

Avoidance Learning of Failure Modes Produces Robust Tool Planners

Hadi Askari¹, Japinder Singh², Tenghao Huang³

Dongwon Jung¹, Anshuman Chhabra⁴, Muhao Chen¹

¹University of California, Davis

²NVIDIA Corporation

³University of Southern California

⁴University of South Florida

Abstract

Multi-step tool planning is vulnerable to structural errors that propagate across dependent calls: a wrong argument, broken dependency, or mis-ordered operation can invalidate an otherwise plausible trajectory. We introduce FORTE, a framework that makes these failure modes explicit training targets. FORTE defines a nine-error taxonomy for multi-step plans, constructs taxonomy-grounded contrastive corpora by injecting minimally edited failures into gold trajectories, and trains a lightweight judge to recognize and score these errors. The same frozen judge then provides dense reward for GRPO planner optimization and verification for Best-of- N inference. Across ToolHop and NESTFUL, FORTE achieves the highest Judge Success Rate across all six open-weight backbone-dataset configurations and the highest observed end-task execution accuracy among the evaluated methods in all six settings. With Qwen2.5-7B, FORTE reaches 96.6 JSR and 99.5 dependency accuracy on ToolHop, and 98.6 JSR and 89.0 dependency accuracy on NESTFUL. Under a matched five-sample selection budget, FORTE-Qwen2.5-7B also outperforms GPT-5.5 across JSR, functional match, parameter accuracy, and dependency accuracy on both benchmarks. Ablations show that taxonomy-grounded contrastive negatives are essential for both judge discrimination and effective inference-time selection. These results demonstrate that explicitly learning which planning failures to avoid provides a strong supervision signal for robust multi-step tool planning.

1 Introduction

Multi-step tool planning requires LLMs to simultaneously select correct tools, bind arguments with valid types, route intermediate outputs as inputs to downstream calls, and sequence operations correctly. A single local mistake, whether a near-miss tool, a mis-typed argument, or a mis-threaded refer-

ence, silently propagates through dependent steps and breaks the entire trajectory. Even frontier models exhibit this brittleness on multi-step tool calling datasets: GPT-4o reaches only 49% accuracy on ToolHop (Ye et al., 2025) and 28% full-sequence match on NESTFUL (Basu et al., 2025). In multi-step tool plans no constraint is independent; a single error cascades forward, producing trajectories that appear syntactically well-formed but are deductively broken.

Prior work has approached robust tool planning along three broad directions. The first is *test-time search*: ReAct (Yao et al., 2022), Reflexion (Shinn et al., 2023), LATS (Zhou et al., 2023), and ToolTree (Yang et al., 2026) explore reasoning trajectories more carefully through reactive loops or tree search. The second is *structural scaffolding*: GTool (Chen et al., 2025), GNN4Plan (Wu et al., 2024c), and ToolPlanner (Wu et al., 2024a) inject dependency-graph priors at inference time. RL-based methods that fine-tune tool policies on coarse execution outcomes (Yu et al., 2025; Li et al., 2025; Wu et al., 2024b) provide a third alternative. All share an assumption that improved inference-time guidance or scalar success signals suffice; none directly trains models to recognize and avoid the specific structural failure modes that derail multi-step plans. Recent diagnostic work has begun characterizing these failure modes systematically: CriticTool (Huang et al., 2025) defines a five-type error taxonomy, while ToolScan (Kokane et al., 2024) and FAIL-TALMS (Treviño et al., 2025) catalog recurring failure patterns. However, these taxonomies remain post-hoc diagnostic instruments rather than training targets, leaving a clear opportunity: train planners and judges to be error-aware by construction.

We introduce FORTE (Figure 1), a framework that trains tool planners through failure avoidance. FORTE starts from a 9-error taxonomy for multi-step plans (Table 1) and uses it to drive a con-

FORTE *Failure-Oriented Robust Tool-planning with Error-aware training*

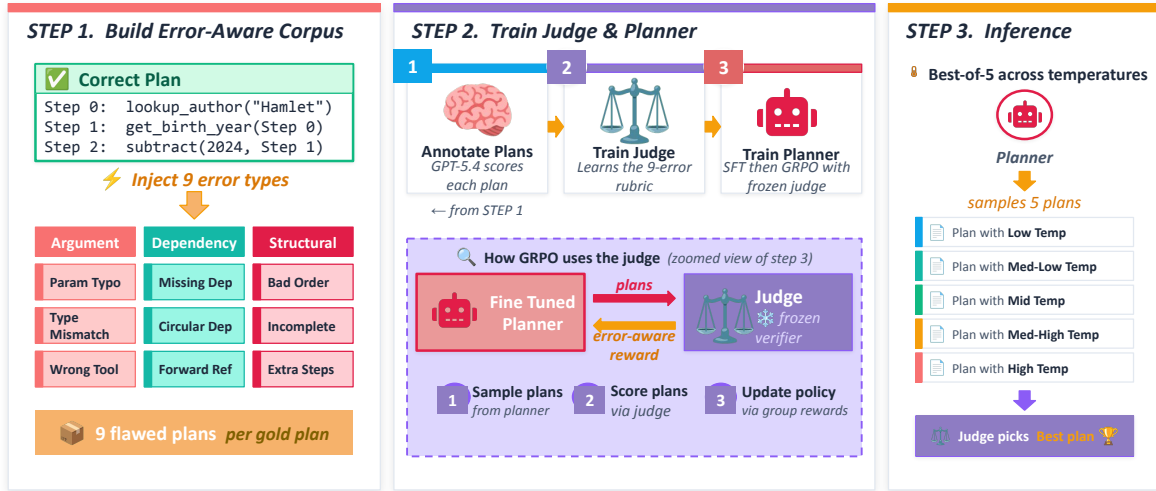


Figure 1: FORTE trains tool-planning models in three steps. We first build an error-aware corpus by injecting nine taxonomy-grounded error types into correct plans. We then annotate plans with GPT-5.4, fine-tune a judge model on the nine-error rubric, and train the planner with GRPO using the frozen judge as a dense reward signal. At inference, we sample five candidates with Best-of- N selection.

trastive data augmentation pipeline: for each gold trajectory we generate nine minimally edited negative plans, each instantiating one taxonomy error while preserving task context. This produces a dense, taxonomy-grounded supervision signal at scale without manual annotation. A lightweight judge is fine-tuned on a deductive rubric over these plan pairs, learning to score plan structure and localize specific error types. The planner is then trained with GRPO using this frozen judge as a structured reward signal, and at test time the same judge selects from a temperature-diverse Best-of- N candidate pool (Cobbe et al., 2021; Lightman et al., 2024).

The central idea behind FORTE is to make failure modes a reusable supervision signal across the entire planning pipeline. The taxonomy determines how contrastive negatives are constructed, teaches the judge to distinguish specific structural failures, supplies dense feedback during GRPO, and provides a verification signal for selecting among candidate plans at inference time. Unlike rewards based only on scalar task outcomes or similarity to a gold trajectory, the judge operates directly on $(q, \mathcal{T}, \pi)$ and can therefore score previously unseen candidate plans without requiring a reference trajectory or tool execution. This creates a unified interface between failure-aware training and inference-time verification.

Across ToolHop and NESTFUL, FORTE

achieves the highest JSR in all six open-weight backbone–dataset configurations, improving over the strongest listed baseline by 2.5–13.2 points. The gains extend beyond the learned judge: FORTE also achieves the highest observed end-task execution accuracy among the evaluated methods in all six backbone–benchmark settings, with particularly strong results on ToolHop. Additional analyses show that the judge detects compound and organically generated failures, agrees strongly with human annotations, transfers across benchmarks in a zero-shot discrimination setting, and supports competitive inference-time selection against GPT-5.5. Our main contributions are:

- A nine-error taxonomy for multi-step tool plans spanning argument-level, dependency, and plan-structure failures, including dependency/ordering categories not represented in the diagnostic frameworks compared in Table 1.
- Two augmented contrastive corpora (ToolHop and NESTFUL) that instantiate the taxonomy as minimally edited negative plans and attach deductive rubric annotations suitable for training error-aware judges and planners.
- A decoupled judge–planner design in which the same frozen, taxonomy-trained judge supplies dense GRPO reward and Best-of- N se-

lection without tool execution during training or selection.

- A comprehensive evaluation across judge-based, structural, and end-task metrics. FORTE leads JSR across all six open-weight settings, achieves the highest observed execution accuracy among the evaluated methods in all six backbone–benchmark groups, and outperforms GPT-5.5 across the reported structural metrics under a matched five-sample selection budget.

2 Related Work

Multi-step tool planning and benchmarks.

Tool-augmented LLMs have advanced from single-step function calling (Schick et al., 2023) to multi-step orchestration at scale (Qin et al., 2023). Two recent benchmarks specifically stress multi-step structure: ToolHop (Ye et al., 2025) requires multi-hop chaining across 995 queries and 3,912 tools (GPT-4o: 49% accuracy), and NESTFUL (Basu et al., 2025) targets nested API sequences where one call’s output seeds the next (full-sequence match: 28%). These results motivate methods that explicitly model cross-step dependencies rather than treating tool calls independently.

Test-time search and structural scaffolding.

Test-time methods improve robustness either by searching more trajectories or by imposing structural priors. Search-based approaches range from ReAct (Yao et al., 2022) to ToolTree (Yang et al., 2026); structural methods inject dependency information through GNNs or graph navigation (Wu et al., 2024c). These approaches primarily change inference-time behavior. FORTE instead changes the planner’s training signal so that recurring structural failures are represented explicitly during learning.

RL and contrastive training for tool use. Step-Tool (Yu et al., 2025) uses step-grained reward shaping based on invocation success and task contribution, ToolPlanner (Wu et al., 2024a) constructs SFT+RL preference data from task-completion and instruction-following scores, and AvaTaR (Wu et al., 2024b) performs contrastive reasoning over positive and negative trajectories. These works establish that standard RL or contrastive machinery can improve tool use. FORTE’s distinction is therefore the supervision representation: its negatives

are indexed by a structured error taxonomy, and the resulting judge can be reused as both a training reward and a test-time verifier.

Failure taxonomies, recovery, and concurrent planning work.

CriticTool (Huang et al., 2025), ToolScan (Kokane et al., 2024), and FAIL-TALMS (Treviño et al., 2025) catalog recurring tool-use failures and use them primarily for diagnosis or evaluation. PALADIN (Vuddanti et al., 2025) uses ToolScan-aligned failure injection for execution-time recovery: a runtime failure triggers retrieval of a recovery exemplar and re-action. FORTE instead targets generation-time avoidance by shifting the planner’s output distribution before execution. PEARL (Wang et al., 2025) is closer in its use of GRPO on ToolHop, but its positional tool-match reward is computed against a gold plan; in contrast, FORTE’s judge scores $(q, \mathcal{T}, \pi)$ without a gold trajectory, allowing the same model to evaluate unseen candidates during Best-of- N inference. Together, these distinctions position FORTE as a failure-aware supervision framework: taxonomy-grounded contrastive examples train a reusable judge whose signal connects planner optimization with inference-time verification.

3 Method

Here, we present the methodology of FORTE.

3.1 Problem Formulation

Let q denote a natural-language query and $\mathcal{T} = \{t_1, \dots, t_K\}$ a set of available tools, each described by its name, parameter schema, and return type. A *tool plan* is a sequence $\pi = \langle s_0, \dots, s_L \rangle$ where each $s_i = (v_i, t_i, \mathbf{p}_i)$ specifies an output variable, a tool invocation, and parameter bindings that may reference earlier outputs via $\{\{j\}\}$ syntax; π^* denotes the gold reference plan. The planner f_θ maps $(q, \mathcal{T})$ to a distribution over plans, and the judge g_ϕ maps $(q, \mathcal{T}, \pi)$ to a quality score $\rho \in [0, 100]$ and a predicted error label e drawn from the 9-type taxonomy defined below.

3.2 Error Taxonomy

We define nine concrete failure modes in three categories.

Argument-level errors occur within individual tool invocations. *Parameter Typo* distorts one parameter value while preserving syntactic validity. *Type Mismatch* replaces a parameter with one of an incompatible type, for example an integer where

Table 1: FORTE’s 9-error taxonomy vs. prior error-characterization and training frameworks (Huang et al., 2025; Kokane et al., 2024; Treviño et al., 2025; Yu et al., 2025; Wu et al., 2024a,b). ✓/✗ indicate whether the error type is explicitly characterized or the methodology property is present. Rows are grouped into argument-level errors (rows 1–3), multi-step dependency errors (rows 4–6), and multi-step plan structure errors (rows 7–9); all three dependency and ordering error types are absent from prior frameworks.

Error Type	Diagnostic Benchmarks			Training Methods			
	Critic Tool	Tool Scan	FAIL-TALMS	Step Tool	Tool Planner	Ava TaR	FORTE (Ours)
Type Mismatch	✗	✓	✗	✗	✗	✗	✓
Parameter Typo	✓	✓	✗	✗	✗	✗	✓
Wrong Tool	✓	✓	✗	✗	✗	✗	✓
Missing Dependency	✗	✗	✗	✗	✗	✗	✓
Circular Dependency	✗	✗	✗	✗	✗	✗	✓
Forward Reference	✗	✗	✗	✗	✗	✗	✓
Inefficient Order	✗	✗	✗	✗	✗	✗	✓
Incomplete Plan	✗	✓	✓	✗	✗	✗	✓
Unnecessary Steps	✗	✓	✗	✗	✗	✗	✓
Errors Covered	2/9	5/9	1/9	0/9	0/9	0/9	9/9
Taxonomy Defined	✓	✓	✗	✗	✗	✗	✓
Used as Training Signal	✗	✗	✗	✓	✓	✓	✓
Contrastive Training Data	✗	✗	✗	✗	✗	✓	✓
Judge Trained on Taxonomy	✗	✗	✗	✗	✗	✗	✓

a string is required. *Wrong Tool* substitutes the correct tool with a semantically adjacent but functionally incompatible alternative.

Multi-step dependency errors arise from incorrect data flow. *Missing Dependency* replaces an inter-step reference $\{\{j\}\}$ with a hardcoded literal. *Circular Dependency* introduces a reference cycle. *Forward Reference* lets an earlier step reference a step not yet executed.

Multi-step plan structure errors affect global plan organization. *Inefficient Order* permutes steps so the logical sequence is broken while individual invocations remain correct. *Incomplete Plan* truncates the plan before the final sub-question is resolved. *Unnecessary Steps* inserts a redundant tool call.

These nine types cover the recurring patterns identified in prior diagnostic work (Huang et al., 2025; Kokane et al., 2024; Treviño et al., 2025), extending coverage to the three dependency and ordering errors unique to multi-step plans (Table 1).

3.3 Contrastive Data Augmentation

Given a gold plan π^* for query q , we construct nine taxonomy-grounded negative plans $\{\pi_e^-\}_{e \in \mathcal{E}}$, one per error type, via deterministic, minimally invasive edits to π^* . Each edit targets a single step and preserves all other steps verbatim, so the negative differs from the gold in exactly the way prescribed by e .

Argument-level negatives apply a character-level perturbation (typo), substitute a type-incompatible

constant from the schema (type mismatch), or swap the tool name for the most-similar alternative in $\mathcal{T}$ (wrong tool). Dependency negatives replace $\{\{j\}\}$ with a hardcoded string (missing dependency), introduce a back-edge that closes a cycle (circular dependency), or advance a reference past the current step (forward reference). Structural negatives swap two steps with a data-flow dependency (inefficient order), drop the final step (incomplete plan), or duplicate an intermediate step (unnecessary steps).

This yields plan pairs $(q, \mathcal{T}, \pi^*, \pi_e^-, e)$ without manual annotation. Applied independently to the training splits of ToolHop and NESTFUL, the pipeline produces two dataset-specific augmented corpora that systematically cover the full error space.

3.4 Judge Training

We fine-tune a judge model g_ϕ to score plans on a deductive rubric aligned with the taxonomy. Given the query, tool schemas, and proposed plan, the judge outputs a JSON object containing a quality score $\rho \in [0, 100]$, a categorical success prediction, a natural-language reasoning trace, and a list of detected issues with their predicted error types (inference prompt in Appendix C.2).

Reference annotations. For each pair $(q, \mathcal{T}, \pi^*, \pi_e^-, e)$ we query GPT-5.4 with the inference-time rubric prompt (Appendix C.1). Starting from a base score of 100, the rubric deducts on a four-tier severity schedule: crit-

ical (−30) for circular dependencies, forward references, missing required parameters, and non-existent tools; high (−20) for hardcoded values where $\{\{j\}\}$ references are expected; medium (−10) for ordering and redundancy issues; and low (−5) for stylistic issues. The injected error type e is *not* revealed to GPT-5.4: the annotator identifies issues independently and applies the rubric, so each negative’s score reflects errors GPT-5.4 actually detects rather than a deterministic mapping from the injection label. Empirically, gold plans score 95 to 100 and negatives score 70 to 95.

Structure-only scoring and its scope. The rubric scores plan structure rather than end-task semantic correctness, and the judge is instructed never to type-check parameter values containing $\{\{j\}\}$, since these are dependency references whose run-time type is determined upstream. This decoupling makes the reward available without a tool execution environment, but it also creates a deliberate blind spot: a plan can be schema-consistent and well-threaded while implementing the wrong arithmetic or semantic decomposition. We therefore keep JSR, gold-plan structural metrics, and end-task execution separate in the evaluation and quantify the resulting syntactic–semantic gap in Section 4.5.

The judge is LoRA fine-tuned (Hu et al., 2021) from Qwen2.5-7B-Instruct on the reference annotations and used identically across all planner experiments. At inference it is served via batched vLLM as both the GRPO reward model and the Best-of- N scorer.

3.5 Planner Training

Supervised fine-tuning. We initialize the planner from a pretrained instruction-tuned base model and perform SFT on the gold plans of the relevant corpus with the prompt template in Appendix C.3. The objective is token-level cross-entropy on response tokens only. The SFT checkpoint serves as both the RL initialization and the KL reference policy.

RL with GRPO. We use Group Relative Policy Optimization (Shao et al., 2024), which removes the value network and computes advantages via group-normalized rewards: for each query we sample G candidate plans, score each with the frozen judge, and normalize the group’s rewards by their mean and standard deviation. The judge weights are frozen throughout RL, so the planner’s reward signal is determined entirely by plan structure.

3.6 Inference: Temperature-Diversity Best-of- N

At test time we generate $N = 5$ candidate plans per query and return the highest-scoring candidate. To maximize candidate diversity without sacrificing per-sample quality, we draw exactly one candidate at each temperature in $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Each candidate is scored by the judge with the same rubric used during RL training (Appendix C.2), and we return $\hat{\pi} = \arg \max_{i \in [N]} g_{\phi}(q, \mathcal{T}, \pi_i)$.

4 Experiments

Here, we expand upon our experimental setup to conduct the experiments and analyze the results.

4.1 Datasets and Evaluation

We evaluate FORTE on two complementary multi-step tool-planning benchmarks.

ToolHop (Ye et al., 2025) contains 995 multi-hop queries each requiring a 2-to-6-step tool plan over a per-query tool set of 3-to-8 tools. The benchmark stresses cross-step reasoning where each query is explicitly decomposed into sub-questions whose answers feed downstream tool calls. We hold out 100 queries for evaluation and apply the augmentation pipeline of Section 3.3 independently to each split: for every query we instantiate one error-free reference plan and nine taxonomy-grounded negatives, one per error type, yielding 1000 annotated evaluation plans that cover the full error space.

NESTFUL (Basu et al., 2025) contains 1861 multi-step instances designed to evaluate nested function calling, where outputs of one tool call are used as arguments of subsequent calls via labeled references such as `$var_1.result$`. The benchmark spans two domains: a mathematical-reasoning subset (1,415 examples drawn from MathQA covering 40 tools with an average of 5.1 tool calls per sample) and a coding subset (446 examples synthesized from StarCoder2-Instruct covering 881 distinct tools with an average of 2.1 tool calls per sample). We apply a similar augmentation pipeline as ToolHop to create our training and test sets.

Metrics. We report complementary metrics rather than treating any one score as complete task correctness. *Judge Success Rate* (JSR) is the fraction of generated plans for which the frozen judge assigns a score of at least 80. Three deterministic structural metrics compare against the gold plan: *Functional Match* (FM) requires step-count agreement, at least

Table 2: Main results on ToolHop and NESTFUL. **Best** (green shading) and second best are marked within each backbone group. JSR: Judge Success Rate (%), score ≥ 80 . FM: Functional Match (%). PA: Mean Parameter Accuracy (%). DA: Mean Dependency Accuracy (%).

Backbone LLM	Method	ToolHop				NESTFUL			
		JSR	FM	PA	DA	JSR	FM	PA	DA
Llama3.2-3B	ReAct	75.5	74.4	<u>69.8</u>	88.2	88.9	30.8	53.2	83.8
	α -UMi	0.0	0.0	20.0	11.5	17.3	0.0	26.1	9.2
	GNN4TaskPlan	14.0	12.0	25.1	51.8	12.6	2.8	29.4	58.6
	ToolPlanner	<u>83.2</u>	91.7	74.4	<u>96.5</u>	<u>95.0</u>	<u>31.4</u>	<u>59.1</u>	85.1
	FORTE (ours)	92.9	<u>80.2</u>	63.6	98.9	98.9	37.2	60.2	<u>84.0</u>
Qwen2.5-3B	ReAct	<u>92.7</u>	<u>84.7</u>	70.0	<u>93.1</u>	83.3	27.5	45.6	67.7
	α -UMi	18.0	5.0	26.3	52.6	46.7	3.9	30.3	49.5
	GNN4TaskPlan	2.0	2.0	8.9	13.9	4.7	1.7	4.7	20.4
	ToolPlanner	60.1	82.7	66.7	85.4	<u>84.5</u>	<u>29.1</u>	<u>52.3</u>	<u>79.7</u>
	FORTE (ours)	96.6	89.2	<u>69.5</u>	98.8	97.7	33.6	56.2	87.8
Qwen2.5-7B	ReAct	74.9	80.0	<u>74.6</u>	91.6	89.4	32.6	52.6	80.6
	α -UMi	29.0	21.3	29.8	49.6	67.3	15.7	42.1	51.1
	GNN4TaskPlan	23.0	25.0	30.9	59.9	24.4	5.3	28.7	57.2
	ToolPlanner	<u>87.7</u>	85.4	67.6	86.2	96.1	41.6	59.0	85.9
	FORTE (ours)	96.6	95.8	75.3	99.5	98.6	44.3	61.6	89.0

50% Jaccard keyword overlap on tool names, and at least 50% parameter accuracy; *Mean Parameter Accuracy* (PA) measures reproduced gold parameters; and *Mean Dependency Accuracy* (DA) measures correctly threaded $\{\{N\}\}$ references. These metrics capture different properties and need not move together. We additionally report *End-task Execution Accuracy* (EA) in Section 4.5, where the final answer is checked without using the judge. For NESTFUL, EA uses the benchmark’s executable functions. For ToolHop, whose original environment is interactive, EA uses an author-designed grounded simulator described below; it is not execution in the original ToolHop environment or real-tool execution.

4.2 Implementation Details

Models. We use three planner backbones: Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, and Llama3.2-3B-Instruct. The judge is always Qwen2.5-7B-Instruct fine-tuned on GPT-5.4 reference annotations, regardless of the planner.

Training. SFT runs for 4 epochs with AdamW (peak learning rate 5×10^{-5} , cosine schedule). GRPO uses group size $G = 5$, learning rate 1×10^{-6} , PPO clip $\varepsilon = 0.2$, and KL coefficient $\beta = 0.01$. We train for 4 epochs and report the checkpoint that maximizes joint performance on judge success and structural metrics on a held-out validation split. All training runs use $8 \times$ NVIDIA RTX 6000 Ada GPUs with bfloat16 mixed preci-

sion.

Inference. Best-of- N uses $N = 5$ with temperatures $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ as described in Section 3.6. Top- p is fixed at 0.9 for all candidates. The judge runs as a vLLM server with batch size 64 and is queried in parallel for the five candidates per query.

4.3 Baselines

We compare FORTE against four established multi-step tool-planning baselines. All baselines use our SFT-trained planner as their underlying generation model unless stated otherwise. **ReAct** (Yao et al., 2022) runs a 3-shot Thought-Action loop with a single generation per query. **α -UMi** (Shen et al., 2024a) uses the multi-model Planner-Caller decomposition, with both roles filled by separate instances of the non-fine-tuned instruction-tuned backbone. **GNN4TaskPlan** (Wu et al., 2024c) is a three-stage pipeline (sub-action decomposition $\rightarrow$ GraphSAGE retriever $\rightarrow$ parameter filling); we train the retriever on the training-set co-occurrence graph with BPR loss, two hard negatives per positive, for 20 epochs. **Tool-Planner** (Wu et al., 2024a) performs reward-guided beam search with a width of 3, using task-completion and instruction-following scores. This protocol grants the strongest baselines (ReAct, Tool-Planner, GNN4TaskPlan) the same SFT initialization as FORTE, so any remaining gap isolates the contribution of contrastive

judge training and GRPO from that of supervised fine-tuning alone. We also report a matched-budget comparison against GPT-5.5.

4.4 Main Results

FORTE achieves the highest judge success in every open-weight setting. Table 2 shows that FORTE obtains the best JSR across all six backbone–dataset configurations. On ToolHop, the margins over the next-best listed method are 9.7, 3.9, and 8.9 points for Llama3.2-3B, Qwen2.5-3B, and Qwen2.5-7B; on NESTFUL they are 3.9, 13.2, and 2.5 points. Because JSR is produced by the same judge family used for FORTE’s training and selection, we treat this result as evidence of structural-rubric compliance and validate end-task behavior separately in Section 4.5. Qualitative examples are in Appendix D.

Gold-plan structural metrics reveal a different picture from JSR. NESTFUL has substantially lower FM than ToolHop for every method. FORTE nevertheless obtains the best NESTFUL FM and PA within each backbone group and the best DA for both Qwen backbones. These metrics should not be collapsed into a single notion of correctness: FM penalizes deviations from the reference decomposition, PA measures argument agreement, and DA isolates dependency threading. The residual gap between these structural metrics and execution is analyzed below.

FORTE nearly saturates dependency accuracy on ToolHop. On ToolHop, FORTE achieves the best DA for every backbone, with scores of 98.9, 98.8, and 99.5. This shows that the generated plans correctly preserve dependencies across multi-step tool calls. Since dependency errors can make otherwise plausible plans non-executable, these gains are important for reliable tool-use planning.

Prior planning baselines are less robust. ToolPlanner and ReAct are competitive in isolated cases, but their performance varies across backbones and datasets. For example, ToolPlanner achieves strong FM on Llama3.2-3B ToolHop, but FORTE dominates JSR across all settings and wins most structural metrics on NESTFUL. Meanwhile, α -UMi and GNN4TaskPlan perform poorly in several configurations; we report these results without attributing the gap to a single mechanism because the baselines differ in both planning structure and inference procedure.

FORTE benefits from stronger backbone models. The strongest structural results are obtained with Qwen2.5-7B, where FORTE wins all four metrics within its backbone group on both ToolHop and NESTFUL. On ToolHop, it reaches 96.6 JSR, 95.8 FM, 75.3 PA, and 99.5 DA; on NESTFUL, it reaches 98.6 JSR, 44.3 FM, 61.6 PA, and 89.0 DA. This indicates that FORTE scales positively with backbone capacity rather than saturating at the 3B scale.

FORTE outperforms GPT-5.5 under matched five-sample selection. We compare FORTE-Qwen2.5-7B with GPT-5.5 under the same five-sample inference budget, using the frozen FORTE judge to select the highest-scoring GPT-5.5 candidate. On ToolHop, FORTE achieves 96.6/95.8/75.3/99.5 JSR/FM/PA/DA compared with 86.7/29.6/36.8/83.2 for GPT-5.5; on NESTFUL, the corresponding scores are 98.6/44.3/61.6/89.0 versus 91.8/35.9/53.1/81.4. Thus, FORTE remains ahead across all four reported metrics on both benchmarks despite using a substantially smaller open-weight planner.

The matched GPT-5.5 samples also reveal an important distinction between judge score and gold-plan agreement. Selecting the highest-scoring candidate increases GPT-5.5 JSR from 70.4 to 86.7 on ToolHop and from 85.9 to 91.8 on NESTFUL, while FM, PA, and DA decrease modestly. This confirms that the metrics capture complementary properties of plan quality rather than interchangeable notions of correctness. Full matched-run results are reported in Appendix B.6.

4.5 Judge-Independent End-Task Execution

We complement judge-based and gold-plan structural metrics with end-task execution accuracy, measuring whether a generated plan ultimately produces the correct benchmark answer. The two benchmarks provide different execution settings. **NESTFUL** plans are evaluated using the benchmark-provided executable functions over 141 executable queries. **ToolHop** is evaluated over 1,000 generated plans from 100 held-out queries (10 plans per query) using a grounded simulator that validates tool choice, argument provenance, dependency threading, ordering, skipped hops, forward and circular references, and plan completeness before returning benchmark sub-answers. The grounded protocol achieves a 99% ceiling on the 100 ToolHop gold plans; additional harness valida-

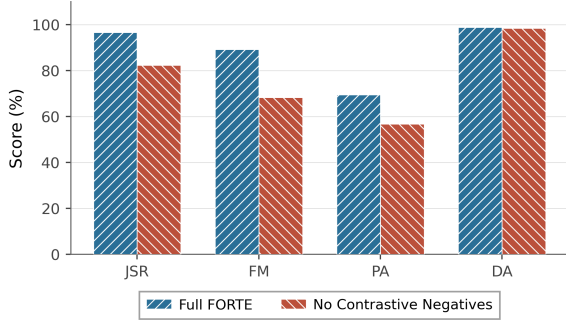


Figure 2: Effect of removing taxonomy-grounded contrastive negatives from judge training.

tion is reported in Appendix B.

FORTE achieves the highest observed execution accuracy among the evaluated methods in all six backbone–benchmark groups. The ToolHop gains are substantial: FORTE reaches 92.7, 97.7, and 97.8% across the three backbones, compared with the strongest baseline scores of 85.0, 88.0, and 80.0%. On NESTFUL, the improvements are smaller but consistent, culminating in 48.2% execution accuracy for Qwen2.5-7B compared with 46.8% for ReAct. These results show that FORTE’s gains extend from structural plan quality to end-task behavior.

The difference between benchmarks also highlights the complementary role of execution evaluation. Among judge-passing Qwen2.5-7B plans, 97.7% execute successfully under the grounded ToolHop protocol, compared with 48.2% on NESTFUL. NESTFUL therefore remains substantially more challenging at the semantic and arithmetic decomposition level even when plan structure is well formed.

5 Discussion

Contrastive negatives are load-bearing.

Retraining the judge on perfect plans only (quality_score=100) degrades the full Qwen2.5-3B/ToolHop system: JSR drops 14.3 points, FM 20.9, and PA 12.8 (Figure 2); the Best-of- N gain also inverts to -2.6 points and the ablated judge labels all selected plans successful. Thus, exposure to flawed plans supplies both the discrimination needed for the GRPO reward and the discrimination needed for inference-time selection.

Structure-only verification is useful but incomplete. JSR, FM/PA/DA, and execution measure different properties. The distinction is most visible

on NESTFUL: among judge-passing Qwen2.5-7B plans, only 48.2% execute to the correct answer, compared with 97.7% under the grounded ToolHop protocol. Many NESTFUL plans are schema-consistent yet implement an incorrect arithmetic decomposition, which a structure-only judge cannot detect. This limitation is the price of making training and Best-of- N selection independent of tool execution.

Failure awareness extends beyond single injected errors.

The learned judge generalizes beyond the single-error negatives used for training. Detection increases from single-error to two- and three-error compositions on both benchmarks, reaching 95.0% on three-error ToolHop plans and 92.2% on three-error NESTFUL plans. On naturally generated ToolHop failures from ReAct, α -UMi, GNN4TaskPlan, and ToolPlanner, the judge flags 86.3% of FM failures and 85.2% of grounded-execution failures. Human evaluation further supports the learned rubric: two annotators agree on 97% of binary judgments ($\kappa = 0.93$), while agreement between each human and the GPT-derived rubric reaches 90–91%. Finally, a NESTFUL-trained judge transfers to ToolHop gold-versus-negative discrimination with AUC 0.860 without additional training. Appendix B provides the full results.

6 Conclusion

We introduced FORTE, a failure-aware framework for multi-step tool planning that turns an explicit taxonomy of structural errors into a shared supervision signal for judge training, planner optimization, and inference-time verification. Across three open-weight backbones and two benchmarks, FORTE achieves the highest JSR in all six settings and the highest observed end-task execution accuracy among the evaluated methods in all six backbone–benchmark groups. Its strongest Qwen2.5-7B planner reaches 96.6 JSR and 99.5 dependency accuracy on ToolHop and 98.6 JSR and 89.0 dependency accuracy on NESTFUL, while also outperforming GPT-5.5 across all reported metrics under a matched five-sample selection budget.

Ablations show that taxonomy-grounded contrastive negatives are essential to both judge discrimination and Best-of- N selection, while compound-error, organic-failure, human-agreement, and transfer analyses show that the resulting judge captures recurring structural failure

Table 3: Judge-independent end-task execution accuracy (%). ToolHop reports per-plan strict accuracy over 1,000 generated plans from 100 held-out queries (10 plans/query) using the author-designed grounded protocol; NESTFUL uses benchmark executable functions on 141 executable queries.

Method	ToolHop grounded			NESTFUL executable		
	Llama3.2-3B	Qwen2.5-3B	Qwen2.5-7B	Llama3.2-3B	Qwen2.5-3B	Qwen2.5-7B
ReAct	77.0	88.0	76.0	14.2	23.3	46.8
α -UMi	5.0	41.0	49.0	0.7	5.7	22.7
GNN4TaskPlan	14.0	5.0	27.0	0.0	0.0	2.1
ToolPlanner	85.0	51.0	80.0	16.3	13.5	38.3
FORTE	92.7	97.7	97.8	23.6	23.4	48.2

patterns beyond individual training examples. Taken together, our results show that explicitly teaching a planner which failures to avoid provides a powerful and reusable supervision signal for robust multi-step tool use.

Limitations

FORTE has several limitations. First, the full pipeline remains dataset-specific: the augmented corpus, judge training, and planner fine-tuning are performed separately per benchmark. Our transfer experiment is limited to a NESTFUL-trained judge scoring ToolHop gold-versus-injected-negative discrimination (AUC 0.860 versus 0.949 for the ToolHop in-domain judge). We do not report the reverse direction, zero-shot planner transfer, or complete-pipeline transfer to a new tool domain.

Second, the judge is intentionally structure-only. This makes training and Best-of- N selection execution-free, but a high JSR is not evidence of semantic task correctness. The execution study makes this gap explicit: among judge-passing Qwen2.5-7B plans, 97.7% execute correctly under our grounded ToolHop protocol, but only 48.2% do so on NESTFUL’s executable functions (ReAct: 86.6% and 46.5%). The NESTFUL failures include semantically incorrect arithmetic decompositions that remain schema-consistent. In addition, ToolHop execution is measured with an author-designed grounded simulator that checks tool choice, argument provenance, dependencies, ordering, and completeness and returns benchmark gold sub-answers; it is not execution in the original interactive ToolHop environment and not real-tool execution.

Third, our robustness tests do not establish unrestricted generalization. Compound errors are created by composing the same taxonomy-specific injectors used to generate single-error training negatives. The organic-error analysis covers ToolHop

failures only and does not measure false-positive rates on naturally generated valid plans. Long-range semantic failures, browser/GUI agents, stateful environments, and other tool domains may contain failure modes outside our nine categories.

Fourth, the annotation pipeline relies on GPT-5.4 applying a fixed deductive rubric, so annotation bias or label noise can propagate into the learned judge. Our 100-plan blinded study shows high agreement, but it uses only two author annotators on a taxonomy-stratified sample rather than independent external annotators. Finally, all headline metrics measure different properties: JSR reflects the learned structural rubric, FM/PA/DA compare to a reference decomposition, and execution measures final-answer success under benchmark-specific protocols. We therefore avoid treating any single metric as a complete measure of planning correctness.

References

- Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, and 1 others. 2025. Nestful: A benchmark for evaluating llms on nested sequences of api calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33526–33535.
- Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.
- Jinpeng Chen, Cheng Gong, Hanbo Li, Ziru Liu, Zichen Tian, Xinyu Fu, Shi Wu, Chenyang Zhang, Wu Zhang, Suiyun Zhang, and 1 others. 2026. Cove: Training interactive tool-use agents via constraint-guided verification. *arXiv preprint arXiv:2603.01940*.

- Wenjie Chen, Wenbin Li, Di Yao, Xuying Meng, Chang Gong, and Jingping Bi. 2025. Gtool: Graph enhanced tool planning with large language model. *arXiv preprint arXiv:2508.12725*.
- Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.
- Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Minsu Jang, Dohyung Kim, Jaehong Kim, and Youngwoo Yoon. 2025. Reactree: Hierarchical llm agent trees with control flow for long-horizon task planning. *arXiv preprint arXiv:2511.02424*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.
- Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.
- Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Xiong Hao, and Wenliang Chen. 2025. Nestools: A dataset for evaluating nested tool learning abilities of large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9824–9844.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Shiting Huang, Zhen Fang, Zehui Chen, Siyu Yuan, Junjie Ye, Yu Zeng, Lin Chen, Qi Mao, and Feng Zhao. 2025. Critictool: Evaluating self-critique capabilities of large language models in tool-calling error scenarios. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26683–26692.
- Tenghao Huang, Dongwon Jung, Vaibhav Kumar, Mohammad Kachuee, Xiang Li, Puyang Xu, and Muhao Chen. 2024. Planning and editing what you retrieve for enhanced tool learning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 975–988, Mexico City, Mexico. Association for Computational Linguistics.
- Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, and 1 others. 2024. Toolscan: A benchmark for characterizing errors in tool-use llms. *arXiv preprint arXiv:2411.13547*.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.
- Zhiwei Li, Yong Hu, and Wenqing Wang. 2025. Encouraging good processes without the need for good answers: Reinforcement learning for llm agent planning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1654–1666.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*.
- Xukun Liu, Zhiyuan Peng, Xiaoyuan Yi, Xing Xie, Lirong Xiang, Yuchen Liu, and Dongkuan Xu. 2024. Toolnet: Connecting large language models with massive tools via tool graph. *arXiv preprint arXiv:2403.00839*.
- Elias Lumer, Pradeep Honaganahalli Basavaraju, Myles Mason, James A Burke, and Vamse Kumar Subbiah. 2025. Graph rag-tool fusion. *arXiv preprint arXiv:2502.07223*.
- Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. 2024. m & m’s: A benchmark to evaluate tool-use for multi-step multi-modal tasks. In *European Conference on Computer Vision*, pages 18–34. Springer.
- Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical

- reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024a. [Small LLMs are weak tool learners: A multi-LLM agent](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA. Association for Computational Linguistics.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugging-gpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.
- Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024b. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.
- Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. 2024. [Tools fail: Detecting silent errors in faulty tools](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14272–14289, Miami, Florida, USA. Association for Computational Linguistics.
- Eduardo Treviño, Hugo Contant, James Ngai, Graham Neubig, and Zora Zhiruo Wang. 2025. Benchmarking failures in tool-augmented language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2916–2934.
- Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chitiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. 2025. Paladin: self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*.
- Qihao Wang, Mingzhe Lu, Jiayue Wu, Yue Hu, and Yanbing Liu. 2025. Pearl: Plan exploration and adaptive reinforcement learning for multihop tool use. In *Pacific Rim International Conference on Artificial Intelligence*, pages 639–651. Springer.
- Cailin Winston and René Just. 2025. A taxonomy of failures in tool-augmented llms. In *2025 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 125–135. IEEE.
- Qinzhao Wu, Wei Liu, Jian Luan, and Bin Wang. 2024a. Toolplanner: A tool augmented llm for multi granularity instructions with path planning and feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18315–18339.
- Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024b. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010.
- Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and 1 others. 2024c. Can graph learning improve planning in llm-based agents? *Advances in Neural Information Processing Systems*, 37:5338–5383.
- Shuo Yang, Soyeon Caren Han, Yihao Ding, Shuhe Wang, and Eduard Hoy. 2026. [Tooltree: Efficient llm agent tool planning via dual-feedback monte carlo tree search and bidirectional pruning](#). *Preprint*, arXiv:2603.12740.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.
- Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, and 1 others. 2025. Toolhop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2995–3021.
- Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2025. Steptool: Enhancing multi-step tool usage in llms via step-grained reinforcement learning. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 3952–3962.
- Shiqi Zhang, Xinbei Ma, Zouying Cao, Zhuosheng Zhang, and Hai Zhao. 2025. Plan-over-graph: Towards parallelable llm agent schedule. *arXiv preprint arXiv:2502.14563*.
- Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.
- Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel,

and Chao Zhang. 2023. Toolchain*: Efficient action space navigation in large language models with a* search. *arXiv preprint arXiv:2310.13227*.

Appendix

A Extended Related Work

We elaborate here on the four families of work surveyed in Section 2, covering papers omitted from the main text for space.

Additional multi-step tool planning benchmarks.

The broader single-step function-calling literature includes Gorilla (Patil et al., 2024) and Huang et al. (2024). Beyond ToolBench/ToolLLM, API-Bank (Li et al., 2023) established large-scale training and evaluation settings, while Hugging-GPT (Shen et al., 2023) and TaskBench (Shen et al., 2024b) demonstrated tool selection over structured dependency graphs. NesTools (Han et al., 2025) complements NESTFUL with four fine-grained evaluation dimensions including nested parameter filling; m&m (Ma et al., 2024) introduces 4K+ multi-step multimodal tasks across 33 tools; and T-Eval (Chen et al., 2024) decomposes multi-step performance into planning, retrieval, and review sub-scores.

Additional test-time search and structural scaffolding methods.

Beyond ReAct and ToolTree, Reflexion (Shinn et al., 2023) added iterative verbal self-refinement, ToT (Yao et al., 2023) introduced tree-of-thought reasoning, and ToolChain* (Zhuang et al., 2023) applied A* search over tool sequences. ReAcTree (Choi et al., 2025) further extends search-based methods by replacing flat trajectories with a dynamically expanding agent tree, using behavior-tree control flow to coordinate subgoal-specialized agents and extending ReAct and least-to-most prompting into hierarchical planning. On the structural scaffolding side, GTool (Chen et al., 2025) and ToolNet (Liu et al., 2024) encode inter-tool dependency graphs via GNNs or directed graph navigation; Graph RAG-Tool Fusion (Lumer et al., 2025) combines retrieval with depth-first graph traversal; and Plan-over-Graph (Zhang et al., 2025) enables parallelizable agent scheduling.

Additional RL and contrastive training methods.

RLTR (Li et al., 2025) uses tool-use completeness rewards with distillation-based cold-start. Tool-Star (Dong et al., 2025) combines cold-start fine-tuning with hierarchical multi-level RL across six tool types. CoVe (Chen et al., 2026) uses deterministic task-specific constraints as verifiable reward signals for SFT and RL. As with the meth-

ods discussed in the main text, supervision in each of these frameworks ultimately reduces to coarse execution outcomes rather than structured error labels.

Additional failure taxonomies and diagnostic benchmarks. A taxonomy of failures in tool-augmented LLMs (Winston and Just, 2025) surfaces dependency, specification, and state-desynchronization errors across Gorilla and Chameleon deployments. Sun et al. (2024) specifically study silent errors from malformed tool outputs. At the multi-agent level, MASFT (Cemri et al., 2025) identifies 14 failure modes across five open-source systems, finding that prompt-engineering interventions yield only modest gains, reinforcing the case for training-time solutions.

B Validation Experiments

Here we present additional validation experiments for the framework.

B.1 Execution Harness Validation

For NESTFUL, we execute plans against the benchmark-provided functions. For ToolHop, we use the grounded simulator described in Section 4.5. The simulator validates argument provenance, dependency structure, ordering, and completeness before returning the benchmark’s gold sub-answer.

The NESTFUL harness succeeds on all 141 executable gold plans. Under the grounded ToolHop protocol, 99 of 100 gold plans execute successfully. None of the seeded circular-dependency, forward-reference, incomplete-plan, or mis-ordered negatives pass the corresponding validation checks.

The ToolHop execution results in Table 3 aggregate 1,000 generated plans over 100 held-out queries (10 plans per query); the 99% gold-plan ceiling reported here instead evaluates one gold plan for each of the 100 held-out queries.

B.2 Compound-Error Stress Test

The judge is trained on single-error negatives only. We create held-out two- and three-error plans by composing taxonomy injectors at random with a fixed seed and verify that each injection changes the serialized plan. A plan is counted as detected when the judge score is below 80; “ Δ gold” is the difference between the gold mean judge score and the row mean.

Plans	n	Score	Det.	Δ gold
ToolHop gold	100	96.9	–	–
ToolHop 1 error	874	72.1	78.7%	24.8
ToolHop 2 errors	300	68.1	93.3%	28.8
ToolHop 3 errors	200	67.8	95.0%	29.1
NESTFUL gold	180	98.4	–	–
NESTFUL 1 error	1501	78.5	61.4%	19.9
NESTFUL 2 errors	540	72.9	81.7%	25.5
NESTFUL 3 errors	360	70.8	92.2%	27.7

Detection increases with error count on both benchmarks, but these examples remain compositions of known injectors; they do not establish robustness to unseen semantic error families.

B.3 Organic ToolHop Failures

We pool plans generated by ReAct, α -UMi, GNN4TaskPlan, and ToolPlanner rather than applying synthetic edits. Under a judge-independent FM-failure criterion, 6,248 failed plans receive mean judge score 73.2 and 86.3% are flagged below 80, compared with mean score 96.9 for gold plans. Under the grounded execution-failure criterion, 6,010 failures receive mean score 73.6 and 85.2% are flagged.

B.4 Human Agreement Study

Two authors independently annotate 100 blinded plans (50 ToolHop, 50 NESTFUL), stratified over gold plans and the nine taxonomy error types. GPT scores and issue lists are hidden. Annotators record binary accept/reject at the score-80 threshold and a primary error type.

Agreement	Binary (% / κ)	Type (% / κ)
Human vs. Human	97.0 / 0.93	90.0 / 0.89
Human A vs. GPT rubric	91.0 / 0.76	79.0 / 0.69
Human B vs. GPT rubric	90.0 / 0.74	73.0 / 0.67

On ToolHop alone the two humans agree on all 50 binary decisions ($\kappa = 1.0$).

B.5 One-Way Judge Transfer

We evaluate whether a judge trained on one benchmark discriminates gold from injected negative plans on the other benchmark without additional training.

Judge $\rightarrow$ scored dataset	AUC	Gold pass
ToolHop $\rightarrow$ ToolHop (in-domain)	0.949	92.0%
NESTFUL $\rightarrow$ NESTFUL (in-domain)	0.863	95.6%
NESTFUL $\rightarrow$ ToolHop (transfer)	0.860	95.0%

B.6 Matched-Budget GPT-5.5 Details

GPT-5.5 is sampled five times and each candidate is scored by the same frozen judge used for FORTE’s

Best-of-5 selection. The API rejected the temperature parameter, so these are five independent draws rather than the temperature-diverse schedule used for the open-weight planners. We report the first sample and Best-of-5 result from the *same run and same sampled candidate sets*: 98 ToolHop examples and 170 NESTFUL examples. We report the first sample and Best-of-5 result from the same run and the same sampled candidate sets: 98 ToolHop examples and 170 NESTFUL examples. This provides a controlled comparison of single-sample and judge-selected inference under an identical sampling budget.

ToolHop ($n = 98$)				
Method	JSR	FM	PA	DA
GPT-5.5, first sample	70.4	35.7	41.7	87.9
GPT-5.5 + Best-of-5 + judge	86.7	29.6	36.8	83.2
FORTE-Qwen2.5-7B	96.6	95.8	75.3	99.5

NESTFUL ($n = 170$)				
Method	JSR	FM	PA	DA
GPT-5.5, first sample	85.9	37.6	53.9	81.7
GPT-5.5 + Best-of-5 + judge	91.8	35.9	53.1	81.4
FORTE-Qwen2.5-7B	98.6	44.3	61.6	89.0

Within GPT-5.5, judge selection raises JSR by 16.3 points on ToolHop and 5.9 points on NESTFUL, but lowers FM/PA/DA on the identical sample sets. Mean judge score also rises from 90.5 to 95.7 on ToolHop and from 95.7 to 97.5 on NESTFUL. This reinforces the metric framing in Section 4.5: a higher judge score reflects better compliance with the learned structural rubric, not a guarantee of improved agreement with the gold decomposition or semantic correctness.

C Prompts

This appendix collects the prompts used by FORTE verbatim from the released codebase. There are three distinct prompts: the *annotation prompt* (§C.1) used by GPT-5.4 to generate training labels for the judge; the *judge inference prompt* (§C.2) sent to the trained judge during GRPO reward computation and Best-of- N selection; and the *planner prompt* (§C.3) used identically for SFT, GRPO, and BoN inference.

C.1 Annotation Prompt

GPT-5.4 is prompted with the deductive rubric below, which assigns a base score of 100 and subtracts points per identified syntactic error according to a four-tier severity scale (critical −30, high −20, medium −10, low −5). The rubric explicitly forbids the judge from flagging semantic issues or

type-checking parameters containing dependency references $\{\{N\}\}$.

System message : annotation judge

You are an expert at evaluating multi-step tool execution plans. You only check syntactic correctness, not semantic correctness. NEVER type-check parameters containing $\{\{N\}\}$ - they are dependency references and are always type-valid.

User message : annotation judge

You are evaluating a multi-step tool execution plan using an OBJECTIVE ERROR-BASED RUBRIC.

```
**USER QUERY:** {query}
**AVAILABLE TOOLS:** {tools_desc}
**PROPOSED PLAN:** {plan_desc}
**GROUND TRUTH PLAN (for reference):** {ground_truth_desc}
```

OBJECTIVE SCORING RUBRIC

SCORING FORMULA:

1. Start with a base score of 100.
2. Identify ALL errors in the plan.
3. For EACH error, deduct points based on severity:
 - CRITICAL error: -30 points
 - HIGH error: -20 points
 - MEDIUM error: -10 points
 - LOW error: -5 points
4. Final score = $\max(0, 100 - \text{total_deductions})$

SYNTAX-ONLY EVALUATION

Check ONLY syntactic correctness:

- Are all required parameters provided?
- Are LITERAL parameter types correct (skip values containing $\{\{N\}\}$)?
- Are dependencies valid (no forward refs, no circular deps)?
- Are all steps present to complete the task?
- Is the tool name spelled correctly?

Do NOT check semantics:

- Whether tool outputs will work together
- Whether the plan produces the correct final answer
- Whether tool choice "makes sense" for the task
- Whether parameter VALUES produce the right result

ABSOLUTE RULE: any parameter value containing {{N}} is a dependency reference and is ALWAYS type-valid. Skip type checking entirely. Quotes around {{N}} do NOT make it a literal string.

ERROR SEVERITY DEFINITIONS

CRITICAL (-30): missing required param; circular dependency; non-existent tool; forward reference; literal-value type mismatch; missing critical step.
HIGH (-20): hardcoded value where {{N}} expected; incorrect parameter value changing output significantly; missing non-critical step.
MEDIUM (-10): inefficient ordering (works but suboptimal); typo in parameter value; unnecessary step that doesn't break the plan.
LOW (-5): minor formatting; non-standard but functional format; trivial inefficiency; style issues.

SUCCESS PREDICTION MAPPING:

90-100 -> "yes" 75-89 -> "likely_yes"
50-74 -> "uncertain" 25-49 -> "likely_no"
0-24 -> "no"

YOUR TASK

1. IDENTIFY ALL ERRORS step by step.
2. CLASSIFY SEVERITY using the rubric above.
3. CALCULATE SCORE: start at 100, deduct per error.
4. SHOW YOUR WORK in the "reasoning" field.

RESPONSE FORMAT (JSON only, no markdown):

```
{
  "quality_score": <0-100>,
  "success_prediction": "<yes|likely_yes|uncertain|likely_no|no>",
  "confidence": <0.0-1.0>,
  "reasoning": "Base score: 100\nError 1: [severity] description (-X)\n...\nFinal score: Z",
  "issues": [
    {
      "type": "type_mismatch|missing_dependency|wrong_tool|parameter_typo|circular_dependency|inefficient_order|incomplete_plan|unnecessary_steps|forward_reference|other",
      "severity": "critical|high|medium|low",
      "step": <step_number or null>,
      "description": "Specific description of the error",
      "suggestion": "How to fix this error",
      "points_deducted": <30|20|10|5>
    }
  ]
}
```

```
}
]
```

C.2 Judge Inference Prompt

At RL training time, the trained judge receives the prompt below to produce a reward signal for GRPO; at inference time, the same prompt is used to score each of the N candidate plans for Best-of- N selection. At RL training time, the trained judge receives the prompt below to produce a reward signal for GRPO; at inference time, the same prompt is used to score each of the N candidate plans for Best-of- N selection. The output schema is aligned with the annotation format used to fine-tune the judge.

System message : inference judge

You are an expert judge for evaluating tool execution plans. Your task is to:

1. Analyze the plan's correctness and efficiency
2. Assign a quality score (0-100)
3. Predict success likelihood (yes/likely_yes/uncertain/likely_no/no)
4. Identify specific issues with severity levels
5. Provide detailed reasoning

Scoring guidelines:

- 100: Perfect execution, no errors
- 80-99: Minor issues, likely to succeed
- 60-79: Moderate issues, uncertain outcome
- 40-59: Major issues, likely to fail
- 0-39: Critical errors, will fail

User message : inference judge

Query: {query}

{tools_desc}

Plan to Evaluate:
{plan_str}

Please evaluate this plan and provide:

1. Quality score (0-100)
2. Success prediction (yes/likely_yes/uncertain/likely_no/no)
3. Detailed reasoning
4. List of issues (if any)
5. Confidence (0.0-1.0)

Format your response as JSON:

```
{
  "quality_score": <int>,
  "success_prediction": "<string>",
  "reasoning": "<string>",
  "issues": [...],
  "confidence": <float>
}
```

```
}
```

C.3 Planner Prompt

The same prompt template is used for SFT, GRPO, and inference, avoiding a prompt-template mismatch across training and deployment. The user message instructs the planner to use $\{\{N\}\}$ variables for dependency threading and the exact parameter names from the tool signatures.

System message : planner

You are an expert at creating multi-step tool execution plans. Given a query and available tools, generate a correct sequence of tool calls to answer the query.

User message : planner

Generate a tool execution plan to answer this query.

Query: {question}

{tools_desc}

Generate a step-by-step plan using the available tools. Each step should:

1. Call exactly one tool
2. Use output variables $\{\{0\}\}$, $\{\{1\}\}$, $\{\{2\}\}$, etc. for results
3. Reference previous step outputs using $\{\{N\}\}$ - never substitute a hardcoded value for an output that comes from a prior step
4. Use the exact parameter names shown in the tool signatures above
5. Provide all required parameters

Generate only the steps the query requires - no redundant steps, no missing steps.

Format each step as: Step N: $\{\{N\}\}$ = tool_name(param1=value1, param2=value2, ...)

D Qualitative Examples

We present five qualitative examples drawn from our held-out evaluation: two perfect plans (one per benchmark), one example of FORTE producing a correct plan despite the reference being seeded with a wrong-tool error, one residual failure that exposes a secondary failure mode, and one closed-source comparison illustrating GPT-5.5's hallucinated parameters and compressed step counts.

Example 1: ToolHop : complete multi-hop plan with full dependency threading (judge=100).

ToolHop query 1 : FORTE-Qwen2.5-7B, $T=1.0$

QUERY:

What is the sum of the (unique) prime factors of the year when the actress that plays Kaylee Frye in the series "Firefly" also stars with Alan Dale in a sci-fi tv film released in?

GENERATED PLAN:

Step 0: $\{\{0\}\}$ = character_actor_lookup(character_name='Kaylee Frye', series_name='Firefly', exact_match=True, case_sensitive=False)
Step 1: $\{\{1\}\}$ = film_release_year_finder(actors=[$\{\{0\}\}$, 'Alan Dale'], genre='sci-fi')
Step 2: $\{\{2\}\}$ = prime_factorization(number= $\{\{1\}\}$, output_format='unique_list', algorithm='general_purpose')
Step 3: $\{\{3\}\}$ = sum_digits(number= $\{\{2\}\}$, output_format='number')

JUDGE: score=100, prediction="yes", confidence=0.98.

All three intra-plan references ($\{\{0\}\}$, $\{\{1\}\}$, $\{\{2\}\}$) are correctly threaded; step count, tool selection, and dependency accuracy all reach 1.0 against the perfect reference.

Example 2: NESTFUL : exact match to the gold plan (judge=100).

NESTFUL query 5 : FORTE-Qwen2.5-7B, $T=0.2$

QUERY:

Find the unique common values in two lists [1, 2, 3, 4, 5] and [4, 5, 6, 7, 8]. Then, swap the values at index 0 and 1 in the resulting list.

GENERATED PLAN:

Step 0: $\{\{0\}\}$ = unique_common_values(list1=[1, 2, 3, 4, 5], list2=[4, 5, 6, 7, 8])
Step 1: $\{\{1\}\}$ = swap_by_index(values= $\{\{0\}\}$, i=0, j=1)

JUDGE: score=100, prediction="yes", confidence=0.99.

The lowest-temperature candidate suffices; all four structural metrics (exact match, functional match, parameter accuracy, dependency accuracy) equal 1.0.

Example 3: Wrong-tool robustness (NESTFUL).

NESTFUL query 5 : reference seeded with wrong_tool

QUERY: (same as Example 2)

REFERENCE PLAN (deliberately flawed : wrong_tool):

```
Step 0: {{0}} = unique_common_values(list1=[1, 2, 3, 4, 5],
list2=[4, 5, 6, 7, 8])
Step 1: {{1}} = non_whitespace_substrings(values={{0}}, i=0, j=1)
```

\textsc{Forte} GENERATED PLAN (\$T{=}0.2\$):

```
Step 0: {{0}} = unique_common_values(list1=[1, 2, 3, 4, 5],
list2=[4, 5, 6, 7, 8])
Step 1: {{1}} = swap_by_index(values={{0}}, i=0, j=1)
```

JUDGE: score=100, prediction="yes", confidence=0.99.

Although the dataset’s reference for this query was seeded with the wrong tool (non_whitespace_substrings), FORTE generates the correct swap_by_index call directly from the query, demonstrating that exposure to wrong-tool negatives during training does not bias the planner toward reproducing them.

Example 4: Residual failure : step-count compression on NESTFUL (judge=100, FM fails).

NESTFUL query 34 : FORTE-Qwen2.5-7B, $T=0.4$

QUERY:

If n is a prime number greater than 5, what is the remainder when n^2 is divided by 12?

GOLD PLAN (4 steps):

```
Step 0: {{0}} = add(arg_0=5, arg_1=2)
Step 1: {{1}} = power(arg_0={{0}}, arg_1=2)
Step 2: {{2}} = multiply(arg_0=12, arg_1=4)
Step 3: {{3}} = subtract(arg_0={{1}}, arg_1={{2}})
```

\textsc{Forte} GENERATED PLAN (3 steps, all 5 candidates scored 100):

```
Step 0: {{0}} = multiply(arg_0=4, arg_1=3)
Step 1: {{1}} = divide(arg_0={{0}}, arg_1=2)
Step 2: {{2}} = power(arg_0={{1}}, arg_1=2)
```

JUDGE: score=100, prediction="yes", confidence=0.99.

STRUCTURAL EVAL: step_count_match=False (3 vs 4),
exact_tool_accuracy=0.0,
functional_match=False.

This is the dominant NESTFUL FM failure mode: the planner compresses the 4-step gold decomposition into a 3-step solution composed of com-

pound arithmetic calls, which the rubric-based judge correctly scores as syntactically valid but which the structural evaluator marks as a functional-match failure due to the step-count mismatch. The judge’s deductive rubric does not verify that the compressed plan computes the same intermediate values as the gold, exposing the syntactic-only scoring limitation discussed in the Limitations section.

Example 5: FORTE vs GPT-5.5 on the same ToolHop query.

ToolHop query 1 : GPT-5.5 (closed-source frontier, $T=0.0$)

QUERY: (same as Example 1)

GPT-5.5 GENERATED PLAN:

```
Step 0: {{0}} = character_actor_lookup(
character_name="Kaylee Frye",
series_name="Firefly", exact_match=
true)
Step 1: {{1}} = film_release_year_finder(
actors=[{{0}}, "Alan Dale"],
genre="sci-fi", additional_filters={"
format": "tv film"})
Step 2: {{2}} = prime_factorization(number=
{{1}},
output_format="
sum_unique_prime_factors")
```

JUDGE: score=70, prediction="uncertain", confidence=0.78.

Step count: 3 vs gold’s 4 (step_count_match=False).

GPT-5.5 exhibits two of the failure modes FORTE’s training data systematically penalises: (i) it hallucinates an additional_filters parameter that does not appear in the film_release_year_finder signature, and an output_format="sum_unique_prime_factors" enum value that does not exist; (ii) it compresses the plan from four steps to three, folding the final sum_digits call into a non-existent prime_factorization output format. FORTE’s plan for the same query (Example 1) issues four schema-compliant steps and scores 100.

E Code and Reproducibility

We release the augmented ToolHop and NESTFUL contrastive corpora, all nine deterministic error injectors and compound-error composition code, rubric-annotation and training scripts, JSR/FM/-PA/DA evaluation code, the benchmark-specific execution harnesses and validation tests, evaluation sets, human-agreement annotations, the judge LoRA adapter, and all six planner checkpoints un-

der the following repository: <https://github.com/HadiAskari/toolhop-planning-emnlp>.