EVOPLANNER: DECOUPLING EVIDENCE OBLIGATIONS FROM EVOLVING TASK GRAPHS FOR RELIABLE TOOL USE

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable tool use depends on maintaining a coherent account of what the original request still requires as an agent revises its plan. Structured planners organize dependent actions, and verification-guided methods assess intermediate outputs, yet task-level status alone cannot express which parts of a request remain supported after a task is split or replaced. An incomplete task may already have obtained useful evidence; retaining its output does not by itself preserve an explicit link to the requirements that evidence satisfies. We present EVOPLANNER, which separates fixed queries-derived evidence obligations from an evolving Task DAG. Accepted observations are linked to these obligations through persistent Witnesses, and unmet obligations guide bounded revisions to unresolved tasks. This design preserves the recorded support across task replacement while allowing the acquisition strategy to change. In StableToolBench, EVOPLANNER achieves macro-average SoPR scores of six-subsets of 50.4% and 62.2% with Qwen-Long and DeepSeek V4 Flash, exceeding our corresponding VeriMap reproductions by 10.8 and 17.0 percentage points. Together with the component ablations, these results support integrating evidence verification with dynamic plan revision, enabling agents to adapt their execution while retaining acquired support and improving task completion.

1 INTRODUCTION

Tool-using agents extend language generation with external computation and information access. Modular systems connect language models to specialized components, while Toolformer learns when and how to invoke APIs (Karpas et al., 2022; Schick et al., 2023). For requests requiring several calls, the challenge shifts to coordinating intermediate results: ReAct interleaves reasoning and action, HuggingGPT plans and executes subtasks, and dependency-aware or recursive planners organize work as observations become available (Yao et al., 2023; Shen et al., 2023; Kim et al., 2024; Prasad et al., 2024).

Verification helps determine whether this work meets its objectives. VeriMap generates subtask-specific checks to guide refinement and replanning (Xu et al., 2026); VMAO assesses sub-question completeness and preserves earlier results across retries (Zhang et al., 2026). Yet task completion and query fulfillment need not coincide. For example, a task assigned to retrieve prices from two stores may obtain one price before the other lookup fails. Replacing that task should leave the accepted price linked to its original requirement, while the missing price remains a target for further execution. Task status alone does not express this distinction. Retaining raw outputs keeps the information accessible, but leaves its relationship to the original request to be recovered. The broader distinction between intermediate progress and final success also motivates analytical agent evaluation (Ma et al., 2024).

We study how to maintain requirement–observation support links across changes in task identity. Our insight is to define requirements directly from the query and retain accepted support against them, independently of the tasks that produced it. EVOPLANNER implements this separation through fixed *Evidence Obligations*, an evolving *Task DAG*, and a persistent *Witness* ledger. *Semantic Commit* assesses whether tool observations support the obligations; unmet obligations then guide

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.

bounded revisions. The execution route can change without erasing recorded support or redefining the remaining requirements (Figure 1).

Our contributions are threefold:

1. **Problem and insight.** We distinguish task completion from requirement-level support and formulate how accepted observations can remain linked to the original query when execution tasks are replaced.
2. **An evidence-guided planning framework.** We propose EVOPLANNER, which maintains fixed query-derived obligations and persistent Witness links independently of task identities. These links supply reusable evidence inputs and identify unmet requirements for bounded graph revision; Semantic Commit determines which observations enter the ledger.
3. **Evaluation on StableToolBench.** Across six subsets, EVOPLANNER reaches macro-average SoPR scores of 50.4 with Qwen-Long and 62.2 with DeepSeek V4 Flash, exceeding the corresponding local VeriMap adapters by 10.8 and 17.0 percentage points.

2 RELATED WORK

Tool Use and Structured Planning. Toolformer learns API invocation, whereas ToolLLM combines instruction tuning with search over API-call paths (Schick et al., 2023; Qin et al., 2023). Planning introduces structure beyond individual calls: Plan-and-Solve separates decomposition from solving in reasoning tasks, and HuggingGPT coordinates specialized models through planned subtasks (Wang et al., 2023; Shen et al., 2023). ReWOO separates planning from external observations to reduce repeated prompting (Xu et al., 2023). In interactive execution, ToolPlanner uses path planning and task-level feedback, ADaPT decomposes tasks as needed, and LLMCompiler schedules dependent calls (Wu et al., 2024; Prasad et al., 2024; Kim et al., 2024). GPTSwarm and DynTaskMAS further expose workflow structure as graphs (Zhuge et al., 2024; Yu et al., 2025). Our focus is how acquired support remains associated with query requirements when the execution structure changes.

Verification and Adaptive Replanning. Iterative refinement differs in both the source of feedback and the object being revised. Self-Refine uses the same model to generate, critique, and refine an output (Madaan et al., 2023); Reflexion retains verbal feedback across trials (Shinn et al., 2023). CRITIC grounds critique in interactions with external tools (Gou et al., 2024). The source of feedback matters: in the reasoning settings studied by Huang et al. (2024), intrinsic self-correction often failed to improve answers without external feedback. Verification-aware planners further connect feedback to execution structure. VeriMap generates subtask-specific criteria and verification functions to guide refinement and replanning (Xu et al., 2026). VMAO checks sub-question completeness, triggering retries or additional queries while preserving and merging earlier results (Zhang et al., 2026). These methods motivate examining both what a verification judgment establishes and how its supporting observations remain usable after a plan changes.

Evidence Attribution and Progress. Data provenance describes where a value originated and how it was derived (Buneman et al., 2001). In language generation, ALCE evaluates answer correctness separately from citation quality, including whether cited passages support generated statements (Gao et al., 2023b). RARR retrieves evidence and revises unsupported content while preserving the original output where possible (Gao et al., 2023a). These studies motivate explicit links between content and supporting sources. For tool-use planning, such links can also inform execution: an accepted observation may satisfy an input dependency or identify a requirement that no longer needs further acquisition. This calls for relating evidence to query requirements throughout the run, in addition to attributing statements in the final answer.

3 METHOD

EVOPLANNER organizes execution around three operations: defining query requirements independently of the task plan, recording the observations that support them, and using unmet requirements to revise the remaining work. This section describes how each operation contributes to that process. Runtime checks and auxiliary model guidance are detailed in Appendix A.

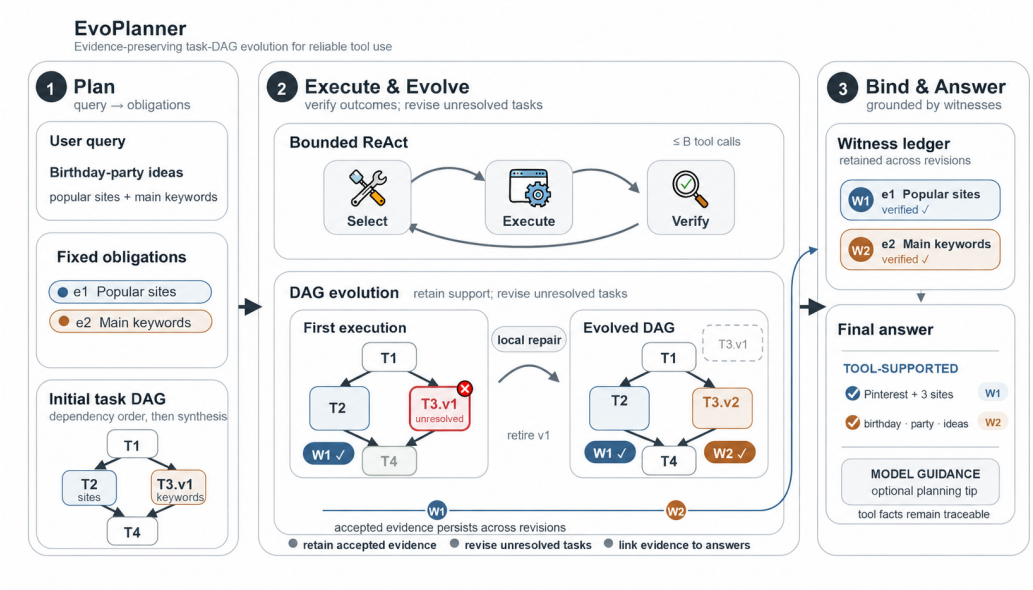

Figure 1: EVOPLANNER overview. Fixed query obligations remain unchanged as the task DAG evolves. In this illustrative example, W_1 is retained when $T3.v1$ is replaced, and W_2 is subsequently acquired for the remaining requirement. Tasks execute sequentially; arrows denote logical dependencies. Model guidance is separate from tool-supported Witnesses.

3.1 PROBLEM FORMULATION

Given a query q and available tools $\mathcal{C}$, an agent must acquire the requested information or outcomes within a tool-call budget B. Its execution plan is a versioned Task DAG, $G_T^{(v)} = (T^{(v)}, D_c^{(v)})$, whose nodes specify tasks and whose edges specify control dependencies. Tasks can also require evidence acquired earlier in the run. The plan may undergo at most R accepted revisions.

An independently generated obligation set, $\mathcal{E} = \{e_1, \ldots, e_m\}$, describes what the query requires. Each *Evidence Obligation* specifies information, an artifact, or an action outcome to be supported by tool observations and retains its corresponding query span. The definitions remain fixed during execution. We denote the required obligations by $\mathcal{E}_{\text{req}}$.

Classical planning distinguishes a goal condition from the operator sequence chosen to achieve it (Fikes & Nilsson, 1971). Here, obligations provide the reference for assessing fulfillment, while tasks specify a revisable acquisition strategy. Unlike symbolic goal predicates, these obligations are extracted from natural language and evaluated through model-mediated judgments; the separation does not confer the guarantees of a symbolic planner.

At step k, a Witness ledger $\mathcal{W}_k$ records accepted values, their supporting tool observations, and support judgments. Let S_k contain obligations whose state is `SUPPORTED` and that have a matching `SUPPORTED` Witness. Partial Witnesses are retained but do not establish strict support. The unmet requirements are

$$U_k = \mathcal{E}_{\text{req}} \setminus S_k. \tag{1}$$

The agent seeks to reduce U_k within the available budgets while retaining acquired support. This objective is defined relative to the extracted obligations and remains sensitive to errors in requirement extraction and semantic judgment.

3.2 INDEPENDENT TASK AND EVIDENCE DECOMPOSITION

The Task DAG describes how to address the query, and the obligation set specifies what the answer must cover. A task planner proposes subproblems and dependencies, leaving tool selection to execution. A separate evidence generator reads the original query and extracts the requested information

and outcomes, preserving their scope and conditions. Generating the obligations directly from the query gives later revisions a reference independent of the initial plan. Before execution, a critic reviews these obligations for omissions or changes in meaning, after which the system freezes their structurally valid definitions.

An aligner associates tasks with the required obligations they are intended to address:

$$A^{(v)} \subseteq T^{(v)} \times \mathcal{E}_{\text{req}}. \tag{2}$$

This many-to-many relation allows one task to pursue several requirements and several tasks to pursue the same requirement. Every task must have a valid target, while obligations without an initial task remain open. The alignment expresses intended coverage, which execution must subsequently establish. During replanning, tasks and their alignments may change while $\mathcal{E}$ remains fixed, so abandoning an acquisition route does not remove the requirement it was meant to satisfy.

Intermediate execution data. Obligations describe requested answer content, rather than every intermediate tool argument. For example, obtaining a product identifier before retrieving its price is handled by successive calls within the same coarse task. The identifier remains available in the observation history without being promoted to a fulfilled price requirement. Cross-task evidence dependencies are available only when the needed input corresponds to a supported obligation; otherwise, a whole-task control dependency remains. The current representation does not provide a separate graph of auxiliary variables, so chains that exceed a task's local budget can still fail. Appendix A details this boundary.

3.3 Semantic Commit and Persistent Witnesses

Execution can produce useful observations before a task meets all its objectives. EvoPlanner therefore evaluates support at the level of evidence obligations and stores it independently of task status. A task becomes eligible when it is pending, has unmet targets, its control predecessors have succeeded, and its required evidence inputs are strictly supported. Eligible tasks execute one at a time through a bounded ReAct loop (Yao et al., 2023), using the task instruction, target obligations, available tools, and previously supported inputs.

To propose support, the executor identifies an obligation and references the relevant locations in recorded tool results. The runtime resolves these references and extracts the value directly from the observations. Semantic Commit then assesses whether the value addresses the requested subject, relation, scope, and conditions. Provenance identifies the observation from which a value was extracted (Buneman et al., 2001); semantic assessment asks whether that value supports the obligation. This distinction is consistent with attribution research, where a cited passage must support the associated statement (Gao et al., 2023b). A resolvable observation reference alone therefore does not establish support. Accepted values are stored as Witnesses with their observation references and a `PARTIAL` or `SUPPORTED` judgment.

Task status and Witness status refer to different objects: a `PARTIAL` task has made progress but still has unmet targets, whereas a `PARTIAL` Witness does not fully support its associated obligation. A partial task can therefore produce a `SUPPORTED` Witness. Only a `SUPPORTED` Witness establishes strict support, removes an obligation from U_k, and satisfies an evidence dependency. A partial Witness retains incomplete information while leaving the obligation open. A task succeeds once all its targets are strictly supported, but failing to complete the task does not discard the observations already accepted. An observation may also support an open obligation outside the task's primary targets. Where a dependency is expressed through an unambiguous supported evidence input, a downstream task can reuse that information even if its producer has other unresolved targets. Witnesses thus retain both the value and its relation to the query when the producing task is later replaced.

Illustrative partial-success case. Figure 1 shows reuse from a successful branch; the same representation also retains evidence from an incomplete task. Suppose T_1 targets the prices of product p at stores A and B, represented by e_1 and e_2. A result from A establishes $W_1 \to e_1$, but B remains unresolved, so T_1 is partial. Revision can replace T_1 with T_2 targeting only e_2, while retaining W_1 for synthesis or as a declared evidence input. This constructed example illustrates the state transition, not an observed benchmark trajectory.

3.4 EVIDENCE-GUIDED RESIDUAL GRAPH REVISION

Replanning begins when requirements remain unmet, no task is eligible to execute, and both tool-call and revision budgets remain. The repair planner receives U_k, task states, acquired evidence, unsuccessful attempts, and available tool capabilities. These inputs identify what is still needed and which acquisition routes have already been attempted.

The planner may replace or split unresolved tasks, or introduce tasks needed to obtain missing evidence. Successful and unaffected tasks are retained. New tasks can use already supported obligations as evidence inputs, while control edges specify dependencies on work still to be executed. The runtime validates the complete candidate graph before applying it, checking targets, dependencies, capabilities, acyclicity, and repeated acquisition routes. A valid proposal is installed atomically; an invalid proposal leaves the current plan unchanged and can be corrected within a bounded retry limit.

Each accepted revision preserves the obligation definitions and the current Witness ledger:

$$\mathcal{E}_{\text{after}} = \mathcal{E}_{\text{before}}, \qquad \mathcal{W}_{\text{after}} = \mathcal{W}_{\text{before}}. \tag{3}$$

Because revision also preserves obligation support states, Equation 3 leaves S_k, and therefore U_k, unchanged at the moment of the update. Splitting or replacing a task cannot by itself satisfy a requirement; further progress must come from accepted observations. This property concerns the consistency of recorded support across graph versions, not the correctness or continuing validity of an external observation. The budget R limits accepted graph updates, and the planner can stop when it finds no useful alternative. Persistence here means retaining an accepted record within a run; the current implementation does not automatically revoke support after an external change or a conflicting later observation.

Effect of revision on executability. A graph can be stalled even when part of the query is already supported: the remaining tasks may have exhausted their local attempts or depend on predecessors that did not succeed. Repair changes this execution structure while retaining the current support state. Newly introduced tasks receive new identities and are evaluated by the same eligibility rules. A supported obligation can provide a declared evidence input even if its producing task was only partially successful; a control dependency still requires its predecessor to have succeeded. Consequently, a revision can enable another acquisition route without immediately reducing U_k. Reducing U_k still requires an accepted observation that strictly supports an unmet obligation.

Budget accounting. Execution and revision consume different budgets. The tool-call budget B counts external interactions, whereas R counts accepted graph updates. Rejected repair proposals leave the graph unchanged and do not consume R, but they still incur model calls and are bounded by a separate retry limit. Thus, preserving evidence does not make repair free, and an unchanged U_k after revision is compatible with both a changed execution route and additional inference cost.

Answer synthesis. Execution ends when all required obligations are strictly supported, the tool budget is exhausted, or no further execution or accepted revision is available. The system synthesizes an answer from supported Witness values and reflects unresolved requirements in the response. When enabled, model-origin guidance is identified separately and cannot establish tool-grounded support. A final check assesses coverage of the extracted deliverables and faithfulness to the synthesis inputs, with a deterministic rendering fallback if needed.

Overall procedure. Algorithm 1 uses G for the Task DAG, X for task and obligation states, and H for observation history. EXECUTECOMMIT returns accepted support and ends the current task attempt, including attempts that consume no tool call. Only pending tasks can execute; the same task identity cannot be selected repeatedly. TRYREPAIR returns a validated update or $\perp$ after bounded proposal attempts. The tool budget B and accepted-revision budget R bound external interactions and graph updates; Appendix A specifies the separate model-call limits.

Algorithm 1 EVOPLANNER with persistent evidence and bounded graph revision

Require: Query q, tools $\mathcal{C}$, total tool-call budget B
Require: Per-task tool-call cap b, accepted-revision budget R
Ensure: Answer y, Witness ledger $\mathcal{W}$, unmet requirements U

$G \leftarrow \textsc{PlanTasks}(q)$
$\mathcal{E} \leftarrow \textsc{ExtractReviewFreeze}(q)$ ▷ Independent of G
$(G, A) \leftarrow \textsc{AlignAndCompile}(G, \mathcal{E}_{\text{req}})$
$X \leftarrow \textsc{InitializeStates}(G, \mathcal{E})$
$\mathcal{W} \leftarrow \varnothing$; $H \leftarrow \varnothing$; $U \leftarrow \mathcal{E}_{\text{req}}$; $c \leftarrow 0$; $r \leftarrow 0$
while $U \neq \varnothing$ **and** $c < B$ **do**
  $t \leftarrow \textsc{SelectEligible}(G, A, X, \mathcal{W})$
  if $t \neq \perp$ **then**
    $(\Delta\mathcal{W}, H, \delta) \leftarrow \textsc{ExecuteCommit}(t, q, \mathcal{E}, \mathcal{W}, \mathcal{C}, H, \min(b, B - c))$
    $\mathcal{W} \leftarrow \mathcal{W} \cup \Delta\mathcal{W}$; $c \leftarrow c + \delta$
    $X \leftarrow \textsc{UpdateStates}(t, G, A, X, \mathcal{W})$ ▷ Terminalize this attempt
  else
    if $r = R$ **then**
      break
    end if
    $(P, H) \leftarrow \textsc{TryRepair}(q, G, A, X, \mathcal{E}, U, \mathcal{W}, \mathcal{C}, H)$
    if $P = \perp$ **then**
      break
    end if
    $(G, A, X) \leftarrow \textsc{ApplyAtomically}(P, G, A, X)$
    $r \leftarrow r + 1$ ▷ $\mathcal{E}$ and $\mathcal{W}$ remain unchanged
  end if
  $U \leftarrow \mathcal{E}_{\text{req}} \setminus \textsc{StrictSupport}(X, \mathcal{W})$
end while
$y \leftarrow \textsc{SynthesizeAndCheck}(q, \mathcal{W}, U)$
return $(y, \mathcal{W}, U)$

4 EXPERIMENTS

We evaluate EVOPLANNER on StableToolBench across six subsets, with component-removal and permitted repair-budget studies on I3 Inst.

4.1 EXPERIMENTAL SETUP

Benchmark and tool environment. StableToolBench is a tool-use benchmark built on ToolBench that reduces variation in tool availability through cached responses and API simulation (Guo et al., 2024). All locally evaluated methods use StableToolBench's MirrorAPI-Cache tool-response environment. We use all 765 queries in its official solvable test set: 163, 153, 158, 124, 106, and 61 queries in I1 Inst, I1 Cat, I1 Tool, I2 Cat, I2 Inst, and I3 Inst, respectively. These subsets cover single- and multi-tool tasks and assess generalization across instructions, tools, and categories. The main comparison covers all six subsets; component and repair-budget studies use all 61 I3 Inst queries with DeepSeek V4 Flash.

Agent evaluation also covers interactive decision-making, intermediate progress, stateful milestones, and repeated-trial reliability (Liu et al., 2023; Ma et al., 2024; Lu et al., 2024; Yao et al., 2024). Our evaluation addresses final-answer performance on StableToolBench; it does not directly measure support accuracy or run-to-run reliability.

Comparison methods. The baselines cover reasoning trajectories, search, and verification-aware planning. Chain-of-thought prompting makes intermediate reasoning explicit (Wei et al., 2022); ToolLLM's DFS explores alternative API-call paths (Qin et al., 2023); VeriMap uses subtask-specific verification to guide retries and replanning (Xu et al., 2026). We import the ToolLLaMA v2, GPT-4o mini, and GPT-4o CoT/DFS scores from StableToolBench's official MirrorAPI-Cache results (Guo et al., 2024). With Qwen-Long and DeepSeek V4 Flash, we compare local CoT, VeriMap, and EVOPLANNER configurations. All local methods have an eight-tool-call ceiling per query. This controls external interaction opportunities, but does not equalize model calls, tokens, or latency.

Our VeriMap adapter uses LLM-based checks in place of generated Python verification; its results describe this reproduction, not the original implementation.

Component and budget studies. We examine three removals from the full framework. *w/o Dynamic DAG* disables task-graph revision. *w/o Evidence Layer & Dynamic Repair* removes the independent evidence representation and dynamic repair together. *w/o Intra-Task Verification Feedback* removes feedback within task execution. The budget study varies the maximum number of accepted graph revisions, R, over $\{0, 1, 2, 3\}$. This is a permitted budget rather than the number of revisions necessarily used by a query.

Reporting. For each locally evaluated configuration, we generate one answer set and evaluate it three times with GPT-4o, reporting mean and standard deviation in SoPR (%). The evaluator averages query scores of 1 (Solved), 0.5 (Unsure), and 0 (Unsolved). The reported variation reflects repeated judging of fixed answers, not independent agent runs or run-to-run stability. Macro-average is the unweighted mean of the six displayed subset scores, including imported baselines, rather than the query-weighted mean over 765 queries. Macro-average and component/budget results are point estimates. Higher SoPR is better; differences are in percentage points.

4.2 Main Results

Table 1: StableToolBench SoPR (%, mean ± standard deviation). The upper block reproduces official MirrorAPI-Cache subset results. Macro-average is recomputed as the arithmetic mean of the six subset scores, without an uncertainty estimate. Our runs are grouped by backbone. Bold marks the best local score within each backbone group. VeriMap rows use our LLM-check adapter.

Method	I1 Inst	I1 Cat	I1 Tool	I2 Cat	I2 Inst	I3 Inst	Macro-avg.
ToolLLaMA v2 CoT	28.0±1.9	30.5±0.8	21.5±0.9	19.9±1.0	22.3±0.4	19.1±0.8	23.6
ToolLLaMA v2 DFS	28.4±0.9	32.5±0.8	22.2±1.0	22.8±1.5	19.2±1.6	18.6±1.5	24.0
GPT-4o mini CoT	27.8±1.4	34.9±0.3	34.2±0.5	24.5±1.0	22.3±2.7	20.8±1.5	27.4
GPT-4o mini DFS	26.8±1.4	36.4±1.6	33.1±1.1	25.8±1.7	25.8±2.7	20.2±0.8	28.0
GPT-4o CoT	33.3±2.0	35.1±0.6	33.6±0.8	32.5±1.7	29.6±1.6	27.9±3.5	32.0
GPT-4o DFS	32.7±1.9	42.3±1.3	34.6±1.3	32.8±1.5	28.3±1.3	23.0±1.3	32.3
Qwen-Long CoT	39.5±1.4	36.5±1.2	37.8±0.9	35.4±2.1	36.5±1.3	44.2±1.1	38.3
Qwen-Long (VeriMap adapter)	38.4±1.6	37.3±1.1	40.2±0.7	38.7±1.3	37.3±2.1	45.9±1.7	39.6
Qwen-Long EvoPlanner (ours)	**49.7±0.8**	**47.8±0.9**	**41.7±1.1**	**48.4±1.8**	**49.3±1.3**	**65.6±1.7**	**50.4**
DeepSeek V4 Flash CoT	39.7±1.4	44.3±1.7	37.6±2.3	38.9±1.3	44.2±1.5	53.3±0.0	43.0
DeepSeek V4 Flash (VeriMap adapter)	40.1±0.9	37.9±1.7	42.2±2.1	43.1±1.2	45.5±0.6	62.3±1.7	45.2
DeepSeek V4 Flash EvoPlanner (ours)	**62.3±1.7**	**59.6±0.8**	**55.4±0.9**	**60.4±1.2**	**56.9±1.6**	**78.7±1.6**	**62.2**

Where the reported differences occur. Table 1 reports higher scores for EvoPlanner than the corresponding CoT and VeriMap configurations on all six subsets. Relative to VeriMap, the macro-average differences are 10.8 points for Qwen-Long and 17.0 for DeepSeek. The subset pattern is more informative than the overall ranking: Qwen-Long's difference is 19.7 points on I3 Inst but only 1.5 on I1 Tool. With DeepSeek, the largest difference is instead 22.2 points on I1 Inst, while I3 Inst improves by 16.4. Thus, the reported advantage is not confined to multi-tool queries, and its distribution depends on the backbone. The small Qwen-Long difference on I1 Tool also cautions against treating all generalization settings as equally improved.

What could account for these differences. Compared with reasoning or search over execution paths, EvoPlanner maintains an explicit account of which query requirements still lack support. Compared with subtask-specific verification in VeriMap (Xu et al., 2026), its proposed distinction is the persistence of those support links when task identities change. This could avoid reacquiring information already obtained by a partly completed task. It is a mechanism hypothesis, however: the aggregate scores do not measure reuse or establish that reuse caused the differences. Additional verification, candidate generation, and final-answer checks also consume inference effort. These comparisons do not isolate persistent support links from additional verification and inference effort.

4.3 Component Ablations

Table 2: Component ablations on I3 Inst with DeepSeek V4 Flash. Δ is the score difference from the full framework.

Variant	I3 Inst	Δ
EvoPlanner (full)	**78.7**	—
w/o Dynamic DAG	68.9	−9.8
w/o Evidence Layer & Dynamic Repair	54.1	−24.6
w/o Intra-Task Verification Feedback	73.8	−4.9

Table 2 shows score reductions of 9.8, 24.6, and 4.9 percentage points after removing Dynamic DAG, the evidence layer with dynamic repair, and intra-task verification feedback, respectively. These comparisons concern plan revision, a joint representation-and-repair removal, and within-task feedback. The joint removal does not isolate the contribution of either component.

4.4 Effect of the Repair Budget

Figure 2 compares SoPR and token consumption across four repair budgets on the 61 I3 Inst queries with DeepSeek V4 Flash.

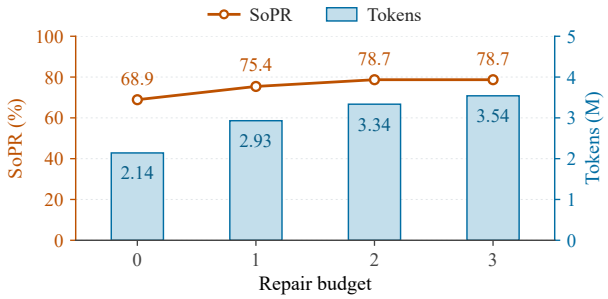

Figure 2: Repair-budget study on I3 Inst. Orange: SoPR (left); blue: input/output tokens, including planning, verification, and repair (right). Axes use independent scales. Labels are rounded; uncertainty estimates are unavailable.

Allowing one revision raises SoPR from 68.9% to 75.4%, and allowing a second raises it to 78.7%. The corresponding token totals increase from 2,140,633 to 2,930,251 and 3,335,204. This sweep therefore shows additional token consumption alongside the higher reported scores.

From zero to two permitted revisions, the score increases by 9.8 percentage points while total tokens increase by approximately 55.8%. At two revisions, the reported total corresponds to about 54,675 tokens per query. This is an accuracy–cost trade-off within the tested configuration; it does not demonstrate token savings relative to VeriMap or a computation-matched retry baseline.

Increasing the budget from two to three leaves SoPR at 78.7% at the reported precision, while the token total rises to 3,540,573, an additional 205,369 tokens (6.2%). A budget of two thus matches the highest reported SoPR in this sweep with fewer tokens than a budget of three. The zero-revision score also matches the Dynamic DAG ablation in Table 2. These aggregate observations describe the tested budgets; they do not identify a universally optimal repair limit.

5 Discussion and Limitations

Scope of support. The ledger preserves an accepted requirement–observation link across task replacement; it does not certify the underlying observation. Provenance and semantic support remain distinct (Buneman et al., 2001; Gao et al., 2023b). Requirement extraction can omit content, and the

verifier can accept irrelevant evidence or reject useful observations. External feedback does not make model-mediated correction infallible (Gou et al., 2024; Huang et al., 2024).

Method boundaries. The implementation has no automatic expiry or conflict-driven support revocation. Separate partial Witnesses do not automatically establish strict support, and intermediate identifiers remain in observation history rather than a separate auxiliary-variable graph. Auxiliary candidates and final-answer checks can affect the answer without creating tool-grounded Witnesses. The current results therefore do not establish reliability under changing external state or correctness of derived claims.

Empirical scope. The controlled study covers 61 development-exposed queries with a different judge. It has one generated answer per condition and query; three judgments do not replace independent generation replications. Equal tool ceilings also leave model effort unequal. Further evidence is needed to establish support accuracy, generalization, and a benefit at matched inference cost.

6 Conclusion

EvoPlanner separates fixed query-derived evidence obligations from an evolving Task DAG. Semantic Commit links accepted tool observations to persistent Witnesses, preserving recorded support when partially completed tasks are replaced, while unmet obligations guide bounded graph revision. Across six StableToolBench subsets, the reported full configurations exceed the corresponding local VeriMap adapters by 10.8 and 17.0 percentage points in macro-average SoPR with Qwen-Long and DeepSeek V4 Flash, respectively. On I3 Inst, component removals lower reported scores, while the repair-budget sweep shows performance gains accompanied by increased token consumption. These comparisons characterize the integrated system but do not isolate the contribution of persistent support links. Further evaluation should directly measure support accuracy and evidence reuse, and assess mechanism-level benefits under matched inference budgets.

AI use statement

Generative AI tools assisted with literature searches, LaTeX formatting, and manuscript polishing. The authors bear full responsibility for all arguments and the final output.

References

Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and where: A characterization of data provenance. In *Database Theory—ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pp. 316–330. Springer, 2001. doi: 10.1007/3-540-44503-X_20. URL https://link.springer.com/chapter/10.1007/3-540-44503-X_20.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971. doi: 10.1016/0004-3702(71)90010-5. URL https://doi.org/10.1016/0004-3702(71)90010-5.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. RARR: Researching and revising what language models say, using language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16477–16508, Toronto, Canada, 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.910. URL https://aclanthology.org/2023.acl-long.910/.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 6465–6488, Singapore, 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.398. URL https://aclanthology.org/2023.emnlp-main.398/.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International*

Conference on Learning Representations, 2024. URL https://arxiv.org/abs/2305.11738.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.664. URL https://aclanthology.org/2024.findings-acl.664/.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2310.01798. URL https://arxiv.org/abs/2310.01798.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholtz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022. URL https://arxiv.org/abs/2205.00445.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2312.04511.

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. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024. URL https://arxiv.org/abs/2408.04682.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. *arXiv preprint arXiv:2401.13178*, 2024. URL https://arxiv.org/abs/2401.13178.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://doi.org/10.52202/075280-2019.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.264. URL https://aclanthology.org/2024.findings-naacl.264/.

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. doi: 10.48550/arXiv.2307.16789. URL https://arxiv.org/abs/2307.16789.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving AI tasks with ChatGPT and its friends in hugging face. *arXiv preprint arXiv:2303.17580*, 2023. URL https://arxiv.org/abs/2303.17580.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://doi.org/10.52202/075280-0377.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023. URL https://arxiv.org/abs/2305.04091.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. URL https://arxiv.org/abs/2201.11903.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 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*, pp. 18315–18339. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.1018. URL https://aclanthology.org/2024.emnlp-main.1018/.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023. URL https://arxiv.org/abs/2305.18323.

Tianyang Xu, Dan Zhang, Kushan Mitra, and Estevam Hruschka. Verification-aware planning for multi-agent systems. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7528–7546, Rabat, Morocco, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-long.353. URL https://aclanthology.org/2026.eacl-long.353/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Junwei Yu, Yepeng Ding, and Hiroyuki Sato. DynTaskMAS: A dynamic task graph-driven framework for asynchronous and parallel LLM-based multi-agent systems. *Proceedings of the International Conference on Automated Planning and Scheduling*, 35(1):288–296, 2025. doi: 10.1609/icaps.v35i1.36130. URL https://doi.org/10.1609/icaps.v35i1.36130.

Xing Zhang, Yanwei CUI, Guanghui Wang, Qucy Wei Qiu, Ziyuan Li, Fangwei Han, Yajing Huang, Hengzhi Qiu, Bing Zhu, and Peiyang He. Verified multi-agent orchestration: A plan-execute-verify-replan framework for complex query resolution. In *Workshop on Multi-Agent Learning and Its Opportunities in the Era of Generative AI*, 2026. URL https://openreview.net/forum?id=WUmz4LUbvU.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Juergen Schmidhuber. Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.16823.

A Runtime Details

The details below describe the local paper-core implementation inspected for this revision.

A.1 Representations and Support Decisions

Stored records. A Task stores its identifier, instruction, control predecessors, required evidence inputs, version, and superseded task identifiers. An Evidence Obligation stores its identifier, natural-language requirement, required flag, original-query span, outcome kind, and associated answer deliverables. A Witness records its producing task, obligation-level support states, extracted values, and supporting observation references. The alignment maps each task to its intended target obligations. These records give task identity and requirement identity separate lifecycles.

Requirement review. The evidence generator is instructed to extract minimal answer-bearing slots directly from the query, preserve scope and conditions, and copy an authorizing query span for each slot. The critic checks omissions, meaning changes, and redundant decomposition. It may request one regeneration; after structural validation, remaining critic disagreement is recorded as a warning. This is a review of the generated requirements, not an independent guarantee of query coverage.

Observation references and semantic verification. A submitted fill identifies an obligation and one or more references of the form `call_id + json_pointer`. The JSON Pointer resolves against the recorded result of that call. Unknown or failed calls, missing paths, empty values, and recognized error results are rejected. Successful observations from earlier tasks in the same run can be cited again; raw observation history and committed Witness support are distinct sources of state. The runtime extracts the selected value rather than accepting newly written claim text. Several references yield a list of extracted values that must be assessed as the candidate answer.

The semantic prompt asks whether the extracted answer supports the frozen requirement with respect to seven aspects: referent, relation, scope, modality, polarity, conditions, and answer shape. Every aspect receives a supported, partial, or unsupported judgment. The runtime aggregates them: any unsupported aspect rejects strict support; otherwise any partial aspect yields partial support, and all-supported yields strict support. Malformed or incomplete verdicts are rejected. The implementation also rejects a scalar string that merely repeats text in the requirement or its query anchor. Relation matching is semantic; the current runtime does not require lexical overlap. These checks can still make mistakes and need an external support audit.

Partial evidence and derived answers. A partial Witness leaves its obligation unmet and cannot release a strict evidence dependency. Separate partial Witnesses are not automatically combined into a supported obligation. A later proposal may cite recorded observations together, but the resulting value must pass a fresh support decision for the complete requirement. Extraction returns observed values or a list of them; it does not establish an arbitrary computed relation. For example, citing two prices can provide inputs for a final comparison without turning the model's computed difference into a tool-observed Witness. Tool-observed derived results can be submitted through the same reference contract.

A.2 Scheduling, Revision, and Termination

Two dependency meanings. A control dependency requires its predecessor task to be succeeded. A declared evidence dependency requires a supported obligation and its committed value, independently of whether the producer has other unresolved targets. Alignment converts a producer edge into an evidence input only when that binding is explicit and unambiguous; other control edges remain. Intermediate identifiers and cursors that are not requested answer content stay in observation history. They may ground later arguments without satisfying a final-answer obligation. The method has no separate auxiliary-variable graph and does not weaken a requested result into an intermediate lookup merely to release a task.

Task and obligation states. Pending tasks with unmet targets and satisfied dependencies can execute. On return, a task is succeeded if all its targets are supported, partial if it committed some

values but still has unmet targets, and failed if it committed none. An executed task is not automatically returned to pending. Replacement retires unresolved tasks and introduces new identities. Obligations begin open; accepted partial values yield partial status, while accepted complete support yields supported status. Although the type definitions include a conflicted state, the inspected commit path does not implement automatic conflict detection, expiry, or support revocation. Persistent records should therefore not be interpreted as guaranteed current truth.

Bounded execution. With an effective local tool cap $b' = \min(b, B - c)$, the inspected executor bounds its action loop by $\max(4, 3b' + 2)$ turns. A finish action can commit supported or partial values and explicitly declare other targets unresolved. Invalid references and some rejected answer projections are returned as feedback within that loop. Repeated invalid transactions and the turn cap terminate unresolved targets while retaining accepted sibling evidence. This mechanism bounds attempts even when no tool call is consumed.

Candidate graph checks. The repair planner diagnoses the remaining evidence gap and proposes a preserve/retire/add/rewire update. The runtime checks valid targets, declared tool capabilities, available dependencies, and acyclicity, and rejects repeated acquisition routes. New committed-evidence inputs must already be supported. The whole candidate is checked before installation; a rejection leaves the current graph unchanged. For the same recorded gap and progress state, the inspected engine allows two rejected proposals before closing that repair attempt. A no-repair response also stops the attempt. Only accepted updates consume R; rejected proposals still incur model cost. These limits and terminal task states prevent an unchanged, nonproductive execution cycle.

A.3 Auxiliary Guidance and Final Synthesis

When enabled, the model-candidate channel supplies separately identified answer candidates and estimates the value of further tool use. It can influence eligible-task ranking and which unmet requirements are selected for repair. It cannot create a Witness, satisfy a strict evidence dependency, or prove completion of an external action. Thus, this channel can affect measured answer quality even though it does not increase tool-grounded support.

The local paper entry point enables candidate generation by default and wires a separate final-answer check. The latter evaluates coverage and faithfulness to the synthesis inputs, with a deterministic rendering fallback. Each phase is passed through the supplied model callable; this wiring does not identify the exact model used in a historical run. Per-configuration enablement, model routing, check and fallback frequencies, and guidance-on/off results are not available in the current aggregate tables. The main results therefore evaluate full configurations rather than isolating either auxiliary component.