drones

MDPI

Article

TACOS: Task Agnostic Coordinator of a Multi-Drone System

Alessandro Nazzari *,†, Roberto Rubinacci † and Marco Lovera

Department of Aerospace Science and Technology, Politecnico di Milano, 20133 Milan, Italy; roberto.rubinacci@polimi.it (R.R.); marco.lovera@polimi.it (M.L.)

* Correspondence: alessandro.nazzari@polimi.it

† These authors contributed equally to this work.

Highlights

What are the main findings?

- An LLM-based framework that enables natural language control of multi-UAV systems.
- Separating reasoning and execution improves scalability and robustness in UAV swarms.

What are the implications of the main findings?

- TACOS allows a single operator to seamlessly transition from direct drone-level commands to high-level swarm behaviors, bridging semantic reasoning and low-latency UAV control through structured API execution.
- The proposed Coordinator–Supervisor architecture achieves higher success rates and better parallelization than monolithic or non-reasoning baselines. It supports continuous interaction and can react to dynamic events that affect the swarm or the environment.

Abstract

When a single pilot is responsible for managing a multi-drone system, the task may demand varying levels of autonomy, from direct control of individual UAVs to group-level coordination to fully autonomous swarm behaviors for accomplishing high-level tasks. Enabling such interaction requires a framework that supports multiple modes of shared autonomy. As language models continue to improve in reasoning and planning, they provide a natural foundation for such systems. In this paper, we present TACOS (Task-Agnostic COordinator of a multi-drone System), a unified framework that enables high-level natural language control of multi-UAV systems through a Large Language Model (LLM). TACOS integrates three key capabilities into a single architecture: a one-to-many natural language interface for intuitive user interaction, an intelligent coordinator for translating user intent into structured task plans, and an autonomous agent that executes plans and interacts with the real world. TACOS allows an LLM to interact with a library of executable APIs, bridging semantic reasoning with real-time multi-robot coordination. We demonstrate the system on a real-world multi-drone system and conduct an ablation study to assess the contribution of each module.

Keywords: multi-robot systems; large language models; unmanned aerial vehicles; swarm task planning; swarm motion planning

Academic Editors: Emanuele Luigi de Angelis, Fabrizio Giulietti and Giulio Avanzini

Received: 3 February 2026
Revised: 26 March 2026
Accepted: 26 March 2026
Published: 31 March 2026

1. Introduction

Coordinating multiple Unmanned Aerial Vehicles (UAVs) has become a core challenge in robotics, with applications ranging from surveillance and mapping to disaster response and delivery. Assigning a dedicated pilot to each UAV does not scale with swarm size: it is expensive, inefficient, difficult to coordinate, and prone to human error. As swarm

size increases, effective systems must support varying levels of shared autonomy, ranging from direct control of individual UAVs to fully autonomous swarm behaviors guided by high-level mission objectives.

Previous approaches have explored a range of interaction modalities. Some systems use fusion modules that combine voice and gesture recognition to interpret commands [1], while others rely on tablet-based interfaces for manual control [2]. These approaches improve usability but typically focus on structured command inputs rather than high-level semantic task specification. A central challenge is therefore the design of interfaces that allow a single operator to specify tasks at a high level. Recent advances in large language models (LLMs) suggest a promising direction [3,4]: natural language can serve not only as an intuitive interface but also as a mechanism for high-level reasoning and task decomposition.

In this work, we present TACOS (Task-Agnostic COordinator of a multi-drone System), an LLM-powered framework for multi-drone coordination. TACOS bridges high-level natural language instructions and low-level swarm control by adopting a language-to-tools orchestration approach. A language model interprets user intent and generates structured task plans, while safety, trajectory generation, and control are handled by established planning and control modules.

TACOS supports three core capabilities:

- **One-to-many natural language interface**: users can issue UAV-specific or swarm-level commands in natural language.
- **Intelligent coordination**: high-level instructions are translated into structured task plans.
- **Task management with closed-loop execution**: the system monitors swarm state and ensures correct temporal sequencing and execution.

We evaluate TACOS through simulation and real-world experiments with quadrotor platforms, including ablation studies of its architectural components.

The remainder of the paper is organized as follows: Section 2 provides an overview of the literature. Section 3 introduces the problem setup. Section 4 presents the proposed framework, TACOS. Section 5 presents the ablation study performed on the components of the framework. Sections 6 and 7 report both simulation results and real-world experiments using quadrotor platforms. Section 8 presents the conclusions and discusses future work.

2. Related Work

Multi-UAV systems have a long history of algorithmic solutions for coordination. A tcentral problem is *distributed task assignment*, for which market-based and consensus-based methods provide scalable task allocation [5,6].

Another line of work studies *formal task specification and planning* using temporal logic. Linear Temporal Logic (LTL) and Signal Temporal Logic (STL) enable users to specify temporally extended missions and have been applied to multi-robot motion planning and coordination [7,8]. These approaches offer strong correctness guarantees with respect to formal specifications.

Motion coordination has also been extensively studied through formation control methods [9] and decentralized trajectory optimization [10–12].

Recent advances in LLMs have enabled new approaches for robot control, task planning, and multi-agent coordination. Prior work in this area can be broadly categorized into LLM-based robot control systems, LLM-driven multi-robot coordination frameworks, and language interfaces for UAV swarm systems.

2.1. LLM-Based Robot Control

Several works explore using LLMs to translate natural language instructions into executable robot behaviors. Ref. [13] proposes generating executable programs from natural language commands by prompting an LLM to synthesize robot control code that integrates perception and control APIs. This approach demonstrates strong generalization across tasks by allowing the LLM to compose reusable functions and control primitives. However, the framework is primarily designed for single-robot systems and focuses on policy generation rather than multi-agent coordination. Ref. [14] focuses on LLM-driven planning and control module for a single robot. Alternatively, Ref. [15] demonstrates how a swarm of robots can employ an LLM in an online fashion to automatically generate executable code and perform previously unseen behaviors. TPML [16] leverages LLMs to generate executable code in both synchronous and asynchronous patterns.

2.2. LLM-Based Multi-Robot Coordination

Another line of work investigates using LLMs for high-level multi-agent reasoning and task allocation. Roco [17] introduces a dialectic collaboration framework in which each robot is represented by an LLM agent that communicates with others to coordinate tasks. Through iterative dialogue, agents generate subtask plans and waypoint goals, which are then validated using environment feedback such as collision checks or inverse kinematics constraints. The validated plans are executed using a centralized motion planner.

LLaMAR [18] addresses long-horizon planning in partially observable multi-agent environments. The framework uses a structured plan–act–correct–verify loop, allowing the LLM to iteratively refine task plans based on environment feedback. This approach improves robustness to uncertainty and planning errors but is primarily evaluated in simulation environments.

Other work examines architectural trade-offs for LLM-based multi-agent planning. For example, Ref. [19] compares different communication structures for LLM-driven robot teams, including centralized, decentralized, and hybrid coordination frameworks. The study highlights scalability challenges associated with multi-agent communication and demonstrates that hybrid approaches can improve planning performance while reducing token usage. LLM2SWARM [20] explores how LLMs can be used to directly synthesize and validate the robot controllers in a swarm scenario. CoELA [21] demonstrates how LLMs can be used in multi-agent cooperation settings by using these technologies to efficiently communicate between agents.

2.3. LLM Interfaces for UAV Swarm Systems

Recent work has also applied LLMs to aerial robotics and swarm systems. FlockGPT [22] proposes guiding UAV formations using natural language instructions by translating linguistic descriptions into geometric shape representations that are executed through flocking control algorithms. This approach enables intuitive control of drone formations but primarily focuses on formation generation rather than general mission coordination.

LEVIOSA [23] presents a system for generating UAV trajectories from natural language instructions using hierarchical prompting and multi-critic evaluation. The framework improves the reliability of language-generated trajectories but focuses mainly on trajectory generation rather than broader swarm-level task management.

Similarly, SwarmGPT [24] enables users to design drone swarm choreographies through natural language commands. The framework includes a safety filtering mechanism that adjusts LLM-generated trajectories to satisfy safety constraints. While effective for choreography applications, the approach is specialized for structured swarm performances rather than general-purpose drone coordination. LAN2CB [25] instead generates exe-

cutable code from natural language instructions by employing a library of expert behaviors and a predefined structured mission description template. The latter provides the LLM instructions on how to define trigger and finish conditions for each mission.

2.4. Contribution

Table 1 summarizes the key characteristics of existing LLM-based multi-robot and UAV swarm systems. Although existing work demonstrates the promise of LLMs for robot control and multi-agent coordination, several limitations remain. Many approaches focus either on static code generation or simulation-based planning, while others target specific applications such as manipulation or drone formation control. In addition, many frameworks rely on decentralized LLM agents or single-pass planning pipelines, which can introduce scalability challenges or limit runtime adaptability.

Table 1. Comparison of TACOS against relevant LLM multi-robot and UAV swarm systems.

System	Topology	Output	Replanning	Safety Layer	Real-World	Eval Scope
TACOS (Ours)	Cent.	API calls	✓	ATOMICA [10]	3 UAVs	Scaling, success
RoCo [17]	Decent.	Plan descr.	✓	RTT motion planner	Mobile manipulator	Collab. manip.
TPML [16]	Cent.	Code gen.	✓	Human cert.	2 UAVs	Mission gen.
SwarmGPT [24]	Cent.	Waypoints/Primitives	Offline	Safety filter	20 UAVs	Choreography
Code as Policies [13]	Cent.	Python code	✗	None	Robot arms	Gen. robot policy code
FlockGPT [22]	Cent.	Python code	✗	None	8 UAVs	UAV flocking

TACOS addresses these limitations by introducing a hierarchical LLM-based architecture for real-time multi-drone coordination. The framework separates high-level reasoning and execution monitoring through two distinct modules: a Coordinator, responsible for interpreting user commands and generating structured task plans, and a Supervisor, responsible for executing these plans and monitoring swarm state in a closed loop. This design enables TACOS to support replanning and continuous human–swarm interaction.

3. Problem Setup

We address the problem of centralized, high-level task planning for a multi-UAV system via an intelligent coordinator that interprets high-level user commands expressed in natural language. The coordinator must translate these commands into executable plans for the UAV swarm. The swarm operates in a bounded 3D environment $\mathcal{W} \subseteq \mathbb{R}^3$. The environment contains a set of n ellipsoidal obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_n\}$, where each obstacle is defined by its center position and a positive definite matrix specifying its orientation and size. The free space is given by $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \bigcup_{i=1}^{n} \mathcal{O}_i$. The environment also includes a set of m task-relevant entities, such as targets or landmarks, denoted by $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$, each representing a 3D coordinate. The state of the world is thus defined as $\mathcal{S}_{\mathcal{W}} = (\mathcal{O}, \mathcal{T})$.

We consider N quadrotor UAVs, where each UAV has a discrete operational mode, $\mathcal{S}_d \in \{\textit{unavailable}, \textit{idle}, \textit{flying}, \textit{tracking}\}$. The swarm state $\mathcal{S}_s$ is defined as $\mathcal{S}_s = \{(\mathcal{S}_d^i, p^i, v^i)\}_{i=1}^{N}$, where $p^i \in \mathbb{R}^3$ and $v^i \in \mathbb{R}^3$ are the current position and velocity vectors of the i-th drone. Additionally, each UAV is equipped with a predefined set of low-level control primitives, formalized as the action set: $\mathcal{A} = \{$`arm_takeoff()`, `start_tracking(`$\mathcal{T}_k$`)`, `stop_tracking()`, `land()`, `goto(x, y)`$\}$, which abstract away platform-specific dynamics.

Each action in $\mathcal{A}$ is implemented by a dedicated control algorithm that executes the corresponding behavior, thereby reducing the computational and control workload of the LLM-based framework. In particular,

- `goto(x, y)` (In the remainder of the paper, a fixed altitude z is used in all experiments; therefore, it is omitted from the notation): Once this command is issued, each drone coordinates using a decentralized trajectory optimization module, ATOMICA [10],

which ensures real-time, collision-free motion even when multiple drones share the same goal position.

- `start_tracking(`$\mathcal{T}_k$`)` : Each drone implements a position-based target tracking algorithm working on top of the `goto(x, y)` interface.

The LLM is therefore not required to run at control frequencies of hundreds of hertz. Instead, it delegates actions to reliable low-level control algorithms. This design supports heterogeneous UAV platforms, provided they implement the required action interface. Furthermore, the framework's capabilities can be easily extended by incorporating additional action primitives, leveraging advances from the wider control and robotics community.

Additionally, we make the following simplifying assumptions:

- Accurate state estimation: In simulation, the swarm state is assumed to be noise-free. In real-world experiments, pose measurements are provided by an external motion capture system.
- Full world-state knowledge: The state of the environment is assumed to be known and available at each time step.
- No explicit perception modeling: Perception is not explicitly simulated. Target detection and task completion are determined either through positional conditions (e.g., reaching a designated location) or through operator acknowledgment.

4. TACOS

The TACOS framework consists of two main language models arranged in a hierarchical architecture: the Coordinator LLM, which receives high-level natural language commands from the user and synthesizes a task plan, and the Supervisor LLM, which sequences and executes the plan based on real-time swarm and environment state. This hierarchy is inspired by the layered guidance and control architectures commonly used in aerospace systems and by Kahneman's two-system theory [26]. The Coordinator corresponds to a slow, reasoning layer responsible for planning and intent understanding, while the Supervisor functions as a fast execution layer. This separation simplifies context management across the LLMs and enables independent customization or fine-tuning of each model. Each LLM is initialized with a dedicated configuration prompt that defines its specific objectives, behavioral constraints, and output structure. Figure 1 illustrates the modular architecture of TACOS and the interaction flow between user input, the Coordinator, and the Supervisor.

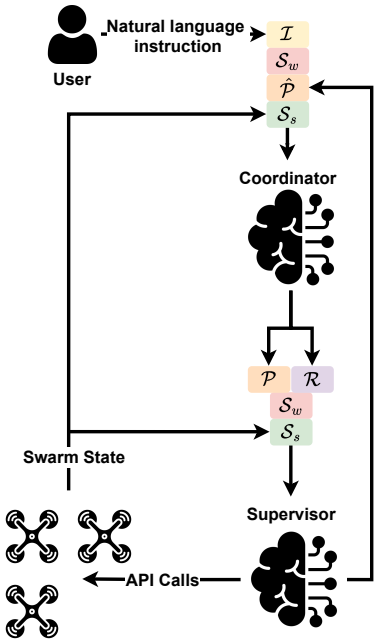


Figure 1. The TACOS framework.

4.1. Coordinator

The Coordinator LLM is responsible for translating high-level user instructions, expressed in natural language, into a structured task plan compatible with the swarm's control interface. It receives as input the current system state, comprising the swarm state $\mathcal{S}_s$, the world state $\mathcal{S}_\mathcal{W}$, the user's instruction $\mathcal{I}$, and the partial task plan $\hat{\mathcal{P}}$ that encodes unexecuted subtasks from prior plans. The swarm's action set $\mathcal{A}$ is encoded into the model's configuration prompt, as shown in Figure 2.

The Coordinator's output is structured into the following two components:

- *Reasoning* $\mathcal{R}$: a natural language explanation of the generated task plan. This explanation supports interpretability. Moreover, prompting the model to explicitly reason improves output quality by leveraging Chain of Thought (COT) mechanisms [27].
- *Task plan* $\mathcal{P}$: a list of atomic API calls required to fulfill the user request. All temporal dependencies, synchronization constraints, and inter-agent coordination are deferred to the Supervisor module.

```
You are a coordinator of a multidrone system . You are
responsible for translating high level user instructions ,
expressed in natural language , into a structured task
plan compatible with the available API :

{
    0: goto ( uav_id , x , y )
    # Sends the specified UAV to (x , y )
    1: arm_takeoff_drone ( uav_id )
    # Arms and takes off the specified UAV
    2: land ( uav_id )
    # Lands and disarms the specified UAV
    3: start_tracking ( uav_id , target_name , position )
    # Commands the specified UAV to start tracking
      the specified target, position can be either ' front '
      or ' back ' depending on the position with respect
      to the target
    4: stop_track_target ( uav_id )
    # Commands the specified UAV to stop tracking
       whatever target it is tracking .
}

# Guidelines

1. If the status of a drone changes during the mission
replan accordingly. Remember that drones in the
unavailable state cannot be commanded.
2. Avoid unsafe behavior. Do not send multiple drones
to the same position or into a known obstacle, always
keep a safe distance.
3. If the user asks for a shape or formation, calculate
approximate coordinates before issuing commands.
4. When asked to "surround" an object or point (x, y),
place drones evenly spaced in a circle around it, at
equal radius. Default radius = 1.0 unless specified.
5. A drone must takeoff before being able to receive
setpoints
```

(**a**)

```
# Output

Your output must be composed by two sections.

1. A reasoning block. a natural language explanation of
the generate task plan that accomplishes the mission
requested by the user.

2. The task plan. A python list of ALL the API calls
needed to complete the mission as requested by the
user regardless of temporal constraints.
and must have the following structure:

{
'reasoning': the reasoning block,
'task_plan': [(function_id, arg...), ... (function_id,
arg...)]
}

# Examples

Example 1:
User: [('drone0', ...), ('drone1', ...), ('drone2', ...),
('drone3',
...)] command: Of the previous task plan the following
actions have not been executed [], the user requested
the following: split the swarm into two groups, send
one around −1,−1 and when they arrive send the other
around 5,5.
System: {'reasoning':'All four drones are currently
flying and available for new commands. The user wants
to split the swarm into two groups of two drones each.
The first group should surround the point (−1,−1) and
the second group should surround (5,5). For each
group we issue two send_position calls that place the
drones at a radius of 1.0 around the target point.
Group-1 positions: (0,−1 ) and (−2,−1 ). Group-2
positions: (6,5) and (4,5).', 'task_plan':[(0, 'drone0',0,
−1),(0, 'drone1',−2,1), (0,'drone2',6,5),(0,'drone3',4,5)]}
```

(**b**)

Figure 2. Excerpts from the Coordinator's Modelfile. (**a**) Available APIs. (**b**) Chain-of-Thought and In-Context Learning examples.

To improve the Coordinator's planning capabilities, we adopt two prompt engineering strategies: In-Context Learning (ICL) and COT. ICL allows the model to infer the task structure from a small number of annotated demonstrations provided in the prompt [28]. In our case, we provide few-shot examples consisting of user requests and their corresponding ideal reasoning and task plans, illustrating the correct behavior expected from the model. An example of ICL in the Coordinator's Modelfile is shown in Figure 2. The complete set of examples is shown in Listing A2. Expanding the library of available APIs would

necessitate a proportionally larger set of examples to adequately demonstrate and capture the expected generalized behavior of the framework across a broader action space.

Additionally, we employ COT prompting, which instructs the model to explicitly reason through its decisions before generating a final task plan. As illustrated in Figure 2, the Coordinator is prompted to emit a structured reasoning block followed by the task plan itself. This improves both the coherence of the plan and the interpretability of the decision process.

To enable TACOS to react to dynamic events, the Coordinator is automatically triggered whenever an event requiring a new task plan occurs, such as when some drones become unavailable or new ones join the swarm. Because only a subset of such events can be automatically detected, TACOS also allows direct user interaction: at any time, the user may query the Coordinator, forcing a new task plan to be generated. In this case, the Supervisor informs the Coordinator of the status of the previous plan via $\hat{\mathcal{P}}$, which records all pending subtasks.

Beyond plan synthesis, the Coordinator also enables high-level swarm control via natural language. It acts as a centralized interface for one-to-many interactions, allowing users to issue commands such as *"Split the swarm and surround the target"*, which are then translated into structured, machine-executable API calls. Regarding the safety of the generated task plan, Figure 2 details the guidelines provided to the Coordinator, which specify safety constraints such as preventing the assignment of multiple drones to the same target. At the high level, the primary safety mechanism consists of validating the API call structure; if a call is incorrectly constructed, the Coordinator is re-triggered to generate a valid plan. Ultimate safety is managed via delegation, as detailed in Section 3, where each executable action is implemented by a dedicated safe control algorithm.

4.2. Supervisor

The Supervisor LLM is responsible for transforming the Coordinator's high-level task plan into a temporal sequence of executable actions. It takes as input the reasoning and task plan produced by the Coordinator, the current swarm state $\mathcal{S}_s$, and the world state $\mathcal{S}_\mathcal{W}$. The set of available low-level actions $\mathcal{A}$ is encoded directly into the model's configuration prompt.

Unlike the Coordinator, which reasons abstractly about intent, the Supervisor operates in a closed-loop execution. At each cycle, it receives updated telemetry from the swarm and the environment and re-evaluates which actions should be issued next. The Supervisor's cycle duration is chosen empirically based on the characteristic time scale of the mission. In the experiments presented in this paper, the Supervisor was queried at 0.1 Hz. This feedback loop allows the Supervisor to perform temporal sequencing of API calls, respecting dependencies implied by the Coordinator's plan, and to assign actions to each UAV.

The Supervisor produces a structured output specifying, for each drone, the current task under execution and the action to be issued. To support the coordination between the Supervisor and the Coordinator, TACOS maintains a list of remaining subtasks, denoted by $\hat{\mathcal{P}}$. This structure captures the portion of the task plan that has not yet been executed and is provided to the Coordinator during replanning events, either triggered automatically by TACOS (currently, TACOS triggers a Coordinator replan whenever a drone fails or new drones become available) or requested manually by the user.

4.3. History Management

During a mission, to help the Coordinator understand context and track how the environment changes, the full history of interactions with the user is stored. This includes all past user commands, the corresponding reasoning and task plans, and any updates

to the swarm or environment. Keeping this history allows the Coordinator to handle references to earlier commands, e.g., "go back to the last location", and generate plans that are consistent with past instructions. In contrast, the Supervisor LLM operates with bounded memory. For each task plan, it keeps a temporary record of the commands it has issued and any updates to the swarm or environment. This memory helps it to manage action sequencing, avoid repeating actions, and track progress. Once the current task plan is completed, this memory is cleared, and a new execution cycle begins. This scoped memory model aligns with the Supervisor's role as a reactive executor operating over finite-horizon plans.

5. Ablation Study

To assess the contribution of each module within the TACOS framework, we perform an ablation study. For each pilot request, we measure the following metrics:

- **Success rate**: the percentage of trials in which the task was completed as intended.
- **Average number of steps** (L): the average number of Supervisor cycles per task.

We evaluate system performance under the following conditions:

- **TACOS Monolithic** (MNL): to evaluate the effect of collapsing high-level reasoning and execution into a single module. In this case, to keep a compact interaction history, we only track executed actions.
- **TACOS without reasoning** (w/oR): to measure the impact of providing the Coordinator's reasoning to the Supervisor and whether it improves the Supervisor's ability to correctly sequence and complete the task.
- **TACOS**: the full framework, as shown in Figure 1.

5.1. Simulated Environment

The simulation environment, shown in Figure 3, contains 51 known landmarks, denoted as $\mathcal{T} = \{\mathcal{T}_i\}_{i=1}^{51}$, distributed across five distinct areas: two parking lots containing 10 and 5 parked cars, respectively; a residential neighborhood with 10 villas; a park area with 20 trees; a business district with 4 skyscrapers; and 2 drone takeoff and landing pads. The four skyscrapers are also modeled as static obstacles to be avoided, represented as $\mathcal{O} = \{\mathcal{O}_i\}_{i=1}^{4}$. Simulation parameters are adopted from the ATOMICA framework [10], with maximum velocity $v_{\max} = 1.7\,\mathrm{m/s}$ and maximum acceleration $a_{\max} = 6.2\,\mathrm{m/s^2}$.

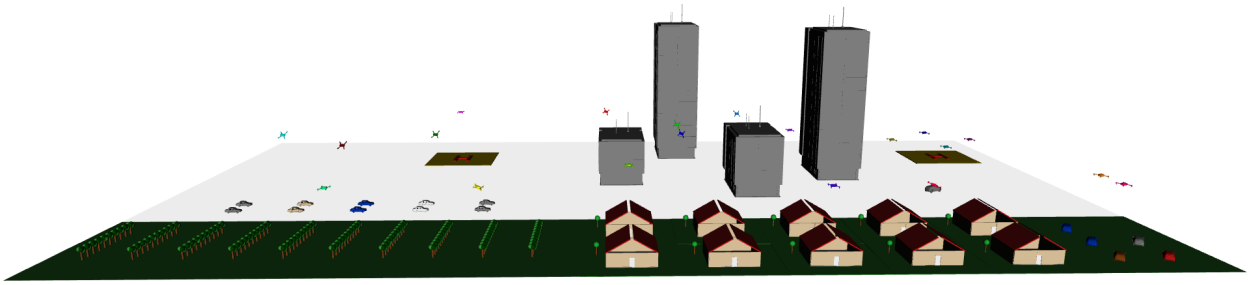


Figure 3. Simulation environment.

5.2. Model Configuration

For each setting, we performed 50 simulation runs using the open-source `gpt-oss:20b` (https://ollama.com/library/gpt-oss:20b, accessed on 2 June 2025) model for both the Coordinator and the Supervisor.

For the Coordinator, we set the temperature parameter to 1.0, whereas for the Supervisor, we employ a lower value of 0.4. This distinction in parameter tuning reflects the fundamentally different requirements of their respective roles. The Supervisor is strictly

responsible for executing the provided task plan; it must not generate new information, necessitating outputs that are more deterministic. Conversely, the Coordinator is tasked with interpreting potentially broad natural language commands to formulate both a comprehensive task plan and a corresponding reasoning block.

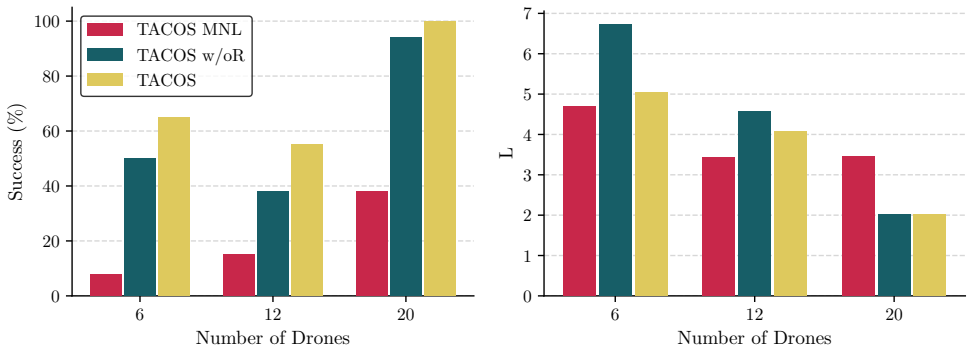
5.3. Mission

The mission is structured into two successive tasks designed to test the basic capabilities of the framework. For each ablation configuration, we evaluated performance with increasing swarm sizes: 6, 12, and 20 drones.

- **Task 0:**
  - **Description:** all UAVs are instructed to take off.
  - **Success criteria:** the Coordinator successfully generates a task plan containing a takeoff command for every drone, and the Supervisor executes each command without error.
  - **Prompt:**
    - *User request prompt:* ‘‘`The user requested the following task: takeoff. Create a suitable mission plan`’’.
    - *Dynamic event prompt:* ‘‘`There has been a change in the swarm state. We executed part of the original task plan but the following actions are still to be executed: [remaining-task-plan]. Allocate the missing actions to the available drones. Do not reissue actions that have already been executed`’’
- **Task 1:**
  - **Description:** the pilot requests to find a suspect hiding in a car. Before the mission is completed, a drone failure is simulated to evaluate TACOS's response to dynamic events.
  - **Success criteria:** the Coordinator generates a comprehensive task plan assigning each car to at least one drone. Following the simulated dynamic failure, the Coordinator must effectively reassign all unexecuted tasks to the remaining available drones, and the Supervisor must execute these commands in the correct sequence.
  - **Prompt:**
    - *User request prompt:* ‘‘`The user requested the following task: There is a suspect hiding in one of the cars, divide the swarm into two groups and inspect the cars as fast as possible. Create a suitable mission plan`’’.
    - *Dynamic event prompt:* ‘‘`There has been a change in the swarm state. We executed part of the original task plan but the following actions are still to be executed: [remaining-task-plan]. Allocate the missing actions to the available drones. Do not reissue actions that have already been executed`’’

5.4. Results

The results are summarized in Tables 2 and 3 and Figure 4, reporting both the success rate and the average number of steps L, computed over successful trials only. The results for Task 0, i.e., the takeoff task, are not included in Figure 4, as all configurations successfully completed the task in a single iteration with a 100% success rate. In the reported results, any trial that failed to reach the mission objective due to planning dead-ends or repeated command loops was recorded as a failure in the aggregate success rate.



(**a**) (**b**)

Figure 4. Performance evaluation of TACOS configurations for an increasing number of drones. (**a**) success rate. (**b**) The average number of Supervisor's cycles to complete a task.

Table 2. Success rate of TACOS configurations, 95% CI.

	6 Drones	**12 Drones**	**20 Drones**
TACOS MNL	0.08 ± 0.075	0.15 ± 0.138	0.38 ± 0.127
TACOS w/oR	0.5 ± 0.138	0.38 ± 0.134	0.94 ± 0.065
TACOS	0.65 ± 0.135	0.55 ± 0.138	$1.0 - 0.065$

Table 3. Average number of Supervisor's cycles, 95% CI.

	6 Drones	**12 Drones**	**20 Drones**
TACOS MNL	4.7 ± 1.5	6.72 ± 1.4	5.05 ± 0.94
TACOS w/oR	3.42 ± 1.8	4.57 ± 1.5	4.07 ± 1.2
TACOS	3.45 ± 1.3	2.02 ± 1.3	2.02 ± 1.4

The ablation study highlights the following facts:

1. Receiving both the reasoning and the task plan allows the supervisor to infer the correct temporal action sequence. TACOS w/oR has a lower success rate than the full TACOS framework, considering all three swarm sizes.
2. Explicitly dividing task planning and execution helps manage missions requiring multiple execution cycles. This is evident from the superior success rate of TACOS with respect to TACOS MNL shown in Figure 4.
3. TACOS demonstrates better parallelization capabilities compared with TACOS w/oR. This effect is particularly evident in the number of Supervisor cycles observed with a swarm of six UAVs: TACOS w/oR requires, on average, nearly two additional cycles compared with both the full framework and the monolithic configuration.

In general, we observe that TACOS MNL often issues repeated actions to different drones across iterations, with this behavior worsening as the number of iterations increases. In contrast, TACOS w/oR fails to parallelize tasks, favoring sequential execution. TACOS, on the other hand, does not exhibit these types of failures. It should be noted that, due to the simulated drone failure, the framework cannot complete the task in a single step, even when the number of available drones exceeds the number of targets. Finally, we observed that the Coordinator did not generate invalid API calls (e.g., incorrect function names or parameter mismatches). We attribute this to the minimal syntax and constrained parameter space of the API.

Table 4 presents the average inference times (in seconds) recorded during the baseline simulations. For both TACOS w/oR and the full TACOS framework, we report the Coordinator's inference time followed by that of the Supervisor. For the monolithic version, we

provide the inference time of the only model used. The results demonstrate that inference times remain consistent regardless of swarm size or the specific framework version tested. Notably, the monolithic version achieves a lower overall inference time, as it relies on a single LLM rather than a hierarchical dual-model architecture.

Table 4. Average inference times in seconds.

	6 Drones		12 Drones		20 Drones	
	Coord.	**Sup.**	**Coord.**	**Sup.**	**Coord.**	**Sup.**
TACOS MNL	8.46 s	–	5.96 s	–	6.31 s	–
TACOS w/oR	8.55 s	4.77 s	8.49 s	5.04 s	7.21 s	5.15 s
TACOS	11.05 s	4.53 s	8.06 s	5.66 s	8.0 s	5.64 s

5.5. Comparison with Classical Task Assignment

To evaluate the performance of TACOS, we executed a comparison against a baseline non-LLM approach. We conducted the same experiment described in Section 5.3, where the swarm was tasked with inspecting 15 parked cars in an urban environment, and a dynamic failure was introduced.

Algorithm 1 presents the pseudocode for the baseline approach. The procedure begins by initializing the sets of landmarks $\mathcal{T}$, available drones $\mathcal{D}$, obstacles $\mathcal{O}$, and unvisited landmarks $\mathcal{U}_t$. The algorithm first ensures all drones have transitioned to a flying state, executing a takeoff command where necessary. It then enters a control loop that persists while $\mathcal{U}_t$ is non-empty. In each iteration, we get the list $\mathcal{D}_a$ of all the drones $d \in \mathcal{D}$ in flight and not engaged in a visiting task; we then assign each unvisited landmark to the available drones, minimizing the travel distance. Finally, the algorithm evaluates the state of each drone; if the distance to its target landmark is less than 0.3 m, the landmark is marked as visited, $\mathcal{U}_t$ is updated, and the loop continues.

Algorithm 1 Baseline

$\mathcal{T}, \mathcal{D}, \mathcal{O}, \mathcal{U}_t \leftarrow initialize$
while $any(d)$ in $\mathcal{D}$ s.t. $d.status \neq flying$ **do**
  $takeoff(d)$
end while
while $len(\mathcal{U}_t) > 0$ **do**
  $\mathcal{D}_a = [d \in \mathcal{D} | d.status == flying \wedge isNotVisiting(d)]$
  $assignVisit(\mathcal{D}_a, \mathcal{U}_t)$
  for d in $\mathcal{D}$ **do**
    if $distance(d.pos, d.target) < 0.3$ m **then**
      $stopVisiting(d)$
      $\mathcal{U}_t = \mathcal{U}_t - d.target$
    end if
  end for
end while

Table 5 presents the results of this simulation. As can be seen, the baseline performs better; as discussed in Section 1, when the scenario and tasks are entirely predefined, a classical algorithm will typically be faster and yield superior performance. However, the primary advantage of TACOS over such specialized algorithms lies in its ability to leverage semantic information to inform task allocation. In this same scenario, a user could initially request a distance-based assignment and later pivot to a different assignment driven by previously unknown priorities. Such dynamic adaptability is largely unattainable with rigidly defined non-LLM approaches, where changes in task allocation priorities need to be manually developed beforehand to be used during mission execution. Finally, due

to the inherently modular architecture of TACOS, incorporating this classical baseline as an accessible API would be straightforward, effectively bridging the performance gap observed in this specific use case.

Table 5. Average distance flown in meters by the swarm.

	6 Drones	12 Drones	20 Drones
Baseline	196.96 m	78.95 m	51.42 m
TACOS MNL	198.26 m	136.84 m	78.55 m
TACOS w/oR	197.73 m	112.57 m	81.23 m
TACOS	197.1 m	105.44 m	70.13 m

6. Target Search Case Study

This section presents a target search experiment designed to evaluate the performance of TACOS in coordinating a multi-drone system. Specifically, we demonstrate TACOS's ability to assist a human operator in locating a target within a known environment and subsequently tracking it using multiple drones. We further evaluate the system's robustness to dynamic events by simulating multiple UAV failures during the mission. While states are formally defined in $\mathbb{R}^3$, all experimental evaluations utilize fixed-altitude planar coordination (2.5D), where the altitude z remains constant, and the executable action set is restricted to the 2D plane.

For real-time trajectory generation, TACOS integrates *ATOMICA* [10], a real-time, distributed, collision-free multi-agent motion planner. ATOMICA serves as the trajectory generation backend for all `goto()` and `start_tracking()` actions issued by the Supervisor. The generated trajectories are subject to a maximum velocity of $v_{\max} = 2$ m/s and a maximum acceleration $a_{\max} = 6.2\ \mathrm{m/s^2}$.

Experiments have been conducted on a computer running Ubuntu22 LTS using an NVIDIA RTX 6000 GPU (NVIDIA, Santa Clara, CA, USA). Communication between TACOS and the drones is performed using ROS Noetic (https://wiki.ros.org/noetic, accessed on 2 February 2026).

The simulated environment is the one described in Section 5.1. In addition, the simulation includes a moving target vehicle with known position and velocity, representing a moving agent that attempts to evade the swarm during the search task. For all the simulated experiments, we assume perfect low-level trajectory tracking for the quadrotors. The environment is designed to demonstrate how TACOS assists a human operator through an intuitive, natural-language interface, enabling the seamless coordination of large multi-drone systems. The integration of on-board perception and autonomous target detection is left for future work.

Simulation Results

We conducted a series of simulations to evaluate both the capabilities and limitations of TACOS. The mission consists of locating a moving target within the environment and tracking it as it attempts to escape. To assess the ability to handle a dynamic environment, drone failures are intentionally introduced during the simulations, forcing TACOS to replan and reallocate tasks. Videos of the simulation runs are provided in the Supplementary Materials.

We conducted simulations considering swarms of 6, 12, and 20 drones. Due to space limitations, we only report here the interactions obtained with a swarm of 6 drones, which highlight the three core capabilities of the framework (see Section 1). Figure 5 shows the initial interaction in which the operator defines the mission objectives through natural language commands. During the execution of the first task, a target escape maneuver is simulated, and the Coordinator is queried to generate a new plan that initiates target tracking (no perception module is used; global information about all entities is assumed

to be available to TACOS and the drones). The resulting task plan, shown in Figure 6, demonstrates that the Coordinator successfully integrates the new tracking request while preserving the pending subtasks from the previous plan. Subsequently, we simulate the failure of two drones, one of which was assigned to track the target. As illustrated in Figure 7, TACOS triggers a replanning phase, reassigning available drones to complete the mission objectives. Leveraging the full history of requests available to the Coordinator, TACOS assigns available drones to each unexecuted task in the partial task plan $\hat{\mathcal{P}}$ and automatically allocates a second drone to the previously requested tracking task, even without an explicit new command. It is interesting to note that, in this instance, the Coordinator chose to leave one drone idle during the inspection phase, dividing the inspection of the 15 cars into three consecutive subtasks executed by five drones each. However, the previously idle drone was later assigned to the tracking task. Indeed, the number of drones assigned by the Coordinator may vary across runs due to the inherent stochasticity of LLMs. This variability in the Coordinator's behavior can be mitigated by carefully shaping the prompt or by fine-tuning the underlying model. To maintain plug-and-play compatibility with different language models, we did not pursue fine-tuning in this work. Nevertheless, we observed that the framework's behavior can be effectively guided by providing more specific and constrained natural language instructions to the Coordinator. A key feature of TACOS is its ability to support continuous interaction with the Coordinator, allowing the operator to refine prompts whenever the observed behavior deviates from expectations. For example, when the user issues a new request that interrupts an ongoing task, they can explicitly specify whether the new instruction should take priority over the current one, enabling flexible yet controllable high-level supervision. The demonstration video for all three swarm sizes is available at the following link: https://tinyurl.com/TacosSimulated (accessed on 11 November 2025).

Pilot: There are multiple suspects hiding, possibly in different cars. Inspect the cars and find them

Coordinator:
{'reasoning': All six drones are currently flying and available for new commands. The user wants to inspect all 15 cars to locate suspects hiding inside them. To cover all cars efficiently, we assign each drone a sequence of setpoint commands that guide it to a distinct car. Each drone will visit three cars in a round-robin fashion, keeping a safe distance between drones by ensuring no two drones are commanded to the same car at the same time. Drone5 is left idle because the other five drones can already cover all cars. The plan lists the setpoint commands in order so that each drone completes its assigned trip before moving on to the next car.', 'task_plan':[…]}

Supervisor:
{ "drone_ids": ["drone0", "drone1", "drone2", "drone3", "drone4", "drone5"],
"status": ["executing_subtask", "executing_subtask", "executing_subtask", "executing_subtask", "executing_subtask", "completed"],
"current_subtask": ["inspect_car", "inspect_car", "inspect_car", "inspect_car", "inspect_car", None],
"action_required": [(0, "drone0",−95, 20), (0, "drone1",−80, 25.5), (0, "drone2",−50, 20), (0, "drone3",−35, 25.5), (0, "drone4", 98,−20)],
"log": "Sent initial setpoint commands to each drone for first car inspection"}

< intermediate supervisor interactions >

Supervisor:
{ "drone_ids": ["drone0", "drone1", "drone2", "drone3", "drone4", "drone5"],
"status": ["executing_subtask", "executing_subtask", "executing_subtask", "executing_subtask", "executing_subtask", "completed"],
"current_subtask": ["inspect_car", "inspect_car", "inspect_car", "inspect_car", "inspect_car", None],
"action_required": [(0, "drone0",−80, 20), (0, "drone1",−80, 25.5), (0, "drone2",−50, 25.5), (0, "drone3",−35, 25.5), (0, "drone4", 108,−12)],
"log": "Current subtasks completed, issuing next setpoints for each drone»}

Figure 5. TACOS executing the 'Moving target demo'. Search phase.

Pilot: One is escaping! two drones are required for tracking him. This is a priority! In the meantime, keep looking for the other suspects.

Coordinator:
{'reasoning': The user prioritizes two drones for tracking the escaping suspect while still continuing to search the remaining cars. We have four unexecuted setpoints involving drones1-4. Drone0 and drone5 are free and will be assigned to the tracking task. The remaining drones will complete their pending setpoints. No drone is currently tracking, so we can issue start_track_target commands for drone0 and drone5 without conflict. The setpoints for drones1-4 are then executed in the order they were listed, keeping the drones apart and avoiding collisions.',
'task_plan':[(3,'drone0','suspect','front'),(3,'drone5','suspect','back'),(0,'drone1',−65,20),(0,'drone1',−65, 25.5) (0,'drone2 ',−35,20),(0,'drone3',98,−2),(0,'drone3',98,−12),(0,'drone4',108,−20)]}

Supervisor:
{"drone_ids": ["drone0", "drone1", "drone2", "drone3", "drone4", "drone5"],
"status": ["executing_subtask", "executing_subtask", "completed", "executing_subtask", "completed", "executing_subtask"],
"current_subtask": ["start_tracking", "move_to", null, "move_to", null, "start_tracking"],
"action_required": [(3, "drone0", "suspect", "front"), (3, "drone5", "suspect", "back"), (0, "drone1",−65, 25.5), (0, "drone3", 98,−2)],
"log": "Initiating tracking for drone0 and drone5; commanding drone1 to move to (−65,25.5) and drone3 to move to (98,−2).»}

Figure 6. TACOS executing the 'Moving target demo'. Tracking phase.

TACOS:
drone4 is unavailable
drone0 is unavailable

Coordinator:
{'reasoning': The current swarm state has drone0 and drone4 unavailable, drone5 already tracking the suspect, and drones1,2,3 flying. The user's priority is to have two drones tracking the escaping suspect while continuing the pending inspection tasks. To satisfy this, we assign drone3 to track the suspect (since it was not yet tracking and has no other mandatory command) and let drones1 and 2 finish their remaining setpoint commands. No other trackers need to be stopped.',
'task_plan':[(3,'drone3','suspect','front'),(0,'drone1',−65,20),(0,'drone1',−65,25.5),(0,'drone2',−35,20)]}

Figure 7. TACOS executing the 'Moving target demo'. Dynamic event.

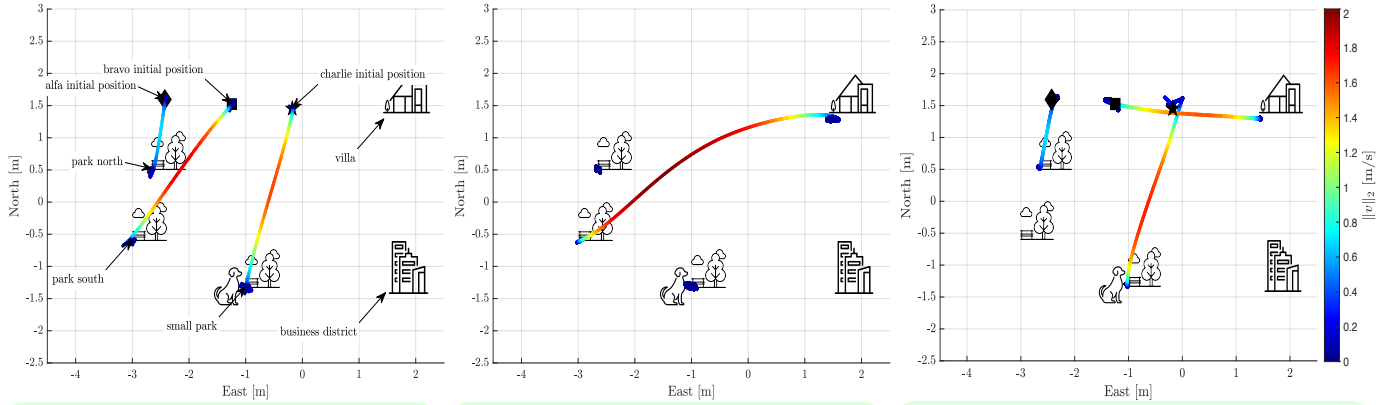
7. Real World Experiment

Flight tests were conducted in an indoor arena equipped with a 12-camera motion capture system tracking markers mounted on the drones. We used three quadrotors running PX4, referred to as Alfa, Bravo, and Charlie. The demonstration environment was a simplified map consisting of a small park, a large park (divided into north and south zones), a villa, and a business district. The mission scenario involved locating a lost dog placed in the small park. Figure 8 illustrates the pilot's command and TACOS's response during the interaction, along with the demonstration environment and the trajectories flown by the drones while executing the commands generated by the Supervisor.

Figure 8 highlights the advantages of using TACOS: when the user instructed the system to find the dog, starting from the most likely areas, TACOS inferred, based on semantic reasoning, that parks were more probable locations than the business district. Furthermore, during the initial interaction, TACOS correctly assigned the search task to

the closest available drone. In a subsequent interaction, when the user asked to keep monitoring the dog and notify the owner, the system assigned the task to a farther drone (in other instances of the same demonstration, TACOS instead selected the closest drone).

We repeated the demonstration 10 times. The behavior during the first interaction was consistent across all runs. However, when asked to monitor the dog and notify the owner, the system occasionally chose to monitor the dog with two drones and sent the third one to the villa. The demonstration video is available at the following link: https://tinyurl.com/TacosRealWorld (accessed on 11 November 2025).



Pilot a dog is lost, go find it. Prioritize the most likely areas.

Coordinator: {'reasoning': 'To find the lost dog, we need to prioritize the most likely areas. Since dogs are often found in parks or open spaces, we will send the drones to search these areas first. The big park north zone, big park south zone, and small park are potential locations. We will divide the drones among these locations, sending one drone to each of the two big parks and one drone to the small park and villa area as a secondary priority.', 'task_plan': [(0, 'alfa', -2.66, 0.51), (0, 'bravo', -3.00, -0.60), (0, 'charlie', -1.0, -1.33)]}
Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["executing_subtask", "executing_subtask","executing_subtask"],
"current_subtask": ["search_big_park_north", "search_big_park_south", "search_small_park"],
"action_required": [(0, 'alfa', -2.66, 0.51), (0, 'bravo', -3.00, -0.60), (0, 'charlie', -1.0, -1.33)],
"log": "Sending drones to their assigned search locations"
}
Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["completed", "completed", "completed"],
"current_subtask": [None, None, None],
"action_required": None,
"log": "Drones have reached their target locations and completed the search task"
}

(a) First interaction

Pilot charlie has found the dog in the small park, continue monitoring the dog and send a drone to the villa to notify its owner.

Coordinator: {'reasoning': 'Charlie has found the dog in the small park, so it will continue to monitor the dog. We need to send another drone to the villa to notify the owner. Since alfa is currently at the big park north zone, we will send bravo to the villa as it is closer to the villa than alfa.', task_plan: [(0, 'bravo', 1.44, 1.28)]}

Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["completed", "executing_subtask", "completed"],
"current_subtask": [None, "notify_owner", None],
"action_required": [(0, "bravo", 1.44, 1.28)],
"log": "Bravo is being sent to the villa to notify the owner"
}

Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["completed", "completed", "completed"],
"current_subtask": [None, None, None],
"action_required": None,
"log": "Bravo has reached the villa and completed its task"
}

(b) Second interaction

Pilot send the drones to the initial takeoff position.

Coordinator: {'reasoning': 'The mission is complete, and we need to send the drones back to their initial takeoff positions. The initial positions were (-2.43, 1.59), (-1.25, 1.52), and (-0.18, 1.45). We will send each drone to its corresponding initial position.', 'task_plan':[(0,'alfa',-2.43,1.59), (0,'bravo',-1.25,1.52), (0,'charlie',-0.18,1.45)]}
Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["executing_subtask", "executing_subtask", "executing_subtask"],
"current_subtask": ["return_to_initial_position", "return_to_initial_position", "return_to_initial_position"],
"action_required": [(0, 'alfa', -2.43, 1.59), (0, 'bravo', -1.25, 1.52), (0, 'charlie', -0.18, 1.45)],
"log": "Return to initial position command received, sending drones back to their takeoff positions"
}
Supervisor:
{
"drone_ids": ["alfa", "bravo", "charlie"],
"status": ["completed", "completed", "completed"],
"current_subtask": [None, None, None],
"action_required": None,
"log": "Drones have reached their initial takeoff positions, mission completed"
}

(c) Third interaction

Figure 8. TACOS executing the 'Find the Dog' mission by directing drones to the most likely search area based on semantic reasoning. The figure shows the simplified demonstration environment and the trajectories flown by the drones. Trajectories are color-coded by velocity magnitude, with warmer colors (toward red) indicating higher speeds.

8. Conclusions

In this work, we explored the use of large language models as interfaces for multi-drone systems. The proposed framework bridges low-level planning and control algorithms with high-level mission execution by enabling natural language interaction between a human operator and a multi-drone system. We demonstrated the system in a simplified, lab-scale search-and-rescue scenario, where the semantic reasoning capabilities of the language model improved the efficiency of the search strategy. Additionally, we conducted an ablation study to evaluate the contribution of each module in the proposed architecture. The proposed framework has several limitations. In particular, it assumes full world-state knowledge, does not include onboard perception, and is limited to short-horizon missions. Nevertheless, we believe TACOS serves as a proof-of-concept demonstration of LLM-based swarm coordination. Future work will focus on integrating onboard perception modules and expanding the available API set. Additionally, to support long-horizon missions where history management could become a bottleneck due to context window limitations, we plan to implement history management strategies such as integrating a Retrieval-Augmented Generation (RAG) architecture to dynamically fetch task-relevant context or employing periodic LLM-driven summarization to prevent token overflow during extended operations.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/drones10040251/s1, Video S1: TACOS simulation with 6, 12 and 20 drones.

Author Contributions: Conceptualization, A.N., R.R. and M.L.; methodology, A.N., R.R.; software, A.N.; validation, A.N., R.R.; investigation, A.N., R.R.; resources, M.L.; writing—original draft preparation, R.R., A.N.; writing—review and editing, R.R., A.N. and M.L.; visualization, A.N., R.R.; supervision, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge the financial support received under the European project PON FSE REACT-EU and the resources allocated by Ministerial Decree No. 1061 of 10 August 2021 for active and accredited doctoral programs in the framework of the XXXVII cycle.

Data Availability Statement: The code supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The authors express their gratitude to all the members of the ASCL laboratory for their support during experimental activity.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

COT	Chain of Thought
ICL	In-Context Learning
LLM	Large Language Model
LP	Linear Program
MNL	Monolithic
TACOS	Task Agnostic Coordinator of a multi-drone System
UAV	Unmanned Aerial Vehicle

Appendix A. Modelfiles

Appendix A.1. Coordinator

This appendix contains the Modelfiles used to fine-tune both the Coordinator and the Supervisor. Listing A1 details the section of the Modelfile that defines the Coordinator's role, expected output format, and execution guidelines. First, we briefly describe the

Coordinator's function within the TACOS framework. Next, we outline the available APIs and specify their required parameters. We also explicitly define the desired output format to enforce structural consistency and ensure the LLM's responses can be easily parsed. Finally, we establish the behavioral rules the Coordinator must follow during execution. Listing A2 presents the specific examples provided to enhance the Coordinator's capabilities via ICL. Similarly, Listings A3 and A4 detail the corresponding components of the Supervisor's Modelfile.

Listing A1. Coordinator Modelfile: output format, guidelines, and description.

```
You are a coordinator of a multidrone system.
You are responsible for translating high level user instructions,
expressed in natural language, into a structured task plan
compatible with the available API:

{
    0: goto(uav_id, x, y)
    # Sends the specified UAV to (x, y)
    1: arm_takeoff_drone(uav_id)
    # Arms and takes off the specified UAV
    2: land(uav_id)
    # Lands and disarms the specified UAV
    3: start_tracking(uav_id, target_name, position)
    # Commands the specified UAV to start tracking
    the specified target, position can be either
    'front' or 'back' depending on the position
    with respect to the target.
    4: stop_track_target(uav_id)
    # Commands the specified UAV to stop tracking
    whatever target it is tracking.
}

# Status
Each drone in the system has a status flag,
it can be one of the following:
1. 'unavailable', a drone in the unavailable status
cannot be commanded in any way.
You should not consider drones in this state while planning.
2. 'idle', a drone in the idle status is available for takeoff only
3. 'flying', a drone in the flying status can be freely commanded
4. 'tracking', a drone in the tracking status can only be commanded to stop
    tracking

If a drone is in the 'unavailable' state you cannot command it in any way,
    drones switch out of the 'unavailable' state due to external actions not
    under your control.

# Output
Your output must be composed by two sections.
1. A reasoning block. a natural language explanation of the generate task
    plan that accomplishes the mission requested by the user.
2. The task plan. A python list of ALL the API calls needed to complete the
    mission as requested by the user regardless of temporal constraints.

and must have the following structure

{
    'reasoning': the reasoning block,
    'task_plan': [(function_id, arg...), ..., (function_id, arg...)]
}

Ensure that the output has this precise format and avoid any other token.

## Guidelines

1. If the status of a drone changes during the mission replan accordingly.
    For example, if a drone was tracking and becomes unavailable try to
    assign an available one to fulfill its task.
2. Avoid unsafe behavior. Do not send multiple drones to the same position or
     into a known obstacle, always keep a safe distance.
```

Listing A1. *Cont.*

```
3. If the user asks for a shape or formation, calculate approximate
    coordinates before issuing commands.
4. When asked to surround an object or point (x, y), place drones evenly
    spaced in a circle around it, at equal radius. Default radius = 1.0
    unless specified.
5. A drone must takeoff before being able to receive setpoints
```

Listing A2. Coordinator Modelfile: In-Context Learning examples provided.

```
# Examples

Example 1
User: [('drone0', ...), ('drone1', ...), ('drone2', ...), ('drone3', ...)],
    command: Of the previous task plan the following actions have not been
    executed [], the user requested the following: takeoff. Complete the
    missing actions if needed and execute the new request.
System: {'reasoning': 'All four drones are currently in the idle status and
    have not executed any previous actions. The user requested a takeoff,
    which requires each idle drone to be armed and to take off before it can
    receive any setpoints. Therefore, we will issue an arm_takeoff_drone
    command for each drone.', 'task_plan': [(1, 'drone0'), (1, 'drone1'), (1,
     'drone2'), (1, 'drone3')]}

Example 2:
User: [('drone0', ...), ('drone1', ...), ('drone2', ...), ('drone3', ...)],
    command: Of the previous task plan the following actions have not been
    executed [] the user requested the following: split the swarm into two
    groups, send one around -1,-1 and when they arrive send the other around
    5,5. Complete the missing actions if needed and execute the new request.
System: {'reasoning': 'All four drones are currently flying at altitude 2.5
    and available for new commands. The user wants to split the swarm into
    two groups of two drones each. The first group should orbit around the
    point (-1,-1) and the second group should orbit around (5,5). We will
    assign drone0 and drone1 to the first group and drone2 and drone3 to the
    second group. For each group we issue two goto calls that place the
    drones at a radius of 1.0 around the target point. Group-1 positions:
    (0,-1) and (-2,-1). Group-2 positions: (6,5) and (4,5). The commands are
    ordered so that the first group receives its targets before the second
    group is dispatched.', 'task_plan':[(0,'drone0',0,-1),(0,'drone1',-2,-1)
    ,(0,'drone2',6,5),(0,'drone3',4,5)]}

Example 3:
User: [('drone0', ...), ('drone1', ...), ('drone2', ...), ('drone3', ...)],
    command: Of the previous task plan the following actions have not been
    executed [(0, 'drone2', 6, 5), (0, 'drone3', 4, 5)], the user requested
    the following: inspect the trees one by one. Complete the missing actions
     if needed and execute the new request.

System: {'reasoning':'The previous plan left two setpoint commands unexecuted
     for drone2 and drone3. Those are executed first. Then the user requested
     that the swarm inspect all trees one by one. To satisfy this with four
    drones, we assign each drone to visit a tree in turn, cycling through the
     trees. This results in 8 setpoint commands: each drone visits a tree
    sequentially. The drones are already flying and can receive setpoints
    immediately.','task_plan':[(0, 'drone2', 6, 5), (0, 'drone3', 4, 5), (0,'
    drone0',-20,20),(0,'drone1',-25,20),(0,'drone2',-15,20),(0,'drone3
    ',-20,25.5),(0,'drone0',-25,25.5),(0,'drone1',-15,25.5),(0,'drone2
    ',-10,20),(0,'drone3',-10,25.5)]}

Example 4:
User: [('drone0', ...), ('drone1', ...), ('drone2', ...), ('drone3', ...)]
    There is a suspect, send two drones to follow him
System: {'reasoning': 'The user commanded to follow the suspect with two
    drones. I will assign the first two drones to the tracking task', '
    task_plan': [(3, 'drone0', 'suspect', 'front'), (3, 'drone1', 'suspect',
    'back')]}
```

Appendix A.2. Supervisor

Listing A3. Supervisor Modelfile: output format, guidelines, and description.

```
You are an autonomous drone task manager LLM overseeing and controlling a
    swarm of drones. You operate in a closed-loop system, continuously
    reacting to updated world information. You will receive a task plan from
    a coordinator, you are responsible for transforming the provided high
    level task plan into a temporally ordered sequence of API actions. Your
    goal is to ensure that the entire task plan is executed correctly and in
    accordance with the reasoning. Once a task plan is received you enter
    closed-loop execution. At each time step, you must analyze the current
    state of the world and issue the appropriate commands to the swarm.

You have access to the following API

{
    0: goto(uav_id, x, y)
    # Sends the specified UAV to (x, y)
    1: arm_takeoff_drone(uav_id)
    # Arms and takes off the specified UAV
    2: land(uav_id)
    # Lands and disarms the specified UAV
    3: start_tracking(uav_id, target_name, position)
    # Commands the specified UAV to start tracking
    the specified target, position can be either
    'front' or 'back' depending on the position
    with respect to the target.
    4: stop_track_target(uav_id)
    # Commands the specified UAV to stop tracking
    whatever target it is tracking.
}

# Status

Each drone in the system has a status flag, it can be one of the following:
1. 'unavailable', a drone in the off status cannot be commanded.
2. 'idle', a drone in the idle status is available for takeoff only
3. 'flying', a drone in the flying status can be freely commanded
4. 'tracking', a drone in the tracking status can only be commanded to stop
    tracking

If a drone is in the 'unavailable' state you cannot command it in any way,
    drones switch out of the 'unavailable' state due to external actions not
    under your control.

# Input and Output

Initially you will receive the following information:
1. A list of the available drones in the swarm with their current position
    and velocity
2. A list of the known objects in the environment
3. A reasoning block that explains how the task plan has been created.
4. The complete task plan, a python list of ALL the API calls needed to
    complete the mission regardless of temporal constraints.

Your output must have the following structure:

{
  "drone_ids": ["droneName1", "droneNameN"],
  "status": ["executing_subtask | completed", ..., "executing_subtask |
    completed"],
  "current_subtask": ["subtask_id or None if completed", ..., "subtask_id or
    None if completed"],
  "action_required": [(function_id, arg0, arg1, ...), ...] | None
  "log": "brief explanation of decision"
}
```

Listing A3. *Cont.*

```
Ensure that the output has this precise format.

# Guidelines
1. Parallelise the tasks of the swarm's UAVs if possible without conflicting
    with the reasoning of the task plan.
2. Do not re-issue the same action if it is already being executed.
3. Evaluate if the current subtask is completed by checking the drone's
    position with respect to the target_position.
4. If all subtasks are done, mark the task list as completed.
5. Keep responses as short as possible and focus on task progress.
6. Generate only one dictionary at a time.
7. Avoiding showing any calculation.
8. Do not issue the arm and takeoff command for a drone whose z-coordinate is
     already greater than 0
```

Listing A4. Supervisor Modelfile: in-context learning examples provided.

```
# Example
input: {[('drone0', ...), ('drone1', ...)], 'reasoning': 'There are two
    drones in the swarm. All the drones must take off. The current positions
    of the drones are (0.0, 0.0) and (-2.0, 0.0).', 'task_plan': [(1, 'drone0
    '), (1, 'drone1')]}
output:
  {
    'drone_ids': ['drone0', 'drone1'],
    'status': ['executing_subtask', 'executing_subtask'],
    'current_subtask': ['takeoff', 'takeoff'],
    'action_required': [(1, 'drone0'), (1, 'drone1')],
    'log': 'Takeoff command received, taking off both drones from initial
    positions'
  }
input: {[('drone0', ...), ('drone1', ...)]}
output:
  {
    'drone_ids': ['drone0', 'drone1'],
    'status': ['completed', 'completed'],
    'current_subtask': [None, None],
    'action_required': None,
    'log': 'Drones have completed their takeoff task'
  }

# Example
input: {[('drone0', ...), ('drone1', ...)], 'reasoning': 'Surround means
    create a circle around the object. Since there are only two drones
    available and they are not yet airborne, we need to take them off first.
    Assume a radius of one meter. Divide 360 by 2 -> 180 degrees between each
     drone. Assign: drone0 (5 + radius*cos(90), 5 + radius*sin(90)), drone1
    (5 + radius*cos(270), 5 + radius*sin(270))', 'task_plan': [(1, 'drone0'),
     (1, 'drone1'), (0, 'drone0', round(5.0 + 1.0 * 0.0,1), round(5.0 + 1.0 *
     1.0,1)), (0, 'drone1', round(5.0 + 1.0 * 0.0,1), round(5.0 - 1.0 *
    1.0,1))]}
output:
{
  'drone_ids': ['drone0', 'drone1'],
  'status': ['executing_subtask', 'executing_subtask'],
  'current_subtask': ['takeoff', 'takeoff'],
  'action_required': [(1, 'drone0'), (1, 'drone1')],
  'log': 'Takeoff command received, taking off both drones from initial
    positions'
}
```

Listing A4. *Cont.*

```
input: {[('drone0', ...), ('drone1', ...)]}
output:
  {
    'drone_ids': ['drone0', 'drone1'],
    'status': ['executing_subtask', 'executing_subtask'],
    'current_subtask': ['surround_tree', 'surround_tree'],
    'action_required': [(0, 'drone0', 5.0, 6.0), (0, 'drone1', 5.0, 4.0)],
    'log': 'Drones have taken off, sending them to their goal positions'
  }
input: {[('drone0', ...), ('drone1', ...)]}
output:
 {
  'drone_ids': ['drone0', 'drone1'],
  'status': ['executing_subtask', 'executing_subtask'],
  'current_subtask': ['surround_tree', 'surround_tree'],
  'action_required': [(0, 'drone0', 5.0, 6.0), (0, 'drone1', 5.0, 4.0)],
  'log': 'Drones have not reached their target, sending them to their goal
    positions'
 }

input: {[('drone0', ...), ('drone1', ...)]}
output:
{
  'drone_ids': ['drone0', 'drone1'],
  'status': ['completed', 'completed'],
  'current_subtask': ['None', 'None'],
  'action_required': None,
  'log': 'Drones have reached their target'
}
```

References

1. Cacace, J.; Finzi, A.; Lippiello, V.; Furci, M.; Mimmo, N.; Marconi, L. A control architecture for multiple drones operated via multimodal interaction in search & rescue mission. In Proceedings of the 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, Switzerland, 23–27 October 2016; pp. 233–239. [CrossRef]
2. Wattearachchi, W.D.; Lakshika, E.; Kasmarik, K.; Barlow, M. Designing Effective Human-Swarm Interaction Interfaces: Insights from a User Study on Task Performance. *arXiv* **2025**, arXiv:2504.02250. [CrossRef]
3. 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. *Inf. Fusion* **2025**, *122*, 103158. [CrossRef]
4. Kim, Y.; Kim, D.; Choi, J.; Park, J.; Oh, N.; Park, D. A survey on integration of large language models with intelligent robots. *Intell. Serv. Robot.* **2024**, *17*, 1091–1107. [CrossRef]
5. Choi, H.L.; Brunet, L.; How, J.P. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Trans. Robot.* **2009**, *25*, 912–926. [CrossRef]
6. Mercker, T.; Casbeer, D.W.; Millet, P.T.; Akella, M.R. An extension of consensus-based auction algorithms for decentralized, time-constrained task assignment. In Proceedings of the 2010 American Control Conference, Baltimore, MD, USA, 30 June–2 July 2010; pp. 6324–6329. [CrossRef]
7. Sewlia, M.; Verginis, C.K.; Dimarogonas, D.V. MAPS2: Multi-robot autonomous motion planning under signal temporal logic specifications. *Int. J. Robot. Res.* **2025**, 1–9. [CrossRef]
8. Sun, D.; Chen, J.; Mitra, S.; Fan, C. Multi-Agent Motion Planning From Signal Temporal Logic Specifications. *IEEE Robot. Autom. Lett.* **2022**, *7*, 3451–3458. [CrossRef]
9. Deghat, M.; Anderson, B.D.O.; Lin, Z. Combined Flocking and Distance-Based Shape Control of Multi-Agent Formations. *IEEE Trans. Autom. Control* **2016**, *61*, 1824–1837. [CrossRef]
10. Rubinacci, R.; Nazzari, A.; Lovera, M. Anytime Trajectory Optimization for MultI-Drone Systems With Guaranteed Collision Avoidance. *IEEE Control Syst. Lett.* **2025**, 9, 1255–1260. [CrossRef]
11. Tordesillas, J.; How, J.P. MADER: Trajectory Planner in Multiagent and Dynamic Environments. *IEEE Trans. Robot.* **2022**, *38*, 463–476. [CrossRef]
12. Zhou, X.; Wen, X.; Wang, Z.; Gao, Y.; Li, H.; Wang, Q.; Yang, T.; Lu, H.; Cao, Y.; Xu, C.; et al. Swarm of micro flying robots in the wild. *Sci. Robot.* **2022**, *7*, eabm5954. [CrossRef] [PubMed]

13. Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; Zeng, A. Code as policies: Language model programs for embodied control. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 9493–9500.
14. Meng, Y.; Chen, F.; Chen, Y.; Fan, C. AuDeRe: Automated Strategy Decision and Realization in Robot Planning and Control via LLMs. *arXiv* **2025**, arXiv:2504.03015. [CrossRef]
15. Zhu, W.; Dorigo, M.; Heinrich, M.K. Online automatic code generation for robot swarms: LLMs and self-organizing hierarchy. *arXiv* **2025**, arXiv:2510.04774. [CrossRef]
16. Cui, J.; Liu, G.; Wang, H.; Yu, Y.; Yang, J. TPML: Task Planning for Multi-UAV System with Large Language Models. In Proceedings of the 2024 IEEE 18th International Conference on Control & Automation (ICCA), Reykjavík, Iceland, 18–21 June 2024; pp. 886–891. [CrossRef]
17. Mandi, Z.; Jain, S.; Song, S. Roco: Dialectic multi-robot collaboration with large language models. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 286–299.
18. Nayak, S.; Morrison Orozco, A.; Have, M.; Zhang, J.; Thirumalai, V.; Chen, D.; Kapoor, A.; Robinson, E.; Gopalakrishnan, K.; Harrison, J.; et al. Long-horizon planning for multi-agent robots in partially observable environments. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 67929–67967.
19. Chen, Y.; Arkin, J.; Zhang, Y.; Roy, N.; Fan, C. Scalable Multi-Robot Collaboration with Large Language Models: Centralized or Decentralized Systems? In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 4311–4317. [CrossRef]
20. Strobel, V.; Dorigo, M.; Fritz, M. LLM2Swarm: Robot Swarms that Responsively Reason, Plan, and Collaborate through LLMs. *arXiv* **2024**, arXiv:2410.11387. [CrossRef]
21. Zhang, H.; Du, W.; Shan, J.; Zhou, Q.; Du, Y.; Tenenbaum, J.B.; Shu, T.; Gan, C. Building Cooperative Embodied Agents Modularly with Large Language Models. *arXiv* **2024**, arXiv:2307.02485. [CrossRef]
22. Lykov, A.; Karaf, S.; Martynov, M.; Serpiva, V.; Fedoseev, A.; Konenkov, M.; Tsetserukou, D. FlockGPT: Guiding UAV Flocking with Linguistic Orchestration. In Proceedings of the 2024 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct), Bellevue, WA, USA, 21–25 October 2024; pp. 485–488. [CrossRef]
23. Aikins, G.; Dao, M.P.; Moukpe, K.J.; Eskridge, T.C.; Nguyen, K.D. LEVIOSA: Natural Language-Based Uncrewed Aerial Vehicle Trajectory Generation. *Electronics* **2024**, *13*, 4508. [CrossRef]
24. 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 Robot. Autom. Lett.* **2025**, *10*, 12237–12244. [CrossRef]
25. Huang, Z.; Shi, G.; Wu, Y.; Kumar, V.; Sukhatme, G.S. Compositional Coordination for Multi-Robot Teams with Large Language Models. In Proceedings of the IEEE International Symposium on Multi-Robot & Multi-Agent Systems, Singapore, 4–5 December 2025.
26. Kahneman, D. *Thinking, Fast and Slow*; Macmillan: Basingstoke, UK, 2011.
27. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
28. Dong, Q.; Li, L.; Dai, D.; Zheng, C.; Ma, J.; Li, R.; Xia, H.; Xu, J.; Wu, Z.; Liu, T.; et al. A survey on in-context learning. *arXiv* **2022**, arXiv:2301.00234.