FlyAI: A Unified NLP and Computer Vision Framework for Autonomous Drone Mission Programming

Gilda Manfredi[1,*], Ugo Erra[1], Nicola Capece[1] and Michele Rinaldi[1]

[1]University of Basilicata, Via dell'Ateneo Lucano 10, Potenza, 85100, italy

Abstract

This paper presents an integrated natural language interface for autonomous drone control, implemented as a mobile application. The system translates unstructured spoken commands into structured drone control pseudocode, enabling intuitive execution of complex flight maneuvers and mission workflows directly from a smartphone. Additionally, it incorporates a lightweight YOLOv8s-based computer vision module optimized for mobile devices, providing real-time object detection and tracking to support vision-driven commands. The application was tested in real-world field scenarios, demonstrating effective performance in both natural language understanding and vision-based drone operations. Furthermore, the natural language-to-pseudocode translation component was evaluated using two state-of-the-art ChatGPT models on dedicated datasets comprising both single-action commands and complex mission-oriented instructions. The results demonstrated promising accuracy in parsing user instructions for drone control.

Keywords

natural language processing, large language model, drone commands, computer vision

1. Introduction

Unmanned Aerial Vehicles (UAVs) are distinguished by their capability to operate autonomously in flight without the need for a human pilot on board. Among the different types of UAVs, quadrotors have become particularly popular due to their advantageous cost-effectiveness and the wide range of readily available programming tools for developing applications [1]. These features have allowed UAVs to be utilized in various fields, such as search and rescue [2], infrastructure inspection [3], precision agriculture [4], and surveillance [5].

Despite the impressive capabilities of modern drones, the way users interact with them remains a significant barrier to wider adoption. Traditional control methods, such as radio controllers, joysticks, and touch-screen interfaces, are often not intuitive, requiring extensive training, constant attention, and refined motor skills. Additionally, the APIs provided with drones necessitate a certain level of expertise from the user. These factors have largely restricted effective drone operation to professionals with specialized skills, limiting accessibility for everyday consumers [6].

A more natural and user-friendly mode of interaction would involve communicating with drones through natural language. This would allow non-technical users to express their intent clearly and directly without needing to understand the complexities of flight control or programming logic.

Simultaneously, camera-equipped commercial drones have become increasingly common, enabling the capture of vast streams of visual data during flight. When interpreted in real-time, this data has the potential to power advanced functionalities such as object identification, obstacle avoidance, and environmental mapping. These capabilities not only can enhance the autonomy and adaptability of drones but can also significantly reduce the cognitive and technical load on the user.

To unlock this potential, we look toward Artificial Intelligence (AI), focusing on the convergence of Natural Language Processing (NLP) and Computer Vision (CV). NLP can enable drones to understand and respond to human instructions in both spoken and written forms. Our approach enhances this

CAIPI '25: ECAI Workshop on AI-based Planning for Complex Real-World Applications, October 25, 2025, Bologna, Italy

*Corresponding author.

gilda.manfredi@unibas.it (G. Manfredi); ugo.erra@unibas.it (U. Erra); nicola.capece@unibas.it (N. Capece); michele.rinaldi@unibas.it (M. Rinaldi)

0000-0003-0633-862X (G. Manfredi); 0000-0003-2942-7131 (U. Erra); 0000-0002-1544-3977 (N. Capece)

capability by leveraging Google's Speech-to-Text (STT) API for robust and accurate transcription of spoken commands, combined with zero-shot Large Language Model (LLM) methodologies that dynamically interpret natural language inputs without requiring task-specific training. The LLM, specifically OpenAI's ChatGPT, semantically parses [7] natural language inputs into drone command pseudocode, whether to execute a single action (e.g., fly straight for 10 meters) or a complex routine (e.g., trace a square). Concurrently, CV allows the drone to perceive and interpret its environment through a YOLOv8-based real-time object detection model. The synergy between these AI disciplines opens the door for autonomous, context-aware, and intuitive drones. In this context, we present the FlyAI System: a practical solution engineered to provide intuitive drone control for consumer-grade UAVs, specifically the DJI Mini 2. Developed for the Android platform, FlyAI offers various flight modalities, including traditional radio control, voice commands for individual actions, mission concatenation for executing sequences of commands, and CV-based interaction for object detection and tracking. According to the state of the art, the combination of modalities contributes to prevent errors, provides alternative communication channels, and allows disambiguation when one modality is unreliable [8, 9]. In building this system, we specifically aimed to overcome several critical challenges: the computational limitations of running AI models on resource-constrained mobile devices, integrating advanced AI functionalities with proprietary consumer SDKs (such as DJI's), and designing an intuitive interface suitable for non-expert users. These priorities guided the development of a system that is not only powerful and flexible but also accessible to a broader audience. To encourage reproducibility, we release the code and data at https://github.com/Unibas3D/FlyAI.

The remainder of this paper is structured as follows: Section 2 discusses related works in the fields of natural language interfaces and vision models applied to drone control. Section 3 presents a detailed overview of the proposed system architecture and its key components. Section 4 describes the structure of the FlyAI mobile application. Section 5 reports the experimental results, evaluating both the language understanding and perception modules. Finally, Section 6 concludes the paper and outlines potential directions for future research.

2. Related Works

The evolution of NLP has been critical in improving human-robot interaction. Early research into natural language interfaces for drones focused on speech-to-command systems that enabled users to issue basic navigation instructions verbally. Meszaros et al. [10] introduced an interface that allows users to define drone flight paths using spoken single or double-word commands, targeting non-technical users. Similarly, Chandarana et al. [11] compared speech, gesture, and mouse interfaces in flight planning, showing that while users performed better with a mouse, they were also able to engage with speech interfaces successfully and expressed interest in using them in the future. Natural language interfaces are a promising alternative, offering a more intuitive method of communication with UAVs. Although previous systems demonstrated that voice-driven control of UAVs is feasible, they often failed in terms of flexible language understanding. This was primarily due to their reliance on a predefined dictionary of acceptable words and command templates, which restricted their ability to interpret natural, unconstrained speech. However, LLMs have the potential to transform this landscape because of their advanced capabilities in understanding complex contexts and multiple modes of communication. For instance, a research study by Savenko [12] demonstrated that an LLM can semantically parse natural language text into commands for controlling a drone by employing a fine-tuned BERT [13] model. To avoid the need for fine-tuning an LLM, Asuzu et al. [14] propose using zero-shot prompting to guide the GPT-4.1 model in generating Python code for robot actions based on a specified distribution of actions. Furthermore, Aikins et al. [15] used multiple fine-tuned LLMs to convert English requests into 3D waypoints for a UAV swarm. However, having multiple LLMs can be computationally intensive and may result in latency when retrieving answers. This issue can quickly drain the UAV's batteries, compromising the time available for mission-critical tasks [16]. These systems demonstrated the feasibility of voice-driven control for UAVs. Our application extends

these ideas by integrating Google's STT API for accurate speech recognition and employing a single zero-shot LLM model for dynamic, context-aware language interpretation. Unlike approaches that rely on multiple specialized models, which can introduce significant computational overhead, latency, and battery consumption, our use of a single, general-purpose LLM avoids these issues while maintaining high flexibility. The model is prompted at runtime to generate drone command pseudo-code without prior task-specific training, whether for simple actions or complex, multi-step routines. This zero-shot approach enables a streamlined pipeline that interprets free-form natural language input, often spanning multiple sentences, and translates it into executable drone commands. As a result, the system supports sophisticated flight behavior while preserving responsiveness and energy efficiency, which is essential for mission-critical UAV applications.

Beyond language alone, multimodal systems combine speech with other inputs. One of the most used complementary modalities is CV, which enables drones to perceive and interpret their surroundings in real-time. For example, Rodríguez et al. [17] propose a CV-based solution for turbine inspection using the Haar Cascade classifier. Their system employs a DJI Tello EDU drone equipped with a front-facing RGB camera and a downward-facing infrared (IR) sensor, with data streamed to a Python application running on a connected PC. While the architecture demonstrates potential for low-cost UAV inspection, the CV algorithm has notable limitations: the Haar Cascade supports only single-class detection, relies on handcrafted features, and is sensitive to lighting variations and occlusions. Moreover, the system's dependence on a PC-based execution environment limits its portability and usability in field scenarios. Other recent studies [18, 19, 20] explore more advanced deep learning-based approaches, such as YOLO, Vision Transformers, and ResNet, to analyze aerial imagery captured by drones. While these models demonstrate good performance, they too often rely on external PCs or high-end computing platforms, which restrict their deployment in mobile or resource-constrained environments. Our approach integrates the YOLOv8 object detection model directly into a smartphone-based Android application, removing the need for off-board processing. This integration enables accurate, multi-class, real-time detection while enhancing mobility, responsiveness, and usability for non-expert users operating consumer-grade UAVs.

Regarding the combination of natural language commands and visual models, Sanyal and Roy [21] proposed a system that integrates language instructions and RGB-D input using a Vision-Language Encoder with cross-modal attention. Their system builds a language-grounded 2D semantic map of landmarks and obstacles to compute planned paths. It was implemented on a Parrot Bebop 2 quadrotor in a ROS-Gazebo simulated environment, relying on an RGB-D sensor and PC-based execution. In contrast, our approach processes spoken natural language commands, operates entirely on a smartphone, and functions without the need for depth data, relying only on RGB images. Moreover, while their evaluation was conducted in a virtual simulation, our system has been validated through real-world field testing on a consumer-grade UAV. A related effort by Sautenkov et al. [22] presents a large-scale Vision-Language-Action system that interprets text-based mission requests using satellite imagery to generate path-action sets. Although this work demonstrates impressive vision-language grounding capabilities, it shares several limitations with the previous approach: it relies on text input, and the computation is performed on a PC. In contrast, our system operates directly from live camera input and user speech in a fully embedded mobile environment, offering a more practical and responsive framework for real-world UAV navigation and task execution.

3. The System Architecture

The FlyAI system is built on a modular architecture that integrates several hardware and software components to deliver AI-powered drone control via natural language and CV. The design prioritizes compatibility with consumer-grade UAVs and emphasizes on-device processing to avoid reliance on external computing platforms. Although the system is demonstrated on a specific configuration, the architecture itself is hardware-agnostic and can be adapted to other drones and smartphones that support similar SDKs and processing capabilities. The system's hardware platform includes: (i) the

DJI Mini 2 drone, a lightweight, consumer-grade quadrotor selected for its availability, stable flight characteristics, and compatibility with the DJI SDK; and (ii) the Poco F3 Android smartphone, a mid-range device equipped with a Snapdragon 870 processor and 6 GB of RAM, running Android 13, on which all computations are performed. The development was conducted using *Android Studio* [23] (Iguana 2023.2.1 Patch 1) based on the Java programming language. To support the multimodal interaction capabilities of the FlyAI system, a range of software libraries and APIs were integrated into the Android application. Central to the functionality of drone control is the *DJI Mobile SDK* [24], which offers a complete set of APIs for interfacing with the DJI Mini 2 and managing its flight behavior, camera operations, and gimbal settings. While the present implementation relies on DJI's ecosystem, the modular software design allows alternative UAV platforms to be integrated by replacing the SDK layer with an equivalent API. As shown in Figure 1, the system converts the voice input to text using the *Google Speech Recognition API* [25], which facilitates the real-time conversion of spoken commands into text. The speech recognition is handled natively on Android using the SpeechRecognizer class, which supports streaming recognition and partial results. This API communicates via a websocket with a local server that manages the entire speech recognition pipeline: it receives the voice recording, processes and encodes the audio, and returns a text transcription. The resulting text is sent to *OpenAI ChatGPT API* [26], specifically a GPT-4.1 model, that interprets the natural language command and translates it into pseudo-code corresponding to a supported drone action. This LLM is accessed via the *Retrofit HTTP client* [27] library, which simplifies communication with RESTful APIs, while the *Gson* [28] Java library is used to manage JSON serialization and deserialization. While OpenAI's API offers powerful natural language understanding, its reliance on a cloud-based proprietary service might introduce potential limitations such as latency, availability, and data privacy concerns. To mitigate these issues, the system architecture is intentionally designed to remain compatible with alternative LLM providers or future on-device language models. This flexibility is enabled by the RESTful communication approach, which abstracts the LLM interaction behind a standardized HTTP interface, allowing different backends to be integrated without substantial modifications to the software pipeline. Furthermore, FlyAI is conceived primarily for private and research-oriented use with consumer-grade UAVs, rather than for deployment in sensitive domains such as defense or critical infrastructure, where stricter safety and data-handling requirements would apply. If the interpreted command refers to a vision-based task, such as "follow the person" or "search for a car", the system automatically invokes the *YOLOv8* object detection model [29] converted to the TensorFlow Lite format. The model analyzes the live video stream from the drone's camera in real time, searching for the requested object class. Inference is conducted using the *TensorFlow Lite Interpreter* [30] on Android, allowing the application to perform real-time object detection directly on the smartphone without requiring external computation. Once the selected class is detected, the drone adjusts its behavior accordingly.

3.1. Natural Language Interpretation Using ChatGPT

To interpret user natural language commands and convert them into drone-executable actions, the FlyAI system uses OpenAI's GPT-4.1 model through the ChatGPT API. The system generates a POST request directed to OpenAI's endpoint, which includes HTTP headers for content type and API key authentication. The body of the request contains three main components: the selected model (GPT-4.1), a temperature setting that regulates the creativity and variability of the response (typically set to a low value such as 0.2 for determinism), and a structured message list defining the conversation context. To guide the model in producing accurate and consistent pseudocode, we use a zero-shot prompting strategy (see Figure 2) consisting of three main components: (i) an initial description d_0 of the task that ChatGPT needs to perform; (ii) a set $S_c = \{c_1 : d_1, ..., c_n : d_n\}$ of drone commands along with their descriptions, which is used to instruct the model to convert natural language commands into pseudocode; and (iii) the instructions I for the drone provided by the user in natural language. The validity of the response from the language model is first checked. In detail, the system verifies that each command matches the expected pseudo-code grammar, that parameters are within operational limits, and that object references correspond to detectable classes. Empty or malformed responses

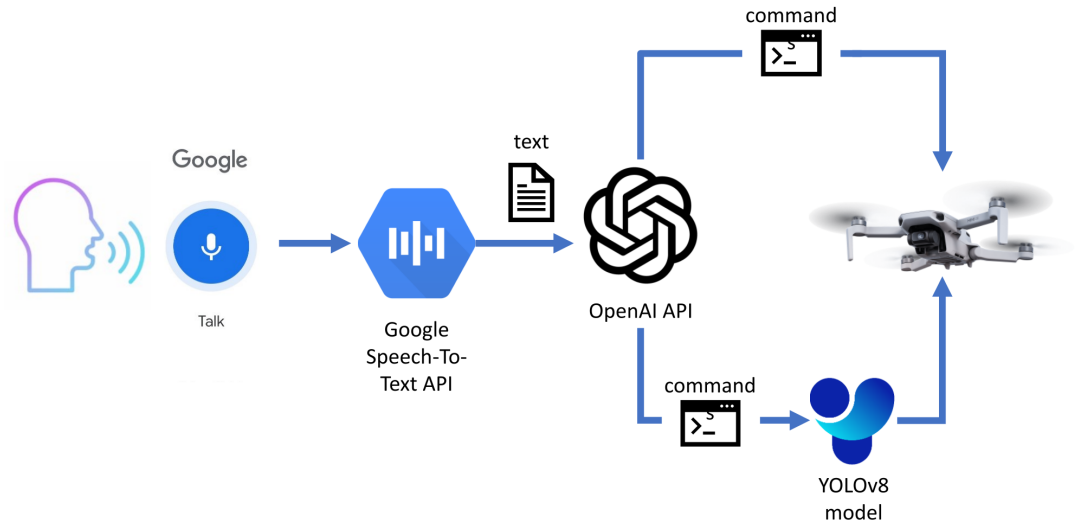


Figure 1: Pipeline for voice-command processing in FlyAI

are rejected, and the user is notified that the instruction is invalid or unsupported. In the case of single commands, the system requests user confirmation before execution of a valid instruction via the command dispatcher to prevent unintended actions.

For mission execution, where multiple commands are generated in a defined sequence, the system introduces an additional verification step to enhance robustness and user control. Upon receiving the list of instructions from ChatGPT, the application displays a confirmation dialog listing the interpreted commands. The user may choose to accept or decline the mission. If the user declines, the mission is aborted, and a notification is shown. If accepted, the mission execution is initiated, and sequential command execution is managed in a dedicated background thread. This design ensures commands are executed serially without concurrency issues. In this thread, the command list contained in the generated pseudocode is iterated through, executing the first command and waiting for its completion before proceeding to the next command. This synchronization mechanism ensures that each command is completed before the next one begins, thereby preserving the intended temporal structure of the mission.

To support emergency interruptions or dynamic halting, the system monitors a boolean flag that is set to true if the user activates the on-screen Stop button during mission execution. The executor checks this flag at each step, and if triggered, it halts the sequence and alerts the user that the mission was aborted.

3.2. Vision Mode

To integrate CV capabilities into the drone control system, a pre-trained YOLOv8 model was adopted, specifically the lightweight yolov8s variant, which supports around 80 object classes. Since this model is originally incompatible with Android environments, it was first converted to TensorFlow Lite (TFLite) format and then imported into the mobile application along with the associated label file. Once integrated, real-time object detection is performed by processing the live video feed from the drone's onboard camera. Each video frame is captured, resized to the model's input resolution (typically 640×640), and converted into a TFLite-compatible tensor format by normalizing pixel values and ensuring that the data format matches the expected floating-point type. Furthermore, GPU acceleration is enabled on supported devices to enhance the efficiency of inference operations.

The application captures and decodes the drone's video stream using the media interface provided by the DJI SDK. Decoded frames are rendered onto a graphical texture interface, which also acts as a

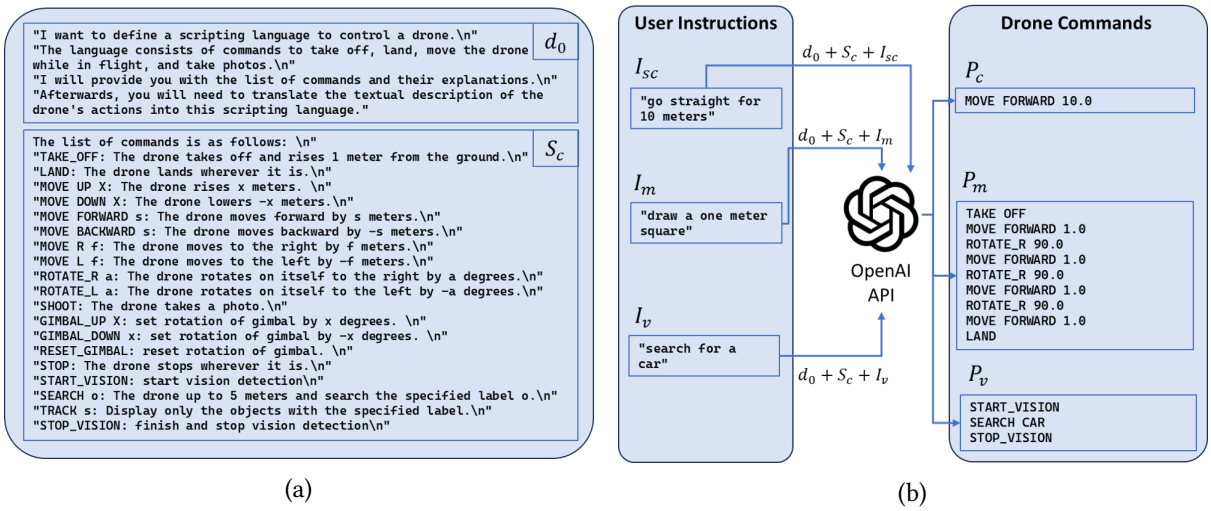


Figure 2: Zero-Shot ChatGPT Strategy. Figure (a) shows the initial description d_0 of the task that ChatGPT must perform and a set $S_c = \{c_1 : d_1, ..., c_n : d_n\}$ of drone commands with their descriptions. Figure (b) shows the user's spoken commands in natural language, and the outputs from ChatGPT are represented. I_sc stands for a single command instruction, I_m indicates a mission command, and I_v indicates a vision command.

source for object detection. For each updated frame, inference is triggered, and the output, which is a set of detected bounding boxes with confidence scores and class labels, is processed and optionally filtered based on user-specified criteria.

Detected objects are overlaid visually on the video preview using a custom rendering layer. This overlay dynamically draws bounding boxes and class labels for each recognized object, updating in real-time as new frames are processed. The drawing system is designed to be responsive and visually distinguishable, enhancing user interaction and situational awareness.

3.3. Command Parsing

To support structured command execution, the system implements a switch-case structure that maps each pseudo-command to a corresponding drone control method. When a natural language instruction is interpreted by the LLM into pseudo-code (for example, `MOVE FORWARD TO 10 m`) the parser identifies the action name (`MOVE FORWARD`) and its parameters (`10 m`) and invokes the corresponding method that calculates the appropriate velocity vector and sends the corresponding VirtualStick control command through the DJI SDK. If the parser fails to find a matching command or detects malformed parameters, the system triggers an error handler that informs the user that the instruction is invalid or unsupported, preventing unintended execution. To promote modularity and maintainability, the application employs the Command Design Pattern. This behavioral software design pattern encapsulates each request as an object, allowing commands to be parameterized, logged, extended, or deleted independently of the invoker. Each command implements a common interface and is executed via a uniform `run()` method. This approach ensures that new commands can be integrated into the system without changing existing logic. Commands are parameterized and treated uniformly, whether they represent simple actions (e.g., LAND, TAKE OFF) or are part of a multi-step mission. To manage mission-mode operations, which involve the sequential execution of multiple commands, the system maintains a command execution status flag. This flag indicates whether a command is currently being executed, allowing the system to queue or delay subsequent commands until the ongoing operation is complete. This mechanism prevents command overlaps and ensures temporal synchronization in chained actions (e.g., TAKE OFF $\rightarrow$ MOVE FORWARD 5 $\rightarrow$ LAND). Once a command is completed, the flag is reset, and the next action in the sequence is triggered.

Certain commands, namely UP, DOWN, MOVE, and ROTATE, require special handling due to the interface provided by the DJI SDK. Indeed, the structure used to control the drone's motion accepts four param-

eters: pitch, roll, yaw, and verticalThrottle, each representing velocity or angular rate depending on the control mode. However, users typically express movement instructions in terms of distances or angles rather than speeds. For example, a user might request the drone to "move forward 10 meters" or "rotate right by 90 degrees". As a result, the system must internally convert these spatial instructions into time-bound velocity commands to achieve the desired displacement or rotation. To achieve this, the system utilizes a periodic timer mechanism, where a Timer object is configured to send control commands every 100 milliseconds until the drone reaches the specified target state.

For the UP and DOWN commands, altitude control is managed by invoking a function that accepts the target altitude as input. If the drone's current altitude is below the target, the system issues a continuous ascent command at a fixed vertical speed (typically $1m/s$); conversely, if it is above the target, a descent command is issued. The altitude is monitored in real time, and the drone is stopped once the desired height is reached. In these cases, the lateral movement components (pitch and roll) are set to zero, the yaw component represents the vertical velocity in meters per second, and the verticalThrottle parameter is initialized to the current yaw angle of the drone before the vertical movement begins, ensuring the drone maintains its original heading throughout the maneuver.

Furthermore, the MOVE command, used to translate the drone along the forward, backward, or lateral axes, is executed by computing the required movement duration based on the specified speed and distance. In detail, the pitch and roll components' values are computed based on the specified movement direction and magnitude, taking into account a constant velocity for temporal estimation. In this configuration, the verticalThrottle parameter is set to zero to restrict motion to the horizontal plane. Crucially, the yaw parameter is initialized to the drone's current heading at the moment the command begins. This ensures that the drone maintains its original orientation throughout the maneuver, regardless of the direction of movement.

For ROTATE commands, angular control is achieved by estimating the required time to perform the rotation. This is computed using the formula:

$$t = \frac{|\theta|}{|v_{\text{yaw}}|} \times 1000 \tag{1}$$

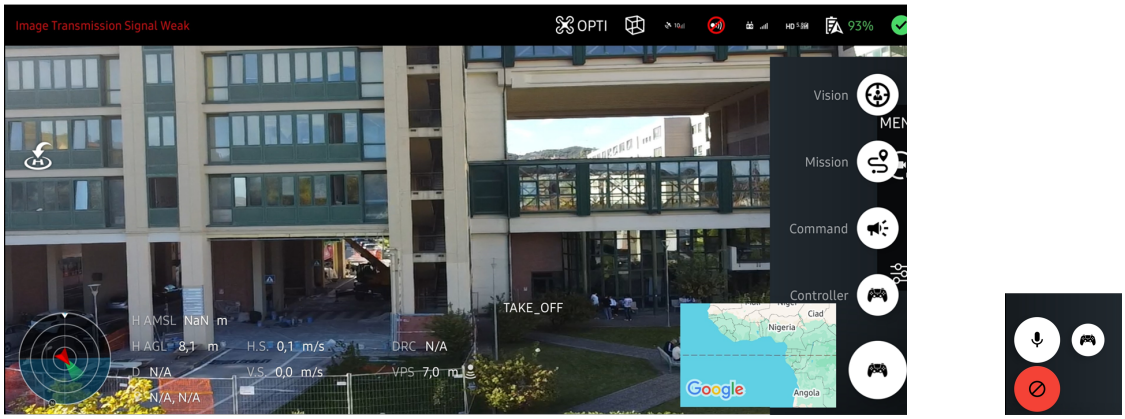
where θ is the target rotation angle in degrees (positive for clockwise, negative for counter-clockwise), v_{yaw} is the fixed angular velocity (30°/s), and t is the rotation time in milliseconds. During this interval, the drone receives continuous commands with the yaw parameter set accordingly, and all other motion parameters held at zero.

In addition to standard navigation instructions, the system supports a set of specialized commands that enable CV-based operations. In particular, the START_VISION and STOP_VISION commands respectively initialize and terminate the vision subsystem. Internally, these commands handle the loading and unloading of the TensorFlow Lite model and associated label file, as well as the GPU delegate for efficient inference on mobile devices.

The TRACK command enables the drone to detect and highlight only specific object classes, such as "Person", in the live video stream. Upon receiving this command, the system verifies whether the requested label exists among the model's known categories. If valid, it begins filtering the detection output to display only the bounding boxes corresponding to the specified objects (see Figure **??**). The SEARCH command defines a more autonomous behavior, where the drone performs a pre-defined sequence to locate a particular object in its surroundings. Upon invocation, the drone ascends to a predetermined height (typically 3 meters), tilts the gimbal down by 45 degrees, and begins rotating in place. Simultaneously, it activates the object detection pipeline with the user-specified label. If the label is found within a bounding box, the drone performs a small additional rotation to ensure visibility of the object. He notifies the user that the target has been detected. If no detection occurs after a full 360-degree rotation, the system informs the user that the search was unsuccessful.

All vision commands are integrated into the same command processing architecture described earlier, allowing them to benefit from modular execution, parameter handling, and queuing logic used across the system.

Figure 3: Execution of the TRACK command in the mobile application. The camera stream from the drone is displayed in real time, with bounding boxes highlighting only the objects corresponding to the selected "car" label.



(a) (b)

Figure 4: Main Activity of the FlyAI mobile application. Figure (a) shows the main buttons related to flight modalities. In particular, The "Controller" button is related to the *Controller Mode*, the "Command" button is related to the *Voice Command Mode*, the "Mission" button is related to the *Voice Mission Mode*, and the "Vision" button is related to the *Object Detection Mode*. Figure (b) shows the Control Sub-Buttons. The "Mircophone" button enables the voice-driven interaction, the "Controller" button disables the VirtualStick mode, and the "Stop" button interrupts any ongoing drone operation.

4. The Mobile Application

The FlyAI mobile application operates as the central interface for connecting, monitoring, and controlling the drone, organized to support multiple modes of human–drone interaction. At the architectural level, the operational dashboard is represented by the *Main Activity* (Figure 4), which integrates the real-time video stream, orientation indicators, and mode-switching controls. Four distinct interaction modes are supported: (i) *Controller Mode*, for manual piloting; (ii) *Voice Command Mode*, for executing atomic spoken commands; (iii) *Voice Mission Mode*, for multi-step natural language instructions; and (iv) *Object Detection Mode*, for vision-driven control tasks. This modular design allows for the use of different interaction paradigms within a unified platform. Safety and reliability are ensured through two dedicated controls (Figure 4b): the *Controller Button*, which restores manual operation by disabling programmatic control through the DJI VirtualStick API, and the *Stop Button*, which halts ongoing drone operations by setting all flight control parameters to zero. These safeguards were critical during system testing, allowing us to evaluate AI-driven modes without compromising operator or equipment safety.

NL Command	GT	Prediction
move backward by 9.0 meters	MOVE BACKWARD 9.0	MOVE BACKWARD 9.0
rotate right by 17.5 degrees	ROTATE_R 17.5	ROTATE_R 17.5
move to the right by 3.6 meters	MOVE R 3.6	MOVE R 3.6
lower by 4.8 meters	MOVE DOWN 4.8	MOVE DOWN 4.8

Table 1
Example entries from the single command dataset, showing natural language input commands, the expected (ground truth) pseudocode, and the corresponding predictions generated by the model.

5. Results

The proposed system was evaluated through a combination of language understanding and visual recognition tasks, reflecting its dual objectives of enabling intuitive drone control based on natural language and supporting real-time visual awareness. To assess the accuracy of the language interface, two custom datasets were constructed. The first dataset consisted of 50 atomic natural language commands, each directly mappable to a single pseudocode instruction. The second dataset comprised 50 multi-step missions expressed in natural language, where each instruction corresponded to a sequence of pseudocode commands. Both datasets were processed by two OpenAI state-of-the-art models, such as GPT-4o and GPT-4.1. Predictions were marked as correct not only when they exactly matched the expected output but also when they were semantically and syntactically equivalent, allowing for minor rewordings (e.g., BACKWARD 5.4 being considered valid for MOVE BACKWARD 5.4). Accuracy was calculated by counting the proportion of correctly matched samples.
As a result, in the single-command dataset (see Table 1), GPT-4o achieved an accuracy of 95.5%, while GPT-4.1 outperformed it with a 100% match rate. When tested on the more complex mission dataset (see Table 2), the gap widened: GPT-4o reached 80% accuracy, while GPT-4.1 maintained robust performance at 98%. Analysis of the errors indicated that GPT-4o occasionally misinterpreted vision labels or created different drone paths in composite missions compared to the requested shape. In contrast, GPT-4.1 demonstrated a better understanding of temporal sequences and control logic.

In terms of vision performance, the YOLOv8s model was evaluated across varied real-world scenarios and delivered reliable results in standard conditions. However, detection accuracy was found to vary significantly based on the drone's altitude. Specifically, when the drone operated at elevations higher than 5 meters, the top-down perspective and reduced object resolution led to increased misclassifications, for example, trees or bushes were occasionally identified as people. These inaccuracies affected the execution of commands such as SEARCH `car`, where a false positive caused the system to halt the search prematurely under the assumption that the target had been found. These findings indicate that object detection performance is not uniform across flight altitudes.
Additional issues arose in the voice interaction layer. While it delivered highly accurate transcriptions in almost quiet environments, it sometimes struggled with command parsing. This was due to misrecognizing words or significantly altering the sentence structure in noisy settings or environments with multiple speakers.

Flight behavior was also affected by low-level control anomalies. In rare cases, even when initialization parameters were set correctly, specific commands were not faithfully executed. For example, during rotation maneuvers, the drone sometimes exhibited a slight unintended descent, compromising positional stability. Furthermore, sequences involving chained movements, such as ROTATE_R 90 followed by MOVE FORWARD 2, occasionally resulted in incomplete displacements. In observed trials, the drone would rotate correctly but then only move forward by 1 meter instead of the requested 2 meters. This inconsistency likely stems from a delayed update of the drone's internal state or a lag in the notification of rotation completion, resulting in the forward command being issued prematurely.

Despite these limitations, end-to-end evaluations across real-world missions showed that the system reliably interpreted and executed natural language instructions in $\approx 94\%$ of single-action cases and $\approx 89\%$ of multi-step missions. Tasks involving integrated vision (e.g., "trace a circle and find a car") were completed successfully in $\approx 92\%$ of trials. These results are summarized in Table 3, which

NL Command	GT	Prediction
rise, move forward 10 meters, then land	TAKE_OFF MOVE UP 2 MOVE FORWARD 10 LAND	TAKE_OFF MOVE UP 1 MOVE FORWARD 10 LAND
draw a square with sides of 5 meters	TAKE_OFF MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 LAND	TAKE_OFF MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 MOVE FORWARD 5 ROTATE_R 90 LAND
start vision mode, track a person, and stop it	START_VISION TRACK person STOP_VISION	START_VISION TRACK person STOP_VISION
take off, look down 60 degrees with the gimbal, search for a bicycle, and take a picture if detected	TAKE_OFF GIMBAL_DOWN 60 START_VISION SEARCH bicycle SHOOT RESET_GIMBAL STOP_VISION LAND	TAKE_OFF GIMBAL_DOWN 60 START_VISION SEARCH bicycle SHOOT STOP_VISION

Table 2
Example entries from the mission dataset, showing natural language input commands, the expected (ground truth) pseudocode, and the corresponding predictions generated by the model.

provides a comparison of LLM accuracy and real-world task success across single-command, multi-step, and vision-integrated tasks. Overall, these findings confirm the feasibility of a unified, voice- and vision-enabled drone control architecture.

Task Type / Dataset	GPT-4o Accuracy (%)	GPT-4.1 Accuracy (%)	Real-World Task Success (%)
Single-command dataset	95.5	100	94
Multi-step mission dataset	80	98	89
Vision-integrated tasks (single + mission)	–	–	92

Table 3
LLM accuracy and real-world task success rates across different task types. Vision-integrated tasks include both single-command and multi-step missions.

In addition to success rates, we also evaluated runtime and energy efficiency. On average, single commands were processed with low latency (≈ 1.3 seconds), although instructions requiring integrated vision incurred higher delays (≈ 3.5 seconds) due to the overhead of YOLO-based object detection. The overall duration of each mission naturally depended on the number of commands executed, with longer sequences proportionally increasing flight time. Battery monitoring during test flights showed a decrease of approximately 16% over a 6-minute mission, reflecting the additional demand introduced by continuous perception and command interpretation. These results suggest that the system remains responsive and energy-feasible for short- to medium-length autonomous missions.

6. Conclusions

This work presented an integrated natural language interface for autonomous drone control, combining speech recognition, LLM, and CV. Experimental results demonstrated that the GPT-4.1 model achieved high accuracy in translating simple natural language commands into drone-executable pseudocode, significantly outperforming GPT-4o, particularly in the interpretation of complex, mission-oriented instructions. These findings underline the ability of LLMs in bridging the gap between unstructured human language and structured robotic control logic. Nonetheless, several challenges remain. Although GPT-4.1 demonstrated overall reliability, certain edge cases exposed limitations in interpreting complex mission instructions, particularly those involving structured path planning. This suggests opportunities for further improvement through targeted prompt tuning and dataset augmentation with a broader set of linguistic variations. On the perception side, the YOLOv8s object detection model demonstrated robust performance under typical conditions. However, performance degraded when detecting small or partially occluded objects from top-down perspectives. For example, vegetation was sometimes misclassified as a human, causing premature termination of vision-related commands. Such limitations underscore the need for model retraining on aerial imagery or integration of adaptive detection strategies that account for perspective and scale. In conclusion, this work provides a promising foundation for natural language-driven drone interaction, offering intuitive control over both navigation and perception tasks. Future research should focus on improving multi-step command interpretation, enhancing object detection under aerial constraints, and ensuring robustness in noisy or ambiguous user input. Extending beyond the current fixed command set and object labels, future work should enable a broader instruction coverage, customizable object classes, and incorporate more advanced detection logic to distinguish between multiple instances of the same class. Since detection accuracy can suffer at higher altitudes, integrating altitude-aware adjustments or leveraging state-of-the-art approaches that adapt YOLO for long-range detection of small aerial targets could significantly improve robustness [31, 32]. Another promising direction is the integration of the Model Context Protocol (MCP) framework, providing a structured, machine-readable representation of drone actions that allows the LLM to verify, plan, and explain commands, thereby improving interpretability, reliability, and safety in autonomous operations. Additionally, future work will include evaluation on user-generated commands to assess the NLP component's robustness and generalization across diverse linguistic styles and task scenarios. As language models and onboard vision systems continue to advance, the integration of these technologies into autonomous robotics holds potential for human–machine collaboration.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] L. Domingos, P. Santos, A semantics of spatial expressions for interacting with unmanned aerial vehicles, in: P. Parameswaran, J. Biggs, D. Powers (Eds.), Proceedings of the 20th Annual Workshop of the Australasian Language Technology Association, Australasian Language Technology Association, Adelaide, Australia, 2022, pp. 148–155.

[2] Z. A. Elashaal, Y. H. Lamin, M. K. Elfandi, A. A. Eluheshi, Autonomous search and rescue drone, Libyan Journal of Informatics 1 (2024) 1–17.

[3] P. Aela, H.-L. Chi, A. Fares, T. Zayed, M. Kim, Uav-based studies in railway infrastructure monitoring, Automation in Construction 167 (2024) 105714.

[4] R. Guebsi, S. Mami, K. Chokmani, Drones in precision agriculture: A comprehensive review of applications, technologies, and challenges, Drones 8 (2024) 686.

[5] S. M. Gilani, A. Anjum, A. Khan, M. H. Syed, S. A. Moqurrab, G. Srivastava, A robust internet of

drones security surveillance communication network based on iota, Internet of Things 25 (2024) 101066.
[6] B. Huang, D. Bayazit, D. Ullman, N. Gopalan, S. Tellex, Flight, camera, action! using natural language and mixed reality to control a drone, in: 2019 International conference on robotics and automation (ICRA), IEEE, 2019, pp. 6949–6956.
[7] P. Jiang, X. Cai, A survey of semantic parsing techniques, Symmetry 16 (2024) 1201.
[8] A. Jaimes, N. Sebe, Multimodal human–computer interaction: A survey, Computer Vision and Image Understanding 108 (2007) 116–134. doi:https://doi.org/10.1016/j.cviu.2006.10.019, special Issue on Vision for Human-Computer Interaction.
[9] N. Capece, M. Gruosso, U. Erra, R. Catena, G. Manfredi, A preliminary investigation on a multimodal controller and freehand based interaction in virtual reality, in: Augmented Reality, Virtual Reality, and Computer Graphics: 8th International Conference, AVR 2021, Virtual Event, September 7–10, 2021, Proceedings 8, Springer, 2021, pp. 53–65.
[10] E. L. Meszaros, M. Chandarana, A. Trujillo, B. D. Allen, Speech-based natural language interface for uav trajectory generation, in: 2017 International Conference on Unmanned Aircraft Systems (ICUAS), IEEE, 2017, pp. 46–55.
[11] M. Chandarana, E. L. Meszaros, A. Trujillo, B. D. Allen, 'fly like this': Natural language interface for uav mission planning, in: International Conference on Advances in Computer-Human Interactions, NF1676L-26108, 2017.
[12] I. Savenko, Command interpretation for uav using language models, in: 2024 IEEE 7th International Conference on Actual Problems of Unmanned Aerial Vehicles Development (APUAVD), IEEE, 2024, pp. 228–231.
[13] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL: https://arxiv.org/abs/1810.04805. arXiv:1810.04805.
[14] K. Asuzu, H. Singh, M. Idrissi, Human–robot interaction through joint robot planning with large language models, Intelligent Service Robotics (2025) 1–17.
[15] G. Aikins, M. P. Dao, K. J. Moukpe, T. C. Eskridge, K.-D. Nguyen, Leviosa: Natural language-based uncrewed aerial vehicle trajectory generation, Electronics 13 (2024) 4508.
[16] S. Javaid, H. Fahim, B. He, N. Saeed, Large language models for uavs: Current state and pathways to the future, IEEE Open Journal of Vehicular Technology (2024).
[17] A. A. Rodriguez, M. Shekaramiz, M. A. Masoum, Computer vision-based path planning with indoor low-cost autonomous drones: An educational surrogate project for autonomous wind farm navigation, Drones 8 (2024) 154.
[18] D.-A. Pham, S.-H. Han, Deploying a computer vision model based on yolov8 suitable for drones in the tuna fishing and aquaculture industry, Journal of Marine Science and Engineering 12 (2024) 828.
[19] Y. Wang, Y. Wang, C. Xu, X. Wang, Y. Zhang, Computer vision-driven forest wildfire and smoke recognition via iot drone cameras, Wireless Networks 30 (2024) 7603–7616.
[20] G. Muradas Odriozola, K. Pauly, S. Oswald, D. Raymaekers, Automating ground control point detection in drone imagery: From computer vision to deep learning, Remote Sensing 16 (2024) 794.
[21] S. Sanyal, K. Roy, Asma: An adaptive safety margin algorithm for vision-language drone navigation via scene-aware control barrier functions, arXiv preprint arXiv:2409.10283 (2024).
[22] O. Sautenkov, Y. Yaqoot, A. Lykov, M. A. Mustafa, G. Tadevosyan, A. Akhmetkazy, M. A. Cabrera, M. Martynov, S. Karaf, D. Tsetserukou, Uav-vla: Vision-language-action system for large scale aerial mission generation, arXiv preprint arXiv:2501.05014 (2025).
[23] Android, Android Studio, 2023. https://developer.android.com/studio, last accessed on 2025-07-01.
[24] DJI, DJI Mobile SDK, 2025. https://developer.dji.com/mobile-sdk/, last accessed on 2025-07-01.
[25] Google, Google Speech Recognition for Android, 2025. https://developer.android.com/reference/android/speech/package-summary, last accessed on 2025-07-01.
[26] OpenAI, OpenAI ChatGPT-4, 2025. https://platform.openai.com/docs/models/chatgpt-4o-latest, last accessed on 2025-07-01.
[27] Retrofit, Retrofit, 2025. https://square.github.io/retrofit/, last accessed on 2025-07-01.

[28] Google, Gson, 2025. https://github.com/google/gson, last accessed on 2025-07-01.
[29] Ultralytics, YOLOv8, 2025. https://docs.ultralytics.com/models/yolov8/, last accessed on 2025-07-01.
[30] Tensorflow, Tensorflow Lite interpreter, 2025. https://ai.google.dev/edge/litert/android/java?hl=en, last accessed on 2025-07-01.
[31] H. H. Nguyen, M. S. Hoang, et al., Leaf-yolo: Lightweight edge-real-time small object detection on aerial imagery, Intelligent Systems with Applications 25 (2025) 200484.
[32] F. Feng, L. Yang, Q. Zhou, W. Li, Yolo-tiny: A lightweight small object detection algorithm for uav aerial imagery, IET Image Processing 19 (2025) e13314.