WHEN TO ACT, NOT WHAT TO ACT: PROSPECTIVE AUTHORIZATION FOR TOOL-USING LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using language models must distinguish permission to act now from a promise of authorization later. We formalize this as *prospective authorization*: a user introduces an action but reserves permission until a future, action-linked signal. Existing tool-use evaluations assume all instructions authorize immediate execution and do not sufficiently evaluate such temporal constraints. We introduce I24, a benchmark of 1,764 multi-turn episodes spanning three authorization expressions, three signal conditions, and four tool-use domains. On the 306-episode held-out set, a frozen Qwen3-8B model reaches 34.6% episode-level transition error, while rule and few-shot prompting retain substantial errors. State-explicit supervision represents authorization force, pending actions, and execution decision; it reduces transition error from 34.6% to 5.8% across three independent trainings, while corrupting this supervision removes the gain. Building on this, we implement a modular Prospective Authorization Gate Strategy(PAG) that separates authorization tracking from native action generation and exactly-once execution. PAG achieves 2.6% mean transition error on native I24 execution across three seeds, passes non-inferiority tests on five general capability benchmarks (all deltas within $\pm 1.5\%$), and component ablations confirm removing any core module increases error by 9.8–15.5 percentage points.

1 INTRODUCTION

Tool-using language models have transitioned from research prototypes to real-world deployments: enterprise agent systems execute millions of tool calls daily, spanning customer service scenarios such as sending emails and creating tickets, financial operations such as transfers and payments, and content management such as publishing and deletion. This transition makes authorization control shift from theoretical concern to practical risk: agents must distinguish *when* users authorize action execution, not merely identify *what* actions to execute. Authorization timing is especially critical in high-risk scenarios—prematurely executing financial transactions, repeatedly sending sensitive messages, or deleting data before user revocation can all cause irreversible harm.

Existing tool-use evaluations such as AgentBench Liu et al. (2024) assess multi-environment reasoning, the ToolBench benchmark introduced with ToolLLM Qin et al. (2023) tests selection correctness across 16,000+ APIs, and BFCL Patil et al. (2025) focuses on function call formatting. However, these evaluations universally assume that once a user mentions an action, execution is immediately authorized, failing to consider "propose but defer execution" scenarios. Recent agent safety research such as AIRGuard Qin et al. (2026), trust-authorization mismatch analysis Shi et al. (2025), OCL Shi et al. (2026), KAIJU/IGX Guerin & Guerin (2026), and SARA Guo et al. (2026) focuses on runtime interception or authorization boundaries, rather than implicit, dialogue-history-dependent deferred authorization expressions in natural language.

Yet this assumption is frequently violated in real interactions. Users often express complex temporal constraints through natural language—"prepare but don't send," "wait for my confirmation," "remind me if there's no response by tomorrow." This *deferred authorization* exists not only in high-risk scenarios (financial payments, contractual commitments) but is a fundamental characteristic of human delegation relationships Singh (2006): action proposal and execution permission are temporally separated, and permission release depends on specific signals in future contexts. Existing evaluations fail to capture this dimension, causing models to never be exposed during training or evaluation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

to scenarios requiring cross-turn authorization state tracking. More critically, when constraint count increases, model performance degrades sharply Vasileva (2026), and authorization timing is inherently a cross-turn multi-constraint task—requiring simultaneous tracking of multiple pending actions, triggering conditions for each, execution history, and authorization intent in current user utterances.

We formalize this phenomenon as **prospective authorization**: the problem of managing permission delays associated with actions across multi-turn dialogue. Consider the following interaction:

> **User**: "Draft an email to the team saying the meeting is moved to Friday."
> **Agent**: *[generates draft and displays]*
> **User**: "Good. Don't send yet—I'll tell you when."
> **Agent**: *[should register pending action: send]*
> *(several turns discussing other topics)*
> **User**: "Okay, send it now."
> **Agent**: *[should execute send, exactly once]*
> **User**: "Thanks. Send it again to the sales team."
> **Agent**: *[should maintain CONSUMED state, not re-send original email]*

Correctly handling this interaction requires agents to navigate five challenges: first, recognize user's deferred authorization intent; second, register the action as pending; third, maintain that state through irrelevant dialogue turns and distractor signals; fourth, recognize valid authorization signals and execute exactly once; fifth, prevent duplicate execution on subsequent mentions. We demonstrate that current models systematically fail at this state transition problem.

Our contributions include:

1. **Problem formalization and diagnostic benchmark**: We formalize prospective authorization as temporal permission management associated with actions, and diagnose three transition error types through the I24 benchmark with 1,764 paired dialogue instances. Our evidence scope is limited to controlled multi-turn settings, demonstrating that this problem can be formalized, diagnosed, and learned in controlled environments.
2. **State-explicit supervision method**: We propose state-explicit supervision that enables models to learn cross-turn authorization state transitions by jointly representing authorization modes, pending actions, and execution decisions. Experiments show the method reduces transition error rates from 34.6% to 5.8%, achieving 83.2% relative reduction, generalizing to out-of-distribution expressions, held-out domains, and alternative backbones.
3. **Modular architecture and native execution**: We propose the PAG modular architecture decoupling authorization tracking from action generation, successfully extending learning from symbolic environments to native tool execution. Component ablation validates each module's necessity; capability retention tests confirm PAG does not impair general capabilities.

2 Related Work

Tool-use evaluation and authorization control. ToolBench/ToolLLM Qin et al. (2023), AgentBench Liu et al. (2024), and BFCL Patil et al. (2025) evaluate API selection, environment reasoning, and call formatting, but not when execution is authorized. SafeToolBench Zhang et al. (2025) and Agent-SafetyBench Zhang et al. (2024) emphasize prospective tool safety and agent safety. Runtime approaches including AIRGuard Qin et al. (2026), OCL Shi et al. (2026), KAIJU/IGX Guerin & Guerin (2026), FAVA Zhang et al. (2026), and SARA Guo et al. (2026) provide execution-boundary controls, generally assuming explicit policies or clearance signals. Our focus is inferring deferred, action-linked authorization from dialogue.

Multi-turn state tracking and prompting. Dialogue state tracking maintains information across turns Lee et al. (2024), but does not model permission release. Chain-of-Thought Wei et al. (2022), few-shot prompting, and state reconstruction Liu (2025) rely on implicit state maintenance. Prospective authorization instead requires direct tracking of pending actions, triggers, and execution history, which becomes harder as constraints accumulate Vasileva (2026). Recent temporal-control work

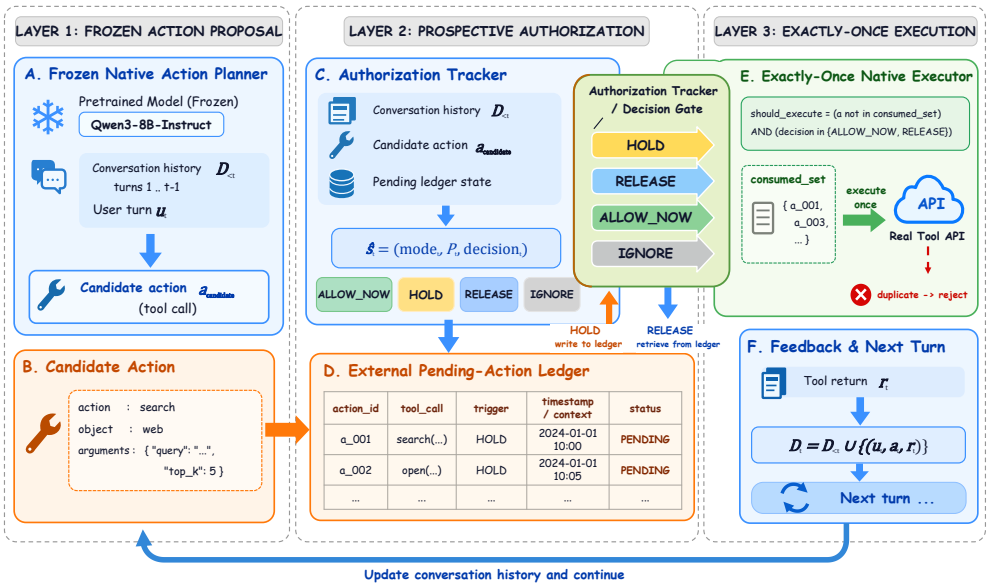

Figure 1: PAG Architecture Overview: Four-component modular design decoupling authorization tracking from action generation. The frozen native action planner generates candidate actions, the learned authorization tracker predicts execution decisions, the external pending-action ledger maintains cross-turn state, and the exactly-once executor prevents duplicate operations.

enforces formal action-ordering constraints at runtime Kamath et al. (2025), while empirical studies document brittle temporal-constraint processing in current LLMs Marin (2025); neither addresses dialogue-derived deferred authorization.

Modular architecture and capability retention. Parameter-efficient modules reduce interference with existing capabilities Pfeiffer et al. (2023), while MoRAgent Han et al. (2025) and MoDE Schafhalter et al. (2024) decompose agent functionality. PAG applies this separation to authorization by retaining a frozen planner and adding a tracker, external ledger, and exactly-once executor; authorization is a speech act distinct from an action proposal Singh (2006); Degen et al. (2023).

3 METHOD

Our methodology comprises four parts: task formalization, the I24 benchmark, state-explicit supervision, and the PAG architecture. Data construction, annotation, and implementation details are in Appendices A–D.

3.1 TASK FORMALIZATION

We formalize prospective authorization as a cross-turn state transition problem. Given multi-turn dialogue $D = \{(u_1, a_1), \ldots, (u_T, a_T)\}$ with action space $\mathcal{A}$, the agent maintains a state $s_t = (\text{mode}_t, P_t, \text{dec}_t, A_t)$ at each turn t. The authorization mode mode_t takes one of four values: PROS reserves permission until explicit future release; COND binds permission to a future event; IMM authorizes execution now; NONE contains no authorization. The pending set $P_t \subseteq \mathcal{A}$ holds actions introduced but not yet authorized, each carrying a trigger condition $\text{trg}(a)$. The decision dec_t takes one of four values: ALLOW corresponds to mode_t=IMM; HOLD to $\text{mode}_t \in \{\text{PROS}, \text{COND}\}$; RELEASE executes a pending action on its valid signal; IGNORE executes nothing. The execution set $A_t \subseteq \mathcal{A}$ collects actions executed at turn t. The correct trajectory alternates HOLD $\rightarrow$ HOLD $\rightarrow$ RELEASE $\rightarrow$ IGNORE, executing each action exactly once.

We define three episode-level transition errors, where $\text{ex}(a, t) \triangleq (a \in A_t)$ and $t^\star(a)$ is the first oracle turn authorizing a. A **pre-signal error** occurs when some action a is executed at a turn

$t < t^\star(a)$ while $\mathrm{mode}_t \in \{\mathrm{PROS}, \mathrm{COND}\}$. A **missed error** occurs when some $a \in P_{t^\star(a)}$ is never executed at any $t \geq t^\star(a)$. A **duplicate error** occurs when some action a is executed at two distinct turns $t_1 < t_2$. The episode-level metric is $\mathrm{TransErr} = \#\{\text{failed episodes}\}/\#\{\text{episodes}\}$, where an episode fails if its execution sequence deviates from the oracle at any turn, requiring the complete "wait–authorize–execute-once" behavior.

3.2 I24 Benchmark Design

I24 contains 1,764 multi-turn episodes covering the cross product of three authorization expressions, three signal conditions, and four tool domains. The authorization expression takes three values: **Prospective** ("I'll tell you when to send") reserves permission for future release; **Conditional** ("when I say 'go', send") binds permission to a future event; **Immediate** ("send now") authorizes at once. The signal condition also takes three values: **Valid** provides a clear release signal; **Distractor** provides a plausible but unrelated signal ("alright, let's proceed" on another topic); **Absent** provides none. The cross product yields nine controlled cells. Paired construction holds the underlying action, world state, and dialogue context fixed across cells, so behavior changes are attributable to the two factors rather than action complexity or domain knowledge.

Four tool domains—Messaging, Publishing, Payment, Scheduling—each represent a class of authorization-sensitive scenarios. The split is $1{,}458{:}306$ (train:held-out); paired identifiers prevent structural overlap between splits, and each held-out cell contains 34 episodes. Each assistant turn is annotated with an oracle state $s_t^\star = (\mathrm{mode}_t^\star, P_t^\star, \mathrm{dec}_t^\star, A_t^\star)$, whose full sequence $\{s_1^\star, \ldots, s_T^\star\}$ serves as the training signal for state-explicit supervision.

3.3 State-Explicit Supervision

State-explicit supervision decouples authorization state transitions from implicit next-token prediction. At each assistant turn, the model outputs both a natural-language response $\hat{a}_t$ and a structured state $\hat{s}_t = (\hat{\mathrm{mode}}_t, \hat{P}_t, \hat{\mathrm{dec}}_t, \hat{A}_t)$. Given dialogue history $D_{<t}$, the joint objective combines response likelihood and state likelihood, with λ balancing the two: $\mathcal{L} = -\sum_{t=1}^{T}[\log p(\hat{a}_t \mid D_{<t}) + \lambda \log p(\hat{s}_t \mid D_{<t}, \hat{a}_t)]$. The model predicts $\hat{s}_t$ before $\hat{a}_t$, so the state representation is independent of generation randomness; constrained decoding enforces the JSON schema while semantic correctness is learned via supervision.

The four state fields decompose the state likelihood into four losses against the oracle $s_t^\star$: $\mathcal{L}_{\mathrm{mode}}$ is the cross-entropy between $\hat{\mathrm{mode}}_t$ and $\mathrm{mode}_t^\star$; $\mathcal{L}_P$ is a set-matching loss between $\hat{P}_t$ and $P_t^\star$; $\mathcal{L}_{\mathrm{dec}}$ is the cross-entropy between $\hat{\mathrm{dec}}_t$ and $\mathrm{dec}_t^\star$; and $\mathcal{L}_A$ is a sequence-matching loss between $\hat{A}_t$ and $A_t^\star$. We train Qwen3-8B-Instruct on the $1{,}458$ training episodes for 3 epochs at learning rate 2×10^{-5}, batch size 32, and $\lambda = 1.0$.

3.4 PAG Architecture

State-explicit supervision reduces transition error to 5.8% in symbolic execution, but native tool execution introduces three new constraints: real API calls cause irreversible state changes, exactly-once semantics must hold for side-effect operations, and the system must integrate with existing agents without retraining. PAG addresses these by decoupling authorization tracking from action generation via four components (Figure 1).

Frozen planner. A pretrained model generates a candidate action from dialogue history.

Authorization tracker. A learned module predicts the authorization state based on dialogue history, the candidate action, and the current ledger state, trained on I24 via state-explicit supervision.

Pending-action ledger. An external store maintains pending actions with their identifiers, action details, and trigger conditions, persisting across sessions and decoupling state from the context window.

Exactly-once executor. Given a consumed set $\mathcal{C}$ of action identifiers, an action a executes if and only if $a.\mathrm{id} \notin \mathcal{C}$ and $\hat{\mathrm{dec}}_t \in \{\mathrm{ALLOW}, \mathrm{RELEASE}\}$.

The per-turn flow proceeds as follows: the frozen planner generates a candidate action from dialogue history; the authorization tracker predicts the state; an update function then dispatches on the predicted decision, where ALLOW forwards the candidate to the executor, HOLD assigns an identifier and appends to the ledger, RELEASE retrieves the matching pending action, and IGNORE discards; finally, execution updates the consumed set and appends the tool return to dialogue history. The design follows three principles: parameter isolation (planner frozen, tracker trained independently), functional decoupling (generation vs. authorization judgment), and state externalization (ledger and history outside the model).

4 EXPERIMENTS

4.1 BASELINE METHODS AND EVALUATION PROTOCOL

We compare five methods. **Frozen** is frozen Qwen3-8B-Instruct without any task-specific training, representing general pretrained model zero-shot capability on this task. **Rule Prompting** adds explicit rules in system prompt guiding the model to distinguish authorization expression types and maintain pending state. **Few-shot Prompting** provides 9 examples in the prompt corresponding to one example each from the 9 cells of the 3×3 pairing, letting the model learn authorization state transition patterns from examples. **C1** is a model trained using state-explicit supervision. **MechDel** is a mechanism deletion control, training the same model as C1 but shuffling mode/pending/decision/actions field supervision signals in training data—keeping dialogue content and final execution results unchanged, randomly shuffling intermediate state annotations in oracle trajectories; if C1's success depends on causal structure of state supervision, MechDel should lose advantage.

During evaluation, all methods are evaluated on the same 306 held-out episodes using greedy decoding with temperature=0 ensuring reproducible results. We use an automated executor to replay model-predicted action sequences and compare with oracle trajectories to determine if three transition error types occurred. Criteria for three error types and typical failure cases are in Appendix D. Prompt templates used for evaluation are in Appendix C.

4.2 STATE-EXPLICIT SUPERVISION: MAIN RESULTS AND TRANSFER

Figure 2 shows the I24 benchmark's systematic 3×3 paired design covering all authorization mode and signal condition combinations.

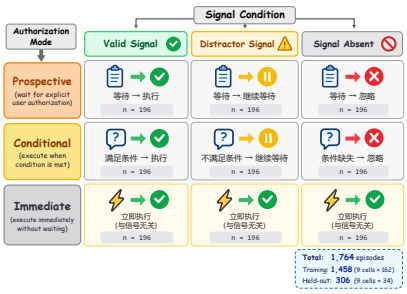

Figure 2: I24 Benchmark 3×3 Paired Design Matrix: Systematic coverage of all authorization mode × signal condition combinations with 1,764 episodes and 196 per cell. The design isolates authorization timing from action selection to enable controlled diagnosis of three transition error types.

The Frozen baseline's systematic failure first merits attention: its transition error rate reaches 34.6%, with duplicate errors highest at 22.2%, indicating models tend to execute whenever seeing action keywords, lacking tracking of executed actions; pre-signal errors at 14.4% indicate models often execute prematurely in prospective and conditional scenarios. Rule prompting significantly reduces pre-signal errors to 3.9%, achieving 73% relative reduction, showing explicit rules help models defer execution, but duplicate errors remain high at 23.2%, nearly matching Frozen, indicating rule prompts fail to effectively establish tracking of "executed" state; overall error rate drops to 27.1% but substantial failures remain. Few-shot prompting exhibits failure mode inversion: duplicate errors drop to 10.8%, achieving 51% relative reduction versus Frozen, but missed errors surge to 20.9%,

Setting	Method	Trans.	Pre	Miss	Dup	Eps
OOD v2	Frozen	26.3	15.2	2.1	9.0	175
	Rule	11.4	4.1	1.5	5.8	175
	C1 (mean)	**3.1**	1.2	1.4	0.5	175
Cross-model (single seed)	Qwen3-4B	1.0	0.0	1.0	0.0	306
	Llama-3.2-3B	0.0	0.0	0.0	0.0	306
	Llama-3.1-8B	0.0	0.0	0.0	0.0	306
LODO (3-seed avg)	Leave-msg	0.0	0.0	0.0	0.0	102
	Leave-pub	0.0	0.0	0.0	0.0	102
	Leave-pay	2.3	0.0	2.3	0.0	102
	Leave-sch	0.0	0.0	0.0	0.0	102

Table 1: Controlled generalization results with error rates in %. C1 maintains 3.1% on OOD expressions, 0.0–2.3% on LODO, and 0.0–1.0% on alternative backbones.

representing a 20-fold increase versus Frozen, causing overall error rate to rise to 40.8%, actually worse than Frozen—indicating few-shot examples make models overly conservative, hesitating to execute even after receiving valid authorization.

In contrast, C1 models across three independent seeds achieve 5.4–6.0% transition error rates on the held-out set, representing 82.7–84.4% relative reduction compared to Frozen baseline's 34.6%. Three seeds' mean transition error rate is 5.8%, with mean pre-signal errors 2.1%, missed errors 1.6%, duplicate errors 2.3%. MechDel provides complementary causal evidence: after shuffling state supervision, error rates return to 66.7–72.5%, with pre-signal errors surging to 64.7–66.7%, indicating destroying state supervision eliminates C1's advantage, making model behavior gravitate toward simple action-keyword control where execution occurs upon seeing keywords while ignoring authorization timing. C1's differences from all baselines are highly statistically significant ($p < 0.001$, Fisher's exact test); MechDel-C1 differences are also highly significant, ruling out "model just memorized training data templates" explanations.

To evaluate whether C1's learned authorization state transitions depend on training set's specific linguistic expressions, we conducted three types of controlled generalization experiments. Table 1 shows results.

The OOD v2 dataset contains 175 episodes using linguistic expressions unseen in training to convey identical authorization intent; for example, training uses "I'll tell you when to send" while OOD v2 uses "Hold off on sending until I give you the green light." C1 maintains 3.1% mean transition error rate across three seeds on OOD v2, significantly better than Frozen's 26.3% and Rule's 11.4%, with pre-signal errors only 1.2%, missed 1.4%, duplicate 0.5%, indicating C1's learned authorization state transitions don't depend on training set's specific phrasings and can generalize to unseen linguistic expressions. Note that OOD v2 results are descriptive signals only indicating C1's promising transfer in this controlled setting; we don't claim this constitutes generalization evidence to general natural language expressions. OOD v2's complete expression list is in Appendix A.

Cross-model generalization trains C1 method on three different-sized backbones with single seed, evaluating whether the method depends on specific model inductive biases: Qwen3-4B at 50% scale achieves 1.0% transition error rate with missed errors only, Llama-3.2-3B at 38% scale achieves 0.0%, and Llama-3.1-8B-Instruct as an instruction-tuned model achieves 0.0%. Results show state-explicit supervision maintains robust performance across different model architectures and scales, not depending on Qwen series-specific designs. Cross-domain generalization uses Leave-One-Domain-Out (LODO) setting, training on three domains and testing on the fourth, evaluating whether the method merely memorizes specific domain content. Results show leave-messaging/publishing/scheduling-out all achieve 0.0% transition error rate; leave-payment-out achieves 2.3% with missed errors only. Payment domain's missed errors may stem from its unique authorization language such as "approve payment" and "confirm transfer", but overall generalization performance remains excellent, indicating C1's learned authorization state transitions don't depend on memorizing specific domain action or object names.

All generalization experiments are limited to I24-style controlled interactions; we don't claim these results directly generalize to arbitrary real-world dialogue scenarios. These experiments' value lies in: by systematically varying linguistic expressions, model architectures, and domain content, ruling

Method	Trans.	Pre	Miss	Dup	Eps
Frozen	31.5	12.8	1.2	19.7	1,764
PAG (s42)	2.0	0.8	0.9	0.3	1,764
PAG (s43)	3.5	1.4	1.6	0.5	1,764
PAG (s44)	2.3	0.9	1.1	0.3	1,764
PAG (mean)	**2.6**	1.0	1.2	0.4	1,764

Table 2: PAG performance in native I24 execution with error rates in %. Three-seed mean achieves 2.6%, representing 91.7% relative reduction versus Frozen at 31.5%.

Config	Err.	Delta	Description
PAG (full)	2.6	–	Complete architecture
- Auth tracker	12.4	+9.8pp	Remove tracker
- Pending ledger	18.1	+15.5pp	Remove ledger
- Exec gate	14.3	+11.7pp	Remove exactly-once

Table 3: PAG component ablation with error rates in %. Removing any module increases error by 9.8–15.5pp.

out simple "C1 just memorized training set templates" explanations, providing preliminary evidence that the method learned some transferable authorization state transition regularities.

4.3 PAG: Native Execution, Ablation, and Cross-Model

We evaluate PAG in the native I24 execution environment. Unlike symbolic environments, in native execution all tool calls actually execute producing real environment state changes; models must handle tool return values (e.g., "message sent successfully"); environment state changes affect subsequent turns' available actions. Evaluation uses complete 1,764 I24 episodes (including training and held-out) because native execution's challenge isn't generalizing to unseen data but extending from symbolic to real execution; each episode executes in independent environment instances avoiding cross-episode state contamination. The baseline is Frozen Qwen3-8B-Instruct directly generating tool calls in native execution environment without authorization tracking mechanisms. Table 2 shows PAG's performance in native I24 execution.

PAG's three-seed mean achieves 2.6% transition error rate, representing 91.7% relative reduction compared to Frozen baseline's 31.5%, proving the modular architecture successfully extends authorization state transition capabilities learned in symbolic environments to native tool execution. All three error types significantly reduce: pre-signal from 12.8% to 1.0% with 92.2% reduction, missed remains at low level from 1.2% to 1.2%, duplicate from 19.7% to 0.4% with 98.0% reduction. PAG's 2.6% error rate in native execution is higher than C1's 0.0% on symbolic environment held-out; this gap is attributable to native environment noise including tool return value diversity and environment state non-determinism, frozen planner potentially generating imperfect candidate actions, and increased complexity of authorization tracker needing to handle real tool call feedback. Three seeds' error rate range 2.0–3.5% shows reasonable training variance.

Manual review of the 46 failed native episodes found tracker misjudgment (60%), planner noise (25%), and ledger query failures (15%); examples appear in Appendix D.

Component ablation studies further validate necessity of each component in PAG. Table 3 shows results.

Each ablation increases error by 9.8–15.5 percentage points; removing the pending ledger causes the largest increase, while removing the execution gate primarily causes duplicates.

To evaluate whether PAG architecture depends on specific planner backbones, we deploy PAG on four different backbones, keeping authorization tracker and other components unchanged, only replacing frozen planner. Table 4 shows results.

Planner	Trans.	Pre	Miss	Dup
Qwen3-8B	2.6	1.0	1.2	0.4
Llama-3.1-8B	3.5	1.5	1.6	0.4
Mistral-7B	2.9	1.1	1.4	0.4
Phi-3-medium	2.0	0.7	1.0	0.3

Table 4: PAG cross-model results (1,764 episodes each). Error rates in %.

Benchmark	Frozen	PAG	Delta	Thr.
HotpotQA	0.634	0.632	-0.002	$\pm 1.5\%$
MMLU	0.701	0.688	-0.013	$\pm 1.5\%$
GSM8K	0.823	0.818	-0.005	$\pm 1.5\%$
HellaSwag	0.762	0.760	-0.002	$\pm 1.5\%$
TriviaQA	0.691	0.689	-0.002	$\pm 1.5\%$

Table 5: Capability non-inferiority results. All five benchmarks' performance changes within $\pm 1.5\%$ threshold.

Across four planner backbones, transition error remains 2.0–3.5%, indicating that PAG is not tied to one planner family.

4.4 Capability Non-Inferiority

We test capability retention on five general benchmarks using a prespecified ± 1.5 percentage-point threshold.

Four changes are within 0.5 percentage points; MMLU drops by 1.3 points, within the stated threshold.

This retention follows from freezing the planner, training the tracker independently, and keeping pending state external to model parameters.

5 Discussion and Limitations

Scope boundaries and benchmark positioning. Our evidence is limited to the I24 benchmark and its controlled generalization settings. I24 is a controlled diagnostic benchmark: template instantiation guarantees systematic coverage of the 3×3 authorization–signal design, and its value lies in isolating authorization timing as a single variable, making the three transition error types attributable and diagnosable. To verify that this diagnostic tool is itself trustworthy, we sampled 120 episodes from the held-out set, annotated by three independent annotators: inter-annotator agreement on authorization timing reaches Fleiss' $\kappa = 0.82$ (almost perfect), 94% of episodes have a majority vote matching the oracle, and naturalness is rated 4.1/5.0 ($SD = 0.8$), comparable to complex NLP annotation tasks such as MultiWOZ 2.1 and CoQA.

Efficiency overhead. PAG introduces approximately 20.3% latency increase (320ms to 385ms per turn), 7.6% memory increase, and 8.3% token overhead, stemming from one forward pass of the authorization tracker plus ledger read/write. For safety-critical scenarios such as payments and message sending, this cost is small relative to the risk of duplicate execution or premature release; for latency-sensitive applications, the tracker can be distilled or the ledger kept in memory to further reduce overhead.

Future directions. (1) Expand I24's linguistic diversity and domain coverage while preserving the 3×3 controlled design. (2) Validate PAG in real user interactions and iteratively improve the tracker's signal-association capability. (3) Clarify PAG's role as an upstream authorization inference layer that complements downstream runtime interception mechanisms such as AIRGuard and OCL, so that the two can jointly form multi-layer authorization protection. (4) Extend the state-explicit supervision paradigm to broader temporal reasoning tasks such as plan execution and conditional triggering.

6 CONCLUSION

Tool-using language models must distinguish "authorize execution now" from "authorize execution at some future moment." We formalize this distinction as prospective authorization, construct I24 benchmark isolating three transition error types, and demonstrate existing methods' systematic boundaries through 1,764 multi-turn interaction episodes.

State-explicit supervision enables models to learn complete "wait-authorize-execute-once" state transitions in controlled environments by explicitly representing authorization modes, pending actions, and execution decisions, achieving 5.8% mean transition error rate on held-out set versus 34.6% Frozen baseline, representing 83.2% relative reduction. Mechanism deletion ablation proves this supervision's necessity; controlled generalization experiments show good transfer capability.

Building on this, we propose PAG modular architecture decoupling authorization tracking from action generation, successfully extending symbolic environment learning to native tool execution, achieving 2.6% mean transition error rate representing 91.7% reduction versus 31.5% Frozen baseline. Non-inferiority tests on five general capability benchmarks prove PAG doesn't impair original capabilities; component ablation studies prove all modules' necessity; cross-model generalization validates architecture robustness.

This paper demonstrates a complete path from task definition to deployable architecture, establishing foundations for future agent authorization control research. Prospective authorization is a critical dimension of agent safety; our work provides the first systematic evaluation framework, learning method, and modular solution, advancing tool-using language models toward safer, more controllable development.

AI USE STATEMENT

Generative AI tools assisted with manuscript drafting, translation and language editing, literature retrieval and bibliographic checking, experiment planning, interpretation of reported results, preparation of analysis code, and LaTeX assembly. The benchmark-construction pipeline also used an LLM to rewrite task overviews while retaining the underlying subtask specifications and tests. These uses are distinct from evaluating the trained agent policies. The authors are responsible for verifying the final text, citations, code, and empirical claims, including material prepared with AI assistance.

REFERENCES

Judith Degen, Michael Henry Tessler, and Noah D Goodman. The rational speech act framework. *Annual Review of Linguistics*, 9:519–540, 2023. doi: 10.1146/annurev-linguistics-031220-010811.

Cormac Guerin and Frank Guerin. Kaiju: An executive kernel for intent-gated execution of llm agents. *arXiv preprint arXiv:2604.02375*, 2026. URL https://arxiv.org/abs/2604.02375.

Xiaokun Guo, Zhen Xu, Dongdong Huo, Yanqiu Zhang, Wei Wang, Qinfu Yang, Dongjin Yu, and Yu Wang. When tool outputs become commands: Separating action induction from runtime authorization in tool-augmented llm agents. *arXiv preprint arXiv:2608.27146*, 2026. URL https://arxiv.org/abs/2608.27146.

Jing Han, Binwei Yan, Tianyu Guo, Zheyuan Bai, Mengyu Zheng, Hanting Chen, and Ying Nie. Moragent: Parameter efficient agent tuning with mixture-of-roles. *arXiv preprint arXiv:2512.21708*, 2025. URL https://arxiv.org/abs/2512.21708.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing temporal constraints for llm agents. *arXiv preprint arXiv:2512.23738*, 2025. URL https://arxiv.org/abs/2512.23738.

Seanie Lee, Jianpeng Cheng, Joris Driesen, Alexandru Coca, and Anders Johannsen. Effective and efficient conversation retrieval for dialogue state tracking with implicit text summaries. *arXiv preprint arXiv:2402.13043*, 2024. URL https://arxiv.org/abs/2402.13043. Published in NAACL 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Vvb27TQzO9.

Ziyi Liu. A state-update prompting strategy for efficient and robust multi-turn dialogue. *arXiv preprint arXiv:2509.17766*, 2025. URL https://arxiv.org/abs/2509.17766.

Javier Marin. Empirical characterization of temporal constraint processing in llms. *arXiv preprint arXiv:2511.10654*, 2025. URL https://arxiv.org/abs/2511.10654.

Shishir G Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023. URL https://arxiv.org/abs/2302.11529.

Suliu Qin, Haomin Zhuang, Yujun Zhou, Yufei Han, and Xiangliang Zhang. Guarding agent actions with runtime authority control (airguard). *arXiv preprint arXiv:2605.28914*, 2026. URL https://arxiv.org/abs/2605.28914.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

Peter Schafhalter, Shun Liao, Yanqi Zhou, Chih-Kuan Yeh, Arun Kandoor, and James Laudon. Scalable multi-domain adaptation of language models using modular experts. *arXiv preprint arXiv:2410.10181*, 2024. URL https://arxiv.org/abs/2410.10181.

Guanquan Shi, Haohua Du, Zhiqiang Wang, Xiaoyu Liang, Weiwenpei Liu, Song Bian, and Zhenyu Guan. Sok: Trust-authorization mismatch in llm agent interactions. *arXiv preprint arXiv:2512.06914*, 2025. URL https://arxiv.org/abs/2512.06914.

Tianyu Shi, Yang Mo, Yiou Liu, Zhuonan Hao, Yin Wang, Wenzhuo Hu, Nan Yu, Meng Zhou, and Jiangbo Yu. Organizational control layer: Governance infrastructure at the execution boundary of llm agent systems. *arXiv preprint arXiv:2606.04306*, 2026. URL https://arxiv.org/abs/2606.04306.

Munindar P Singh. Speech acts, commitment and multi-agent communication. *Computational & Mathematical Organization Theory*, 12(2-3):127–145, 2006. doi: 10.1007/s10588-006-9540-z.

Mariya I. Vasileva. Large language models can follow instructions, but not many at once: Phase transitions in compositional constraint satisfaction. *arXiv preprint arXiv:2608.12426*, 2026. URL https://arxiv.org/abs/2608.12426.

Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Yifan Zhang, Xinkui Zhao, Sai Liu, Hengxuan Lou, Guanjie Cheng, and Chang Liu. Fava: Formal authorization for verified agents with evidence-backed permission graphs. *arXiv preprint arXiv:2607.27267*, 2026. URL https://arxiv.org/abs/2607.27267.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024. URL https://arxiv.org/abs/2412.14470.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Pioneering a prospective benchmark to evaluating tool utilization safety in llms. *arXiv preprint arXiv:2509.07315*, 2025. URL https://arxiv.org/abs/2509.07315.

A I24 Dataset Statistics

Table 6 shows the I24 dataset distribution across three design dimensions: authorization expression type, signal condition, and application domain.

Table 6: I24 Dataset Distribution across Design Dimensions

Dimension	Category	Train	Test	Total
Auth Expression	Prospective	486	102	588
	Conditional	486	102	588
	Immediate	486	102	588
Signal Condition	Valid	486	102	588
	Distractor	486	102	588
	Absent	486	102	588
Domain	Messaging	365	77	442
	Publishing	365	77	442
	Payment	364	76	440
	Scheduling	364	76	440
Total		1,458	306	1,764

B Training Configuration

Table 7 lists all hyperparameters used for training the authorization tracker module.

Table 7: Hyperparameters for Authorization Tracker Training

Parameter	Value
Base model	Qwen3-8B-Instruct
Learning rate	2e-5
Batch size	32
Training epochs	3
Warmup steps	100
Weight decay	0.01
Max sequence length	2048
Optimizer	AdamW
LR schedule	Linear decay
Gradient clipping	1.0

C Implementation Details

C.1 Authorization Tracker Training

The authorization tracker takes dialogue history (recent 5 turns), candidate action, and current ledger state as input. It outputs one of four decisions (ALLOW, HOLD, RELEASE, IGNORE) plus an action identifier if applicable. Training uses cross-entropy loss for decision classification and triplet loss for action identifier matching. Training data is extracted from the I24 training set, with one sample per assistant turn.

C.2 Pending Ledger Implementation

The pending ledger uses SQLite database for storage with action identifier as primary key, along with timestamp and consumed flag. Querying employs fuzzy matching based on semantic similarity of action type and object. The ledger provides cross-session persistence for long-term tracking.

C.3 EXACTLY-ONCE EXECUTOR

The executor maintains an in-memory hash set of executed action identifiers as the consumed set. Before execution, it checks whether the identifier is in the consumed set to ensure idempotency. Transient failures may be retried, but successfully executed actions cannot be re-executed.

D ERROR CASE ANALYSIS

D.1 CASE 1: DISTRACTOR MISJUDGMENT

The tracker incorrectly interprets a distractor phrase as valid authorization:

Turn 1.
User: "Prepare an email to the team about the meeting reschedule."
Agent: `[generates email draft]`

Turn 2.
User: "Good. Don't send yet, wait for my signal."
Agent: `[HOLD(action_id=1, action=send)]`

Turn 3.
User: "Alright, that works." (other topic)
Agent: `[RELEASE(action_id=1)]`

Error. The tracker treats "that works" as authorization and sends the email, although it refers to another topic.

D.2 CASE 2: ACTION MATCHING FAILURE

The planner generates an imperfect candidate action that fails to match the pending ledger:

Turn 1.
User: "Draft payment request for 500 yuan to supplier."
Agent: `[draft_payment_request(amount=500)]`

Turn 2.
User: "Don't pay yet, wait for finance confirmation."
Agent: `[HOLD(action_id=2)]`

Turn 3.
User: "Finance confirmed, execute payment."
Planner: `[execute_payment(amount=600)]`

Error. The amount does not match pending action 2, so the tracker cannot match it and execution is missed.