

A Universal Large Language Model - Drone Command and Control Interface

Javier Ramos-Silva^{1,2*} and Peter Burke^{2,3}

^{1*}Department of Electrical Engineering and Computer Science,
University of California, Irvine, MS 2625, Irvine, 92697, CA, USA.

^{2*}BME, MSE, CBEMS, Physics, University of California, Irvine, MS
2625, Irvine, 92697, CA, USA.

*Corresponding author(s). E-mail(s): pburke@uci.edu;
Contributing authors: pburke@uci.edu;

Abstract

The use of artificial intelligence (AI) for drone control can have a transformative impact on drone capabilities, especially when integrated with drone sensing, command, and control, part of a growing field of physical AI. Large language models (LLMs) for agentic drone control provide a new opportunity for drone autonomy. However, challenges remain in the interface, with each use case requiring a tedious, labor intensive effort to connect the LLM to the drone. Here, using the new model context protocol (MCP), we solve that problem, by developing and demonstrating in flight and simulation an LLM agnostic and drone agnostic approach, providing the first universal, versatile, comprehensive and easy to use drone control interface. This provides a universal way for AI systems to access drone data, tools, and services. Our cloud based MCP server supports the Mavlink protocol, an ubiquitous drone control language used almost universally by millions of drones including Ardupilot and PX4 framework. We demonstrate integration with multiple commercial including all major frontier models and open source LLMs. In further testing, we demonstrate extensive flight planning and control capability in a simulated drone, integrated with a Google Maps MCP server for up to date, real time navigation information. This demonstrates a universal approach to integration of LLMs with drone command and control, a paradigm that leverages and exploits virtually all of modern AI industry with drone technology in an easy to use interface that translates natural language to drone control.

Keywords: Large language model (LLM), drone, unmanned aerial vehicles (UAVs), unmanned aircraft systems (UAS)

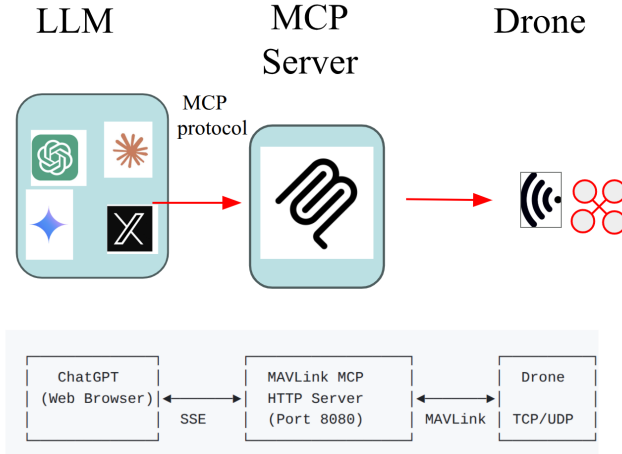


Fig. 1 Architecture. Any LLM that supports the MCP standard can connect to the MCP server, which in turn provides a low level interface with a drone using Mavlink protocol.

1 Introduction

Autonomy in drones is usually based on onboard image and state processing from onboard sensors[1]. An outstanding example is the 2023 AI controlled racing drone[2] that beat a human drone pilot, similar to the IBM supercomputer beating the world chess master in the 1990s. However, challenges remain in the amount of compute power onboard due to weight and energy restrictions for all drone classes and sizes, especially small drones. With the coming of age of internet connected drones, using AI in the cloud opens new opportunities to harness the virtually unlimited compute power of data centers deployed globally. LLMs are one such technology, trained at scale. To date, the interface between the AI LLM and the drone has been an unsolved challenge for general use cases. Here we show, using the model context protocol (MCP) standard, a comprehensive and complete drone control interface using the Mavlink protocol for drone communication. We demonstrate autonomous control of a real world internet connected drone (with real-time dynamic decision making by the LLM), and demonstrate more comprehensive missions with a virtual (simulated) drone. This approach is key to unlocking the power of AI from the virtual world to the physical world.

2 Architecture

Fig. 1 shows the architecture, and Fig. 2 the concept of operations, designed to bridge the virtual world of AI with the material (physical) world of drones.

2.1 Concept of operations

Any LLM (private proprietary or open source, cloud or locally hosted) that supports the MCP standard can connect to the MCP server developed in this work, which we call "DroneServer".



Fig. 2 Concept of operations. An LLM has access to multiple services, tools, and MCP servers. One of them is this drone control server, but the LLM can access any of thousands of other MCP servers. An example mission is shown integrating both a map (Google maps) and drone control server.

A list of LLMs as of the writing of this paper that support the MCP server architecture is:

- **Anthropic Claude** – Native support, including Claude 3.5 Sonnet, Claude Opus 4, and Claude Sonnet 4.5. Claude Desktop app has built-in MCP client support.
- **OpenAI** – Adopted MCP in March 2025. Works with GPT-4, GPT-4 Turbo, and the GPT-5 series models. The OpenAI Agents SDK includes MCP support.
- **Google Gemini** – Announced support in April 2025. Works with Gemini 2.0 Flash, Gemini 2.5 Pro, and Gemini 2.5 Flash. Google launched production-ready managed MCP servers in December 2025.
- **Local/Open Source Models** – Any LLM that supports function calling can work with MCP, including:
 - Llama 3.2 and Llama 3.3 70B
 - Qwen 2.5 72B
 - Models running on Ollama (such as qwen2.5:7b)

In turn, the MCP server can connect to any drone that supports the Mavlink protocol and has an internet connection. This includes PX4 and Ardupilot, the two

largest drone software packages, already used on millions of drones. The MCP server handles all drone communications and provides LLMs with a description of capabilities suitable for the context. We describe each component of the architecture in detail next.

2.2 Mavlink protocol

Mavlink[3] is the protocol used by open source autonomous drones running Ardupilot and PX4. This includes millions of drones (land, air, sea) and the vast majority of the global fleet of open source drones.

Mavlink supports hundreds of commands for drone control and telemetry, and is supported by libraries in most common modern high level programming languages such as C, Python, etc.

2.3 Model context protocol: From manual tools to universal standard LLM interface

2.3.1 Tools: Tedious and manual agentic AI programming

A large language model (LLM) supports inference and text prediction based on tokens, but does not otherwise perform any function. Recently, LLMs have been programmed with the ability to call "tools" in order to interact outside the context of pure text. This enables LLMs to perform as agents, accessing internet content, reading and modifying files and databases, and even writing code. Typically, each tool is hand coded or formalized as an API, a process that requires manual attention for each set of tools and each variation of an LLM.

2.3.2 Model Context Protocol

In order to enable more seamless connection between LLMs and tools, the model context protocol concept was introduced by Anthropic in Nov. 2024[4]. In December 2025, Anthropic donated the MCP standard the Linux Foundation, in order to secure it as an open, industry wide standard[5]. The MCP serves as a layer between LLMs and the rest of the virtual world by standardizing the LLM communication protocol. An MCP server can connect any LLM to any tool. MCP is now supported by all the major LLM providers (OpenAI, Anthropic, Gemini). The prime use cases initially envisioned included interaction between LLMs and (broadly) the virtual world, such as banking, finance, files, databases, websites, and social media, even to write and modify code. Fig. 3 shows the detailed specification of the MCP standard.

3 Development method

The development method was based primarily on Cursor IDE, a fork of Microsoft's Visual Studio Code, which allows access to modern coding tools including frontier models such as Anthropic's Claude, Google Gemini, and OpenAI ChatGPT. All of the code was generated this way. This is an extension of the concept that we recently developed and published[6]. This work is forked as a major extension,

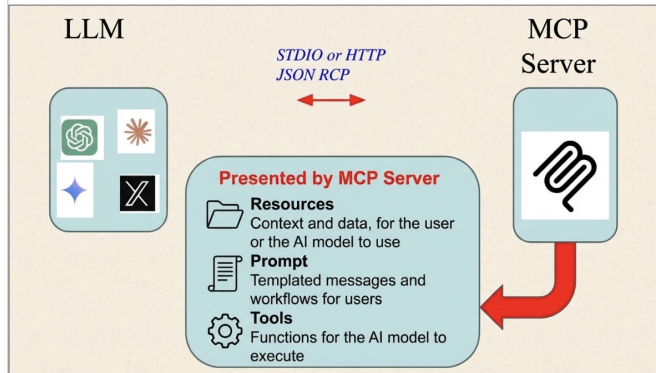


Fig. 3 Model context protocol (MCP) standard. The MCP server exposes resources, prompts, and tools to the LLM. The LLM does not need to know the details of the implementation of these, and uses them based on the context of the prompt.

test and demonstration of the original project basic MavlinkMCP (github.com/ion-g-ion/MAVLinkMCP), which had 10 tools. This project has 15,000 lines of code, 45 tools, detailed install instructions, as well as systemd service files to enable the MCP server to run continuously in the background without the need for manual start. Github was used for continuous development (as recommended in [6]), and provides detailed usage instructions. Testing was performed on both a virtual and a real drone, described in more detail below.

4 Server architecture

Fig. 4 shows the detailed architecture (the "tech stack") used in this paper. The entire server is hosted on a cloud Linux Ubuntu instance. An MCP server (the main contribution of this work) is coded using primarily the Python language, deployed on a cloud linux instance, and connected to a drone. The MCP server tells the LLM what support it provides for the LLM to control and get information about the drone. This is high level, in the next section we dive deep into the MCP server design and implementation.

The MCP server consists of a set of code run on a cloud Linux instance with internet connectivity. The codebase is custom for this work and is available on github at github.com/PeterJBurke/droneserver. The codebase is mostly written in Python, because of all of the packages available both for the drone command and control interface, and the LLM interface. Fig. 4 shows in detail the tech stack used by the MCP server. We describe each below.

4.1 Internals

The internal code provides custom, hand coded interface between the drone and the LLM. This write once, use multiple times provides drone pilots and LLM context engineers with a pre-configured interface, so the hand coding is not necessary. Natural language prompts are automatically translated into drone command and control

interactions. This represents the first versatile, universal interface between any drone supporting Mavlink (which is most of them), and any LLM supporting MCP. Thus, the impact is expected to be vast for applications of individual single drones with the significant resources of LLMs, as well as at scale with coordinated swarms of drones and swarms of agents.

4.2 LLL-MCP server link

The MCP Python SDK[7] is the official Python implementation of the Model Context Protocol, serving as a framework for building servers that expose tools, resources, and prompts to AI applications like Claude. It enables developers to build MCP servers that expose resources, tools, and prompts to Large Language Models in a standardized way. Once deployed, it provides a standardized way to create integrations that allow AI assistants and agents to interact with external systems, databases, APIs, and services through a well-defined protocol. It uses standard transports like stdio, SSE, and Streamable HTTP for communication. (Here, we use HTTP). It is the primary machine that provides context and LLM interaction.

The package includes base classes and utilities for defining server capabilities, handling client connections, and managing the request/response lifecycle. Developers can use it to build custom MCP servers that extend an AI's capabilities - for example, connecting to databases, file systems, or third-party APIs - while maintaining a consistent interface that MCP-compatible clients can consume.

4.3 Drone to MCP server link

4.3.1 Low level Mavlink and TCP/IP are abstracted from the LLM

Starting from the bottom, the low level bit movement from the MCP server to the drone is through TCP/IP. At the next level up, the Mavlink protocol is used. Mavlink messages are very low level messages, and there are hundreds of messages defined. Although it would be possible to code all of them up into the server, the LLM may not need or be able to handle such a fine grained control of the drone detailed state and configuration. In addition, it may be too taxing on the context window (see below) to define hundreds of tools, one for each Mavlink message. This is discussed in more detail below in the section called "Number of tools vs context size". The other reason is that having a tool for each Mavlink message type would be much lower level than a typical use case for an LLM, which we envision as integrating at a higher level other training data and even other MCP servers and tools for integrative systems meta-engineering rather than low level control such as throttle setting, bank angle, etc. Therefore, in this work, we did not use Mavlink as the base set of commands to expose to the LLM.

4.3.2 MavSDK is a higher level set of commands used for this work

While Mavlink is the protocol, there are two Python packages that handle communications, links, and provide higher level commands and methods within the Python application. These are Pymavlink and MavSDK. Pymavlink provides a direct python implementation of the MAVLink protocol. For example, PyMavlink can be used to

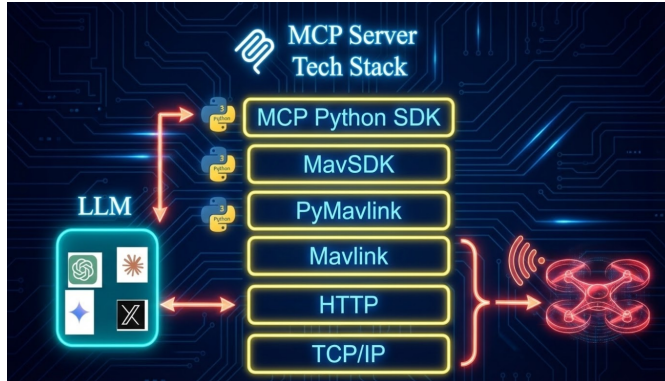


Fig. 4 Tech stack. The tech stack of the MCP server developed in this work. The drone communicates over TCP/IP using Mavlink protocol, while the LLM communications over HTTP using the MCP protocol. The server contains custom code to coordinate all the interactions to provide seamless integration between the LLM and the drone.

set the throttle or read the attitude and IMU. MavSDK is higher level abstraction, which enables more mission oriented commands, such as “go to xyz location” “take off” “land”. MavSDK also handles establishing and maintaining the communications links under the hood. Therefore, we used MavSDK in this work. Our model exposes the key MavSDK methods as tools to the LLM via the MCP server, discussed next.

MavSDK provides several high level commands such as “take off to xxx meters”, or “fly to xyz position”, commands which are not available as simple Mavlink commands. Of the 155 MavSDK methods, we chose a subset to implement in this initial work. Implementing all of the methods was not deemed necessary at this time, and this would likely consume too many tokens in the LLM context (see below). Table B1 in the appendix lists all 155 of the methods in MavSDK (grouped by major function), whether they are exposed as tools to the LLM, and a brief description of their tasks. Of the 155, 40 are exposed. Table 1 below shows as a summary how many of each class are exposed. We discuss the selection process in more detail below.

4.4 Beyond one tool per MavSDK method

In developing the MCP server and testing it on real world and simulated real world scenarios, we found that the LLM was not well suited to handle simply MavSDK methods presented as tools. For example, the LLM would immediately assume that the drone was at its new location after sending a goto command via the goto tool. In response to a prompt “Take off, fly to xyz location and land”, the LLM would send all commands simultaneously, resulting in the drone landing before it reached the target location. This could also lead to dangerous situations resulting in a crash. For example, if prompt was takeoff to 100 m and fly to xyz, the LLM may send the take off and fly to command in rapid succession, and the drone would not have time to ascend to the initial take off height, resulting in a low altitude flight towards the final destination, which may be below the level of obstacles, resulting in a crash. Thus, a simple “one tool per MavSDK” method is not advisable for an MCP drone command and control server.

Table 1 MAVSDK Python Method Implementation Summary

Category	Total	Implemented	Coverage
Action	22	10	45%
Telemetry	31	17	55%
Mission	10	6	60%
MissionRaw	7	2	29%
Param	7	5	71%
Camera	21	0	0%
Gimbal	8	0	0%
Offboard	10	0	0%
FollowMe	7	0	0%
Geofence	2	0	0%
ManualControl	3	0	0%
Info	5	0	0%
Calibration	6	0	0%
LogFiles	3	0	0%
FTP	9	0	0%
Tune	1	0	0%
Shell	2	0	0%
Transponder	1	0	0%
TOTAL	~155	40	26%

Therefore, we decided to manually add some additional tools, such as “wait for xxx” to be enabled. Several additional tools were defined, based on the author’s extensive experience with programming drones [6, 8]. In the future, it would be a good idea to automate this or even use AI to improve the tool set based on a database of mission profiles. For now, the tools were manually curated and tested. Table 2 shows a summary of how many custom and MavSDK commands were developed.

4.5 One off vs. continuous drone control

Modern LLMs are designed for prompt/response operation, which does not fit well with continuous command and control of drones for long missions. The architect of modern LLMs is “fire and forget”. However, for drone flight, there should in many cases be continuous monitoring, which LLMs are not good at. For example, the LLM may say it checks the progress every once in a while but it does not do it. The LLM also told the drone to take off and then immediately told it to fly to xyz location, before the takeoff maneuver was complete, causing the (virtual) drone to crash into an obstacle. Also, the monitor location method we used during development did not provide the user with realtime feedback of the drone location or status. In fact one option was to just wait for the set_location to return complete, which could block the LLM for the length of the mission, which might be as long as 30 minutes or longer. Therefore, we had to implement some logic of real time monitoring of the drone into the MCP server itself. This made the MCP server a kind of ground control station

with its own internal memory and logic, which is not the goal. Ideally, all the memory, logic, real time, long term monitoring would be done by the LLM, and the MCP server would just be an interpreter between the drone and the LLM. We do not know what the correct long term solution to this is. LLM technology with real time, long term situational awareness and memory needs to be developed for this approach to scale and reach its full potential.

The specific timing and coordination is specific to the LLM model used, and the chatbot and/or agentic wrapper. For closed source providers, these details are usually not exposed to the public. For example, ChatGPT’s agentic mode works like this: Run ALL tool calls in sequence. Only show output to the user AFTER the turn is complete. It does not pause between tool calls to show intermediate results. We attempted to work around this with the DISPLAY_TO_USER tool in order to give the user incremental progress reports on the status of the flight, but ChatGPT waited until the end to show them in some cases

It is an important topic for future research to develop agentic LLM systems for continuous drone command and control. Once developed, the interface here can serve as a continuous, real time, standardized, and easy to use bridge between the virtual LLM world and the real world.

Table 2 Summary of Exposed MCP Tools

Category	MAVSDK Eq.	Custom	Total
Flight Control	4	1	5
Safety	3	0	3
Navigation	4	5	9
Mission Management	6	4	10
Telemetry	14	0	14
Parameter Management	3	0	3
Other	0	1	1
TOTAL	34	11	45

5 Demonstration and testing

Demonstration and testing was performed on a real drone and a virtual drone. The real drone provided real world testing, while the virtual drone provided more extensive testing in situations not suitable, safe, or allowed with a real drone. In the main text, we use OpenAI. Claude Desktop was also demonstrated (not shown). In the appendix, we show and discuss demonstrated with open-source, locally run LLMs using LMStudio.

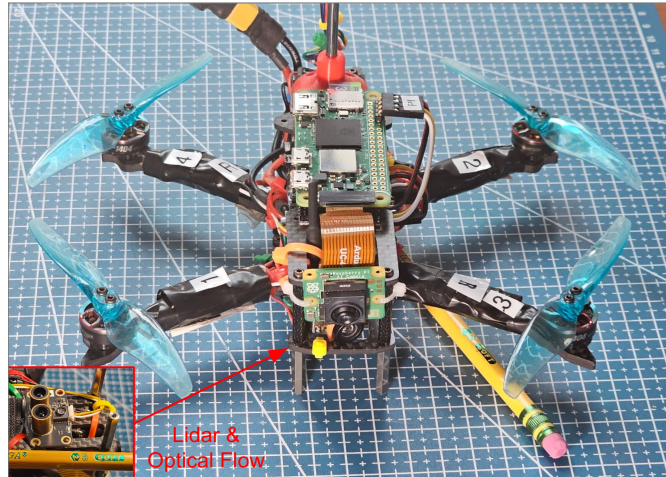


Fig. 5 Picture of drone used in this work. A LIDAR and optical flow sensor is used for GPS-denied flights, for example in the drone cage.

5.1 Real drone

5.1.1 Drone design

The drone design was a sub-250g 4 inch quad running Ardupilot, as described in ref. [9]. Some modifications were needed for stable flight in our small (10 x 10 x 10 foot) drone cage. A lidar and optical flow sensor was used for position stabilization with 1 cm accuracy. This enabled indefinite stable hover, despite the weak or no GPS signal. As GPS hold stability is not enough, even in the presence of a good lock in our small cage, the GPS functionality was completely disabled in the software. A picture of the drone and the LIDAR is shown in Fig. 5.

5.1.2 Demo in cage: Initial flight test

The drone was equipped with a Raspberry Pi Zero W, connected to the drone via UART, and providing WiFi connection to the internet. Mavlink Router[10] passed all Mavlink traffic from the internet to the flight controller. A connection to the drone with a laptop running Mission Planner ground control station (using TCP connectivity) was initially used to confirm the drone could take off, loiter, and land autonomously in the cage safely before the LLM control was tested.

5.1.3 Demo in cage: LLM controlled flight test

Once stable hover and flight was demonstrated under internet control, the LLM was then connected to the drone via the MCP server. The MCP server has the IP address of the drone as an environmental parameter, and runs as a service on a cloud Linux instance. During the flight, the drone was also monitored by a simultaneous connection to Mission Planner, still connected from the first test above.

5.1.4 Details of MCP server deployment

In order to handle firewalls, we used a Tailscale VPN to allow transparent, secure communication from the MCP server in the cloud to the drone. The MCP server runs continuously without the need for user monitoring. However, for debugging purposes, MCP activities are programmed to be logged to the terminal for status monitoring. Since the MCP server runs as a systemd service/daemon, it is always on at boot. The demands on the Linux instance are nominal, so the cheapest Linux cloud instances provide plenty of processing and I/O power. These are readily available from multiple service providers for a low cost.

5.1.5 Demo in cage: Actual flight tests

Fig. 6 shows the actual flight test. The drone was on the ground, without the propellers spinning. The LLM was asked to flip a coin, and take off if it came up heads. Then, the LLM was asked a question about movies, and if it was true, to land the drone. Both maneuvers were executed flawlessly by the LLM, as shown in Fig. 6. This demonstrates for the first time LLM natural language control of a real drone in response to unpredictable or pre-trained world knowledge using a universal MCP interface.

5.2 Virtual drone

5.2.1 SITL instance

A SITL Software in the loop [11, 12] instance of a virtual drone was deployed on a cloud hosted Ubuntu 22.04 instance. The virtual drone thus had an IP address. Again, we used Tailscale VPN for easy secure connection between firewalled systems. A ground control station (QGroundControl) was used to continuously monitor the status and position of the virtual drone on a map.

5.2.2 LLM virtual drone control

Extensive (virtual) test flights were performed to confirm the full functionality of the software MCP server during development. The project was able to control the drone take off, land, fly to, arm, disarm, and other basic functionalities. This demonstrated the ability of an LLM to fly the drone virtually anywhere in the world.

5.2.3 LLM virtual drone control: Hicups

One of the disadvantages to this approach is the LLM was only willing to do a certain number of tool calls while monitoring the drone flight. Also, the LLM would not always follow the prompt instruction to loop between tool calls to check the drone status. For this reason, at this juncture, the MCP server is not able to follow the drone on long missions (longer than about 5-10 minutes). This is a limitation of the LLM model used, and not of this work.

5.2.4 Integration with other MCPs servers: Google maps

During testing, we would ask the drone to fly to the nearest grocery store, but the LLM model did not have up to date information about the world map. By a series of fortunate events, during the writing of this paper, Google decided to open up Google Maps to an MCP server on Dec. 10, 2025[13]. Therefore, we used this as an opportunity to demonstrate multiple MCP servers in a single agent for drone control. Shown in Fig.7 is a demo where google maps was used to provide up to date real time information about local store, and fly the drone there. This is a major milestone. Thus, we have demonstrated MCP LLM drone control, and other MCP LLM integration for real time, global navigation information for drone command and control.

6 Discussion

6.1 Number of tools vs context size

A typical maximum number of tokens for a commercial LLM is around 100k-1M tokens. Local LLMs with less compute power have smaller limits. Anthropic has noted that MCP servers can consume a large number of tokens[14], and has suggested strategies to address this. In this work, we used about 5k tokens for the tool definitions (45 tools). For this application, this is acceptable. However, this needs to be taken into account for future agentic systems with many different MCP servers.

6.2 Limits and extensions of this approach

This approach is the first demonstration of LLM control of a drone through a scalable, industry standard interface platform MCP. It abstracts away from the LLM the need to know the details of how the drone operates or communicates. As such, it represents the first step towards physical AI, where the LLM has knowledge of the physical world and, in this case, control of it.

However, it only has a small amount of information in this realization. The amount of information about the drone in this work is only the gps location, orientation, velocity. In principle additional data such as temperature, humidity, wind, etc, could be beamed down from the drone's sensors.

However, the work does not provide enough information for the LLM to enable the LLM to provide a more sophisticated 3d model of the physical world. A possible extension of this project would be to enable this capability. One such strategy could be to equip swarms of drones with Lidar, Radar, sonar, or other 3d mapping abilities, and to give dynamic information into the LLM directly. This would be a step towards cyber-physical integration at scale: Similar to how google earth is a computer map of the world, one could envision an extension of this work to an LLM model of the entire physical world, with dynamic as well as static, high resolution 3d dimensional representations of objects and their interactions. Such a technology would be transformative and bring AI from the virtual to the physical world for applications we cannot yet even imagine.

6.3 Safety

We did our test flights in a drone cage. Obviously, there are safety issues. For one, we firmly believe that there should always be a human in the loop for possible manual override.

In addition, there should be some reliable way to ensure the LLM does not break out of any firewall rules. For example, the MCP should not allow the AI to override the override, e.g. lock the human out of the loop problematically. If the MCP server is properly configured, the LLM will not be able to do this.

There is another important question: LLM behavior is not deterministic and therefore not predictable exactly. How should the safety of AI drone control be tested? What standards should be used? What should the test protocols be? These are questions beyond the scope of this paper.

7 Enabled Use Cases

The following are concepts enabled by this work.

7.1 Integrated autonomous missions

Real time, real world data could be included in separate MCP servers. The LLMs could be trained to incorporate this for specific missions, which will be discussed next.

7.1.1 Firefighting

Firefighting requires response to dynamic and rapidly changing situations. Here, LLM models integrated with drones could be used for rapid assessment and planning to strategically deploy precious resources for maximum impact.

7.1.2 SAR

The search and rescue of people, ground vehicles, sea vehicles, and downed aircraft could be enabled using agentic AI for planning, search grid optimization, and image processing.

7.1.3 BVLOS

Beyond visual line of sight could benefit from LLMs for enhanced planning and prediction for collision avoidance and air traffic control.

7.2 Scaling to multiple drones

Finally, scaling from one to multiple drones could enable agent-AI coordination of swarms of drones for coordinated, complex missions. Swarms of agents could control swarms of drones.

8 Related Work and Prior Art: Summary

The integration of Large Language Models (LLMs) into Unmanned Aerial Vehicle (UAV) systems has emerged as a transformative frontier for autonomous mission control, situational awareness, and human-drone interaction. To date, research in this domain [15–50] has predominantly focused on two primary paradigms: (1) task-specific architectures and (2) custom code generation and parsing pipelines.

8.1 Current State-of-the-Art: custom and Task-Specific Implementations

A significant portion of prior art relies on specialized architectures tailored to specific mission profiles. For instance, multi-modal LLM-enabled UAV swarms have been explored for forest firefighting and search-and-rescue (SAR) operations [16, 43], utilizing custom retrieval-augmented generation (RAG) and Bayesian belief models. Other works, such as “Aeroverse” [28] and “DeepSeek-R1” implementations [42], focus on fine-tuning aerospace-specific models or hardware-software co-designs for onboard task planning. While effective within their defined scopes, these systems are inherently non-universal and require significant engineering effort to adapt to new platforms or LLMs.

Another prevalent approach involves using the LLM as a parser or code generator to bridge the gap between natural language and drone control APIs [26, 35, 50]. Systems like “FLUC” [35] and “Typefly” [41] prompt LLMs to synthesize Python scripts or custom domain-specific languages (e.g., “MiniSpec”) which are subsequently interpreted by a custom runtime. Similarly, projects like “Chat with UAV” [30] and “SwarmChat” [33] translate natural language into proprietary JSON schemas or machine language vectors (MLVs) for execution via ROS or ROS2 middleware. These “hand-coded” tool definitions and custom parsing pipelines represent a significant bottleneck, as they are often brittle, LLM-dependent, and labor-intensive to maintain.

8.2 Proposed MCP-Based Architecture

In contrast to the fragmented and platform-locked nature of existing implementations, this work introduces the first universal, LLM-agnostic, and drone-agnostic command and control interface leveraging the **Model Context Protocol (MCP)**. The novelty and significance of this approach compared to the prior art [15–50] are detailed below (10.1):

1. **Universal Standardization vs. custom Integration:** While prior works (Refs [16, 26, 33, 35]) rely on custom, one-off bridges between specific LLMs and drone APIs, our architecture adopts MCP, an open, industry-wide standard. This moves the field from tedious, labor-intensive manual integration to a standardized, plug-and-play paradigm.
2. **LLM and Drone Agnosticism:** Previous research often ties the implementation to a specific model (e.g., GPT-4 in [50]) or a specialized local model [35]. Our MCP server, “DroneServer,” provides a common interface that any modern LLM supporting the MCP standard (Claude, ChatGPT, Gemini, Llama) can natively

discover and utilize. Furthermore, by abstracting Mavlink through MavSDK, the interface is compatible with millions of drones running Ardupilot or PX4, a level of cross-platform compatibility not achieved in the surveyed literature.

3. **Native Tool Discovery vs. Prompt Engineering:** Unlike “LLM-as-parser” approaches [26, 46] or code-synthesis frameworks [35, 41, 50] which require extensive prompt engineering to “teach” the LLM a custom API, our MCP-based approach enables **native tool discovery**. The LLM automatically understands the available drone capabilities (e.g., `arm_drone`, `takeoff`, `go_to_location`), their schemas, and their constraints directly from the MCP server’s metadata. This significantly reduces latency, syntax errors, and the need for complex verification wrappers.
4. **Seamless Multi-Service Integration:** A key limitation of prior art is the siloed nature of the drone control system. In this work, we demonstrate the unique ability to integrate multiple MCP servers—such as Google Maps for real-time navigation and Mavlink for drone control—into a single agentic workflow. This allows the LLM to autonomously decide to fetch external data (e.g., “find the nearest grocery store”) and translate that information into precise flight coordinates in a single, cohesive mission, a capability that represents a significant leap over the task-specific frameworks found in [15–50].

By shifting from “fire and forget” custom implementations to a standardized, real-time monitored, and universally accessible protocol, this work provides the necessary infrastructure to scale LLM-drone integration across the broader industry and research community.

9 Comprehensive Prior Art Analysis: The Evolution of LLM-Integrated UAV Systems

This section provides a deep dive, detailed thematic analysis of the 36 foundational works [15–50] that define the current state of the art, followed by a critical contrast ((10.7)) with the universal, standardized architecture proposed in this work.

9.1 Foundational Surveys and Global Frameworks

The initial research landscape is characterized by comprehensive surveys that map the convergence of Artificial Intelligence (AI) and aerial platforms. Kanigür et al. [15] explored the role of LLMs in 6G-enabled Non-Terrestrial Networks (NTNs), positioning UAVs as intelligent nodes capable of autonomous decision-making in space-air-ground integrated networks. They specifically highlighted the shift from 0.1 ms latency targets in 6G to the need for real-time mission planning where LLMs handle user intent interpretation and multi-agent coordination. This is complemented by the work of Tian et al. [19], who provided a systematic overview of “Agentic Low-Altitude Mobility.” Their “UAVs Meet LLMs” roadmap identifies unique challenges such as variable altitudes and three-dimensional mobility, proposing an “Agentic UAV” framework that leverages foundation models for autonomous perception and memory.

The technical evolution from traditional deep learning to Transformer-based architectures is reviewed by Kheddar et al.[20]. They provided a unified taxonomy of

Transformer-based UAV models, evaluating CNN-Transformer hybrids and reinforcement learning Transformers across applications like precision agriculture and military supervision. Similarly, Yang et al.[18] and Javaid et al.[27] discussed the broader implications of AI-driven safety and security. Yang et al. noted that while traditional AI excels at pattern recognition, LLMs are required for human-level reasoning to address flight disruptions and unauthorized flights. Javaid et al. evaluated the suitability of architectures like GPT, BERT, and T5 for UAV spectral sensing, identifying opportunities for LLMs to optimize spectrum sharing and data processing in emergency restoration. More recently, Cidjeu et al.[24] and Navarro et al.[44] have summarized the GenAI adoption period (2021–2024). Navarro et al. introduced a budget-friendly quadcopter platform that unites 5G, Edge AI, and Virtual Reality (VR), using LLMs to extract actionable insights from 360-degree panoramic streams with low latency.

9.2 Swarm Intelligence and Multi-Agent Collaboration

A significant subset of the prior art focuses on the complexity of controlling multiple drones simultaneously. Ping et al.[16] investigated LLM-enabled UAV swarms for forest fire fighting. Their framework uses retrieval-augmented generation (RAG) and human-machine interaction to perform fire assessment and swarm task planning, demonstrating an experiment video where drones adapt to dynamic fire fronts. Han et al.[32] introduced “SwarmChain,” a collaborative inference system for UAV swarm control. By designing a tensor parallelism-based reasoning framework (CoLLM) and an adaptive load scheduling algorithm (Also), they enabled Raspberry Pi-based swarms to achieve a 1.9–2.3x performance improvement over standard MPI implementations, reducing inference latency by up to 52%.

The interaction layer for swarms has also seen innovation. Eumi et al.[33] presented “SwarmChat,” a context-aware multimodal system that integrates four LLM-based modules: Context Generator, Intent Recognition, Task Planner, and Modality Selector. This system allows users to manage large groups of autonomous robots with minimal input, achieving high user satisfaction in preliminary evaluations. In the realm of creative applications, Schuck et al.[34] developed “SwarmGPT.” This language-based choreographer uses the reasoning capabilities of LLMs to design synchronized, rhythmic drone performances (up to 200 drones in simulation). By augmenting the LLM with a safety filter, they ensured that non-experts could iteratively refine choreographies using natural language without risking collisions.

9.3 Safety, Security, and Mission Reliability

As autonomy increases, so do the risks associated with unconstrained AI decision-making. Chagas et al.[17] provided a comparative review of AI approaches for UAV deconfliction, identifying deep learning, reinforcement learning, and bio-inspired learning as the three primary categories. They emphasized the critical need for automated flight plan approval and real-time surveillance to manage congested airspace. To mitigate the “hallucination” risks of LLMs, Cleland-Huang et al.[43] proposed “Cognitive Guardrails.” This framework, CAIRN (Context-Aware Inference for Reasoning

and plaNning), combines Bayesian inference with natural language analysis to constrain LLM decisions, ensuring they remain safe and aligned with mission goals in search-and-rescue (SAR) contexts.

Furthermore, Majumdar et al.[47] introduced a hybrid architecture (RL+LP) for autonomous fire response, specifically designed for Mars simulation rovers and disaster environments. Their “ORION-FR” agent uses dialog-driven planning to bridge natural language interaction with mathematical optimization, providing mission resilience in extraterrestrial exploration. These works collectively highlight a move toward “verifiable autonomy,” where the flexibility of language models is constrained by rigorous safety filters and logical solvers.

9.4 Embodied Intelligence and “Agentic” UAVs

The concept of the “Agentic UAV”—a system that perceives, reasons, and acts within its environment—is a dominant theme. Sapkota et al.[23] and Koubaa et al.[37] defined the architectural layers of these systems, focusing on “tool-calling” and cognitive reasoning. Koubaa’s “Agentic UAVs” framework uses a five-layer architecture (Perception, Reasoning, Action, Integration, Learning) and achieved higher person detection rates (91% vs 75%) and a 92% action recommendation rate in SAR simulations. Yao et al.[28] introduced “AeroVerse,” construction of the AerialAgent-Ego15k and CyberAgent-Ego500k datasets. Their “SkyAgentX” model integrates perception-reasoning-navigating-planning, outperforming mainstream models by 8.5% across core tasks.

Task-specific agentic implementations include “LogisticsVLN” by Zhang et al.[25], which focuses on terminal delivery. This system uses lightweight LLMs for window-level localization and target attribute extraction, filling a gap in long-range VLN benchmarks. Mishra et al.[36] presented “AERMANI-VLM,” which integrates open-vocabulary perception (via CLIPSeg) and structured reasoning for aerial manipulation. By decoupling symbolic reasoning from physical action, they achieved robust pick-and-place performance in multi-step tasks. Other works, such as those by Ahmmad et al.[39] and Zhao & Lin [42], have demonstrated cloud-controlled and edge-optimized agents. Zhao & Lin’s prototype achieves 5–6 tokens/sec inference for 14B-parameter models on-board, enabling sugarcane monitoring and power grid inspection in communication-constrained environments. Ahmmad et al. achieved sub-1 second end-to-end latency for indoor GPS-denied navigation by offloading LLM inference to the cloud. Finally, Lim et al.[38] provided an open-source framework using Ollama and ROS2, benchmarking models like Gemma3 and Qwen2.5 for valid flight command generation (100% validity).

9.5 Natural Language Control and “LLM-as-Parser” Approaches

A prevalent paradigm is the use of the LLM as a “bridge” between natural language and robot code. Chen et al.[21] conducted an empirical study involving 997 papers and 1,509 GitHub projects, revealing that while 40% of developers attempt LLM integration, many still face challenges with technical maturity and safety. Yuan et al.[26]

proposed the “NeLV” system, establishing a five-level automation taxonomy that charts the evolution from LLM-as-Parser to LLM-as-Autopilot. They demonstrated missions spanning multi-UAV patrol and multi-hop relocation.

Modular frameworks like “FLUC” [35] and the voice-driven system by Choutri et al.[40] translate bilingual (English/Arabic) commands into executable mission code. Choutri’s system achieved 95% speech recognition accuracy and 300–500 ms execution latency for real-time control. To address the verbosity of Python, Chen et al. [41, 49] introduced “TypeFly” and the “MiniSpec” language. MiniSpec reduced response time by up to 62% (and up to 74% in the updated 2025 version) compared to Python-based planning, using “Stream Interpreting” to allow the drone to act while the LLM is still generating the next token. Other “hand-coded” bridges include “Chat with UAV” [30] and “From Words to Flight” [50], which integrate OpenAI models with PX4/Gazebo stacks.

9.6 Autonomous Navigation, Testing, and Optimization

Finally, several papers address navigation optimization and testing. Wu et al.[22] introduced an LLM-driven Pareto-optimal RL framework that reduced average flight time by 16% and collision rates by 50% in urban wind environments by dynamically switching between aggressive and cautious policies. Khan et al.[31] focused on context-aware navigation using DeepSeek-v3, achieving a 94% success rate in simple environments through a weighted prioritization system for obstacle avoidance and mission completion.

Automated testing is addressed by Duvvuru et al.[29] with “AutoSimTest,” where multiple agents (Env-Agent, S-Agent, M-Agent, Analytics-Agent) collaborate to generate unique test scenarios and analyze flight logs. For specialized services, Wassim et al.[46] proposed “LLM-DaaS” (Drone-as-a-Service), which maps free-text requests to structured delivery tasks using fine-tuned LLaMA-3.2 and Phi-3.5 models. Traffic and scheduling optimization are explored by Moraga et al.[45] for smart city congestion reduction and Zhou et al.[48] through “LLM-QL,” an LLM-enhanced Q-Learning approach that achieves a 1.35x improvement in completion time and UAV utilization for the Multiple Flying Sidekicks Traveling Salesman Problem (mFSTSP).

9.7 Contrast and Significance of the Proposed MCP Architecture

While the 36 papers analyzed above [15–50] represent a monumental effort, they share several fundamental limitations that our work addresses:

1. **Custom vs. Universal Integration:** Most prior art [26, 35, 50] relies on “custom” parsing pipelines or custom Python scripts. These are platform-locked and require significant engineering to adapt to new drone APIs or LLMs. In contrast, our **Model Context Protocol (MCP)** approach provides a standardized, industry-wide interface that is naturally “drone-agnostic” and “LLM-agnostic.”
2. **Prompt Engineering vs. Native Tool Discovery:** Systems like “TypeFly” [41] or “FLUC” [35] depend on extensive prompt engineering to “teach” the LLM a custom API. Our architecture leverages MCP’s **Native Tool Discovery**, allowing

the LLM to automatically understand available drone capabilities (e.g., `arm_drone`, `takeoff`, `go_to_location`) and their schemas directly from the server’s metadata, significantly reducing syntax errors and latency.

3. **Siloed vs. Multi-Service Workflow:** Prior works typically treat the drone as a siloed system. Our MCP-based “DroneServer” allows for the seamless integration of multiple services. For example, our system can combine a “Google Maps MCP” server for real-time location data with a “Mavlink MCP” server for flight control, enabling a single agentic workflow that was previously fragmented across different papers.
4. **Scaling and Standardization:** While works like “AeroVerse” [28] and “Agentic UAVs” [37] move toward cognitive reasoning, they lack a unified communication protocol. By adopting MCP, we move the field from tedious, labor-intensive manual integration to a plug-and-play paradigm that scales across millions of Ardupilot/PX4-enabled drones, providing a robust path toward the “Agentic Low-Altitude Mobility” vision envisioned in [19].

In conclusion, the surveyed literature [15–50] provides the building blocks for LLM-drone interaction, but the transition from fragmented, custom implementations to a universal, standardized, and natively interoperable architecture is the defining contribution of the current work.

10 Standardizing Physical Intelligence: A Comparative Analysis of the Universal Large Language Model Drone Interface

Here we provide a deep technical discussion of significance and novelty of this work, as well as a detailed, in depth discussion of the performance evaluation procedure.

The evolution of autonomous systems has historically been defined by a progressive decoupling of high-level objective formulation from low-level locomotive execution. Traditional robotics relied heavily on rigid, rule-based architectures where every potential environmental state required a pre-defined response. The emergence of “Physical AI”—a field where real-world information is integrated with sensing, command, and control—marked the first significant shift toward adaptive systems. However, the integration of Large Language Models (LLMs) into this domain has, until recently, been hampered by a lack of standardization, resulting in a fragmented landscape of custom, non-interoperable solutions. This work establishes a foundational paradigm shift by introducing the first universal, LLM-agnostic, and drone-agnostic interface predicated on the Model Context Protocol (MCP)[4]. To appreciate the significance of this work, one must examine the specific technological bottlenecks it overcomes and the nuanced ways in which it diverges from the prior art that characterized the early era of LLM-robotic integration.

Table 3 Comparison of Traditional Approach vs. Universal MCP Approach (this work)

Integration Component	Traditional Approach (Prior Art)	Universal MCP Approach (this work)
Model Compatibility	Model-specific (e.g., GPT-only)[51]	LLM-agnostic (Claude, GPT, Llama)
Hardware Link	Custom API wrappers	Ubiquitous Mavlink Protocol
Tool Definition	Hand-coded JSON schemas per model[51]	Standardized MCP tool definitions[51]
Deployment	Local script execution[35]	Cloud-based Linux “DroneServer”
Contextual Sourcing	Single-source telemetry[42]	Multi-server (e.g., Drone + Google I)

10.1 The Architectural Crisis of Pre-Standardized Drone Control

Prior to this work, the primary challenge in the field was the “NxM” integration problem. Every new AI model (N) and every unique drone hardware platform (M) required a custom-coded bridge.[51] This resulted in redundant development efforts and excessive maintenance burdens, as any update to an LLM’s API or a drone’s firmware could potentially break the entire control stack.[51] Early attempts to bridge this gap primarily utilized direct function calling or specialized wrappers that were inherently tied to specific ecosystems, such as OpenAI’s GPT Actions.[51]

The significance of this work lies in its move away from these “siloed” integrations toward an open-standard architecture. By leveraging the Model Context Protocol—an open standard introduced by Anthropic in 2024 and donated to the Linux Foundation in 2025—we provide a “universal remote” for AI systems. This protocol standardizes how AI applications interact with external tools and data, allowing an LLM to discover and execute drone commands with the same ease as querying a database or reading a file.[51]

10.2 Quantitative and Qualitative Divergence from Prior Art

The research identifies and addresses specific limitations in high-profile prior art, most notably in the works of Zhao and Lin (2025)[42] and Nunes and Amorim (2025)[35]. Understanding these distinctions is critical to situating this work within the broader trajectory of autonomous systems research.

10.2.1 The Contrast with Edge-Optimized Computing

The work by Zhao and Lin, “General-Purpose Aerial Intelligent Agents Empowered by Large Language Models,” emphasized a hardware-software co-design approach.[42]

Their methodology focused on achieving onboard LLM operation via an edge-optimized computing platform.[42] While this addressed the problem of communication latency and connectivity in GPS-denied or signal-jammed environments, it introduced substantial physical constraints.

Zhao and Lin’s system achieved an inference rate of only 5–6 tokens per second for a 14B-parameter model, while consuming 220W of peak power.[42] For a medium-sized UAV, such a power draw significantly curtails flight time and mission endurance. In contrast, this work’s cloud-based Linux architecture offloads the computational heavy lifting to data centers, allowing for the use of far more powerful “frontier” models (such as Claude Sonnet or GPT-5 series) that possess deeper reasoning capabilities than any edge-deployed model. This work argues that with the advent of high-speed satellite and 5G/6G connectivity, the “connected drone” paradigm is often more versatile than the “onboard compute” paradigm, as it allows the drone to remain lightweight and energy-efficient while accessing virtually unlimited compute power.

10.2.2 Beyond Script Generation: The FLUC Comparison

Another major milestone in the prior art was the FLUC framework (A Framework Leveraging Large Language Models for Autonomous UAV Control in Flying Networks).[35] FLUC acted as a translator, converting natural language commands into executable Python code that would then be run by the drone’s autopilot.[35]

The primary limitation of FLUC, and similar mission-code generators, is the “one-off” nature of their execution. Once the code is generated and deployed, the LLM’s active role in the flight often ends, unless a new cycle of code generation is triggered.[35] This work moves beyond this “code-and-deploy” model to an “agentic tool-use” model. Their implementation includes 45 distinct tools that allow the LLM to interact with the drone in real-time. This enables a conversational, closed-loop control system where the LLM can query telemetry (e.g., `get_battery_level`, `get_position`), reason about the state, and issue immediate corrective commands (e.g., `set_return`, `land`) without needing to regenerate a full mission script.

10.3 Technical Sophistication of the MCP-Mavlink Bridge

This work is not merely a conceptual proposal but a massive technical implementation, involving 15,000 lines of code. The current iteration’s 45 tools provide a granular level of control that mirrors industrial ground control stations (GCS).

10.3.1 The Role of MavSDK and JSON-RPC

A critical technical nuance in this work’s methodology is the use of MavSDK to wrap the low-level Mavlink protocol. Mavlink is the ubiquitous language of the open-source drone fleet, used by millions of vehicles running Ardupilot and PX4. However, raw Mavlink messages are often too low-level for efficient LLM processing. By using MavSDK, we provide a higher-level set of commands (`takeoff`, `goto_location`, `orbit`) that are then exposed as MCP tools.

The communication between the LLM and the MCP server uses JSON-RPC 2.0 over HTTP and Server-Sent Events (SSE).[51] This is a significant departure from prior

art that often used serial communication or proprietary websocket links.[35] The use of JSON-RPC 2.0 provides a standardized, error-checked way to execute commands and receive telemetry, making the system robust against the “hallucinations” or malformed requests that can occur with LLMs.[51]

10.3.2 Multi-Server Contextual Integration

One of the most profound innovations in this work is the demonstration of multi-server synergy. Because the system is built on the MCP standard, the LLM is not restricted to a single data source. In further testing, we integrated “DroneServer” with a Google Maps MCP server.

This allows the LLM to solve problems that were previously impossible for autonomous drones. For example, a user could command the drone to “Fly to the nearest hospital,” and the LLM would:

1. Query the Google Maps MCP server to find the coordinates of the nearest hospital.
2. Assess the drone’s current battery level and range via the DroneServer.
3. Calculate the optimal path and issue the necessary `goto_location` commands.

Prior art, such as the Tazir et al. (2023)[50] study, typically required all “knowledge” to be either contained within the LLM’s training data (which is static and often out of date) or provided via a manually curated prompt.[35] This work enables a dynamic, real-time knowledge retrieval process that turns the drone into an agent capable of navigating complex, real-world geography based on up-to-the-minute data.

10.4 Addressing the Continuity Problem in Physical Control

A significant portion of this work (specifically Section IV-E) is dedicated to the distinction between “one-off” prompts and continuous drone control. This is a critical observation that was largely absent from the prior art, which often treated drone missions as a series of discrete, successful steps in a simulation.

10.4.1 The “Takeoff and Crash” Failure Mode

We documented a specific failure mode where an LLM (in this case, functioning in an “agentic” fire-and-forget mode) issued a takeoff command and then immediately followed it with a movement command before the takeoff maneuver was complete. In the virtual SITL (Software in the Loop) environment, this caused the drone to crash into an obstacle.

This insight is significant because it highlights the temporal mismatch between LLM reasoning (which is nearly instantaneous for a single turn) and physical movement (which is bound by the laws of physics and aerodynamics). To solve this, we had to implement internal logic within the MCP server itself to handle real-time monitoring and state verification. This effectively turns the MCP server into a “Ground Control Station” with its own internal memory and logic—a hybrid architecture that ensures safety while allowing the LLM to remain the primary decision-maker.

10.4.2 User Interface and Incremental Feedback

The paper also critiques existing LLM wrappers, such as ChatGPT’s agentic mode, which often waits until all tool calls are complete before showing any output to the user. This is unsuitable for drone flight, where a mission might last 30 minutes or more. We attempted to work around this with a custom `DISPLAY_TO_USER` tool to provide incremental progress reports. This focuses on a neglected aspect of the prior art: the transparency and “explainability” of the AI pilot to the human supervisor in real-time.

10.5 Development Methodology: The “AI-Generated” Paradigm

A meta-innovation of this work is its development methodology. We emphasize that the 15,000 lines of code were authored using the Cursor IDE, leveraging AI models to build the very system that controls the robot. This is related our recent 2025 paper [6], which suggests a new paradigm for robotics engineering where AI co-designs, develops, and verifies the control systems for future robots.[6]

This differs from the prior art where human engineers spent months or years hand-coding mission planners like QGroundControl or Mission Planner.[6] By using AI-assisted coding, this work demonstrates that complex, functionally complete command-and-control stacks can be delivered at “orders-of-magnitude faster development cycles”. [6] This suggests that the “significance” of the paper is not just the resulting software, but the proof that the barrier to entry for advanced aerospace engineering is being lowered by the same AI technology it seeks to harness.[6]

10.6 Comparative Experimental Validation

The experimental rigor of this work further distinguishes it from the more limited “simulation-only” studies that preceded it.

10.6.1 Real-World Deployment and GPS-Denied Flight

We conducted testing in a physical drone cage using a UAV equipped with LiDAR and optical flow sensors for GPS-denied environments. They demonstrated that the LLM could make autonomous decisions—such as taking off based on a virtual “coin flip”—and successfully land the drone based on its internal knowledge base.

While previous papers like Tazir et al. (2023)[50] or Nunes and Amorim (2025)[35] validated language-based interaction in simulations, they often lacked the leap to real-world, unpredictable flight conditions.[35] This work’s demonstration of a “real world internet connected drone” with “real-time dynamic decision making” provides the empirical proof of concept required to move LLMs from the laboratory to industrial application.

10.6.2 Simulation and Scalability

In addition to real-world flight, we used a virtual drone (SITL instance on Ubuntu 22.04) to test extensive flight planning. This dual-track validation (Real + Virtual) is significant because it proves that the MCP-based interface is robust across different

Table 4 Comparison of Traditional AI Drones vs. MCP-Enabled Cognitive Agents

Mission Parameter	Traditional AI Drones	MCP-Enabled Cognitive Agents
Object Detection	Label-based (e.g., “Person”)	Semantic (e.g., “Potential Casualty”)
Navigation Rules	Static Geo-fencing	Dynamic, Contextual Regulation Inj
Memory	None or simple logs	Episodic retrieval via MCP servers
Tool Execution	Sequential Scripts	Dynamic, mission-phase-aware tool c

“physical” substrates. Whether the drone is a physical unit in a cage or a digital twin in a simulator, the LLM interacts with it using the exact same set of tools, confirming the “universal” claim of this research.

10.7 Impact on Reconnaissance and Specialized Missions

The significance of the MCP interface is particularly pronounced in specialized domains like reconnaissance. As noted in related research on Model Context Protocols in reconnaissance drones, the MCP acts as a “standardized cognitive interface”. It shifts the drone from simple object recognition to semantic reasoning about sensory data.

Prior systems might detect a “red truck” via a YOLO-based perception layer. An MCP-enabled system, however, can integrate that detection with episodic mission history (stored in a memory server) or geographical regulations (retrieved from a policy server) to reason about the significance of the truck. If the drone enters a restricted area, the MCP server can dynamically inject location-specific instructions or modified operational constraints into the LLM’s context window, ensuring immediate compliance without manual pilot input.

10.8 Strategic Preferences and Model Divergence

A deeper insight offered by the broader context of this research is the variability in how different LLMs utilize the universal interface. Related studies comparing LLMs in conflict or strategic scenarios have found that models like GPT-3.5 tend to be more “aggressive” in their chosen actions compared to GPT-4.[\[52\]](#)

Because the interface in this work is “LLM agnostic,” it allows researchers for the first time to benchmark different “AI pilots” on the exact same hardware and mission set. This is a fundamental advance over prior art that was typically model-specific. It enables the systematic study of “pilot personality”—how different models handle risk, efficiency, and safety when given the “keys” to a physical aircraft.

10.9 Technical Summary of the Universal Interface

The novelty of this work can be distilled into its departure from the “hand-coded” era toward a standardized “protocol-driven” era. It successfully integrates the virtual world of data centers with the physical world of robotics through three key pillars:

1. **The Universal Layer:** Using MCP to eliminate the need for custom integrations across the NxM model-tool landscape.[51]
2. **The Ubiquitous Language:** Using Mavlink and MavSDK to ensure the interface works with millions of existing drones.
3. **The Continuous Brain:** Implementing internal server logic to bridge the gap between “fire-and-forget” reasoning and real-time physical movement.

Unlike the “one-off” code generators (FLUC) or the “onboard-only” agents (Zhao & Lin), this work creates a persistent, extensible ecosystem where a drone is just one of many tools available to an LLM, alongside maps, databases, and communication services. This approach is not merely a new way to fly a drone; it is a new way to integrate physical machines into the fabric of the modern AI industry.

11 Conclusions

We have demonstrated a comprehensive, versatile, easy to use drone command and control interface, connecting LLMs with drones. The method applies to a large class of drones and LLMs, and can be cheaply and easily implemented for a broad variety of use cases. The work establishes a high-water mark for the field of Physical AI. By solving the interface challenge, it allows the research community to shift its focus from “how to connect the drone” to “how to best use the drone” for complex mission objectives.

This work paves the way for the development of agentic LLM systems for continuous drone command and control. While the interface exists, the long-term situational awareness and memory required for multi-hour autonomous missions remain areas for future research. However, by providing a continuous, real time, standardized, and easy to use bridge, this work provides the essential infrastructure upon which the next generation of truly autonomous aerial agents will be built. The transition from narrow, task-specific robots to general-purpose aerial intelligent agents is now a matter of software scaling rather than fundamental architectural invention. In this new paradigm, the “drone” is finally becoming what the industry has long envisioned: a flexible, cognitive agent capable of reasoning through the complexities of the physical world in partnership with human intent.

Acknowledgments. This work was done as a demonstration for the EECS 195 Drone course during Fall 2025[9]. We thank lab manager Shawn Davis for help maintaining the lab and drone cage necessary for this work, and the UC Irvine EECS department for allowing us to develop and teach this hands on drone class with AI and autonomy as a feature. Generative AI was used to assist writing the prior art literature review sections.

Declarations

- Funding Not applicable
- Conflict of interest/Competing interests (check journal-specific guidelines for which heading to use) The authors declare no competing interests.
- Ethics approval and consent to participate Not applicable
- Consent for publication Not applicable
- Data availability Not applicable
- Materials availability Not applicable
- Code availability Code is available at <https://github.com/PeterJBurke/droneserver>.
- Author contribution PJB conceived, developed, and demonstrated the code and flights. NS assisted with the flight demonstrations. All authors participated in the analysis and writing.

Appendix A Open-source local LLM demo

In Fig. [A1](#), we use LM Studio running on a Macbook Air M4 with 16 GBytes of RAM. (Thus this can be done on a local machine with cost around one thousand dollars.) The model used is the open-source model qwen2.5-7b-instruct. Note that the nominal context window size of 4k tokens was too small to fit the MCP server. Therefore, a context window of size 32k tokens was used.

Appendix B Table of complete list of tools exposed to the LLM

A comparison of all MavSDK methods with MCP tools is given in Table B1.

Table B1: MAVSDK Python Methods and MCP Implementation Status

Method	Impl	MCP Tool	Description
ACTION Plugin - Basic Flight Commands (10/22 implemented)			
arm()	Yes	arm_drone	Arm motors for flight
disarm()	Yes	disarm_drone	Disarm motors
takeoff()	Yes	takeoff	Autonomous takeoff
land()	Yes	land	Land at current position
return_to_launch()	Yes	return_to_launch	Return to home/launch position
goto_location()	Yes	go_to_location	Fly to GPS coordinates
hold()	Yes	hold_position	Hold/hover at current position
kill()	Yes	kill_motors	Emergency motor cutoff
set_takeoff_altitude()	Yes	takeoff	Set takeoff target altitude
set_maximum_speed()	Yes	set_max_speed	Set max flight speed
get_takeoff_altitude()	No	-	Get current takeoff altitude setting
get_maximum_speed()	No	-	Get current max speed setting
set_return_to_launch_altitude()	No	-	Set RTL altitude
get_return_to_launch_altitude()	No	-	Get RTL altitude setting
transition_to_fixedwing()	No	-	VTOL: switch to fixed-wing mode
transition_to_multicopter()	No	-	VTOL: switch to multicopter mode
do_orbit()	No	-	Orbit around a point
reboot()	No	-	Reboot autopilot
shutdown()	No	-	Shutdown autopilot
terminate()	No	-	Flight termination
set_actuator()	No	-	Direct actuator/servo control
set_current_speed()	No	-	Set current target speed
TELEMETRY Plugin - Sensor Data & Status (17/31 implemented)			
position()	Yes	get_position	GPS position (lat, lon, alt)
home()	Yes	get_home_position	Home/launch position
attitude_euler()	Yes	get_attitude	Roll, pitch, yaw angles
velocity_ned()	Yes	get_speed	Velocity (North, East, Down)
battery()	Yes	get_battery	Battery voltage & percentage
gps_info()	Yes	get_gps_info	Satellite count, fix type
flight_mode()	Yes	get_flight_mode	Current flight mode

continued on next page

Table B1 – *continued from previous page*

Method	Impl	MCP Tool	Description
health()	Yes	get_health	System health checks
in_air()	Yes	get_in_air	Is drone flying?
armed()	Yes	get_armed	Are motors armed?
status_text()	Yes	print_status_text	Status messages stream
health_all_ok()	Yes	get_health_all_ok	All health checks passed?
landed_state()	Yes	get_landed_state	On ground / taking off / in air / landing
rc_status()	Yes	get_rc_status	RC controller status & signal
heading()	Yes	get_heading	Compass heading (degrees)
imu()	Yes	get_imu	Raw IMU data (accel, gyro)
odometry()	Yes	get_odometry	Position + velocity + orientation
attitude_quaternion()	No	–	Attitude as quaternion
attitude_angular_velocity_body()	No	–	Angular velocity (body frame)
ground_speed_ned()	No	–	Ground speed (NED frame)
fixedwing_metrics()	No	–	Airspeed, climb rate (fixed-wing)
scaled_imu()	No	–	Scaled IMU readings
raw_imu()	No	–	Unprocessed IMU data
distance_sensor()	No	–	Rangefinder/lidar distance
scaled_pressure()	No	–	Barometer pressure
actuator_control_target()	No	–	Commanded actuator values
actuator_output_status()	No	–	Actual actuator outputs
vtol_state()	No	–	VTOL mode (MC/FW/-transition)
unix_epoch_time()	No	–	System time
position_velocity_ned()	No	–	Combined position & velocity
ground_truth()	No	–	Simulation ground truth
MISSION Plugin - Waypoint Missions High-Level (6/10 implemented)			
start_mission()	Yes	initiate_mission	Start uploaded mission
mission_progress()	Yes	print_mission_progress	Current waypoint progress
clear_mission()	Yes	clear_mission	Clear all waypoints
set_current_mission_item()	Yes	set_current_waypoint	Jump to specific waypoint
is_mission_finished()	Yes	is_mission_finished	Check if mission complete
set_return_to_launch_after_mission()	Yes	initiate_mission	RTL after mission ends
upload_mission()	No	–	Upload mission plan (high-level API)
download_mission()	No	–	Download mission from drone (high-level)
pause_mission()	No	–	Pause current mission
get_return_to_launch_after_mission()	No	–	Get RTL-after-mission setting
MISSION_RAW Plugin - Waypoint Missions Low-Level (2/7 implemented)			

continued on next page

Table B1 – *continued from previous page*

Method	Impl	MCP Tool	Description
upload_mission()	Yes	upload_mission	Upload raw mission items
download_mission()	Yes	download_mission	Download raw mission items
start_mission()	No	–	Start mission (raw API)
pause_mission()	No	–	Pause mission (raw API)
clear_mission()	No	–	Clear mission (raw API)
set_current_mission_item()	No	–	Set current waypoint (raw)
import_qgroundcontrol_mission()	No	–	Import QGC mission file
PARAM Plugin - Parameter Management (5/7 implemented)			
get_param_int()	Yes	get_parameter	Get integer parameter
get_param_float()	Yes	get_parameter	Get float parameter
set_param_int()	Yes	set_parameter	Set integer parameter
set_param_float()	Yes	set_parameter	Set float parameter
get_all_params()	Yes	list_parameters	List all parameters
get_param_custom()	No	–	Get custom parameter type
set_param_custom()	No	–	Set custom parameter type
CAMERA Plugin - Photo & Video Control (0/21 implemented)			
take_photo()	No	–	Capture single photo
start_photo_interval()	No	–	Start time-lapse capture
stop_photo_interval()	No	–	Stop time-lapse
start_video()	No	–	Start video recording
stop_video()	No	–	Stop video recording
start_video_streaming()	No	–	Start video stream
stop_video_streaming()	No	–	Stop video stream
set_mode()	No	–	Set photo/video mode
set_setting()	No	–	Adjust camera setting
get_setting()	No	–	Get camera setting
set_zoom_level()	No	–	Set zoom level
zoom_in()	No	–	Increase zoom
zoom_out()	No	–	Decrease zoom
format_storage()	No	–	Format SD card
select_camera()	No	–	Select camera by index
information()	No	–	Get camera info
status()	No	–	Get camera status
capture_info()	No	–	Last capture info
current_settings()	No	–	Current camera settings
possible_setting_options()	No	–	Available setting options
list_photos()	No	–	List captured photos
GIMBAL Plugin - Camera Gimbal Control (0/8 implemented)			
set_pitch_and_yaw()	No	–	Set gimbal angles
set_pitch_rate_and_yaw_rate()	No	–	Set gimbal angular rates
set_mode()	No	–	Set yaw follow/lock mode
set_roi_location()	No	–	Point at GPS location
take_control()	No	–	Take gimbal control
release_control()	No	–	Release gimbal
control()	No	–	Gimbal control stream
attitude()	No	–	Get gimbal attitude

continued on next page

Table B1 – continued from previous page

Method	ImplMCP Tool	Description
OFFBOARD Plugin - Direct Control Mode (0/10 implemented)		
start()	No –	Enter offboard mode
stop()	No –	Exit offboard mode
is_active()	No –	Check if offboard active
set_position_ned()	No –	Set position (NED frame)
set_position_global()	No –	Set position (GPS)
set_velocity_ned()	No –	Set velocity (NED frame)
set_velocity_body()	No –	Set velocity (body frame)
set_attitude()	No –	Set attitude angles
set_attitude_rate()	No –	Set attitude rates
set_actuator_control()	No –	Direct actuator control
FOLLOW_ME Plugin - Target Following (0/7 implemented)		
start()	No –	Start follow mode
stop()	No –	Stop follow mode
is_active()	No –	Check if following
set_config()	No –	Set follow behavior
get_config()	No –	Get follow config
set_target_location()	No –	Update target GPS
get_last_location()	No –	Get last target location
GEOFENCE Plugin - Flight Boundaries (0/2 implemented)		
upload_geofence()	No –	Upload geofence polygons
clear_geofence()	No –	Clear all geofences
MANUAL_CONTROL Plugin - Joystick Control (0/3 implemented)		
start_position_control()	No –	Start position control
start_altitude_control()	No –	Start altitude control
set_manual_control_input()	No –	Send joystick inputs
INFO Plugin - System Information (0/5 implemented)		
get_version()	No –	Get firmware version
get_product()	No –	Get product/vendor info
get_flight_information()	No –	Flight time, distance
get_identification()	No –	System identification
get_speed_factor()	No –	Simulation speed factor
CALIBRATION Plugin - Sensor Calibration (0/6 implemented)		
calibrate_gyro()	No –	Calibrate gyroscope
calibrate_accelerometer()	No –	Calibrate accelerometer
calibrate_magnetometer()	No –	Calibrate compass
calibrate_level_horizon()	No –	Level horizon calibration
calibrate_gimbal_accelerometer()	No –	Calibrate gimbal accel
cancel()	No –	Cancel calibration
LOG_FILES Plugin - Flight Logs (0/3 implemented)		
get_entries()	No –	List available logs
download_log_file()	No –	Download specific log
erase_all_log_files()	No –	Delete all logs
FTP Plugin - File Transfer (0/9 implemented)		
reset()	No –	Reset FTP server
download()	No –	Download file from drone

continued on next page

Table B1 – *continued from previous page*

Method	ImplMCP Tool	Description
upload()	No –	Upload file to drone
list_directory()	No –	List directory contents
create_directory()	No –	Create directory
remove_directory()	No –	Remove directory
remove_file()	No –	Delete file
rename()	No –	Rename file/directory
are_files_identical()	No –	Compare files (CRC)
TUNE Plugin - Audio Feedback (0/1 implemented)		
play_tune()	No –	Play buzzer tune
SHELL Plugin - MAVLink Shell (0/2 implemented)		
send()	No –	Send shell command
subscribe_receive()	No –	Receive shell output
TRANSPONDER Plugin - ADS-B (0/1 implemented)		
transponder()	No –	Get nearby aircraft data
CORE Plugin - Connection Management (1/2 implemented)		
connection_state()	Yes (internal)	Connection status
set_mavlink_timeout()	No –	Set connection timeout

The complete set of tools exposed to LLM and their description are given in Table B2.

Table B2: Complete MCP Tools Reference (45 Tools)

MCP Tool	MAVSDK Method	Description
Flight Control (5 tools)		
arm_drone	action.arm()	Arm motors for flight
disarm_drone	action.disarm()	Disarm motors
takeoff	action.set_takeoff_altitude() + takeoff()	Autonomous takeoff to specified altitude
land	action.land()	Land at current position
hold_position	Custom	Hold current position in GUIDED mode
Safety (3 tools)		
return_to_launch	action.return_to_launch()	Return to home/launch position
kill_motors	action.kill()	Emergency motor cutoff
get_battery	telemetry.battery()	Battery voltage & percentage
Navigation (9 tools)		
get_position	telemetry.position()	Current GPS position (lat, lon, alt)
get_home_position	telemetry.home()	Home/launch position
go_to_location	action.goto_location()	Fly to GPS coordinates
move_to_relative	Custom	Move relative distance (N/E/D meters)
set_max_speed	action.set_maximum_speed()	Set maximum flight speed
set_yaw	Custom	Rotate to specified heading
reposition	Custom	Move to location and loiter
check_arrival	Custom	Check if drone arrived at destination
monitor_flight	Custom	Monitor flight progress with auto-land
Mission Management (10 tools)		
initiate_mission	mission_raw.upload() mission.start()	+ Upload and start mission
upload_mission	mission_raw.upload_mission()	Upload mission waypoints
download_mission	mission_raw.download_mission()	Download mission from drone
print_mission_progress	mission.mission_progress()	Current waypoint progress
pause_mission	Deprecated	Pause mission (unsafe, do not use)
hold_mission_position	Custom	Safe pause alternative (GUIDED mode)
resume_mission	mission.start_mission()	Resume paused mission
clear_mission	mission.clear_mission()	Clear all waypoints
set_current_waypoint	mission.set_current_mission_item()	Jump to specific waypoint
is_mission_finished	mission.is_mission_finished()	Check if mission complete
Telemetry (14 tools)		
get_flight_mode	telemetry.flight_mode()	Current flight mode
get_health	telemetry.health()	Detailed system health checks

continued on next page

Table B2 – *continued from previous page*

MCP Tool	MAVSDK Method	Description
<code>get_health_all_ok</code>	<code>telemetry.health_all_ok()</code>	Quick health check (boolean)
<code>get_speed</code>	<code>telemetry.velocity_ned()</code>	Velocity (North, East, Down)
<code>get_attitude</code>	<code>telemetry.attitude_euler()</code>	Roll, pitch, yaw angles
<code>get_gps_info</code>	<code>telemetry.gps_info()</code>	Satellite count, fix type
<code>get_in_air</code>	<code>telemetry.in_air()</code>	Is drone flying? (boolean)
<code>get_armed</code>	<code>telemetry.armed()</code>	Are motors armed? (boolean)
<code>get_landed_state</code>	<code>telemetry.landed_state()</code>	Landed state (ground/air/landing)
<code>get_rc_status</code>	<code>telemetry.rc_status()</code>	RC controller status & signal
<code>get_heading</code>	<code>telemetry.heading()</code>	Compass heading (degrees)
<code>get_imu</code>	<code>telemetry.imu()</code>	IMU data (accel, gyro)
<code>get_odometry</code>	<code>telemetry.odometry()</code>	Combined position, velocity, orientation
<code>print_status_text</code>	<code>telemetry.status_text()</code>	Status messages from autopilot
Parameter Management (3 tools)		
<code>get_parameter</code>	<code>param.get_param_int/float()</code>	Get parameter value (auto-detects type)
<code>set_parameter</code>	<code>param.set_param_int/float()</code>	Set parameter value (auto-detects type)
<code>list_parameters</code>	<code>param.get_all_params()</code>	List all parameters
Other (1 tool)		
<code>set_flight_mode</code>	Custom	Set flight mode by name (HOLD, RTL, LAND, GUIDED)

References

- [1] Caballero-Martin, D., Lopez-Guede, J.M., Estevez, J., Graña, M.: Artificial intelligence applied to drone control: A state of the art. *Drones* **8**(7), 296 (2024)
- [2] Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V., Scaramuzza, D.: Champion-level drone racing using deep reinforcement learning. *Nature* **620**(7976), 982–987 (2023)
- [3] Meier, L.: MAVLink: Micro Air Vehicle Communication Protocol. <https://mavlink.io>. Accessed: August 1, 2025 (2013)
- [4] Anthropic: Model Context Protocol (MCP) Specification. Model Context Protocol. Accessed: Dec. 19, 2025. <https://modelcontextprotocol.io/specification/>
- [5] The Linux Foundation: Linux Foundation Announces the Formation of the Agentic AI Foundation (AAIF), Anchored by New Project Contributions Including Model Context Protocol (MCP), goose and AGENTS.md. Press Release. Accessed: Dec. 19, 2025 (2025). <https://www.linuxfoundation.org/press/linux-foundation-announces-the-formation-of-the-agentic-ai-foundation>
- [6] Burke, P.: Robot builds a robot’s brain: Ai generated drone command and control station hosted in the sky. arXiv preprint arXiv:2508.02962; in press *npj Artificial Intelligence* (2025)
- [7] Anthropic: Model Context Protocol Python SDK. GitHub. Accessed: Dec. 19, 2025 (2024)
- [8] Hu, L., Pathak, O., He, Z., Lee, H., Bedwany, M., Mica, J., Burke, P.J.: “cloud-station:” a cloud-based ground control station for drones. *IEEE Journal on Miniaturization for Air and Space Systems* **2**(1), 36–42 (2020)
- [9] Burke, P., Wei, P., Fakhri, M., Burke, D.: Drones from a to z: Experiential learning at its finest. <https://doi.org/10.36227/techrxiv.175695608.82891552/v1> (2025)
- [10] MAVLink Router Project: mavlink-router. GitHub. Accessed: Dec. 19, 2025 (2024)
- [11] ArduPilot Dev Team: SITL Simulator (Software in the Loop). <https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html>. Accessed: 2025-08-01 (2025)
- [12] Burke, P.: A simulated, virtual drone for testing and development. <https://github.com/PeterJBurke/CreateSITLenv>. Accessed: 2025-08-01 (2025)
- [13] Bachman, M., Berenberg, A.: Announcing Official MCP Support for Google Services. Google Cloud. <https://cloud.google.com/blog/products/ai-machine-learning/announcing-official-mcp-support-for-google-services>

Accessed 2025-12-19

- [14] Anthropic: Code Execution with MCP: Building More Efficient AI Agents. Anthropic Engineering. Accessed: Dec. 19, 2025. <https://www.anthropic.com/engineering/code-execution-with-mcp>
- [15] Ak Kanigur, N., Mert, M., Duru, I.: Leveraging large language models and artificial intelligence for uavs in 6g-enabled non-terrestrial networks. In: 2025 9th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), pp. 1–7. IEEE, ??? (2025). <https://doi.org/10.1109/ISMSIT67332.2025.11267927>
- [16] Ping, Y., Liang, T., Ding, H., Lei, G., Wu, J., Zou, X., Shi, K., Shao, R., Zhang, C., Zhang, W., et al.: Multimodal large language models-enabled uav swarm: Towards efficient and intelligent autonomous aerial systems. arXiv preprint arXiv:2506.12710 (2025)
- [17] Chagas, F.S., Ruseno, N., Bechina, A.A.A.: Artificial intelligence approaches for uav deconfliction: A comparative review and framework proposal. *Automation* **6**(4), 54 (2025)
- [18] Yang, Z., Zhang, Y., Zeng, J., Yang, Y., Jia, Y., Song, H., Lv, T., Sun, Q., An, J.: Ai-driven safety and security for uavs: From machine learning to large language models. *Drones* **9**(6), 392 (2025)
- [19] Tian, Y., Lin, F., Li, Y., Zhang, T., Zhang, Q., Fu, X., Huang, J., Dai, X., Wang, Y., Tian, C., et al.: Uavs meet llms: Overviews and perspectives towards agentic low-altitude mobility. *Information Fusion* **122**, 103158 (2025)
- [20] Kheddar, H., Habchi, Y., Ghanem, M.C., Hemis, M., Niyato, D.: Recent advances in transformer and large language models for uav applications. arXiv preprint arXiv:2508.11834 (2025)
- [21] Chen, Y., Que, X., Zhang, J., Chen, T., Li, G., Chen, J.: When large language models meet uavs: How far are we? arXiv preprint arXiv:2509.12795 (2025)
- [22] Wu, J., You, H., Sun, B., Du, J.: Llm-driven pareto-optimal multi-mode reinforcement learning for adaptive uav navigation in urban wind environments. *IEEE Access* (2025)
- [23] Sapkota, R., Roumeliotis, K.I., Karkee, M.: Uavs meet agentic ai: A multidomain survey of autonomous aerial intelligence and agentic uavs. arXiv preprint arXiv:2506.08045 (2025)
- [24] Cidjeu, D.D., Fendji, J.L.K.E., Kamla, V.C., Tchappi, I.: Uav leveraging genai/llms, a brief survey. *Procedia Computer Science* **265**, 382–389 (2025)

- [25] Zhang, X., Tian, Y., Lin, F., Liu, Y., Ma, J., Szatmáry, K.S., Wang, F.-Y.: Logisticsvl: Vision-language navigation for low-altitude terminal delivery based on agentic uavs. arXiv preprint arXiv:2505.03460 (2025)
- [26] Yuan, L., Deng, C., Han, D.-J., Hwang, I., Brunswicker, S., Brinton, C.G.: Next-generation llm for uav: From natural language to autonomous flight. arXiv preprint arXiv:2510.21739 (2025)
- [27] Javaid, S., Fahim, H., He, B., Saeed, N.: Large language models for uavs: Current state and pathways to the future. *IEEE Open Journal of Vehicular Technology* (2024)
- [28] Yao, F., Yue, Y., Liu, Y., Sun, X., Fu, K.: Aeroverse: Uav-agent benchmark suite for simulating, pre-training, finetuning, and evaluating aerospace embodied world models. arXiv preprint arXiv:2408.15511 (2024)
- [29] Duvvuru, V.S.A., Zhang, B., Vierhauser, M., Agrawal, A.: Llm-agents driven automated simulation testing and analysis of small uncrewed aerial systems. arXiv preprint arXiv:2501.11864 (2025)
- [30] Wang, H., Chen, Z., Li, G., Ma, B., Li, C.: Chat with uav–human-uav interaction based on large language models. arXiv preprint arXiv:2512.08145 (2025)
- [31] Khan, A.-M., Rehman, I.U., Saeed, N., Sobnath, D., Khan, F., Khattak, M.A.K.: Context-aware autonomous drone navigation using large language models (llms). In: *Proceedings of the AAAI Symposium Series*, vol. 6, pp. 102–107 (2025)
- [32] Han, B., Chen, Y., Li, J., Li, J., Su, J.: Swarmchain: Collaborative llm inference for uav swarm control. *IEEE Internet of Things Magazine* (2025)
- [33] Eumi, E.M., Abbass, H., Marcus, N.: Swarmchat: An llm-based, context-aware multimodal interaction system for robotic swarms. In: *International Conference on Swarm Intelligence*, pp. 181–192 (2025). Springer
- [34] Schuck, M., Dahanaggamaarachchi, D.O., Sprenger, B., Vyas, V., Zhou, S., Schoellig, A.P.: Swarmgpt: Combining large language models with safe motion planning for drone swarm choreography. *IEEE Robotics and Automation Letters* (2025)
- [35] Nunes, D., Amorim, R., Ribeiro, P., Coelho, A., Campos, R.: A framework leveraging large language models for autonomous uav control in flying networks. arXiv preprint arXiv:2506.04404 (2025)
- [36] Mishra, S., Yadav, R.D., Das, A., Gupta, S., Pan, W., Roy, S.: Aermani-vlm: Structured prompting and reasoning for aerial manipulation with vision language models. arXiv preprint arXiv:2511.01472 (2025)

- [37] Koubaa, A., Gabr, K.: Agentic uavs: Llm-driven autonomy with integrated tool-calling and cognitive reasoning. arXiv preprint arXiv:2509.13352 (2025)
- [38] Lim, S.K., Chong, M.J.Y., Khor, J.H., Ling, T.Y.: Taking flight with dialogue: Enabling natural language control for px4-based drone agent. arXiv preprint arXiv:2506.07509 (2025)
- [39] Ahmmad, S., Aditto, Z.A., Hossain, M.M., Yeasmin, N., Hossain, S.: Autonomous navigation of cloud-controlled quadcopters in confined spaces using multi-modal perception and llm-driven high semantic reasoning. arXiv preprint arXiv:2508.07885 (2025)
- [40] Choutri, K., Fadloun, S., Khettabi, A., Lagha, M., Meshoul, S., Fareh, R.: Leveraging large language models for real-time uav control. *Electronics* **14**(21), 4312 (2025)
- [41] Chen, G., Yu, X., Ling, N., Zhong, L.: Typefly: Flying drones with large language model. arXiv preprint arXiv:2312.14950 (2023)
- [42] Zhao, J., Lin, X.: General-purpose aerial intelligent agents empowered by large language models. arXiv preprint arXiv:2503.08302 (2025)
- [43] Cleland-Huang, J., Granadeno, P.A.A., Bernal, A.M.R., Hernandez, D., Murphy, M., Petterson, M., Scheirer, W.: Cognitive guardrails for open-world decision making in autonomous drone swarms. arXiv preprint arXiv:2505.23576 (2025)
- [44] Navarro, A., Quinto, C., Hernández, J.A.: Beyond visual line of sight: Uavs with edge ai, connected llms, and vr for autonomous aerial intelligence. arXiv preprint arXiv:2507.15049 (2025)
- [45] Moraga, Á., Curtò, J., Zarzà, I., Calafate, C.T.: Ai-driven uav and iot traffic optimization: Large language models for congestion and emission reduction in smart cities. *Drones* **9**(4), 248 (2025)
- [46] Wassim, L., Mohamed, K., Hamdi, A.: Llm-daas: Llm-driven drone-as-a-service operations from text user requests. In: *The International Conference of Advanced Computing and Informatics*, pp. 108–121 (2024). Springer
- [47] Majumdar, S., Kirkley, S.E., Mallik, B.B.: Llm-guided hybrid architecture for autonomous fire response: Dialog-driven planning in space and disaster missions. In: *2025 IEEE World AI IoT Congress (AIIoT)*, pp. 1049–1054 (2025). IEEE
- [48] Zhou, Q., Wu, J., Zhu, M., Zhou, Y., Xiao, F., Zhang, Y.: Llm-ql: a llm-enhanced q-learning approach for scheduling multiple parallel drones. *IEEE Transactions on Knowledge and Data Engineering* (2025)
- [49] Chen, G., Yu, X., Ling, N., Zhong, L.: Chatfly: Low-latency drone planning with

- large language models. *IEEE Transactions on Mobile Computing* (2025)
- [50] Tazir, M.L., Mancas, M., Dutoit, T.: From words to flight: Integrating openai chatgpt with px4/gazebo for natural language-based drone control. In: *International Workshop on Computer Science and Engineering* (2023)
- [51] Deepscope Learning Center: What Is the Model Context Protocol (MCP) and How It Works. <https://securityboulevard.com/2025/03/what-is-the-model-context-protocol-mcp-and-how-it-works/> Accessed 2026-04-08
- [52] Lamparth, M., Corso, A., Ganz, J., Mastro, O.S., Schneider, J., Trinkunas, H.: Human vs. machine: Behavioral differences between expert humans and language models in wargame simulations. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, vol. 7, pp. 807–817 (2024)

LLM control of a real drone



<p>PROMPT</p> <p>User</p> <p>flip a coin. if it is heads, arm the drone and takeoff to 1 m</p> <p>Assistant</p> <p>Let's flip a coin:</p> <p>● Flipping...</p> <p>It's heads!</p> <p>LLM RESPONSE TO CALL MCP</p>	<p>User</p> <p>flip a coin. if it is heads, arm the drone and takeoff to 1 m</p> <p>Assistant</p> <p>Let's flip a coin:</p> <p>● Flipping...</p> <p>It's heads!</p> <p>Now, I will proceed to arm the drone and takeoff to 1 meter.</p> <p>☰ mavlink</p> <ul style="list-style-type: none">arm_dronetakeoff <p>The coin flip was heads, so the drone has been armed and takeoff to an altitude of 1 meter has been initiated. If you need live updates on status, location, or want to proceed with more commands, just let me know!</p>	 <p>Drone taking off:</p> <p>A photograph showing a drone in a cage with a red arrow pointing upwards, indicating the direction of takeoff.</p>
<p>PROMPT</p> <p>User</p> <p>land if darth vader is lukes father</p> <p>Assistant</p> <p>Yes, DARTH VADER IS INDEED LUKE'S FATHER IN STAR WARS.</p> <p>LLM RESPONSE TO CALL MCP</p>	<p>User</p> <p>land if darth vader is lukes father</p> <p>Assistant</p> <p>Yes, DARTH VADER IS INDEED LUKE'S FATHER IN STAR WARS.</p> <p>Following your instruction, I will proceed to land the drone</p>	 <p>Drone landing:</p> <p>A photograph showing a drone in a cage with a red arrow pointing downwards, indicating the direction of landing.</p>

Fig. 6 LLM control of a real drone. (A) Demonstration of LLM controlled take off in a drone cage. The LLM decides based on a virtual coin flip if it should command the drone to take off. (B) Demonstration of LLM controlled landing. The LLM is asked a question based on its trained data, and uses the answer to that question to decide autonomously whether to command the drone to land.

LLM control of a simulated drone

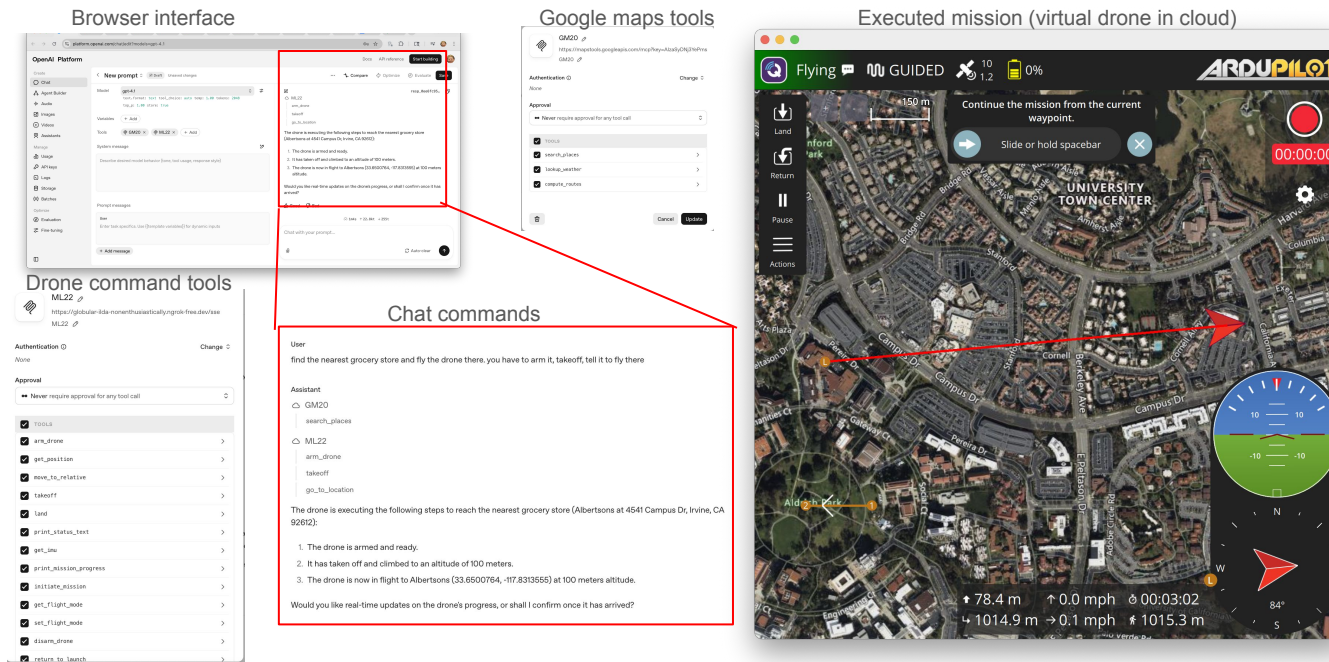


Fig. 7 LLM control of a virtual drone. (A) Browser interface, with inset showing some of the exposed tools for the DroneServer MCP, and Google maps MCP, as well as an example prompt for a drone mission. (B) Drone mission flown by the virtual drone, under control of the LLM, monitored in real time by a separate connection to the drone using QGroundControl.

Open-source, local LLM control of a simulated drone

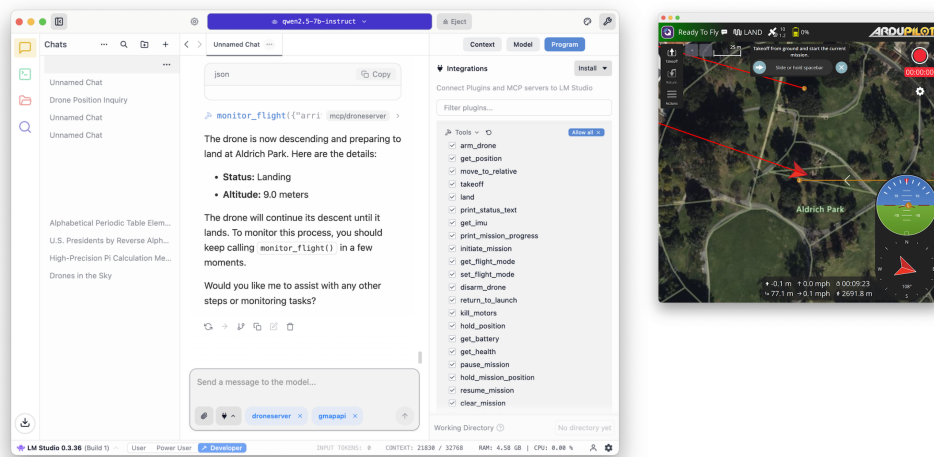


Fig. A1 LLM control of a virtual drone. (A) LM Studio interface, showing the chat window and the list of tools for the MCP. (B) Drone mission flown by the virtual drone, under control of the LLM, using LM Studio.