)
[12] Swagato Chatterjee - *Explaining customer ratings and recommendations by combining qualitative and quantitative user generated contents,* Decision Support Systems,119,14-22, 2019
[13] Balinski, Michel, Laraki, Rida. *Majority Judgment: Measuring, Ranking, and Electing.* MIT Press, 2011
[14]Alvarez, R. Michael; Kiewiet, D. Roderick; Núñez, Lucas - A Taxonomy of Protest Voting. Annual Review of Political Science. 21: 135–154. 2018
[15] Application and data GitHub Repository https://github.com/Giorgiana18/Rated-Voting-System-and-the-future-of-voting-in-Romania (accessed April 2025)
[16] ElectionBuddy platform - https://electionbuddy.com (accessed Oct 2025)
[17] Polyas platform - https://www.polyas.com (accessed Oct 2025)
[18] Voteer platform - https://www.voteer.com/ (accessed Oct 2025)
[19] OpaVote platform - https://opavote.com (accessed Oct 2025)
[20] Election Runner platform - https://electionrunner.com/ (accessed Oct 2025)
[21] Simply Voting platform - https://www.simplyvoting.com accessed Oct 2025)

Giorgiana-Maria MARANGOCI
Vasile Alecsandri University of Bacău
Department of Mathematics & Informatics
Calea Mărășești, nr. 157, Bacău, 600115
ROMANIA
E-mail: giorgianamarangoci@gmail.com

Alex-Andrei RÎPAN
Vasile Alecsandri University of Bacău
Department of Mathematics & Informatics Calea Mărășești, nr. 157, Bacău, 600115 ROMANIA
E-mail: darkdragondd100@gmail.com

Ștefan-Ioan ISTINA
Vasile Alecsandri University of Bacău
Department of Mathematics & Informatics Calea Mărășești, nr. 157, Bacău, 600115 ROMANIA
E-mail: istinastefanioan@gmail.com

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

SmartGarden: An Economical IoT-Enabled System for Indoor Plant Surveillance and Irrigation

Alexandra Onose

Abstract

This paper presents SmartGarden, an economical and adaptable IoT-based technology intended for real-time monitoring and automated irrigation of indoor plants. The system integrates open-source microcontrollers, ambient sensors, and a locally managed Flutter-based mobile application. In contrast to current commercial solutions that frequently depend on cloud infrastructure and proprietary ecosystems, SmartGarden prioritizes user autonomy, offline functionality, and straightforward customization. The system's novelty is in its open, scalable architecture, enabling both novice and experienced users to customize and enhance it based on particular plant care needs. The prototype was evaluated in real-world situations and shown dependable performance, affirming its potential for practical and instructional applications.

1 Introduction

Houseplants form an important organic element in indoor environments, serving its aesthetic purpose while playing an important role in improving well-being and mental comfort. They provide many advantages: energy, stability, and serenity in life. However, while many people may keep one or more of these plants in their homes, caring for them indoors can prove more challenging than expected. Most problems like under-watering, over-watering, or incorrect location come from the lack of timely and plant-specific information. According to a survey from the National Gardening Association conducted in 2020, nearly 50 percent of first-time plant owners kill at least one plant in their first year because they did not understand their needs or how to care for them.

To overcome these issues, we introduced SmartGarden: an open-source, low-cost platform that does not require horticultural knowledge and delivers moisture and nutrients without the need for subscription services. The main goal is to help the novice and the experts in the measuring and caring process of their plants—by using manual irrigation, automated irrigation, and real-time monitoring of environmental data. The system is scalable, from small potted plants to larger indoor gardens, and the adjustable settings supply it with reliability and flexibility.

SmartGarden benefits from its open and modular nature, which distinguishes it from similar solutions. It is agnostic to cloud services or proprietary ecosystems and relies entirely on on-premise hardware and software. The entire system consists of a set of wireless modules that communicate directly with the Wi-Fi through a lightweight mobile application and can monitor each plant. Users of this app can monitor temperature, humidity, and soil moisture levels; set alerts; and configure watering options. Its hardware is commonly available, cheap, and easy to assemble, which makes the system approachable for students, hobbyists, beginners, and individuals seeking to broaden their horticultural experience and their horticultural scope.

At this point, commercial smart plant care systems are widely available but are limited to single plants, or their applications are limited to polluting agricultural technologies like hydroponics. In contrast, the SmartGarden system is a middle-ground approach, a plug-and-play, reusable offering that holds the advantages of existing tools without sacrificing autonomy, cost, and education/learning. It effectively bridges the gap between entry-level consumer devices and complex industrial systems, at the same time enabling users to hone their personal and intellectual skills with horticulture.

This paper details the development and testing of the SmartGarden system, an offline, low-cost solution for indoor plant monitoring and irrigation. Unlike many commercial alternatives, SmartGarden operates entirely without cloud services, prioritizing data privacy and full user control. The platform is built using accessible, open-source hardware and software, and its modular design allows users to easily expand it—for instance, by adding AI-based diagnostics or new types of environmental sensors. These aspects together define the project's core value: a flexible, decentralized, and educational tool that combines affordability with real-world applicability and technical relevance.

The paper is organized as follows: In Section 2, the hardware architecture and circuit design are described. In Section 3, the software components (firmware and mobile application) are introduced; In Section 4, the experimental validation and results are shown. Finally, in Section 5, findings and future development directions are summarized.

2 Related Work

Smart irrigation and plant monitoring systems have been widely explored in both academic and maker communities, leveraging IoT technologies for real-time data acquisition, automation, and remote control.

Wu et al. [1] designed an intelligent flowerpot that integrates soil moisture sensing and automatic watering. While user-friendly, their solution is tied to a cloud-based infrastructure and supports a single-plant setup, limiting its scalability and privacy. Zhang et al. [3] proposed a more complex system for precision agriculture using wireless sensor networks, suitable for large-scale outdoor farming rather than household applications.

Ahmed et al. [2] presented a multi-sensor smart plant monitoring system using Arduino and Wi-Fi modules. Their setup demonstrates strong environmental monitoring capabilities but lacks modular expansion features. In contrast, SmartGarden supports flexible scaling and localized control for each plant.

Piyare [4] explored ubiquitous home control using smartphones and Android-based IoT integration, illustrating the early potential of mobile interfaces in home automation. Our project builds on similar principles but emphasizes offline-first functionality and platform independence.

Choudhari et al. [5] implemented an IoT-based smart gardening system that collects environmental data and controls irrigation. However, their solution lacks detailed modular configuration per plant and focuses more on general automation. Similarly, Zhang et al. [6] introduced a smart irrigation system using LoRa and edge computing for farms. While innovative, it introduces unnecessary complexity for small indoor setups.

Agarwal et al. [7] advanced the field by incorporating large language models into mobile apps for human-plant interaction, demonstrating potential in diagnostics and interactivity. Risheh et al. [8] proposed using transfer learning with neural networks for smart irrigation—a direction relevant for future iterations of SmartGarden, especially in AI-enhanced diagnostics, as discussed in section 2.4.5.

Foundational insights into IoT development are presented by Bahga and Madisetti [9], offering a practical approach to system design that influenced the modularity of SmartGarden. Margolis [10] and Barnett et al. [13] provide in-depth coverage of Arduino programming, contributing to the firmware development logic in our system.

Broader perspectives on IoT are outlined in Giusto et al. [11] and Xu et al. [12], who highlight the importance of decentralization and security in industrial systems. Our project embraces these principles by prioritizing local data ownership and avoiding cloud reliance. Design best practices from McEwen and Cassimally [14] also informed the physical and software integration aspects of SmartGarden, emphasizing maintainability and modularity.

In conclusion, existing solutions often rely on cloud connectivity [1,2,5], complex hardware [3,6], or high computational resources [7,8]. In contrast, SmartGarden bridges the gap between basic maker projects and complex industrial systems, offering a scalable, user-friendly, and fully offline alternative tailored for educational and indoor use.

3 System Design

The SmartGarden system is built around a handful of simple, affordable components that are easy to find and even easier to work with. Everything is designed to do just what is needed: track plant conditions and control watering—without adding unnecessary complexity. The setup is low-power, easy to tweak, and can be expanded at any time. Whether monitoring a single plant or an entire indoor collection, the system makes it easy to connect multiple modules over the same Wi-Fi network and scale up as you go.

3.1 Hardware Components

At the heart of the SmartGarden system is a simple collection of low-cost, easy-to-find electronics, carefully chosen to manage the basic tasks of monitoring and watering indoor plants. The design is meant to be as straightforward and energy-efficient as possible while staying flexible enough to grow with the user—whether they are caring for a single plant or expanding to a full indoor garden connected over Wi-Fi.

The main “brain” of the system is the Arduino Pro Mini, running at 3.3V. This compact microcontroller coordinates all local operations, from reading sensors to deciding when the plant needs water. It connects directly to two key environmental sensors: one for air and one for soil. The first is the SHT21, mounted on a GY-21 breakout board, which tracks temperature and humidity using I²C communication. The second is a resistive soil moisture sensor, which sends analog signals that the Arduino translates into soil moisture levels—essential for deciding when to water.

To get data from the hardware to the user (and vice versa), the system uses the ESP-01 module, powered by the ESP8266 chip. It creates a local Wi-Fi connection that lets the SmartGarden system talk to the mobile app in real time. Whether it is sending sensor readings or receiving a command to turn on the water pump, this tiny Wi-Fi module keeps everything coordinated.

Since some parts of the system need more power than others, especially the pump, the design includes a voltage step-up using an MT3608 DC-DC converter. This allows a regular USB power source or even a battery pack to power the entire system reliably, raising lower voltages to the 5V needed by more demanding components.

The pump itself is a compact unit that runs on 5 to 12 volts, more than enough to keep houseplants hydrated. It is controlled by an IRF520 MOSFET relay module, which safely switches the pump on and off based on commands from the Arduino. This keeps the watering process smooth and completely automated—or manual if the user prefers.

For quick testing or temporary setups, everything can be wired on a standard breadboard. But for something more permanent or polished, the system can be soldered onto a custom PCB. Both options work well, depending on how much flexibility or durability is needed. Diagrams of the physical layout and wiring can be seen in Fig. 1 and Fig. 2.

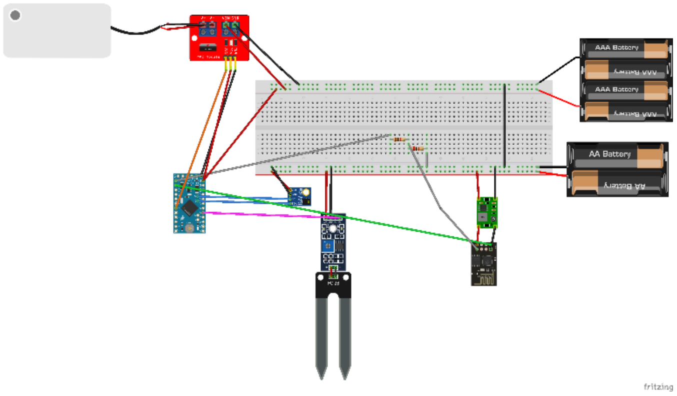


Figure 1. Physical wire configuration of the Smart Garden prototype.

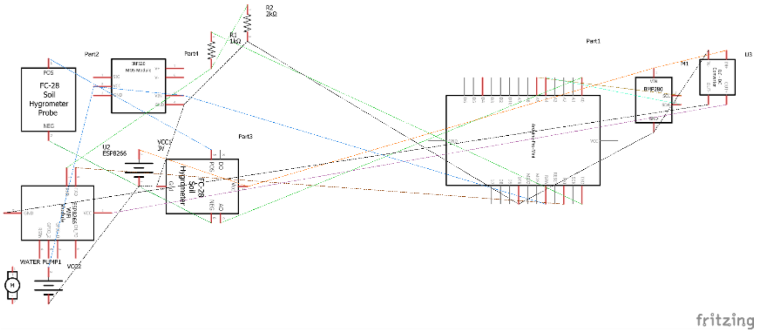


Figure 2. Diagram illustrating the electrical connections of all components.

3.2 Circuit Design

The system's wiring is uncomplicated and follows a straightforward logical architecture, as shown in Figure 3. The Arduino Pro Mini microcontroller serves as the central unit, receiving analog and digital inputs from a soil moisture sensor, a temperature sensor, and a humidity sensor. It processes the collected data and decides whether to activate the water pump through a relay module, depending on the plant's needs. The relay acts as a switch between the pump and the power source, allowing safe control via the Arduino. Water is delivered from a tank directly to the plant through this mechanism. Communication with a mobile application is established through the ESP-01 wireless module, which provides connectivity via Wi-Fi. This allows users to monitor real-time sensor data and control the irrigation process remotely. The overall circuit design ensures energy efficiency, modularity, and compatibility with low-cost components suitable for home automation applications.

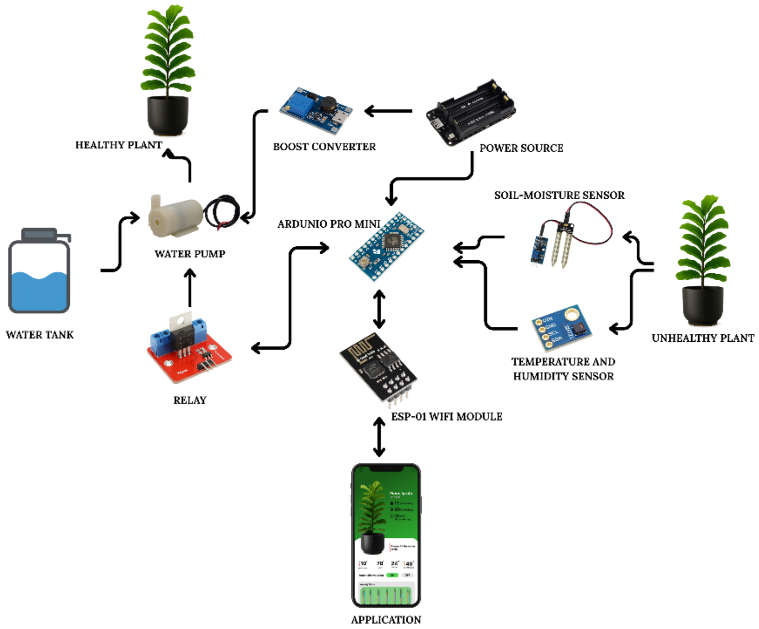


Figure 3. Logical architecture of the Smart Garden system.

3.3 Supported Configuration

The SmartGarden system is designed to manage multiple plant modules at once, with each one equipped with its own set of sensors—measuring soil moisture, air temperature, humidity, and light—along with a resolute watering controller. All these modules connect to the same Wi-Fi network, so everything communicates through a single mobile app using the ESP-01 module as the central hub.

Each plant gets its own unique ID, which the app uses to keep track of them individually. That way, you can monitor and manage several plants at the same time, even if they have different care needs. Whether you are looking after a succulent that barely needs water or a tropical plant that thrives in humidity, each one can run on his own schedule.

The system supports both manual and automatic watering modes, and it collects real-time sensory data from each plant so you can always see how things are going. If you want to expand, you can simply add more modules without changing anything in the core system—if they are on the same network, they will integrate smoothly. There are also plans to support grouped irrigation setups in the future, where several plants can share the same water source while still being watered differently.

Thanks to this flexibility, SmartGarden works just as well for someone caring for a few houseplants as it does for more ambitious indoor gardening projects.

3.4 Enhancements for Accuracy and Reliability

In addition to core system functionality, several design decisions were made to improve the accuracy, reliability, and long-term usability of the SmartGarden platform. This section highlights essential considerations such as sensor calibration and placement, which directly impact system performance.

2.4.1 Supplementary Design Considerations

The sensors in SmartGarden work right out of the box, but if you want your plants to benefit from that data overall, a little calibration goes a long way. Viewing raw sensor values is insufficient without proper calibration to ensure their interpretability and practical relevance.

Calibration begins with the soil moisture sensor, which outputs raw analog values ranging from 0 to 1023. These values represent voltage levels and do not inherently convey meaningful soil hydration levels unless the system is calibrated against known dry and saturated conditions. A simple calibration procedure involves inserting the sensor into completely dry soil and then again into fully soaked soil and taking note of those two values. From there, the system can map that range onto something much more intuitive—like 0% to 100% moisture. Following calibration, a displayed value such as 33% corresponds to a quantifiable level of soil dryness, offering users a more interpretable and actionable measurement rather than an arbitrary number.

The air sensor (SHT21, via the GY-21 module) is already solid. It gives temperature and humidity readings that are generally accurate enough for plant care, with typical variances of about ±0.3°C and ±2% RH. But if you really want to dial it in, especially if you are setting up multiple modules, you can compare it to a reliable thermometer or hygrometer. If the values are a bit off, it is easy to apply a correction in the software so everything stays consistent.

Ultimately, calibration ensures reliability and consistency in system responses, which is essential for automated decision-making. If the system says your soil is dry, it should *be* dry. If you are automating watering, you want it to kick in at the right time—not too early, not too late. Taking a few minutes to calibrate your sensors makes the entire system smarter, more dependable, and better equipped to help your plants thrive, especially if you are working with distinct species, soil types, or changing conditions throughout the year.

2.4.2 Justification for Component Selection

The components selected for the SmartGarden system were determined by principles of modularity, energy efficiency, cost-effectiveness, and interoperability. Each module was chosen for its technical compatibility and its accessibility in educational or maker-oriented contexts, where rapid prototyping and adaptation are crucial.

The wireless communication module is centered around the ESP-01, which utilizes the ESP8266 chipset. This compact and economical module features integrated Wi-Fi capabilities, facilitating effortless integration with local networks for real-time data transmission. Although it has a restricted number of GPIO pins, the ESP-01 is adept at transmitting basic sensor data and receiving control commands within small-scale IoT environments.

Environmental monitoring is chiefly performed using the SHT21 sensor, integrated through the GY-21 module. This sensor offers superior accuracy in temperature and humidity measurements compared to basic alternatives like DHT11 or DHT22. Its I^2C interface guarantees dependable data transmission and facilitates integration in multi-sensor configurations with reduced wiring complexity.

Soil moisture is assessed with an analog resistive sensor, which provides a cost-effective method for detecting hydration levels. When properly calibrated, the sensor delivers adequate precision to guide irrigation decisions and alert thresholds. While more sophisticated capacitive or digital options are available, the selected analog sensor strikes a balance between simplicity, accessibility, and satisfactory performance for general houseplant maintenance.

The IRF520 MOSFET relay module facilitates actuation, allowing for the secure and efficient management of high-power components such as the water pump via logic-level signals from the

Arduino. This relay setup ensures electrical isolation between control and power circuits, thereby minimizing the potential for damage caused by voltage fluctuations.

The power infrastructure is augmented by the MT3608 DC-DC boost converter, which elevates the input voltage from low-capacity sources, such as USB ports or battery packs, to satisfy the demands of downstream peripherals, including pumps and relays. Its adjustable output and extensive input range render it exceptionally versatile for variable power conditions.

The irrigation system utilizes a 5–12V submersible pump, chosen for its compact design, silent operation, and compatibility with USB and battery systems. The pump's performance characteristics are well-suited for standard indoor plant arrangements, delivering dependable and adequate flow for periodic watering tasks.

Collectively, these elements provide a system that is both functionally resilient and easily repeatable by students, researchers, and enthusiasts, facilitating instructional applications and small-scale implementations.

2.4.3 Safety and Reliability

In the SmartGarden system, electronic components are close to water sources, so special attention must be paid to safety and long-term reliability. Moisture and electronics do not go together. Different risk factors can develop, which can be prevented through certain design strategies.

One of the key principles is to keep components physically isolated. Sensitive electronics (e.g., microcontrollers and sensors) should be located above the soil level or contained inside protective housings, limiting their exposure to splashes, humid air, and potential corrodibility. When they are properly sealed, these enclosures can enhance the safety and the mechanical integrity and life of the system.

Another paramount safety element relates to electrical isolation of the actuation circuitry. Because we are using an IRF520 relay module to control the water pump, we need a flyback diode across the pump terminals. This part reduces high-voltage surges, which occur from the inductive load of the pump when switching on and off, protecting the microcontroller and other low-voltage logic devices.

Choosing the right power supply is crucial in ensuring reliability in the system. It is recommended to use a stable 5 V power supply, with a minimal current of 1 ampere, to ensure that both control and actuation subsystems are always on. In the case of larger configurations, or for those with certain pumps/peripherals used in parallel, it could be advisable to implement a dedicated power rail for the irrigation hardware to prevent voltage dropping or instability that would affect the logic controller.

These design philosophies allow the SmartGarden system to function properly under moist environments, ensuring reliability while minimizing hardware failures.

2.4.4 Physical Containment

The first version of the SmartGarden system was built on a breadboard—quick to assemble, great for testing, and perfect for early development. But for long-term use, especially around water, a more durable and polished enclosure is strongly recommended. Having proper physical housing is not about making the system look better; it is about protecting the electronics from moisture, dust, and daily wear.

One practical solution is to use a custom 3D-printed enclosure. These can be designed to neatly fit all the core components, such as the ESP-01 Wi-Fi module, the Arduino Pro Mini, the SHT21 sensor, and the wiring that connects everything. A compact, well-fitted enclosure does not just save space; it also makes the entire system easier to manage and more visually integrated into a home or indoor garden setup.

To prevent problems caused by moisture, it is important to seal off any openings where water could sneak. Simple additions like silicone gaskets or rubber grommets around cable entry points can go a long way in keeping things safe. These kinds of water-resistant features help ensure that the system stays dependable, even in high-humidity environments or when the irrigation pump is running regularly. With appropriate enclosures and minor adjustments, SmartGarden transitions from a functional prototype to a robust, user-ready product.

2.4.5 Prospective Expansion

The SmartGarden system was designed from the beginning as a modular and extensible platform, capable of evolving with user requirements and technological progress. Future iterations of the system are anticipated to include several enhancements, focusing on increasing its autonomy and the accuracy of its environmental diagnostics.

An enhancement entails the incorporation of a water-level sensor in the irrigation reservoir. This feature would notify users of low water levels, thus averting dry-run incidents that may harm the pump or jeopardize plant hydration. Additionally, light monitoring could be implemented via a photoresistor (LDR), enabling the system to evaluate light exposure at the plant's site. Based on these measurements, the mobile application could provide recommendations for relocating the plant or activating supplementary lighting, thereby enhancing photosynthetic efficiency.

To enhance sophisticated plant care methodologies, especially in hydroponic systems or for species with particular nutritional needs, the integration of pH and electrical conductivity (EC) sensors is recommended. These sensors would allow for real-time monitoring of substrate chemistry, enabling prompt interventions such as nutrient modifications or water replenishment.

Moreover, a promising avenue for growth resides in the realm of computer vision and artificial intelligence. The mobile application could integrate a feature encouraging users to periodically capture images of their plants. These photographs could be assessed using lightweight AI models to identify visual signs of plant distress or disease, such as chlorosis, wilting, or fungal manifestations. This functionality would enable the system to provide proactive care recommendations and early alerts prior to the emergence of critical problems.

These enhancements would substantially improve the diagnostic capabilities and autonomy of the SmartGarden system, establishing it as a holistic solution for indoor plant management.

4 Software and Communication

The SmartGarden system combines embedded microcontroller logic with a cross-platform mobile application for real-time plant monitoring and intelligent irrigation management. It utilizes lightweight protocols for communication over a Wi-Fi network, ensuring efficiency, scalability, and adaptability for various plant modules.

4.1 Microcontroller Logic

The SmartGarden system's embedded firmware is allocated between two microcontroller platforms: the Arduino Pro Mini (3.3V) and the ESP-01 Wi-Fi module. These components collaboratively manage sensor data acquisition, local server interaction, and irrigation regulation. The Arduino Pro Mini interfaces with environmental sensors, notably the soil moisture probe (through analog input) and the SHT21 module (utilizing the I^2C protocol). Sensor readings are obtained at specified intervals and processed locally to maintain system responsiveness and energy efficiency.

The Arduino transmits data via UART to the ESP-01 module, serving as a communication conduit between the sensor subsystem and the mobile application. The ESP-01 operates a lightweight HTTP server, functioning within a local area network independent of cloud infrastructure. This server responds to incoming requests by providing real-time environmental data in JSON format and executing irrigation commands received through the mobile interface.

The ESP-01 firmware incorporates automatic reconnection protocols to identify and rectify Wi-Fi disconnection incidents, thereby preserving system reliability. Furthermore, timing mechanisms are employed to stagger sensor readings and network transmissions, effectively reducing electromagnetic interference and congestion within the local network. Additionally, error-handling protocols are integrated to manage failed transmissions or incomplete data exchanges, ensuring consistent functionality during extended deployment periods.

The communication protocol between the Arduino and ESP-01 is succinct and efficient, emphasizing simplicity and minimal latency. Commands are conveyed as plain text messages, while sensor data is structured in JSON objects. This design approach preserves compatibility with lightweight applications and guarantees system functionality in limited networking conditions.

Segment of Arduino Pro Mini code:

```
#include <Wire.h>
#include <SHT2x.h>
SHT21 sht;
#define SOIL_MOISTURE_PIN A0
#define PUMP_PIN 8
bool pumpState = false; // Controlled by ESP
void setup() {
  Serial.begin(9600);     // Communication with ESP-01
  sht.begin();            // Start SHT21
  pinMode(PUMP_PIN, OUTPUT);
  digitalWrite(PUMP_PIN, LOW);
}
void loop() {
  //  Read sensors
  int soilValue = analogRead(SOIL_MOISTURE_PIN);
  float temperature = sht.getTemperature();
  float humidity = sht.getHumidity();
  // Send sensor data to ESP-01 in JSON format
  Serial.print("{\"temperature\":");
  Serial.print(temperature);
  Serial.print(",\"humidity\":");
  Serial.print(humidity);
  Serial.print(",\"soil\":");
  Serial.print(soilValue);
  Serial.println("}");
  // Check if ESP-01 sent a command
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    if (cmd == "WATER ON") {
      pumpState = true;
    } else if (cmd == "WATER OFF") {
      pumpState = false;
  }
  }
  // Set pump based on received command
  digitalWrite(PUMP_PIN, pumpState ? HIGH : LOW);
  delay(5000); // Delay between readings (adjustable)
}
```

Segment of ESP-01 code:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
const char* ssid = "Alexandra";
const char* password = "26012005";
ESP8266WebServer server(80);
String lastSensorData = "{}";
void setup() {
Serial.begin(9600); // Communicate with Arduino
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("\nWiFi connected!");
Serial.print("ESP IP: ");
Serial.println(WiFi.localIP());
// Endpoint: GET /data
server.on("/data", HTTP_GET, []() {
server.send(200, "application/json", lastSensorData);
});
// Endpoint: POST /water
server.on("/water", HTTP_POST, []() {
if (!server.hasArg("plain")) {
server.send(400, "text/plain", "Missing body");
return;
}
String command = server.arg("plain");
command.trim();
if (command == "WATER ON" || command == "WATER OFF") {
Serial.println(command); // Send to Arduino
server.send(200, "text/plain", "Command sent: " + command);
} else {
server.send(400, "text/plain", "Invalid command"); }
});
server.begin();
Serial.println("HTTP server started"); }
void loop() {
server.handleClient();
// Read sensor data from Arduino
if (Serial.available()) {
lastSensorData = Serial.readStringUntil('\n');
lastSensorData.trim();
Serial.println("[ESP] Received from Arduino: " + lastSensorData); } }
```

4.2 Mobile Application

The SmartGarden mobile application was created utilizing the Flutter framework to guarantee cross-platform compatibility and deliver an accessible interface for monitoring and managing plant modules.

The application interacts with each hardware node via a local Wi-Fi network and offers intuitive visual representations of environmental data along with irrigation controls.

Upon initiating the application, users encounter a dashboard that consolidates data from all operational plant modules. Each module is depicted by a tile exhibiting real-time sensor metrics, including soil moisture percentage, ambient temperature in degrees Celsius, and relative humidity in percentage. Color-coded indicators and succinct status labels (e.g., "Needs Watering," "Healthy") facilitate a quick evaluation of plant conditions.

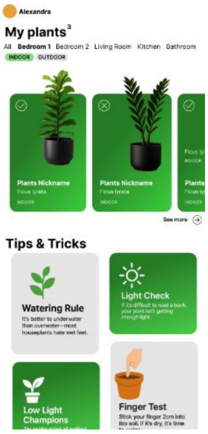
By selecting a designated plant module, users can obtain a comprehensive view featuring individual sensor data and control functionalities. The application accommodates both manual and automatic irrigation modes. In manual mode, users can initiate immediate watering events or regulate the irrigation duration via on-screen commands. In automatic mode, users establish soil moisture thresholds and maximum watering durations, allowing the system to function autonomously based on sensor feedback.

Notifications are generated in real time to inform users when soil moisture levels drop below critical thresholds or when environmental parameters, such as temperature or humidity, deviate from acceptable ranges. Other notification triggers encompass communication delays or data update failures, aiding in the identification of malfunctioning modules or network problems.

Each plant module is distinctly recognized within the application, enabling users to manage multiple plants autonomously and customize configurations according to the requirements of various species. This modularity guarantees that the application is scalable and adaptable to diverse use cases, ranging from individual plant monitoring to multi-node indoor gardening systems.

Flutter's cross-platform capabilities make development much more efficient while ensuring users get the same great experience whether they're on Android or iPhone. This approach also makes it easier to add exciting features down the road—like using your phone's camera to diagnose plant problems or syncing your garden data to the cloud, which would really expand what the SmartGarden system can do.

At the current stage of development, the application's interface is represented through early-stage mockups (Figures 4 and 5), which illustrate the intended layout and user interactions of the final implementation.

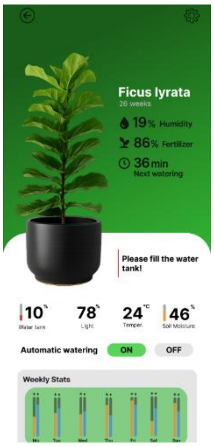


It's better to underwater than overwater—most houseplants hate wet feet.

If it's difficult to read a book, your plant isn't getting enough light.

Stick your finger 2cm into the soil. If it's dry, it's time

Figure 4. Early-stage UI mockup of the SmartGarden dashboard interface (designed in Figma).



Please fill the water tank!

Figure 5. Early-stage UI mockup of the individual Plant Details interface(designed in Figma).

4.3 Data Communication

The communication architecture of the SmartGarden system consists of a light and decentralized model, based on a local HTTP server hosted on the ESP-01 module. This approach eliminates the need to rely on external cloud/hosting infrastructure, message broker, or third-party services, which enhances data privacy and gives the ability for the user to control the entire functionality of the system

ESP-01 module works on RESTful server, offering a small number of endpoints for data transmission and commands execution in real-time. The mobile application can retrieve sensor data from the GET /data endpoint, which returns the latest environmental values in a JSON-encoded format. Mobile interface constantly polls this endpoint so that it reflects the plant conditions accurately.

In addition to passive monitoring, users can access the POST /water endpoint to trigger control actions. This interface fetches system commands such as "WATER ON" or "WATER OFF," allowing the user to turn the water pump on or off from the application interface. We provide a highly efficient communication protocol, which maximizes raw bandwidth usage while minimizing negative content latency.

Our SmartGarden system is designed to run over the local Wi-Fi in your house, so you do not need to connect it to the internet. This provides the ability to reduce operational costs while enhancing the platform's robustness and safety, making it suitable for environments where cloud accessibility is not possible or ideal.

5 Experimental Configuration and Outcomes

For the practical assessment of the SmartGarden system, a complete prototype was built, and it was evaluated for several days with a single indoor plant: a Monstera Deliciosa. The selected species was a well-known houseplant with a moderate sensitivity to watering conditions to assess overall accuracy and responsiveness of the system.

The main objective of the assessment was to ensure the operability of the sensor modules, the fault-free operation of the irrigation control logic, and the functionality of real-time data exchange between the hardware and the mobile application in a real-life condition inside a room.

5.1 Setup Description

The experimental assessment utilized a solitary indoor plant specimen, Monstera Deliciosa, contained in a conventional flowerpot. This species was chosen for its prevalence among indoor horticulturists and its moderate sensitivity to watering frequency, rendering it an appropriate subject for evaluating the responsiveness and dependability of automated irrigation systems.

The SmartGarden system was comprehensively implemented in this configuration, with all components interconnected via a solderless breadboard to enable swift prototyping. Power was provided through a 5V USB adapter, regulated by an MT3608 step-up converter to satisfy the operational voltage demands of modules. The hardware interfaced with the mobile application through a local Wi-Fi network, facilitating real-time data collection and remote irrigation management.

Sensor instrumentation comprised a soil moisture sensor positioned adjacent to the plant's root zone to accurately measure hydration levels with significant spatial relevance. The SHT21 sensor module (GY-21) was affixed above the pot to continuously monitor ambient temperature and humidity, reflecting the microclimatic conditions directly influencing the plant. The ESP-01 module served as the wireless communication interface, establishing and sustaining a local network connection with the application, thereby ensuring uninterrupted data transmission and command execution.

Irrigation was conducted utilizing a compact submersible water pump, operable either manually through a mobile interface or automatically via sensor feedback. The pump was regulated through an IRF520 MOSFET relay module, which facilitated the requisite switching mechanism to connect the low-voltage control circuit with the pump's power supply. The complete configuration is presented in Figure 6, highlighting the arrangement of the electronic components and their spatial positioning.

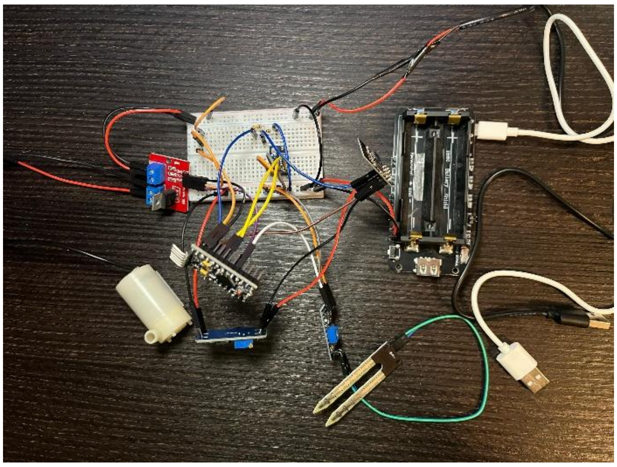


Figure 6. SmartGarden experimental setup with core components assembled on a breadboard for demonstration purposes.

5.2 Observations

During the experimental testing phase, the SmartGarden system demonstrated consistent and reliable performance under standard indoor environmental conditions. A significant observation involved the dynamics of soil moisture. After each irrigation event, moisture levels decreased at a natural rate, which varied according to plant species and soil composition. When sensor readings dropped below a user-defined threshold—established at 30% for the test plant—the system either activated automatic watering or dispatched an alert to the mobile application, contingent upon the chosen operational mode.

The SHT21 module exhibited consistent and precise readings throughout the testing period regarding environmental sensing. Variations in ambient temperature and relative humidity logically aligned with diurnal cycles and room ventilation occurrences, confirming the sensor's responsiveness and stability. These measurements yielded a dependable dataset for monitoring the plant's growing environment and guided potential future modifications to care routines.

The system's network resilience was equally significant. The ESP-01 module sustained uninterrupted Wi-Fi connectivity, with no critical failures or communication disruptions noted. The embedded firmware's automatic reconnection protocols functioned as intended during simulated disconnection scenarios, guaranteeing data continuity and maintaining user control through the mobile application.

The SmartGarden prototype demonstrated its effectiveness as a low-maintenance and responsive instrument for indoor plant care. Its capacity to combine sensor-driven automation with manual override options facilitated a flexible plant management approach. Users could oversee and irrigate multiple plants via a centralized, intuitive interface without dependence on external cloud services, validating the proposed design's feasibility for both hobbyist and research-oriented applications.

6 Conclusions

We presented SmartGarden, an open-source, modular, and affordable IoT platform designed for the monitoring and irrigation of indoor plants. By integrating low-cost microcontrollers (Arduino Pro Mini and ESP-01), high-precision environmental sensors (SHT21), and a cross-platform mobile application built using Flutter, this system provides a robust and cost-effective alternative to current smart gardening technologies.

Unlike many commercial systems that rely on proprietary cloud services, SmartGarden operates entirely offline and is fully decentralized. This promotes data privacy, long-term maintainability, and complete user control. Compared to similar solutions, such as the IoT-enabled flowerpot proposed by Wu et al. [1] or the precision irrigation controller evaluated by Zhang et al. [3], SmartGarden prioritizes simplicity, full user ownership, and hardware flexibility. Its modular architecture allows users to scale the platform as needed, accommodating a wide variety of plants and enabling deployment in both personal and educational contexts.

The project was successfully implemented and tested in real-world conditions. The system proved to be reliable, flexible, and easy to replicate, supporting both manual and automated irrigation for various plant species. Its offline functionality makes it particularly suitable for environments where internet connectivity is limited or unavailable, such as rural homes or classrooms.

SmartGarden also serves as an accessible platform for students and enthusiasts to explore embedded systems, sensor networks, wireless communication, and environmental monitoring. Its open-source nature encourages experimentation, customization, and further development.

Future enhancements may include the integration of additional sensors (e.g., pH, EC, light intensity via LDR), computer vision modules for plant health diagnostics, and optional cloud connectivity for extended functionality. These additions would significantly increase the autonomy, diagnostic precision, and usability of the system, transforming SmartGarden into a comprehensive smart horticulture platform.

The novelty of SmartGarden lies not in the individual components used, but in the practical integration of offline-first functionality, modular hardware design, and open educational accessibility. While similar projects exist, few combine these attributes into a cohesive platform that can serve as a low-barrier entry point for IoT experimentation, especially in rural, academic, or budget-constrained environments. As such, this system contributes to bridging the gap between DIY prototyping and scalable smart infrastructure in the context of indoor horticulture.

Acknowledgement: This work was supervised by Professor *Anca Vasilescu*, from *Transilvania University of Brasov.*

References

[1] Y. Wu, C. Zhai, Y. Tian, *Design of Intelligent Flowerpot Based on Internet of Things*, Asian Journal of Applied Science and Technology, vol. 4, no. 3, pp. 1–6, 2020.

[2] M. Ahmed, A. Hannan, A. Basaruddin, M. F. Zolkipli, *Development of Smart Plant Monitoring System using IoT*, Indonesian Journal of Electrical Engineering and Computer Science, vol. 19, no. 3, pp. 1490–1498, 2020.

[3] T. Zhang, W. Zhang, J. Wang, *Wireless Sensor Network-Based Intelligent Irrigation Control System for Precision Agriculture*, Agricultural Engineering International: CIGR Journal, vol. 21, no. 1, pp. 97–107, 2019.

[4] R. Piyare, *Internet of Things: Ubiquitous Home Control and Monitoring System using Android based Smart Phone*, International Journal of Internet of Things, vol. 2, no. 1, pp. 5–11, 2013.

[5] G. R. Choudhari, P. A. Dagale, I. S. Dashetwar, R. R. Desai, A. A. Marathe, *IoT-based Smart Gardening System*, Journal of Physics: Conference Series, vol. 2601, no. 1, p. 012006, 2023.

[6] Y. Zhang, X. Wang, L. Jin, J. Ni, Y. Zhu, W. Cao, X. Jiang, *Research and Development of an IoT Smart Irrigation System for Farmland Based on LoRa and Edge Computing*, Agronomy, vol. 15, no. 2, p. 366, 2025.

[7] K. Agarwal, S. Ananthanarayanan, S. Srinivasan, A. S, *Enhancing IoT-based Plant Health Monitoring through Advanced Human-Plant Interaction using Large Language Models and Mobile Applications*, *arXiv preprint*, arXiv:2409.15910, 2024.

[8] A. Risheh, A. Jalili, E. Nazerfard, *Smart Irrigation IoT Solution using Transfer Learning for Neural Networks*, *arXiv preprint*, arXiv:2009.12747, 2020.

[9] A. Bahga, V. Madisetti, *Internet of Things: A Hands-On Approach*, Universities Press, 2014.

[10] M. Margolis, *Arduino Cookbook*, 2nd ed., O'Reilly Media, 2011.

[11] D. Giusto, A. Iera, G. Morabito, L. Atzori (Eds.), *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications*, Springer, 2010.

[12] L. D. Xu, W. He, S. Li, *Internet of Things in Industries: A Survey*, IEEE Transactions on Industrial Informatics, vol. 10, no. 4, pp. 2233–2243, 2014.

[13] R. H. Barnett, L. O. Mazurek, S. C. Cox, *Embedded C Programming and the Atmel AVR*, Delmar Cengage Learning, 2008.

[14] A. McEwen, H. Cassimally, *Designing the Internet of Things*, Wiley, 2013.

Onose Alexandra
Transilvania University of Brasov
Faculty of Mathematics and Informatics
Bd. Iuliu Maniu nr. 50
500091 Brașov Romania
E-mail: alexandra.onose@student.unitbv.ro

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

TransitAI: An AI-Powered Conversational Assistant for Public Transportation Information Access

Eduard-Alexandru Oprea, Elena-Luiza Buzatu, Ioana-Valeria Turcin

Abstract

Public transportation systems generate vast amounts of structured data through GTFS (General Transit Feed Specification) feeds, yet accessing this information through conventional interfaces often presents significant usability challenges. This paper introduces TransitAI, an AI-driven conversational assistant that bridges the gap between complex transit data and intuitive user interaction. By integrating large language models with domain-specific transit knowledge, the system enables users to query transportation information using natural language. The architecture combines a Flask-based backend with specialized services, including a natural language understanding component (Ollama), real-time data processing (GTFS Realtime), journey planning (OpenTripPlanner), and data retrieval (Elasticsearch). Through a modular design, the system accommodates various query types, from stop information and route details to complex journey planning. Technical evaluations demonstrate the system's ability to handle complex transit queries with high accuracy while maintaining responsive performance. This research demonstrates how AI can transform public service information access, providing insights for future smart city applications.

1 Introduction

Public transportation is a cornerstone of urban mobility, yet accessing accurate and timely information about transit services remains a persistent challenge for many users. Traditional methods of obtaining transit information — such as printed schedules, mobile apps with complex interfaces, or website navigation — often create barriers, particularly for users unfamiliar with the transit system or those with accessibility needs.

The widespread adoption of General Transit Feed Specification (GTFS) and GTFS Realtime has standardized how transit data is structured and shared, enabling innovative approaches to information delivery. Simultaneously, advances in natural language processing and conversational AI have created new possibilities for human-computer interaction.

In this paper, we present TransitAI, a conversational assistant for public transportation that combines large language models with transit data processing to provide users with an accessible interface for transit information. The application allows the following:

- Ask questions about transit services in their natural language;
- Receive personalized journey planning assistance;
- Access real-time information about arrivals and departures;
- Obtain detailed information about specific routes and stops.

This integration of AI with transit data represents a significant advance in making public transportation more accessible to a broader audience, potentially increasing ridership and improving the overall transit experience.

2 Related work

2.1 Transit information systems

Transit information systems have evolved significantly over the past decade. Web-based journey planners like Google Maps Transit offer structured interfaces for planning trips [1]. Mobile applications such as Moovit and Citymapper provide real-time tracking and personalized alerts [2]. However, these systems generally require users to adapt to their interface paradigms rather than allowing natural language interaction.

Research has shown that complex transit apps can present significant usability barriers, particularly for elderly users and those with cognitive impairments [3]. Conversely, studies have demonstrated that natural language interfaces can reduce cognitive load and increase success rates for transit information tasks [4].

2.2 Conversational AI in public services

The application of conversational AI to public services has gained momentum in recent years. Previous research has examined the use of chatbots for government services in Europe, finding significant potential for reducing administrative overhead while improving citizen satisfaction [5]. Similarly, design patterns for chatbots in public sector applications emphasize the importance of context awareness and clear communication of system capabilities [6].

In the transit domain specifically, studies have demonstrated that real-time information delivery increases ridership and improves user satisfaction with transit services [7], suggesting that more intuitive interfaces for accessing this information could further enhance these benefits.

2.3 NLP for Transit Applications

Recent work in natural language processing for transit applications has focused on entity extraction and intent recognition. Techniques for identifying transit-specific entities such as stop names and route numbers from user queries have been developed [8]. Building on this, domain-specific intent classification systems for public transport queries have shown improved performance over general-purpose NLP systems [9].

Our work extends these approaches by implementing a comprehensive architecture that not only understands transit-specific queries but also integrates with multiple data sources to provide comprehensive, real-time responses in a conversational manner.

3 System architecture and design

TransitAI employs a modular architecture that integrates several specialized components to process natural language queries, retrieve relevant transit data, and generate human-like responses.

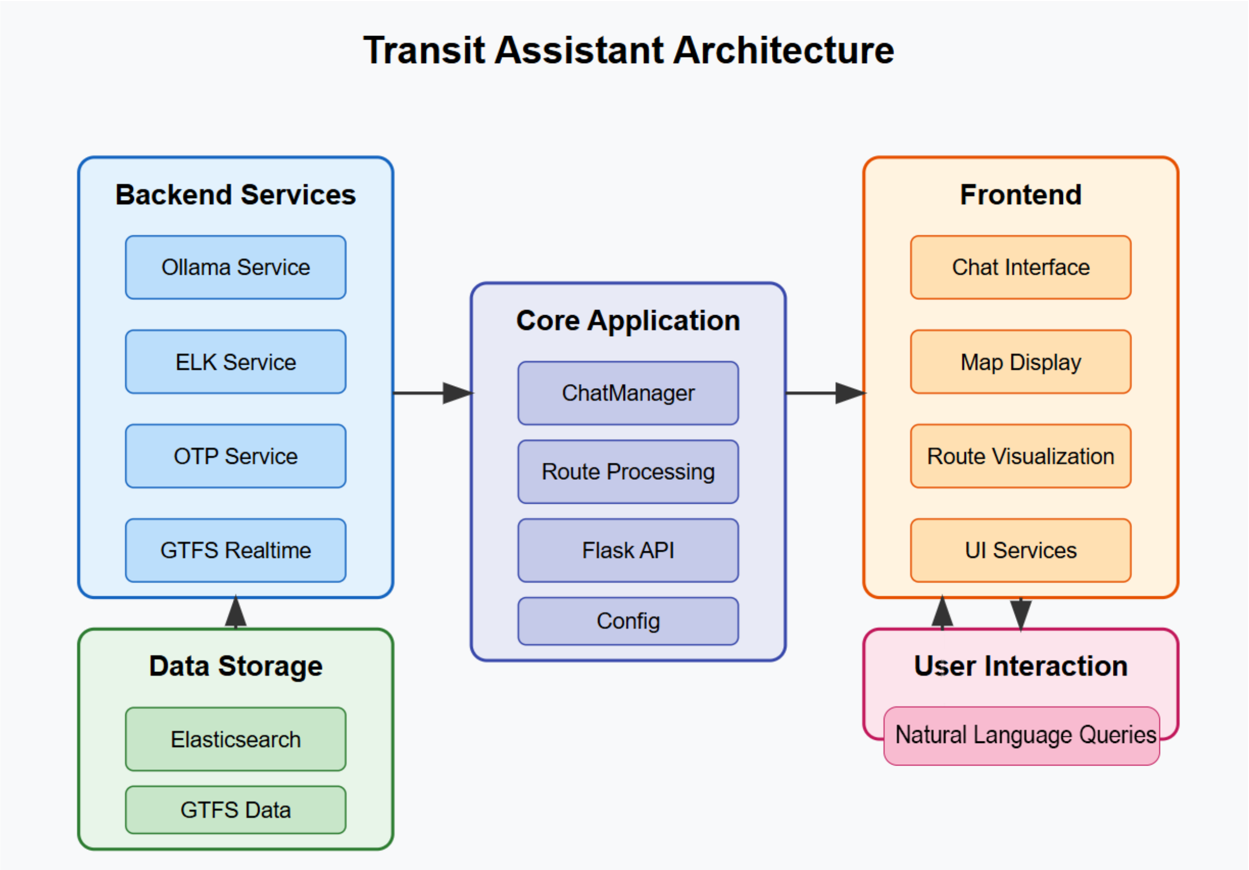


Fig. 1 Diagram showing the high-level system architecture with components and data flow

The system consists of the following main components.

3.1 Core services

3.1.1 Dynamic QR implementation

The ElkService handles data retrieval from Elasticsearch, which indexes the GTFS static data. This component is responsible for:

- Finding stops by name or proximity;
- Retrieving route information and schedules;
- Obtaining upcoming departures for specific stops;
- Storing conversation history for context-aware responses.

For example, the implementation of the stop search function uses Elasticsearch’s fuzzy matching capabilities to handle typos and variations in stop names as previously shown.

3.1.2 OllamaService

The OllamaService provides natural language understanding capabilities using a large language model. It performs several critical functions:

- Intent detection: Determining whether the query relates to stop information, route information, journey planning, or general transit questions;

- Entity extraction: Identifying transit-specific entities such as stop names, route numbers, and locations;
- Natural language generation: Creating human-like responses based on retrieved data.

The service interacts with the Ollama API as follows:

```
messages = [
    {
        "role": "system",
        "content": "You are a helpful assistant that extracts transit stop information from user messages."
    },
    {
        "role": "user",
        "content": f"""
        Extract the 'from' and 'to' stop names from the following message.
        These are likely bus, tram, or subway stops.
        Message: {message}

        Return ONLY a JSON object in this format:
        {{
          "from": "exact stop name",
          "to": "exact stop name"
        }}

        If you cannot extract both stop names, return an empty JSON object.
        """
    }
]
```

The service uses specialized prompts engineered to optimize the language model's performance for transit-specific queries, enhancing accuracy for domain terminology.

3.1.3 ElkService

The ElkService handles data retrieval from Elasticsearch, which indexes the GTFS static data. This component is responsible for:

- Finding stops by name or proximity;
- Retrieving route information and schedules;
- Obtaining upcoming departures for specific stops;
- Storing conversation history for context-aware responses.

For example, the implementation of the stop search function uses Elasticsearch's fuzzy matching capabilities to handle typos and variations in stop names:

```
es_query = {
    "query": {
        "multi_match": {
            "query": query_text,
            "fields": ["doc.stop_name^3", "doc.stop_desc"],
            "type": "best_fields",
            "fuzziness": "AUTO"
        }
    },
    "size": limit
}
```

3.1.4 OTPService

The OTPService interfaces with OpenTripPlanner to handle journey planning requests. When users request directions between locations, this service.

The service translates natural language journey requests into structured GraphiQL queries and processes the responses to extract the most relevant journey options for users.

3.2 Support services

3.2.1 GTFSRealtime service

This service processes GTFS Realtime feeds to provide up-to-date information about vehicle positions, service alerts, and trip updates. The implementation leverages Google's GTFS Realtime protobuf specification:

```
feed = gtfs_realtime_pb2.FeedMessage()
feed.ParseFromString(response.content)
return feed
```

This service uses a caching mechanism to minimize API calls while ensuring data freshness.

3.2.2 LocationParser

The LocationParser component specializes in extracting and resolving location references from user queries.

This component enhances the system's ability to handle ambiguous location references in natural language by combining multiple extraction strategies.

3.3 Web Application

The system is exposed through a Flask application that provides API endpoints for various transit-related functionalities. This backend server acts as the central hub that connects all parts of the system together.

The chat endpoint serves as the primary interface for natural language interaction. When users send messages, this endpoint processes them by first determining the intent of the query. It analyzes whether the message is asking about specific stops, routes, or journey planning between locations. Based on this analysis, it then fetches relevant transit data from the Elasticsearch service and generates appropriate responses. For general transit questions, it uses the Ollama language model service to create helpful, natural language responses with context from the transit database. The endpoint maintains conversation history to provide continuity in interactions.

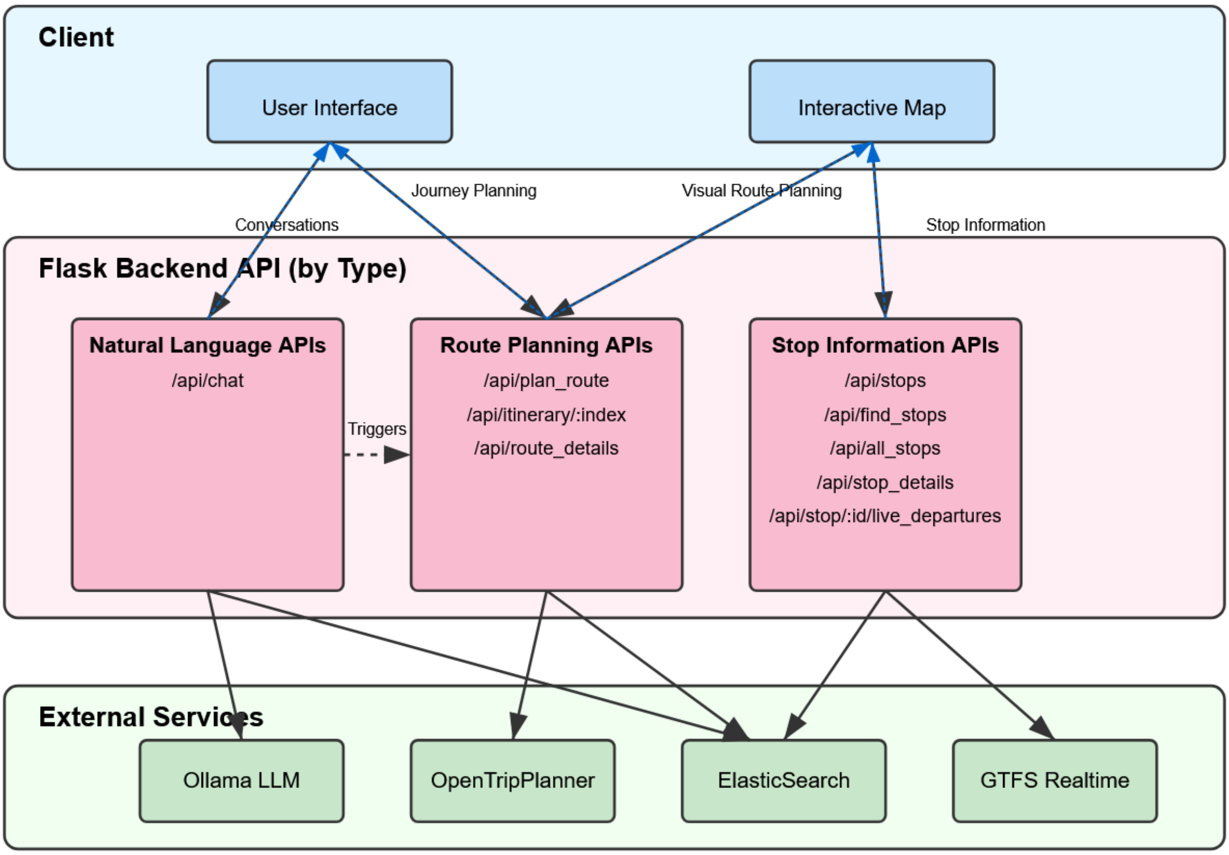


Fig. 2 General architecture of TransitAI

The route planning endpoint handles the complex task of finding optimal journeys between locations. When receiving a request with starting and ending coordinates, it connects to the OpenTripPlanner service to calculate multiple possible routes. It returns detailed information about each route option, including duration, distance, transport modes, and step-by-step instructions. The endpoint processes these routes into a user-friendly format with summaries and visualization data for the map interface. It also stores journey planning requests in the conversation history to maintain context for future interactions.

3.4 Frontend Services

The frontend components facilitate user interaction with the transit data through a web interface. These are implemented using JavaScript modules that interact with the Flask API.

3.4.1 MapService

Manages all interactions with the interactive map interface. It initializes the map with appropriate base layers and configures the visual appearance. This service handles the creation and positioning of various map elements like stop markers, route lines, and location indicators. It provides functions to create different types of markers based on their purpose, such as standard stop markers, highlighted stops, starting points, destinations, and transfer points.

The service manages map layers to organize different types of visual elements, separating stops, routes, and selected locations into different groups that can be toggled independently. It also handles map events like zooming and panning, refreshing the displayed stops as the user navigates the map view.

Additionally, it provides functions to load and display detailed information about stops, including upcoming departures and service alerts, when users interact with stop markers on the map.

3.4.2 StopService

Focuses on managing transit stop data and related functionality. It maintains a cache of all transit stops to reduce server requests and improve performance. The service provides functions to load stop data from the backend, find stops by ID or name, and filter stops based on various criteria. It handles the display of stops on the map, showing them only at appropriate zoom levels to prevent visual clutter.

This service also manages the selection of stops for journey planning, with specialized functions for setting starting points and destinations. It can highlight specific stops on the map when users search for them or select them from the interface. The service connects to the backend to retrieve detailed information about stops, including the routes serving them, upcoming departures, and real-time status updates. It also provides search functionality to help users quickly find specific stops by name or location.

3.4.3 RouteService

Handles all aspects of transit route planning and visualization. It provides functions to send route planning requests to the backend with starting and ending locations. The service processes the resulting route data and displays route options to the user in an understandable format, showing duration, distance, transport modes used, and other relevant details.

When users select a specific route option, the service displays detailed step-by-step instructions and visualizes the route on the map with color-coded lines representing different transportation modes. It handles the display of start points, endpoints, transfer points, and intermediate stops along the route. The service also provides functions to clear route planning data and reset the interface when needed.

Additionally, this service can highlight specific transit routes on the map when users inquire about them, showing all stops along the route and providing information about schedules and service frequency.

4 System Architecture and Design

4.1 Intent Detection Pipeline

A critical aspect of TransitAI is its ability to accurately identify the user’s intent from natural language queries. The system implements a specialized intent detection pipeline that classifies queries into four main categories:

1. Stop Information Queries: Questions about specific stops, such as “What buses serve Central Station?” or “When is the next bus at Market Street?”;
2. Route Information Queries: Questions about specific routes, such as “Tell me about Route 10” or “Where does the 42 bus go?”;
3. Journey Planning Queries: Requests for directions, such as “How do I get from Downtown to the Airport?” or “What’s the fastest way to reach the University?”;
4. General Transit Queries: Other transit-related questions that don’t fit the above categories, such as “How much is a bus ticket?” or “Are there any service disruptions today?”.

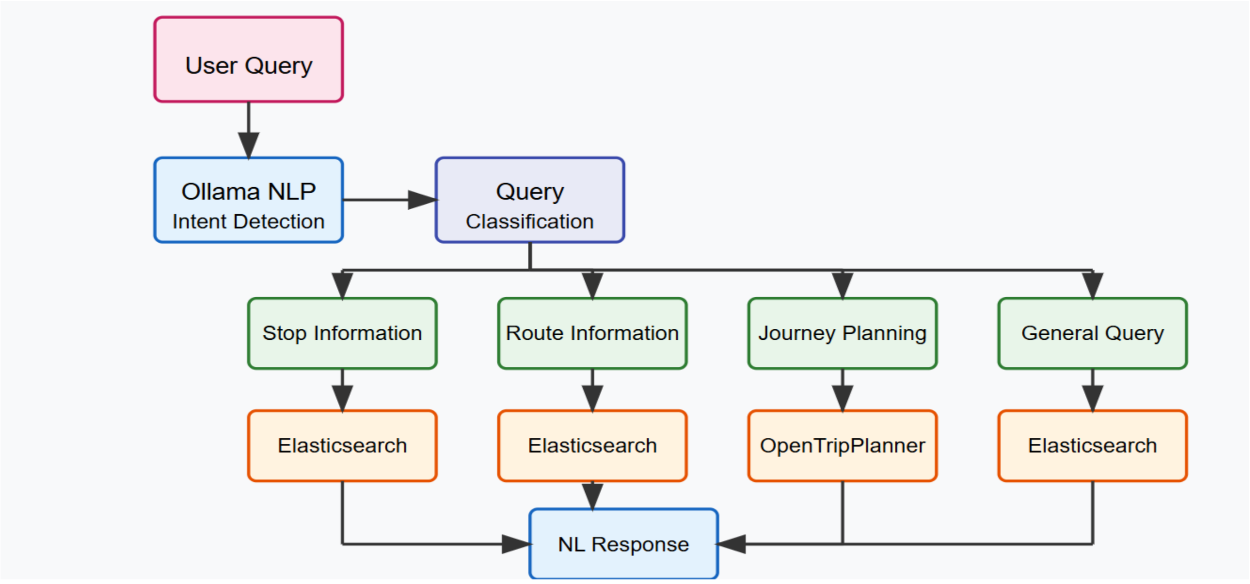


Fig. 3 Query Processing Workflow

The intent detection process uses a two-stage approach:

1. The OllamaService performs initial intent classification by analyzing linguistic patterns, keywords, and semantic context.
2. The ChatManager applies domain-specific rules to refine the classification, considering entities extracted from the query.

4.2 Entity Extraction

Entity extraction is performed on all queries to identify key information elements:

- Stop names: Identified through a combination of pattern matching and fuzzy search against the transit database;
- Route identifiers: Extracted using regular expressions and validated against available routes;
- Locations: Processed by the LocationParser to resolve to geographic coordinates;
- Temporal references: Identified to determine whether the query relates to current, future, or specific time information.

The entity extraction process is customized for transit terminology, improving accuracy for domain-specific terms that might be misinterpreted by general-purpose NLP systems.

4.3 Query Handling Workflows

Based on the identified intent and extracted entities, the system selects the appropriate processing workflow:

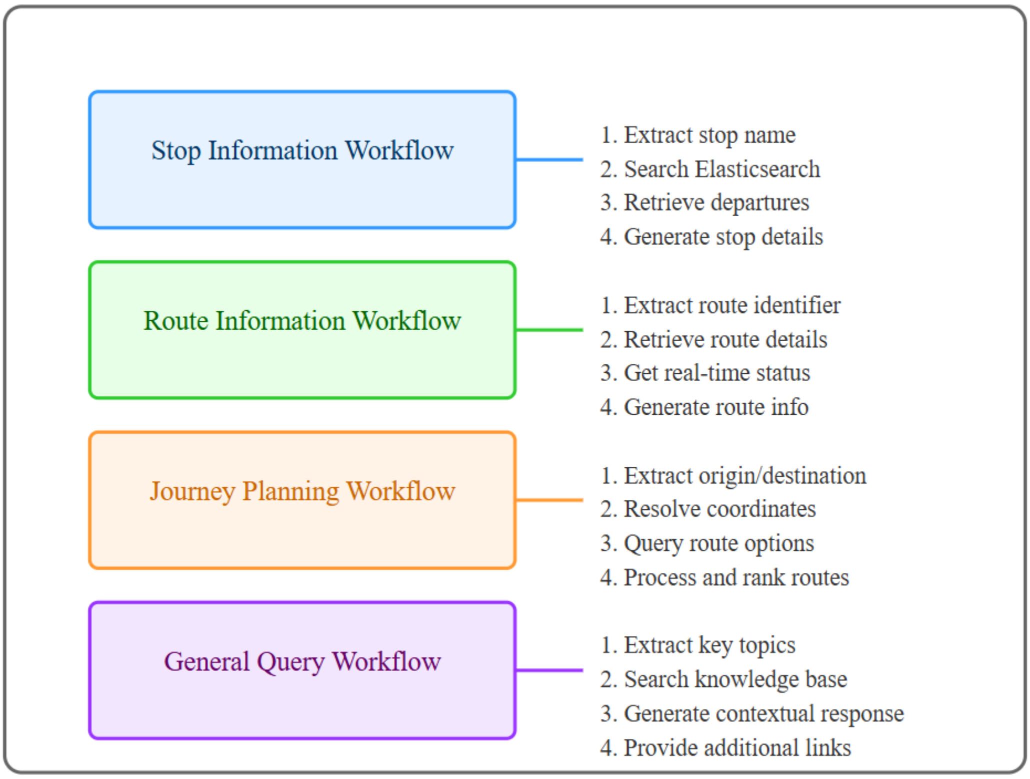


Fig. 4 Transit Query Handling Workflows

1. Route Information: Extracts the route identifier, retrieves route details (including stops and schedules) from Elasticsearch, obtains real-time vehicle status, generates a response with route description and status, and includes interactive visualization elements.
2. Journey Planning: Extracts origin and destination, resolves locations to coordinates using a LocationParser, queries OpenTripPlanner for route options, processes and ranks alternatives, generates a response with journey details and timing, and includes interactive visualization elements. Elasticsearch may be used to store location data.
3. General Query: Extracts key topics and entities, searches a transit knowledge base (within Elasticsearch and the model's own information base – but we plan on allowing internet access) for relevant information, generates a contextual response, and provides links to further information.
4. Stop Information: Extracts stop names, searches Elasticsearch for stop details, retrieves departure times, and generates stop information.
5. All workflows leverage Elasticsearch for data storage and retrieval, ensuring efficient access to transit information.

5 Data Integration and Processing

5.1 GTFS Static Data Processing

TransitAI relies on GTFS static data as its foundation for transit information. This data is processed and indexed into Elasticsearch to enable efficient querying. The processing pipeline includes:

1. Data Validation: Checking for consistency and completeness of the GTFS feed
2. Geospatial Indexing: Creating spatial indexes for stops and routes to enable proximity-based queries;
3. Text Normalization: Standardizing stop names and descriptions for improved search

4. Relationship Mapping: Establishing connections between routes, stops, and schedules

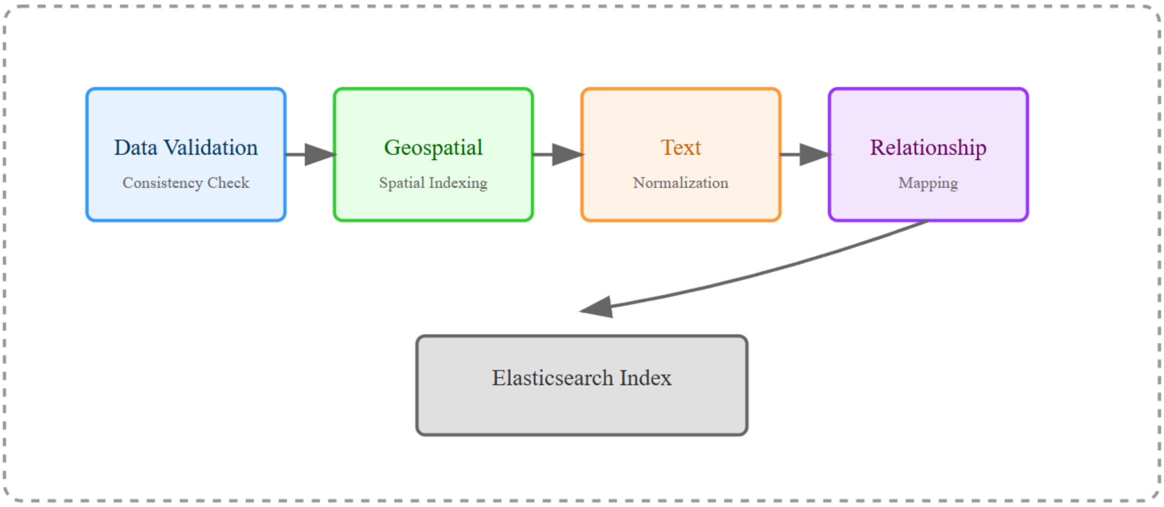


Fig. 5 GTFS Static Data Processing Pipeline

The indexed data structure optimizes for the types of queries most common in conversational interactions, prioritizing fast retrieval of stop and route information.

5.2 GTFS Realtime Integration

Real-time transit updates are essential for providing accurate and timely information to users. The GTFSRealtimeService operates continuously in the background, fetching live transit data from a GTFS Realtime producer server every 30 seconds. The service processes three key protobuf files:

- Trip Updates (tripUpdates) – Contains real-time modifications to scheduled trips, such as delays, cancellations, and changes in stop sequences.
- Vehicle Positions (vehiclePositions) – Provides the real-time geographic locations of transit vehicles, along with attributes like speed and occupancy status.
- Service Alerts (serviceAlerts) – Delivers notifications about unexpected service disruptions, detours, or major operational changes.

These real-time updates are structured using Protocol Buffers (protobuf), a compact and efficient data format optimized for transmitting structured data. The GTFSRealtimeService decodes these protobuf files, integrating the live feed with the static GTFS dataset. Where real-time information is available, it overwrites the corresponding static GTFS data, ensuring that users always receive the most up-to-date transit status.

By combining real-time data with static GTFS information, the system ensures accurate, up-to-date updates that reflect changes in transit networks. With frequent data refreshes, passengers receive precise, real-time details, reducing confusion and improving their travel planning. Additionally, the GTFS Realtime Service allows the application to continuously consume the real-time data feed, facilitating faster responses to disruptions and providing valuable insights into service performance. This dynamic system enhances the passenger experience and contributes to the overall efficiency and sustainability of public transportation.

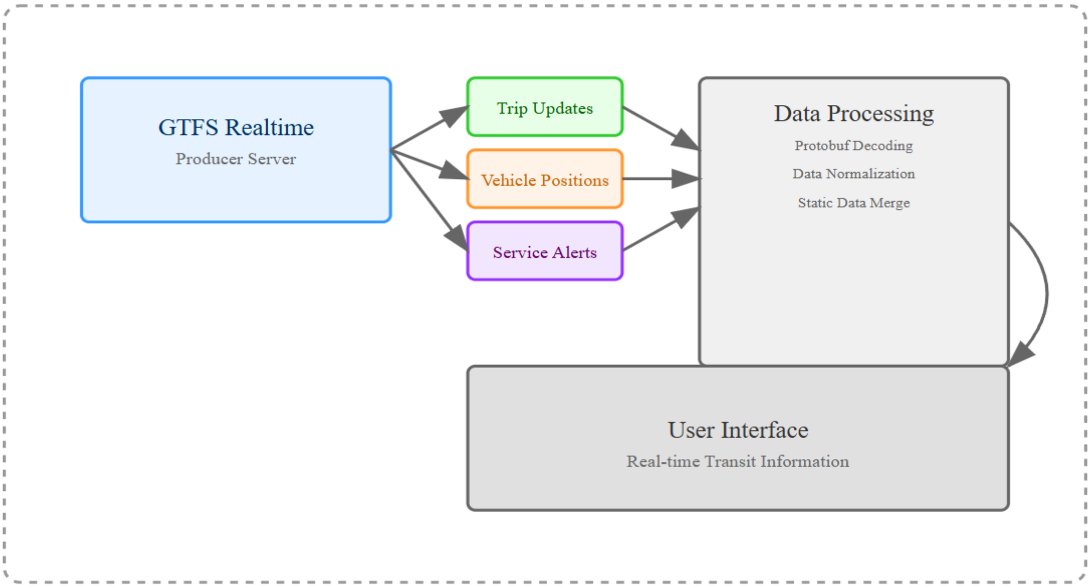


Fig. 6 GTFS Realtime Data Integration Pipeline

These real-time data points are integrated with the static schedule information to provide users with the most current transit status.

5.3 OpenTripPlanner Integration

For journey planning, TransitAI integrates with OpenTripPlanner through its GraphiQL API. This integration enables:

- Multimodal Routing: Planning journeys that combine various transportation modes;
- Time-Based Planning: Finding optimal routes based on departure or arrival times;
- Preference Handling: Incorporating user preferences for shorter walking distances.

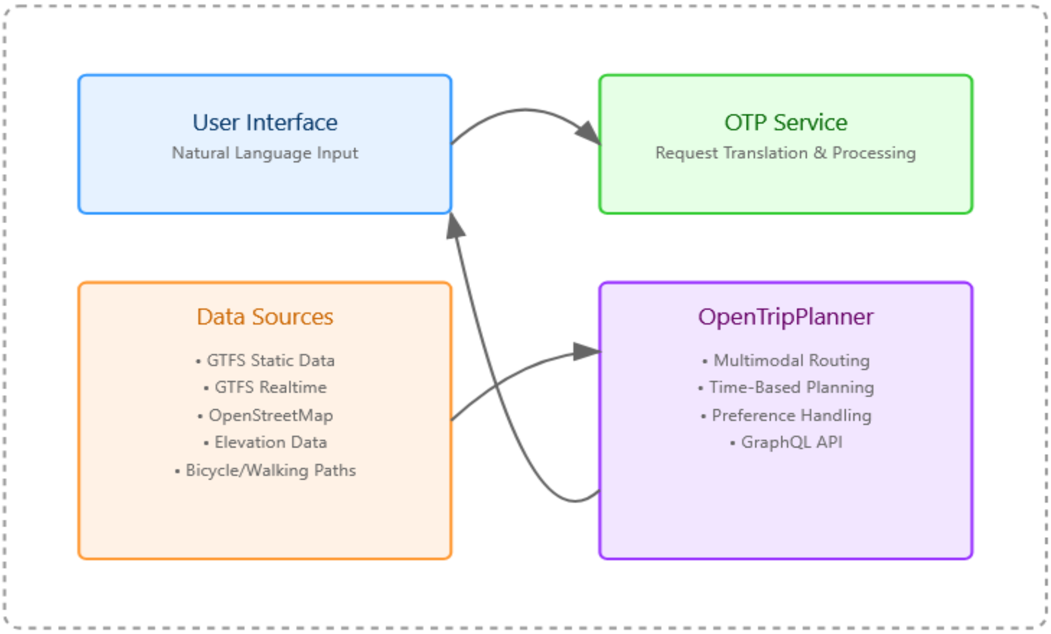


Fig. 7 OpenTripPlanner Integration Architecture

The OTPService translates natural language journey requests into structured API calls and processes the responses to extract the most relevant information for users.

6 Response Generation and Presentation

6.1 Natural Language Response Generation

TransitAI generates natural language responses that combine retrieved transit data with conversational elements. The response generation process:

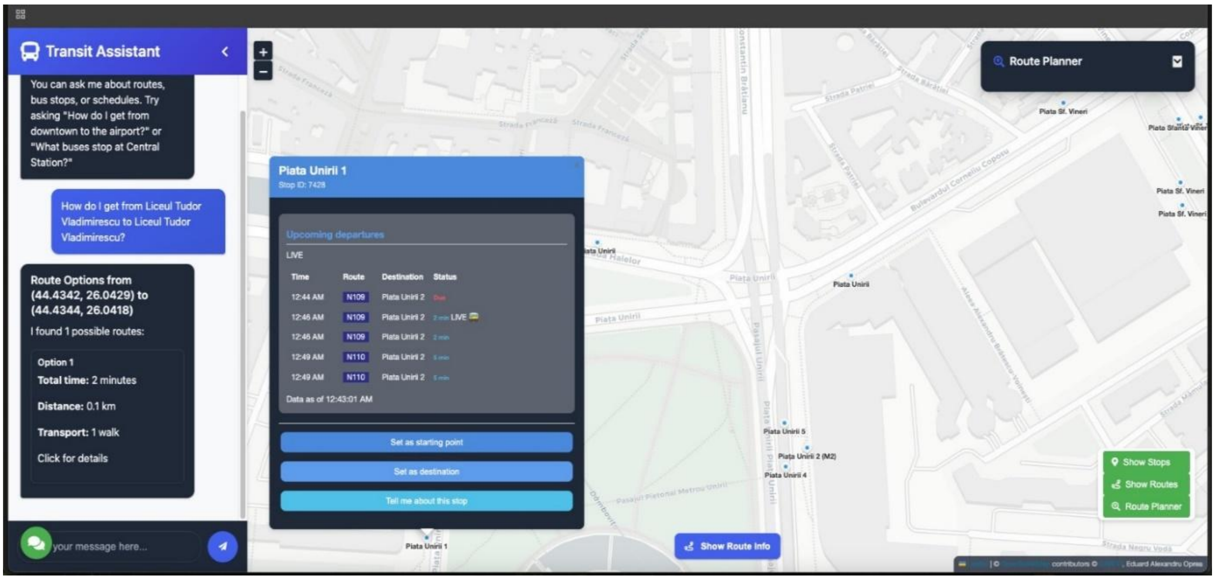
1. Structures the retrieved data based on query intent.
2. Selects the most relevant information to address the user's query.
3. Applies natural language templates with dynamic content insertion.
4. Adds contextual elements such as time references and location descriptions.
5. Incorporates conversational markers for a more human-like interaction.

This approach ensures that responses are informative yet conversational, avoiding the robotic quality of many transportation information systems.

6.2 Map Visualization

For spatial information such as stops, routes, and journeys, TransitAI provides interactive map visualizations. The MapService component creates:

- Stop Markers: Visual indicators of stop locations with popup information;
- Route Lines: Color-coded route visualizations with direction indicators;
- Journey Paths: Step-by-step visual representation of planned journeys;
- Vehicle Positions: Real-time locations of transit vehicles when available.



You can ask me about routes, bus stops, or schedules. Try asking "How do I get from downtown to the airport?" or "What buses stop at Central Station?"

How do I get from Liceul Tudor Vladimirescu to Liceul Tudor Vladimirescu?

Route Options from (44.4342, 26.0429) to (44.4344, 26.0418)

I found 1 possible routes:

Option 1

Total time: 2 minutes

Distance: 0.1 km

Transport: 1 walk

Click for details

Piata Unirii 1

Stop ID: 7428

Upcoming departures

LIVE

Time	Route	Destination	Status
12:44 AM	N109	Piata Unirii 2	Due
12:46 AM	N109	Piata Unirii 2	2 min LIVE
12:46 AM	N109	Piata Unirii 2	2 min
12:49 AM	N110	Piata Unirii 2	5 min
12:49 AM	N110	Piata Unirii 2	5 min

Data as of 12:43:01 AM

Fig. 8 Chatbot and Map Interface

These visual elements complement the text-based responses, providing users with spatial context that is difficult to convey in words alone.

6.3 Multi-Turn Conversations

TransitAI supports multi-turn conversations, maintaining context across interactions. This enables:

- Follow-up Questions: Users can ask for additional details without repeating context;
- Refinement Requests: Journey preferences can be adjusted based on initial results;
- Comparative Queries: Users can compare multiple routes or times;
- Clarification Dialogues: The system can request additional information when queries are ambiguous.

This conversational capability creates a more natural and efficient interaction model compared to traditional form-based transit information systems.

7 Conclusions and Future Work

7.1 Implications for Transit Information Access

TransitAI demonstrates that conversational interfaces can significantly improve access to transit information. The natural language approach removes barriers associated with complex transit terminology and interface navigation, potentially making public transportation more accessible to a broader audience.

The system’s ability to integrate real-time data with contextual understanding creates a more valuable user experience than static information delivery. By providing personalized, timely information, conversational assistants may help increase transit ridership by reducing uncertainty and improving the passenger experience.

7.2 Challenges and Limitations

Several challenges emerged during the development and testing of TransitAI:

- Location Ambiguity: Common place names and informal location references remain challenging to resolve;
- Complex Journey Preferences: Some nuanced preferences are difficult to express and fulfil in natural language;
- Data Quality Dependencies: The system’s accuracy is limited by the quality and timeliness of underlying GTFS data;
- Dialect and Language Variations: Regional terminology and non-native speakers present recognition challenges.

These limitations highlight areas for future improvement and research.

7.3 Future Directions

Several promising directions for future work include:

- Multilingual Support: Extending the system to handle multiple languages, particularly important in diverse urban areas;
- Multimodal Input/Output: Incorporating voice interaction and accessibility features;
- Personalization: Learning from user history to provide more tailored suggestions;
- Cross-Agency Integration:Expanding to support seamless planning across multiple transit agencies;
- Fare Information: Adding detailed fare calculation and payment integration;

Additionally, exploring the application of this architecture to other public service domains could yield valuable insights for smart city development.

7.4 Conclusion

TransitAI represents a significant step forward in making public transportation information more accessible by leveraging natural language interfaces. By combining large language models with specialized transit data processing, it effectively addresses the limitations of traditional transit information systems. The system's modular design also provides a flexible framework that could be applied to other domains where complex data needs to be made accessible to the general public.

As cities increasingly prioritize public transportation and sustainability, TransitAI has the potential to enhance the passenger experience, encourage greater use of transit systems, and contribute to smarter, more efficient urban environments. The fusion of artificial intelligence and public service information marks an exciting frontier for the future of smart cities.

References

[1] Hansen M., Glover, J., *Human-Computer Interaction: Designing for Diverse Users and Domains*, CRC Press, 2011
[2] *Moovit and Citymapper* URL: https://www.pcmag.com/news/transit-app-face-off-citymapper-vs-transit-vs-moovit
[3] Kramer A., Hoon, D., Challenges in Public Transport Systems for Elderly and Disabled Users, Springer, 2015.
[4] Helen Sharp, Jennifer Preece, Yvonne Rogers, *Interaction Design: Beyond Human-Computer Interaction (5th Edition)*, Wiley, 2019.
[5] Alonso J., Castillo E., *The Impact of Chatbots in Public Administration: Case Studies from European Governments, International Journal of Public Administration, vol. 41 (7)*, pp. 576-586, 2018.
[6] Zhao Y., Liu X., *Chatbot Design for Public Services: Enhancing Contextual Awareness and User Experience in Government Applications*, *Journal of Public Administration Research and Theory, vol. 31 (2)*, pp. 314-327, 2021.
[7] Nitsche A., Sieber A., *Real-Time Information and its Impact on Ridership and User Satisfaction in Public Transport Systems, Transportation Research Part A: Policy and Practice*, pp. 107-120, 2015.
[8] Zhang Y., Yang Q., *Named Entity Recognition for Public Transport Queries Using Natural Language Processing Techniques, International Journal of Transportation Science and Technology, 8(2)*, pp. 90-102, 2019.
[9] Tur G., De Mori R*., Spoken Language Understanding: From Theory to Practice*, Springer, 2011.

Eduard-Alexandru OPREA
National University of Science and Technology Politehnica Bucharest, Pitești University Center
Faculty of Sciences, Physical Education and Informatics
Department of Mathematics and Informatics
Str. Targul din Vale, nr. 1, Pitești
ROMANIA
E-mail: edyoprea69@gmail.com

Elena-Luiza BUZATU
National University of Science and Technology Politehnica Bucharest, Pitești University Center
Faculty of Sciences, Physical Education and Informatics
Department of Mathematics and Informatics
Str. Targul din Vale, nr. 1, Pitești
ROMANIA
E-mail: elenaluiza10@gmail.com

Ioana-Valeria TURCIN
National University of Science and Technology Politehnica Bucharest, Pitești University Center
Faculty of Sciences, Physical Education and Informatics
Department of Mathematics and Informatics
Str. Targul din Vale, nr. 1, Pitești
ROMANIA
E-mail: turcin.ioana@yahoo.com

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

AI Platform for Real-Time Cyber Threat Detection

Vlad-Matei Poienariu, Rares Muntean, Vlad-Stefan Alexandrescu

Abstract

This paper introduces an AI-powered application platform model for cyber threat detection in real-time. The System uses machine learning that is constantly adapting to new threat patterns. By efficiently monitoring the network traffic, the application's security is enhanced. The purpose of the application is to be able to handle many types of threats and attacks by adapting to malicious threats in various shapes and forms. After each interaction with malware, the AI is capable of learning new tactics, ensuring the application is constantly up to date with the new threats available. Implementing the AI in the program results in regularly fortifying the network's safety, reliability, and the User's experience. These advantages are provided by a properly trained machine learning model capable of smoothly receiving data from the System. The User is able to view everything and is notified when a new thread is being discovered by using a modern User-Interface dashboard. Automating security duties for the System, such as keeping track of the logs, alerts, or common incidents, helps minimize human errors and concentrates the team on more complex security threats, making it more efficient when it comes to traditional cybersecurity. Additionally, AI can predict future attacks, allowing organizations to prepare for them.

1 Introduction

The broadening attack surface and the development of advanced adaptive threats have created a landscape in cybersecurity that made old-guard defenses less effective than in the past. The rise of advanced persistent threats (APTs), polymorphic malware, and zero-day exploits has highlighted the need for dynamic, real-time protection systems. Tools like AI (artificial intelligence) and ML (machine learning) are no longer tools in the laboratory but an essential part of the infrastructure of modern cybersecurity [1-3].

The top AI cybersecurity platforms are able to utilize real-time data analytical processing, behavioral modeling, and abnormality recognition in order to analyze potential threats before they become breaches. Platforms like IBM's QRadar, Palo Alto's Cortex XDR, and Microsoft Defender for Endpoint utilize machine learning algorithms to sift through immense amounts of network traffic and identify bad activity with minimal human oversight. These platforms are focused on automation and predictive capabilities to reduce the mean time to detect (MTTD) and respond (MTTR) to threats. Collecting threat intelligence through unstructured data parsing of open-source intelligence (OSINT) has become more efficient, facilitated by developments in natural language processing (NLP) and deep learning that have made it possible for systems to analyze and recognize phishing campaigns, suspicious domains, or vulnerable assets. Reinforcement learning is also being investigated to enable AI agents to learn defense strategies autonomously in simulated environments of potential attacks as well [4-9]. However, there are still hurdles to overcome. Many current solutions depend on static models that need frequent retraining and have difficulty generalizing to various threat landscapes [10]. Moreover, because of their black-box nature, some AI models are perceived as not exposing model-decision-oriented accountability, resulting in concerns about transparency in some decision-making processes.

The main objective of this paper is to present a holistic software architecture for an AI-based platform for real-time cyber threat detection, laying out the advanced data ingestion pipelines, machine learning engines, and methods of alerting that can ultimately be used to help support SOC teams and cybersecurity analysts in being able to defend against threats proactively.

This goal requires the following specific objectives to be established in this paper:

- The research aims to synergize various data streams in a modular and scalable architecture, enabling real-time integration of diverse input sources, including logs, network traffic, and endpoint telemetry.
- To integrate AI/ML-based detection mechanisms, such as supervised and unsupervised learning models for anomaly detection, pattern recognition, and threat scoring.
- Create an automated detection and correlation engine that can aggregate indicators and correlate events through both AI output and rule-based logic.
- To formalize alerting and reporting mechanisms for timely notification and visualization, including integrations with SIEM, dashboards, and external intelligence feeds.

This will help with secure access control and robust data storage in line with cybersecurity best practices and compliance requirements. To this end, we present a new AI platform capable of building on preceding limitations by utilizing machine learning mechanisms such as learning from everyday malware interaction. We do this through improved detection, automated incident response, and feedback in a new UI. We endeavor to close this loop through our platform by invoking a combination of self-healing, continuous learning, and human-in-the-loop capabilities to provide expert foot-tracking without missing a beat.

1.1 The proposed AI Platform

In the following, we will introduce our proposed AI Platform. On each attack, the logs are saved while AI receives a set of data to prevent further attacks from happening. This way, the app's ability to detect and remove malware becomes more performant. The User can manually train the model with different data sets to efficiently reduce learning time. Each security log that emerges from different attacks has various parameters, like a solution and main cause. AI can shorten the time it takes to solve problems by connecting future problems with the ones that are currently known. Each malware attack differs in parameters; that is why the AI is recognizing problems with ease. To be able to observe the efficiency and security boost, we need a large data set to start with. In this line, it's important to create data samples that include common problems and solutions in order to detect attacks, resulting a more reliable security.

When a new technology emerges, and the program is not up to date with it, we can already tell that it is left behind, and the chances of risk grow exponentially. That is why AI, which is becoming more intelligent day by day, and many more abilities are becoming public with unlimited access for everyone, such as image generators and voice generators, has more chances of going undetected. In order to resolve this issue, the app must be updated with the latest data samples and issues on the User's device.

The application is able to offer tips regarding the User's privacy protection, like choosing a proper password and warning him about potentially dangerous websites. The user interface is modern and simple, making it straightforward for the User to access these features. The AI can recognize the programs that the User is normally using in a span of a few days. The User will be notified if a program has suddenly appeared and is being used at an inordinate rate in comparison to other standard programs, allowing him to take immediate action. Furthermore, the application can confirm the device's resource usage, and if it exceeds a specific threshold, it automatically isolates or deletes it. Because of the interface and the modern app, the User is up to date with everything that is going on, having access at any moment to eliminate or ignore the malicious files detected by the app.

The subsequent step will involve an examination of the most prevalent cyber-attacks that have a high success rate. We will observe their functionality and the extent to which an AI can avert these issues prior to their inflicting any harm.

2 Types of cyber attacks

A cyber attack is a type of malicious attack that an individual or an organization typically initiates to get sensitive data from another individual or organization, usually for their benefit. Moreover, 53% of the attacks typically caused $500,000 in damages or more; that is why it's critical to keep up with the new malware types and put a stop to them. The more hidden a virus is, the higher the There are chance it has to affect the device over a longer period. That is why the identification of the viruses is an essential procedure.

DDoS Attack—Once a site becomes popular, the chances of being a target of a cyber-attack are growing exponentially. Unfortunately, this popularity can make the website known to bad-intentioned people. *DDoS, or Distributed Denial of Service*, is a cyber-attack where computers infected with a common virus can be simultaneously controlled to make the number of accesses impossible to recreate by a human. An infected device can also be called a *bot*. A group of connected devices is called a *botnet*, and it can be used for various purposes, including launching *DDoS* attacks. Excessive calls to an *IP*, which can be a website or a device, overwhelm the network and make normal use impossible.

The detection of *DDoS* attacks is relatively simple because suddenly, the app or website is running very slowly. The logs and the exponential growth of calls to the server can also reveal *DDoS* attacks. By constantly analyzing the traffic, an AI security app can easily realize that a *DDoS* attack is happening. These calls can come from a single device (single IP) or multiple devices (multiple IPs). The OSI protocol stack is distributed in 7 layers. The number 7 layer is "Application", the layer that the DDoS attack acts on. Unfortunately, distinguishing between a normal *User* and a *bot* makes identifying attacks on this layer quite difficult.

Phishing Attack - Phishing is another type of cyber attack that first occurred in 1995. The word 'phishing' is a variation of 'fishing', which refers to attracting sensitive information. It is a type of social engineering and a scam where people are lured into revealing their personal information, like credit cards or login credentials, making it the most used cyber-attack method since 2020. Usually, these are present in email spam, SMS phishing, phone calls, social media, hyperlinks, and even webpages that claim to be another by replicating the interface 1:1. To be able to prevent these kinds of attacks, we should integrate into our AI models the most used social engineering tactics, the latest phishing campaign indicators, and also the known dangerous IPs or domains. We can also train the AI to look out for very similar domains (*g00gle.com* instead of *google.com*). Simulating phishing attacks is recommended because this type of malware, which usually interacts directly with humans, is difficult to train models with, allowing people to learn and adapt to these kinds of problems.

Malware Attack - Very popular and common, a malware attack is the usual dangerous cyber assault that leads to malicious software infiltrating a user's or company's System to cause harm. The usual and common types are viruses, trojans, spyware, and so on. Malware infections can occur in various ways, such as downloading a file that contains a virus or using an infected USB stick. Usually, if there are suspicious processes and memory usage, it means some malware is inside the System. By analyzing incoming and outgoing network traffic and file system activity, including concealed files, we could train the AI to detect these and warn of or prevent future problems.

3 The process of creating the Machine Learning model

In order to develop the machine learning model, we start by defining a variable called *"data"*, where we will stock *pd.read_csv('UNSW_NB15_training-set.csv', index_col=0)* [11]. The *read_csv()* function reads the data set that is extracted from a library which allows the download of more data from different Kagglehub domains. Furthermore, *data.shape* is a tuple that contains a number of rows and columns from the **.csv* type data set. This thing will help us make an idea about the file dimensions that we are working on.

```
data = pd.read_csv('UNSW_NB15_training-set.csv', index_col=0)
data.shape
data.info()
```

Herein, *data.info()* returns information about rows and columns present in the file, including the data type for each line from **.csv,* as well as the total number of columns displayed in the header.

```
data.duplicated().values.any()
data.count()
```

The next function *duplicated().values.any()* analyses the given data frame and returns *true* if there are duplicate rows inside. Additionally to the previously presented function, the *data.count()* function will return the total number of lines. This thing is being used because, through a small difference, we can find the total number of duplicated lines very easily.

```
data = data.drop_duplicates()
```

After calling these functions, it is necessary to call one more with the goal of eliminating the duplicates, the function being *data.drop_duplicates()*. To be able to see exactly the number of deleted duplicates, we will display again the previous function *data.count()*. The quantity of duplicated lines can be determined from this point. One other way to check if the duplicated lines have been eliminated is to call again, the *data.duplicates().values.any().* The result should now be 0. In case the result is not *0*, we can certainly state that an error has been encountered. Some of the present values in the **.csv* file are not the correct type for the machine learning algorithm. That is why they must be converted to the real numeric data type (float) using the *LabelEncoder()* function. We will select every column from the file using *data["columnName"]*, and we will assign to it the returned value from the *label_encoder.fit_transform(data[“columnName”]).*

```
label_encoder = LabelEncoder()
data["columnName"] = label_encoder.fit_transform(data["columnName"])
```

The *train_test_split* function separates *30%* of the features and *target_attack_category* variables. This option is set by the *test_size=0.3* parameter, value 0.3 representing 30% of a maximum of 1. Next, these *30%* will be kept separate so that at the end we can compare the predicted results to what we aim to generate an accuracy percentage for the trained model. The remaining *70%* of the data will be used for the actual training of the model. *X* represents the data provided by us through training and future results are illustrated by *Y*.

```
target_attack_cat = data["attack_cat"]
features = data.drop("label", axis=1).drop("attack_cat", axis=1)
```

```
X_train_attack_cat, X_test_attack_cat, y_train_attack_cat, y_test_attack_cat = train_test_split(features,
target_attack_cat, test_size=0.3)
```

The *RandomForestClassifier()* function creates an instance through which, after using the *fit()* function, we define which datasets the model should use to train. The data used are the previously specified *X* and *Y*.

```
model_label = RandomForestClassifier()
model_label.fit(X_train, y_train)
```

Next, we will save the model using the *joblib* library, from which we will call the *.dump()* function. We will use the data model to predict the attack categories from the data frame that we kept for testing (30%). We compare the predicted results with the actual ones to demonstrate the model's results.

```
joblib.dump(model_label, 'model_label.pkl')
predict_label = model_label.predict(X_test)
score = accuracy_score(y_test, predict_label)
print(f"{score * 100}%")
```

Having trained a Random Forest for a complex dataset of near real traffic and maliciousness, we had obtained pretty good performance as seen by the confusion matrix (see Fig. 1). This visual tool effectively illustrated the extent to which the algorithm was able to differentiate legitimate connections from cyberattacks.

It eventually learnt to classify most of the data packets accurately, classifying both normal and malicious traffic correctly. The main diagonal of the matrix displays instances where the model's predicted labels perfectly matched the actual labels — a desired outcome. The tall values along this axis suggests that the system does not only learn the traffic patterns, but can also detect them reliably on new data.

There were also far fewer in the off-diagonal regions — where the model gets it wrong. This is very important for a threat detection system in order to:

- False negatives (missed attacks) can result in serious security issues.
- Whereas false positives lead to unnecessary alerts and alert fatigue in SOC teams.

Using Random Forest this way validated the strength of Random Forest as an algorithm, as well as being a solid candidate for an AI-based threat detection system. The ability to incorporate a variety of features — including ports and protocols, response patterns and latency — enables complex and accurate learning.

These results confirm the vision outlined in the project's architecture, where a high performance machine learning engine is incorporated into a modular pipeline for automated real-time detection, event correlation and relevant alerts for cybersecurity teams.

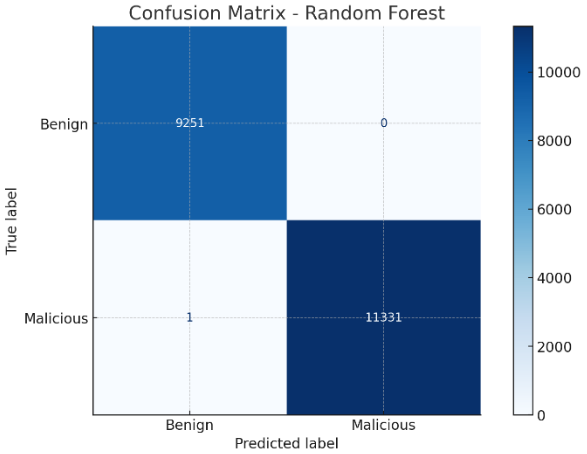


Fig. 1: The confusion Matrix – Random Forest

Now that we've built the Random Forest model (training during the previous Instantiated Random Forest), in this section, we'll be examining the model deeper and seeing how each of the individual features contributed towards the classification. This analysis showed what features were most important to the decision-making in the model.

The resulting diagram orders the top 15 features according to their corresponding importance in the prediction of a network connection being benign or malicious. Where some of the most important features were: sbytes (source bytes), dbytes (destination bytes), and rate. These metrics represent the volume and velocity of data transfer — often a strong indicator of abnormal or aggressive behavior typical of attacks.

Also of significance were the Time-to-Live (TTL) attributes for both the source sttl and destination dttl packets. A difference in these values can indicate packet manipulation or evasion techniques used in cyberattacks. Similarly, connection-level statistics like ct_state_ttl (state and TTL-based flows) and dmean (mean destination packet size) also emerged as important indicators of suspicious traffic patterns.

What this says to us is something significant: the model isn't just looking for easy flags (such as protocols used, service ports, etc.) — it's learning to identify traffic nuances, differences in behavior. This highlights the power of the synergy between rich telemetry and ensemble machine learning, which enables the system to go beyond rule-based designed logic to pattern recognition at scale.

The feature importance profile builds trust in the model's predictions and can help cybersecurity analysts with their work. Prefixing the attention towards the relevant attributes leads to focused monitoring, smarter rules, and real-time anomaly detectors — and stimulate all these in evidence-based and data-driven reason.

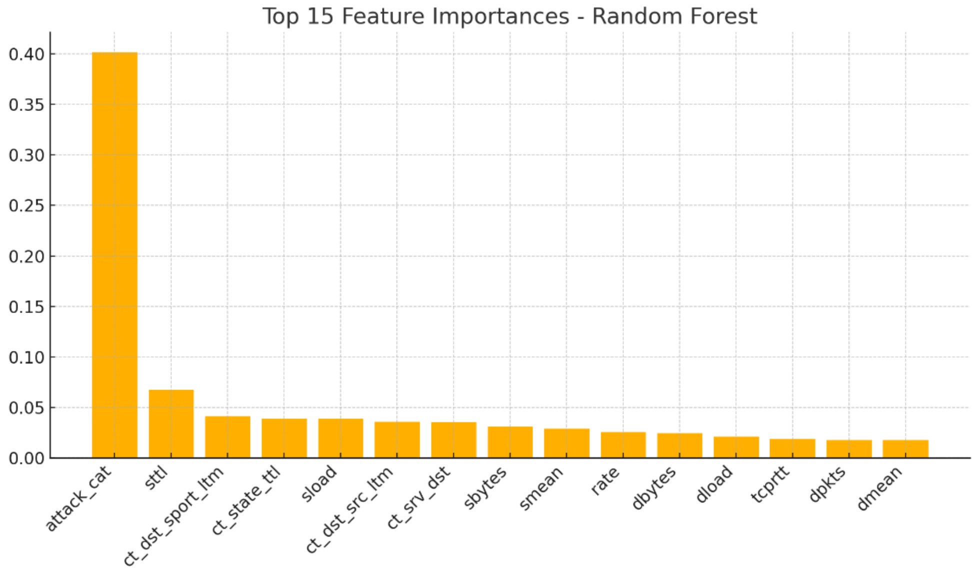


Fig. 2: Feature importances – Random Forest

4 The proposed AI Platform

Fig. 3 shows the component diagram of the proposed AI Platform. In the following, we will present a short description of our components within the AI Platform for Real-Time Cyber Threat Detection.

Data Ingestion Layer. Collects system logs, network traffic, endpoint activity, and external threat intelligence (e.g., OSINT) on an ongoing basis. It serves as the raw intake funnel for the System.

Preprocessing & Feature Engineering. Goes through raw input data, cleans it, and converts it to machine-readable formats. Things like normalization, feature extraction, and encoding to make the data ready for ML processing.

Machine Learning Core. The core of the platform applies supervised (i.e., RandomForest) and unsupervised models to identify anomalies and score threats. Compatible with future reinforcement learning integration and supports continuous learning.

Detection & Correlation Engine. Integrates AI outcomes with rule-based logic to correlate events, identify threats, and assign priority based on risk. It gives more context to improve the accuracy of threat detection.

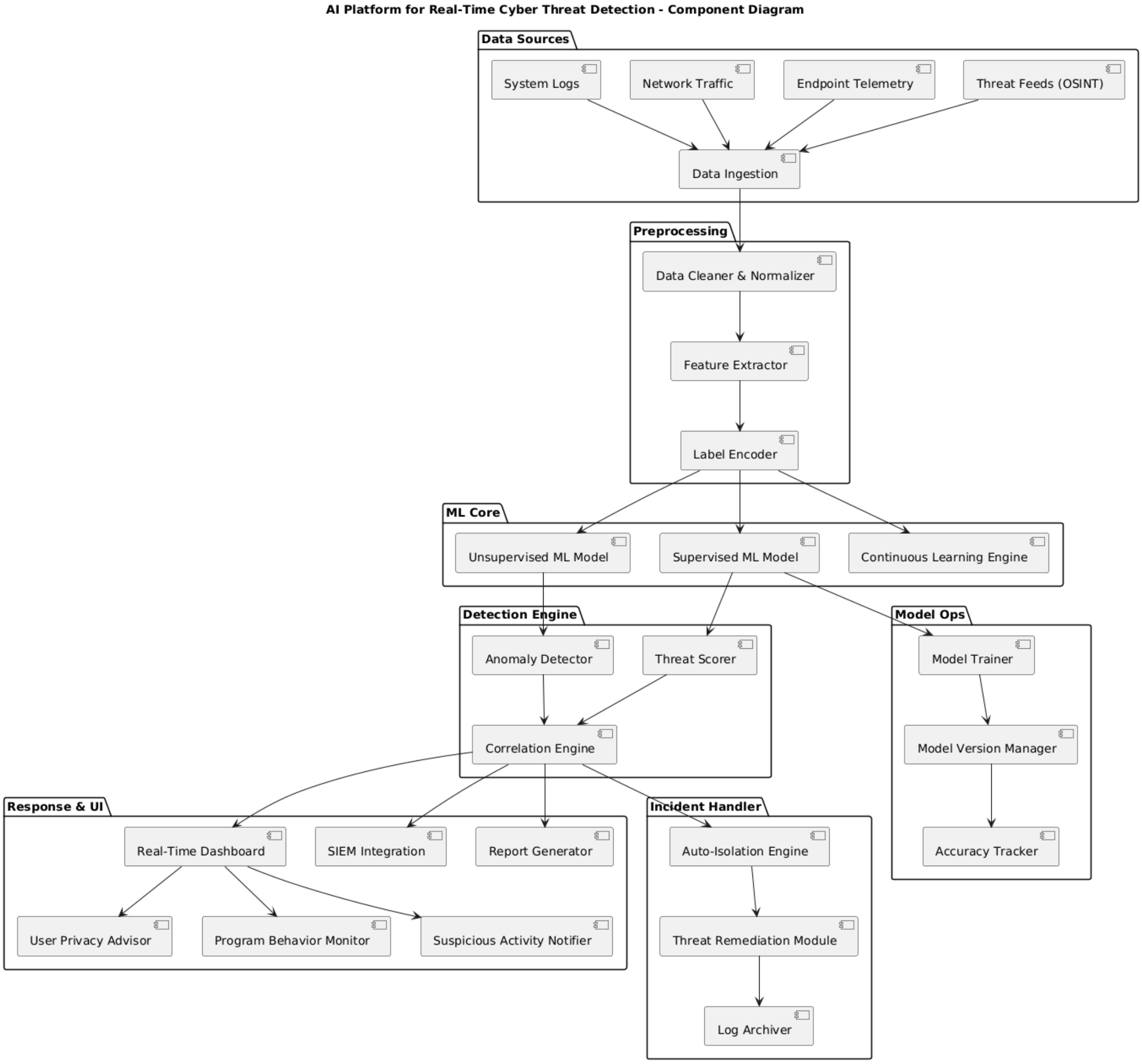


Fig. 3: Component Diagram of the proposed AI Platform

Alerting & Visualization Layer. Creates alerts and visual insights through a live dashboard. SIEM Tool Integrator and Automated Reports Generate for Situational Awareness

User Interaction Layer. It displays alerts, makes recommendations for privacy maneuvers, and keeps an eye out for strange behavior in apps. Keeps the users in the loop and aware.

Incident Response Module. Quarantines or removes detected threats and logs any actions taken. Allows for fast response while preserving forensic evidence.

Model Management Layer. Development of ML lifecycle using training pipelines tracking and monitoring of accuracy Keeps the model current and reliable.

In Fig. 4, we have inserted the *Use Case Diagram* of our proposed AI Platform. The *Use Case Diagram* describes the interactions of the main actors in the core AI Platform for Real-Time Cyber Threat Detection. This includes technical actors (e.g., AI Engine, Data Sources) as well as human actors (e.g., User, Security Analyst), describing what each one does within the System. Herein, the

User acknowledges alerts, provides feedback, and tracks system progress. Also, the *System* automates ingestion, ML inference, and threat correlation. Further on, the *Security analysts* deal with advanced monitoring and model updates. Moreover, external threat feeds ensure model relevancy by providing real-time threat indicators.

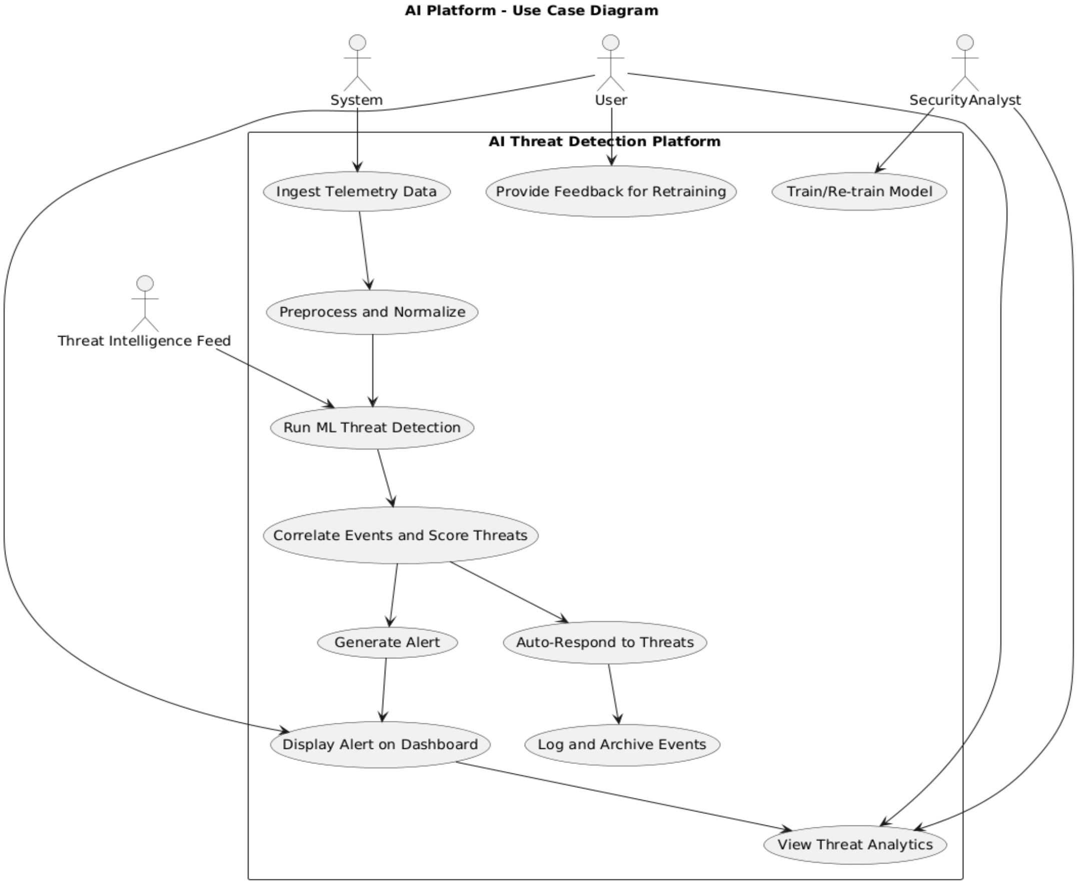


Fig. 4: Use Case Diagram of the proposed AI Platform

In Fig.5 it can be observed the *State Diagram* of the proposed AI Platform. The state diagram modeling the cyber threat lifecycle in the AI Platform. It shows how an event detected traverses through the System from detection to containment, followed by a user review and learning feedback. The lifecycle phases are the following: *detection* (the System receives the data stream in real-time), *analysis* (AI/ML tells), *target* (the machine and/or human), and *log* (feedback informs continuous improvement). This shows the development, analysis, and stakeholders to understand how the System performs in real-time response to different conditions of threats.

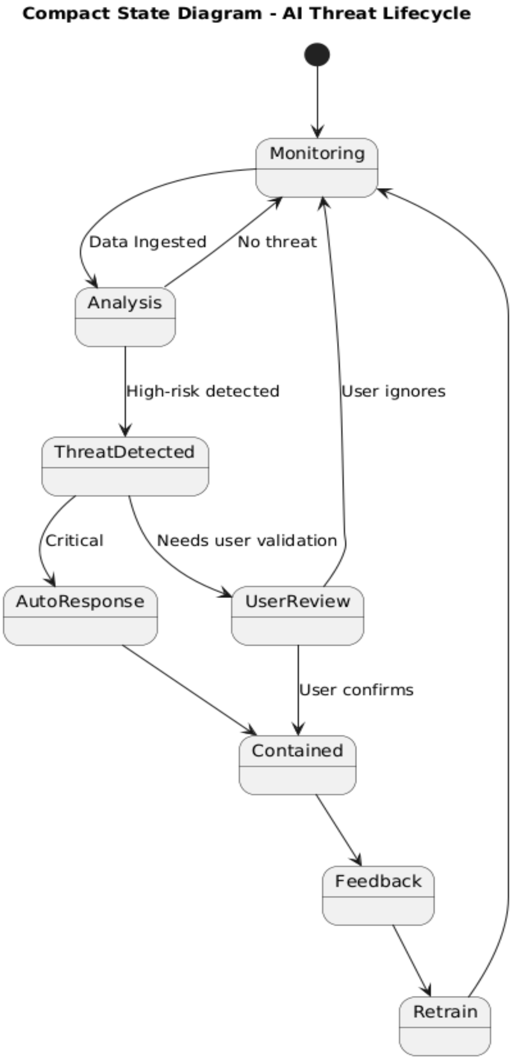


Fig. 5: The State Diagram of the proposed AI Platform.

Fig. 6 highlights the *Threat Flow Sequence diagram* of our proposed AI Platform. This is a sequence diagram where we are modeling how your AI platform processes and responds to a cyber threat in real-time. It assumes that the threat is being detected either through endpoint telemetry or network traffic, such as:

- Malware Execution
- Suspicious login (Phishing)
- Traffic spike (DdoS)

Fig. 7 shows the *Deployment Diagram for AI Platform for Real-Time Cyber Threat Detection*. This shows how the ML engine can be cloud-based and process local data, or a cloud-based UI can respond to local incidents, etc*, and the Agent + UI* is running on user endpoints (laptops, servers, and so on). The ML workloads are executed in the cloud for scaling, model version control, and external feeds.

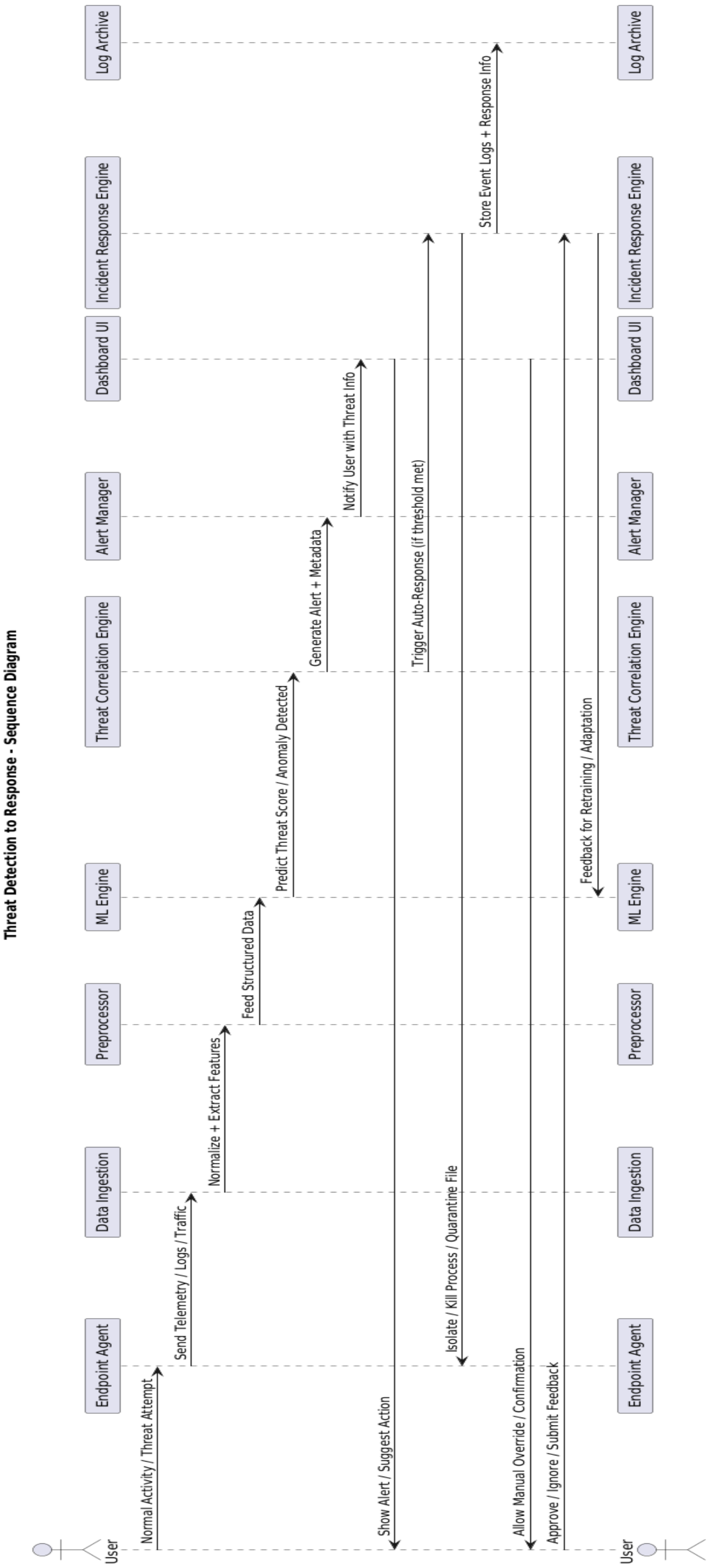

Threat Detection to Response - Sequence Diagram
User
Endpoint Agent
Data Ingestion
Preprocessor
ML Engine
Threat Correlation Engine
Alert Manager
Dashboard UI
Incident Response Engine
Log Archive
Normal Activity / Threat Attempt
Send Telemetry / Logs / Traffic
Normalize + Extract Features
Feed Structured Data
Predict Threat Score / Anomaly Detected
Generate Alert + Metadata
Notify User with Threat Info
Show Alert / Suggest Action
Trigger Auto-Response (if threshold met)
Isolate / Kill Process / Quarantine File
Store Event Logs + Response Info
Allow Manual Override / Confirmation
Approve / Ignore / Submit Feedback
Feedback for Retraining / Adaptation

Fig. 6: The Threat Flow Sequence Diagram (Detection to Response)

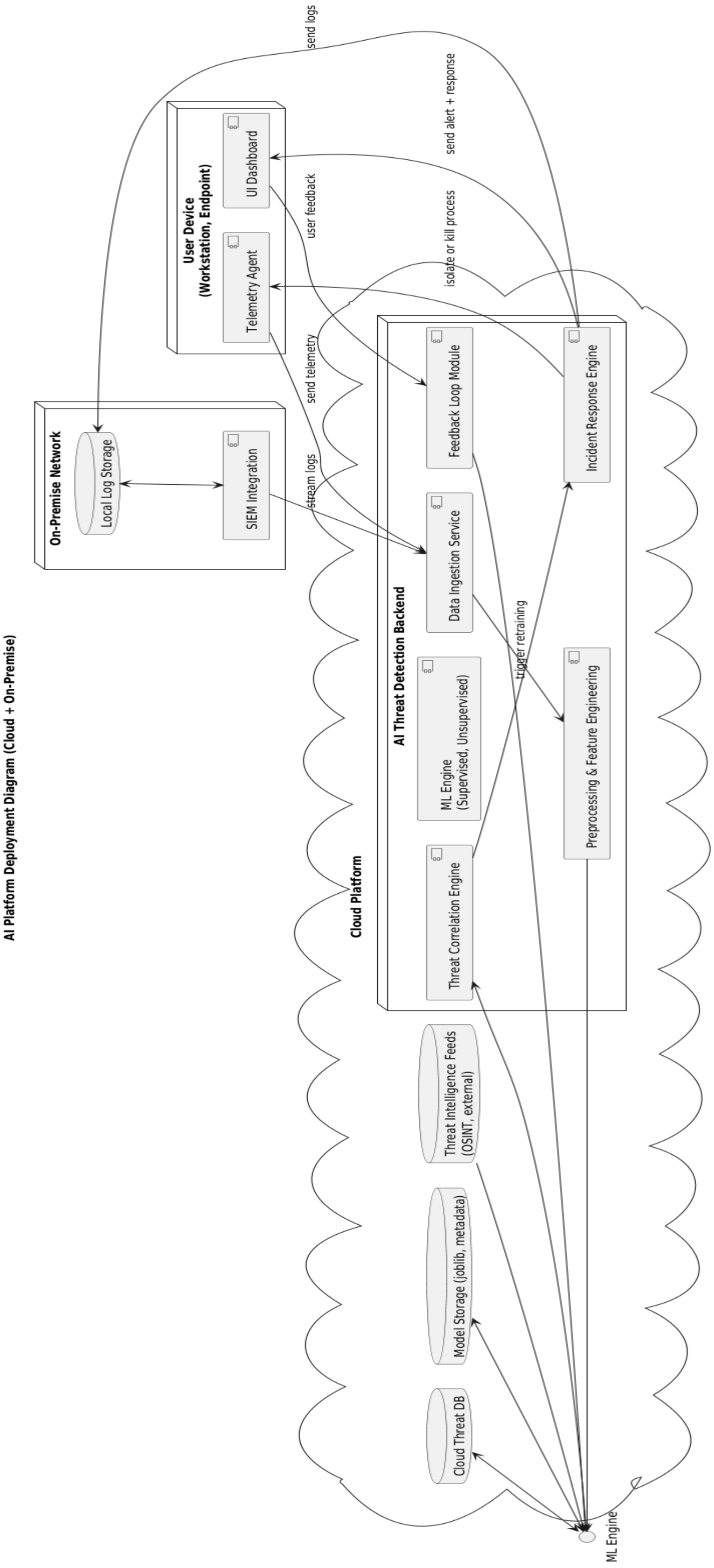
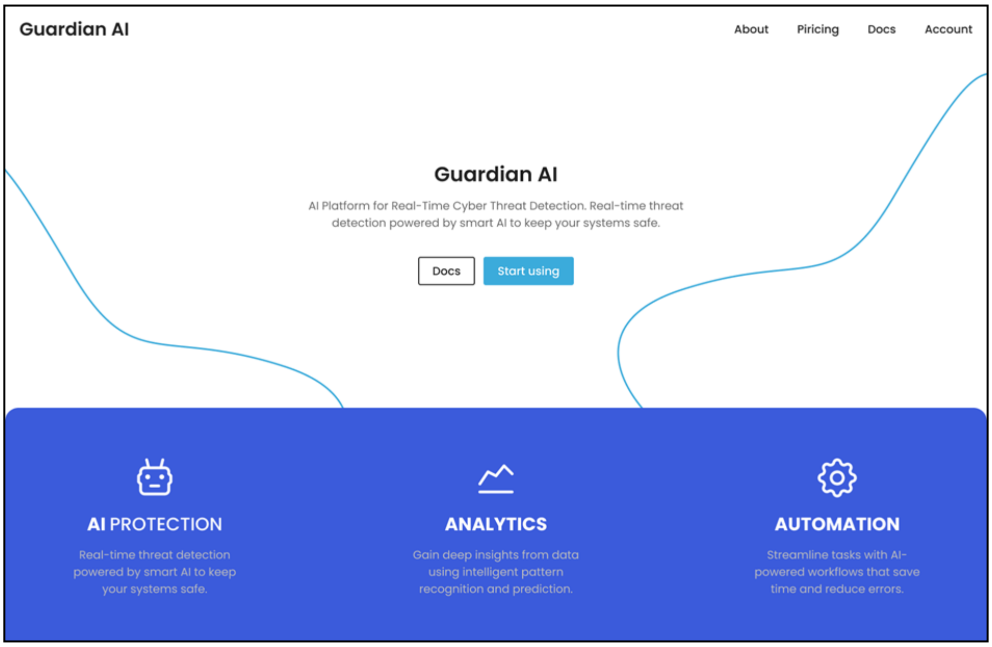


Fig. 7: Deployment Diagram – Hybrid AI Threat Detection Platform

The UI of the AI Platform for Real Time Cyber Threat Detection, which is presented in Fig.8, is designed to be modern, intuitive, and responsive, providing clear, actionable insights for common users as well as security analysts. This layout strikes a balance between functional clarity and streamlined aesthetics, enabling effortless navigation without losing the depth of control required for critical threat response operations. It facilitates real-time responsiveness and easy integration with back-end components, including the ML engine, alert manager, and incident response module.



AI Platform for Real-Time Cyber Threat Detection. Real-time threat detection powered by smart AI to keep your systems safe.

Real-time threat detection powered by smart AI to keep your systems safe.

Gain deep insights from data using intelligent pattern recognition and prediction.

Streamline tasks with AI-powered workflows that save time and reduce errors.

Fig. 8: Interface (UI) of the proposed AI Platform

5 Conclusions

Building a next-gen real-time cyber threat detection platform powered by the latest AI models. Unlike traditional approaches, the proposed-based platform detects and responds to known cyber threats while continuing to train and learn new attacks leveraging machine learning techniques. With automated incident recording, anomaly-based detection, and a user-friendly interface, the application improves cybersecurity reliability and responsiveness.

Based on the UNSW-NB15 dataset experimental outcomes, the accuracy in the number of attack categories has also been achieved at a high level, which confirms the reliability of the model training and feature engineering process. Moreover, the modularity of the platform allows for constant updates and scalability, making it a practical answer to the ever-evolving threat landscape.

Future work will focus on:

- Improve the explainability of information and AI decisions using model interpretability tools.
- Data set increases for zero-day threats and simulations of real-world attacks.
- SAP-enabling capabilities for broader use with SIEM, cloud-native platforms.

This work helps bridge the gap between automated threat detection and actionable cybersecurity intelligence, providing resiliency and real-time protection in modern IT environments.

Acknowledgement: This work was supervised by Lecturer Dr. Valentina Marascu, from National Institute for Laser, Plasma & Radiation Physics (INFLPR) and Faculty of Engineering and Computer Science, SPIRU HARET University; Associate Professor Dr. Marius Iulian Mihailescu from the Faculty of Engineering and Computer Science, SPIRU HARET University; and Lecturer Dr. Stefania Loredana NITA from Institute for Computers & Faculty of Information Systems and Cybernetic Security", Military Technical Academy "Ferdinand I."

References

[1] Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Publisher : Pearson; 1st edition (August 1, 2008), ISBN-10: 9780132350884.
[2] Jon Erickson, *Hacking: The Art of Exploitation*, Publisher: No Starch Press; 2nd edition (February 4, 2008), ISBN-10 : 1593271441.
[3] Andreas C. Müller, Sarah Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*, Publisher: O'Reilly Media; 1st edition (November 15, 2016), ISBN-10 : 1449369413.
[4] S. P. Satpathy, S. Raj, A. K. Dash, S. K. Gouda and S. Jha, *Enhancing Cybersecurity with Machine Learning: Anomaly Detection and Threat Prediction*, 2024 International Conference on Artificial Intelligence and Emerging Technology (Global AI Summit), Greater Noida, India, 2024, pp. 1259-1264, doi: 10.1109/GlobalAISummit62156.2024.10947851.
[5] James Kurose, Keith Ross, *Computer Networking: A Top-Down Approach*, Publisher: Pearson; 7th edition (April 26, 2016), ISBN-10 : 9780133594140.
[6] V. S. S. R. Nallapareddy and S. K. R. Katta, *AI-Enhanced Cyber Security Proactive Threat Detection and Response Systems*, 2025 4th International Conference on Sentiment Analysis and Deep Learning (ICSADL), Bhimdatta, Nepal, 2025, pp. 1510-1514, doi: 10.1109/ICSADL65848.2025.10933436.
[7] N. K. Alapati and S. Dhanasekaran, *The Performance Analysis of Cyber Threat Detection Model Using AI based Classifiers*, 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-5, doi: 10.1109/GCCIT63234.2024.10862442.
[8] V. R. Saddi, S. K. Gopal, A. S. Mohammed, S. Dhanasekaran and M. S. Naruka, *Examine the Role of Generative AI in Enhancing Threat Intelligence and Cyber Security Measures*, 2024 2nd International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2024, pp. 537-542, doi: 10.1109/ICDT61202.2024.10489766.
[9] R. Vadisetty and A. Polamarasetti, *Generative AI for Cyber Threat Simulation and Defense*, 2024 12th International Conference on Control, Mechatronics and Automation (ICCMA), London, United Kingdom, 2024, pp. 272-279, doi: 10.1109/ICCMA63715.2024.10843938.
[10] D. Chaudhary, S. K. Verma, V. Mohan Shrimal, R. Madala, R. Baliyan and S. M, *AI-Based Methods to Detect and Counter Cyber Threats in Cloud Environments to Strengthen Cloud Security*, 2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT), Greater Noida, India, 2024, pp. 1-6, doi: 10.1109/ICEECT61758.2024.10739173.
[11] Muntean Rares, *Basic Model Training Cyberthreats*, https://github.com/MunteanRares/basic-model-training-cyberthreats (last accessed on April 11, 2025).

Vlad-Matei Poienariu
SPIRU HARET University
Faculty of Eng. and Computer Science
46G Fabricii Street, District 6, Bucharest
ROMANIA
Email: g.vlad.poienariu@spiruharet.ro

Rares Muntean
SPIRU HARET University
Faculty of Eng. and Computer Science
46G Fabricii Street, District 6, Bucharest
ROMANIA
Email: g.rares.muntean@spiruharet.ro

Vlad-Stefan Alexandrescu
SPIRU HARET University
Faculty of Eng. and Computer Science
46G Fabricii Street, District 6, Bucharest
ROMANIA
Email: g.vlad.alexandrescu@spiruharet.ro

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Multi-Task Learning vs. Individual Models in Scarce Data Scenarios - Stock Price Prediction

Andrei Priboi

Abstract

This paper analyzes the application of a Multi-task learning methodology for financial time series forecasting, particularly stock price prediction. We will compare and analyze two prediction models, one trained on individual company data, and one trained on data ranging from different stocks and domains, which leverages cross-stock knowledge transfer. The empirical foundation is historical closing price data for 50 major stocks, from 2020 to 2023, and the model architecture was optimized via a random search for the best hyperparameter values. Results demonstrate a +70% increase in prediction accuracy when exploiting Multi-task learning compared to the individual models. Evidence is revealed that major trends in stock price movements can be transferred effectively between firms and market sectors. Furthermore, this study identifies market capitalization, volatility, and sector classification as significant factors influencing the benefits of Multi-task learning. The benefits can also be extended to time series analysis in other domains, such as healthcare patient monitoring systems or in energy consumption forecasting.

Keywords: Multi-task Learning (MTL), Long Short-Term Memory (LSTM) Networks, Hyperparameter Optimization, Stock Prediction, Financial Time Series

1 Introduction

Stock market prediction has always been a major goal for companies and individuals seeking to thrive in the domain of financial analysis, specifically in investment strategies, risk management, and economic forecasting. Traditional approaches usually depend on both statistical models and technical indicators, which tend to fail at capturing the complex and non-linear patterns in stock evaluation movements. Recent discoveries in the field of deep learning, specifically Long Short-Term Memory (LSTM) networks, have opened new horizons for a more accurate prognosis of time series in the financial domain.

This research is motivated by two key observations. First, while a stock's evaluation and price patterns indicate certain particularities shaped by company fundamentals, they are also influenced by overall market dynamics, sector performance, and the overall macroeconomic state. Second, effective training of deep learning models is dependent on huge quantities of data, which may be nonexistent for newer companies or companies with a shallow trading history.

Multi-task learning is a method in which multiple learning tasks are solved simultaneously, while exploiting commonalities and differences across tasks. This procedure can be leveraged in our case, where a model trained on cumulative data from multiple stocks can potentially achieve far better results when compared to training individual models, in cases where data might be insufficient. Also, this approach might uncover general trends in market behavior that are not tied to individual firms or industries, aiding in the understanding of market dynamics.

1.1 Related work

When it comes to the application of machine learning to financial forecasting, we have made great strides over the last decade. Early work by Gers et al. [1] showed the capability of recurrent neural networks for time series forecasting, while subsequently Fischer and Krauss [2] applied Long Short-Term Memory (LSTM) networks in stock market prediction, achieving superior performance when compared to classic approaches.

More recently, the focus has turned to hyperparameter tuning of the model architecture. Li et al. [3] explored different hyperparameter optimization (HPO) methods of LSTM networks for stock prediction and achieved a significant improvement in prediction accuracy when fine-tuning. In addition, Song et al. [4] used transfer learning and adaptive HPO, but applied it only to pairs of similar stocks rather than across a wider set of firms and domains.

Furthermore, hybrid approaches leveraging graph neural networks (GNN) [5] or convolutional autoencoders (CAE) [6] have been leveraged for even lower prediction errors in financial forecasting. Selvin et al. [7] studied stock prediction as identifying the latent dynamics of the evaluation, leveraging RNN, LSTM, and CNN-sliding window models. This work is complemented by Istiake Sunny et al. [8], applying LSTM and Bi-Directional LSTM networks. The proposed deep learning approach and HPO process achieved precise predictions for the stock market.

Jeong and Kim studied the idea of transfer learning in financial domains [9] and showed transfer of knowledge between similar stocks from the same sector. Merello et al. [10] studied a transfer learning approach, where the optimization problem was formulated as *"regression of market returns"*. This approach proved beneficial. Building on these ideas, Qiu et al. [11] demonstrated transfer learning across a wide range of market indices and illustrated that learning on one market could enhance predictive conditions on other markets.

1.2 Objectives

In spite of these advances, few studies have evaluated the effectiveness of Multi-task learning across a wider set of stocks or identified specific conditions under which knowledge transfer is beneficial. Also, comprehensive HPO for Multi-task learning in stock prediction remains underexplored.

Going forward, we will analyze the advantages of multi-task learning across the stock market, glancing at the different hyperparameter configurations and their implications in the prediction of company evaluations. We will explore how limited historical data may affect model performance across different metrics, such as Root Mean Squared Error (RMSE) for accuracy prediction and other such benchmarks.

2 Methodology

2.1 Data aggregation and preprocessing

The financial data for this study is downloaded via the Yahoo Finance API. [12] As mentioned above, our purpose is to show improvements in prediction accuracy for companies with a short trading history. For these purposes, we will use only 3 years of closing prices (evaluations at market close), in order to simulate the lack of historical data. We will use 50 top companies on The New York Stock Exchange from multiple fields, such as technology (Apple, Google, IBM, Microsoft), e-Commerce and retail (Amazon, AliBaba, GameStop), finance and asset management (BlackRock, Goldman Sachs), pharmacy (AbbVie, Pfizer) and others.

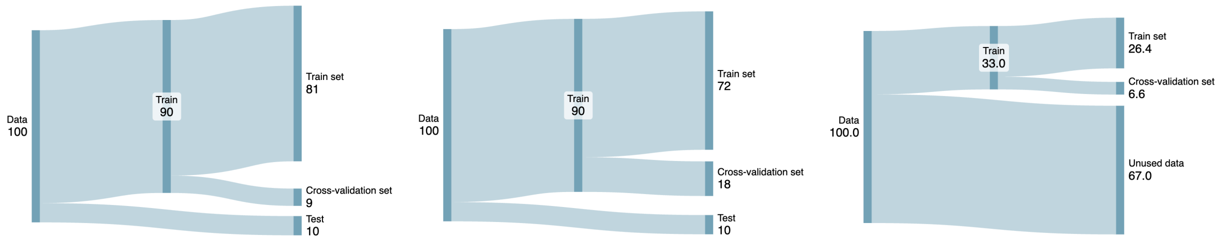
All of the data is organized as (x, y) pairs, where x is considered the feature vector and y the label. In this case, x is a vector consisting of closing prices for a certain stock in the past 60 days (our desired look-back period), and y is the price value for the following day.

For the train-test split, we reserve 10% of the data for testing model performance. When optimizing the hyperparameters, we use 20% of the training data for cross-validation; when training the model, we use just 10%. This approach ensures more reliable estimates for different hyperparameter configurations while keeping enough data for the actual training. The smaller cross-validation set used for training ensures more data for the model to learn from while avoiding overfitting.

When training the Multi-task learning model, we will train on 45 of the 50 companies we have selected. However, we won't use all training data for tuning the hyperparameters, as this is computationally expensive and, in practice, proven to not impact model performance. As a result, a third of the training data will be used, spread equally over all tickers and the entirety of the time-frame. Finally, we will test on 5 random stocks that were completely excluded from the training process to prevent information leakage and selection bias. [15]

Since this model uses an LSTM network, the order in which we feed it data is crucial. Despite this, for the multi-task model, all the data is shuffled pretraining, disrupting the temporal flow. Despite this inconvenient approach, the multi-task model still yields better prediction accuracy compared to the individual models, which feed on temporally ordered data.

The data split can be observed in Figure 1, below (for network training - both models in Figure 1a, for hyperparameter tuning - individual model in Figure 1b and for hyperparameter tuning - general model in Figure 1c):



(a) Data split for LSTM network training - both models

(b) Data split for hyperparameter tuning - individual model

(c) Data split for hyperparameter tuning - general model

Figure 1: Data split

2.2 LSTM network architecture and hyperparameters

Both the individual and the multi-task model share the same neural network architecture. The recurrent neural network starts with one input layer, whose size is the desired look-back distance, followed by two LSTM layers with forget gates. The first LSTM layer outputs an exit value for every time step in the input sequence, in order for the second layer to capture the temporal data of the look-back window better. Given that the second LSTM layer is followed by a simple Dense layer, it outputs just the exit value for the last time step in the sequence. After each LSTM layer, we introduce Dropout layers for regularization. We train the model using the Adam optimizer (Adaptive Moment Estimation), which leverages inertia and other concepts to avoid local minima while being computationally efficient. [16]

Hyperparameter	**value interval**	**step**
lstm_units_1	32 – 128	32
dropout_rate_1	0.1 – 0.5	0.1
lstm_units_2	32 – 128	32
dropout_rate_2	0.1 – 0.5	0.1
dense_units	20 – 80	20
learning_rate	{1e-3, 1e-4, 5e-4}	-

Table 1: Hyperparameter value intervals and steps

For hyperparameter optimization, the number of neurons in both LSTM layers and the Dense layer is of interest, as well as the dropout rate after each LSTM layer. Learning rate is also something we want to fine-tune in this process. All hyperparameter value intervals and steps can be observed in Table 1. Using these values, we define the hyperparameter space and perform randomized search with 20 trials.

While batch size was not discussed yet, we manually played with batch fractions of 0.01, 0.005, and 0.001, in order to check for any improvements in model accuracy. Because no real differences were recorded, all of the results below are generated while running a batch fraction of 0.01.

2.3 System architecture and implementation details

This research was carried out on an Apple Mac Mini, with 10 CPU cores (M4 chip), 10 GPU cores, and 24 GB of RAM (unified memory). The model was written in Python 3.10 (not the latest, due to incompatibilities with TensorFlow versions), using TensorFlow 2.18 and Keras 3.9. For hyperparameter optimization, we will use KerasTuner (version 1.4.7).

The foundation of our research is a simple, 2-layer, LSTM network, implemented below:

```
def build_lstm_model(hp, input_shape):
    model = Sequential()
    model.add(Input(shape=input_shape))

    model.add(LSTM(units=hp.Int('lstm_units_1', min_value=32,
        max_value=128, step=32), return_sequences=True))
    model.add(Dropout(hp.Float('dropout_rate_1', min_value=0.1,
    max_value=0.5, step=0.1)))

    model.add(LSTM(units=hp.Int('lstm_units_2', min_value=32,
        max_value=128, step=32), return_sequences=False))
    model.add(Dropout(hp.Float('dropout_rate_2', min_value=0.1,
    max_value=0.5, step=0.1)))

    model.add(Dense(units=hp.Int('dense_units', min_value=20,
        max_value=80, step=20), activation='relu'))
    model.add(Dense(1))

    learning_rate = hp.Choice('learning_rate', values=[1e-3, 1e-4, 5e-4])

    model.compile(optimizer=Adam(learning_rate=learning_rate),
        loss='mean_squared_error')

    return model
```

All hyperparameters specified above are being set in the code above, using KerasTuner. This network architecture is used throughout the research, for each stock and for the Multi-task model, with only the training data changing. The random search for hyperparameter optimization is implemented below:

```
    tuner = kt.RandomSearch(
        lambda hp: build_lstm_model(hp, input_shape),
        objective='val_loss', max_trials=max_trials,
        executions_per_trial=executions_per_trial, directory='hyper_tuning',
    )

    tuner.search(X_train, y_train, validation_split=0.2,
        epochs=epochs, verbose=verbose)
```

The optimization process searches for 20 trials (our desired count) inside the hyperparameter space, using 10 epochs and a 20% validation split, as mentioned above. The optimization objective is validation loss, which the process is trying to minimize.

The rest of the code used for this experiment, model training, and result exploration is publicly available at https://github.com/andreiprb/MTL_HPO_stock_predictor.

3 Individual model performance

As expected, individual model performance falls relatively short, with some particular stocks achieving Root Mean Squared Errors (RMSE) above 0.2 (20% prediction error). We will later observe that the cause is the (simulated) lack of trading data. We will evaluate the following stocks, for relevance and diversity reasons: AbbVie, BlackRock, GameStop, Google and AT&T. Out of the five companies of interest, AT&T performed most poorly in terms of RMSE, scoring 0.26, followed by BlackRock with 0.15 and AbbVie with 0.12. Google and GameStop performed well in comparison: 0.08 and 0.05 (see Table 5, IM-t). The price predictions can be observed in the plots below:

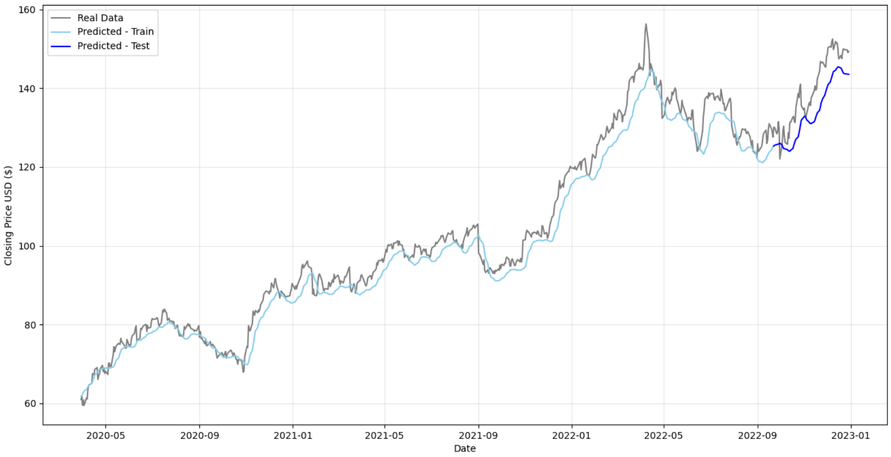


Figure 2: Individual model performance - ABBV

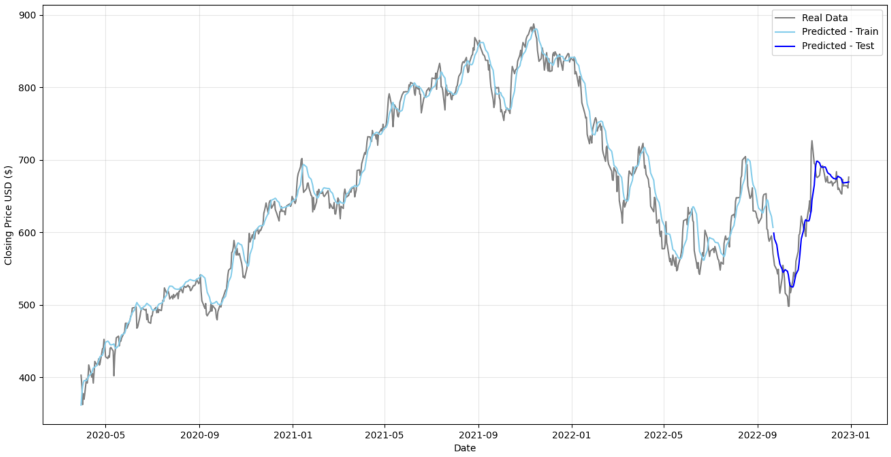


Figure 3: Individual model performance - BLK

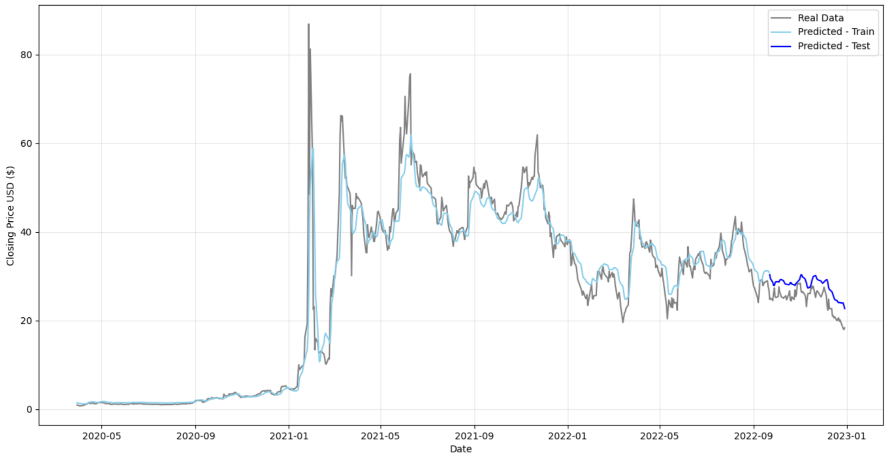


Figure 4: Individual model performance - GME

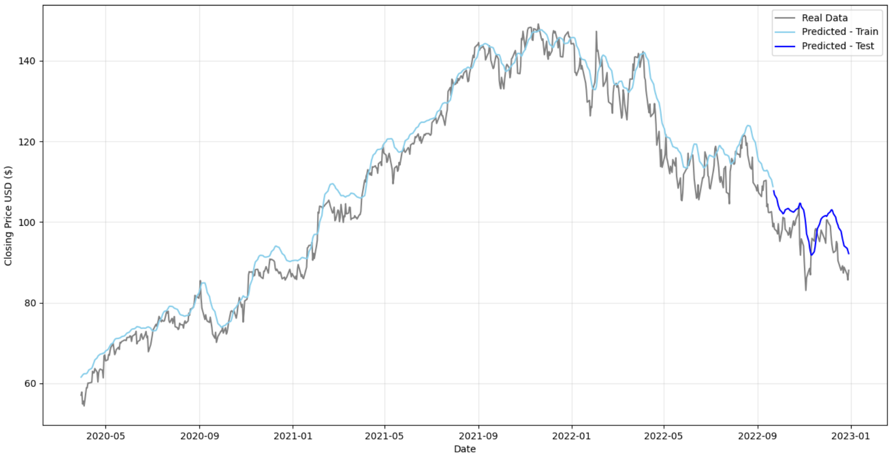


Figure 5: Individual model performance - GOOGL

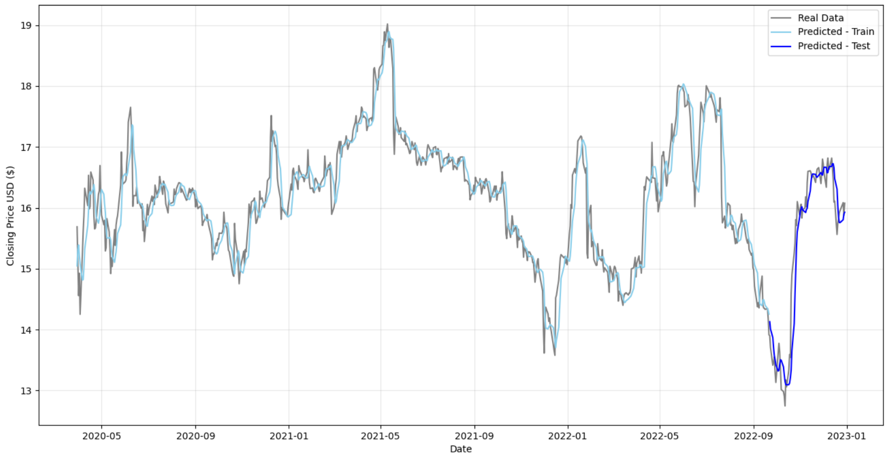


Figure 6: Individual model performance - T

The observation clearly shows the model's incapacity to completely learn the stocks' price fluctuations, as the gray and blue squiggles often fail to overlap. We can see a clear tendency to underevaluate the price of AbbVie (Figure 2) shares throughout the entirety of the time frame. A similar event is present for Google, where the model constantly overevaluates the stock price (see Figure 5).

For comparison, we will look at the performance of the GameStop individual model, which performed significantly better. Despite this, when taking a look at Figure 4, it is evident that the LSTM network failed to both predict and evaluate the huge price gain at the end of 2021, with a consistent overvaluation throughout 2023. During 2022, the model managed to correctly capture stock fluctuations, explaining the low RMSE.

Despite the higher RMSE, BlackRock and AT&T generated good plots (see Figures 3 and 6).

3.1 Hyperparameter observations

When analyzing the chosen hyperparameters, the individual models often opt for denser and last layers. Given that data is scarce, the tuning process tries to overcome this by increasing the number of nodes, in hopes of extracting richer feature representations (see Table 2).

This approach is not optimal, and leads to overfitting, which the model tries to fix by implementing higher Dropout rates and smaller second layers. This is especially visible for the ABBV model, where the hyperparameter optimization process chose the same rates of 0.5 and 0.4. In general, the hyperparameters were found to take identical values, despite tuning them on three different batch sizes.

Model performance also didn't really depend on batch size, with some models, like AbbVie and AT&T, performing better on lower batch sizes, while other models did the reverse, like BlackRock or Google.

RMSE (normalized)	**Stock**	**LSTM units 1**	**Dropout units 1**	**LSTM units 2**	**Dropout units 2**	**Dense units**	**Learning rate**
0.1327	ABBV	96	0.5	128	0.4	40	0.001
0.1488	BLK	128	0.1	96	0.1	40	0.001
0.0837	GME	96	0.5	64	0.3	60	0.001
0.0993	GOOGL	128	0.3	96	0.5	80	0.001
0.2571	T	128	0.2	96	0.1	60	0.001

Table 2: Individual models' hyperparameters and performance

4 Multi-task model performance

Just by looking at the plots below (Figures 7 to 11), we can see significant improvement in prediction accuracy. Training on notably more data proves to greatly aid the learning process. Overall, we see a +70% increase in prediction accuracy compared to the individual models. Note that these 5 companies' data was not used while training the model, and that model architecture is identical.

We can observe that multi-task learning completely fixed some of the problems mentioned above, like the consistent undervaluation/overvaluation present in the evaluation of AbbVie and Google stocks. Moreover, in the cases of AbbVie and GameStop, given the abundant training set, the model managed to predict the price spikes better.

One mistake of the multi-task learning model we can observe is GameStop's overvaluation before 2021. This can be traced to both the small stock price compared to the general market and the overall market growth. More complex systems can easily fix this error.

4.1 Hyperparameter observations

As there is more training data, we notice dropout rates decreasing significantly, as the vaster volume of training data reduces the risk of overfitting. The fine-tuning process finds the same hyperparameters as optimal, despite using only 30% of the data for the process. Learning rate does not suffer any change between the individual model and this one.

LSTM units 1	**Dropout units 1**	**LSTM units 2**	**Dropout units 2**	**Dense units**	**Learning rate**
128	0.2	128	0.2	20	0.001

Table 3: MTL models' hyperparameters

	ABBV	**BLK**	**GME**	**GOOGL**	**T**
Normalized	0.0277	0.0360	0.0208	0.358	0.0562
Original	\$2.23	\$21.51	\$1.79	\$3.46	\$0.42

Table 4: MTL models' RMSE for test stocks

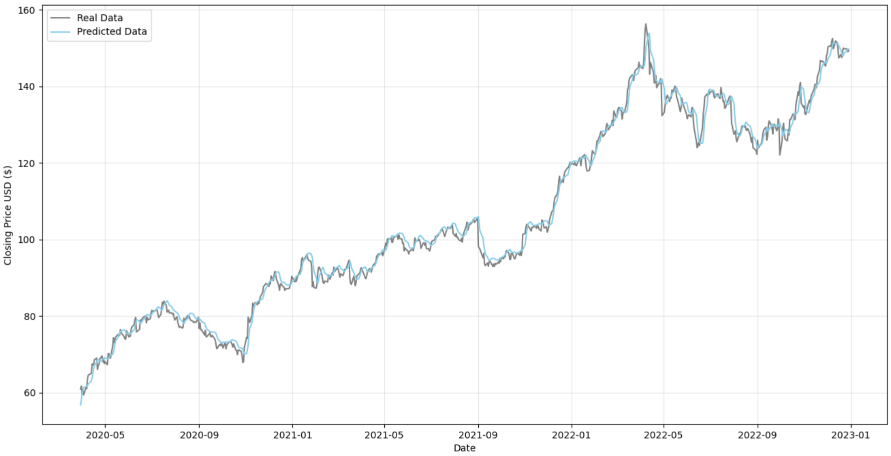


Figure 7: MTL model performance - ABBV

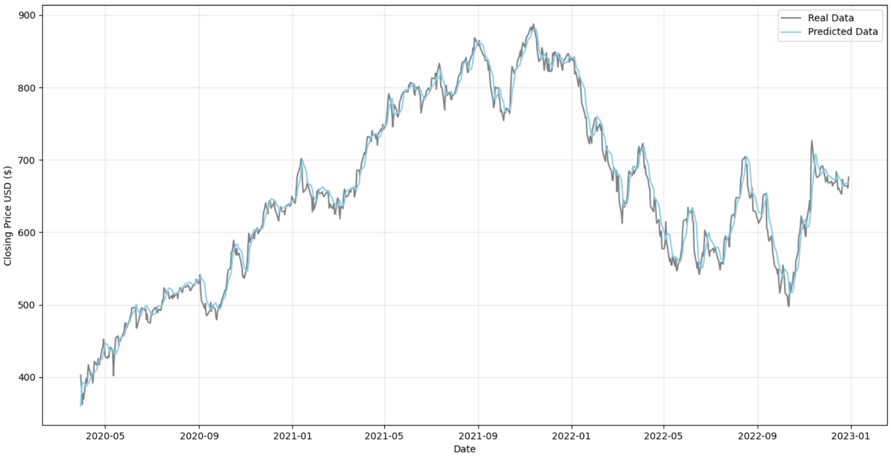


Figure 8: MTL model performance - BLK

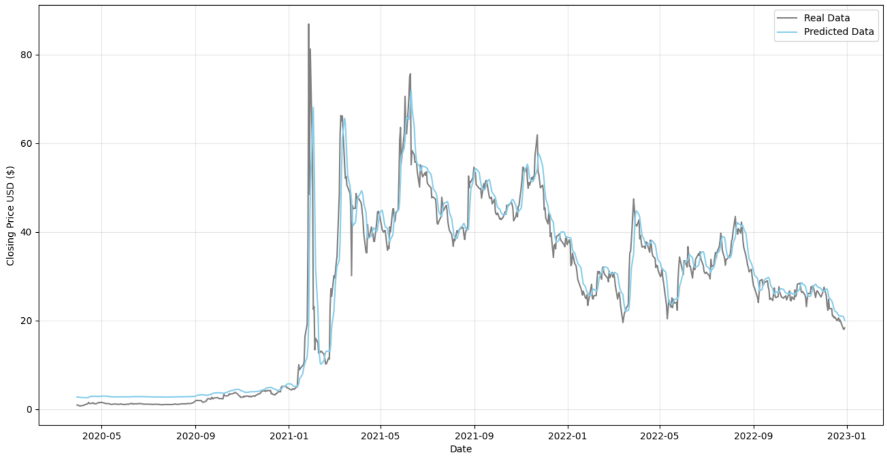


Figure 9: MTL model performance - GME

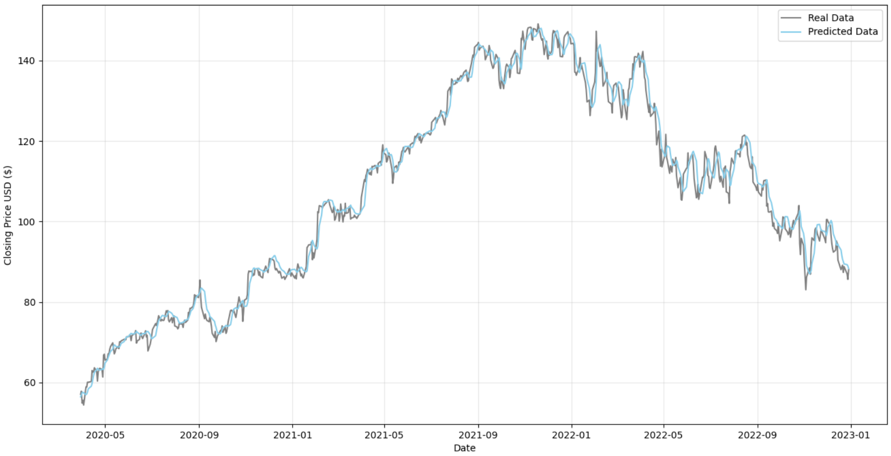


Figure 10: MTL model performance - GOOGL

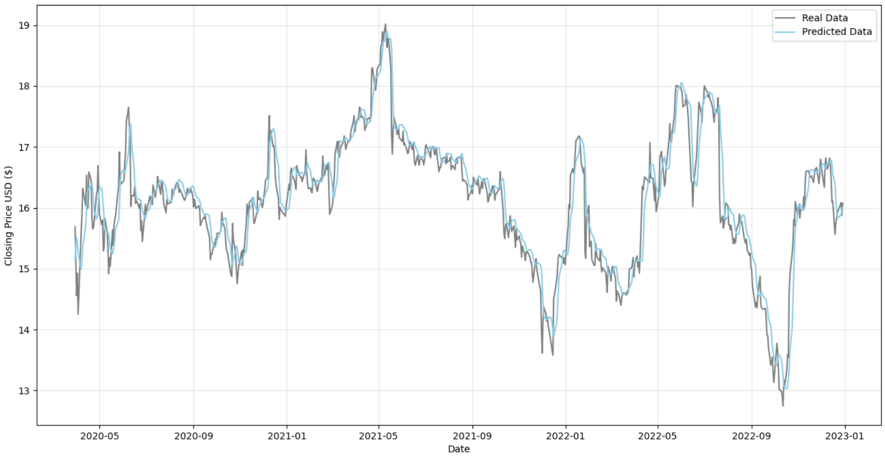


Figure 11: MTL model performance - T

Stock	**Model**	**RMSE (normalized)**	**RMSE (original)**	**MAE (original)**	**MAPE**
ABBV	MTL	0.0227	$2.36	$1.57	1.54%
	IM-t	0.1233	$12.83	$5.88	4.17%
	IM-T	-	-	$3.15	2.99%
BLK	MTL	0.0360	$21.51	$12.20	1.95%
	IM-t	0.1473	$88.10	$18.14	3.01%
	IM-T	-	-	$14.21	2.26%
GME	MTL	0.0208	$1.79	$2.68	34.35%
	IM-t	0.0523	$4.51	$3.21	13.62%
	IM-T	-	-	$2.47	13.36%
GOOGL	MTL	0.0358	$3.46	$2.10	2.02%
	IM-t	0.0849	$8.21	$5.80	6.25%
	IM-T	-	-	$3.57	3.49%
T	MTL	0.0562	$0.42	$0.25	1.57%
	IM-t	0.2555	$1.92	$0.28	1.86%
	IM-T	-	-	$0.23	1.42%

Table 5: Full experiment metrics & results (1)

Stock	**Model**	**Direction accuracy**	**Maximum error**	**Strategy return**	**Buy & Hold return**
ABBV	MTL	50.00%	$9.57	42.82%	144.95%
	IM-t	54.41%	$11.57	5.06%	14.83%
	IM-T	52.00%	$14.72	51.65%	109.30%
BLK	MTL	51.17%	$80.76	49.06%	67.89%
	IM-t	44.12%	$82.90	10.42%	20.08%
	IM-T	55.04%	$67.85	61.16%	41.58%
GME	MTL	50.29%	$67.31	3226.45%	1908.77%
	IM-t	39.71%	$6.22	-18.18%	-25.79%
	IM-T	52.32%	$62.15	3300.35%	2843.56%
GOOGL	MTL	48.56%	$12.69	-39.77%	54.32%
	IM-t	44.12%	$16.13	-13.46%	-11.67%
	IM-T	49.44%	$13.67	19.48%	73.22%
T	MTL	47.55%	$1.56	-11.63%	2.47%
	IM-t	50.00%	$1.26	7.70%	15.63%
	IM-T	48.48%	$1.47	-16.83%	-11.32%

Table 6: Full experiment metrics & results (2)

5 Comparative analysis of all test metrics

Tables 5 and 6 capture the full experiment metrics and results, which we will analyze below. Consider the following notations:

- MTL - Multi-task learning model
- IM-t - individual model - testing set
- IM-T - individual model - training set

We have previously discussed the huge improvement in RMSE when using the MTL approach. Training RMSE for individual models is not of interest, but it can easily be computed with some simple tweaks to the code. Note that for the values expressed in $USD, the price range plays an important role in drawing insights. While BlackRock presents a bigger Mean Absolute Error (MAE), this is due to the fact that its stock valuation sits somewhere between $300 and $900 during the experiment period (see Figure 8). Similarly, there is a huge disparity between GameStock's train and test Maximum Error: $62.15 vs. $6.22. This can be traced to the fact that the individual model's test set excludes most of the high fluctuations and the price spikes (see Figure 4). A more detailed analysis of all metrics follows.

Regarding MAE, we can see the Multi-task learning model performing at least, if not better than the individual model. Note that the metrics computed for the MTL model are done on unseen test data. Knowing this, the model's performance compared to the individual model's performance on its training data is very impressive, with the gap to the individual test set performance being even bigger. This trend can be observed also by analyzing Mean Absolute Percentage Error (MAPE), where the individual model's test performance falls short compared to the training performance and the MTL model's performance. One exception can be spotted for GameStop, where Multi-task learning appears to perform much worse. This is caused by the fact that the entire period is considered as the test set. Because of this, the huge price spike significantly altered the mean of the data, skewing the calculations.

Direction accuracy and Maximum error didn't show any constant trends between the three models, but inferences can definitely be drawn on an extended test set. The GameStop phenomena is also present here. One intriguing observation can be made regarding AT&T, where the individual model made a smaller maximum error compared to the MTL model. Generally, though, the Multi-task learning approach proved to have similar performances to the training set performance.

As expected, a simple trading strategy return, where we buy when the model predicts a price increase and sell when it predicts a fall, proves to work worse than just buying and holding on to the shares. This was expected, since the purpose of our study, and, by extension, our model, was to analyze time-series prediction, not to use it for stock brokerage. Again, due to the high price rise and fluctuations of GameStop's evaluation, the models actually managed to outperform the buy & hold strategy, but this is a mere coincidence and should not be taken as proof of our model's trading capabilities.

6 Conclusions and further developments

In conclusion, this study exemplifies the promising applications of Multi-task learning using LSTM networks for this specific market setting. By using a wide variety of stocks to train a generalized model, overall market trends can be captured, which counteracts the issues that arise when there is limited trading history among newer companies. Hyperparameter optimization also improves prediction accuracy immensely on new, unseen stocks.

Not only do these results validate the proposed Multi-task learning approach's effectiveness in the finance domain, but they also open avenues for the application of this methodology to other domains, including patient monitoring in health care or energy consumption forecasting.

For further developments, the model's architecture can be tweaked further. Using powerful machines, complete case studies can be made on the entire stock market, using all sorts of data and metrics (not just closing price). As a fix to the disrupted temporality of the dataset, we will experiment with approaches leveraging convolutions (CNNs), where an entire day of information is merged into a single data unit.

Acknowledgments: This research was substantially shaped by Associate Professor PhD Bocu Răzvan and Assistant Professor Boca Liana Luminiţa from the Department of Mathematics and Computer Science, Faculty of Mathematics and Computer Science, Transilvania University of Brașov, Romania.

References

[1] F. A. Gers, J. Schmidhuber, & F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10): 2451–2471, 2000.

[2] T. Fischer, & C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2): 654–669, 2018.

[3] Y. Li, W. Zheng, & Z. Zheng. Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access*, 7: 108014–108022, 2019.

[4] Y. Song, J. W. Lee, & J. Lee. A study on novel filtering and relationship between input-features and target-vectors in a deep learning model for stock price prediction. *Applied Intelligence*, 51(2): 885–909, 2021.

[5] M. S. Sonani, A. Badii, & A. Moin. Stock Price Prediction Using a Hybrid LSTM-GNN Model: Integrating Time-Series and Graph-Based Analysis. *arXiv preprint arXiv:2502.15813*, 1-13, 2025.

[6] H. Liu, L. Qi & M. Sun. Short-Term Stock Price Prediction Based on CAE-LSTM Method. *Wireless Communications and Mobile Computing*, 1-7, 2022.

[7] S. Selvin, et al. Stock price prediction using LSTM, RNN and CNN-sliding window model. *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 1643-1647, 2017.

[8] M. A. Istiake Sunny, M. M. S. Maswood & A G. Alharbi, Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model. *2nd Novel Intelligent and Leading Emerging Sciences Conference*, 87-92, 2020.

[9] Y. Jeong, & S. Kim. Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, 117: 125–138, 2019.

[10] S. Merello, et al. Ensemble Application of Transfer Learning and Sample Weighting for Stock Market Prediction. *International Joint Conference on Neural Networks (IJCNN)*, 1-8, 2019.

[11] J. Qiu, B. Wang, & C. Zhou. Forecasting stock prices with long-short term memory neural network based on attention mechanism. *PLoS ONE*, 15(1): e0227222, 2020.

[12] R. Aroussi. yfinance: Yahoo! Finance market data downloader. (last accessed: 25.03.2025). Available at: https://github.com/ranaroussi/yfinance

[13] F. A. Gers, & J. Schmidhuber. LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages. *IEEE Transactions on Neural Networks*, 12(6): 1333–1340, 2001.

[14] X. Shi, et al. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Proceedings of the 28th International Conference on Neural Information Processing Systems*, 802–810, 2015.

[15] F. Karl, et al. Multi-Objective Hyperparameter Optimization in Machine Learning – An Overview. *arXiv preprint arXiv:2403.07438v3*, 4, 2024.

[16] D. P. Kingma, & J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 1, 2014.

Andrei PRIBOI
Transilvania University of Braşov
Faculty of Mathematics and Computer Science
No. 50, Iuliu Maniu st., Brașov, Romania
ROMANIA
E-mail: *andrei.priboi@student.unitbv.ro*

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Game Engine Development: Research, optimization and performance enhancement

Serkan Sadulov, Mustafa Mustafov

Abstract

This paper reviews the process of developing a game engine and explores the methodologies of development, best practices, and challenges involved in creating such software. This study provides an in-depth analysis of the tools, frameworks, and design patterns used for development, along with real-world examples of code and assets utilized throughout the process. Special attention is given to performance considerations, rendering techniques, physics simulations, and scripting integration to ensure a comprehensive understanding of the development workflow. Furthermore, this research presents a hands-on demonstration of the final product, showcasing the engine's capabilities and potential applications for game developers

1 Introduction

The game engine serves as a platform for developing video games, creating animated films, and running complex simulations. It integrates a wide range of essential functionalities including real-time graphics rendering, audio processing, physics simulation, artificial intelligence, and networking. Over the years, numerous game engines and development tools have emerged, such as Unity, Unreal Engine, Godot, and various proprietary engines developed by major studios. Although these engines offer powerful features and have become industry standards, they still have limitations in terms of optimization, flexibility, and high resource consumption. Recognizing these shortcomings, the decision was made to develop a new and improved game engine that not only enhances performance, but also pushes the boundaries of what is possible in game development.

Without a game engine, developers would have to write complex low-level codes that are not only time-consuming but also inefficient. Using a game engine, developers can focus on designing gameplay mechanics, storytelling, and immersive worlds without reinventing core functionalities that have already been optimized by engine developers.

2 Fundamentals of Game Engine Development

2.1 Overview of game engine architecture

Our game engine was developed using C++20, which leverages modern programming techniques to ensure high performance, maintainability, and scalability. It is designed with Vulkan as its primary rendering backend, taking advantage of low-level access to GPU resources for optimal rendering efficiency. However, the engine is built with a fully modular architecture, allowing additional rendering options such as DirectX to be integrated seamlessly without significant rework.

The modular design extends beyond rendering; a core component of the engine, from physics to audio, is structured as an independent module. Each module was compiled separately and then integrated into the core framework to ensure flexibility, maintainability, and ease of extension. This approach allows for the independent development, testing, and optimization of each subsystem without affecting the entire engine.

Furthermore, engine modules are interconnected, enabling seamless communication between different systems. For instance, the 2D physics module directly interacts with the rendering system to display the debug collider bounds, ensuring that developers have clear visual feedback when working with physics simulations. This modular architecture provides a solid foundation for a powerful and extensible game engine that can adapt to various needs and technologies with minimal effort [1].

2.1.1 Key technologies used in our Game Engine

To develop our game engine, we carefully select a range of key technologies that provide optimal performance, flexibility, and scalability. Each component was chosen to ensure the highest level of efficiency and ease of integration, while maintaining a modular and extensible architecture. Below is a detailed breakdown of the core technologies used in our engine [5]:

- **C++20 as the Main Programming Language**: Our engine was built using C++20, taking advantage of its modern features, such as concepts, ranges, coroutines, and improved memory management. Language’s high-performance nature and fine-grained control over system resources make it an ideal choice for developing a game engine. By utilizing C++20, we ensure that our engine remains efficient, scalable, and capable of handling complex real-time computations with minimal overheads.
- **Vulkan for Rendering**: Vulkan serves as the primary rendering API for our engine, providing low-level access to GPU resources and enabling fine-tuned optimization for high-performance graphics. Unlike traditional APIs such as OpenGL, Vulkan allows for better multithreading, reduced CPU overhead, and explicit memory management, resulting in superior rendering efficiency. However, owing to our modular design, alternative rendering backends, such as DirectX, can be integrated with minimal effort if needed [3].
- **Box2D for 2D Physics Simulation**: To handle 2D physics, we integrated Box2D, a lightweight yet powerful physics engine known for its accuracy and reliability. Box2D is widely used in the gaming industry because of its robust collision detection and realistic physics simulations. It provides essential features, such as rigid body dynamics, contact resolution, and joints, ensuring smooth and natural interactions for 2D game elements.
- **Jolt Physics for 3D Physics Simulation**: For 3D physics, our engine utilizes Jolt Physics, which is a high-performance physics library designed for real-time applications. Jolt offers efficient and stable physical simulations, including rigid-body dynamics, soft-body simulations, and character controllers. It is optimized for modern hardware, ensuring a smooth performance even in complex scenes with a large number of objects. By leveraging Jolt, our engine delivers realistic and responsive physical interactions in three-dimensional (3D) environments.
- **MiniAudio for Audio Processing**: Our engine’s audio system is powered by MiniAudio, a lightweight cross-platform audio library that supports multiple formats and playback features. MiniAudio provides a simple yet powerful API for handling 3D spatial audio, sound effects, and music playback, with minimal latency. Its low-level design allows us to integrate it tightly into our engine while maintaining full control over audio processing and performance optimization.

- **ImGui for User Interface (UI)**: For the graphical user interface of the engine, we integrated ImGui (Immediate Mode GUI), a fast and flexible UI library designed for real-time applications. ImGui enables rapid prototyping and tool development, and provides an intuitive and highly responsive interface for debugging, in-engine editors, and developer tools. Its lightweight nature ensures minimal impact on performance while allowing the creation of powerful and user-friendly UI elements [6].
- **Custom Reflection Framework (Reflexion) for Scripting**: Instead of relying on external scripting languages, such as Lua or Python, we have developed our own custom reflection framework, Reflexion, which allows us to use C++ itself as a scripting language. Reflexion enables runtime type introspection, dynamic property modification, and serialization, providing an efficient way to expose engine functionalities to scripts without compromising performance. This approach ensures seamless integration with the existing C++ code, while offering a robust scripting solution tailored to our engine's needs.
- **EASTL as a Replacement for STL**: To improve performance and memory efficiency, we replaced the standard C++ STL (Standard Template Library) with EASTL (Electronic Arts Standard Template Library). EASTL was optimized for game development, offering faster container operations, better memory management, and reduced heap fragmentation. By using EASTL, we ensure that our engine benefits from optimized data structures that enhance overall performance, particularly in real-time applications.
- **Google Test (GTest) for Unit Testing**: Ensuring the reliability and stability of our engine is crucial, which is why we utilized Google Test (GTest) as our primary testing framework. GTest provides a robust and flexible environment for writing and executing unit tests, allowing us to validate individual engine components and detect issues early in development. With automated testing, we can maintain a high code quality and ensure that each system behaves as expected under different scenarios.

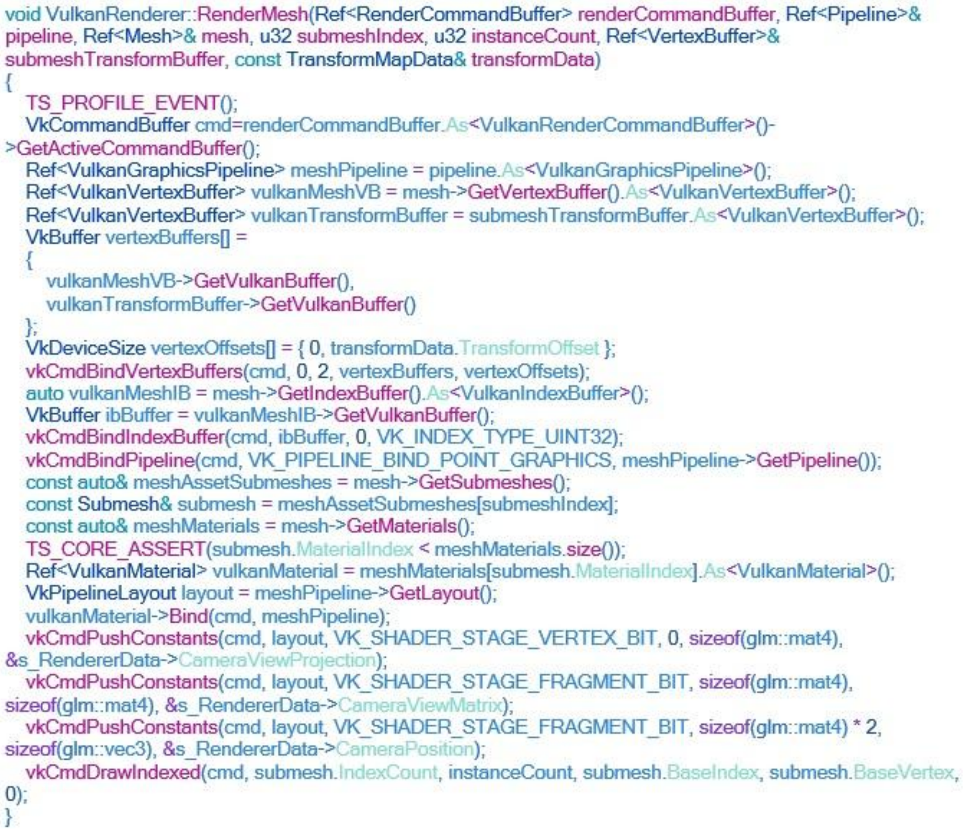
3 Core Systems of a Game Engine

3.1 Rendering System

The rendering system is a core component of any game engine and is responsible for converting the game world's data into images that are displayed on a screen. It involves several stages, from loading assets to applying shaders and lighting, and finally, rendering the scene in real time [7].

3.1.1 Rendering Pipelines

The rendering pipeline is a sequence of stages **(Fig. 2),** through which the engine processes graphical data to produce the final image **(Fig. 1).**



```
void VulkanRenderer::RenderMesh(Ref<RenderCommandBuffer> renderCommandBuffer, Ref<Pipeline>&
pipeline, Ref<Mesh>& mesh, u32 submeshIndex, u32 instanceCount, Ref<VertexBuffer>&
submeshTransformBuffer, const TransformMapData& transformData)
{
   TS_PROFILE_EVENT();
   VkCommandBuffer cmd=renderCommandBuffer.As<VulkanRenderCommandBuffer>()-
>GetActiveCommandBuffer();
   Ref<VulkanGraphicsPipeline> meshPipeline = pipeline.As<VulkanGraphicsPipeline>();
   Ref<VulkanVertexBuffer> vulkanMeshVB = mesh->GetVertexBuffer().As<VulkanVertexBuffer>();
   Ref<VulkanVertexBuffer> vulkanTransformBuffer = submeshTransformBuffer.As<VulkanVertexBuffer>();
   VkBuffer vertexBuffers[] =
   {
      vulkanMeshVB->GetVulkanBuffer(),
      vulkanTransformBuffer->GetVulkanBuffer()
   };
   VkDeviceSize vertexOffsets[] = { 0, transformData.TransformOffset };
   vkCmdBindVertexBuffers(cmd, 0, 2, vertexBuffers, vertexOffsets);
   auto vulkanMeshIB = mesh->GetIndexBuffer().As<VulkanIndexBuffer>();
   VkBuffer ibBuffer = vulkanMeshIB->GetVulkanBuffer();
   vkCmdBindIndexBuffer(cmd, ibBuffer, 0, VK_INDEX_TYPE_UINT32);
   vkCmdBindPipeline(cmd, VK_PIPELINE_BIND_POINT_GRAPHICS, meshPipeline->GetPipeline());
   const auto& meshAssetSubmeshes = mesh->GetSubmeshes();
   const Submesh& submesh = meshAssetSubmeshes[submeshIndex];
   const auto& meshMaterials = mesh->GetMaterials();
   TS_CORE_ASSERT(submesh.MaterialIndex < meshMaterials.size());
   Ref<VulkanMaterial> vulkanMaterial = meshMaterials[submesh.MaterialIndex].As<VulkanMaterial>();
   VkPipelineLayout layout = meshPipeline->GetLayout();
   vulkanMaterial->Bind(cmd, meshPipeline);
   vkCmdPushConstants(cmd, layout, VK_SHADER_STAGE_VERTEX_BIT, 0, sizeof(glm::mat4),
&s_RendererData->CameraViewProjection);
   vkCmdPushConstants(cmd, layout, VK_SHADER_STAGE_FRAGMENT_BIT, sizeof(glm::mat4),
sizeof(glm::mat4), &s_RendererData->CameraViewMatrix);
   vkCmdPushConstants(cmd, layout, VK_SHADER_STAGE_FRAGMENT_BIT, sizeof(glm::mat4) * 2,
sizeof(glm::vec3), &s_RendererData->CameraPosition);
   vkCmdDrawIndexed(cmd, submesh.IndexCount, instanceCount, submesh.BaseIndex, submesh.BaseVertex,
0);
}
```

Figure 1: *Partial mesh rendering code*

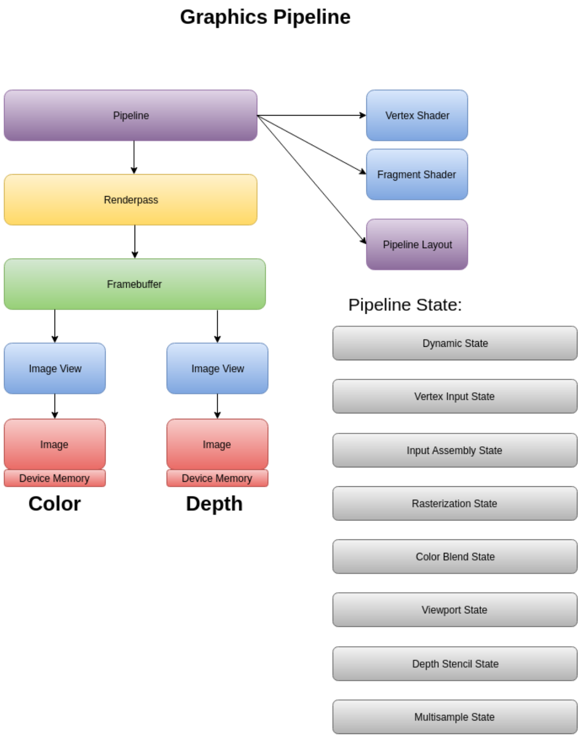


Figure 2: *Graphics pipline scheme*

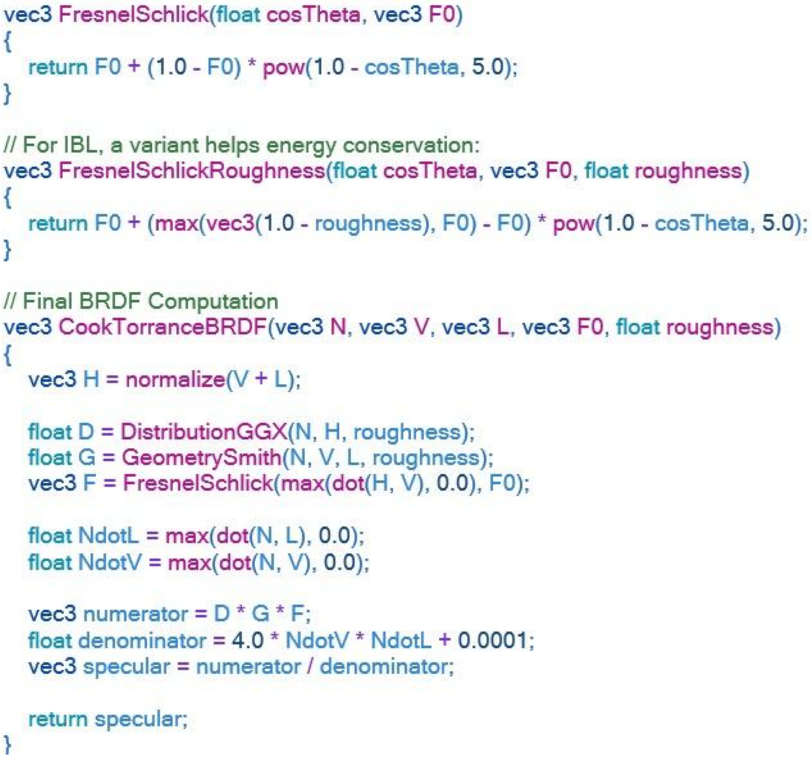
3.1.2 Real-Time Rendering Techniques

Real-time rendering aims to produce visually rich images at high frame rates, which is critical for interactive applications, such as video games. Several advanced techniques have been used to achieve high-quality real-time graphics [8].

3.1.2.1 Physically Based Rendering (PBR)

PBR is a shading model that simulates the interaction of light with surfaces in a way that more accurately represents real-world materials. It models how light interacts with roughness, reflectivity, and other surface properties to produce realistic results. PBR uses the principles of energy conservation, ensuring that materials behave consistently under different lighting conditions. It has become a standard technique in modern game engines due to its ability to produce photorealistic visuals across different lighting environments and hardware configurations. The key components of PBR include **(Fig. 3)**:

- Albedo (Diffuse): The color of the material.
- Metallic: A value indicating whether the material is metallic or not.
- Roughness: The smoothness or roughness of the surface affects light scattering.
- Normal Maps: Used to add fine details to the surface without adding extra polygons.



```
vec3 FresnelSchlick(float cosTheta, vec3 F0)
{
   return F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
}

// For IBL, a variant helps energy conservation:
vec3 FresnelSchlickRoughness(float cosTheta, vec3 F0, float roughness)
{
   return F0 + (max(vec3(1.0 - roughness), F0) - F0) * pow(1.0 - cosTheta, 5.0);
}

// Final BRDF Computation
vec3 CookTorranceBRDF(vec3 N, vec3 V, vec3 L, vec3 F0, float roughness)
{
   vec3 H = normalize(V + L);

   float D = DistributionGGX(N, H, roughness);
   float G = GeometrySmith(N, V, L, roughness);
   vec3 F = FresnelSchlick(max(dot(H, V), 0.0), F0);

   float NdotL = max(dot(N, L), 0.0);
   float NdotV = max(dot(N, V), 0.0);

   vec3 numerator = D * G * F;
   float denominator = 4.0 * NdotV * NdotL + 0.0001;
   vec3 specular = numerator / denominator;

   return specular;
}
```

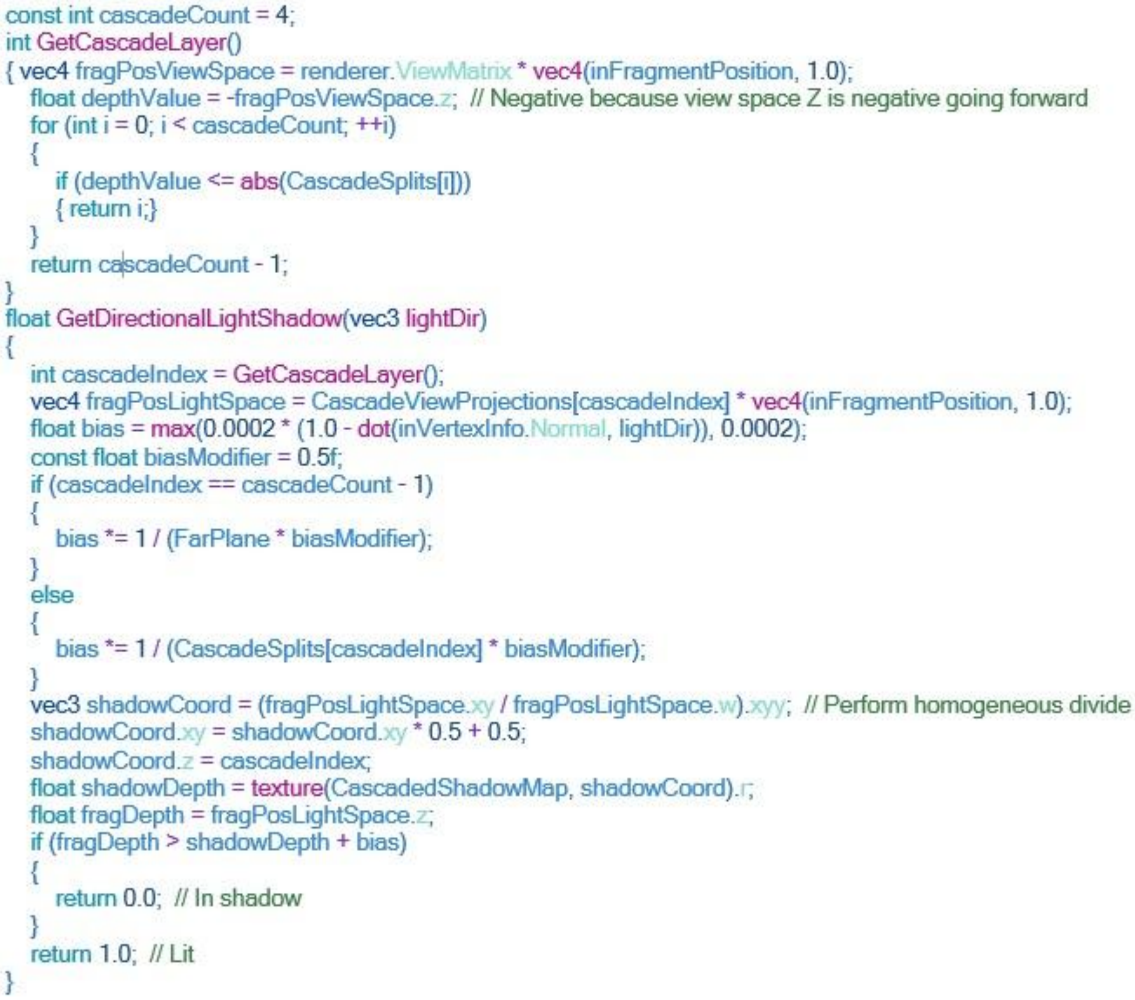
Figure 3: *Partial code of PBR pipeline*

3.1.2.2 Shadows

Shadows are crucial for adding depth and realism to scenes. In real-time rendering, in our engine, we used cascade shadow mapping (CSM), Spotlight Shadow Mapping and Point Light Shadow Mapping ***(Fig. 4)***:

- **Cascaded Shadow Maps (CSM)** is the variation in the shadow mapping used for large outdoor environments. It uses multiple shadow maps for different view distances, ensuring that shadows are rendered with high detail close to the camera and less detail farther away.

- **Spotlight shadow mapping** was designed for light with a conical shape that emits light in a specific direction, similar to a real-world spotlight. This technique involves rendering the scene from the perspective of the spotlight's view, which allows the shadow map to capture only the area within the spotlight's cone of influence.
- **Point Light Shadow Mapping** used for light that emits light in all directions, similar to a light bulb. This type of lighting is ideal for simulating omnidirectional light sources that affect an entire environment.



```
const int cascadeCount = 4;
int GetCascadeLayer()
{ vec4 fragPosViewSpace = renderer.ViewMatrix * vec4(inFragmentPosition, 1.0);
  float depthValue = -fragPosViewSpace.z;  // Negative because view space Z is negative going forward
  for (int i = 0; i < cascadeCount; ++i)
  {
    if (depthValue <= abs(CascadeSplits[i]))
    { return i;}
  }
  return cascadeCount - 1;
}
float GetDirectionalLightShadow(vec3 lightDir)
{
  int cascadeIndex = GetCascadeLayer();
  vec4 fragPosLightSpace = CascadeViewProjections[cascadeIndex] * vec4(inFragmentPosition, 1.0);
  float bias = max(0.0002 * (1.0 - dot(inVertexInfo.Normal, lightDir)), 0.0002);
  const float biasModifier = 0.5f;
  if (cascadeIndex == cascadeCount - 1)
  {
    bias *= 1 / (FarPlane * biasModifier);
  }
  else
  {
    bias *= 1 / (CascadeSplits[cascadeIndex] * biasModifier);
  }
  vec3 shadowCoord = (fragPosLightSpace.xy / fragPosLightSpace.w).xyy;  // Perform homogeneous divide
  shadowCoord.xy = shadowCoord.xy * 0.5 + 0.5;
  shadowCoord.z = cascadeIndex;
  float shadowDepth = texture(CascadedShadowMap, shadowCoord).r;
  float fragDepth = fragPosLightSpace.z;
  if (fragDepth > shadowDepth + bias)
  {
    return 0.0;  // In shadow
  }
  return 1.0;  // Lit
}
```

Figure 4: *Code of shadow calculation of diractional light, without filltering*

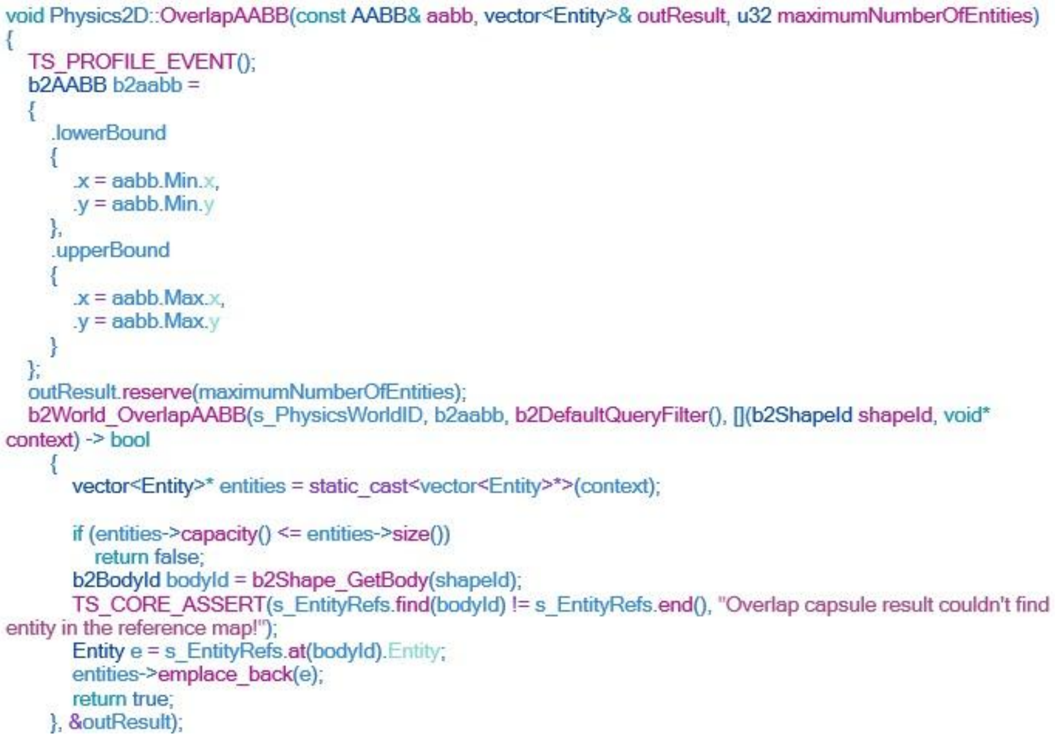
3.2 Physics and Collision Detection

Physical simulation plays a crucial role in modern games, enhancing realism by accurately simulating the physical interactions between objects in the game world. From character movement and gravity to object collisions and destructible environments, a robust physics engine ensures that these interactions behave in a manner that is both believable and responsive to the player. In this section, we explore the physics simulation techniques used in our engine, including the collision-detection algorithms and the decision to integrate Box2D as our primary 2D physics engine [2].

3.2.1 Collision detection algorithms

Collision detection is one of the core components of physics simulations, as it allows objects to interact with one another in a meaningful way. The main goal of collision detection is to determine whether two objects intersect or come into contact with each other, and if so, to calculate the response [4].

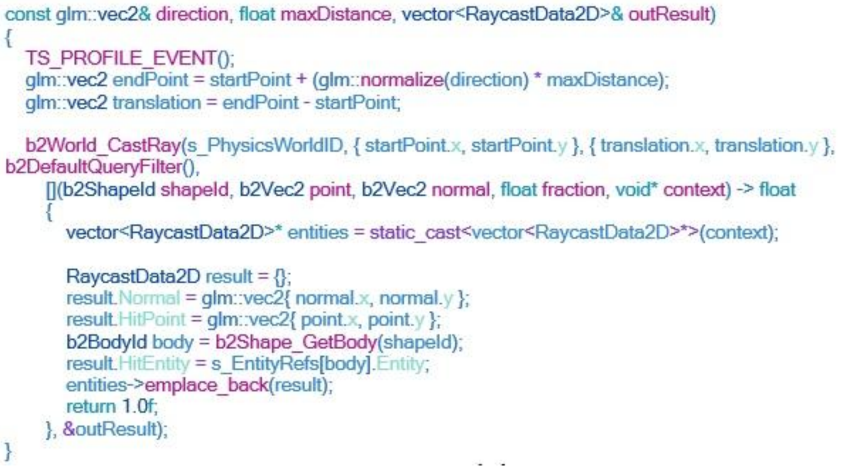
- **AABB (Axis-Aligned Bounding Box)** is a simple and efficient collision detection algorithm used to detect potential overlaps between rectangular objects in 2D. Each object is enclosed by an axis-aligned box, which is easier to compute than general bounding volumes. AABB checks if the boxes of two objects intersect by comparing their minimum and maximum coordinates along the X and Y axes. This approach is fast and commonly used for broad-phase collision detection to filter out objects that are far apart and cannot possibly collide ***(Fig. 5)***.



```
void Physics2D::OverlapAABB(const AABB& aabb, vector<Entity>& outResult, u32 maximumNumberOfEntities)
{
   TS_PROFILE_EVENT();
   b2AABB b2aabb =
   {
      .lowerBound
      {
         .x = aabb.Min.x,
         .y = aabb.Min.y
      },
      .upperBound
      {
         .x = aabb.Max.x,
         .y = aabb.Max.y
      }
   };
   outResult.reserve(maximumNumberOfEntities);
   b2World_OverlapAABB(s_PhysicsWorldID, b2aabb, b2DefaultQueryFilter(), [](b2ShapeId shapeId, void*
context) -> bool
      {
         vector<Entity>* entities = static_cast<vector<Entity>*>(context);

         if (entities->capacity() <= entities->size())
            return false;
         b2BodyId bodyId = b2Shape_GetBody(shapeId);
         TS_CORE_ASSERT(s_EntityRefs.find(bodyId) != s_EntityRefs.end(), "Overlap capsule result couldn't find
entity in the reference map!");
         Entity e = s_EntityRefs.at(bodyId).Entity;
         entities->emplace_back(e);
         return true;
      }, &outResult);
```

Figure 5: *Code of AABB overlap*

- **Raycasting** is another essential algorithm used for detecting collisions along a path, often in the context of shooting mechanics or line-of-sight detection. A ray is projected from a starting point in a specific direction and the engine checks for intersections with objects along the path of the ray. Raycasting is especially useful for detecting collisions with walls or obstacles in first-person games or determining where objects in the environment intersect with player actions ***(Fig. 6)***.

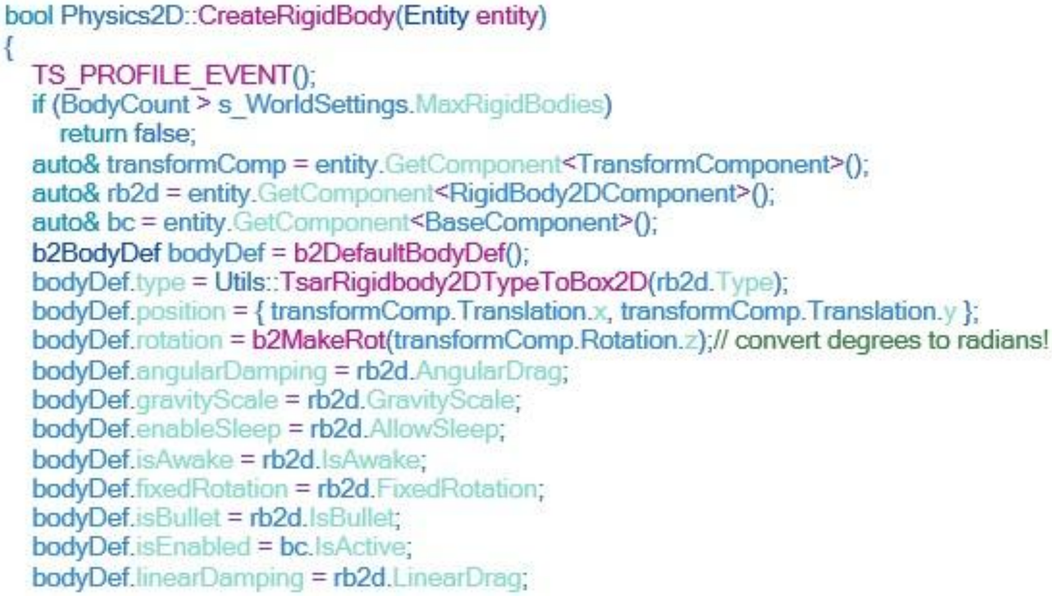


```
const glm::vec2& direction, float maxDistance, vector<RaycastData2D>& outResult)
{
   TS_PROFILE_EVENT();
   glm::vec2 endPoint = startPoint + (glm::normalize(direction) * maxDistance);
   glm::vec2 translation = endPoint - startPoint;

   b2World_CastRay(s_PhysicsWorldID, { startPoint.x, startPoint.y }, { translation.x, translation.y },
b2DefaultQueryFilter(),
      [](b2ShapeId shapeId, b2Vec2 point, b2Vec2 normal, float fraction, void* context) -> float
      {
         vector<RaycastData2D>* entities = static_cast<vector<RaycastData2D>*>(context);

         RaycastData2D result = {};
         result.Normal = glm::vec2{ normal.x, normal.y };
         result.HitPoint = glm::vec2{ point.x, point.y };
         b2BodyId body = b2Shape_GetBody(shapeId);
         result.HitEntity = s_EntityRefs[body].Entity;
         entities->emplace_back(result);
         return 1.0f;
      }, &outResult);
}
```

Figure 6: *Code for Raycasting*

3.2.2 Box2D for 2D Physics

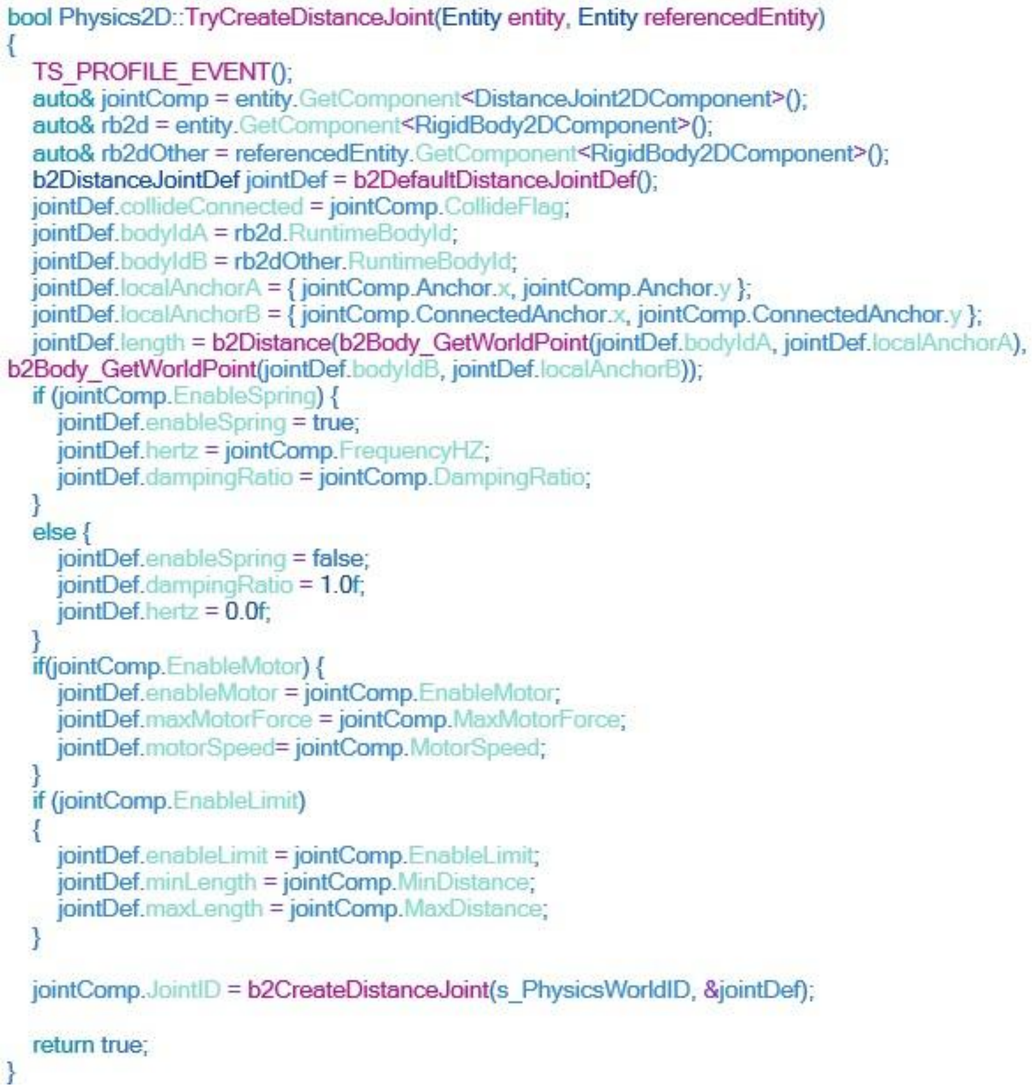
Rigid Body Simulation: Box2D accurately simulates the motion of rigid bodies in response to forces such as gravity, friction, and user input. It supports both dynamic bodies, which move and interact with other objects, and static bodies, which are immovable but can collide with dynamic objects ***(Fig. 7)***.



```
bool Physics2D::CreateRigidBody(Entity entity)
{
   TS_PROFILE_EVENT();
   if (BodyCount > s_WorldSettings.MaxRigidBodies)
      return false;
   auto& transformComp = entity.GetComponent<TransformComponent>();
   auto& rb2d = entity.GetComponent<RigidBody2DComponent>();
   auto& bc = entity.GetComponent<BaseComponent>();
   b2BodyDef bodyDef = b2DefaultBodyDef();
   bodyDef.type = Utils::TsarRigidbody2DTypeToBox2D(rb2d.Type);
   bodyDef.position = { transformComp.Translation.x, transformComp.Translation.y };
   bodyDef.rotation = b2MakeRot(transformComp.Rotation.z);// convert degrees to radians!
   bodyDef.angularDamping = rb2d.AngularDrag;
   bodyDef.gravityScale = rb2d.GravityScale;
   bodyDef.enableSleep = rb2d.AllowSleep;
   bodyDef.isAwake = rb2d.IsAwake;
   bodyDef.fixedRotation = rb2d.FixedRotation;
   bodyDef.isBullet = rb2d.IsBullet;
   bodyDef.isEnabled = bc.IsActive;
   bodyDef.linearDamping = rb2d.LinearDrag;
```

Figure 7: *Example code of creating a rigid body*

Joints: Box2D offers a wide variety of joint types, including revolute joints (like hinges), prismatic joints (like sliders), and distance joints, enabling complex interactions between objects. This makes it ideal for simulating various mechanical systems, such as vehicles, ropes, or even character animation through physics-based rigging ***(Fig. 8).***

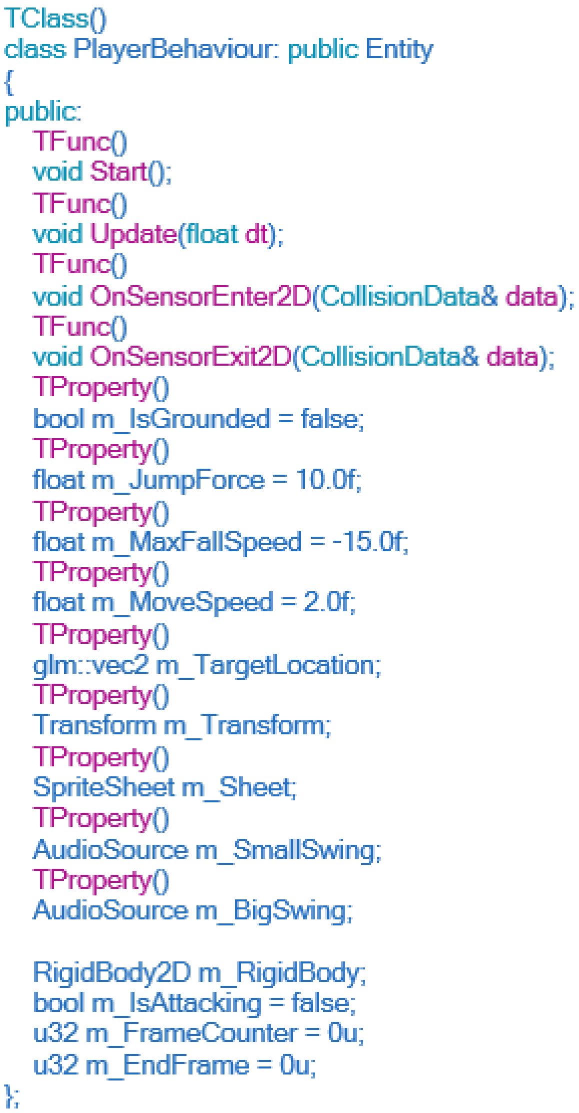


```
bool Physics2D::TryCreateDistanceJoint(Entity entity, Entity referencedEntity)
{
    TS_PROFILE_EVENT();
    auto& jointComp = entity.GetComponent<DistanceJoint2DComponent>();
    auto& rb2d = entity.GetComponent<RigidBody2DComponent>();
    auto& rb2dOther = referencedEntity.GetComponent<RigidBody2DComponent>();
    b2DistanceJointDef jointDef = b2DefaultDistanceJointDef();
    jointDef.collideConnected = jointComp.CollideFlag;
    jointDef.bodyIdA = rb2d.RuntimeBodyId;
    jointDef.bodyIdB = rb2dOther.RuntimeBodyId;
    jointDef.localAnchorA = { jointComp.Anchor.x, jointComp.Anchor.y };
    jointDef.localAnchorB = { jointComp.ConnectedAnchor.x, jointComp.ConnectedAnchor.y };
    jointDef.length = b2Distance(b2Body_GetWorldPoint(jointDef.bodyIdA, jointDef.localAnchorA),
b2Body_GetWorldPoint(jointDef.bodyIdB, jointDef.localAnchorB));
    if (jointComp.EnableSpring) {
        jointDef.enableSpring = true;
        jointDef.hertz = jointComp.FrequencyHZ;
        jointDef.dampingRatio = jointComp.DampingRatio;
    }
    else {
        jointDef.enableSpring = false;
        jointDef.dampingRatio = 1.0f;
        jointDef.hertz = 0.0f;
    }
    if(jointComp.EnableMotor) {
        jointDef.enableMotor = jointComp.EnableMotor;
        jointDef.maxMotorForce = jointComp.MaxMotorForce;
        jointDef.motorSpeed= jointComp.MotorSpeed;
    }
    if (jointComp.EnableLimit)
    {
        jointDef.enableLimit = jointComp.EnableLimit;
        jointDef.minLength = jointComp.MinDistance;
        jointDef.maxLength = jointComp.MaxDistance;
    }

    jointComp.JointID = b2CreateDistanceJoint(s_PhysicsWorldID, &jointDef);

    return true;
}
```

Figure 8: *Code of distance joint*

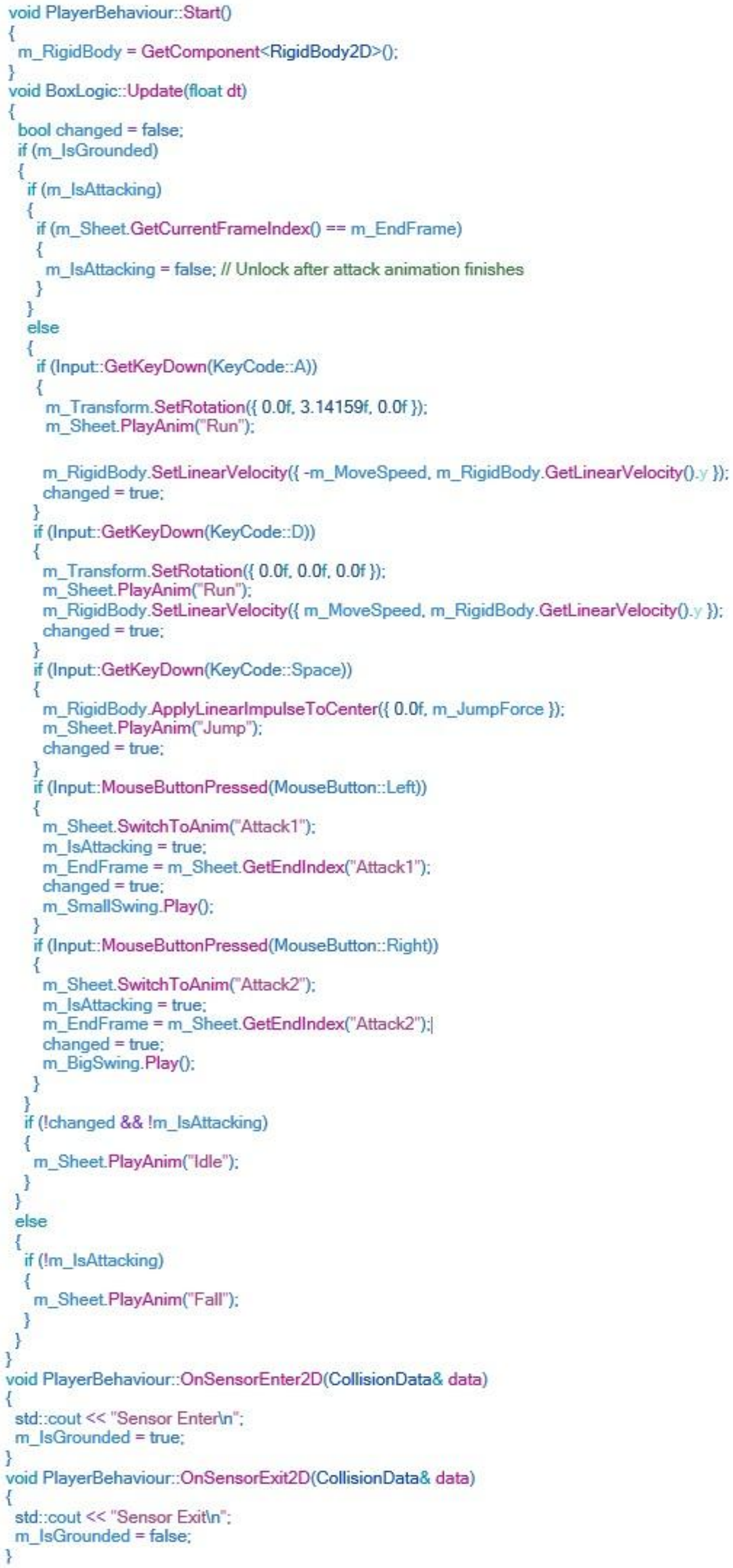
3.3 Scripting System

A scripting system is a vital component of any modern game engine, allowing developers to implement game logic, AI behavior, and interactive elements without modifying the core engine code. For our engine, we designed a custom reflection-based scripting system called Reflexion, which allows C++ to be used as scripting language. This system enables game developers to write high-level game logic in a dynamic manner while still benefiting from the raw power and performance of C++ ***(Fig. 9)***. Reflexion provides an abstraction layer for scripting, making it easier to modify gameplay mechanics without modifying the engine core ***(Fig. 10)***.



```
TClass()
class PlayerBehaviour: public Entity
{
public:
    TFunc()
    void Start();
    TFunc()
    void Update(float dt);
    TFunc()
    void OnSensorEnter2D(CollisionData& data);
    TFunc()
    void OnSensorExit2D(CollisionData& data);
    TProperty()
    bool m_IsGrounded = false;
    TProperty()
    float m_JumpForce = 10.0f;
    TProperty()
    float m_MaxFallSpeed = -15.0f;
    TProperty()
    float m_MoveSpeed = 2.0f;
    TProperty()
    glm::vec2 m_TargetLocation;
    TProperty()
    Transform m_Transform;
    TProperty()
    SpriteSheet m_Sheet;
    TProperty()
    AudioSource m_SmallSwing;
    TProperty()
    AudioSource m_BigSwing;

    RigidBody2D m_RigidBody;
    bool m_IsAttacking = false;
    u32 m_FrameCounter = 0u;
    u32 m_EndFrame = 0u;
};
```

Figure 9: *Scripting code header*



```
void PlayerBehaviour::Start()
{
 m_RigidBody = GetComponent<RigidBody2D>();
}
void BoxLogic::Update(float dt)
{
 bool changed = false;
 if (m_IsGrounded)
 {
  if (m_IsAttacking)
  {
   if (m_Sheet.GetCurrentFrameIndex() == m_EndFrame)
   {
    m_IsAttacking = false; // Unlock after attack animation finishes
   }
  }
  else
  {
   if (Input::GetKeyDown(KeyCode::A))
   {
    m_Transform.SetRotation({ 0.0f, 3.14159f, 0.0f });
    m_Sheet.PlayAnim("Run");

    m_RigidBody.SetLinearVelocity({ -m_MoveSpeed, m_RigidBody.GetLinearVelocity().y });
    changed = true;
   }
   if (Input::GetKeyDown(KeyCode::D))
   {
    m_Transform.SetRotation({ 0.0f, 0.0f, 0.0f });
    m_Sheet.PlayAnim("Run");
    m_RigidBody.SetLinearVelocity({ m_MoveSpeed, m_RigidBody.GetLinearVelocity().y });
    changed = true;
   }
   if (Input::GetKeyDown(KeyCode::Space))
   {
    m_RigidBody.ApplyLinearImpulseToCenter({ 0.0f, m_JumpForce });
    m_Sheet.PlayAnim("Jump");
    changed = true;
   }
   if (Input::MouseButtonPressed(MouseButton::Left))
   {
    m_Sheet.SwitchToAnim("Attack1");
    m_IsAttacking = true;
    m_EndFrame = m_Sheet.GetEndIndex("Attack1");
    changed = true;
    m_SmallSwing.Play();
   }
   if (Input::MouseButtonPressed(MouseButton::Right))
   {
    m_Sheet.SwitchToAnim("Attack2");
    m_IsAttacking = true;
    m_EndFrame = m_Sheet.GetEndIndex("Attack2");
    changed = true;
    m_BigSwing.Play();
   }
  }
  if (!changed && !m_IsAttacking)
  {
   m_Sheet.PlayAnim("Idle");
  }
 }
 else
 {
  if (!m_IsAttacking)
  {
   m_Sheet.PlayAnim("Fall");
  }
 }
}
void PlayerBehaviour::OnSensorEnter2D(CollisionData& data)
{
 std::cout << "Sensor Enter\n";
 m_IsGrounded = true;
}
void PlayerBehaviour::OnSensorExit2D(CollisionData& data)
{
 std::cout << "Sensor Exit\n";
 m_IsGrounded = false;
}
```

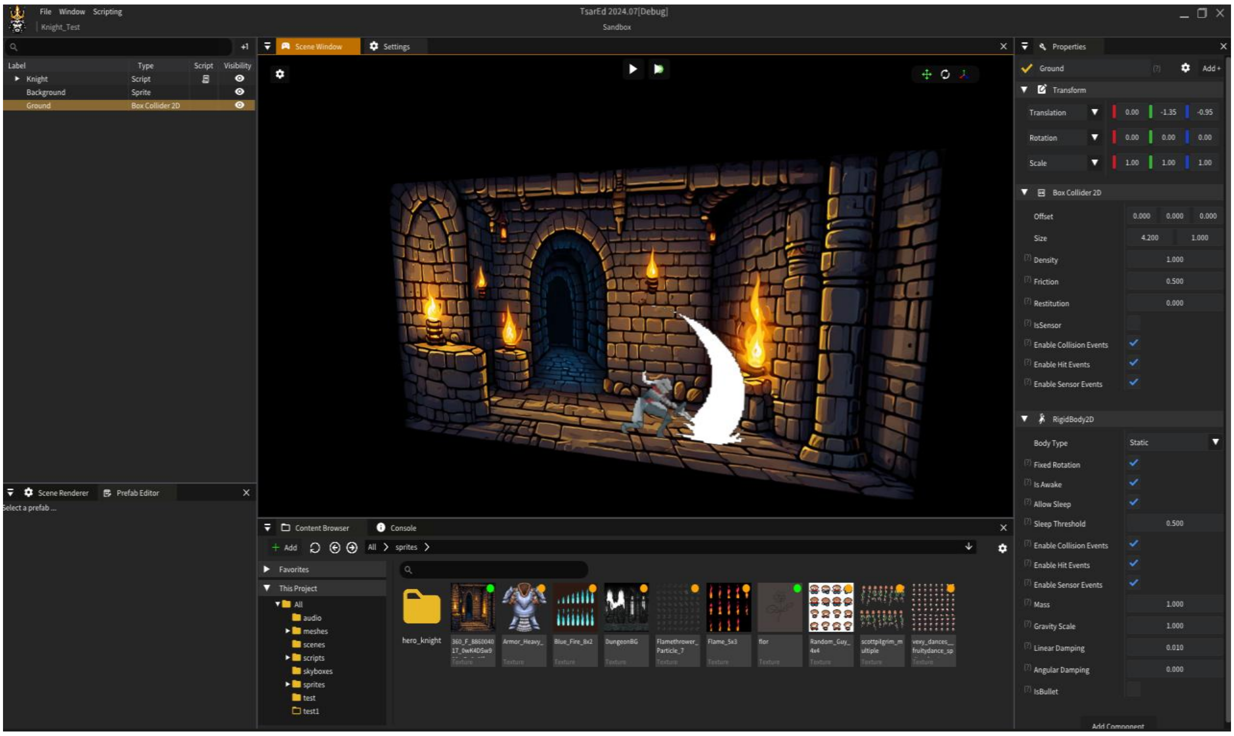
Figure 10: *Custum scripting example for player behaviour*

3.4 Entity-Component System (ECS)

ECS (Entity Component System) is a data-oriented design which binds together everything in the game engine. It provides precise control over every game object and allows for the creation of scalable systems owing to its high performance. Each object in the game world, known as the Game Object, is referred to as an entity. An Entity can contain multiple components. A Component is a structure that holds specific information, such as ***image*** and ***sound***. This information is processed by the systems to realize its intended functionality. A System is essentially a process that acts on the data for specific components. For example, the Physics system acts on all Entities with RigidbodyComponent by calculating their mass, velocity and then applying a change in TransformComponent based on the results.

3.5 User Interface (UI)

A well-designed User Interface (UI) is essential for modern game engines, as it provides developers with an intuitive way to interact with tools, debug features, and configure game settings. In our game engine, we used Dear ImGui, a highly efficient immediate-mode GUI library, to create a responsive and easy-to-use interface for the engine tools ***(Fig. 11)***.



Label	Type	Script	Visibility
Knight	Script		
Background	Sprite		
Ground	Box Collider 2D		

☑ Ground

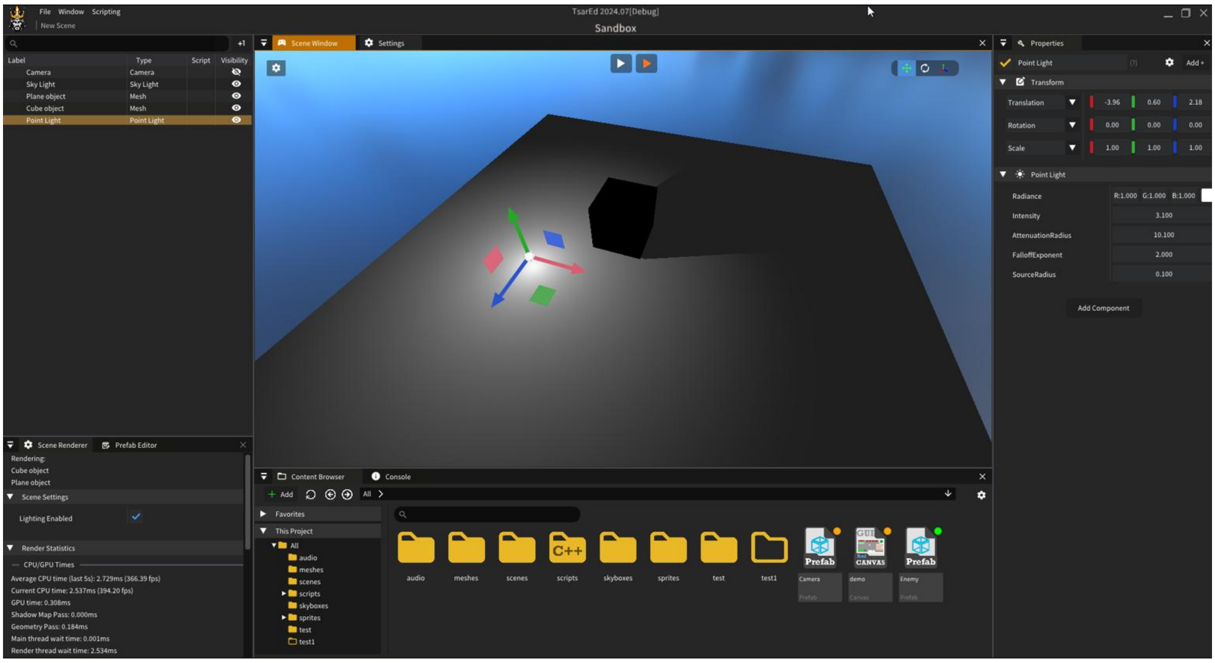
Transform
Translation 0.00 -1.35 -0.95
Rotation 0.00 0.00 0.00
Scale 1.00 1.00 1.00

Box Collider 2D
Offset 0.000 0.000 0.000
Size 4.200 1.000
Density 1.000
Friction 0.500
Restitution 0.000
IsSensor ☐
Enable Collision Events ☑
Enable Hit Events ☑
Enable Sensor Events ☑

RigidBody2D
Body Type Static
Fixed Rotation ☑
Is Awake ☑
Allow Sleep ☑
Sleep Threshold 0.500
Enable Collision Events ☑
Enable Hit Events ☑
Enable Sensor Events ☑
Mass 1.000
Gravity Scale 1.000
Linear Damping 0.010
Angular Damping 0.000
IsBullet ☐

Figure 11: *User interface of the game engine*

The images show the user interface of the engine editor, where developers can efficiently manage properties through a custom reflection library integrated with the entity component system. This system enables a structured and dynamic approach to handling components and their attributes ***(Fig. 12)***.



Label	Type	Script	Visibility
Camera	Camera		
Sky Light	Sky Light		
Plane object	Mesh		
Cube object	Mesh		
Point Light	Point Light		

Rendering:
Cube object
Plane object

Scene Settings

Lighting Enabled ☑

Render Statistics

CPU/GPU Times
Average CPU time (last 5s): 2.729ms (366.39 fps)
Current CPU time: 2.537ms (394.20 fps)
GPU time: 0.308ms
Shadow Map Pass: 0.000ms
Geometry Pass: 0.184ms
Main thread wait time: 0.001ms
Render thread wait time: 2.534ms

Translation	-3.96	0.60	2.18
Rotation	0.00	0.00	0.00
Scale	1.00	1.00	1.00

Radiance	R:1.000 G:1.000 B:1.000
Intensity	3.100
AttenuationRadius	10.100
FalloffExponent	2.000
SourceRadius	0.100

Figure 12: *User interface of the game engine*

3.6 Performance optimization

Throughout the development of the game engine, several key decisions were made with performance optimization as a priority. In this section, we summarize and compare the optimization techniques and architectural choices that led to measurable performance gains:

- Rendering Efficiency with Vulkan vs. OpenGL: By choosing Vulkan as main rendering backend, we achieved significant reductions in CPU overhead and improved multithreading performance. In internal benchmarks, scenes rendered using Vulkan performed 20–30% faster than equivalent OpenGL implementations due to its low-level GPU access and better memory management.
- EASTL vs. STL: Replacing the Standard Template Library (STL) with EASTL significantly reduced memory fragmentation and improved container operation speed, especially in scenarios with frequent allocation/deallocation of game objects.
- Custom Scripting with Reflexion: Using Reflexion for scripting avoids the overhead of external scripting engines (e.g., Lua or Python). Reflexion enables runtime access and manipulation of properties while staying within the high-performance bounds of C++.

4 Conclusion

We decided to develop our own game engine because existing options, while powerful, are often inefficient and do not fully meet our needs. Many popular engines available today have unnecessary overhead, restrictive licensing models, and limitations that hinder customization and optimization. We believe that game development can perform better with a more efficient, flexible, and well-optimized engine that provides developers with greater control over core functionalities. By creating our own engine, we aim to improve performance, streamline workflows, and introduce innovative features that enhance the overall development process. Our goal is not just to build another engine but to push the boundaries of what is possible in game development.

References

[1] Sadulov S., M. Mustafov, K. Shoilekova (2023). GAME ENGINE AND ENGINE COMPONENTS DESIGN AND DEVELOPMENT Reports Awarded with "Best Paper" Crystal Prize'23, 62nd Annual Scientific Conference - University of Ruse and Union of Scientists, , 173 - 178

[2] Sadulov S., M. Mustafov, K. Shoilekova (2024). Advantages and Disadvantages of developing a System for Interaction Between the Player and the Game Environment in the Game Eternal Vigil Reports Awarded with "Best Paper" Crystal Prize .

[3] Sellers, G., & Kessenich, J. (2016). Vulkan programming guide: The official guide to learning vulkan. Addison-Wesley Professional.

[4] Parberry, I. (2017). *Introduction to Game Physics with Box2D*. CRC Press.

[5] Gregory, J. (2018). *Game engine architecture*. AK Peters/CRC Press.

[6] Knoblauch, J., Sethuraman, A., & Hey, J. (2017). IMGui—a desktop GUI application for isolation with migration analyses. *Molecular Biology and Evolution*, *34*(2), 500-504.

[7] Akenine-Moller, T., Haines, E., & Hoffman, N. (2019). *Real-time rendering*. AK Peters/crc Press.

[8] Engel, W. (Ed.). (2014). *GPU Pro 5: advanced rendering techniques*. CRC Press.

Acknowledgement: This work was supervised by associate professor *Kameliya Shoylekova*, from *University of Ruse, department of Natural science and education, Ruse, Bulgaria*.

Serkan Sadulov
University of Ruse
Natural science and education
Bulgaria
E-mail: s216261@stud.uni-ruse.bg

Mustafa Mustafov
University of Ruse
Natural science and education
Bulgaria
E-mail: s216259@stud.uni-ruse.bg

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Evaluating Log Messages Using a Big Data Approach

Tobias Schneider

Abstract

Large log files, with hundreds of thousands of entries, can get hard to monitor. Due to the sheer size of the file, detecting if a system is to be considered in an error-state and finding the underlying root-cause gets increasingly challenging. This paper addresses the issues by detecting if the system is in an error-state, using the exponential moving average algorithm and providing a pattern-based proposal of the root-cause by employing a suffix array. The introduced log analyzing system is able to query the log file in various manners, speeding up analysis. Measurements, using log messages of a simulated microservice environment, showed that the approach introduced in this paper enhances user-driven analysis significantly. In numbers, for a log file with over 400,000 entries, it enables the operator of the log analyzing system to be 34 times faster in finding meaningful patterns, such as the most repeated message.

1 Introduction

Log files can quickly expand in volume by creating different types of log messages, resulting in data sets increasing in size and complexity [1]. In some cases, this can reach up to 120 million messages per hour, making it difficult and labor-intensive to identify meaningful insights[2]. However, efficiently capturing and analyzing log messages is crucial to prevent outages and diagnose potential errors, since log messages state information in regard to the state of the system [3, sec. 2.1]. While solutions have been proposed to manage log messages using big data technologies [4], extracting operational profiles from log messages via suffix arrays [5], and using neural networks to detect anomalies [6], this work addresses those challenges by introducing an automated, pattern-based approach without having to train a machine-learning model first. In terms of this paper, it is assumed that the monitored environment does not stop running once an error is encountered, but keeps producing log messages that have to be handled. Specifically, this work considers incoming new log messages as an online problem and treats the log messages for analysis as strings, which allows the use of existing algorithms designed for these cases. The novel log analyzing system introduced in this paper, is based on a combination of the exponential moving average algorithm (EMA) and a suffix array. It performs two main functions: first, it reads a log file containing informational and error messages, initiates auto-analysis and provides querying options once a predefined threshold is reached. Second, it processes log files consisting solely of error messages to generate auto-analysis and enable queries. It is important to note that the objective is not to achieve absolute error detection, but rather to provide a first recommendation and assistance tool for supporting the identification of outages and their underlying causes.

The remainder of the paper is structured as follows: Section 2 provides a brief introduction and definition of the used terminology and technology. A structured overview of the log analyzing system, as well as the ways it can be queried is provided in section 3. Additionally, section 3 introduces the format of the log messages used. In section 4, the detailed concepts and novel approaches developed for the analysis of log messages are described. Section 5 shows the measured time using the log analyzing system and compares it to pure user-driven analysis. The final section 6, concludes the work of this paper and gives an outlook on how it can be used in future directions of study.

2 Technology Fundamentals

2.1 Big Data

According to the National Institute of Standards & Technology (NIST), Big Data consists of extensive datasets, primarily in the characteristics of volume, variety, velocity, and/or variability [7, p. 6]. As outlined in section 1 log files can hold a huge amount of log messages and are therefore considered Big Data. For the scope of the paper, the focus lays mainly on the volume and velocity part, by addressing the frequency of the incoming log messages as velocity and the amount of log messages (and therefore the size of the corresponding log file) as volume.

2.1.1 Treating Incoming Log Messages as an Online Problem

An online problem is a setting, where incoming data is considered as a stream, with each item arriving one at a time. Upon arrival, it has then to be processed without knowing future items; the algorithms tackling this setting are called online algorithms [8]. By abstracting the problem of handling incoming log messages to an online problem, it allows the use of online algorithms, such as the exponential moving average algorithm.

2.1.2 Treating Cumulated Log Messages as a String

String algorithms constitute a well-researched domain, extensively covered in literature such as Gusfield [9]. By treating the log file under analysis as a string, these established algorithms can be leveraged. This approach facilitates the application of string-based algorithms to analyze large volumes of log data effectively.

2.2 Exponential Moving Average Algorithm

The exponential moving average algorithm is an online algorithm that re-calculates the average for each new element by only using a weight α and the previous calculated EMA_{t-1}[10]. The EMA is defined in Brown [11, p. 3] as follows:

$$\begin{aligned} &EMA_0 = x_0, \\ &EMA_t = \alpha \cdot x_t + (1-\alpha) \cdot EMA_{t-1}, \quad 0 \leqq \alpha < 1 \end{aligned} \tag{1}$$

The parameter α is controlling how much weight is given to the most recent object, compared with all previous ones [10]. The EMA is highly memory efficient, as it is only necessary to maintain the record of the last computation [11, p. 3].

2.3 Suffix Array

In order to answer the question what is the longest common factor of two given strings, the suffix tree data structure was developed [14, p. 241]. A suffix tree is defined as a rooted directed tree for an m-character string S with exactly m leaves numbered 1 to m. Essentially covering all possible suffixes of S [9, p. 90]. As a more memory-efficient alternative, Manber and Myers [15] proposed a data structure called suffix array, requiring three to five times less space [15, p. 1], while solving the substring problem or exact matching almost as efficiently as the suffix tree [9, p. 149]. Gusfield formally defines the suffix array as follows:

Definitie 1 *"Given an m-character string T, a suffix array for T, called Pos, is an array of the integers in the range 1 to m, specifying the lexicographic order of the m suffixes of string T." [9, p. 149]*

Meaning that the first element in the suffix array is the integer value of the starting point of the lexically smallest suffix, followed by the second smallest and so on. At the end of the string a $ is added as a terminal symbol and given the lexical smallest value [9, p. 149]. The suffix array can then be extended by the longest common prefix (LCP), that is the length of the common prefix of two suffixes [9, p. 152]. For example, the `lcp(`i`, `j`)`, where $i = na\$$ and $j = nana\$$, is 2. Those values can be stored in an additional array `LCP` of size $N = |T|$ [14, p. 242]. `LCP[0]` is set to 0 and `LCP[i] = lcp(`$A_{Pos[i-1]}, A_{Pos[i]}$`)`, $1 \leq i \leq N - 1$ [15, p. 9], where A refers to the string value of the substring starting at i. Figure 1 illustrates the suffix array and its corresponding LCP values for the string *banana\$*.

No.	**Suffix**
1	banana$
2	anana$
3	nana$
4	ana$
5	na$
6	a$
7	$

(a) Suffixes of **banana$**

No.	**Suffix**	***lcp***
7	$	*0*
6	a$	*0*
4	ana$	*1*
2	anana$	*3*
1	banana$	*0*
5	na$	*0*
3	nana$	*2*

(b) Suffix array and LCP for the word **banana$**

Figure 1: Suffixes for the word **banana$**

Based on the values stored in the LCP array, the longest repeated substring can be acquired; that is finding the longest substring that appeared at least twice in the text [16, p. 875]. Obtaining the longest repeated substring is done by determining $\max\{LCP\}$ [16, p.876]. Additionally, counting the occurrence of a sequence is possible by finding that sequence and counting the number of neighboring LCP values until a smaller value is reached [5, sec. 4.3]. Due to the sorted nature of the suffix array it is possible to identify all occurrences of a word W of length $P \leq N$ in $\Theta(P \log N)$ time by utilizing binary search [15, p. 3]. Considering the LCP values during the search, reduces the runtime to $\Theta(P + \log N)$ [15, p. 5]. To construct the suffix array and its corresponding LCP array, various algorithms exist, differing in runtime and memory usage [17]. Still, optimizing the creation remains a current field of study [18].

3 Setup

3.1 Setup of the Log Analyzing System

As described in section 1 the log analyzing system is able to be utilized in two manners. One is to detect if the system under analysis is in an error-state by calculating the current success-ratio, that is the ratio between informational and error messages. The moment the error-state is detected, it starts the analysis based on a sliding window of the last n error messages. As the size of n is highly dependent on the monitored environment, it has to be determined for each monitored environment individually. This is called live-monitoring. The second manner, called post-analysis, is to provide a log file consisting solely of error messages on which the analysis is then started directly. An illustration of the possible ways to employ the log analyzing system can be found in Figure 2. In terms of this paper, analysis is a pattern-based proposal of the root-cause (auto-analysis) and the offering of ways to query the error messages (user-driven analysis).

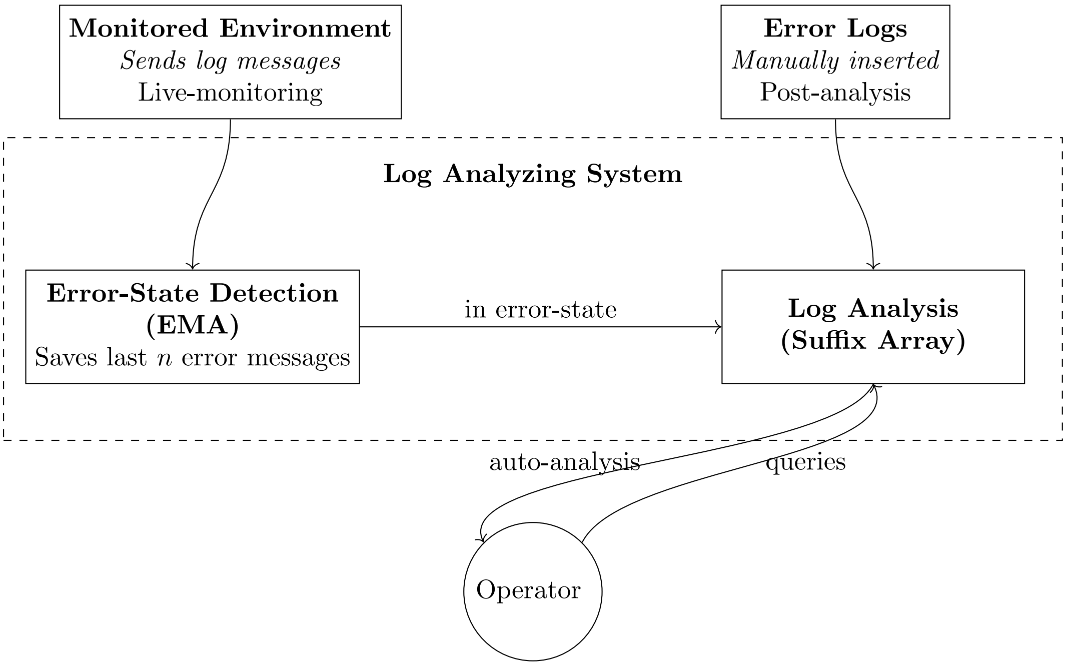


Figure 2: Overview Log Analyzing System

3.2 Format of the Log Messages

The log messages used for this work have been conducted by employing a simulated microservice environment, consisting of four microservices, that communicate circularly. The microservices state, by using a logging framework, if their operations were successful or not. The framework is based on JavaScript Object Notation (JSON) and comprises a timestamp, log level, service name, message, and region field. The message field is the only field that can be freely set by the microservice. The log level can either be `Info` or `Error`. The region field states in which geographical region the microservice runs. An example log message looks like this for the log level `Info`:

```
{"timestamp": "2025-01-02T15:52:32.078453Z",
"logLevel": "Info",
"serviceName": "MS1",
"message": "Task 1 succeeded",
"attributes": {
    "region": "eu-central-1"
}}
```

4 Development

Note, that the methods described in this section rely on the format of the log messages introduced in section 3.2. However, all of them can easily be adjusted to another JSON-based logging framework.

4.1 Error-State Detection

Pursuant to the description of the setup of the log analyzing system in section 3.1 it is essential to design an algorithm that quantifies the current success ratio of the system, monitoring if it remains above a predefined threshold. This threshold can be conceptualized as the minimum percentage value, which is derived as the average ratio of `Info` to `Error` log messages. The algorithm should be designed to prioritize newer log messages, assigning them a higher weight compared to older messages. To fulfill this requirement, the exponential moving average algorithm, introduced in section 2.2 has been selected. For each new incoming log message, the current average is recalculated in accordance to Equation 1. Depending on the log level of each message, the current x_t value can assume two values: 1, if the message is of level `Info` or 0, for `Error`. As α is the parameter that controls how much weight is given to the most recent observation, compared with all previous observations [10], it is advisable to set it to a relatively low value. This prevents a significant drop in the success ratio in case of a few consecutive error messages. In accordance with Brown, the value of α was set to 0.1 [11, p. 10]. Finding a general proposal for α goes beyond the scope of this paper. To consider the monitored environment in an error state, the current success ratio (x_t) has to be below the threshold for a pre-defined consecutive number of times. This is based on the notion that it may not be desirable for an alarm to be triggered immediately upon the application attaining a particular critical point. Rather, it may be preferable for the alarm to be triggered after a designated interval, thereby ensuring that the cause of the drop is not merely a handful of error messages. As this is highly dependent on the application, the log analyzing system is monitoring, it has to be set manually. Once the application is considered in an error state, the suffix array is constructed with the error messages as the input and the analysis started.

4.2 Patterns for Auto-Analysis

The following describes the approaches taken for patterns that should be auto-analysed using a suffix array. Meaning, at the start of the analysis by the log analyzing system, the suffix array will always be queried for those. For the scope of this paper, it will always determine the most repeated log message and the longest repeated message. The most repeated message is the string that occurred the most often inside the `message` field, while the longest repeated message is defined as the longest repeated substring in the same field. The results of these operations will then be returned as the pattern-based root-cause proposal. Both the longest repeated message and the most frequent message are substrings of the string s within the `message` field, commencing from the beginning of s. This means that, while they can be substrings of the messages, they must start from the outset. For the four log messages shown in Figure 3, the most repeated message is `" FATAL ERROR at MS2"`. In this particular case, the longest repeated message is the same as the most repeated message. However, in case at least two more log messages appear, starting with `" FATAL ERROR at MS2"` but followed by some string β, the longest repeated message would be `" FATAL ERROR at MS2 + `β`"`, while the most repeated message would be `" FATAL ERROR at MS2`.

```
1:  {"timestamp": "2025-01-21T13:55:42.699393Z",
    "logLevel": "Error","serviceName": "MS1",
    "message": "Error received, aborting",
    "attributes":{"region": "eu-central-1"}}
2:  {"timestamp": "2025-01-21T13:55:42.699394Z",
    "logLevel": "Error","serviceName": "MS2",
    "message": "Error caught: FATAL ERROR at MS2 aborting!",
    "attributes":{"region": "eu-central-1"}}
3:  {"timestamp": "2025-01-21T13:55:42.699398Z",
    "logLevel": "Error","serviceName": "MS3",
    "message": "FATAL ERROR at MS2",
    "attributes":{"region": "eu-west-1"}}
4:  {"timestamp": "2025-01-21T13:55:42.699400Z",
    "logLevel": "Error","serviceName": "MS4",
    "message": "FATAL ERROR at MS2",
    "attributes":{"region": "us-east-1"}}
```

Figure 3: Logs Example Auto-Analysis

Note that, while various different, and sometimes simpler, data-structures for counting occurrences exist [12],[13], the suffix array has been chosen for the following reasons. For patterns based on the values of the `message` field, it is necessary to only consider these information and not the whole log message; this can be ensured using a suffix array (see section 4.2.1) Additionally, the suffix array holds all log messages of the log file, making it therefore possible to use these to develop further patterns that are specific to the monitored application. Last, the suffix array can be further used during user-driven analysis, such as free-text search, without having to introduce a new data structure.

4.2.1 Most Repeated Message

Assuming the presence of the LCP array and all characters set to lowercase, the most frequently repeated message can be identified similarly to locating the most common sequence, as described in section 2.3. However, a few changes have to be made to appeal to the structure of the log messages and to prevent the outcome from being insignificant in terms of analysis. Otherwise, according to the procedure described in 2.3, the result would be the character the most entries in the suffix array start with. Due to the structure of the log messages introduced in section 3.2, all messages to be considered are the string values of the `message` key. Finding the most repeated message can then be done by finding all occurrences of `message":` in the suffix array and determining the prefix that occurred the most often. The flow of this operation is visualized in Figure 4.

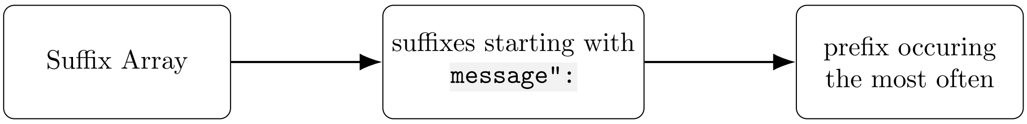

Suffix Array
suffixes starting with `message":`
prefix occuring the most often

Figure 4: Most Repeated Message Flow

A naive approach would be to count and compare the LCP values from the top (first occurrence of `message":`) to bottom (last occurrence of `message":`). So for an LCP array $LCP = [a_1, a_2, \ldots, a_n]$ the element a_i with the highest count $C(a_i) = |\{j > i \mid a_j \geq a_i \text{ and } \forall k \in [i, j],\ a_k \geq a_i\}|$ is the most frequent value. To obtain the string value, a lookup is done in the suffix array at the index of a_i and the corresponding string value, starting from the beginning to the $LCP[i]$ character, returned. However, for the log messages shown in Figure 3 this approach would return `message": "` instead of the expected `message": "fatal error at ms2"`.

The issue is caused by the structure of the log message, the `message` field always starts with `message": "` and thus resulting in an LCP value of 11, if the messages have only the `message": "` prefix in common. Using the naive approach described above, once a LCP value of 11 is reached at index i, all following values will only increase the counter for i, disregarding how often their LCP value appears. To prevent this from happening, the suffixes starting with `message": "` are separated in contiguous sections. Due to the nature of the LCP array [15, p. 9], a LCP value of 11, the length of their common prefix, states that the string value in the `message` field starts with a different character. So, each part from a LCP value of 11 to the next LCP value of 11 is considered a section *sec*. The counter *C(sec)* holds the number of messages in one section and is increased for each consecutive entry with a LCP value greater 11. Upon entering a new section, the *C(sec)* is set to zero. In case a LCP value smaller than 11 is encountered, the operation stops, since the corresponding suffix is not starting with `message":` anymore [9, p. 152]. It then defines the most repeated message as the substring of the entry in the suffix array at index i, where i is the index of the smallest LCP value $\neq 11$ for the section with the highest counter. Table 1 shows a snippet of the suffix array expanded with the section counter for the example log messages shown in Figure 3.

index i	**Suffix**	***lcp***	***sec***	***C(sec)***
5000	`message": "error caught: [...]`	*11*	*0*	*0*
5001	`message": "error received, aborting",[...]`	*17*	*0*	*1*
5002	`message": "fatal error at ms2",[...]`	*11*	*1*	*0*
5003	`message": "fatal error at ms2",[...]`	*30*	*1*	*1*
5004	`message": "fatal error at ms2",[...]`	*30*	*1*	*2*
5005	`ms2",[...]`	*1*	*end*	-

Table 1: Snippet of example messages in suffix array with corresponding lcp and counter-value

4.2.2 Longest Repeated Message

To get a second suggestion and starting point for analysis conducted by the operator, the longest repeated message is identified as well. In this work it refers to the longest repeating substring starting with `message":`. It can be obtained by tracking down the largest LCP value with that prefix [16, p. 876]. Figure 5 visualizes the process.

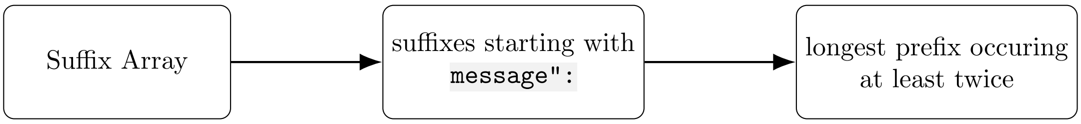

Suffix Array
suffixes starting with message":
longest prefix occuring at least twice

Figure 5: Longest Repeated Message Flow

Finding the longest repeated message is done by saving the max LCP value that came across during the finding of the most repeated message. Since obtaining the most repeated message requires the comparison of all numeric values of the LCP for all entries in the suffix array starting with `message":` anyway, obtaining the most and longest repeated message can be done in one pass.

4.3 Queries for User-Driven Analysis

To verify the proposals from section 4.2 or to gain additional insight into the log messages, the log analyzing system offers the operator the ability to query it as well. In contrast to queries and patterns described in section 4.2, the results from those queries are not automatically pre-computed when creating the suffix array. The functionality for user-driven analysis includes the identification of the most affected region, the facilitation of free-text search, and the retrieval of all log messages for a designated microservice, or a specified timestamp.

4.3.1 Most Affected Region

This operation determines the value that occurred the most frequently for the `region` key. For instance, as illustrated in Figure 3, the most affected region is `"eu - central -1"`, as it appears in two of the four log messages. Finding the most affected region, can be done in a manner analog to the identification of the most repeated message, as described in section 4.2. In contrast to the evaluation of the `message` part of the log message, it is not necessary to determine the lowest LCP value of a section, since the entire value of `region` is of interest. Therefore, the sectioning can be accomplished by identifying the initial instance of `region":` and grouping all subsequent items into sections with the same LCP value [19, p. 1222] until an element that does not begin with `region":` is encountered. As the suffix array is sorted, each section is contiguous [19, p. 1222]. Given these considerations and the fact that the structure of the log messages ensures that the same region always has the same length, it can be concluded that each section represents a different region. To identify the most affected region, the LCP value *lcp* of the section with the highest number of items is selected, and the substring, starting at the beginning of the first element to the *lcp*th character returned. For the example shown in Table 2 the selected section is *sec 0*.

index i	**Suffix**	***lcp***	***sec***	***C(sec)***
5000	`region": "eu-central-1" [...]`	*0*	*0*	*0*
5001	`region": "eu-central-1",[...]`	*23*	*0*	*1*
5002	`region": "eu-west-1",[...]`	*13*	*1*	*0*
5003	`region": "us-east-1",[...]`	*10*	*2*	*0*
5004	`rvice[...]`	*1*	*end*	-

Table 2: Snippet of example regions in suffix array with corresponding LCP and section

4.3.2 Free-Text Search

In the context of log analysis, this works defines free-text search as the ability to search for a string S and retrieve the occurrence of it in the log file. In essence, this entails locating all instances of S in the suffix array. As demonstrated by Manber and Myers [15], this task can be addressed by employing two binary searches: one that finds the first and another that finds the final occurrence of S. The total runtime complexity of this approach is that of the used binary search algorithm [15, p. 3]. For the log analyzing system, the string value of each entry in the determined range is returned to the operator of the log analyzing tool.

4.3.3 Microservice and Timestamp Search

The objective is to find all log messages and timestamps for a designated microservice. Given the characteristics of the structure of the log messages, it is assured that the `serviceName`, as well as the `timestamp` key, which holds the name and timestamp of the microservice, is succeeded by the `message` field. This attribute enables the utilization of the same approach as for free-text search, but instead of searching for any string, the input string S is set to either `serviceName": "` plus the name of the sought microservice or to `timestamp": "` followed by the queried timestamp. Note that S was augmented with the key of the value in question to ensure a correct result and prevent unwanted results.

In contrast to the microservice search, where it has to be ensured that the input is a valid name, it is not necessary for the user to enter the whole timestamp. Instead, it is possible to look for multiple timestamps by entering sub-values of the timestamp, as long as they are in accordance of the employed internet timestamp format [20]. For example, looking up `2025-01-21` returns all log messages for the 25th of January 2025, since `2025-01-21` is a prefix of all timestamps for that day [20] and therefore being the first value behind `timestamp": "`.

4.4 Dynamic Updating

In the event of an ongoing outage, it might be desirable to retrieve new log messages for analysis, thereby updating the existing suffix array. Note that none of the patterns for analysis, except free-text search, described in section 4.2 and 4.3 necessarily require the rest of the entry following the log message in question. Utilizing this characteristic, the original suffix array SuA_{og} can be updated by inserting all suffixes of the new log file into SuA_{og}, while maintaining the lexicographic order. Resulting in a new, but formally incomplete, suffix array$SuA_{updated}$. Consider Figure 6 for a visualization of the procedure.

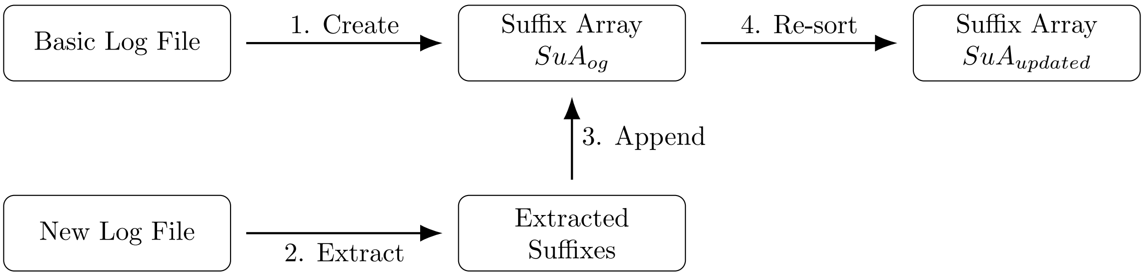


Figure 6: Dynamic Updating Flow

This approach comes at the cost of losing the ability to search over the last log message in the old log file into the new one. An operation that might be interesting to an operator, who wants to verify that there was a consecutive appearance of certain log messages. All other introduced patterns work without a disadvantage. To construct $SuA_{updated}$ all suffixes from the considered log file are appended to the end of SuA_{og}, which is then re-sorted. To take advantage of the existing sorted suffix array TimSort is employed. This sorting algorithm is particularly effective on partially ordered lists [21]. While the runtime complexity of TimSort still remains at $\mathcal{O}(n \log n)$ [22], it has been shown that for the implemented version of TimSort in certain programming languages, the runtime complexity is actually $\mathcal{O}(n + n \log \rho)$, where ρ is the number of runs [22]. The number of runs can be reduced, if the new suffixes are appended in an (almost) sorted order [21]. Still, if in further usage of the log analyzing system, it is established that updating is considered a heavy-used feature, a more complex suffix array implementation optimized for dynamic extension [14] should be considered.

4.5 Implementation

All the concepts introduced in sections 4.2 and 4.3 were implemented in accordance with the described algorithms and ideas. The programming language used for the implementation of the log analyzing system is Java. This comes at the benefit that Java provides libraries for file-reading [23] as well as an implementation of TimSort in the `java.util.List` library [24]. To construct the suffix array, the construction algorithm proposed by Sedgewick and Wayne [25] is employed. Due to its simplicity, this construction algorithm was chosen. In theory, it has a quadratic runtime which, in practice, is not considered problematic as long as there are no long runs of identical characters [16, p.884]. The space required for the utilized suffix array implementation is $\Theta(n)$ [16, p. 882]. As the LCP values are only of interest for the auto-analysis and for obtaining the most affected region (both described in section 4) the LCP values are only computed and stored for the spans in the suffix array needed to execute those operations. Additionally, for printing the results of the queries (and therefore the value of the suffix array at that index), a function called `extractLogJson(String message)` was added. The structure of the log messages ensures that each log message starts with `{"timestamp ":`, based on this given structure `extractLogJson(String message)` cuts the input `message` at the first occurrence of `{"timestamp ":` to only return the log message in question. This adds an additional runtime of $\Theta(n)$, where n is the length of `message`, but otherwise the returned results showed to be hard to read, especially for larger suffix arrays.

5 Measurements

The following shows the results of the measurements conducted with the Java implementation of the log analyzing system and compares it to time spent by the author of this paper gathering the same information using pure user-driven analysis. For this scenario, pure user-driven analysis is defined as using no additional tools except a text editor to open the log file and its search function. All the measurements have been conducted on a 2021 MacBook Pro with 16 GB of memory and an Apple M1 Pro CPU. Measurements in terms of memory usage have been conducted by creating and examining a memory snapshot via IntelliJ IDEA[26]. Time measurements were done by employing the Instant.now() function of the java.time package [27].

5.1 Measurements Error-State Detection

Live-analysis took 247ms for detecting the error-state of the monitored environment, after considering 127,884 log messages. The combination of the EMA calculation and setting the interval for the amount of EMA_t values that have to be lower than the threshold to 10, determined the error-state correctly for 20 different log files of various sizes. Taking a look at the same log file and evaluating if the system is in an error state manually, took significantly longer. That is 2 minutes or 120,000 microseconds.

5.2 Measurements Suffix Array Analysis

Table 3 shows the necessary space required for the suffix arrays of log files of various sizes.

Entries log file	Log file space	Suffix array entries	Suffix array space	LCP values space
400,067	78 MB	65,466,852	1.57 GB	32.02 MB
200,151	32.5 MB	32,512,925	78.31 MB	16.02 MB
100,027	16,04 MB	16,395,419	393.49 MB	8.02 MB

Table 3: Comparison disk-space based on different log files

The Tables 4, 5, and 6 provide comparisons in terms of time between pure user-driven analysis and the same operations done by the log analyzing system. Note, that finding all log messages for a time range is significantly faster when done using pure user-driven analysis. This is because the range was set to the whole day, for which the log analyzing system has to consult each entry, compared to scrolling to the end of the file and verifying that all entries have the same day. Additionally, due to the setup of the testing environment, it can happen that the name of one specific task occurs more often in a log file than in one that has more entries.

Operation	Time Spent (ms)	Frequency	User Time (ms)
Building	20,345	N/A	N/A
Auto-Analysis	401	N/A	720,000
Search String `task 1`	4,432	2,395	60,000
Microservice Search	99,487	100,118	120,000
Search Time Single	90	1	22,000
Search Time Range	385,257	400,067	35,000
Most Affected Region	157	N/A	480,000
Updating (130 entries)	9,524	N/A	N/A

Table 4: Performance for 400,067 entries

Operation	Time Spent (ms)	Frequency	User Time (ms)
Building	10,049	N/A	N/A
Auto-Analysis	310	N/A	540,000
Search String `task 1`	4,884	4,942	60,000
Microservice Search	27,831	49,986	90,000
Search Time Single	86	1	19,000
Search Time Range	118,905	200,151	28,000
Most Affected Region	93	N/A	270,000
Updating (130 entries)	5,080	N/A	N/A

Table 5: Performance for 200,151 entries

Operation	Time Spent (ms)	Frequency	User Time (ms)
Building	4,895	N/A	N/A
Auto-Analysis	125	N/A	210,000
Search String `task 1`	170	142	25,000
Microservice Search	7,192	25,029	50,000
Search Time Single	42	1	5,000
Search Time Range	31,035	100,027	12,000
Most Affected Region	56	N/A	110,000
Updating (130 entries)	2,866	N/A	N/A

Table 6: Performance for 100,027 entries

Further measurements showed that the difference between pure user-driven analysis and using the log analyzing system remains high, even for smaller log files. Specifically,the measurements indicated that the sweet-spot at which the log analyzing system becomes faster than pure user-driven analysis is for log files consisting of over 2,000 log messages, as this seems to be the point where simply scrolling through the file and looking for candidates for the operation becomes impracticable.

6 Conclusion

In this paper, a novel log analyzing system has been introduced that enhances analysis significantly. Based on log messages, the system is able to detect if the monitored environment is in an error-state, provides pattern-based proposals for an extracted root-cause of what could have brought the monitored environment into an error-state, and offers querying capabilities over the underlying error log messages. To do so, the incoming log messages are considered an online problem, allowing the use of the exponential moving average algorithm for calculating the success-ratio of the monitored environment. For analysis, the log messages under consideration are treated as strings and analyzed by employing a suffix array. This suffix array is then used to provide automized patter-based root-cause proposals, by determining the most repeated message and the longest repeated message. Additionally, its querying options offer the possibility to get all log messages from a specific microservice, search for a given timestamp, receive the geographic region with the most error messages, as well as using free-text search. The fact that the proposed log analyzing system is able to provide auto-proposals to detect the root-cause and uses a suffix array to do so, sets it apart from other log analyzing solutions using a big data approach [1], [4]. While there have already been solutions on how to extract operational profiles using a suffix array [5] and how to cluster sequences via suffix arrays [19], this work proposed to split the values of the suffix array into sequences to extract information based on the LCP values. For future usage, the log analyzing system could be adjusted to work with an observability framework such as OpenTelemetry, a widely used open-source observability framework that instruments, generates, collects, and exports telemetry data including log messages [28].

Acknowledgement: This work was supervised by Professor *Ph.D. Andreas Siebert*, from *Hochschule Landshut*. The author also thanks XIBIX Solutions GmbH that sponsored the work of this thesis by allowing him to write and research it during working hours.

References

[1] A. Balliu, D. Olivetti, O. Babaoglu, M. Marzolla, and A. Sîrbu, A big data analyzer for large trace logs, *Computing*, vol. 98, no. 12, pp. 65–78, 2016.

[2] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, An evaluation study on log parsing and its use in log mining, *46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pp. 654–662, 2016.

[3] K. Kent and M. P. Souppaya, *SP 800-92. Guide to Computer Security Log Management,* National Institute of Standards & Technology, Gaithersburg, MD, USA, 2006.

[4] M. Lemoudden and B. E. Ouahidi, Managing cloud-generated logs using big data technologies, *2015 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, pp. 1–7, 2015.

[5] M. Nagappan, K. Wu, and M. A. Vouk, Efficiently Extracting Operational Profiles from Execution Logs Using Suffix Arrays, *Proceedings of the 2009 20th International Symposium on Software Reliability Engineering*, pp. 41–50, 2009.

[6] M. Catillo, A. Pecchia, and U. Villano, AutoLog: Anomaly detection by deep autoencoding of system logs, *Expert Systems with Applications*, 2022.

[7] W. Chang and N. Grady, *NIST Big Data Interoperability Framework: Volume 1, Definitions*, Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, USA, 2019.

[8] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen, Online algorithms with advice: A survey, *ACM Computing Surveys (CSUR)*,vol. 50, no.2, pp. 1–34, 2017.

[9] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, USA, 1997.

[10] J. Loveless, S. Stoikov, and R. Waeber, Online algorithms in high-frequency trading, *Commun. ACM*, vol. 56, no. 10, pp. 50-56, 2013.

[11] R. G. Brown, *Exponential smoothing for predicting demand*, Little, Massachusetts, USA, 1956.

[12] G. Cormode, S.Muthukrishnan An improved data stream summary: the count-min sketch and its applications, *Journal of Algorithms*, vol. 55, no. 1, pp. 58-75

[13] A. Burstein, M. Toufik Counting occurrences of some subword patterns, *Discrete Mathematics and Theoretical Computer Science*, vol. 6, no.1, pp. 1-12

[14] M. Salson and T. Lecroq and M. Léonard and L. Mouchard, Dynamic extended suffix arrays, *Journal of Discrete Algorithms*, vol. 8, no. 2, pp. 241–257, 2007.

[15] U. Manber and G. Myers, Suffix Arrays: A New Method for On-Line String Searches, *SIAM Journal on Computing*, vol. 22, no. 5, pp. 935–948, 1993.

[16] R. Sedgewick and K. Wayne, *Algorithms Fourth Edition*, Pearson Education Inc., Stanford, USA, 2011.

[17] S. J. Puglisi, W. F. Smyth, and A. H. Turpin, A taxonomy of suffix array construction algorithms, *acm Computing Surveys (CSUR)*, vol. 39, no. 2, pp.4-es, 2007.

[18] J. Olbrich, E. Ohlebusch, and T. Büchler, Generic Non-recursive Suffix Array Construction, *ACM Transactions on Algorithms*, vol. 20, no. 2, pp. 1-42, 2024.

[19] K. Malde, E. Coward, and I. Jonassen, Fast sequence clustering using a suffix array algorithm, *Bioinformatics*, vol. 19, no. 10, pp. 1221-1226,2003.

[20] C. Newman and G. Klyne, Date and Time on the Internet: Timestamps, RFC 3339, 2002.

[21] T. Peters, Timsort - A Python Sorting Algorithm, `https://svn.python.org/projects/python/trunk/Objects/listsort.txt`, 2002 accessed: 2025-02-02.

[22] N. Auger, V. Jugé, C. Nicaud, and C.Pivoteau, On the Worst-Case Complexity of TimSort, *26th Annual European Symposium on Algorithms (ESA 2018),* vol. 112, pp. 4:1 – 4:13, 2018.

[23] Oracle Corporation, FileReader (Java SE 23 API), `https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/io/FileReader.html` 2024, accessed: 2025-02-08.

[24] Oracle Corporation, List (Java SE 23 API), `https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/util/List.html`, 2024, accessed: 2025-02-02.

[25] R. Sedgewick and K. Wayne, SuffixArray.java, `https://algs4.cs.princeton.edu/63suffix/SuffixArray.java.html`, 2022, accessed: 2025-02-10.

[26] JetBrains, Create a Memory Snapshot, `https://www.jetbrains.com/help/idea/create-a-memory-snapshot.html`, 2025, accessed: 2025-02-18.

[27] Oracle Corporation, Instant (Java SE 23 API), `https://docs.oracle.com/en/java/javase/23/docs/api/java.base/java/time/Instant.html`, 2024, accessed: 2025-02-18.

[28] OpenTelemetry, OpenTelemetry Documentation, `https://opentelemetry.io/docs/`, 2025, accessed: 2025-03-03.

Tobias Schneider
Hochschule Landshut
Faculty of Computer Science
Am Lurzenhof 1, 84036 Landshut
Germany
E-mail:
*t*obiasschneider1@acm.org

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Tecky

Alexandru Emil Sofonea

Abstract

Tecky is an AI-powered automation application for macOS, designed to transform how users interact with the operating system. By combining a modern Swift and SwiftUI-based interface with a local AI server running LLMs optimized for Apple Silicon via MLX, Tecky can interpret natural language commands and translate them into concrete actions on active applications. The agent leverages various Foundation libraries to navigate and control the graphical interface, enabling access to UI elements, simulating clicks and typing, and interacting intelligently with multiple applications simultaneously. Through tool-calling for LLMs and structured JSON output, Tecky not only understands user intent but can also execute tasks in a transparent and extensible manner. This paper documents the system's architecture, technical challenges, and potential, offering a clear perspective on how local LLMs can be transformed into practical agents within the Apple ecosystem.

1 Introduction

Task automation on desktop operating systems is currently limited to traditional tools such as AppleScript, Automator, or RPA-type applications. With the emergence of large language models (LLMs), it has become possible to interpret tasks expressed in natural language and convert them into contextual, concrete sequences of actions. This paper presents a system that integrates local custom trained LLMs and Vision models with a native macOS application to perform complex tasks through interaction with the graphical interface of other applications. This intelligent agent operates entirely offline, effectively functioning as a virtual employee. It supports multiple models and runs on macOS starting from version Big Sur, compatible with both Apple Silicon and Intel architectures (post-2017).

2 Architecture

The software architecture is built specifically for macOS and centers around the main component of the application, the "Tecky Controller," which integrates modules for application logic control (Controller), the graphical interface (UI), and model management (ModelController). The application uses macOS libraries such as Foundation, Cocoa, ApplicationServices, and CoreGraphics to access services like accessibility permissions and control over keyboard and mouse events. Interaction with AI models (such as DeepSeek Coder, LLaMA 3.2, Mistral) is handled through an LMS server hosted on a Python Execution platform. This server receives commands from the application, processes them using the LLMs, and returns the results for display in the UI. The architecture is modular, extensible, and designed for integration with advanced AI services.

2.1 MacOS Controller

The macOS control component, internally referred to as the Tecky Controller, is responsible for direct interaction with the operating system and the interfaces of open applications. It is composed of several specialized services working together to allow the agent to perceive and manipulate elements in the graphical interface in an automated way.

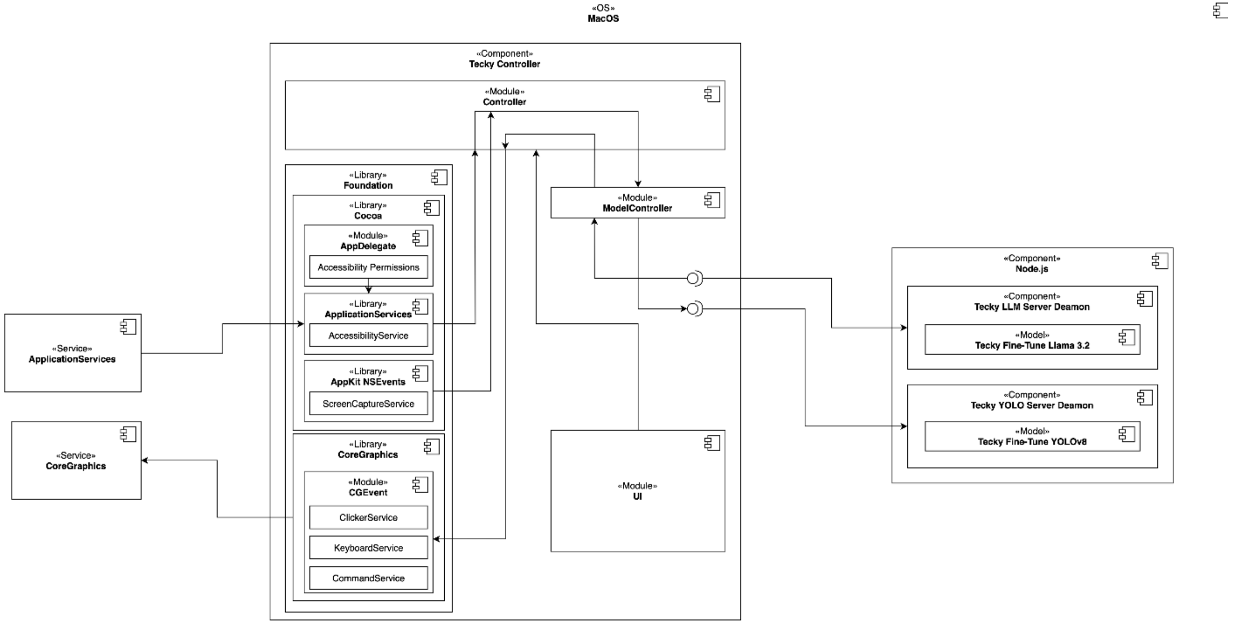


Fig. 1: Tecky Controller Architecture

The main service, AccessibilityService, part of the Cocoa framework, is used to access and traverse the UI hierarchy of active applications. It identifies the target application, extracts all visible windows, and recursively retrieves interactive elements (such as buttons, text fields, links) along with their textual descriptions (title, role) and exact screen coordinates. To add further context to the AccessibilityService, I implemented an additional Vision Layer that receives screenshots from the user's screen and passes them to a Vision Model Server. For each element, a unique identifier is generated, which can later be referenced by the LLM. Elements are filtered based on predefined interaction roles (e.g., AXButton, AXTextField, AXCheckBox), and the results are returned as separate descriptions and positions.

```
let workspace = NSWorkspace.shared
let runningApps = workspace.runningApplications
func traverseAccessibilityHierarchy(element: AXUIElement, appName: String)
{
     var value: CFTypeRef?
     let result = AXUIELementCopyAttributeValue(element,
kAXChildrenAttribute as CFString, &value)
     if result = .success, let childrenArray = value as? [AXUIELement] {
          for child in childrenArray {
               processElement (child, appName: appName)
               traverseAccessibilityHierarchy (element: child, appName:
appName)
```

```
          }
     }
}
```

To execute actions on these elements, I created ClickerService and TyperService, both built on top of the CoreGraphics library. ClickerService is responsible for moving the cursor to a specific position on the screen and simulating a mouse click. It uses the CoreGraphics framework to generate synthetic events such as mouseMoved, mouseDown, and mouseUp via CGEvent. Complementing this, TyperService allows for simulating text input character by character. Each character is mapped to its corresponding key code (CGKeyCode), and actions are executed using keyDown and keyUp events, also through CoreGraphics. At this stage, the service supports basic alphanumeric characters, with the potential for future expansion to include symbols and complex key combinations.

```
func moveMouse(x: CGFloat, y: CGFloat) {
     let targetPoint = CGPoint(x: x, y: y)
     let eventSource = CGEventSource(stateID: .hidSystemState)

     // Move the mouse to the specified coordinates
     let moveEvent = CGEvent(mouseEventSource: eventSource,
                             mouseType: .mouseMoved,
                             mouseCursorPosition: targetPoint,
                             mouseButton: .left)
     moveEvent?.post(tap: .cghidEventTap)
}
func pressKey(_ keyCode: CGKeyCode, withModifiers modifiers: CGEventFlags =
[]) {
     let eventSource = CGEventSource(stateID: .hidSystemState)
     let keyDown = CGEvent (keyboardEventSource: eventSource,
                             virtualKey: keyCode,
                             keyDown: true)
     let keyUp = CGEvent(keyboardEventSource: eventSource,
                             virtualKey: keyCode,
                             keyDown: false)

     keyDown?.flags = modifiers
     keyUp?.flags = modifiers

     keyDown?.post(tap: .cghidEventTap)
     keyUp?.post(tap: .cghidEventTap)
}
```

Additionally, for system-level actions or applications requiring shell commands, I created CommandService. This module spawns a Bash process, configures a standard output pipe, and executes received commands with optional arguments, returning the result as plain text. This mechanism is useful, for instance, for quickly launching applications, listing files, or performing other command-line accessible operations.

```
@discardableResult
func runCommand(_ command: String, arguments: [String] = []) → String {
     let process = Process
     let pipe = Pipe

     process.launchPath = "/bin/bash"
     process.arguments = ["-c", command + " " + arguments.
joined(separator: " ")]
     process.standardOutput = pipe
```

```
    process.standardError = pipe

    let fileHandle = pipe.fileHandleForReading
    process.launch()

    let data = fileHandle.readDataToEndOfFile()
    process.waitUntilExit()

    return String(data: data, encoding: .utf8) ?? "Error executing
command"
}
```

Together, these services define a complete control layer that enables the agent to observe, interpret, and interact efficiently with the macOS interface, regardless of the application in use. The modular architecture and use of standard Apple frameworks ensure compatibility and stability starting from macOS Big Sur, and allow for easy extension of capabilities in the future.

2.2 LLM Server

The LLM server is the core component responsible for processing user requests and running the LLM models in the Tecky application. Instead of relying on external services or hosting models on remote servers, I chose to implement a local server as a service using the MLX infrastructure, which enables optimized model execution on macOS. The server runs on Uvicorn, a high-performance ASGI server.

We started with the otriscon/llm-structured-output repository, which enables LLM responses to be mapped to a predefined JSON structure for server configuration. First, I installed and configured Uvicorn, an ideal ASGI server for microservices applications, allowing fast integration with LLMs. It runs on port 8080 and accepts POST requests with prompts, which are processed by the trained models and return structured JSON results.

```
uvicorn.run(
    "examples.server:app”,
    host=“0.0.0.0",
    port=args.port
)
```

To create a server that is easily distributable and macOS-compatible, I used PyInstaller to transform the Python script into a native executable. This approach removes the dependency on an external Python environment, making it possible to integrate the server directly into the native macOS application without requiring an external development setup. PyInstaller packages the entire codebase and dependencies into a single executable file that is easy to distribute and can run on any macOS system—even those without Python installed.

```
pyinstaller --onefile
             --add-data "src/examples: examples"
             --add-data "src/llm_structured_output:llm_structured_output"
             --collect-all fastapi
             --collect-all mLx_lm
             --collect-all mlx
             --collect-all tokenizers
             --collect-all sentencepiece
             --collect-all pydantic
             --collect-all uvicorn src/run_server.py
```

The LLM server runs models compatible with or converted for MLX (such as those from the mlx-community on Hugging Face), and is configured to accept prompt-completion requests and use the tool-calling functionality from LLM Structured Output (allowing the LLM to invoke functions). These requests are processed by the server, and the results are returned to the macOS application as structured JSON objects. These are then translated into a sequence of actions for the user interface.

By using Uvicorn and PyInstaller, I successfully created a fast, self-contained, and user-friendly server for running LLMs locally on macOS—without relying on external services or complex configurations. This server fully localizes request processing, ensuring the application can function entirely offline, without compromising performance or user experience.

2.3 Vision Model Server

In addition to the LLM server, Tecky integrates a Vision Model Server designed to handle visual understanding of the user interface, especially in areas where accessibility APIs fall short—such as custom-rendered views using Canvas, Metal, or non-AX UI elements. This server runs locally using FastAPI and Uvicorn, and serves a YOLO model provided by the Ultralytics library for object detection.

```
@app.post("/detect")
async def detect(req: ImageRequest):
    try:
        image_data = base64.b64decode(req.image)
        image = Image.open(io.BytesIO(image_data)).convert("RGB")
    except Exception as e:
        return {"error": f"Invalid image: {str(e)}"}
    results = model(image, verbose=False)[0]
    output = []
    for box in results.boxes.data.tolist():
        x1, y1, x2, y2, score, cls = box
        label = results.names[int(cls)]
        output.append({
            "label": label,
            "score": float(score),
            "bbox": {
                "x": float(x1),
                "y": float(y1),
                "width": float(x2 - x1),
                "height": float(y2 - y1)
            }
        })
    return output
```

To facilitate seamless integration on macOS, the server is packaged into a self-contained executable using PyInstaller, removing the need for an external Python runtime. Once running, the server listens for POST requests containing image data (screenshots), performs object detection on the input using the YOLO model, and returns bounding boxes and class labels in a structured JSON format. This enables Tecky to interpret screen content visually and interact with UI components that cannot be programmatically accessed through macOS's accessibility APIs.

On the Swift side, a specialized component called ScreenCaptureService is responsible for generating the input for the Vision Model Server. This service captures individual screenshots of each open application window, not just the full screen, by using macOS APIs such as CGWindowListCreateImage to obtain high-fidelity images of visible windows. Each screenshot is tagged with the corresponding window's metadata (e.g., application name, window ID, bounds) and sent to the Vision Server for analysis.

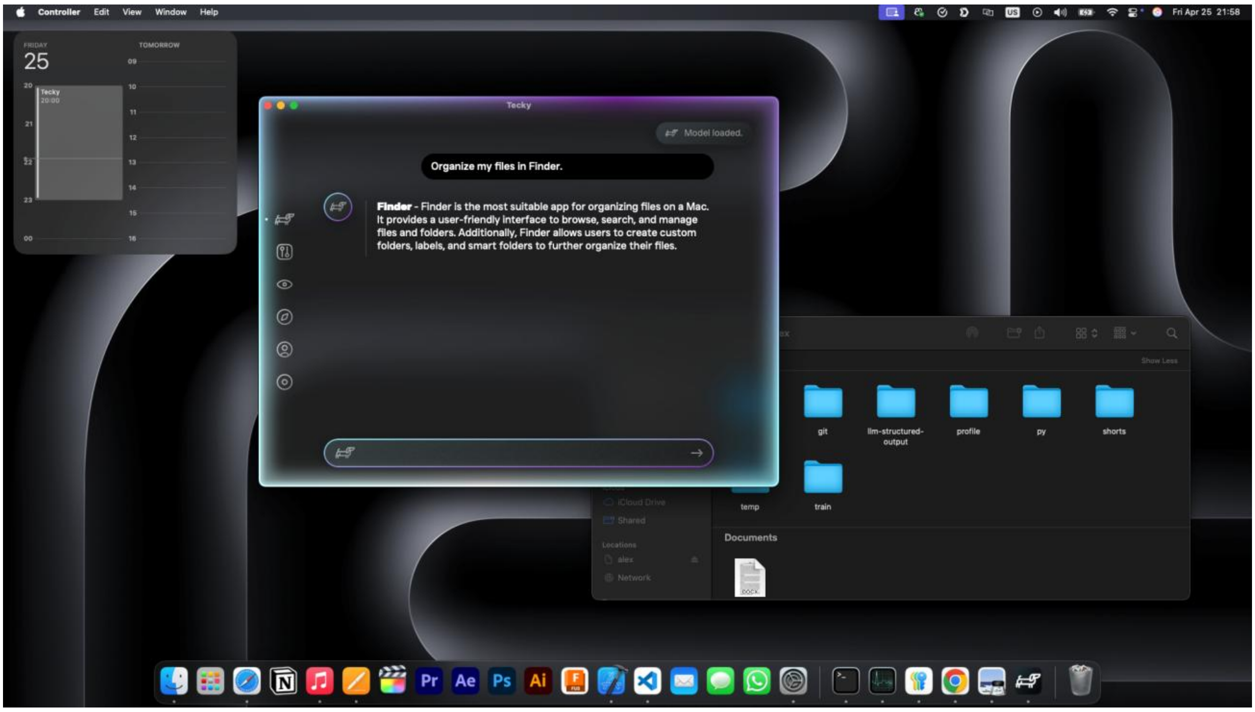
```
Func captureScreen(forApp appName: String) -> [(image: CGImage, x: CGFloat,
y: CGFloat)] {
    let options: CGWindowListOption = [.optionOnScreenOnly,
.excludeDesktopElements]
    guard let windowListInfo = CGWindowListCopyWindowInfo(options,
kCGNullWindowID) as NSArray? Else { return [] }

    var captured: [(CGImage, CGFloat, CGFloat)] = []

    for windowInfo in windowListInfo {
        guard let dict = windowInfo as? NSDictionary,
              let windowOwnerName = dict[kCGWindowOwnerName as String] as?
String,
              windowOwnerName.lowercased().contains(appName.lowercased()),
              let boundsDict = dict[kCGWindowBounds as String] as?
NSDictionary,
              let x = boundsDict[“X”] as? CGFloat,
              let y = boundsDict[“Y”] as? CGFloat,
              let width = boundsDict[“Width”] as? CGFloat,
              let height = boundsDict[“Height”] as? CGFloat,
              let windowID = dict[kCGWindowNumber as String] as? CGWindowID
else {
            continue
        }

        let bounds = CGRect(x: x, y: y, width: width, height: height)
        if let image = CGWindowListCreateImage(bounds,
.optionIncludingWindow, windowID, [.bestResolution]) {
            captured.append((image, x, y))
        }
    }
    return captured
}
```

This coordinated setup, ScreenCaptureService on the client side and the YOLO-based Vision Server on the local backend, creates a powerful visual perception layer within Tecky. It allows the agent to understand and act on any UI layout, even those dynamically rendered or inaccessible via accessibility APIs. The system remains fully offline, fast, and privacy-preserving, further enhancing Tecky's ability to operate as an intelligent, local automation assistant for macOS.

3 UI & Controller

The Tecky application adopts a minimalist and modern design, based on SwiftUI, with a focus on providing a clean and intuitive visual experience. At the core of the design is the use of a WebView, which integrates HTML, CSS, and JavaScript to render the graphical interface. The WebView provides the flexibility needed for dynamic interactions, allowing users to select AI models, view responses, and interact with the Tecky agent in an efficient and elegant manner.

A key aspect of the visual design is the use of a blur effect applied to the application's background, achieved through the NSVisualEffectView component. This component creates a transparent, aesthetically pleasing background that aligns perfectly with the macOS design language, delivering a soft blur effect that integrates naturally with the user interface without visually overloading the application. This visual depth enhances the overall sophistication without distracting from functional UI elements.



Organize my files in Finder.

Finder - Finder is the most suitable app for organizing files on a Mac. It provides a user-friendly interface to browse, search, and manage files and folders. Additionally, Finder allows users to create custom folders, labels, and smart folders to further organize their files.

Fig. 2: Tecky User Interface

In terms of application logic, Tecky.swift is the central component that connects all of the app's services and modules. It acts as an orchestrator between the UI, the LLM server, and the internal services (such as ClickerService, TyperService, and CommandService, which are part of AccessibilityService). Tecky.swift is responsible for wiring together these components and managing the data flow between them.

```
let process = Process()
process. executableURL = URL(fileURLWithPath: binaryPath)
process.arguments = arguments
process. environment = [
     "PATH": Bundle.main.bundlePath + "/Contents/MacOS",
     "HOME": NSHomeDirectory()
]

DispatchQueue.global(qos: .userInitiated).async {
     do {
          try process.run(
     } catch {
          print("Failed to run Server: \(error. localizedDescription)")
     }
}
```

When the user selects a model from the UI, Tecky.swift handles downloading the selected model via the Tecky API, ensuring it is correctly installed in the Application Support directory. Once available, Tecky.swift launches the local LLM server on localhost:8080, using the previously described server configuration.

```
func sendRequest(jsonBody: [String: Any], completion: @escaping (Data?,
URLResponse?, Error?) → Void)
     // Create request
     var request = URLRequest(url: url)
     request.httpMethod = "POST"
     request. setValue("application/json", forHTTPHeaderField: "Content-
Type")
     // Convert dictionary to JSON data
     do {
          let jsonData = try JSONSerialization.data(withJSONObject:
jsonBody, options: [])
          request.httpBody = jsonData
     } catch {
          print( "Error serializing JSON: \(error)")
          return
     }
     // Perform request
     URLSession.shared.dataTask(with: request, completionHandler:
completion).resume()
}
```

Further, Tecky.swift mediates between the WebView (which receives user input) and the LLM server. When a user submits a request, Tecky.swift constructs a structured prompt in JSON format and sends it to the LLM server. The server processes the prompt and returns a structured JSON response, which Tecky.swift then converts into concrete actions—sending them to the relevant internal services like ClickerService (for simulating clicks), TyperService (for typing text), or CommandService (for executing shell commands).

This approach enables seamless and efficient communication across all application components, making Tecky.swift the central hub of application logic. It also allows users to interact directly with the AI model, with responses quickly and accurately translated into actions that directly impact the operating system.

4 Custom Model Fine-tuning

To ensure both the language understanding and visual perception components of Tecky are finely tuned for the macOS environment, we developed custom training pipelines for both the LLM and Vision models.

For the language model, I started with a quantized version of LLaMA 3.2 8B Instruct, selected for its balance between performance and efficiency on Apple Silicon. The model was fine-tuned specifically for LLM function-based answering, aligned with the structured JSON output format required by the Tecky Controller.

The training data was constructed through a custom crawler algorithm that parsed various publicly available websites documenting macOS workflows, command-line utilities, application usage, and productivity tasks. This raw data was passed through a preprocessing script which restructured the content into dialog-style datasets—formatted as user prompts and assistant JSON responses that mirrored the expected outputs of the Tecky Controller. Essentially, the model was taught to translate natural language instructions into machine-executable structured responses.

The dataset generation and fine-tuning loop were executed fully locally, using the same LLM server Tecky uses in production, allowing us to test and iterate rapidly. This ensured that the LLM not only understands context-specific requests but also outputs directly executable actions in a format compatible with the tool-calling infrastructure.

For the computer vision component, I developed a dataset and training loop tailored to UI detection on macOS. Using an automated screenshotting script, I collected thousands of application window images from real macOS sessions. A portion of these screenshots were manually annotated using LabelImg, focusing on common UI elements such as buttons, inputs, toggles, and toolbars.

With this initial model trained, I constructed a self-bootstrapping pipeline: the trained YOLO model was used to detect elements in new screenshots, generating pseudo-labels that were then validated and refined through heuristic filters. These automatically labeled images were added to the dataset, and the model was retrained on this expanded corpus. This process was repeated iteratively, creating a continuous learning cycle that scaled the dataset and improved model accuracy with minimal manual intervention.

The result is a custom YOLO model, highly optimized for recognizing visual UI patterns on macOS, even in apps with non-standard or custom-rendered interfaces. Combined with the LLM, this enables Tecky to perceive, understand, and act on both semantic and visual cues with high precision.

5 Limitations and Conclusion

Despite the custom training pipelines, data scarcity remains a core limitation. While the fine-tuned LLM and Vision models perform well on common macOS workflows and standard UI layouts, the diversity and complexity of real-world applications require significantly more data to achieve robust generalization.

For the LLM, the rephrased dataset derived from public documentation provides a strong baseline, but it still lacks full coverage of edge cases, nuanced user intentions, and rare workflow patterns. This means that in certain contexts, the model might generate structurally valid but semantically incorrect actions, or misinterpret ambiguous prompts, especially for lesser-known apps or non-standard user phrasing.

On the Vision side, the challenge is even more pronounced. Every new application introduced into Tecky's automation scope may feature a unique layout, custom-drawn components, or dynamic UI behavior that the model has not previously seen. As a result, achieving reliable automation requires a large number of labeled images per application, which introduces a significant bottleneck. Although the automated labeling pipeline helps scale data collection, manual verification is still essential to maintain accuracy and avoid compounding errors.

Together, these challenges mean that predictive inaccuracies may occur—such as clicking the wrong element, mistyping inputs, or failing to detect critical UI components. Overcoming this will require continuous dataset expansion, community-driven contribution pipelines, and potentially incorporating few-shot or on-device fine-tuning strategies to personalize behavior for each user and application.

Acknowledgement: This work was supervised by Professor Delilah Florea, from Samuel von Brukenthal National College Sibiu, Romania.

References

[1] LLM Structured Output: JSON Schema, Function Calling, Tools, https://github.com/otriscon/llm-structured-output/

[2] Turn images into AI to get useful insights with no code, https://www.ultralytics.com/

[3] A fast, open source web browser engine. https://webkit.org/

[4] FastAPI framework, https://fastapi.tiangolo.com/

[5] Application Services, https://developer.apple.com/documentation/applicationservices
[6] Core Graphics, https://developer.apple.com/documentation/coregraphics
[7] PyInstaller Manual, https://pyinstaller.org/en/stable/
[8] Uvicorn, https://www.uvicorn.org/
[9] MLX Community, https://huggingface.co/mlx-community

Alexandru Emil SOFONEA
Samuel von Brukenthal National College Sibiu
ROMANIA
E-mail: alex@alexsofonea.com

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Web-based System for Dynamic Text Rendering and Interactive Content Management in Full-screen Environments in the Music Industry

Kristian P. Spasov, Martin S. Dzhurov, Serkan H. Sadulov

Abstract

This scientific report explores the design, development, and implementation of a web-based system created for dynamic text rendering tailored specifically for full-screen usage in musical performances. Addressing existing limitations of conventional methods, such as printed song lyrics and manually managed teleprompters, the proposed solution offers real-time interactive content management, automatic resizing of text to optimally fit various display devices, and intuitive control via specialized hardware – foot switches and numeric keypad interactions. The developed system leverages modern web technologies (HTML5, CSS3, JavaScript, PHP, MySQL, AJAX, JSON, Bootstrap, and jQuery) to ensure efficient, seamless, and intuitive user experience. This paper provides a comprehensive analysis of existing solutions, clearly defines the objectives and tasks of the new system, and presents the architectural decisions, functional design, data organization strategies, and detailed descriptions of the implemented functionalities. The report concludes by evaluating the effectiveness of the proposed solution and identifying directions for future enhancements.

1 Introduction to the problem

In recent years, the rapid advancement of technology has profoundly reshaped numerous fields, drastically improving efficiency, accuracy, and user experience. However, within the realm of live musical performances, many musicians continue to rely heavily on printed materials for lyrics and chords, creating practical inconveniences during live performances. This reliance on traditional, paper-based methodologies complicates spontaneous changes in the performance repertoire, as musicians must manually search through extensive printed materials, causing unintended pauses, disrupting audience engagement, and detracting from the overall performance. Furthermore, conventional solutions like karaoke software **[1-5]** or stage prompters **[7]** either fail to offer flexibility for real-time changes or rely heavily on third parties, increasing the risk of human error and further complicating live performances.

Recognizing these limitations, there is a clear and immediate *need for a dynamic solution, capable of managing musical content interactively and intuitively*. Such a system would eliminate the reliance on cumbersome paper-based documentation by digitizing song lyrics and chords, ensuring swift retrieval, dynamic adjustment, and immediate accessibility directly on screen. Additionally, by implementing automatic text resizing optimized for full-screen viewing, the proposed solution addresses critical readability issues associated with traditional methods, ensuring maximum visibility and ease of use in dynamic stage environments. This web-based approach not only streamlines content management but also significantly enhances performers' autonomy, facilitating seamless transitions, spontaneous repertoire changes, and ultimately improving both audience and performer satisfaction.

2 Current solutions

The analysis of current solutions highlights various existing technologies designed to facilitate dynamic text display for live musical performances. Existing solutions range from dedicated teleprompter software specifically tailored for musicians, such as SingerPro [1], Baraoke [2], and LivePrompter [3], to broader multimedia presentation systems like OpenLP and EasyWorship [4, 5], as well as versatile browser-based applications like EasyPrompter [6] and robust hardware-integrated solutions, such as the BLACK BOX Prompter [7]. While these solutions offer various levels of usability, flexibility, and functionality, each presents limitations, particularly concerning automated text resizing, platform independence, and ease of real-time interaction.

2.1 SingerPro Music Teleprompter

SingerPro Music Teleprompter is a software application designed to display song lyrics, chords, and musical notes directly onto screens, making them easily accessible for live performers. It effectively replaces traditional paper-based methods, helping musicians maintain eye contact with their audience. Key features include adjustable font size and colour, scrolling speed, and the ability to import song texts and chords. The portability and compatibility with mobile devices make it highly versatile for live performances. However, the application lacks advanced flexibility for dynamic changes during performances, relying heavily on manual controls which can lead to errors if managed improperly [1].

- **Pros:** User-friendly interface, customizable visual elements, portable design, mobile compatibility, and audio playback functionality.
- **Cons:** Dependent on manual interactions for dynamic adjustments during live performances; potential musician’s distraction due to continuous manual operation.

2.2 Baraoke

Baraoke combines lyric prompting and audio playback, specifically tailored for karaoke and live bar performances. This software supports various audio and video file formats and offers advanced features, such as synchronized lyric scrolling, playlist management, tempo, pitch adjustments, and audio effects processing. The ability to control the software using MIDI (Musical Instrument Digital Interface) controllers, touchscreens, or foot switches adds significant versatility. Its main strength lies in the combination of audio effects processing and lyric synchronization, making it highly suitable for dynamic live scenarios [2].

- **Pros:** Versatile interface compatible with multiple input devices, audio effects processing, real-time playlist management, and continuous software updates.

- **Cons:** Limited support for automated text resizing; primarily focused on bar and karaoke scenarios, potentially less suitable for complex stage performances.

2.3 LivePrompter

LivePrompter is specifically crafted to aid musicians during live performances, rehearsals, and studio recordings by displaying synchronized song lyrics, chords, and notes. This solution enables automatic text scrolling aligned with musical performances, enhancing ease of use. Its compatibility with chordpro files ensures quick integration with commonly used music documentation formats [3].

- **Pros:** Automated text scrolling synchronized with music playback, lightweight and portable design suitable for stage use, support for various file formats including chordpro.
- **Cons:** Not web-based; limited cross-platform compatibility (primarily Windows-based), which restricts its accessibility and flexibility in diverse performance environments.

2.4 OpenLP and EasyWorship

OpenLP and EasyWorship are multimedia presentation management solutions popular among religious services, seminars, and multimedia presentations. Both solutions offer functionalities to manage songs, verses, and multimedia content. They provide a robust interface to integrate various multimedia elements, supporting external input from remote controls. Their strength lies in broad multimedia capabilities and easy integration with media libraries [4, 5].

- **Pros:** Excellent multimedia management capabilities, multi-screen setup, and user-friendly interfaces suitable for varied presentation contexts.
- **Cons:** Lack specialized tools for automatic text scaling, limited direct control for live dynamic adjustments specific to musical performances.

2.5 EasyPrompter

EasyPrompter provides a web-based teleprompting solution suitable for various scenarios, including broadcasting, speeches, and film production. It is highly accessible through browsers and supports dual-screen setups, script management, and smooth scrolling adjustments. The web-based nature ensures wide compatibility across multiple platforms and easy access without installation requirements [6].

- **Pros:** Accessible through browsers, easy script management, adjustable scrolling features, dual-screen support.
- **Cons:** Lacks specific features tailored to musicians, such as chord integration or dynamic, real-time text resizing based on content.

2.6 BLACK BOX Prompter

The BLACK BOX Prompter is a specialized teleprompter system designed to enhance live musical performances by providing musicians with clear, real-time access to lyrics, chords, and sheet music. Featuring a 24-inch shatter-proof display, the device is built for durability and stage use. It supports various file formats, including text-based and PDF files, allowing for versatile content integration. The prompter includes a three-pedal foot controller for hands-free operation, enabling musicians to navigate through their content seamlessly during performances. Additional features include Wi-Fi connectivity, an internet browser for quick access to online resources, and smart scrolling with adjustable speeds. While the BLACK BOX Prompter offers a comprehensive solution for on-stage prompting, its hardware-centric design may limit portability compared to software-based alternatives, and its premium build quality reflects a higher price point [7].

- **Pros:** Durable 24-inch shatter-proof display, supports multiple file formats, hands-free operation with foot switches, Wi-Fi connectivity, smart scrolling, and internet browsing capabilities.
- **Cons:** Hardware-centric design may limit portability; higher price point compared to software-based solutions.

Table 1: *Comparative Table*

Functionality	**SingerPro**	**Baraoke**	**LivePrompter**	**OpenLP & EasyWorship**	**EasyPrompter**	**BLACK BOX Prompter**
Web-based	❌	❌	❌	❌	✅	❌
Automated text resizing	❌	❌	❌	❌	❌	❌
Manual scrolling	✅	✅	✅	✅	✅	✅
Automatic synchronized scrolling	❌	✅	✅	✅	✅	✅
Chord and lyrics integration	✅	✅	✅	✅	❌	✅
Multi-screen support	✅	✅	✅	✅	✅	✅
Specialized hardware integration	✅	✅	✅	❌	❌	✅
Real-time playlist editing	✅	✅	✅	✅	✅	✅
Durable hardware design	❌	❌	❌	❌	❌	✅
Internet browsing capabilities	❌	❌	❌	❌	❌	✅

The table of comparison between the different software solutions is presented on Table 1. It can be noted that BlACK BOX Prompter has the most features. The one that conforms the least to the criteria are EasyPrompter and OpenLP & EasyWorship.

3 The proposed solution

3.1 Software system architecture

The proposed software solution features a sophisticated and carefully structured architecture, embracing modern web technologies and adhering to the Model-View-Controller (MVC) design pattern, ensuring clear separation and optimal maintainability. The application was developed using HTML5, CSS3, and JavaScript libraries, such as jQuery and AJAX on the frontend, while the backend is implemented with PHP for server-side logic and MySQL for robust relational data management. The real-time communication layer between the server side and the client sides is implemented via WebSockets, leveraging PHP Ratchet, a powerful PHP-based WebSocket server framework. Ratchet efficiently manages persistent socket connections, significantly outperforming older methods, such as HTTP Long Polling by reducing latency, minimizing data overhead, and eliminating unnecessary repeated server polling. Complementing Ratchet on the client side is the native JavaScript WebSocket API, enabling efficient, low-latency, bidirectional communication channels that maintain continuous connectivity, crucial for live interactive sessions [8].

Moreover, the client-side implementation distinguishes between two specialized interfaces, each optimized for distinct user roles. The first interface is specifically designed for musicians performing on stage, emphasizing simplicity, clarity, and instant readability, ensuring performers can quickly interpret and follow lyrics, chords, and performance cues. This interface dynamically renders and resizes text, automatically optimizing visibility for full-screen displays (Fig.1). The second interface serves the backstage crew and technical support, providing extensive control and monitoring functionalities. It enables real-time tracking of the current playlist status, seamless updates to the playlist during performances, and a secure messaging capability to communicate directly with musicians on stage through the WebSocket connection. This two-tier client-side implementation significantly enhances the overall effectiveness of live musical performances, facilitates swift adaptation to unexpected situations, and ensures smooth coordination among all team members involved.

3.2 Automatic text-resizing algorithm and full screen visualization

The automatic text-resizing algorithm implemented in the proposed system is specifically designed to optimize the visibility and readability of song lyrics, chords, and related textual information dynamically on varying screen sizes and resolutions. The algorithm operates client-side, using JavaScript and jQuery to dynamically adjust the font size and layout based on the current viewport dimensions. Initially, the maximum font size (`maxSize`) is retrieved from server-side PHP settings stored in a MySQL database and loaded via an AJAX request. The client application then attempts to render the song content within the available display space (view-port boundaries). The core logic employs a multi-step approach involving two primary functions: `checkMaxRows()` and `checkMaxColsAndRows()`. The `checkMaxRows()` function calculates the maximum number of text rows that can fit within a column without overflowing vertically by incrementally appending rows until the container's height limit (`$(".content").outerHeight()`) is exceeded. The iterative procedure of this function ensures efficient measurement and precise fitting.

The core implementation of the row measurement function is shown in Listing 1.

Listing 1: Source code of checkMaxRows function

```
function checkMaxRows(startIndex, endIndex, textArray, column){
   var rows = 0;
    var mainBody = "<div class='curCol"+column+"' style='display: grid; grid-auto-flow: row; height: 100%;'>";
   var maxHeight = $(".content").outerHeight();
   for(var i=startIndex; i<endIndex; i++){
      mainBody += textArray[i];
      $(".content").html(mainBody + "</div>");
      if($(".curCol"+column).outerHeight() > maxHeight)
         break;
      rows++;
   }
   return rows;
};
```

Following this, the checkMaxColsAndRows() function repeatedly calls checkMaxRows() to determine how many columns of text the display can accommodate. It generates an array representing the number of rows per column, ensuring optimal use of horizontal space. The implementation is shown in Listing 2.

Listing 2: Source code of checkMaxColsAndRows function

```
function checkMaxColsAndRows(textArray){
   var col = [];
   var repeat = 0;
   var countCols = 0;
   while(repeat < textArray.length){
      col[countCols] = checkMaxRows(repeat, textArray.length, textArray, 0);
      repeat += col[countCols];
      countCols++;
   }
   return col;
};
```

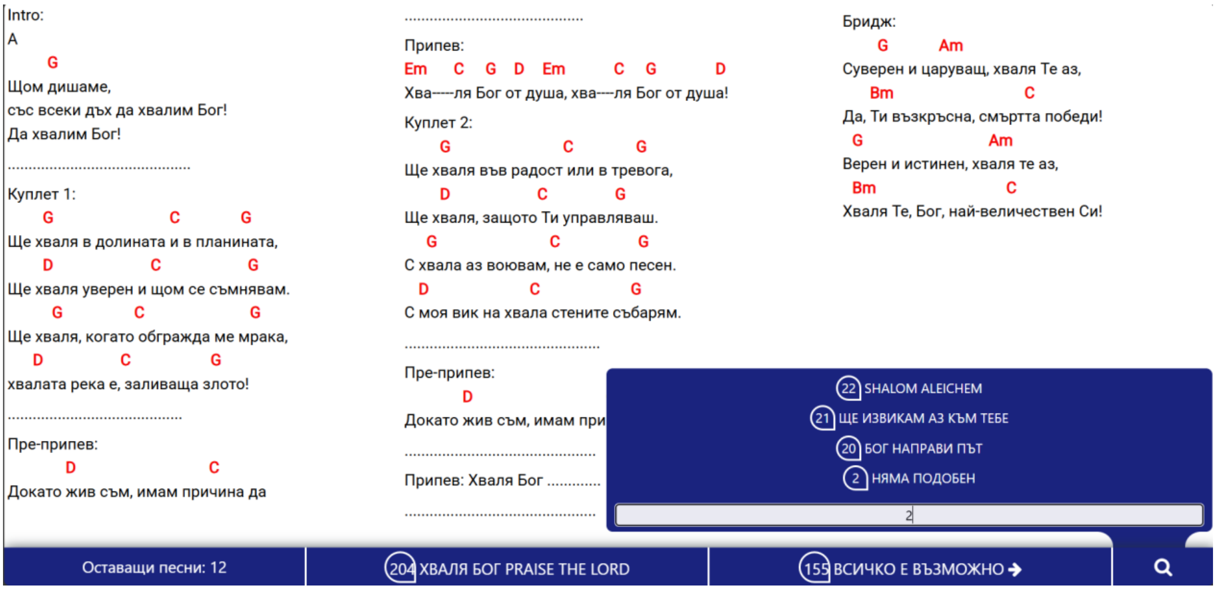
Once the grid structure is determined, the resizeText function integrates these calculations, dynamically resizing the font until the entire content perfectly fits the available screen width and height without overflow. The resizing loop progressively reduces the font size until horizontal scrolling is no longer necessary. The implementation of the logic is shown in Listing 3.

Listing 3: Source code of displayText & resizeText functions

```
function displayText(gridStructure, textArray){
   var textBody = "";
   var count = 0;
   for(var i=0; i < gridStructure.length; i++){
         textBody += "<div class='curCol"+i+"' style='display: grid; grid-auto-flow: row; height: fit-content;'>";
      for(var j=0; j<gridStructure[i]; j++){
         textBody += textArray[count++];
      }
      textBody += "</div>"
   }
   return textBody;
};
```

```
function resizeText(textArray){
   var display = checkMaxColsAndRows(textArray);
   var textToDisplay = displayText(display, textArray);
   $(".content").html(textToDisplay);
   while($(".content").outerWidth() < $(".content").prop('scrollWidth')){
      currentSongSize--;
      $(".content").css("font-size", currentSongSize);
      display = checkMaxColsAndRows(textArray);
      textToDisplay = displayText(display, textArray);
      $(".content").html(textToDisplay);
   }
};
```

If initially rendered content exceeds the width of a full HD (High-definition) (1920px×1080px) screen, the algorithm reduces the currentSongSize() incrementally until the textual content fits without horizontal overflow. Each iteration precisely recalculates the grid dimensions, achieving optimal balance between readability and available screen space. Additionally, the event handler attached to the window resize event ensures that dynamic adjustments occur seamlessly, maintaining optimal visualization during real-time viewport changes. A practical rendered example is shown in Fig.1.



Intro:
A
G
Щом дишаме,
със всеки дъх да хвалим Бог!
Да хвалим Бог!

....

Куплет 1:
G C G
Ще хваля в долината и в планината,
D C G
Ще хваля уверен и щом се съмнявам.
G C G
Ще хваля, когато обгражда ме мрака,
D C G
хвалата река е, заливаща злото!

....

Пре-припев:
D C
Докато жив съм, имам причина да

....

Припев:
Em C G D Em C G D
Хва----ля Бог от душа, хва---ля Бог от душа!

Куплет 2:
G C G
Ще хваля във радост или в тревога,
D C G
Ще хваля, защото Ти управляваш.
G C G
С хвала аз воювам, не е само песен.
D C G
С моя вик на хвала стените събарям.

....

Пре-припев:
D
Докато жив съм, имам при

....

Припев: Хваля Бог ....

....

Бридж:
G Am
Суверен и царуващ, хваля Те аз,
Bm C
Да, Ти възкръсна, смъртта победи!
G Am
Верен и истинен, хваля те аз,
Bm C
Хваля Те, Бог, най-величествен Си!

Fig.1: *Stage view of loaded and resized lyrics*

3.3 Hardware system solution

The hardware architecture accompanying the proposed text-rendering system encompasses carefully selected and specialized components tailored explicitly for live stage performances. Central to this setup is the primary visualization device, typically implemented as a large Full-HD or 4K display strategically positioned on stage and connected to a dedicated stage computer or laptop running the web-based application. The stage computer (in the current solution it is All-In-One PC) is primarily controlled by specialized foot switches (Fig.2) operated by the lead musician, enabling hands-free navigation through song sections, playlist progression, and real-time song selection without interrupting their musical performance. Additionally, the lead musician can use a separate numeric keypad (numpad) connected via USB, which provides extended functionality beyond basic navigation, such as the ability to rapidly add songs directly into the active playlist or send predefined text messages instantly to other musicians or the backstage crew. To ensure consistent visibility for all performers, the stage computer's screen content is duplicated across multiple monitors connected via HDMI (High-Definition Multimedia Interface) splitters, allowing each stage performer clear access to lyrics and chords in real time.

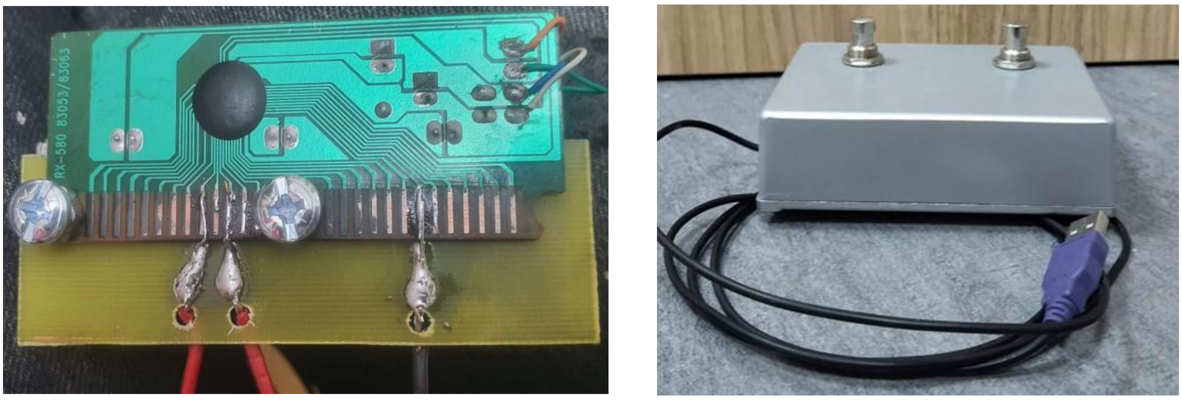
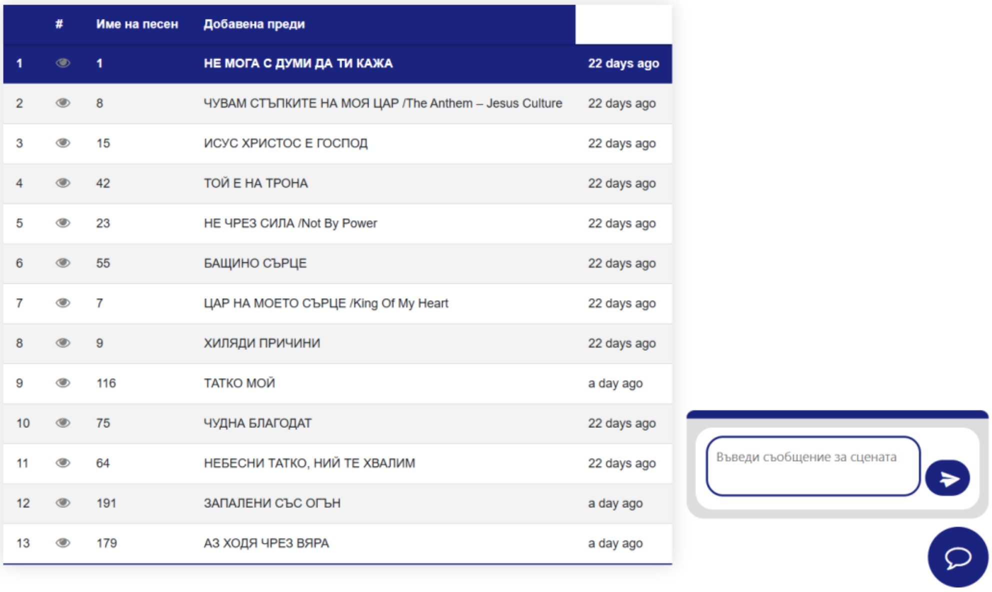


Fig.2: *Hardware foot switch prototype from old USB keyboard*

The backstage crew interacts with the system through a separate web interface (Fig.3), accessible via the web-based application on a device independent of the main stage computer. Unlike the stage equipment, the backstage control interface is completely detached from the stage hardware and is accessible via any browser-enabled device connected to the same network (Fig.4). Leveraging the established WebSocket-based communication infrastructure managed through the PHP Ratchet WebSocket server, the backstage panel offers functionality to monitor real-time updates on playlist progression, to send custom messages directly to musicians during performances, and to perform remote adjustments to the playlist without interrupting the performance flow. This architecture, with clearly defined and separated user interfaces, facilitates coordinated, real-time communication and adjustments between performers and backstage crew, substantially enhancing both the flexibility and effectiveness of live musical presentations.



	#	Име на песен	Добавена преди	
1	👁	1	НЕ МОГА С ДУМИ ДА ТИ КАЖА	22 days ago
2	👁	8	ЧУВАМ СТЪПКИТЕ НА МОЯ ЦАР /The Anthem – Jesus Culture	22 days ago
3	👁	15	ИСУС ХРИСТОС Е ГОСПОД	22 days ago
4	👁	42	ТОЙ Е НА ТРОНА	22 days ago
5	👁	23	НЕ ЧРЕЗ СИЛА /Not By Power	22 days ago
6	👁	55	БАЩИНО СЪРЦЕ	22 days ago
7	👁	7	ЦАР НА МОЕТО СЪРЦЕ /King Of My Heart	22 days ago
8	👁	9	ХИЛЯДИ ПРИЧИНИ	22 days ago
9	👁	116	ТАТКО МОЙ	a day ago
10	👁	75	ЧУДНА БЛАГОДАТ	22 days ago
11	👁	64	НЕБЕСНИ ТАТКО, НИЙ ТЕ ХВАЛИМ	22 days ago
12	👁	191	ЗАПАЛЕНИ СЪС ОГЪН	a day ago
13	👁	179	АЗ ХОДЯ ЧРЕЗ ВЯРА	a day ago

Fig.3: *Backstage view of loaded songs in a current playlist*

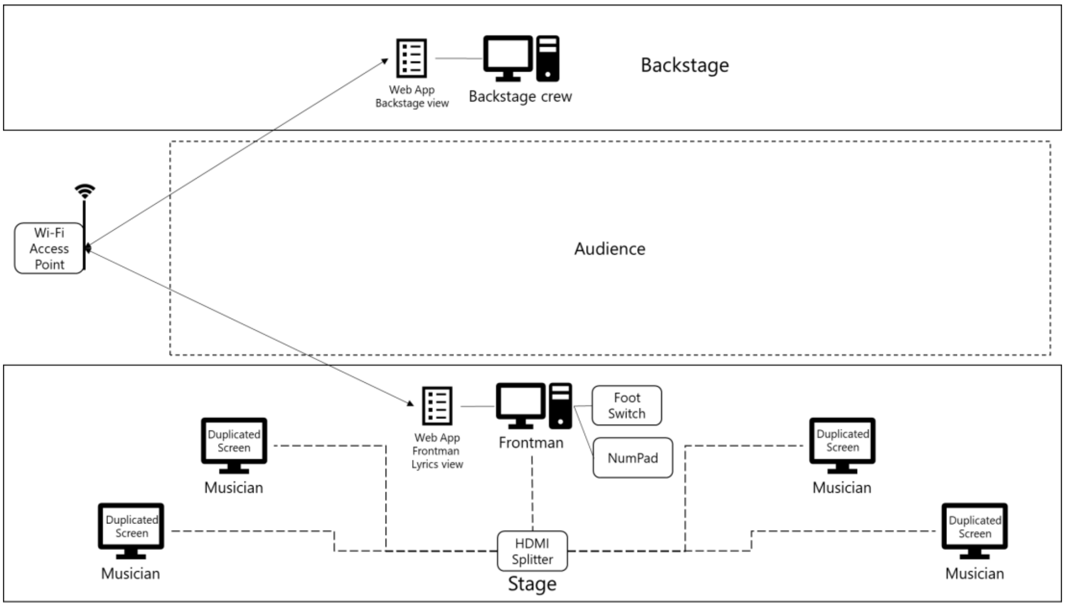


Fig.4: *Schematic representation of the theoretical framework*

4 Practical application

4.1 Observed behaviour and changes during the iterative development process

During experimental evaluations, the automatic text-resizing algorithm exhibited specific limitations when handling exceptionally lengthy song lyrics or chord notations. In scenarios involving songs with extensive text content or numerous verses, the algorithm often needs multiple recalculations to achieve an optimal font size, thus potentially causing visible delays during the resizing process. Although the implemented algorithm effectively adapts to common content lengths, it faces difficulties with exceptionally long song texts, where iterative recalculations become computationally intensive and inefficient. For instance, testing revealed that when loading content exceeding typical display dimensions (such as lyrics surpassing standard viewport heights by multiple screen lengths), the resizing logic might require more iterations, resulting in noticeable latency and reduced responsiveness. Future improvements should include predictive optimization techniques, such as initial estimation based on historical data or adaptive caching mechanisms to significantly reduce computational overhead and ensure consistently fast rendering performance.

Additionally, the current automatic resizing logic encounters limitations when dealing with constrained display environments characterized by limited columns and rows. Specifically, in performances using smaller screens or monitors with lower resolutions, the algorithm must substantially reduce font size to accommodate the entire content within the restricted space. This excessive downsizing negatively impacts readability, diminishing the legibility and usability of the displayed text for performers. This limitation poses a significant practical issue, particularly for stage setups with restricted space or distant viewing positions. In future improvements, the algorithm should incorporate advanced content segmentation strategies, such as line splitting, column-balancing heuristics, or scrolling capabilities, ensuring readability without sacrificing visibility or compromising user experience. Implementing predictive text-layout strategies or adaptive pagination could also effectively manage such spatial constraints, offering enhanced readability even in limited-display scenarios.

4.2 Practical implementation

The proposed solution is deployed (Fig.4), and it is working successfully in real industrial environment in the span of 2 years, as of the date of this paper. The system, both hardware and software are deployed in "Church of God of Prophecy" – Ruse, Bulgaria, and it is used during worships each Sunday. The practical deployment of the system is shown on Fig.5.

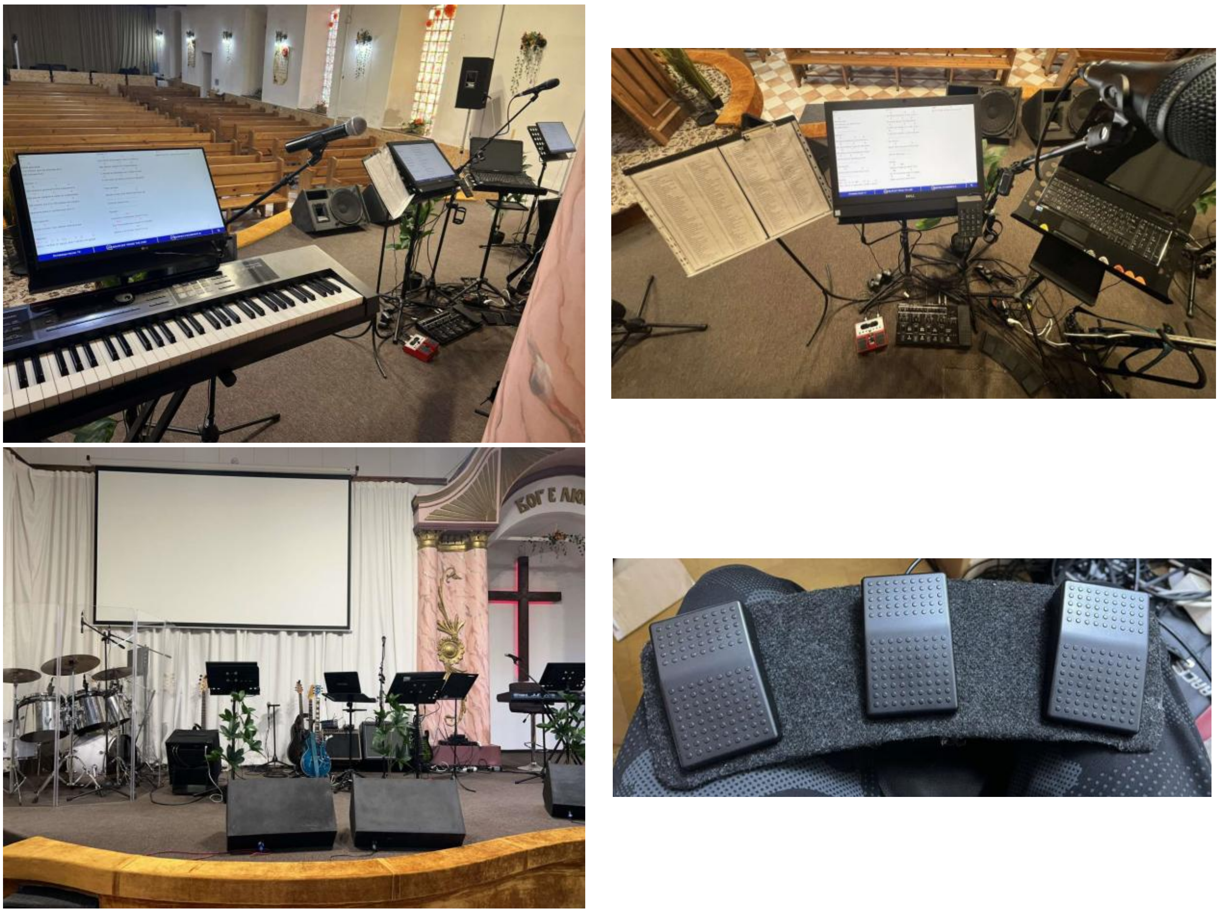


Fig.5: *Practical deployment of the argued solution*

5 Conclusion

In summary, the proposed automatic text-resizing and full screen visualization algorithm, effectively solves the common issue of dynamically adapting textual information for varying resolutions, offering a robust, automated solution specifically suited to live-performance scenarios.

In conclusion, this paper presents an advanced web-based solution designed specifically to address the limitations of existing text visualization methods in live musical performance settings. By integrating latest web technologies (HTML5, CSS3, JavaScript, PHP, MySQL, Ratchet WebSockets), specialized hardware interactions (foot switches and numeric keypads), and a sophisticated automatic text-resizing algorithm, the proposed system significantly enhances live performance efficiency and reliability. Although existing solutions partially address content management and real-time interactivity, the current work successfully combines automated resizing, dedicated hardware control, and dual-interface designs for musicians and backstage crew, addressing previously unresolved challenges.

Nevertheless, future developments should focus on optimizing performance for extensive textual content, refining computational efficiency, and addressing spatial constraints on limited-resolution displays, thereby ensuring improved scalability and enhanced usability. The proposed solution establishes a strong foundation for future research and practical innovations in real-time dynamic text rendering for interactive web environments, promising continued improvements and wider applicability across various performance contexts.

Acknowledgement: This work was supervised by chief assistant Vasil Kozov, Ph.D. from University of Ruse “Angel Kanchev” – Natural Sciences and Education Faculty – Informatics and Information Technologies Department.

References

[1] SingerPro Music Teleprompter, Official Website. Available online: https://play.google.com/store/apps/details?id=com.easyapps.singerpro (accessed 9 March 2025).
[2] Baraoke, Official Website. Available online: https://www.baraoke.co.uk (accessed 9 March 2025).
[3] LivePrompter, Official Website. Available online: https://www.liveprompter.com (accessed 9 March 2025).
[4] OpenLP, Official Website. Available online: https://openlp.org (accessed 9 March 2025).
[5] EasyWorship, Official Website. Available online: https://easyworship.com (accessed 9 March 2025).
[6] EasyPrompter, Official Website. Available online: https://www.easyprompter.com (accessed 9 March 2025).
[7] BLACK BOX Prompter, Official Website. Available online: https://blackboxprompter.com (accessed 9 March 2025).
[8] PHP WebSockets With Ratchet, Medium. Available online: https://medium.com/@winni4eva/php-websockets-with-ratchet-5e76bacd7548 (accessed 9 March 2025).

Kristian P. Spasov
University of Rousse “Angel Kanchev”
Faculty of Natural Sciences
8 Studentska str., POB 7017, Ruse
BULGARIA
E-mail: kspasov@uni-ruse.com

Martin S. Dzhurov
University of Rousse “Angel Kanchev”
Faculty of Natural Sciences
8 Studentska str., POB 7017, Ruse
BULGARIA
E-mail: mdzhurov@uni-ruse.com

Serkan H. Sadulov
University of Rousse “Angel Kanchev”
Faculty of Natural Sciences
8 Studentska str., POB 7017, Ruse
BULGARIA
E-mail: ssadulov@uni-ruse.com

Organ 3D Model Reconstruction using Point Clouds

Matei-Cristian Steavu

Abstract

There are multiple techniques when trying to reconstruct the 3D model of an organ, as well as multiple formats in which to process all this data. Each of these methods is unique and has its own strength and weaknesses. This paper's scope is to describe the best format to use in the context of 3D bioprinting, comparing each method's strengths as well as to describe the use of a technique generally found in geographic information systems, namely Delaunay triangulation.

1 Introduction

The motivation for this paper stems from an interest in 3D bioprinting and 3D printing in general. When searching for a bioprinting technique that has enough resolution for genuine artificial organs there are no standard methods to save data for G-Code generation or post processing methods for volumetric 3D printing.

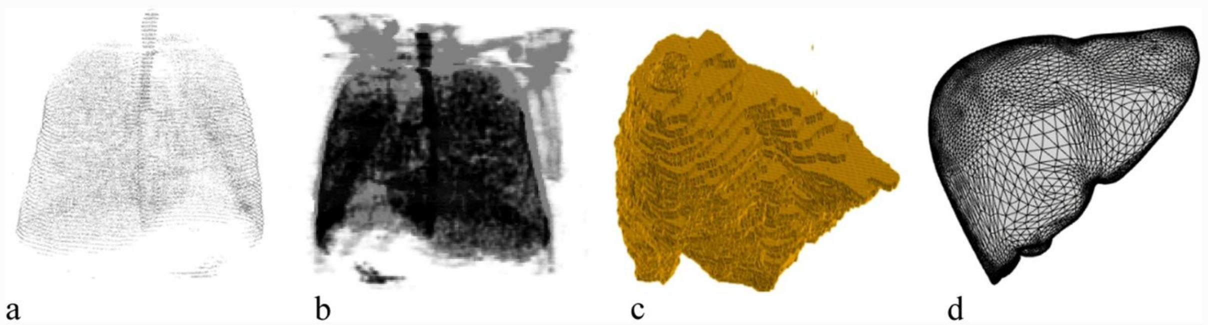
The standard file format for organs, dcm, is used to share data, but a method to share 3D models of organs is still unheard of. Thus, a standard method for generating a 3D model from dcm files is necessary.

Generally, when trying to reconstruct organs one either generates the point clouds of an object, uses the marching cubes algorithm or surface triangulation with key points. Each of these has its own advantages and disadvantages those will be presented below.

The method used in this paper is 3D object generation through point clouds, using 16-bit Canny edge detection with Delaunay triangulation. Usually, the most used method is image processing is Canny edge detection, the standard being 8-bit Canny (used for jpg and png files, not for dcm files), while Delaunay triangulation is a process generally used in geographic information systems (GIS [5]). The original contribution of this paper is the use of a method from GIS generation in organ 3D model reconstruction. Due to recent advances made by NVIDIA, a suitable upgrade to our method would be using their MESHTRON [2] model.

2 General Approaches

While organ data is shared in the format of dcm files (independent from what hardware they were obtained from), with most databases having dcm files there are a lot of approaches when trying to reconstruct the 3D model of the organ. Some of these include generation of point clouds, mesh cube generation and triangulation of important points.



a Point cloud from CT of lungs; **b** Point cloud reconstructed from CT scans of rib cage (gray) and lungs (black); **c** Coarse liver reconstructed from 28 CT scans and triangulated using marching cubes algorithm; **d** Smooth liver surface triangulation of key points extracted from 56 CT scans

Fig. 1. Processing methods [1]

Yet, what we specifically look for when discussing bioprinting is which method offers the best resolution, the best scalability for future advancements in 3D bioprinting and uses the least amount of computing power (table 1, [1]).

	Point Clouds	Marching Cubes	Surface Triangulation
Resolution	Limited by dcm resolution	Limited by iteration steps	Limited by point extraction
Scalability	Occupies more memory	Necessitates know normals for planes	Needs more computing power
Computing power	Depends on memory usage.	Multiple iterations	Needs more computing power

Table 1.

Since point clouds seem to be limited by the memory usage and existing methods to construct objects from point clouds; it seems to be the most fitting approach, even though surface triangulation is a close contender.

Now, what we wanted to test is whether using point clouds in addition to an algorithm that turns these point clouds into the 3D printable object would be any better than existing methods. Here, there are also multiple options such as ConvONet, Points2Surf [7] and MESHTRON[2]. Models such as Points2Surf were immediately ruled out as they did not have the performance needed, producing holes in meshes. Thus, Delaunay algorithm was chosen to connect the points from a point cloud in the hopes of creating a high-resolution 3D model of the file.

Since starting to work on this project, MESHTRON came out, a model that seems to outperform all variants in this task, creating high resolution with better topology, better than triangulation, but more work is needed to asses its usefulness in this usecase.

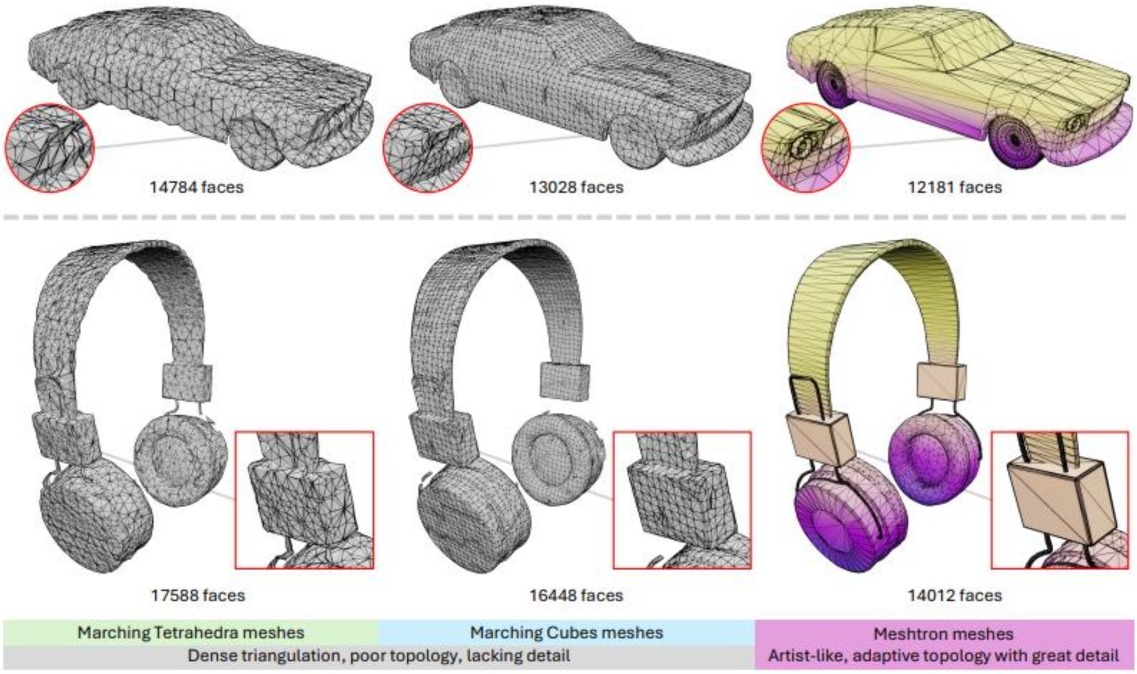

14784 faces
13028 faces
12181 faces
17588 faces
16448 faces
14012 faces
Marching Tetrahedra meshes
Marching Cubes meshes
Meshtron meshes
Dense triangulation, poor topology, lacking detail
Artist-like, adaptive topology with great detail

Fig. 2. Meshtron [2]

3 Implementationz

When it comes to how we implemented everything, we used the python programming language as it has the necessary libraries to work with all these files. Combined with its simplicity it seemed to be the best option. In addition to python we use the “pydicom” library for working with dcm files, opencv to work with jpeg’s and png’s, a dataset with normal brain MRI slices [7], Canny edge detection (16 bit canny with dcm files and opencv canny with jpeg’s and png’s) and “scipy” for Delaunay triangulation.

Now, the code starts with extraction of the data from either jpegs or dcm files. This is done with the following piece of code. This code extract all the data from dcm files or other files and puts all the pixel data in an 3D array with each pixel. The pixel data from jpg files is turned into grayscale as the canny algorithm can only process grayscale images.

```
def extract_data_files_file_type(self, folder_directory, file_type):
    if os.path.exists(folder_directory:
        i=0
        for name in os.listdir(folder_directory):
            if pathlib.Path(folder_directory+name).suffix == file_type
                #self.show_image(folder_directory+name)
                if file_type == ".dcm":
                    dicom = pydicom.dcmread(folder_directory + name)
                    self.Data_set_file[i] = dicom.pixel_array
                    self.Data_set_file[i] = canny_edge_detection(self.Data_set_file[i])
                    #self.plotter(self.Data_set_file[i])
                else:
                    image = openCV.imread(folder_directory + name)
                    edge = np.uint8(self.rgb2grayscale(image))
                    self.Data_set_file[i] = openCV.Canny(edge, 100, 200)
                    #self.plotter(self.Data_set_file[i])
                i=i+1
        self.Data_numbers = i
    else:
        print("File directory does not exist.")
```

After this step, the files are processed by Canny edge detection algorithms [3][4], 16-bit canny for dcm and opencv canny for the other file types. This process turns the dcm files from the one on the left into the one on the right. This approach has its drawbacks, as finding the right thresholds for the detection to look properly can be rather difficult. Another important fact is that usually this algorithm should be used with segmentation tools (this is a future update that will be added).

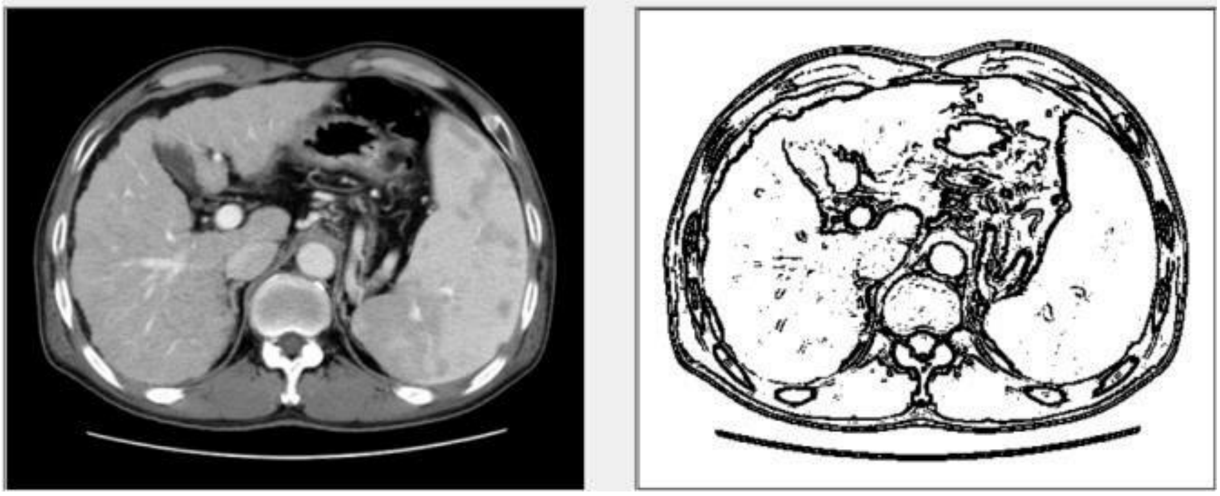


Fig. 3. Canny edge detection on dcm file

The canny edge detection algorithm is the usual one, except for the fact that calculations are done on eiter 64 bits (for sobel gradients) and the output is on 16 bits, not 8. Lower, you will see the whole algorithm for the 16-bit canny edge detection. It is not as efficient as the one made by opencv to process images, as it does not leverage parallel computing and C++. Similar work is presented in the following github repository [8].

```
def gaussian_blur(image, kernel_size=5, sigma=1.4):
    blurred = cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)
    return blurred
def sobel_gradients(image):
    grad_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    grad_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    magnitude = np.hypot(grad_x, grad_y)
    angle = np.arctan2(grad_y, grad_x)
    return magnitude, angle
def non_maximum_suppression(magnitude, angle):
    M, N = magnitude.shape
    Z = np.zeros((M, N), dtype=np.float64)
    angle_deg = angle * 180.0 / np.pi
    angle_deg[angle_deg < 0] += 180
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            try:
                q = 0.0
                r = 0.0
                if (0 <= angle_deg[i, j] < 22.5) or (157.5 <= angle_deg[i, j] <= 180):
                    q = magnitude[i, j + 1]
                    r = magnitude[i, j - 1]
                elif 22.5 <= angle_deg[i, j] < 67.5:
                    q = magnitude[i + 1, j - 1]
                    r = magnitude[i - 1, j + 1]
                elif 67.5 <= angle_deg[i, j] < 112.5:
                    q = magnitude[i + 1, j]
                    r = magnitude[i - 1, j]
                elif 112.5 <= angle_deg[i, j] < 157.5:
                    q = magnitude[i - 1, j - 1]
                    r = magnitude[i + 1, j + 1]
                if (magnitude[i, j] >= q) and (magnitude[i, j] >= r):
                    Z[i, j] = magnitude[i, j]
                else:
                    Z[i, j] = 0
            except IndexError:
                pass
    return Z
def double_thresholding(img, lowThresholdRatio=0.05, highThresholdRatio=0.15):
    highThreshold = img.max() * highThresholdRatio
    lowThreshold = highThreshold * lowThresholdRatio
    M, N = img.shape
    res = np.zeros((M, N), dtype=np.float64)
    strong_value = img.max()
    weak_value = strong_value / 3.0
    strong_i, strong_j = np.where(img >= highThreshold)
    weak_i, weak_j = np.where((img < highThreshold) & (img >= lowThreshold))
    res[strong_i, strong_j] = strong_value
    res[weak_i, weak_j] = weak_value
    return res, weak_value, strong_value
def hysteresis(img, weak, strong):
    M, N = img.shape
    for i in range(1, M - 1):
        for j in range(1, N - 1):
            if img[i, j] == weak:
                if ((img[i + 1, j - 1] == strong) or (img[i + 1, j] == strong) or (img[i + 1, j + 1] == strong)
                        or (img[i, j - 1] == strong) or (img[i, j + 1] == strong)
                        or (img[i - 1, j - 1] == strong) or (img[i - 1, j] == strong) or (img[i - 1, j + 1] == strong)):
                    img[i, j] = strong
                else:
                    img[i, j] = 0
    return img
```

Now that we have the processed images, we need to generate the point cloud of those objects. We do this by keeping the 3D coordinates of all the strong pixels after the canny edge detection.

```
def edge_to_3D(self, scale_x, scale_y, dim_z, distance):
    step_z = dim_z/self.Data_numbers
    z_curent = -dim_z/2
    for k in range(self.Data_numbers):
        for x in range(self.Data_set_file[k].shape[0]):
            for y in range(self.Data_set_file[k].shape[1]):
                coord = [scale_x*(512-x),scale_y*(512-y),z_ curent]
                if self.Data_set_file[k][x][y] == 1 and self.point_condition(self.Point_Array, coord,distance):
                    self.Point_Array.append(coord)
        z_ curent = z_ curent + step_z
```

After this process, we now have some of the strong pixels (selected at a certain distance from each other) stored into the Point array that, thus we have created a point cloud that looks similar to the one below.

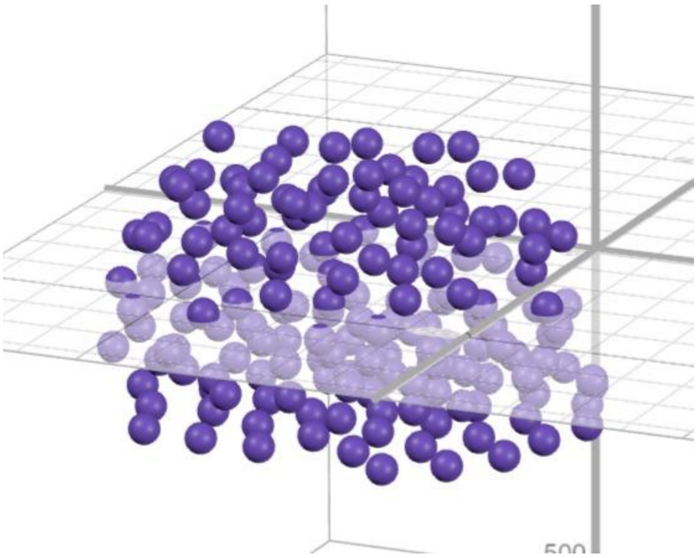


Fig. 4. Point Cloud

At this point the inside points are filtered out and the convex hull is computed. After using Delaunay triangulation, the final result will look like the picture below. Unfortunately, it is not detailed enough as the current code is not efficient, but it did not produce any holes in the mesh while reconstructing the object which is an advantage compared to other methods.

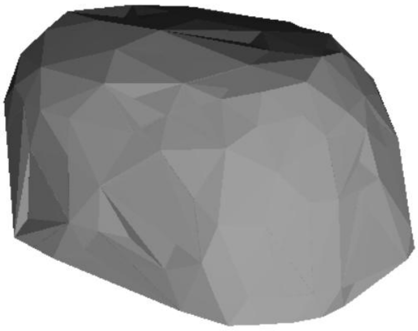


Fig. 5. Final result

4 Future advancements

Since the current method lacks any segmentation algorithm, a segmentation algorithm would really have a great impact, allowing the method to reconstruct each organ specifically. More specifically, a segmentation algorithm like AlexNET would be ideal.

The second improvement would be to write the code in such a way that it uses all CPU cores and ideally, even see whether some code can run efficiently on the GPU, but more work is needed here.

Additionaly, implementing an algorithm like MESHTRON for object generation would improve the output reasult greatly as the algorithm also improves geometry, while preserving important details.

Since there are no algorithms that can quite reach the barrie sub-micrometer accuracy without suffering from hefty computing time increases, an algorithm based on Brownian motion or fractals would be great for generating the vasculatory system in case of potential resolution increases for 3D bioprinters.

5 Conclusions

While all current methods have the ability to produce impressive results, when it comes to 3D bioprinting, ideally the process would have a tool for organ segmentation (like AlexNET), then use point clouds for organ reconstruction and MESHTRON for object generation and have methods that generate the vasculatory system from scratch.

Of couse that there might be better alternatives for organ segmentation (depending on what we segment), but the rest seem to be the best tools for this specific domain. Since these methods are still not perfect, more work is still needed.

Acknowledgement: This work was supervised by Assoc. Prof. Dr. Eng. Angel Cataron from University of Transylvania, Brasov.

References

[1] Processing methods, https://vciba.springeropen.com/articles/10.1186/s42492-023-00142-7
[2] MESHTRON, https://arxiv.org/abs/2412.09548
[3] Canny edge detection, https://medium.com/data-science/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
[4] Canny edge detection, https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
[5] GIS, https://www.tandfonline.com/doi/abs/10.1080/02693799308901979
[6] Points2Surf, https://arxiv.org/abs/2007.10453
[7] The used dataset, https://www.kaggle.com/datasets/trainingdatapro/dicom-brain-dataset
[8] Similar canny edge detection implementation, https://github.com/sbme-tutorials/sbe404-cvtoolbox-ver2-group_25/blob/master/mainwindow.py

Matei-Cristian Steavu
Transilvania University of Brasov
B-dul Eroilor nr. 29
ROMANIA
E-mail: mateisteavu@yahoo.com

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Road Condition Classification and Predictive Maintenance Using (OBD-II) Data

Tanya Teresse, Jarin Justin Victoria

Abstract

The On-Board Diagnostics II (OBD-II) system changed the course of car maintenance by offering low-level access to critical vehicle data in real time. This paper discusses the modern-day applications of OBD-II in predictive maintenance, along with examples of incorporating machine learning algorithms for improved diagnosis and anomaly detection. We analyze road conditions using a labeled dataset containing OBD-II sensor readings from three categories: Normal, Free, and Busy roads. Feature selection was carried out using mutual information scores to identify the most informative attributes out of over 120 available. We then predict maintenance needs as well as identify anomalies using models such as Random Forests, Support Vector Machines, and Neural Networks. Anomalies are defined as unusual sensor patterns potentially indicating mechanical or electrical faults, and are detected using One-Class SVM. Random Forests provide high accuracy and robustness for both classification and regression tasks and have desirable traits for practical applications. Our study highlights the potential of OBD-II data and machine learning in transforming how we optimise the reliability and safety of vehicles.

1 Introduction to OBD and Predictive Maintenance

The revolution in automotive diagnostics has been the core of the change in automotive management that will steer better, smarter maintenance strategies. In its early introduction in the 1980s, On-Board Diagnostics (OBD) was developed for monitoring engine health and failure detection. The OBD systems of early times were relatively limited, often to company-proprietary technologies unique to each manufacturer. However, everything changed in 1996 with the introduction of OBD-II when the whole standard used in cars in most parts of the world was implemented. Along with the broadening of the diagnostic possibilities for OBD-II, there was also an added provision of a universal interface that permits the real-time monitoring of any set of many car subsystems and easy fault finding on mainstream vehicle makes and models [1,2].

Predictive maintenance is a proactive and data-driven concept that utilizes the rich stream of sensor data provided by OBD-II systems. Rather than relying on reactive or scheduled maintenance cycles, predictive maintenance uses models trained on historical and real-time OBD-II data to identify early signs of issues. These patterns and anomalies—such as irregular sensor readings or unexpected combinations of parame- ters—can suggest mechanical or electrical problems before they become critical. This methodology cuts down on safety risks and downtime, improves performance, and extends the lifespan of key components. It represents a forward step in achieving greater reliability and operational efficiency in the automotive industry [7].

1.1 On-Board Diagnostics II (OBD-II)

The OBD-II systems act as a link between the Engine Control Unit (ECU) and the automotive diagnostic equipment. OBD-II offers real-time as well as stored information and is globally recognized under legal guidelines. A 16-pin Data Link Connector (DLC) enables real-time or stored diagnostic data retrieval [7][2]. This global interface allows accurate monitoring of a vehicle's subsystems including its engine, transmission, exhaust systems, and ensuring compliance with eco-friendly measures [6][4]. The standardized structure also enables consistent data access across over 120 sensor attributes, which can be selectively used for various diagnostic and predictive tasks.

1.1.1 Key Components of OBD-II

The OBD-II system serves as the vehicle's self-diagnostic and reporting system. It collects information from a network of sensors and electronic control units (ECUs) to continuously monitor numerous subsystems in the vehicle. The main components of the OBD-II system are as follows:

- Electronic Control Unit (ECU): The ECU acts as the brain of the OBD-II system, processing inputs from numerous sensors to control engine and other vehicle system functions. It also captures Diagnostic Trouble Codes (DTCs) if it detects a fault or malfunction [5][7].
- Sensors: Sensors are like the vehicle's 'nervous system,' and are dispersed throughout the vehicle to measure parameters like oxygen levels in the exhaust, engine temperature, and fuel economy. The sensors then transmit the information to the ECU to analyze and control specified engine and vehicle systems [5][2]. In this study, we focused on a subset of these sensor readings, selected through statistical feature analysis for their relevance to predictive maintenance and road condition detection.
- Diagnostic Trouble Codes (DTCs): When the ECU detects a fault, it assigns a DTC - a pre-defined code corresponding to the detected fault. These codes can be read with an OBD-II scanner, and can assist mechanics and vehicle owners with diagnosing problems and addressing any issues before they become larger problems [5][6].
- Standardized Interface: OBD-II uses a communication protocol and a universal connector to facilitate the reading of data from any OBD-II scanner or diagnostic instrument. Regardless of the variety of vehicle manufacturers, this standardized interface can assist with accurate diagnostic and repair [4][7].

1.1.2 Types of Data

OBD-II supplies a range of parameters necessary for predictive maintenance [6] such as:

- Real-Time Data: Continuously generating data from live sensors to provide real-time information at the vehicle's operating speed e.g. engine RPM, coolant temperature, fuel system health, and vehicular speed [7].
- Freeze Frame Data: Freeze frame data is a snapshot of important engine characteristics (e.g. throttle position, intake manifold pressure) recorded when a Diagnostic Trouble Code (DTC) is generated to provide context at the time the problem was reported [5]. These snapshots can also aid in identifying anomalies in the context of specific events.
- Diagnostic Trouble Codes (DTCs) are a standard set of instructions used to identify specific defects in car systems (e. g. engine, body, chassis, network defects) [6].
- Emission Readiness Data Status of on-board diagnostic monitors that assess the efficiency of emission control systems [7].
- Permanent DTCs Regulatory (or other long term diagnostic fault codes) kept in ECU memory until resolved [7].
- Event Data Recorder (EDR) Information Commonly used in accident investigation, EDR data is recorded in incidents like sudden deceleration and or airbag deployment [4].

2 Literature Review

2.1 Evolution of OBD-II and Predictive Maintenance

On-Board Diagnostics II (OBD-II), standardised in 1996, was originally designed to monitor emissions from vehicles and ensure adherence to environmental regulations. However, the role of OBD-II has grown and today it is also an essential diagnostic instrument gathering information from all subsystems of the vehicle such as the engine, transmission and exhaust [3].

OBD-II data is fully utilized by predictive maintenance, which adopts a proactive approach to automotive care, to foresee issues before they result in failure. This technique has evolved into a successful and economical means of reducing unplanned malfunctions, enhancing safety, and optimizing maintenance plans by using machine learning to examine data patterns [7]. In the context of this study, predictive maintenance is specifically applied to detect early signs of failure based on sensor readings that vary under different road conditions, such as stop-and-go traffic or continuous highway driving.

2.2 Applications of OBD-II Data

Diagnostic Trouble Codes (DTCs), time-series data, and real-time data streams are only a few of the diagnostic data types that can be produced by OBD-II and play an important role in the diagnosis process. Additional parameters such as engine RPM, throttle position, and air flow rate are particularly valuable when evaluating performance patterns over different road scenarios. The following are examples of the type of data that Long Term Predictive Analysis (LTA) and Anomaly Detection (AD) can benefit from using. Nagy and Lakatos (2023) stressed the need for integrating OBD-II with state-of-the-art analytical toolkits for real time diagnostics. Fransson (2015) analyzed associative classification of DTC to identify patterns [6][8].

2.3 Machine Learning for Predictive Maintenance

There is a great deal of usage in OBD-II data with the help of machine learning. Examples of typical models are:

- Random forests and decision trees are popular mainly because of their interpretability and ability to hold high dimensions of data.
- SVMs (Support vector machines) also work well for binary categories like fault vs no fault.
- Neural networks Time series data analysis is a very good case for using deep learning algorithms such as LSTMs which enable very precise fail prediction.

Generally speaking, in terms of accuracy and robustness comparative studies (e. g. Maheshwari et al., 2024) indicate that ensemble models like Random Forest perform better than simpler models [9]. In this study, a comparison of multiple models—including Random Forest, SVM, MLP, Gradient Boosting, and Gaussian Process Regressors—was performed to evaluate their effectiveness in classifying road conditions and predicting maintenance needs.

2.4 Anomaly Detection in Automotive Systems

OBD-II data anomalies frequently indicate mechanical or electrical problems. Outliers have been found using clustering and One-Class SVM methods, which allow insights into unusual operating conditions. Moreover, studies have shown that anomalies are more frequent in stressful or congestion-mode applications, such as chaotic (busy) road traffic [8]. In our implementation, anomalies are defined as sensor behaviors that deviate significantly from the learned normal operating patterns. These anomalies are detected without explicit failure labels, using One-Class SVM, making this approach suitable for real-world scenarios where annotated fault data is often unavailable.

3 Experiment

This section examines the predictive maintenance and sensor functionality tests for vehicle telemetry systems through OBD-II data. The goal of this analysis is to leverage predictive OBD-II analytics in vehicle maintenance optimization. The project seeks to automate the analysis of road conditions and irregularities and the prediction of maintenance work through the application of machine learning technologies. The approach focuses on reducing the unavailability of the vehicle for use, improving safety, and increasing efficiency by deriving value from real-time diagnostic data. We also examine how road conditions—categorized as Normal, Free, and Busy—may influence the frequency of anomalies and prediction accuracy, linking driving environments to component stress.

3.1 Dataset

The dataset comprises 2,693,087 rows and 13 columns of road condition, taken from the OBD-II sensors [11] and categorizes the road condition into Normal Road (1,975,080 samples), Free Road (238,032 samples), and Busy Road (479,975 samples). The raw data was extracted from multiple CSV files named according to these road condition labels and processed to create a unified structure. Each file was assigned standardized column names, and missing values were removed to ensure consistency and reliability. The date of data collection was extracted from the filenames and appended to the records. All datasets were merged into a single dataframe, sorted by date, and labeled with a 'Road Condition' column.

It is important to note that these labels were not instrument-measured but were assigned based on the filenames provided in the original dataset. As such, they reflect traffic context as observed by the data collector.

Feature selection was carried out using the mutual_info_classif method from scikit-learn, which quantifies the mutual information between each feature and the target variable, resulting in the selection of: Absolute Throttle Position, Engine Coolant Temperature, Intake Air Temperature, Engine RPM, Air Flow Rate, Vehicle Speed Sensor, Intake Manifold Pressure, and Accelerator Pedal Position E. This technique is suitable for selecting the most informative features in a high-dimensional sensor dataset.

3.2 Methodology

The experimental setup was implemented in the Kaggle notebook environment, configured with no accelerator, no persistence, and using the latest Python runtime. The programming language used was Python, and the primary libraries employed in this study were Pandas and NumPy for data manipulation, and scikit-learn for machine learning model development, evaluation, and feature selection. Additional libraries such as Matplotlib and Seaborn may have been used for data visualization where necessary. These results support potential use in near real-time scenarios, though further timing evaluations are needed for deployment.

The data preparation process involved reading the OBD-II CSV files from a predefined input directory. The files were categorized based on keywords in their filenames: 'Normal', 'Frei', and 'Stau', corresponding to Normal, Free, and Busy road conditions, respectively. Each dataset was assigned uniform column names and a date field, and a categorical label for road condition was appended. After removing rows with missing values, the datasets were combined into a single dataframe, and sorted by the date column to prepare for time-series aware modeling where necessary.

The models used in the study included Random Forest Classifier and Regressor, Support Vector Classifier and Regressor, Gradient Boosting Regressor, Multi-layer Perceptron Regressor, and Gaussian Process Regressor. Default hyperparameters were used unless otherwise noted. For MLP, the architecture included two hidden layers with 100 and 50 neurons respectively. SVM models used RBF kernels, while Gaussian Process models were applied with an RBF kernel for consistency. The Random Forest models were selected due to their strong performance in non-linear environments and their robustness to overfitting, achieved through ensemble learning. The Support Vector Machines were included to compare performance with margin-based classifiers, though they showed limitations in scalability and sensitivity to parameter scaling. The Gradient Boosting and MLP models were employed to explore the effectiveness of boosting and neural networks in regression-based anomaly detection. The Gaussian Process Regressor was used to investigate uncertainty estimation, but it exhibited difficulties in modeling due to data scale and non-Gaussian noise.

Evaluation metrics were chosen based on the specific objectives of the tasks. For classification models, precision, recall, F1 score, and ROC AUC were used. Precision and recall were critical to balance false positives and false negatives, especially in detecting anomalies or abnormal road conditions. The F1 score provided a harmonic mean of precision and recall, making it suitable for imbalanced datasets. The ROC AUC score measured the model's discriminative capability across all classification thresholds. For regression models, root mean squared error (RMSE) and mean absolute error (MAE) were used. RMSE penalized larger errors, making it appropriate for sensitive predictive maintenance forecasts, while MAE gave an average error magnitude that is easy to interpret.

3.3 Classification Models

3.3.1 Road Condition Classification Metrics

Table 1: Performance Metrics of the Random Forest Classifier

Model	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**ROC AUC (%)**
Random Forest Classifier (RFC)	99.25	99.26	99.25	99.98

The RFC showed strong performance in classifying road conditions, achieving a high F1 Score of 99.25%, which reflects a good balance between precision and recall. It achieved a ROC AUC value of 99.98%, indicating excellent class separation. These results suggest that the model could be considered reliable for practical use. The ensemble method was used to optimize the model, that is makes use of multiple decision trees to make the model more accurate and robust.

3.3.2 Predictive Maintenance Regression Metrics

Table 2: Error Metrics of the Random Forest Regressor

Model	**RMSE**	**MAE**
Random Forest Regressor (RFR)	0.15	0.04

The low root mean square error (RMSE) of 0.15 and mean absolute error (MAE) of 0.04 indicate that the RFR model performs well in predicting maintenance requirements. These low error rates are due to the model's strong ability to capture non linear relationships present in OBD-II data such as changes in engine parameters or sensor readings over time.

3.4 Anomaly Detection Models

3.4.1 Anomaly Detection Metrics

Table 3: Comparison of Classification Model Performance

Model	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**ROC AUC (%)**
Support Vector Classifier (SVC)	99.08	99.20	99.02	92.78
Random Forest Classifier (RFC)	99.41	99.46	99.40	97.26
Gaussian Process Regressor (GPR)	98.85	99.02	98.59	89.21

RFC showed the highest precision (99.41%) and recall (99.46%), indicating strong performance in identifying anomalies while minimizing false negatives. SVC was less effective, likely due to its sensitivity to parameter scaling and difficulty handling large-scale datasets. GPR's lower metrics may be attributed to its scalability limitations and reliance on Gaussian assumptions, which may not align well with the complexity of OBD-II sensor patterns.

3.4.2 Anomaly Regression Metrics

Table 4: Comparison of Regression Model Error Metrics

Model	**RMSE**	**MAE**
Random Forest Regressor (RFR)	0.07	0.01
Gradient Boosting Regressor (GBR)	0.08	0.01
Multi-Layer Perceptron Regressor (MLP)	0.08	0.01
Support Vector Regressor (SVR)	0.10	0.06
Gaussian Process Regressor (GPR)	0.10	0.01

The road condition labels used in classification were derived from the original filenames (e.g., 'Normal', 'Frei', 'Stau') provided in the dataset, and were used as categorical targets for supervised learning. RFR outperformed the other models narrowly, with a root mean square error (RMSE) of 0.07 and a mean absolute error (MAE) of 0.01. GBR and MLP displayed competitive results but underperformed slightly, likely due to their sensitivity to hyperparameter tuning. SVR exhibited the highest error rates (RMSE: 0.10, MAE: 0.06), likely due to its limitations in modeling non-linear data effectively without extensive parameter adjustment. GPR, while consistent in performance, struggled with the complexity of the data, resulting in RMSE values similar to SVR.

4 Result

Among the various experiments, we find that most of the used machine learning models can extract useful features from OBD-II data. The Random Forest approaches appeared to perform the best in gaining different insights from OBD-II data including classification of road conditions, anomaly detection and predictive maintenance of automobiles.

This classification performance is notably strong for a random forest classifier. The RFC Matrices are near perfect for road condition classification. They demonstrate the reliability and accuracy of the proposed model that distinguishes between different road conditions. These results support the potential use of random forest classifiers in near real-time scenarios, where classification accuracy is critical for optimizing vehicle behavior and safety.

A good result in terms of predictive maintenance is that the Random Forest Regressor (RFR) exhibits the highest degree of accuracy in terms of error distribution, and consistently shows a significantly lower margin of error when compared to other model formulations. This is of great benefit in terms of the ability to anticipate failures and to act in a timely manner, which enhances reliability and operation efficiency of vehicles.

In the case of anomaly detection, the RFC model showed a superior performance compared to other approaches with regard to the precision and recall values. It proved to be reliable in terms of finding anomalies in large-scale complex datasets. In contrast, models such as Support Vector Classifier (SVC) and Gaussian Process Regressor (GPR) were less effective in scaling to large datasets or capturing complex sensor interactions.

In the anomaly regression problem again the RFR appeared to be the most reliable model, giving the lowest error rates. Gradient boosting regressor (GBR) and multilayer perceptron (MLP) also carried a competitive performance, but not by much. These results show that Random Forest Regressor is able to take the details of the relationship in the data and make accurate predictions.

The performance of both Random Forest models (RFC and RFR) in terms of robustness, training stability, and low sensitivity to parameter tuning suggests their value as reliable tools for the development of predictive maintenance and anomaly detection from OBD-II data for automotive industries.

Finally, anomaly frequency was found to be higher under Busy Road conditions, further supporting the link between driving environment and maintenance needs.

5 Conclusion

Random Forest models (RFC and RFR) consistently outperformed others on both classification and regression tasks, demonstrating robustness and high accuracy. Anomaly detection showed that a major part of the anomalies were caused by the high traffic conditions (Busy Road), with this result we recommend proactive maintenance. Deploying these models in a real-time or near real-time system could increase vehicle reliability, reduce unexpected breakdowns, and optimize maintenance cycles in both cost and safety aspects. Future research can explore deep learning (such as LSTMs) to model temporal dependencies in OBD-II data, and incorporate OBD-II data with other external data sources such as weather or traffic patterns, to further improve prediction accuracy.

Acknowledgement: This work was supervised by Dr. *Deepthi Das*, Associate Dean, School of Science from *Christ (Deemed to be) University, Bangalore.*

References

[1] I. Aris, M.F. Zakaria, S. Bashi, R. Sidek, DEVELOPMENT OF OBD-II DRIVER INFORMATION SYSTEM, In: Proceedings of the 2007 International Conference on Communication Systems, pp. 1–6, 2007.

[2] J. Cumin, D. Novoselović, D. Maric, T. Šolić, Maintenance of automobiles and motorcycles through the prism of OBD II diagnostic tools, In: 2024 International Conference on Intelligent Systems, pp. 1–16, Springer, Berlin, 2024.

[3] California Air Resources Board, On-board diagnostic II (OBD II) systems fact sheet, 2019. https://ww2.arb.ca.gov/resources/fact-sheets/board-diagnostic-ii-obd-ii-systems-fact-sheet

[4] A. Shafi, Vehicle remote health monitoring and prognostic maintenance system, In: Proceedings of the 2022 International Conference on Computer Applications, pp. 1–5, 2022.

[5] D. Rimpasa, A. Papadakis, M. Samarakou, OBD-II sensor diagnostics for monitoring vehicle operation and consumption, In: Proceedings of the 2023 IEEE International Conference on Smart Transportation and Smart Economy, pp. 1–10, 2023.

[6] M. Fransson, L. Fåhraeus, Finding Patterns in Vehicle Diagnostic Trouble Codes: A Data Mining Study Applying Associative Classification, Master's thesis, Department of Information Technology, Uppsala University, Sweden, June 2015.

[7] A. Theissler, J. Pérez-Velázquez, M. Kettelgerdes, G. Elger, Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry, Reliability Engineering & System Safety, vol. 215, 107864, 2021. https://doi.org/10.1016/j.ress.2021.107864

[8] J. Nagy, I. Lakatos, The Past, the Present and the Future of Online Road Vehicle Diagnosis from Car and Scan Tool Communication Perspective, In: Proceedings of the 2023 Conference on Advances in Transdisciplinary Engineering, pp. 1–8, IOS Press, 2023. https://doi.org/10.3233/ATDE230419

[9] S. Maheshwari, S. Tiwari, S. Rai, S.V. Singh, Comprehensive Study Of Predictive Maintenance In Industries Using Classification Models And LSTM Model, arXiv preprint arXiv:2403.10259, 2024. https://arxiv.org/abs/2403.10259

[10] D.P. Viana, D.H.C. de Sá Só Martins, A.A. de Lima, F. Silva, M.F. Pinto, R.H.R. Gutiérrez, U.A. Monteiro, L.A. Vaz, T. Prego, F.A.A. Andrade, L. Tarrataca, D.B. Haddad, Diesel Engine Fault Prediction Using Artificial Intelligence Regression Methods, Machines, vol. 11, no. 5, 530, 2023. https://doi.org/10.3390/machines11050530

[11] M. Weber, Automotive OBD-II Dataset, Karlsruhe Institute of Technology, doi: 10.35097/1130, 2023.

Tanya Teresse
Christ (Deemed to be) University
Department of Statistics and Data Science
Bangalore, 560029
INDIA
E-mail: *tanya@msds.christuniversity.in*

Jarin Justin Victoria
Christ (Deemed to be) University
Department of Statistics and Data Science
Bangalore, 560029
INDIA
E-mail: *jarin.jv@msds.christuniversity.in*

Dr. Deepthi Das
Christ (Deemed to be) University
Associate Dean, School of Science

Bangalore, 560029
INDIA
E-mail: *deepthi.das@christuniversity.in*

9th International Conference on Applied Informatics
Imagination, Creativity, Design, Development
ICDD 2025, May 29-31
Sibiu, Romania

Assessing Visual Tracking in Children with Special Needs: A Tool for Ergotherapists

Gokul Perumbayil Vijayakrishnan, Anagha Manikathuparambil Baby, Blesson Manjakunnel

Abstract

Assessing visual tracking ability in children with special needs is challenging due to the limitations of traditional observational methods, which are often time-consuming, costly, and imprecise, while also struggling to maintain the engagement of the child. Additionally, these methods can be difficult for therapists to interpret. This study proposes a cost-effective eye-tracking tool that enables therapists to evaluate and enhance visual tracking abilities without the need for specialised hardware.

The tool integrates an interactive game to sustain engagement while using a standard webcam to capture gaze data. A deep learning model then processes this data, mapping gaze direction to screen coordinates to generate an interpretable representation of visual tracking performance. Our analysis quantifies the correlation between predicted gaze points and the actual trajectory of a moving object along the x and y axes, providing therapists with a visual representation of gaze behaviour.

By enabling remote assessments and minimizing logistical barriers, this tool enhances accessibility, precision, and efficiency in therapeutic evaluations. It provides an affordable, open-source alternative to conventional eye-tracking systems, usable on any standard computer with a webcam, and supports data-driven intervention strategies to enhance therapeutic outcomes.

1 Introduction

Therapists who work with children with special needs generally rely on observational techniques to assess and address developmental challenges related to communication, attention, and motor skills [5]. However, such traditional assessment methods are inherently time-consuming, subjective, and constrained in their ability to capture the complex dynamics of a child's behaviour and participation over time. Furthermore, the requirement for in-person evaluations in specialised facilities poses significant logistical and financial challenges for families, particularly those in geographically remote or resource-limited areas [2]. These limitations highlight the need for innovative tools that improve the precision and efficiency of therapeutic evaluations while enabling remote accessibility.

Eye tracking technology has emerged as a promising non-invasive approach to assessing gaze patterns and visual attention, offering objective insights into cognitive and motor functions [10]. Despite its potential to improve therapeutic practices, adoption of such technology in clinical and therapeutic settings remains limited due to the high cost, technical complexity, and dependence on high-precision specialised equipment [1]. These barriers disproportionately affect underfunded therapy centres and families in rural regions [3], increasing inequalities in access to effective interventions.

This research seeks to address these challenges by developing a cost-effective therapist-focused eye-tracking application designed to provide meaningful, actionable insights into children's eye movements during therapy sessions. Unlike traditional systems, the proposed tool eliminates the need for expensive high-precision hardware and facilitates remote access through a server-based platform. This approach allows therapists to monitor children's eye movement data during interactive, game-like activities conducted in home environments. The platform processes and analyses this data, enabling therapists to design personalised intervention strategies without requiring in-person evaluations.

2 Related Work

Eye tracking has been a subject of research interest for many years, with commercial eye trackers available on the market. As of 2020, the most accurate stationary eye-tracking device was the SMI Red, offering an accuracy of 0.4° at a cost of approximately $40,000 [15]. However, there remains a need for an affordable, budget-friendly eye-tracking solution, which this study aims to address.

The model-based approach [14] involves the use of mathematical or computational models to interpret eye movement data. A model-based eye tracking approach [8] was used to monitor gaze and provide information to the therapists. Despite its innovative intent, this method had several limitations, including the need for users to maintain a fixed distance from the screen, which restricted flexibility, its ineffectiveness for children who wore glasses, and the generation of insights that lacked significant value. Our approach aims to address these challenges, enhancing both the system's utility and inclusivity.

In contrast, appearance-based models focus on extracting visual features from static images or individual frames to analyse, predict, and interpret patterns [13]. Compared to model-based approaches and traditional appearance-based methods, deep learning appearance-based methods demonstrate greater robustness in unconstrained environments, effectively handling extreme head-pose variations, diverse illumination conditions, and occlusions of the eyes and face [19]. This study uses an appearance-based model to estimate the eye gaze.

Eye-tracking technology is a valuable tool for assessing cognitive and social processing in children with special needs, including autism spectrum disorders (ASD), attention deficit hyperactivity disorder (ADHD), and learning disabilities. A recent meta-analysis of 20 eye-tracking studies on face processing in children with ASD found significantly reduced gaze fixation on the eye region, suggesting it as a potential biomarker for ASD [4]. The study highlights the importance of gaze behaviour in ASD diagnosis and reinforces the need for further research to enhance eye-tracking applications in special education and therapy.

3 Methodology

The study adopts a quantitative framework to design, implement, and validate a novel tool to assist ergo therapists in precisely analysing eye movements in children with special needs. The primary goal is to assess participants' visual tracking capabilities through an interactive game, leveraging cutting-edge deep learning techniques and geometric transformations to deliver actionable insights. A camera calibrated using the chequerboard method will be provided to the user for operating this tool; alternatively, users can perform the camera calibration with the chequerboard themselves at their convenience before using the tool. The study is structured into three comprehensive phases: data acquisition through a custom-designed game, gaze estimation and screen coordinate mapping, and performance evaluation and therapeutic insight generation. Figure 1 shows the overall architecture of the software.

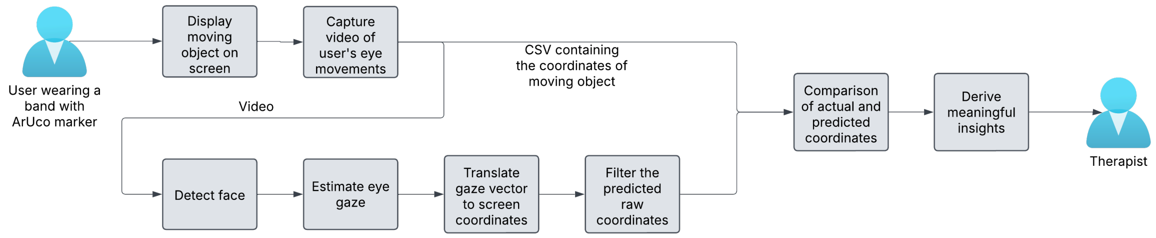

User wearing a band with ArUco marker
Display moving object on screen
Capture video of user's eye movements
CSV containing the coordinates of moving object
Video
Detect face
Estimate eye gaze
Translate gaze vector to screen coordinates
Filter the predicted raw coordinates
Comparison of actual and predicted coordinates
Derive meaningful insights
Therapist

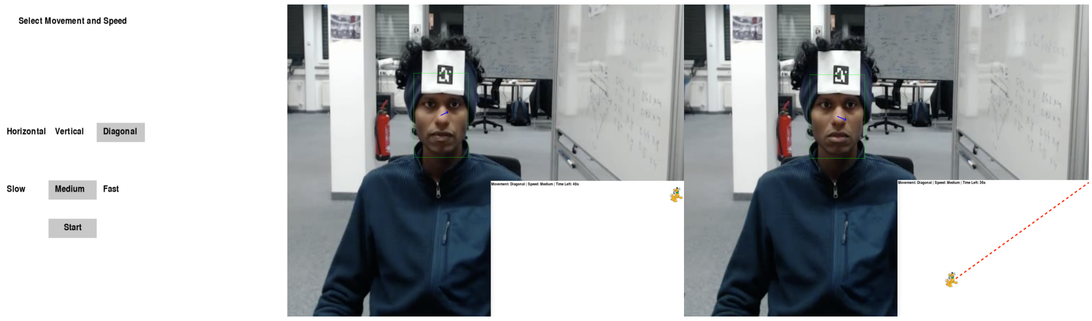
Figure 1: The diagram illustrates the system architecture for gaze tracking and analysis. The process begins with a user wearing a band with an ArUco marker. The recorded video is processed to detect the face, estimate eye gaze, and translate gaze vectors into screen coordinates. A filtering step refines the predicted raw coordinates before they are compared with actual object coordinates. The final insights, derived from this comparison, assist a therapist in understanding gaze behaviour.

3.1 Data Acquisition via Interactive Game

The data acquisition phase is central to ensuring the accuracy and relevance of subsequent analysis. A custom-designed game engages participants while systematically capturing their gaze behaviour. The game features a visually appealing cartoon character that moves across the screen along predefined trajectories: horizontal, vertical, and diagonal. These simple trajectories focus on baseline gaze-tracking metrics and coordination, without imposing undue cognitive load on the participants. During the game, the speed of the character is adjustable, and the interface is designed to maintain the attention and interest of children, thereby ensuring high-quality data capture.

The trajectories are logged in real time, and the precise screen coordinates of the moving character are recorded at each frame in a CSV file for later analysis. This structured data serves as a reference for evaluating gaze alignment. Although more intricate motion patterns, such as circular or zigzag trajectories, could provide deeper insights into complex gaze-tracking behaviours, they are deliberately excluded in this initial phase to minimize complexity and enhance the robustness of the foundational metrics.

Participants are seated approximately 60 cm from the screen, ensuring a consistent spatial geometry between the camera, monitor, and gaze direction. A standard webcam records video footage of the participant's face during the game, capturing fine-grained frame-by-frame details of their eye movements. To account for participants' positioning relative to the screen, each participant wears a headband equipped with an ArUco marker [7], a computer vision tool that facilitates accurate real-time distance measurement. This ensures the tool remains cost-effective by eliminating the need for expensive, specialised hardware to measure the user's distance from the screen, a critical factor affecting the conversion of gaze coordinates into screen coordinates. Figure 2 illustrates the game interface alongside the participant actively tracking the moving object.



Movement: Diagonal | Speed: Medium | Time Left: 49s

Movement: Diagonal | Speed: Medium | Time Left: 36s

Figure 2: A person wearing a headband with an ArUco marker is trying to track the movement of a cartoon character in a particular direction (in this case, diagonal movement).

3.2 Gaze Estimation and Mapping to Screen Coordinates

In the second phase, the captured video frames undergo processing using L2CS-Net [6], a pre-trained deep learning model specifically designed for gaze estimation and trained on the Gaze360 dataset [9]. L2CS-Net predicts yaw and pitch angles, which correspond to gaze direction's horizontal and vertical components, respectively. This model is selected for its demonstrated superiority over traditional approaches such as regression-based methods or geometric models [6]. While conventional techniques often require controlled environments and manual feature engineering, L2CS-Net's reliance on large-scale datasets allows it to generalize effectively across diverse settings, making it highly robust and adaptive.

One critical challenge addressed during this phase is the variability in participants' distance from the screen. The ArUco marker enables the computation of the participant's distance from the screen for each video frame, which is subsequently utilised in further calculations. The distance d of the participant from the screen can be calculated using the magnification formula for a lens as

$$d = \frac{f \cdot h_o}{h_i} \tag{1}$$

where f is the focal length of the camera obtained using camera calibration, h_o is the actual width of the ArUco marker in cm and h_i is the width of the ArUco marker in pixels. Once the distance d is calculated, a geometric transformation is applied to map the gaze angles (yaw and pitch) to precise screen coordinates, effectively accounting for the position of the participant relative to screen, as shown in figure 3. The equations for converting gaze angles (yaw and pitch) to pixel coordinates on the screen are given by:

$$x = -d \cdot \tan(\text{yaw}), \qquad y = -d \cdot \arccos(\text{yaw}) \cdot \tan(\text{pitch}) \tag{2}$$

where d is the screen distance in cm, yaw and pitch are angular values in radians. The additional term arccos(yaw) in equation (2) is an empirical scaling factor to adjust vertical displacement based on horizontal deviation. The conversion from cm to pixel coordinates with top left as origin is as follows:

$$x_{\text{pixel}} = \left(\frac{x + \frac{W_{\text{cm}}}{2}}{W_{\text{cm}}} \right) \times W_{\text{pixels}}, \qquad y_{\text{pixel}} = \left(\frac{y + \frac{H_{\text{cm}}}{2}}{H_{\text{cm}}} \right) \times H_{\text{pixels}} \tag{3}$$

where W_{cm} and H_{cm} are the width and height of the screen in cm, and W_{pixels} and H_{pixels} are the screen width and height in pixels.

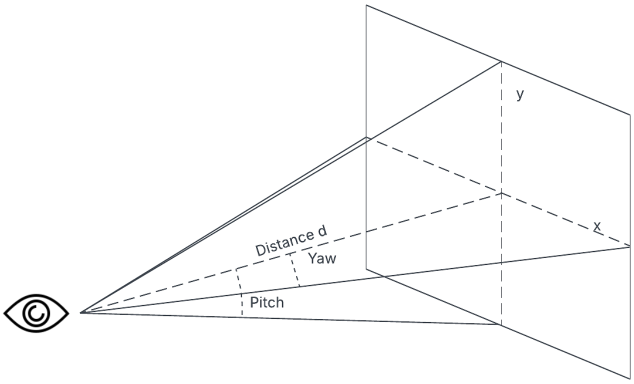


Figure 3: Translation of yaw and pitch angles into screen coordinates (x and y).

Since the raw screen coordinates obtained from gaze tracking are often noisy due to head movements, eye blinks, and lighting variations, a Kalman filter [18] is applied to smooth the data and handle missing values. By recursively estimating the true gaze position based on both past predictions and the current noisy measurements, the filter improves tracking accuracy. A constant velocity motion model is assumed, as the video is recorded at 30 FPS. The state vector of the Kalman filter consists of four components:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{4}$$

where x and y represent the estimated gaze position on the screen (after filtering), while v_x and v_y represent the estimated gaze velocity in the horizontal and vertical directions, respectively. The system follows a linear motion model:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{w}_k \tag{5}$$

where $\mathbf{F}$ is the state transition matrix defined as:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

with $\Delta t = \frac{1}{\text{FPS}}$ s, with FPS denoting the number of frames per second. The process noise $\mathbf{w}_k$ accounts for small random variations in motion. The Kalman filter updates its estimates using the measurement model, which includes only two components, x and y, corresponding to the observed gaze positions from equation (3).

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \tag{7}$$

where $\mathbf{v}_k$ represents measurement noise, and $\mathbf{H}$ is the observation matrix given by equation (8).

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{8}$$

At each time step, the filter predicts the next gaze position based on the current state estimate and corrects it using the incoming measurements, effectively reducing noise and ensuring smoother tracking.

3.3 Performance Evaluation and Metrics Extraction

The final phase involves a rigorous comparison of the mapped gaze coordinates with the recorded coordinates of the moving cartoon character. This comparison yields quantitative metrics that offer insights into the participant's visual tracking capabilities. The key metrics include:

- Cross correlation: Measures how well the gaze trajectory follows the object movement over time. A high correlation (closer to 1) means the gaze movement aligns well with the object movement. Cross-correlation has been widely used in gaze tracking research to quantify alignment [11].

  $$\rho_x = \frac{\sum_{i=1}^{N}(x_{\text{g},i} - \bar{x}_{\text{g}})(x_{\text{obj},i} - \bar{x}_{\text{obj}})}{\sqrt{\sum_{i=1}^{N}(x_{\text{g},i} - \bar{x}_{\text{g}})^2 \sum_{i=1}^{N}(x_{\text{obj},i} - \bar{x}_{\text{obj}})^2}} \tag{9}$$

  where $x_{\text{g},i}$ represents the gaze position at time i, and $\bar{x}_{\text{g}}$ denotes the mean gaze position. Similarly, $x_{\text{obj},i}$ represents the object position at time i, while $\bar{x}_{\text{obj}}$ refers to the mean object position.

- Gaze Jitter: Measures how much the gaze position fluctuates within short time intervals. High jitter (closer to 1) indicates noisy tracking, while low jitter suggests stable gaze tracking. This metric has been used to assess gaze stability and visual attention in experimental psychology [12].

  $$\text{Jitter} = \frac{1}{N}\sum_{i=1}^{N}\left((x_{\text{g},i} - x_{\text{g},i-1})^2 + (y_{\text{g},i} - y_{\text{g},i-1})^2\right) \tag{10}$$

  where $x_{\text{g},i}$ and $y_{\text{g},i}$ represent the gaze coordinates at time i, while $x_{\text{g},i-1}$ and $y_{\text{g},i-1}$ denote the gaze coordinates at time $i-1$. Additionally, N represents the total number of gaze samples.

These metrics collectively provide a comprehensive assessment of visual tracking performance, helping to identify specific visual-motor challenges. A low correlation in certain axes (e.g., horizontal vs. vertical) may indicate difficulties in gaze coordination or tracking consistency. Such insights enable ergo-therapists to design personalised interventions. For instance, if a child demonstrates strong horizontal tracking but weaker vertical tracking, targeted exercises can be introduced to improve vertical gaze control, ultimately enhancing overall tracking ability and engagement in daily activities.

The proposed approach prioritizes cost-effectiveness without compromising performance. By utilizing a standard webcam and open-source software tools, the methodology avoids the prohibitive costs associated with specialised eye-tracking hardware, such as Tobii [16] or Pupil Labs systems [17]. The proposed tool enables children to complete the test at home without requiring visits to a testing centre, while therapists can remotely access insights and provide personalised recommendations. The software-driven nature of this tool achieves an optimal balance between precision, affordability, and ease of implementation.

4 Experiments and evaluation

4.1 Evaluation Framework

To ensure the reliability and generalization of the proposed eye-tracking tool, a Python-based evaluation framework has been developed. This framework systematically analyses the gaze data collected during the interactive game by comparing participants' gaze behaviour with the predefined trajectories of the moving character. A robust baseline for gaze behaviour in individuals with normal vision is established using mean and standard deviation. This baseline serves as a critical reference point for future comparative studies using Z-score involving children with special needs, recognizing the challenges of directly acquiring data from this population. By adopting a quantitative and systematic approach, the framework validates the ability of the tool to provide actionable information to therapists while ensuring its adaptability across diverse user groups.

4.2 Experimental Setup

To evaluate the performance of the tool under simulated real-world conditions, validation experiments were conducted with seven participants, all of whom had normal vision. Three participants completed the experiment twice to assess repeatability. The evaluation focused on two key metrics: correlation of gaze position along the x and y axes and gaze jitter. All experiments were conducted in a controlled environment to reduce external influences, such as inconsistent lighting or screen glare. Since participants are expected to complete this assessment in a home environment, a well-lit indoor room was selected for the experimental setup. The participants were placed at a standardised distance (60 cm) from the screen, ensuring uniform spatial alignment between the camera, the monitor, and the participant.

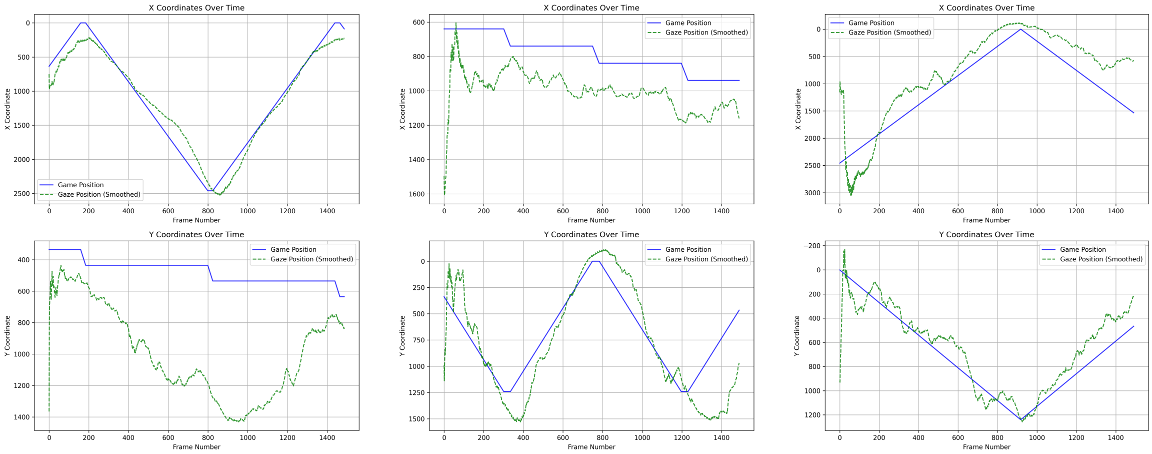
To support these experiments, the system used by the participants consisted of a standard computer equipped with an octa-core processor, 8 GB of RAM, a 1080p webcam capable of recording at 30 frames per second, and a 25-inch Full HD display. The system responsible for video processing and result generation was configured with a more powerful setup, featuring a 32-core CPU, 16 GB or more of RAM, at least 8 GB of GPU memory to handle advanced processing tasks and a minimum of 1 TB of storage. These specifications ensured seamless integration between the two systems, enabling efficient remote monitoring and accurate analysis of gaze data.

4.3 Results

Figure 4 illustrates the trajectory of the object in the game along with the gaze trajectories of the participants after applying filtering to the raw gaze coordinates. The plot reveals that, during the initial frames, gaze positions deviate significantly from the object's path. This is likely because, at the start of the game, participants anticipate the object's initial position and direction of movement rather than actively tracking it. However, after these initial frames, gaze trajectories align more closely with the object's motion, indicating successful tracking.

Additionally, the results show that for horizontal object movement, tracking accuracy is higher along the x-axis, while deviations occur along the y-axis. Conversely, for vertical object movement, tracking is more accurate along the y-axis, with greater deviations in the x-axis. These deviations may be attributed to slight changes in head orientation or involuntary eye blinks, which are difficult to control.

Table 1 provides a comprehensive summary of the experimental data. It details the mean values and the associated standard deviations for the correlations along the x and y axes. In addition, the table



(a) Horizontal movement (b) Vertical movement (c) Diagonal movement

Figure 4: Plots showing the actual movement of an object in the game and the predicted gaze trajectory along the x and y axes. The displayed plots correspond to the best-performing participant for each of the horizontal, vertical, and diagonal movements.

includes the normalised gaze jitter. The data represent the averages computed from seven participants over ten experimental trials. High correlation values in the x-direction for horizontal object movement and in the y-direction for vertical object movement, along with overall low jitter, suggest that the tracking performance aligns with the expected results for the typical population. The data presented in Table 1 can be used as a baseline for comparison. When the tool is applied to children with special needs, individual performance can be evaluated using Z scores, which indicate how many standard deviations a specific data point is from the population mean. For example, if the mean y-axis correlation (μ) for the diagonal movement is 0.83 with a standard deviation (σ) of 0.13, and a child's score (X) is -0.37, the Z-score is computed as,

$$Z = \frac{X - \mu}{\sigma} = \frac{-0.37 - 0.83}{0.13} = -9.23 \tag{11}$$

This indicates that the child's performance is nine standard deviations below the typical mean, suggesting a potential difficulty in horizontal tracking. Such quantifiable comparisons enable tracking of a child's progress over time, allowing for the refinement of interventions and personalised therapeutic strategies.

Metric	Horizontal	Vertical	Diagonal
Correlation X	**0.97 ± 0.01**	0.58 ± 0.26	**0.94 ± 0.03**
Correlation Y	0.33 ± 0.46	**0.84 ± 0.05**	**0.83 ± 0.13**
Gaze Jitter	0.13 ± 0.06	0.11 ± 0.08	0.21 ± 0.16

Table 1: Result statistics — mean and standard deviation of metrics. The table indicates that the x-coordinate correlation is high during horizontal movements. The y-coordinate correlation is notably elevated during vertical movements. Additionally, for diagonal movements, both the x- and y-coordinate correlations are high.

Table 2 presents the mean and standard deviation of metrics without the application of the Kalman filter. The comparison between Table 1 and Table 2 indicates that the application of the Kalman filter enhances the correlation values. In Table 1, the x-coordinate correlation is higher for horizontal and diagonal movements compared to Table 2. Similarly, the y-coordinate correlation is notably stronger in Table 1, especially in vertical and diagonal movements. Table 2 shows a higher gaze jitter value and the application of the Kalman filter gives a significant reduction in noise in gaze tracking.

Metric	Horizontal	Vertical	Diagonal
Correlation X	**0.92 ± 0.06**	0.34 ± 0.20	**0.89 ± 0.05**
Correlation Y	0.27 ± 0.36	**0.78 ± 0.10**	**0.71 ± 0.14**
Gaze Jitter	16.22 ± 9.96	15.28 ± 8.13	16.37 ± 8.99

Table 2: Result statistics — mean and standard deviation of metrics without using Kalman filter. The comparison with Table 1 highlights the impact of filtering.

5 Conclusion and future work

This study introduced a cost-effective, deep learning-based eye-tracking tool to assess visual tracking abilities in children with special needs. Given the challenges of directly testing this population, we established a baseline using data from a neurotypical population, demonstrating the accuracy of the tool with minimal jitter. Using a standard webcam and an interactive game, our approach provides an accessible and engaging alternative to traditional observational methods, reducing logistical barriers while enhancing the precision of the evaluation. The structured methodology of the tool, which combines gaze estimation, motion trajectory analysis, and quantitative performance metrics, creates a solid framework for future improvements.

In future work, we intend to integrate head pose estimation to address involuntary head movements, thus further enhancing tracking accuracy. We also plan to incorporate additional gaze metrics—such as latency, fixation sequences, and gaze heatmaps—to provide more comprehensive insights into visual tracking behaviours and improve the overall interpretability of results.

We intend to increase the pool of participants beyond the 7 individuals involved in this study, with the aim of approximately 30 to 40 participants to establish a more robust baseline and increase the validity of our findings. Since children with normal vision and children with special needs are our primary focus, we intend to include them in the participant pool. In addition, further refinements to the interactive game design will enhance participant engagement, yielding higher-quality data and more accurate tracking. We also plan to develop dedicated user profiles for children, therapists, and parents, allowing for more tailored assessments and supporting long-term progress monitoring.

Finally, we plan to extend the tool's applicability to other therapeutic contexts such as cognitive rehabilitation and neurological assessments broadening its impact and affirming its role as a practical, scalable solution for visual tracking evaluation across diverse clinical environments.

6 Program Code

The implementation of this paper can be found in the GitHub link: https://github.com/Project-EyeTracking/EyeGaze

Acknowledgment: We would like to express our deepest gratitude to Professor *Dr. Magda Gregorová* from the *Technical University of Applied Sciences Würzburg-Schweinfurt* for her invaluable guidance, insightful feedback, and continuous support throughout this work. In addition to the guidance received, we also utilised digital tools to support the writing process. ChatGPT was used to generate ideas for structuring the paper and to assist in drafting clear, cohesive sections. It helped in organising the content and suggesting ways to present the information effectively. Furthermore, Grammarly was used to review grammar, punctuation, and style, helping to ensure the paper adhered to academic language standards and maintained clarity throughout.

References

[1] Amanda Cyntia Lima Fonseca Rodrigues, Keun-Hwa Jung, Advancing Post-Stroke Cognitive Assessments: The Potential and Challenges of Integrating Eye Tracking Technology in Clinical Practice.

Cerebrovascular Diseases, 1–2, 2024.

[2] R. Ramezani, S. Iranmanesh, A. Naeim, and P. Benharash, Bench to Bedside: AI and Remote Patient Monitoring, *Frontiers in Digital Health*, Volume 7, 2025.

[3] X. Hunt, A. Saran, H. White, and H. Kuper, Effectiveness of interventions for improving educational outcomes for people with disabilities in low- and middle-income countries: A systematic review, *Campbell Systematic Reviews*, 2025.

[4] Eleni A. Papagiannopoulou, Kate M. Chitty, Daniel F. Hermens, Ian B. Hickie, and Jim Lagopoulos, A systematic review and meta-analysis of eye-tracking studies in children with autism spectrum disorders, *Social Neuroscience*, Pages 610-632, 2014.

[5] Michael D. Powers, Mark J. Palmieri, Kristen S. D'Eramo, and Kristen M. Powers, Behavioral Intervention Techniques for Reducing Problem Behavior, *Springer Nature Switzerland*, Pages 251–287, 2025.

[6] Ahmed A.Abdelrahman, Thorsten Hempel, Aly Khalifa, Ayoub Al-Hamadi, L2CS-Net: Gaze Estimation in Unconstrained Environments, *arXiv preprint arXiv:2203.03339*, 2022.

[7] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez, Automatic generation and detection of highly reliable fiducial markers under occlusion, *Pattern Recognition*, Pages 2280-2292, 2014.

[8] Christeena Varghese, Vincent Wahyudi, and Viony Tengguna, App to Help Children with Special Needs Improve Their Eye Movements and Focus, *SCORES'23, Proceedings of the 9th Student Computing Research Symposium*, October 2023.

[9] P. Kellnhofer, Gaze360: Physically Unconstrained Gaze Estimation, *arXiv preprint arXiv:1910.10088*, 2019.

[10] Alexandra Wolf and Kazuo Ueda, Contribution of eye-tracking to study cognitive impairments among clinical populations, *Frontiers in Psychology*, Volume 12, 2021.

[11] Andrew T. Duchowski, Eye Tracking Methodology: Theory and Practice, *Springer*, 2007.

[12] R. Engbert and R. Kliegl, Micro-saccades uncover the orientation of covert attention, *Vision Research*, 2003.

[13] Kar-Han Tan, D.J. Kriegman, and N. Ahuja, Appearance-based eye gaze estimation, *Proceedings of the Sixth IEEE Workshop on Applications of Computer Vision (WACV 2002)*, 2002.

[14] Shuo Wang, Ming Jiang, Xavier Morin Duchesne, Elizabeth A. Laugeson, Daniel P. Kennedy, Ralph Adolphs, and Qi Zhao, A typical Visual Saliency in Autism Spectrum Disorder Quantified through Model-Based Eye Tracking, *Neuron*, Pages 604-616, 2015.

[15] I. Rakhmatulin, A review of the low-cost eye-tracking systems for 2010-2020, *arXiv preprint arXiv:2010.13454*, 2020.

[16] Tobii Technology, Tobii Eye Tracking Systems, *Tobii Technology*, 2024.

[17] Pupil Labs, Pupil Labs Eye Tracking, *Pupil Labs*, 2024.

[18] Rudolph Emil Kalman, A new approach to linear filtering and prediction problems, *Journal of Basic Engineering*, Pages 8167-179, 1960.

[19] Primesh Pathirana, Shashimal Senarath, Dulani Meedeniya, Sampath Jayarathn, Eye gaze estimation: A survey on deep learning-based approaches, *Expert Systems with Applications*, 2022.

Gokul Perumbayil Vijayakrishnan
Technical University of Applied Sciences Würzburg-Schweinfurt
MAI, Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail:
gokul.perumbayilvijayakrishnan@study.thws.de

Anagha Manikathuparambil Baby
Technical University of Applied Sciences Würzburg-Schweinfurt
MAI, Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail:
anagha.manikathuparambilbaby@study.thws.de

Blesson Manjakunnel
Technical University of Applied Sciences Würzburg-Schweinfurt
MAI, Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
Germany
E-mail:
*blesson.manjakunnel@study.*thws.de

LIST OF AUTHORS

Vlad-Stefan ALEXANDRESCU	SPIRU HARET University Faculty of Eng. and Computer Science 46G Fabricii Street, District 6, Bucharest ROMANIA E-mail: g.vlad.alexandrescu@spiruharet.ro
Anagha Manikathuparambil BABY	Technical University of Applied Sciences Würzburg-Schweinfurt MAI, Computer Science and Business Information Systems Sanderheinrichsleitenweg 20, 97074 Würzburg GERMANY E-mail: anagha.manikathuparambilbaby@study.thws.de
Elena-Luiza BUZATU	National University of Science and Technology Politehnica Bucharest, Pitești University Center Faculty of Sciences, Physical Education and Informatics Department of Mathematics and Informatics Str. Targul din Vale, nr. 1, Pitești ROMANIA E-mail: elenaluiza10@gmail.com
Andrei DĂIAN	Lucian Blaga University of Sibiu Faculty of Science Doctor Ion Ratiu Street 5-7, Sibiu ROMANIA E-mail: andrei.daian@ulbsibiu.ro
Deepthi DAS	Christ (Deemed to be) University Department of Statistics and Data Science Bangalore, 560029 INDIA E-mail: deepthi.das@christuniversity.in
Marian-Daniel DRĂGHICI	Lucian Blaga University of Sibiu Faculty of Science Doctor Ion Ratiu Street 5-7, Sibiu ROMANIA E-mail: mariandaniel.draghici@ulbsibiu.ro
Martin S. DZHUROV	University of Rousse “Angel Kanchev” Faculty of Natural Sciences 8 Studentska str., POB 7017, Ruse BULGARIA E-mail: mdzhurov@uni-ruse.com

Ştefan EMINOVICI

Samuel von Brukenthal National College Sibiu
Mathemathics-Informatics, intensive Informatics
ROMANIA
E-mail: stefaneminovici@yahoo.com

Sophie GEISLER

Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business
Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: sophie.geisler@study.thws.de

Felix HUSAC

Lucian Blaga University of Sibiu
Department of Mathematics and
Informatics
5-7 Dr. Ratiu Str, Sibiu 550012
ROMANIA
E-mail: felix.husac@ulbsibiu.ro

Ştefan-Ioan ISTINA

Vasile Alecsandri University of Bacău
Department of Mathematics &
Informatics Calea Mărăşeşti, nr. 157,
Bacău, 600115
ROMANIA
E-mail: istinastefanioan@gmail.com

Blesson MANJAKUNNEL

Technical University of Applied Sciences Würzburg-Schweinfurt
MAI, Computer Science and Business Information
Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: blesson.manjakunnel@study.thws.de

Giorgiana-Maria MARANGOCI

Vasile Alecsandri University of Bacău
Department of Mathematics & Informatics
Calea Mărăşeşti, nr. 157, Bacău, 600115
ROMANIA
E-mail: giorgianamarangoci@gmail.com

Peter MÖHLE

Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business
Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: peter.moehle@study.thws.de

Rares MUNTEAN

SPIRU HARET University
Faculty of Eng. and Computer Science
46G Fabricii Street, District 6, Bucharest
ROMANIA
E-mail: g.rares.muntean@spiruharet.ro

Mustafa Mustafov

University of Ruse
Natural science and education
BULGARIA
E-mail: s216259@stud.uni-ruse.bg

Alexandra ONOSE

Transilvania University of Brasov
Faculty of Mathematics and Informatics
Bd. Iuliu Maniu nr. 50
500091 Brașov
ROMANIA
E-mail: alexandra.onose@student.unitbv.ro

Eduard-Alexandru OPREA

National University of Science and
Technology
Politehnica Bucharest, Pitești
University Center
Faculty of Sciences, Physical Education
and Informatics
Department of Mathematics and
Informatics
Str. Targul din Vale, nr. 1, Pitești
ROMANIA
E-mail: edyoprea69@gmail.com

Vlad-Matei POIENARIU

SPIRU HARET University
Faculty of Eng. and Computer Science
46G Fabricii Street, District 6,
Bucharest
ROMANIA
E-mail: g.vlad.poienariu@spiruharet.ro

Andrei PRIBOI

Transilvania University of Bra¸sov
Faculty of Mathematics and Computer Science
No. 50, Iuliu Maniu st., Bras,ov, Romania
ROMANIA
E-mail: andrei.priboi@student.unitbv.ro

Alex-Andrei RÎPAN

Vasile Alecsandri University of Bacău
Department of Mathematics &
Informatics Calea Mărășești, nr. 157,
Bacău, 600115
ROMANIA
E-mail: darkdragondd100@gmail.com

Serkan Sadulov

University of Ruse
Natural science and education
BULGARIA
E-mail: s216261@stud.uni-ruse.bg

Serkan H. SADULOV

University of Rousse “Angel Kanchev”
Faculty of Natural Sciences
8 Studentska str., POB 7017, Ruse
BULGARIA
E-mail: ssadulov@uni-ruse.com

Tobias SCHNEIDER

Hochschule Landshut
Faculty of Computer Science
Am Lurzenhof 1, 84036 Landshut
GERMANY
E-mail: tobiasschneider1@acm.org

Alexandru Emil SOFONEA

Samuel von Brukenthal National College Sibiu
ROMANIA
E-mail: alex@alexsofonea.com

Kristian P. SPASOV

University of Rousse “Angel Kanchev”
Faculty of Natural Sciences
8 Studentska str., POB 7017, Ruse
BULGARIA
E-mail: kspasov@uni-ruse.com

Matei-Cristian STEAVU

Transilvania University of Brasov
B-dul Eroilor nr. 29
ROMANIA
E-mail: mateisteavu@yahoo.com

Tanya TERESSE

Christ (Deemed to be) University
Department of Statistics and Data Science
Bangalore, 560029
INDIA
E-mail: tanya@msds.christuniversity.in

Ioana-Valeria TURCIN

National University of Science and Technology
Politehnica Bucharest, Pitești University Center
Faculty of Sciences, Physical Education and Informatics
Department of Mathematics and Informatics
Str. Targul din Vale, nr. 1, Pitești
ROMANIA
E-mail: turcin.ioana@yahoo.com

Jarin Justin VICTORIA

Christ (Deemed to be) University
Department of Statistics and Data Science
Bangalore, 560029
INDIA
E-mail: jarin.jv@msds.christuniversity.in

Gokul Perumbayil VIJAYAKRISHNAN

Technical University of Applied Sciences Würzburg-Schweinfurt
MAI, Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: gokul.perumbayilvijayakrishnan@study.thws.de

Marcel WERNISCH

Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: marcel.wernisch@study.thws.de

Felix ZORN

Technical University of Applied Sciences Würzburg
Faculty of Computer Science and Business Information Systems
Sanderheinrichsleitenweg 20, 97074 Würzburg
GERMANY
E-mail: felix.zorn@study.thws.de

Organized with support of Romanian Ministry of National Education

Romanian Ministry of National Education

SPONSORS (in alphabetical order)

Asociatia Sibiu IT

Asociația BIT

CodexWorks technologies

***Ardeleana Academy* Foundation**

Global Solutions for Development

NXP

NTT Data

PAN FOOD

ROPARDO

Wenglor