Proceedings of the ASME 2026
International Symposium on Flexible Automation
ISFA2026
July 19–22, 2026, Okayama, Japan

ISFA2026-8921

OPERATIONALIZING FOUNDATION AGENTS IN MANUFACTURING VIA PERSISTENT WORKFLOW MEMORY

Danny Hoang[1], David Gorsich[2], Matthew P. Castanier[2], and Farhad Imani[1]*

[1]School of Mechanical, Aerospace, and Manufacturing Engineering, University of Connecticut, Storrs CT
[2]US Army DEVCOM Ground Vehicle Systems Center, Warren, Michigan, USA

ABSTRACT

Large language models (LLMs) have the potential to serve as high-level orchestration layers for manufacturing decision-making by composing deterministic software tools over measurements, simulations, and process knowledge; however, this capability remains unreliable. Deployed agents frequently misjudge when computation is required, confuse near-duplicate tools, and generate brittle multi-step plans that fail under noisy or underspecified inputs. We introduce persistent tool memory (PTM), a validation-gated, retrieval-augmented memory that captures successful tool-use episodes as reusable workflow priors. For each new query, PTM retrieves the top-K most similar validated episodes and distills them into a constrained suggestion set comprising candidate tools, execution orderings, and mandatory consistency checks. A routing agent may follow, adapt, or reject these priors under safety constraints. PTM is embedded in a tool-grounded multi-agent architecture with critic-driven verification of argument validity, dependency structure, and unit consistency. We evaluate PTM on a rotor-blade machining benchmark of 300 natural-language queries requiring multi-step interaction with deterministic machining-analysis tools. PTM significantly improves tool-use recall and first-pass planning accuracy, with the largest gains occurring in deeper workflows where baseline agents systematically derail. Moreover, PTM reduces model-size dependence; a 7B-parameter model equipped with PTM approaches the performance of 30B+ models by grounding planning decisions in previously validated execution traces.

Keywords: Large Language Models; Manufacturing Copilots; Workflow Reuse; Retrieval-Augmented Generation; Quality Assurance

*Corresponding author: farhad.imani@uconn.edu

1. INTRODUCTION

Manufacturing planning, diagnosis, and corrective action are executed through deterministic tool chains constrained by physics, machine limits, and procedure. In practice, engineers translate inspection measurements, simulation outputs, controller logs, and process plans into a sequence of validated analyses, then commit bounded decisions such as compensation offsets, parameter updates, or root-cause hypotheses. This paradigm is physically grounded and auditable, but it scales poorly [1]. Workflows are brittle to context changes, procedural expertise remains tacit, and adapting pipelines to new parts or sensing conditions requires repeated human effort.

LLM-based agents offer a different interface to tool chains. Rather than hard-coding workflows as fixed scripts, an agent can interpret natural-language intent, reconcile heterogeneous context, and dynamically compose tool interactions at runtime. Recent work in manufacturing demonstrates the promise of LLMs as orchestration layers that bridge enterprise knowledge, inspection data, simulation software, and planning tools, reducing the burden of specifying and maintaining analysis pipelines [2, 3].

However, manufacturing exposes a sharp reliability gap between probabilistic generation and deterministic execution [4]. Tool use offloads numerical computation, but it does not remove the need for discrete control decisions that are highly error-sensitive; deciding whether tool invocation is necessary, selecting the correct tool among overlapping capabilities, binding arguments with valid units and ranges, and composing multi-step traces whose intermediate artifacts remain consistent and auditable. These decisions are discontinuous and weakly aligned with the smooth inductive biases of autoregressive generation. Consequently, small mistakes compound such that an early tool mismatch or unit error can propagate through downstream computations and yield plausible text that is procedurally invalid [5].

State-of-the-art tool-learning methods improve executable tool calling via self-supervision, retrieval over tool documentation, and execution feedback [6–9]. Yet benchmarks continue to report persistent failure modes directly relevant to manufac-

turing; incorrect decisions about whether to use tools, confusion among near-duplicate tools, and sharp degradation under noise or interface drift [10, 11]. In chained manufacturing pipelines, such errors accumulate across steps, undermining first-pass success and eroding trust.

We argue that a central systems-level cause is stateless planning. Most tool-grounded agents re-synthesize tool choice and step order from scratch for each query, even when task structure recurs with minor parameter variation [1, 12]. Manufacturing quality loops, inspection coverage refinement, deviation attribution, parameter tuning, and corrective offset computation, repeatedly execute the same procedural skeleton across parts, batches, and operating conditions. Treating each instance as an independent stochastic planning problem discards validated procedural structure and increases variance in first-pass behavior.

This paper introduces persistent tool memory (PTM), a validation-gated memory mechanism that converts past successful tool-use episodes into reusable workflow priors. PTM stores structured episodes, context summaries, ordered tool traces with arguments, accepted outputs, and explicit validation signals, and retrieves similar episodes at inference time. Retrieved episodes are compiled into constrained suggestions for tool selection and execution patterns; similarity thresholds and execution-time verification prevent unsafe or irrelevant reuse. This paper makes three contributions:

- **PTM: Validation-Gated Procedural Memory.** We formulate *persistent tool memory* as episodic workflow memory that retrieves validated tool-use traces and reuses them as priors to reduce variance in discrete decisions: tool selection, argument binding, and dependency order.
- **Retrieval-to-Plan Compilation with Safety Guards.** We implement PTM as a governed episode store and retriever that converts the top-K nearest neighbors into a constrained suggestion set (candidate tools + step patterns), enforced by tool-name hygiene, similarity thresholds, and critic-driven checks (schema, units, ordering) before execution.
- **Rotor-Blade CNC Evaluation for Tool Reliability.** We evaluate PTM on a rotor-blade machining benchmark that stresses near-duplicate tools and multi-step deterministic execution, reporting tool-use reliability and first-pass planning accuracy stratified by difficulty and required tool depth.

2. RESEARCH BACKGROUND

Manufacturing decision support is a cyber-physical problem in which reasoning errors can induce irreversible physical consequences. Corrective actions are not inferred from static knowledge alone, but are produced by executing ordered computations over measured and simulated artifacts and then mapping validated outputs to bounded decisions [13]. In inspection-driven quality loops, data acquisition and sensing parameters are tuned based on outcomes, reconstruction and comparison tools are executed, and deviations are localized before any physical intervention is authorized [14]. Units, ranges, and dependency order encode safety margins, machine limits, and process feasibility. When an agent selects an incorrect tool, binds an argument with an invalid scale, or breaks dependency order, the result is not merely a wrong answer. It can be scrap, wasted machine time, violated tolerances, or unsafe actuation. Reliability in this setting is therefore defined by correct tool choice, valid argument binding, correct sequencing across multi-step executions, and auditable provenance that ties conclusions to deterministic tool outputs [15, 16].

Recent manufacturing research positions LLMs as orchestration layers that translate intent into interactions with enterprise knowledge and engineering tools, offering a path beyond rigid, hand-engineered pipelines [1, 2, 4]. This shift reflects the appeal of agentic systems that can compose workflows at runtime rather than requiring experts to hard-code every variation. Yet manufacturing also exposes a reliability requirement that is sharper than in many software-only domains. Tool-grounded assistants must operate under partial context, noisy inputs, evolving interfaces, and libraries with overlapping capabilities, while maintaining low variance across repeated executions. Under these conditions, the central question is not whether LLMs can call tools, but whether they can call the right tools with valid arguments and stable execution order under small contextual perturbations, since small perturbations are common in production.

The general tool-learning literature supplies the enabling techniques for such orchestration and simultaneously clarifies why manufacturing remains challenging. Recent approaches have moved from prompt-only tool invocation toward learned tool-use policies that incorporate self-supervision, retrieval over tool documentation, large-scale instruction tuning, and execution feedback [6, 7, 9, 17, 18]. This progression improves coverage and average tool-call capability, which is necessary for any realistic agent. However, improvements in average capability do not automatically yield the reliability profile required in cyber-physical pipelines. Manufacturing is sensitive to tail events where a single discrete mistake cascades. Benchmarks that isolate tool-use decisions show that core failure modes remain persistent, including errors in deciding whether computation is required, errors in disambiguating near-duplicate tools, and sharp degradation under noise or distribution shift in tool interfaces and arguments [5, 10, 11]. These are precisely the failure modes that translate into compounded downstream error when tool outputs feed subsequent tools and when final outputs inform physical action.

A unifying systems-level explanation is that many tool-grounded agents operate without persistent procedural state. Planning decisions are synthesized independently for each query, even when task structure recurs, which introduces avoidable variance in tool choice and ordering. In domains characterized by repeated workflows with bounded variation, such variance becomes a reliability bottleneck rather than a benign property of stochastic generation. Manufacturing diagnosis and quality assurance frequently revisit the same analysis patterns across parts, batches, and operating conditions, differing primarily in parameters rather than structure. When each instance is treated as an independent planning problem, previously validated execution structure is discarded, and the agent is forced to repeatedly solve the same discrete tool-composition problem under stochastic uncertainty, increasing the probability of first-pass failure.

This motivates a taxonomy of memory than is typically used in manufacturing copilots. Semantic memory retrieves facts, documents, or descriptive context, and it is well aligned with retrieval-augmented generation. It addresses missing informa-

tion and improves citation grounding, but it does not directly constrain procedural correctness because factual retrieval does not specify tool order, argument binding, or validated step patterns. Episodic memory stores structured traces of how tasks were successfully executed, including action sequences, tool arguments, and validated intermediate artifacts. Recent agent research argues that long-horizon reliability requires such procedural memory and demonstrates that retrieving reusable workflows can reduce planning variance and improve success on complex tasks [1, 12]. Manufacturing assistants predominantly rely on semantic retrieval even though many failures arise from procedural errors rather than missing facts. This gap motivates memory mechanisms that store validated tool-use episodes as reusable procedures and retrieve them, so that reuse improves first-pass tool correctness while preserving safety and auditability.

3. METHODOLOGY

Figure 1 provides an end-to-end overview of PTM, including the multi-agent control flow, the persistent episode store, and the Meta Agent loop that retrieves, validates, and applies workflow priors during execution. We formalize interaction with a tool-grounded agentic system as a sequence of user turns indexed by $t \in \{1, 2, \dots\}$. At turn t, the user provides a natural-language query q_t, and the system maintains an internal state s_t containing references to loaded data, intermediate artifacts, and prior tool calls. Numerical results and manufacturing decisions are produced by deterministic tools, while an LLM-mediated controller decomposes tasks, selects tools, and generates explanations.

Our baseline consists of four agents: (1) a Central Agent (CA) that routes instructions to downstream components, (2) an Analysis Agent that invokes deterministic manufacturing-analysis tools, (3) a Knowledge Agent that retrieves structured domain knowledge when needed, and (4) a Meta Agent that verifies tool-use correctness and triggers repair. The Central Agent implements a routing policy

$$(a_t, \tilde{q}_t) = \pi_{\mathrm{CA}}(q_t, s_t), \tag{1}$$

where a_t selects the downstream component and $\tilde{q}_t$ is a structured instruction. Downstream execution produces artifacts and tool outputs o_t, and the system state is updated via a (deterministic) transition operator

$$s_{t+1} = T(s_t, a_t, \tilde{q}_t, o_t). \tag{2}$$

In the baseline, the CA must re-derive tool choice and multi-step ordering from scratch at each turn, which can yield inconsistent tool selection and brittle sequences under noisy inputs or underspecified contexts.

PTM addresses this limitation by introducing a persistent retrieval-augmented episode database and assigning responsibility for memory execution to the Meta Agent as shown in Figure 2. PTM stores curated and validated tool-use episodes from prior interactions as reusable workflow priors. Each stored episode is

$$m_i = (x_i, \tau_i, y_i, v_i, c_i, \mathrm{id}_i), \tag{3}$$

where x_i is a compact summary of the question and context (including relevant state features), τ_i is the executed tool trace, y_i is the accepted final response, v_i is a validation signal (critic acceptance, tool-level sanity checks, and optional operator approval), c_i are lightweight tags (e.g., task category or tool family), and id_i is a stable identifier used for uniqueness and deduplication (e.g., a question/part-family signature). Episodes are committed only after completion and validation, and persist across sessions so that successful workflows become reusable operational assets rather than transient chat history.

We represent the tool trace as a finite sequence of tool calls: $\tau_i = (c_{i,1}, \dots, c_{i,L_i})$, $c_{i,\ell} = \big(\mathrm{tool}_{i,\ell}, \mathrm{args}_{i,\ell}, \mathrm{out}_{i,\ell}, \mathrm{dep}_{i,\ell}\big)$, where $\mathrm{dep}_{i,\ell}$ encodes dependency links to prior calls (e.g., which intermediate artifacts are consumed), enabling explicit verification of dependency order and trace provenance. We also define the (unordered) set of tools used in the trace as $T_i = \mathrm{Tools}(\tau_i)$.

The episode database is backed by a persistent vector store that supports similarity search over historical workflows. Let $M = \{m_i\}_{i=1}^{|M|}$ denote the stored episode set. For each episode, PTM computes an embedding over a combined representation that includes the episode summary x_i, tool metadata derived from the trace (e.g., T_i), and lightweight tags c_i. In addition to the embedding index, PTM maintains auxiliary indexes by category and by tool to support filtered retrieval (e.g., restricting candidates to episodes that invoke required inspection or compensation tools). Entry uniqueness is enforced via id_i, and near-duplicate episodes may be deduplicated during indexing to reduce redundancy and improve prompt efficiency.

At runtime, the Meta Agent executes PTM and serves as the workflow memory controller. Given (q_t, s_t), it constructs a retrieval context

$$r_t = \phi(q_t, s_t), \tag{4}$$

embeds it as $\mathbf{z}_t = f(r_t) \in \mathbb{R}^d$, and retrieves similar episodes by nearest-neighbor search in embedding space. Each stored episode m_i is rendered for retrieval as $r_i = g(m_i)$ and embedded as $\mathbf{z}_i = f(r_i)$. We use cosine similarity

$$\mathrm{sim}(\mathbf{z}_t, \mathbf{z}_i) = \frac{\mathbf{z}_t^\top \mathbf{z}_i}{\|\mathbf{z}_t\| \, \|\mathbf{z}_i\|}, \tag{5}$$

Then we apply a minimum similarity threshold δ (for cosine similarity, $\delta \in [-1, 1]$). The retrieved neighbor set is defined by $I_t = \arg\mathrm{topK}_{i:\, m_i \in M,\ \mathrm{sim}(\mathbf{z}_t, \mathbf{z}_i) \geq \delta} \mathrm{sim}(\mathbf{z}_t, \mathbf{z}_i)$, $N_t = \{m_i : i \in I_t\}$, $u_t = \mathrm{Suggest}(N_t)$. The suggestion set u_t encodes candidate tools and step patterns (e.g., common call orderings and mandatory checks) distilled from the retrieved traces. Conditioned on u_t, the PTM-augmented routing policy becomes

$$(a_t, \tilde{q}_t) = \pi_{\mathrm{CA}}^{\mathrm{PTM}}(q_t, s_t, u_t), \tag{6}$$

where the CA may (i) follow a retrieved trace, (ii) adapt it by inserting or removing steps, or (iii) ignore it when similarity is low or constraints are violated.

Critically, the Meta Agent enforces validity constraints over any selected or adapted plan before allowing downstream tool execution, including argument schema checks, dependency-order checks, unit consistency, and plausibility/range checks. When violations are detected, the Meta Agent triggers bounded repair loops that revise tool choice, arguments, or step order while preserving safety and auditability. This places PTM in a supervisory

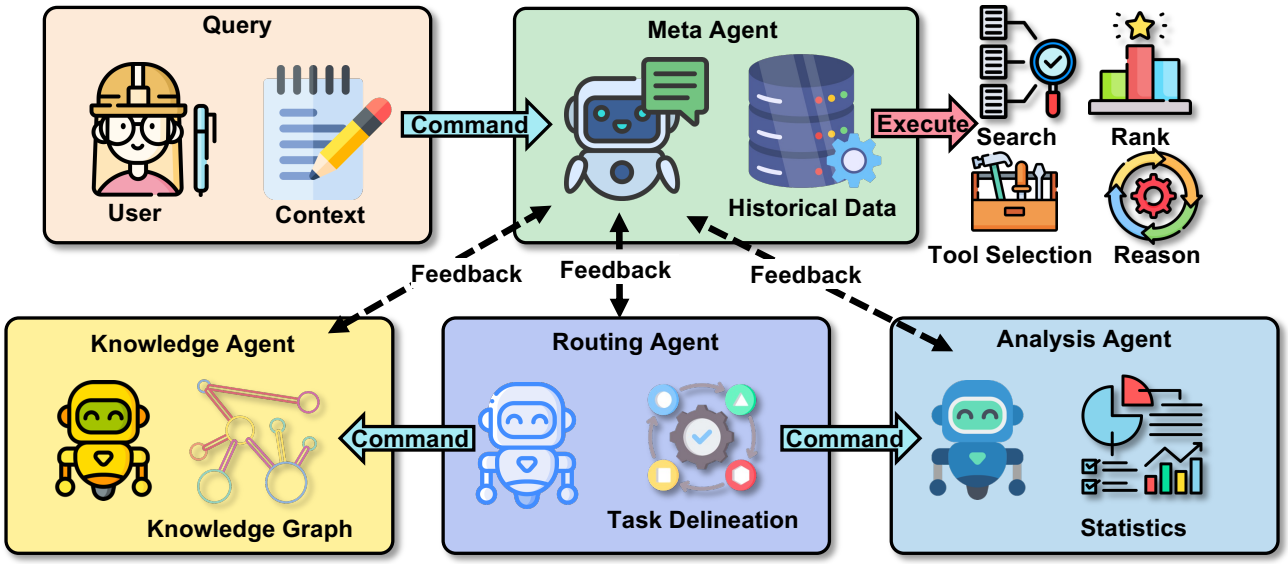

FIGURE 1: OVERVIEW OF THE PERSISTENT TOOL MEMORY (PTM) SYSTEM.

role; retrieved workflows inform planning, while the Meta Agent preserves correctness under explicit constraints.

Operationally, PTM is realized as a memory store together with a retriever invoked by the Meta Agent. The retriever performs (1) semantic search over episode embeddings, (2) optional filtering by category or required tools, and (3) trace summarization and deduplication before inserting retrieved guidance into the Central Agent prompt. To prevent prompt contamination, it enforces tool-name hygiene: tool names must match a strict pattern (alphanumeric with underscores/dashes) and fall within length bounds before inclusion. Retrieval applies a minimum similarity threshold and returns the top-K episodes; when deduplication is enabled, the retriever oversamples ($3K$) and retains the highest-similarity entry per stable identifier. Each selected episode is rendered as a compact, prompt-ready block containing the prior question/context summary, tags, tools used, and a condensed tool sequence (duplicate calls removed; only the first few arguments shown per step). Algorithm 1 summarizes this retrieval and formatting procedure.

After an interaction completes, the executed trace and output become eligible for storage. To reduce drift, PTM commits only episodes that satisfy validation criteria (critic acceptance, argument schema checks, unit/range checks, and optional operator confirmation). Episodes may be tagged (e.g., task category or tool family) to support filtered retrieval, and entry uniqueness is enforced through id_i to avoid near-duplicate memories.

Finally, we apply the same safety and robustness guards with and without PTM: structured outputs for routing and tool calls (e.g., JSON schemas), tool-level range checks (units and allowable offset bounds), bounded critic repair iterations, and deterministic fallbacks when router outputs are invalid. This ensures that measured differences isolate the contribution of persistent workflow memory rather than changes in execution safeguards.

Algorithm 1 PTM RETRIEVAL AND PROMPT FORMATTING EXECUTED BY THE META AGENT (TOOLMEMORYRETRIEVER)

Require: query q_t, state s_t, optional filters (category, required_tools), K, min_sim δ, max_steps $L_{\max}$
Ensure: formatted memory block for the Central Agent prompt

if $|M| = 0$ **then return** empty string
end if
$r_t \leftarrow \phi(q_t, s_t)$
results $\leftarrow$ SEARCH(r_t, top_k=$3K$, filters, min_similarity=δ)
results $\leftarrow$ DEDUPLICATEBYID(results, keep=K)
for each (m_i, sim_i) in results **do**
    $x_i \leftarrow$ SUMMARY(m_i)
    $\tau_i \leftarrow$ TRACE(m_i)
    $T_i \leftarrow$ TOOLS(τ_i)
    tools_used $\leftarrow$ CLEANTOOLNAMES(T_i)
    trace $\leftarrow$ SUMMARIZETRACE(τ_i, max_steps=$L_{\max}$)
    append formatted block: x_i, tags c_i, tools_used, trace, similarity sim_i
end for
return concatenated prompt section

4. EXPERIMENTAL DESIGN & RESULTS

We evaluate whether PTM improves tool decision making in a tool-grounded manufacturing assistant. Our central question is whether PTM enables the Meta Agent to more reliably select the correct tools (and avoid spurious tools) for a given query, particularly when instructions contain distractors or when the required tool chain is not explicitly specified. To isolate the contribution of PTM, we compare two system configurations that are identical in tools, routing interfaces, and validation budget, differing only in whether the Meta Agent is allowed to retrieve workflow priors from PTM.

We construct a benchmark of 300 natural language ques-

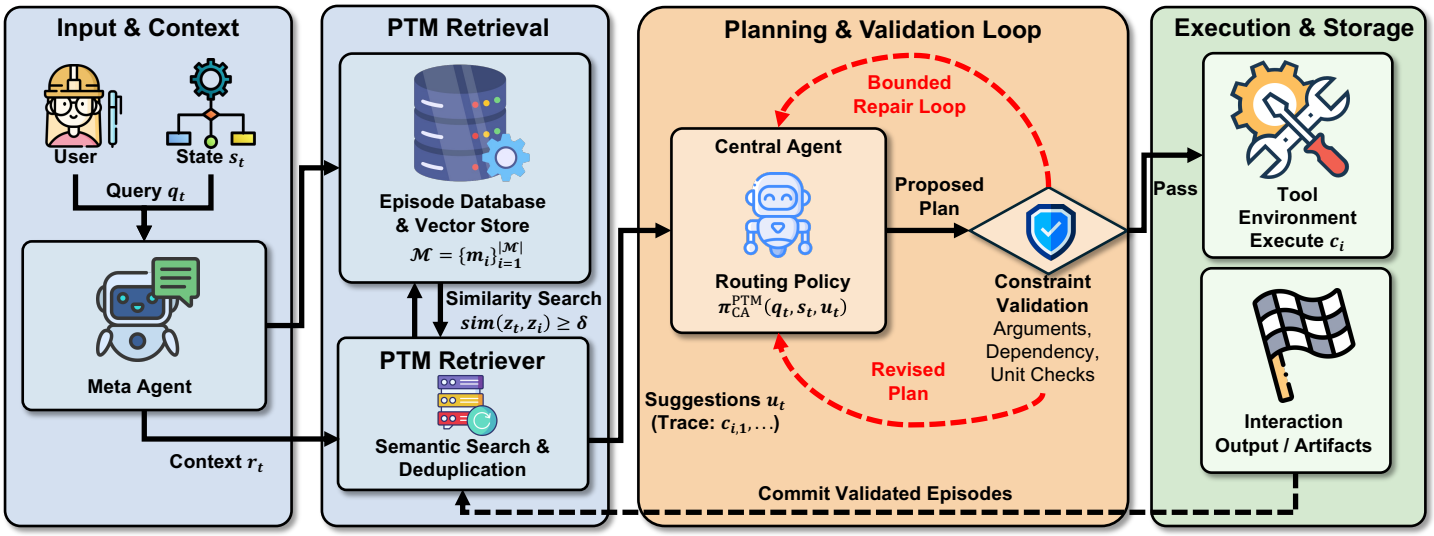

FIGURE 2: PTM SYSTEM WHERE MEMORIES ARE RETRIEVED FROM A PERSISTENT DATABASE TO INFORM PLANNING, WHILE A META AGENT-LED VALIDATION LOOP ENFORCES EXECUTION CONSTRAINTS AND MANAGES THE LIFECYCLE OF REUSABLE TOOL-USE EPISODES.

tions for a rotor-blade machining scenario using a strict equal-difficulty generator. The benchmark contains exactly 75 questions at each difficulty level (Easy/Medium/Hard/Expert) and spans four tool-requirement types: analysis-only questions that require deterministic machining-analysis tools, KG-only questions that require knowledge-graph retrieval, hybrid questions that require both analysis tools and KG retrieval, and three-turn memory episodes designed to require cross-turn state reuse. Each memory episode contributes one Hard turn (initialization) and two Expert turns (follow-on queries that depend on previously defined aliases/thresholds and intermediate outcomes), while preserving the global equal-difficulty constraint. Every question is annotated with a ground-truth set of required tools provided by the generator templates, and memory-episode turns include explicit dependencies and a requires memory flag indicating that correct tool selection depends on retrieving prior episode state rather than re-deriving it.

We evaluate two conditions where in the without memory (WoM) condition, the Meta Agent predicts the tool set using only the current query q_t and internal state s_t, without PTM retrieval. In the with memory (WM) condition, the Meta Agent queries PTM and obtains a retrieved suggestion set consisting of candidate tools and step patterns distilled from validated past episodes; this suggestion set is passed to the Central Agent as guidance for routing and tool planning, while all safety constraints and downstream execution interfaces remain unchanged. Both conditions use the same bounded critic-driven refinement budget for repairing invalid tool arguments, dependency violations, or inconsistent intermediate outputs, ensuring that observed differences reflect the presence or absence of persistent workflow memory rather than changes in validation policy.

For each benchmark item, we run the tool prediction pipeline under both conditions and compare the predicted tool set to the ground-truth required tools. To prevent trivial reuse, we apply a leave-one-out protocol at the question level where the evaluated question is excluded from the PTM index during its evaluation, and for multi-turn memory episodes we additionally ensure that later turns are not inserted into memory prior to evaluating those turns. Our primary metrics quantify tool decision quality via set-based precision, recall, and F1 between predicted and required tool sets, with an error decomposition into missing tools (required but not selected) and extra tools (selected but not required). We report results stratified by question difficulty level and overall results to showcase PTM's capability.

We implemented the PTM system in Python and evaluated it across three open source models spanning a wide range of capacity namely the Qwen3-VL [19] series of models specifically Qwen3-VL 4B, Qwen3-VL 8B, and Qwen3-VL 30B. Testing across these scales allows us to assess whether PTM yields consistent gains in tool decision making as model capability increases, and to demonstrate that the benefits of persistent workflow memory generalize beyond a single backbone model. Figure 3 showcases the F1 scores across all four question difficulties for each of the three models. Across all difficulties, PTM yields consistent improvements in tool-decision F1, with the largest uplifts occurring on Easy and Medium questions for the smaller models. For Qwen3-VL 4B, PTM improves F1 by 24.4 points on Easy and 14.3 points on Medium, with continued gains of 9.3 points on Hard and 6.9 points on Expert. For Qwen3-VL 8B, the corresponding uplifts are 36.5 points (Easy), 24.6 points (Medium), 14.4 points (Hard), and 4.2 points (Expert). For Qwen3-VL 30B, baseline performance is already high, and PTM produces smaller but uniformly positive gains across the difficulty spectrum (e.g., 8.5 points on Medium, 6.2 points on Hard, and 1.7 points on Expert). This pattern directly supports PTM's core premise where retrieving validated workflow priors reduces missing or incorrect tool selections and stabilizes tool choice under ambiguity and distractor noise. The fact that gains persist even for the biggest models suggests PTM acts as a complementary reliability layer rather than merely compensating for limited model capacity, improv-

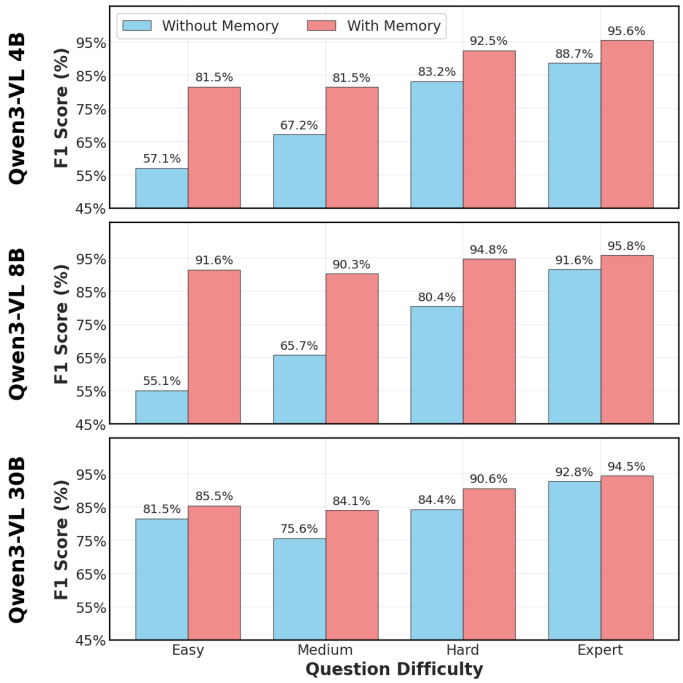

FIGURE 3: RESULTS OF THE PTM SYSTEM WITH MEMORY AND WITHOUT MEMORY ACROSS QUESTIONS DIFFICULTY FOR THREE DIFFERENT SIZED QWEN3-VL MODELS.

ing repeatability without modifying the underlying deterministic tools or safety constraints.

Figure 4 summarizes average tool-selection precision, recall, and F1 across all questions for each backbone. PTM increases recall for Qwen3-VL 4B from 89.2% to 96.7% (+7.5) and for Qwen3-VL 8B from 91.8% to 98.3% (+6.5), indicating substantially fewer missing required tools, while Qwen3-VL 30B shows a smaller increase from 97.1% to 97.9% (+0.8) consistent with its already high baseline. The largest effect is on precision, where PTM reduces over-selection of unnecessary tools as precision rises for Qwen3-VL 4B from 66.3% to 82.8% (+16.5), for Qwen3-VL 8B from 62.3% to 90.0% (+27.7), and for Qwen3-VL 30B from 76.7% to 84.0% (+7.3). These improvements translate into consistent F1 gains across all model sizes where Qwen3-VL 4B improves from 74.0% to 87.8% (+13.8), Qwen3-VL 8B from 73.2% to 93.1% (+19.9), and Qwen3-VL 30B from 83.6% to 88.7% (+5.1). Together with the difficulty-stratified results, this aggregate view supports the PTM hypothesis where retrieved workflow priors improve tool decision making by simultaneously reducing missing-tool errors (higher recall) and spurious-tool errors (higher precision), with the largest benefits on lower-capability backbones but persistent gains even for the strongest model.

5. CONCLUSION

In this paper we propose persistent tool memory (PTM) to address a central reliability bottleneck in tool-grounded manufacturing agents. Despite strong progress in tool calling, many agentic systems still treat each query as a cold start and re-derive tool choice and execution order from scratch leading to inconsistent and brittle multi-step tool use under ambiguity, noise, and partial specification. PTM introduces a validation-gated episodic memory that stores successful tool-use episodes as structured procedural artifacts and retrieves them at inference time as constrained workflow priors. PTM is executed by the Meta Agent, which queries a persistent episode store, converts retrieved neighbors into a tool suggestion set of candidate tools and step patterns, and enforces argument schema, dependency order, unit consistency, and plausibility constraints before allowing downstream execution. We evaluate PTM on a rotor-blade CNC machining benchmark of 300 questions with strict equal-difficulty coverage and demonstrate consistent improvements in tool-decision quality across three Qwen3-VL models (4B, 8B, 30B). PTM increases tool-decision F1 across all difficulty levels and all model sizes, with the largest gains on easier and moderately complex questions for lower-capability models, and it also improves aggregate recall by reducing missing required tools while improving precision by reducing spurious tool selection. These results support the PTM premise that retrieving validated procedural priors reduces tool-selection uncertainty and stabilizes tool choice in multi-step decision pipelines without modifying deterministic tools or relaxing safety constraints. In future work, we will explore policy-level changes that more directly condition routing on retrieved episodes, uncertainty-aware retrieval and acceptance criteria to further reduce variance under distribution shift, and end-to-end evaluations that incorporate full multi-step execution success and numerical fidelity. We also plan to study governance and continual refinement of the episode store, including curation, aging, and deprecation, as tools and manufacturing conditions evolve.

ACKNOWLEDGMENTS

This work was supported under Cooperative Agreement W56HZV-21-2-0001 with the U.S. Army DEVCOM Ground Vehicle Systems Center (GVSC), through the Virtual Prototyping of Autonomy Enabled Ground Systems (VIPR-GS) program; the National Science Foundation (grant no. 2434519); and the Department of Energy (grant no. DE-EE0011029).

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA and shall not be used for advertising or product endorsement purposes.

REFERENCES

[1] Zhang, Chao, Xu, Qingfeng, Yu, Yongrui, Zhou, Guanghui, Zeng, Keyan, Chang, Fengtian and Ding, Kai. “A survey on potentials, pathways and challenges of large language models in new-generation intelligent manufacturing.” *Robotics and Computer-Integrated Manufacturing* Vol. 92 (2025): p. 102883.

[2] Li, Yiwei, Zhao, Huaqin, Jiang, Hanqi, Pan, Yi, Liu, Zhengliang, Wu, Zihao, Shu, Peng, Tian, Jie, Yang, Tianze, Xu, Shaochen et al. “Large language models for manufacturing.” *arXiv preprint arXiv:2410.21418* (2024).

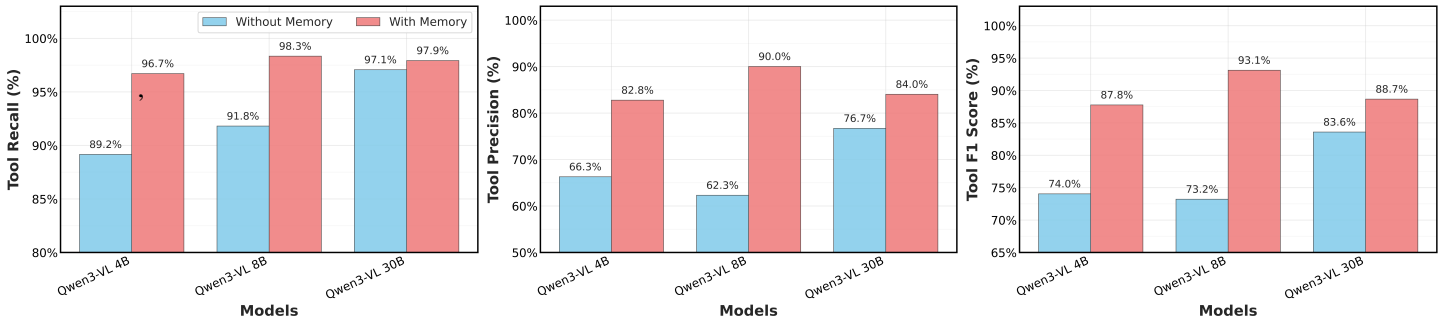

FIGURE 4: OVERALL PERFORMANCE OF PTM WITH MEMORY AND WITHOUT MEMORY ACROSS THREE DIFFERENT SIZED QWEN3-VL MODELS.

[3] Lim, Jonghan, Vogel-Heuser, Birgit and Kovalenko, Ilya. "Large language model-enabled multi-agent manufacturing systems." *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*: pp. 3940–3946. 2024. IEEE.

[4] Werheid, Jonas, Melnychuk, Oleksandr, Zhou, Hans, Huber, Meike, Rippe, Christoph, Joosten, Dominik, Keskin, Zozan, Wittstamm, Max, Subramani, Sathya, Drescher, Benny et al. "Designing an llm-based copilot for manufacturing equipment selection." *Manufacturing Letters* (2025).

[5] Ye, Junjie, Wu, Yilong, Gao, Songyang, Huang, Caishuang, Li, Sixian, Li, Guanyu, Fan, Xiaoran, Zhang, Qi, Gui, Tao and Huang, Xuan-Jing. "Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning." *Proceedings of the 2024 conference on empirical methods in natural language processing*: pp. 313–333. 2024.

[6] Schick, Timo, Dwivedi-Yu, Jane, Dessì, Roberto, Raileanu, Roberta, Lomeli, Maria, Hambro, Eric, Zettlemoyer, Luke, Cancedda, Nicola and Scialom, Thomas. "Toolformer: Language models can teach themselves to use tools." *Advances in Neural Information Processing Systems* Vol. 36 (2023): pp. 68539–68551.

[7] Patil, Shishir G, Zhang, Tianjun, Wang, Xin and Gonzalez, Joseph E. "Gorilla: Large language model connected with massive apis, 2023." *URL https://arxiv.org/abs/2305.15334* (2023).

[8] Du, Yu, Wei, Fangyun and Zhang, Hongyang. "Anytool: Self-reflective, hierarchical agents for large-scale api calls." *arXiv preprint arXiv:2402.04253* (2024).

[9] Qiao, Shuofei, Gui, Honghao, Lv, Chengfei, Jia, Qianghuai, Chen, Huajun and Zhang, Ningyu. "Making language models better tool learners with execution feedback." *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*: pp. 3550–3568. 2024.

[10] Huang, Yue, Shi, Jiawen, Li, Yuan, Fan, Chenrui, Wu, Siyuan, Zhang, Qihui, Liu, Yixin, Zhou, Pan, Wan, Yao, Gong, Neil Zhenqiang et al. "Metatool benchmark for large language models: Deciding whether to use tools and which to use." *arXiv preprint arXiv:2310.03128* (2023).

[11] Liu, Jian, Ning, Kangyun, Su, Yisong, Han, Wenjuan, Xu, Jinan and Zhang, Yuanzhe. "Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models." *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*: pp. 1–5. 2025. IEEE.

[12] Wang, Zora Zhiruo, Mao, Jiayuan, Fried, Daniel and Neubig, Graham. "Agent workflow memory." *arXiv preprint arXiv:2409.07429* (2024).

[13] Hoang, Danny, Gorsich, David, Castanier, Matthew P and Imani, Farhad. "Knowledge Graph Fusion with Large Language Models for Accurate, Explainable Manufacturing Process Planning." *arXiv preprint arXiv:2506.13026* (2025).

[14] Zhou, Qianyu, Zhang, Yang, Kim, Jeongho, Imani, Farhad and Tang, Jiong. "Spatially-Informed Online Prediction of Milling Surface Deformation Using Multiphysics-Infused Graph Neural Network for Digital Twinning." *Journal of Manufacturing Science and Engineering* Vol. 147 No. 12 (2025): p. 121003.

[15] Hoang, Danny, Chen, Ruimin, Bollas, George and Imani, Farhad. "Hyperdimensional computing for explainable information fusion and multi-task adaptation in advanced manufacturing." *Information Fusion* (2025): p. 103898.

[16] Hoang, Danny, Gorsich, David, Castanier, Matthew and Imani, Farhad. "Enabling Grounded Answers Through Knowledge Graphs and Retrieval Augmented Generation." Technical report no. SAE Technical Paper. 2025.

[17] Qu, Changle, Dai, Sunhao, Wei, Xiaochi, Cai, Hengyi, Wang, Shuaiqiang, Yin, Dawei, Xu, Jun and Wen, Ji-Rong. "Tool learning with large language models: A survey." *Frontiers of Computer Science* Vol. 19 No. 8 (2025): p. 198343.

[18] Qin, Yujia, Liang, Shihao, Ye, Yining, Zhu, Kunlun, Yan, Lan, Lu, Yaxi, Lin, Yankai, Cong, Xin, Tang, Xiangru, Qian, Bill et al. "Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023." *URL https://arxiv.org/abs/2307.16789* (2023).

[19] Yang, An, Li, Anfeng, Yang, Baosong, Zhang, Beichen, Hui, Binyuan, Zheng, Bo, Yu, Bowen, Gao, Chang, Huang, Chengen, Lv, Chenxu et al. "Qwen3 technical report." *arXiv preprint arXiv:2505.09388* (2025).