Unleashing Tool Engineering and Intelligence for Agentic AI in Next-Generation Communication Networks

Yinqiu Liu, Ruichen Zhang, Dusit Niyato, *Fellow, IEEE*, Abbas Jamalipour, *Fellow, IEEE*, Trung Q. Duong, *Fellow, IEEE*, and Dong In Kim, *Life Fellow, IEEE*

***Abstract*—Nowadays, agentic AI is emerging as a transformative paradigm for next-generation communication networks, promising to evolve large language models (LLMs) from passive chatbots into autonomous operators. However, unleashing this potential requires bridging the critical gap between abstract reasoning and physical actuation, a capability we term tool intelligence. In this article, we explore the landscape of tool engineering to empower agentic AI in communications. We first analyze the functionalities of tool intelligence and its effects on communications. We then propose a systematic review for tool engineering, covering the entire lifecycle from tool creation and discovery to selection, learning, and benchmarking. Furthermore, we present a case study on tool-assisted uncrewed aerial vehicles (UAV) trajectory planning to demonstrate the realization of tool intelligence in communications. By introducing a teacher-guided reinforcement learning approach with a feasibility shield, we enable agents to intelligently operate tools. They utilize external tools to eliminate navigational uncertainty while mastering cost-aware scheduling under strict energy constraints. This article aims to provide a roadmap for building the tool-augmented intelligent agents of the 6G era.**

***Index Terms*—Agentic AI, tool intelligence, model context protocol, and uncrewed aerial vehicle.**

I. Introduction

Recent breakthroughs in large language models (LLMs) are demonstrating a profound impact on communication intelligence, with proven successes in complex tasks (e.g., resource allocation [1] and channel prediction [2]), where traditional automation is increasingly strained [1]. Building on this momentum, the communications sector is rapidly advancing from standalone LLMs toward sophisticated agentic AI systems. By embedding LLMs as the core reasoning engine within a framework of perception, memory, and action components, agentic AI emerges as an intelligent entity [3]. This agent-based paradigm is pivotal, as it enables the autonomous pursuit of complex and multi-step communication tasks with minimal human supervision, directly aligning with the vision for fully intelligent sixth-generation and beyond communications [3].

The key to unlocking this agentic potential is *tool intelligence*: the capability for an agent to effectively interact with real-world infrastructures through the use of external tools. Tool intelligence bridges the gap from abstract reasoning to concrete, real-world actuation, moving beyond isolated task automation to reshape workflows. For instance, upon detecting user complaints about poor connectivity, HPE Marvis[1] can proactively initiate a diagnostic workflow. Instead of merely generating alerts, it invokes a network telemetry analytics tool to correlate user-experience data with real-time performance metrics, a configuration inspection utility to identify anomalies across underlying switches, and a traffic analysis module to trace packet loss patterns. Once a misconfigured aggregation switch is identified as the root cause, the system can autonomously generate and deploy the corrective configuration, completing the loop from perception to action.

As the application of agentic AI expands, tool intelligence is developing rapidly. In the industry, mainstream platforms, such as OpenAI's GPT and Google's Gemini, have seamlessly integrated foundational tools such as code interpreters and web browsers. They have also fostered a burgeoning ecosystem for third-party tools, from general utilities like *WolframAlpha* for complex computation to platform-specific plugins for services like *Expedia* and *Zapier*. In academia, foundational research has demonstrated how LLMs can be trained to master thousands of real-world application programming interface (API) [4], learn to self-correct tool usage [5], and even teach themselves to use tools they have never seen before [3]. By connecting an agent's reasoning to external capabilities, tools profoundly enhance its power, providing many benefits.

- **Grounding and Actuation:** Tools provide the essential interface for environmental grounding and real-world actuation, enabling agents to query real-time states and execute control commands. For instance, an agent can use

This research is supported by Seatrium New Energy Laboratory, Singapore Ministry of Education (MOE) Tier 1 (RT5/23 and RG24/24), the Nanyang Technological University (NTU) Centre for Computational Technologies in Finance (NTU-CCTF), and the Research Innovation and Enterprise (RIE) 2025 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) (Award I2301E0026), administered by Agency for Science, Technology and Research (A*STAR). The work of T. Q. Duong was supported in part by the Canada Excellence Research Chair (CERC) Program CERC-2022-00109 and in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant Program RGPIN-2025-04941. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2026-25470140).

Y. Liu, R. Zhang (corresponding author), and D. Niyato are with the College of Computing and Data Science, Nanyang Technological University, Singapore (E-mail: yinqiu001@e.ntu.edu.sg, ruichen.zhang@ntu.edu.sg, and dniyato@ntu.edu.sg).

A. Jamalipour is with the School of Electrical and Computer Engineering, University of Sydney, Australia, and with the Graduate School of Information Sciences, Tohoku University, Japan (E-mail: a.jamalipour@ieee.org).

T. Q. Duong is with the Faculty of Engineering and Applied Science, Memorial University, Canada, and with the School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, U.K., and also with the Department of Electronic Engineering, Kyung Hee University, South Korea (E-mail: tduong@mun.ca).

D. I. Kim is with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea (E-mail: dongin@skku.edu).

[1] Available at: https://www.hpe.com/sg/en/marvis-ai.html

a *Prometheus* client to query live network telemetry or invoke a *gNMI* API to reconfigure a router.
- **Specialized Expertise:** Tools grant access to a vast library of domain-specific algorithms that an LLM cannot accurately reproduce from its pre-trained knowledge. This includes invoking a *MATLAB* simulator[2] for complex beamforming calculations or calling a retriever to fetch 3GPP standards for protocol compliance.
- **Scalability & Modularity:** Agents can decompose complex problems into a manageable chain of sub-tasks via tools, fostering a modular and scalable approach to automation. For example, they can orchestrate a workflow involving a fault detection tool, a root-cause analysis tool, and a trouble-ticketing tool to manage the entire lifecycle of a network incident.

Nonetheless, unleashing the full potential of tool intelligence in agentic AI-empowered communications requires filling the following gaps. First, our investigation reveals that mainstream LLM tool ecosystems are dominated by user- and web-centric applications[3]. In contrast, there is a significant scarcity of professional tools specifically designed for communications and networking. Second, the unique characteristics of communication networks impose much stricter requirements on tool operations. For instance, the additional latency and computational overhead incurred by tool invocation, while acceptable for a web search, can be prohibitive for real-time communication functions, such as vehicle-to-vehicle transmissions. Finally, the deterministic and high-stakes nature of communication networks means that a tool failure, or an LLM's probabilistic error in managing tools, could lead to service degradation/outages [6].

In this article, we explore tool engineering and intelligence for agentic AI in next-generation communication networks. Specifically, we first analyze the importance of tool intelligence by examining the core limitations of LLMs, followed by the role tools play within the agentic AI framework and how they transform communication systems. Moreover, we review the crucial techniques to implement this vision in communication scenarios, a process that we term *tool engineering*. Finally, we conduct a case study on optimizing tool-assisted uncrewed aerial vehicle (UAV) communications. Our goal is to equip readers with a systematic understanding of how to integrate existing tools, optimize tool usage, and create new tools to build next-generation wireless systems.

The main contributions of this paper are as follows.

- *To the best of our knowledge, this is the first work to comprehensively explore tool intelligence in next-generation communications.* We first identify the motivation for tool intelligence, then describe the foundational role of tools within agentic AI frameworks. In addition, we analyze the transformative effects of tool intelligence on communication functionality and efficiency.
- We provide a systematic review of tool engineering, which we define as the set of key techniques required to realize tool intelligence in communication networks. Particularly, our review highlights five core engineering aspects, namely tool creation, discovery, selection, learning, and standard & benchmarking.
- We present a case study on realizing tool intelligence in UAV communications. We propose a teacher-guided reinforcement learning framework to enable tool-assisted trajectory planning, optimizing the agent's capability to intelligently schedule tool activations while strictly adhering to resource constraints.

[2]Available at: https://github.com/neuromechanist/matlab-mcp-tools

[3]Representative tools can be found at: https://mcp.so/

II. Agentic AI with Tool Intelligence for Next-Generation Communications

A. Motivation for Tool Intelligence

Although conventional AI agents have existed for decades (e.g., deep reinforcement learning (DRL) agents [1]), the recent integration of LLMs as core reasoning engines has dramatically expanded their cognitive capabilities [7]. Nonetheless, LLM-based agents are constrained by three limitations inherent in their LLM cores, which directly motivate the need for external tools.

The first challenge is regarding knowledge grounding and reliability. Since agents' LLMs are trained on static data, they often operate with outdated information and without access to real-time, communication-specific knowledge. This leads to the risk of error and hallucination, where the agent might generate factually incorrect or nonsensical content. Therefore, a communication agent must base its decisions on up-to-date and domain-specific data, creating a critical need for tools that connect to external knowledge sources, such as real-time network monitors, industrial standards (e.g., 3GPP), and scientific literature [7].

Second, agents face the challenge of translating reasoning into action. LLMs are reasoning and content-generation engines, which cannot directly execute a communication command, run a complex simulation, or perform a mathematical optimization. Therefore, tools are indispensable actuators that bridge the gap between the agent's cognitive processes and the physical or virtual network infrastructure. This is especially critical in the communications field, where operational environments are heterogeneous ecosystems of multi-vendor equipment, diverse protocols, and proprietary APIs that require precise, structured commands for interaction [8].

Third, a fundamental trade-off exists between an LLM's generalist knowledge and the specialized precision required for many communication tasks. The primary strength of an LLM is its vast, pre-trained knowledge, providing generalizability and zero-shot capabilities [7]. However, LLMs still exhibit inherent limitations in deep logical inference, precise mathematical computation, and the kind of high-fidelity simulation essential for network engineering. Consequently, when faced with a complex task like optimizing transmission power, an LLM will rarely match the performance of highly optimized, domain-specific algorithms.

B. Tools in Agentic AI

In the agentic AI context, a tool refers to an externally callable capability that an agent can invoke, thereby bridging the gap between the agent's internal intelligence and

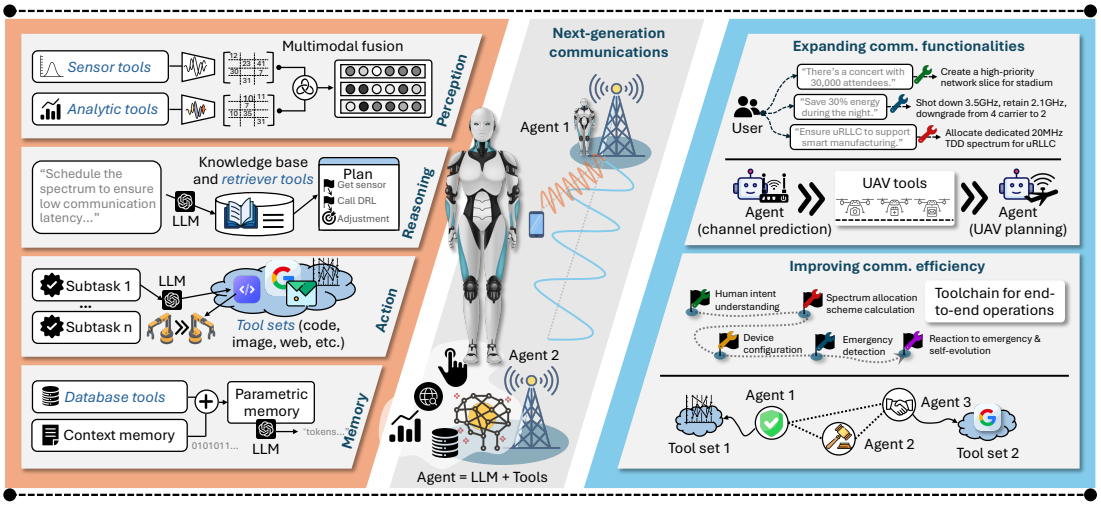

Fig. 1. (left): The general agentic AI framework, illustrating how tool intelligence is deeply integrated into all four core components: perception (e.g., sensor and data analytics tools), reasoning (e.g., knowledge base and retriever tools), action (tool sets), and memory (e.g., vector databases). We can observe that Agent = LLM + Tools. (right): The effects of tool intelligence in communications.

real-world computation or actuation. A modern agentic AI framework typically consists of four core components, namely perception, reasoning & planning, action, and memory & evolution, thereby implementing a continuous cycle of autonomous operation [7], [8]. As illustrated in Fig. 1(left), tool intelligence plays a critical role in all components.

- **Perception:** Perception refers to the agent's ability to receive and interpret environmental states. In communication networks, this component involves processing diverse data modalities. Hence, sensors and data analytics software tools are called to acquire and parse raw environmental data, creating the situational awareness upon which all decisions are based.
- **Reasoning & Planning:** This is the cognitive core of an agent, often driven by LLMs, which analyzes perceived information and formulates a multi-step plan. Tool intelligence is critical in this component. The agent may invoke a knowledge base retriever to fetch 3GPP standards or leverage knowledge graphs to improve reasoning capabilities, leading to a more factually grounded plan.
- **Action:** Based on the generated plan, the agent executes actions by interacting with its environment. This component is the most explicit application of tools, which act as the agent's actuators. For a communication agent, this means calling a tool, such as a resource allocation algorithm, a Python script to reconfigure network slices, or a *gNMI* API[4] to reconfigure a router.
- **Memory & Evolution:** This refers to the agent's capacity to learn and adapt over time. Tools are the mechanisms that enable this process, allowing the agent to read from and write to external memory modules, such as vector databases or files. This self-reflection mechanism allows it to refine its capabilities, leading to continuous improvement in dynamic communication networks.

C. Effects of Tool-enhanced Agentic AI in Communications

1) Expanding Communication Functionalities: An agent's LLM core can be fused with a growing suite of specialized tools [4]. This is particularly transformative for the communications domain, which relies on a vast library of professional knowledge and complex mathematical calculations (e.g., channel models in 3GPP or beamforming simulations). External tools directly compensate for LLMs' inherently limited domain-specific knowledge and mathematical reasoning capabilities [7]. Furthermore, a single agent can achieve broad generalization: its capabilities are no longer defined by its LLM's internal training, but by the diverse set of tools it can orchestrate (see Fig. 1(right)).

2) Improving Communication Efficiency: Compared with traditional single-purpose methods, agents achieve superior efficiency in communication management through intelligent tool chain orchestration. As shown in Fig. 1(right), an agent can autonomously execute the entire spectrum management lifecycle. This involves orchestrating a tool chain that invokes a traffic prediction tool to forecast network-layer demand. In parallel, it activates a spectrum sensing tool to identify the current physical-layer supply. The agent's LLM core then fuses these statistics and activates an optimization tool to calculate the optimal allocation policy. By unifying these traditionally separate tools, the agent minimizes overhead and the potential for cumulative errors.

Furthermore, tools facilitate the realization of collective intelligence, where multiple agents, each possessing distinct and specialized tool sets, collaborate to manage communications. As illustrated in Fig. 1(right), a multi-agent spectrum management system comprises three collaborative agents with different roles and tool sets. Upon detecting an emergency, a public safety agent initiates a negotiation. A regulatory agent then invokes its policy-checking tool to validate the request against established policies. This, in turn, enables a commercial agent to use its optimization tool to calculate an optimal block of spectrum while minimizing its own service disruption. This tool-driven collaboration among specialized agents results in a context-aware reallocation that is far more resilient than monolithic, pre-programmed systems.

[4]Available at: https://github.com/openconfig/gnmi

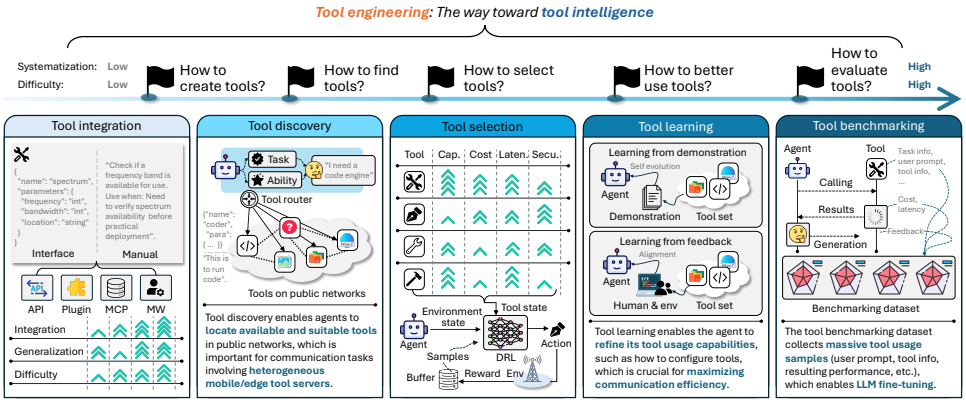

Fig. 2. The major aspects of tool engineering. The complexity and systematization increase from left to right, addressing more and more advanced questions. In this way, a complete ecosystem of tool-augmented agentic AI for communications can be implemented.

III. Tool Engineering: Realizing Tool Intelligence in Communications

A. Tool Creation and Integration

To create a tool for LLM-based agents, two components are essential, i.e., a well-defined interface and a comprehensive natural language description. As illustrated in Fig. 2, the interface provides a structured way for the agent to call the tool with specific parameters and receive the output. Equally important is the natural language description, which serves as the "manual" for the agent's LLM. This description must clearly articulate the tool's purpose, its required input parameters (including their data types and formats), and the structure of its output. For example, a beamforming tool might be described as "Function `calculate_beamforming_weights(location, channel_state)`". By parsing these descriptions, the agent can understand what tools are available in its arsenal and how to call them correctly [4].

Multiple integration paradigms have emerged. As illustrated in Fig. 2, direct API integration is the most fundamental, where developers expose REST, gRPC, or Python-callable functions and provide schemas manually to the LLM. Similarly, plugins wrap APIs inside a platform-specific manifest (e.g., ChatGPT or Gemini). Although widely adopted, these approaches can be rigid and only suitable for offline usage. In contrast, the model context protocol (MCP) [9] generalizes tool integration by defining a model-agnostic protocol. The tools are registered on MCP servers and automatically declare their capabilities in a standardized schema. MCP-compatible agents can dynamically discover and invoke tools over the networks, thereby reducing the integration costs while embedding audit controls. Finally, middleware, such as LangChain and LangGraph[5], acts as an orchestration layer that operates APIs, plugins, or MCP endpoints, providing higher-level functionality such as multi-tool reasoning and error handling.

B. Tool Discovery in Open Networks

6G and future communications are envisioned as open and distributed ecosystems where new services, functions, and corresponding tools may become available on the fly. An agent operating in such an environment cannot rely solely on a fixed, pre-configured toolkit. This necessitates a mechanism for automatic tool discovery, allowing agents to dynamically identify and incorporate new tools as they emerge in the network. Recent proposals like MCP-zero [9] aim to address this challenge by creating a decentralized framework for tool registration and discovery. Specifically, network functions or services can broadcast their availability and expose their callable tools via a standardized protocol. An AI agent can then query this distributed registry to find tools that match its current task requirements. For instance, an agent tasked with optimizing a new low-latency service could discover a nearby specialized DRL optimization tool.

C. Tool Selection

1) Static Binding & Simple Tool Selection: The most basic form of tool selection involves static binding or simple keyword matching. In this approach, each tool is hard-coded to a particular type of task, or the agent selects a tool based on a rudimentary analysis of the query's keywords. For example, any user prompt containing the words "channel quality" might automatically trigger a tool that queries the network's CSI database. Although simple and predictable, this approach is highly inflexible. It fails to account for the nuances of user intent or task context, and it cannot handle situations where multiple tools are required to address one problem.

2) Dynamic & Context-aware Tool Selection: To perform dynamic tool selections, the agent analyzes the full context of the task, including the specific user intent, interaction history, and the current network state. Concurrently, it checks its available toolset, parsing the functionality and description of each tool. Crucially, for a communication agent, this analysis also incorporates practical wireless indicators, such as transmission latency, channel condition, and resource consumption. This allows the agent to formulate a complicated objective function for tool selection with a comprehensive set of KPIs. The agent can then leverage LLM's analytical ability to reason about these trade-offs and select the most suitable tool [10]. Alternatively, as the modeling is highly complex, DRL is also typically used to train a sophisticated selection policy.

3) Tool Chain & Orchestration: Many complex communication tasks cannot be solved with a single tool and thus require a sequence of actions, called tool chains. The agent

[5]Available at: https://docs.langchain.com/

can act as an intelligent planner, decomposing a high-level goal into a multi-step workflow and orchestrating a chain of tool invocations [11]. This can range from a simple diagnostic workflow (e.g., monitoring → inspection tool → configuration) to complex, hierarchical multi-agent systems where a central planning agent delegates tasks to specialized sub-agents with their own tools [11].

D. Tool Learning

The intelligence of tool usage is not just calling predefined tools but learning to use them better over time, a concept known as tool learning. This involves the agent refining its ability to select and operate tools based on experience and feedback. Through a trial-and-error process, often guided by DRL [1], an agent can learn the subtle nuances of a tool's behavior. For instance, it might learn that a particular optimization algorithm performs best under certain channel conditions or that a diagnostic tool is unreliable during peak network load. Moreover, this learning process can also be self-driven. For example, SPORT [12] proposes an iterative framework, where agents optimize tool usage autonomously through step-wise preference tuning. After executing a tool chain, the agent can observe the network's response to assess the outcome. If the result was suboptimal, the agent can store this experience in its memory and adjust its planning process to avoid making the same mistake in the future [12]. This capability is vital in dynamic communication networks, allowing an agent to continuously refine its strategies.

E. Tool-Usage Standard, Protocols, and Benchmarking

Standardization is paramount for scaling the tool ecosystems of communication agents. Although no single universal standard exists yet, the ecosystem is coalescing around several key approaches, such as OpenAI's API and MCP. For tool assessment, early benchmarks, e.g., API-Bank [13] and ToolBench[6] focus on evaluating an agent's ability to handle a massive number of diverse APIs. More recent benchmarks provide deeper, more nuanced evaluations by focusing on specific real-world challenges. For example, ToolQA tests an agent's ability to answer questions that can only be solved by using tools to query external knowledge, thereby minimizing the influence of the LLM's internal memory. Concurrently, Tool Playground [6] evaluates an agent's ability to handle tool failures or correct invalid parameters. Moreover, General Tool Agents (GTA) [14] construct a real-world benchmarking dataset, containing numerous multimodal human-written queries. Similarly, TOUCAN [15] contains 1.5 million trajectories synthesized from nearly 500 real-world MCP servers. It can be used to evaluate agents on complex tasks, such as BFCL V3 and MCP-Universe [15]. Nonetheless, the communication-specific tool benchmarking dataset remains unsolved.

IV. Case Study

A. System Model

1) System Overview: UAVs in next-generation communications are evolving from simple data relays to intelligent agents capable of orchestrating various devices. As shown in Fig. 3, we consider that a UAV acting as a flying agent, equipped with a lightweight on-board LLM, is tasked with navigating from a starting point to a destination. Due to environmental complexities (e.g., fog and signal jamming) and limited sensing range, the UAV lacks accurate global situational awareness and suffers from navigational uncertainty.

2) Tool Settings: To overcome these limitations, the UAV should leverage external capabilities. We consider two categories of tools deployed on ground servers, representing two different operating points: high-fidelity information acquisition and lightweight semantic interaction. In particular, the semantic tool is included to reflect the emerging semantic communication paradigm, where task-relevant meaning is exchanged instead of high-volume raw data.

- **Standard Tool:** The UAV streams raw, high-volume sensor data (e.g., video feeds) to the server. The server processes this input to reconstruct a high-fidelity 3D map for navigation. Such functionalities can be built using *DroneDeploy*[7] or *Google Earth Engine*[8] APIs.
- **Semantic Tool:** The UAV sends lightweight text queries (e.g., coordinates) to the server. The server calls *OpenStreetMap Overpass*[9] APIs and transmits compact semantic metadata (e.g., environmental attributes such as building height and obstacles). Crucially, the UAV employs its onboard LLM to reason over these textual descriptors to reconstruct the navigation path locally.

These APIs are encapsulated by MCP [9] to form standardized agentic tools. We suppose that the UAV employs the active tool discovery mechanism described in Section III-C, which can activate available tools within a certain range. To facilitate further research, we open-source a tutorial on wrapping these third-party APIs into MCP-compliant tools[10].

B. Problem Formulation

We consider a tool-assisted trajectory planning problem. The decision variables at each step include the UAV's flight velocity vector and a binary tool activation signal. Specifically, the incorporation of tool intelligence reshapes the optimization formulation, necessitating the consideration of the following critical characteristics (see Fig. 3(right)).

1) Functional Augmentation: Tool execution helps the UAV eliminate navigation uncertainty. This functional augmentation exerts a long-term temporal influence: a single tool activation can effectively reshape the trajectory evolution over multiple subsequent time steps, rather than merely delivering an instantaneous reward in conventional offloading.

2) Tool Necessity: Tool execution incurs resource overhead. The UAV consumes significant energy to transmit data (for standard tools) or perform LLM inferences (for semantic tools), creating a coupling between communication bandwidth and computational resources. Therefore, the agent must learn to refrain from tool invocation when the tools are far away or

[6]Available at: https://github.com/OpenBMB/ToolBench

[7]https://www.dronedeploy.com/

[8]https://earthengine.google.com/

[9]https://github.com/hrbrmstr/overpass

[10]https://github.com/Lancelot1998/Agentic_AI_Tool_Magazine

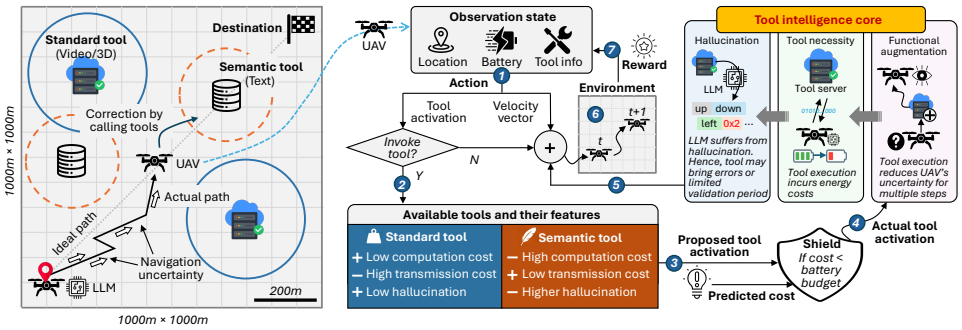

Fig. 3. (left): The case study scenario. (right): The procedure of the proposed algorithm. ❶: The action space; ❷: The illustrations of heterogeneous tools; ❸: The proposed tool activation; ❹: The proposed shield for training; ❺: The three characteristics of tool intelligence that effects problem formulation; ❻: The UAV motion; ❼: The resulting reward.

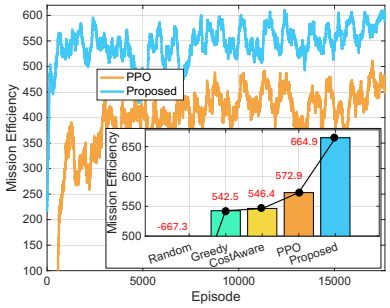

Fig. 4. The training curves of vanilla PPO and the proposed algorithm, and the mission efficiency of five methods.

the remaining energy is critically constrained, thus ensuring mission survival under strict resource budgets.

3) Hallucination: The primary role of tools is to support the LLM's reasoning with external information. However, since LLMs function via probabilistic token generation rather than deterministic logic, they introduce an intrinsic risk of hallucination, i.e., generating factually wrong or irrelevant outputs. This risk is inversely correlated with the information richness provided by the tool. Standard tools that transmit uncompressed data provide richer information, resulting in low hallucination rates. In contrast, semantic tools provide limited or abstract information, significantly increasing the probability of generating factually incorrect guidance.

The functional augmentation and the associated resource overhead of tools construct a fundamental trade-off. Moreover, hallucination imposes a limited validity horizon on the acquired guidance. The agent may need to perform repeated tool activations to maintain navigational accuracy throughout the flight. Therefore, we maximize mission efficiency, which entails jointly minimizing total flight time and energy consumption (including flight propulsion, wireless data transmission, and LLM computation onboard). Accordingly, the optimization is subject to a constraint: the UAV's energy must never be depleted before mission completion.

C. Teacher-Guided Reinforcement Learning

We use proximal policy optimization (PPO) [1] to train the agent. Moreover, to help agents efficiently operate tools while flying, we introduce a teacher shield during training. The shield explicitly predicts the energy cost of a proposed tool call by calculating the underlying communication and computation expenditure (see Fig. 3)[11]. If the predicted cost exceeds the budget, the shield overrides the agent's action with a safe alternative, i.e., continuing to fly. Moreover, it shapes the training process by applying a small penalty to the reward. This mechanism transforms sparse delayed failure signals, such as crashing after many steps, into immediate and dense feedback. The agent learns to associate low-battery states with the negative consequences of activating tools and internalizes the underlying physical constraints. Consequently, the trained agent only activates tools when necessary.

D. Experimental Results

1) Experimental Settings: We simulate a service area and randomly deploy four MCP servers (two with standard tools and two with semantic tools). To simulate the aforementioned navigational uncertainty, we inject continuous Gaussian noise into the UAV's motion dynamics at each time step. Without external correction, this noise accumulates, leading to significant trajectory drift. The activations of standard/semantic tools serve as discrete state corrections of the UAV's MDP process that eliminate the deviation. Crucially, to capture the impact of hallucination, we set different performances for different tools. Specifically, standard tools can provide long-term stable navigation. In contrast, reflecting the higher hallucination rate, semantic tools are restricted to a shorter validity horizon.

We compare our proposed teacher-guided PPO against four distinct baselines: 1) Vanilla PPO; 2) Random Policy; 3) Greedy Policy, where the UAV navigates toward the destination and attempts to activate any accessible tool; and 4) Cost-Aware Heuristic, where the UAV navigates directly but only activates tools if the remaining energy is enough.

2) Performance Analysis: Here, we analyze the performance of the proposed algorithms in tool-assisted trajectory planning from three perspectives.

Mission Efficiency: As illustrated in Fig. 4, the training curves demonstrate that the proposed method significantly outperforms all baselines. The random policy fails to complete missions effectively, producing the worst performance. The greedy policy suffers from frequent energy depletion caused by reckless tool activations, resulting in a suboptimal reward of 542.5. The cost-aware heuristic performs competitively by avoiding fatal crashes. Finally, the proposed teacher-guided

[11]For simplicity, the communication and computation costs of each tool are assumed to be known through offline profiling or service specifications.

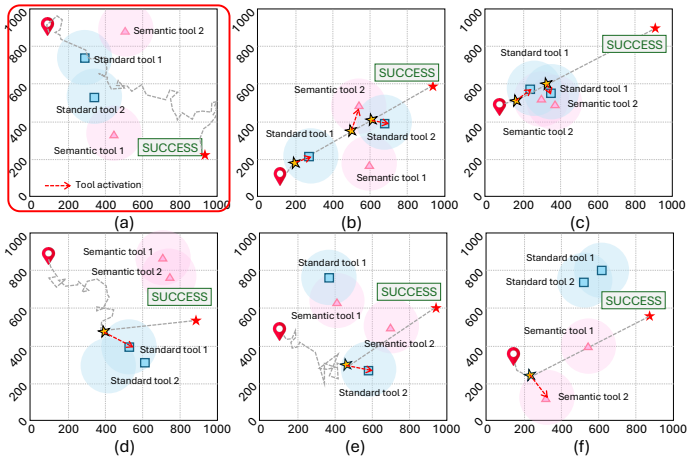

Fig. 5. The illustrations of UAV trajectories and tool activations. (a): The trajectory of the UAV without tools. (b)-(f): The trajectories and tool activations of the UAV trained by the proposed algorithm.

PPO achieves rapid convergence and the highest stable reward of 664.9, validating its ability to dynamically balance functional augmentation and energy costs of tools.

Tool Activation Accuracy: Figs. 5(b)-(f) reveal that the agent evolves context-aware trajectory planning and tool activation behaviors compared with the case without tools (Fig. 5(a)). As observed in Figs. 5(d) and (e), the agent proactively deviates from its course to enter MCP coverage, specifically when navigational uncertainty accumulates. Crucially, the agent further optimizes its spatial entry strategy according to the specific physics of the tool type. When targeting a standard tool (e.g., standard tool 1 in Fig. 5(b)), the UAV flies deep into the cell center, learning that calling standard tools costs distance-dependent transmission power. Conversely, for a semantic tool (e.g., semantic tool 2 in Fig. 5(b)), the UAV adopts an edge skimming strategy. Since semantic transmission costs are distance-independent, this strategy minimizes flight energy consumption while still securing the necessary guidance.

Tool Necessity: Finally, we evaluate whether the agent achieves the cognitive ability to refrain from tool activations when they are redundant or unsafe. Regarding redundant invocations, Figs. 5(c) and (f) illustrate that the agent correctly minimizes unnecessary tool usage. Even when encountering overlapping tool coverage or passing through a tool's range while already on a low-uncertainty trajectory, the agent refrains from activation. This demonstrates a learned judgment that the marginal utility of further guidance is negligible compared to the activation cost.

V. Future Directions

Secure Tool Executions: As agents dynamically discover and invoke third-party tools in open 6G networks, ensuring operational safety, security, and privacy becomes paramount. Future research should establish zero-trust tool execution environments, secure sandboxing, and audit mechanisms for tool invocation. In addition, privacy-preserving mechanisms are needed to protect sensitive network and user data from unauthorized exposure during tool interaction.

Semantic and Latent Tool Interfaces: To meet the stringent latency requirements of beyond 6G communications, tool engineering must evolve beyond current text-based interfaces, which incur significant computational overhead. Semantic and latent tool interfaces can be explored, where agents bypass high-dimensional natural language to interact directly through compressed semantics or abstracted latent representations.

VI. Conclusion

In this article, we have explored the transformative potential of tool intelligence in empowering agentic AI for next-generation communication networks. First, we have introduced a comprehensive landscape for tool engineering, detailing the essential lifecycle from tool creation and discovery to selection, learning, and benchmarking. Moreover, we have conducted a case study about tool-assisted UAV trajectory planning, optimizing the agent's ability to activate tools while strictly adhering to the strict energy constraints.

References

[1] J. Shao *et al.*, "WirelessLLM: Empowering large language models towards wireless intelligence," *Journal of Communications and Information Networks*, vol. 9, no. 2, pp. 99–112, 2024.
[2] B. Liu *et al.*, "LLM4CP: Adapting large language models for channel prediction," *Journal of Communications and Information Networks*, vol. 9, no. 2, pp. 113–125, 2024.
[3] W. Tong *et al.*, "A-core: A novel framework of agentic ai in the 6G core network," in *ICC Workshops*, 2025, pp. 1104–1109.
[4] Y. Qin *et al.*, "Facilitating large language models to master 16000+ real-world APIs," in *ICLR*, 2024, pp. 1–9.
[5] C. Qu *et al.*, "Tool learning with large language models: a survey," *Frontiers of Computer Science*, vol. 19, no. 8, pp. 1–21, 2025.
[6] Z. Dong *et al.*, "Tool Playgrounds: A comprehensive and analyzable benchmark for LLM tool invocation," in *ICASSP 2025*, 2025, pp. 1–5.
[7] F. Jiang *et al.*, "From large AI models to agentic AI: A tutorial on future intelligent communications," *arXiv preprint arXiv:2505.22311*, 2025.
[8] J. Tong *et al.*, "Wirelessagent: Large language model agents for intelligent wireless networks," *arXiv preprint arXiv:2505.01074*, 2025.
[9] X. Fei, X. Zheng, and H. Feng, "MCP-Zero: Active tool discovery for autonomous LLM agents," *arXiv preprint arXiv:2506.01056*, 2025.
[10] M. Fore, S. Singh, and D. Stamoulis, "GeckOpt: LLM system efficiency via intent-based tool selection," in *GLSVLSI*, 2024, p. 353–354.
[11] W. Wang *et al.*, "AgentOrchestra: Orchestrating hierarchical multi-agent intelligence with the tool-environment-agent (TEA) protocol," *arXiv preprint arXiv:2506.12508*, 2025.
[12] P. Li *et al.*, "Iterative tool usage exploration for multimodal agents via step-wise preference tuning," in *NeurIPS 2025*, 2025.
[13] M. Li *et al.*, "API-bank: A comprehensive benchmark for tool-augmented LLMs," in *EMNLP*, 2023, p. 3102–3116.
[14] J. Wang *et al.*, "GTA: A benchmark for general tool agents," in *NeurIPS 2024 Dataset and Benchmark Track*, 2024.
[15] Z. Xu *et al.*, "TOUCAN: Synthesizing 1.5M tool-agentic data from real-world MCP environments," *arXiv preprint arXiv:2510.01179*, 2025.