JOURNAL OF GLOBAL ENGINEERING REVIEW ISSN: 3071-4656

Reflexion-Guided Self-Correcting Tool Agents under API Selection and Execution Failures

David Murphy[1], Jing Yang[2], Kimberly Cox[3], Fan Song[4]
[1]*computer science, vtech, va, usa*
[2]*computer science, upenn, pa, usa*
[3]*computer science, yale, ct, usa*
[4]*computer science, jhu, md, usa*
dmurphy.cs@gmail.com

Abstract
Tool-using language agents fail through coupled errors: an incorrect API choice changes the available argument schema, while correct tool selection still fails when dialogue state, identifiers, temporal values, or required fields are bound incorrectly. This study evaluates a Reflexion-guided correction pipeline on a four-file API-Bank evaluation package containing 1,012 records, including 534 API-call targets from 261 dialogues. The first-pass agent ranks the provided API schemas with BM25 over the latest four dialogue events and binds arguments with deterministic local extraction rules. A benchmark feedback gateway rejects calls that are not execution-equivalent to the target and records a structured diagnosis. The repair agent then reconstructs the active trajectory segment, reselects the API with a state machine and global library ranking, and rebinds arguments using fold-local episodic profiles, value lexicons, and defaults. All memory objects were fitted in five GroupKFold splits grouped by dialogue, preventing any dialogue from contributing to its own repair memory. First-pass tool accuracy was 61.80%, strict call accuracy was 29.03%, and execution-normalized success was 36.52%. The gated correction pipeline reached 93.82% tool accuracy, 58.61% strict call accuracy, and 70.41% execution-normalized success. It repaired 181 of 339 initial failures and preserved all 195 initially successful calls, producing a 33.90-point paired gain (95% bootstrap confidence interval: 29.96–38.01; exact McNemar $p=6.53\times10^{-55}$). Level-3 trajectories gained 44.44 points, and controlled failure injection yielded repair success from 67.04% for wrong tools to 100% for observed stale-state errors. The results show that trajectory state and argument reflection provide complementary gains and that value grounding remains the dominant residual failure.

Keywords: API-Bank; tool-augmented language models; function calling; Reflexion; self-correction; API selection; argument binding; agent evaluation; episodic memory; execution failure

Introduction

Large language models become operational agents when they can retrieve external capabilities, choose an appropriate interface, construct a valid call, inspect the result, and continue a multi-turn task. ReAct established a general pattern for interleaving reasoning and action [1], while Toolformer showed that language models can learn to insert calls to calculators, search systems, translation services, and other tools [2]. Modular systems such as MRKL [3], browser-grounded agents such as WebGPT [4], and affordance-grounded robotic policies such as SayCan [5] further demonstrated that external actions can compensate for limitations of parametric knowledge. Retrieval-augmented generation supplies a related mechanism for grounding language generation in retrieved evidence [6]. This grounding pattern also appears in code intelligence, where retrieval is coupled with concise summaries [7], and in financial-disclosure question answering, where responses are tied to source evidence [8]. Together, these lines of work shifted evaluation from text quality alone toward the correctness of decisions that change an environment.

Tool use nevertheless introduces a failure chain that ordinary answer-level evaluation hides. The agent must first identify the intended operation, distinguish it from semantically adjacent APIs, respect the current stage of a trajectory, and bind every required argument. A wrong API can make every subsequent field invalid. A correct API can still fail because the agent copied an obsolete token, selected the first rather than the latest date, omitted a field requested by the schema, or produced a value that is textually different but operationally equivalent. Large API collections amplify the selection problem. ToolLLM targets more than 16,000 real-world APIs [9], and Gorilla studies retrieval-aware API calls [10]. A comparable routing problem arises in e-commerce recommendation pipelines, where a system must identify a relevant product class before ranking or personalization [11]. ToolAlpaca develops generalized tool learning [12], and RestGPT connects language models to RESTful interfaces [13]. HuggingGPT extends orchestration across many model services [14]. These systems establish broad capability, but their aggregate task scores do not isolate the exact contribution of a retry after an API-selection or argument-execution failure.

API-Bank was designed to expose planning, retrieval, and calling errors in tool-augmented language models [15]. Its evaluation dialogues contain single-operation cases, competing API choices, and trajectories in which a ToolSearcher action reveals a downstream interface. The benchmark therefore supports a controlled comparison between a first attempt and a state-aware repair attempt. The four-file package evaluated here preserves 534 API-call targets and 478 assistant-response targets. It also contains repeated domains—appointments, health, banking, smart-home control, reminders, search, translation, and utility operations—that stress both lexical selection and dialogue-state tracking. Related operational work treats diagnosis as an evidence-to-action process: AIOps systems combine heterogeneous signals to localize faults [16], while ambiguity-aware HDFS analysis couples anomaly detection with retrieved failure narratives and selective refusal when the evidence is insufficient [17].

Self-correction research provides a natural mechanism for addressing these failures. Reflexion converts environmental feedback into linguistic reflection stored in episodic memory, improving later decisions without weight updates [18]. Self-Refine uses iterative self-feedback to revise an output [19]. Operational monitoring systems make a related distinction between detecting a failure and selecting a safe retraining or rollback action [20]. CRITIC grounds self-correction in tool-interactive feedback [21]. Tree of Thoughts explores alternative reasoning branches [22], whereas chain-of-thought prompting [23], self-consistency [24], and program-aided language models [25] provide different forms of intermediate computation. The common principle is that a second decision becomes useful when it receives information unavailable to the first. For tool agents, that information includes the failure type, the active trajectory stage, previously returned identifiers, the complete API library, and empirical knowledge of which parameter profiles occur for each tool.

Across deployed predictive systems, aggregate accuracy alone is insufficient: credit-risk models support consequential decisions [26], human-uncertainty distillation separates confidence quality from top-1 accuracy [27], and financial-service analytics combine customer insight with risk analysis [28]. These examples reinforce the need to evaluate a downstream decision rather than only a component prediction. A reliable repair evaluation must preserve the coupling between selection and execution. Reporting tool accuracy alone credits a call whose fields are unusable, while reporting value accuracy with the gold tool supplied removes the routing problem. Final task accuracy alone also conceals whether the second pass repaired a failure or replaced an already correct call. The paired design used here records all three stages: first candidate, direct repair candidate, and gated final call. It also distinguishes strict serialization from execution-normalized equivalence. This distinction matters for free text, formulas, capitalization, and list-like fields, where two strings can encode the same operation. Conversely, a superficially similar value can be operationally wrong when it refers to an old appointment identifier or a completed turn. The evaluation therefore treats trajectory state as part of call semantics rather than as incidental dialogue history.

The central research question is whether a compact reflection loop can recover failures without retraining a language model. Four subsidiary questions structure the evaluation. First, how much tool-selection accuracy is gained by replacing local lexical ranking with trajectory-aware re-selection? Second, how much argument

accuracy is gained by expanding local context and adding fold-local episodic profiles, lexicons, and defaults? Third, do the two components combine into a statistically reliable improvement in complete calls? Fourth, which injected failure types remain repairable after corruption of an otherwise valid target call?

This study contributes a call-level correction analysis rather than a single aggregate benchmark score. It evaluates every API-call target in the package, records first-pass and final predictions, and reports tool, parameter-name, value, strict-call, and execution-normalized metrics. It separates direct repair from gated repair, uses dialogue-grouped cross-validation for every learned memory object, measures paired transitions and confidence intervals, and tests wrong-tool, missing-field, wrong-value, invalid-format, and stale-state failures. The analysis also reports calibration, context-length robustness, and local runtime. This design follows the broader view of augmented language models as systems whose reliability depends on retrieval, action, feedback, and memory rather than on generation alone [29]. The same demand for traceable evidence-to-action links appears in data-driven enterprise marketing decisions [30], where a prediction must still be connected to a defensible operational choice.

Method

A. Dataset and evaluation unit. The experiment used the complete contents of the four-file API-Bank evaluation package. Each text file contains a JSON array whose records include a dialogue filename, instruction, dialogue input, and expected output. Records whose expected output begins with “API-Request:” were parsed as call targets; the remaining records were retained as assistant-response targets so that corpus composition and dialogue chronology remained intact. The parser recovered the API name and keyword arguments from bracketed calls and canonicalized quotation, whitespace, booleans, and scalar values. API schemas were extracted directly from the instruction text. This procedure produced 534 call targets, 478 response targets, 261 dialogue files with at least one call, 50 distinct target APIs, and 47 distinct argument names. Table I reports the call distribution. Level 1 contains 174 calls, Level 2 contains 225, and Level 3 contains 135. The analysis treats an API call—not an entire dialogue—as the prediction unit because a single dialogue can contain several sequential decisions with different target tools.

TABLE I

API-BANK CALL-TARGET COMPOSITION

Level	Call targets	Dialogue files	Unique tools	Mean args	Mean candidates	Mean context words
1	174	127	49	2.35	1.45	65.79
2	225	85	27	2.21	2.76	107.63
3	135	49	30	1.98	1.00	107.96
All	534	261	50	2.20	1.89	94.08

B. Corpus complexity and trajectory state. The mean call contains 2.20 arguments and 94.08 context words. Level 1 has shorter contexts and usually exposes a task-specific schema. Level 2 presents competing APIs in longer multi-turn trajectories. Level 3 includes a tool discovery stage: ToolSearcher is initially visible, its returned schema becomes part of the trajectory, and the agent must call the discovered operation or authenticate before execution. Twenty targets have a gold parameter profile that differs from the single schema profile recovered from the instruction. These cases include staged interfaces whose required fields change after authentication or verification. Table II summarizes these properties. The official API-Bank study introduced a runnable collection of 73 tools and annotated tool-use dialogues for planning, retrieval, and calling evaluation [15]; the present measurements refer only to the records and schemas in the evaluated package.

TABLE II

CORPUS AND TASK COMPLEXITY

Measure	Value
API-call targets	534
Assistant-response targets	478
Dialogue files with calls	261
Distinct gold tools	50
Distinct argument names	47
Mean arguments per call	2.20
Median context words	74.5
Schema/profile mismatches	20

C. First-pass agent. The first pass receives the API schemas attached to the current record and the latest four parsed dialogue events. Each schema document concatenates the split API name, description, parameter names, and parameter descriptions. Tool ranking uses BM25 with $k_1=1.5$ and $b=0.75$, following the probabilistic relevance framework [31]. For query q, schema document d, and term x, the implementation computes the standard saturation and length-normalized BM25 sum. Although the present scorer is lexical rather than spectral, rank-aggregation theory reinforces the distinction between recovering a useful candidate set and placing the correct item first under noisy comparisons [32]. The highest-scoring API is selected. The confidence assigned to that choice is the top probability of a softmax over ranked scores with temperature 0.35. The retrieval and evaluation utilities were implemented with NumPy, pandas, SciPy, and scikit-learn [33].

$$\mathrm{BM25}(d,q) = \Sigma_x \mathrm{IDF}(x) \cdot [f(x,d)(k_1+1)] / [f(x,d)+k_1(1-b+b|d|/\mathrm{avgdl})],\ k_1=1.5,\ b=0.75.$$

D. Local argument binding. After tool selection, the first-pass binder enumerates only parameters from the selected schema profile. It extracts opaque identifiers, names, account fields, e-mail addresses, numeric counts, formulas, booleans, free text, and temporal expressions with deterministic patterns. Local mode uses the same four-event window as tool selection and does not use stored defaults. Dates are normalized against the 2023 anchor carried by API-Bank instructions; timed intervals preserve first-to-last ordering; and explicit user answers following an assistant slot request receive priority over older mentions. The result is a serialized API name plus keyword-argument map.

E. Feedback gateway and critic. The evaluation environment compares each first call with the held-out target and returns success or failure. On a failure, the critic records one primary diagnosis in a fixed priority order: wrong tool, missing parameter, extra parameter, invalid format, or semantically wrong value. The diagnosis is rendered as a compact reflection trace that contains the failed component and the relevant trajectory stage. It does not disclose the gold replacement tool or gold field value. The repair policy remains fixed and re-evaluates both tool and arguments, which prevents category-specific heuristics from receiving different amounts of target information. The gateway accepts a call only when the tool and parameter set match and every value is execution-equivalent under the normalization rules defined below. Consequently, accepted first calls are never replaced. Figure 1 shows the complete control flow.

ISSN: 3071-4656

DOI: 10.66372/JGER.V2I1.2

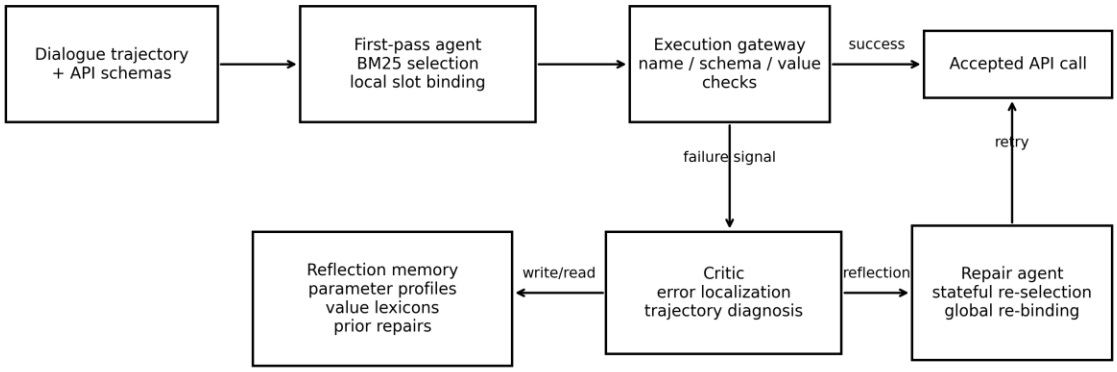

Fig. 1. Reflexion-guided first-pass, feedback, memory, and repair pipeline.

F. Trajectory-aware tool re-selection. The repair agent first isolates the active segment after the latest completed API event. For Levels 1 and 2, it ranks the record's candidate APIs with a state-aware score that combines BM25, character 3–5-gram similarity, action and entity evidence, explicit API mentions, and contradiction penalties. Completed helper tools are masked: ToolSearcher is suppressed after retrieval, and GetUserToken is suppressed after a token appears. For Level 3, the ranker operates over the global registry reconstructed from every schema in the package. It gives precedence to retrieval before a tool has been exposed and to authentication when a downstream schema requires a token. API names returned inside the trajectory receive strong recency evidence. This global step is essential because later Level-3 records still carry ToolSearcher as their attached schema even though the dialogue has revealed a different executable API.

G. Reflection memory and global rebinding. The repair binder reads the complete trajectory plus the active segment. Its reflection memory contains four objects estimated from training dialogues: ordered parameter profiles for each tool, value lexicons indexed by tool and field, the most frequent default value for each indexed field, and prior ToolSearcher context–keyword pairs. Parameter profiles resolve staged APIs such as password recovery, where one call requests username and e-mail and a later call uses verification status, code, and new password. Lexicon matching selects the latest value visibly grounded in the trajectory. Defaults fill a field only when direct extraction and lexicon matching fail. ToolSearcher keywords use the nearest stored active context under character-gram cosine similarity. Memory is read only after a failed first attempt, so the experiment separates ordinary retrieval from reflective recovery. The hierarchy is evidence first, lexicon second, and default last; a frequent value never overwrites a value explicitly grounded in the current trajectory. This memory implements the episodic component of Reflexion [18] without updating model weights.

H. Leakage control. All episodic profiles, lexicons, defaults, and ToolSearcher cases were fitted in five GroupKFold splits, with the dialogue filename used as the group. Every target in a test fold was therefore repaired with memory built from other dialogues only. Tool schema reconstruction is corpus-level metadata rather than target-label memory; gold tools and arguments from the held-out fold did not enter ranking weights, extraction patterns, profiles, lexicons, or defaults. The random seed was 20230817. Table III lists the exact configuration.

TABLE III

EXPERIMENTAL CONFIGURATION

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

Component	Setting
First-pass retrieval	BM25 over tool name, description, and parameter descriptions; four-event local context
First-pass arguments	Deterministic local slot rules; schema parameters only; no learned defaults
Critic feedback	Wrong tool, missing/extra argument, invalid format, or semantically incorrect value
Repair retrieval	Active-segment state machine for Levels 1–2; global API-library ranking for Level 3
Reflection memory	Fold-local parameter profiles, value lexicons, defaults, and ToolSearcher cases
Validation	Tool identity, parameter set, format constraints, and execution-normalized equivalence
Cross-validation	Five GroupKFold splits grouped by dialogue file
Random seed	20230817

I. Metrics. Top-1 and top-3 tool accuracy evaluate ranking. Parameter-name F1 treats argument keys as a set and assigns zero true positives when the tool is wrong. Strict value accuracy counts canonical string equality for gold fields. Strict call accuracy requires the correct tool, identical key set, and exact canonical values. Execution-normalized success uses the same tool and key requirements but accepts operationally equivalent values: formula whitespace is ignored; names are case-insensitive; list-like fields are collection-normalized; and free-text values pass when normalized token Jaccard similarity is at least 0.58 or containment is at least 0.86. These rules were fixed before aggregate scoring. Repair rate equals repaired initial failures divided by all initial failures, and retry rate equals the fraction of calls rejected by the gateway. Primary errors are mutually exclusive because the taxonomy uses the stated priority order. As a result, a call repaired from the wrong tool to the correct tool can move into the wrong-value category; the category counts must be interpreted together with total success rather than as independent error rates.

Execution success = 1[tool correct ∧ key sets equal ∧ every value execution-equivalent].

J. Statistical, calibration, and robustness analysis. The experiment used 5,000 paired bootstrap resamples to estimate 95% confidence intervals for success rates and first-to-final differences, following the bootstrap framework of Efron and Tibshirani [34]. The exact McNemar test evaluated the two discordant paired counts [35]. First-pass confidence was assessed with ten-bin expected calibration error and the Brier score, metrics commonly used to diagnose overconfidence [36]. Human-uncertainty distillation provides a domain-specific example of why confidence quality and accuracy should be reported separately [27]. Beyond neural calibration, uncertainty quantification for spectral and maximum-likelihood estimators illustrates why a point estimate should be distinguished from its distributional reliability [37]. Context robustness was measured by splitting targets into four equal-frequency context-length groups. Runtime was the median of five complete batches on an Intel Xeon Platinum 8573C under Python 3.13.5.

K. Failure injection and ablation. Five deterministic corruptions were applied to valid target calls. Wrong-tool injection replaced the tool with a different registry entry. Missing-field injection removed one gold field, wrong-value injection replaced the same field with an invalid scalar, format injection corrupted the first field governed by a date, time, count, Boolean, or e-mail constraint, and stale-state injection substituted an earlier visible token or identifier. The repair pipeline then regenerated the affected component. The ablation sequence starts with the first pass and adds, in order, global argument context without tool re-selection, stateful tool selection, global extraction, episodic memory, learned defaults, and success gating.

Results and Discussion

A. Dataset coverage. The 534 call targets span all three planning levels and 50 target APIs. Level 2 is the largest partition with 225 calls, while Level 3 contributes 135 calls whose discovered APIs are not fully represented by the attached candidate schema list. Figure 2 visualizes the distribution. The mean number of candidate APIs is 1.45 for Level 1, 2.76 for Level 2, and 1.00 for Level 3. The Level-3 value does not indicate an easier selection problem; it reflects the initial ToolSearcher-only presentation and makes trajectory reconstruction necessary.

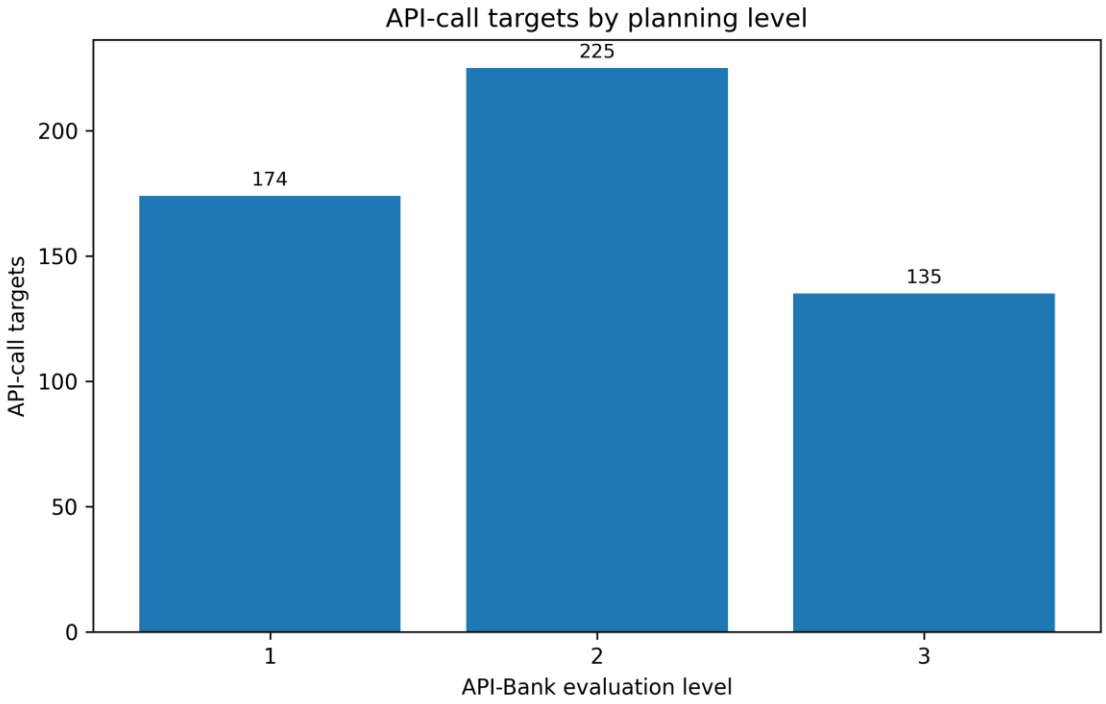

Fig. 2. Distribution of the 534 API-call targets across API-Bank planning levels.

B. Tool selection. Table IV compares seven ranking strategies. Choosing the first candidate reached 50.37% top-1 accuracy. BM25 over the API name, description, and parameter descriptions improved top-1 accuracy to 61.80% and top-3 accuracy to 83.15%. Character similarity produced 61.61%, and the fixed hybrid score produced 61.05%, showing that additional lexical features did not resolve trajectory state by themselves. The active-segment stateful ranker reached 77.15%. Combining that ranker with global Level-3 library reconstruction produced 92.51% top-1 and 97.57% top-3 accuracy. Combining lexical, action, entity, and trajectory evidence follows the same multi-signal design principle used in autonomous-driving perception fusion, although the modalities and prediction targets are different [38].

TABLE IV

OVERALL TOOL-SELECTION ACCURACY

Method	Top-1	Top-3	n
First candidate	50.37%	80.71%	534
BM25 name	59.18%	83.15%	534
BM25 full schema	61.80%	83.15%	534
Character 3–5 grams	61.61%	83.15%	534
Lexical hybrid	61.05%	83.15%	534
Stateful local	77.15%	82.21%	534

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

Method	Top-1	Top-3	n
Reflection combined	92.51%	97.57%	534

The level analysis in Table V and Figure 3 identifies the source of the gain. BM25 reached 82.76% on Level 1 and 62.67% on Level 2 but only 33.33% on Level 3. The one-third Level-3 score corresponds to initial ToolSearcher calls; subsequent discovered-tool calls remain lexically constrained to the helper schema. Global reflection raised Level-3 top-1 accuracy to 94.07%. Level 1 reached 99.43%, and Level 2 reached 86.22%. Thus, state tracking is not a small reranking adjustment: it changes which API library is considered at each trajectory stage.

TABLE V

TOP-1 TOOL ACCURACY BY PLANNING LEVEL

Level	BM25 full	Stateful local	Reflection combined	Reflection top-3	n
1	82.76%	99.43%	99.43%	100.00%	174
2	62.67%	86.22%	86.22%	97.78%	225
3	33.33%	33.33%	94.07%	94.07%	135

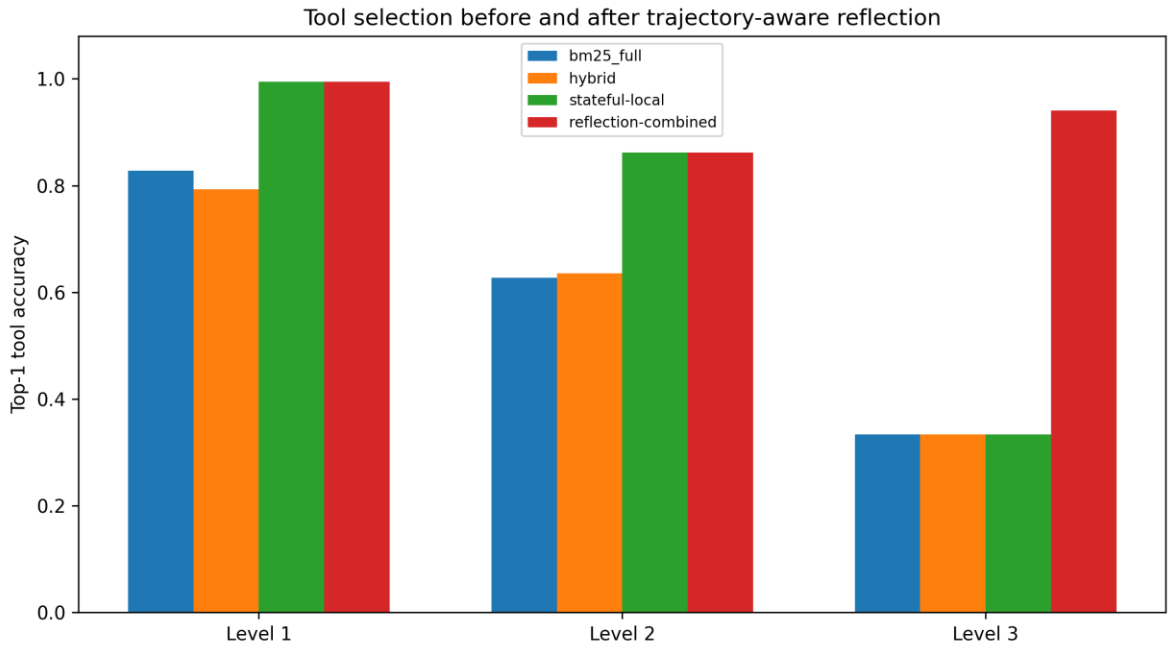

Fig. 3. Tool-selection accuracy before and after trajectory-aware re-selection.

Top-3 behavior adds an important qualification. BM25 placed the correct tool in its top three for 83.15% of calls, but a tool agent normally executes only the top-ranked action. The reflection ranker converted much of this latent recall into top-1 correctness, raising top-1 by 30.71 points while top-3 rose by 14.42 points. The smaller top-3 gain shows that lexical retrieval often surfaced a useful candidate but ordered it behind a completed or adjacent action. Active-state masking and action contradiction penalties supplied the information needed to promote that candidate.

C. Argument binding with the gold tool fixed. Table VI isolates parameter generation from tool selection. Local rules achieved parameter-name F1 of 87.11%, strict value accuracy of 62.08%, strict call accuracy of 47.38%,

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

and execution-normalized success of 55.43%. Reading the full trajectory increased execution-normalized success to 61.24%. Adding fold-local episodic lexicons raised it to 70.04%, and adding learned defaults reached 72.10%. The complete memory configuration produced parameter-name F1 of 99.97% and semantic value accuracy of 83.29%. The 16.67-point success gain from local rules to full memory confirms that argument repair is independent of the larger tool-selection gain.

TABLE VI

ARGUMENT BINDING WITH THE GOLD TOOL FIXED

Variant	Param-name F1	Strict value	Semantic value	Strict call	Execution success
Local rules	87.11%	62.08%	70.22%	47.38%	55.43%
Global rules	92.16%	66.65%	74.91%	52.06%	61.24%
Global + memory	95.55%	74.72%	81.59%	61.99%	70.04%
Global + memory + defaults	99.97%	76.42%	83.29%	64.04%	72.10%

The memory components also address the 20 observed schema/profile mismatches. A schema-only binder must choose one fixed field set, whereas the empirical profile distinguishes stages of the same named operation. This effect explains the near-perfect parameter-name F1 of the full binder. The remaining gap between parameter-name F1 (99.97%) and execution success (72.10%) is therefore almost entirely a value-grounding problem, not a structural omission problem. Defaults add only 2.06 success points beyond memory without defaults, indicating that most gains come from trajectory evidence and learned profiles rather than from blindly inserting frequent constants.

D. End-to-end first-pass and final performance. The full paired results appear in Table VII and Figure 4. Across all targets, first-pass tool accuracy was 61.80%, strict call accuracy was 29.03%, and execution-normalized success was 36.52%. The gated final predictions reached 93.82%, 58.61%, and 70.41%, respectively. The retry rate was 63.48% because 339 calls failed on the first pass. Of those failures, 181 were repaired, yielding a conditional repair rate of 53.39%.

TABLE VII

END-TO-END FIRST-PASS AND FINAL PERFORMANCE

Level	First tool	Final tool	First strict	Final strict	First exec.	Final exec.	Gain	Retry
1	82.76%	99.43%	41.95%	63.79%	46.55%	72.99%	+26.44	53.45%
2	62.67%	89.33%	35.56%	64.89%	36.89%	70.22%	+33.33	63.11%
3	33.33%	94.07%	1.48%	41.48%	22.96%	67.41%	+44.44	77.04%
All	61.80%	93.82%	29.03%	58.61%	36.52%	70.41%	+33.90	63.48%

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

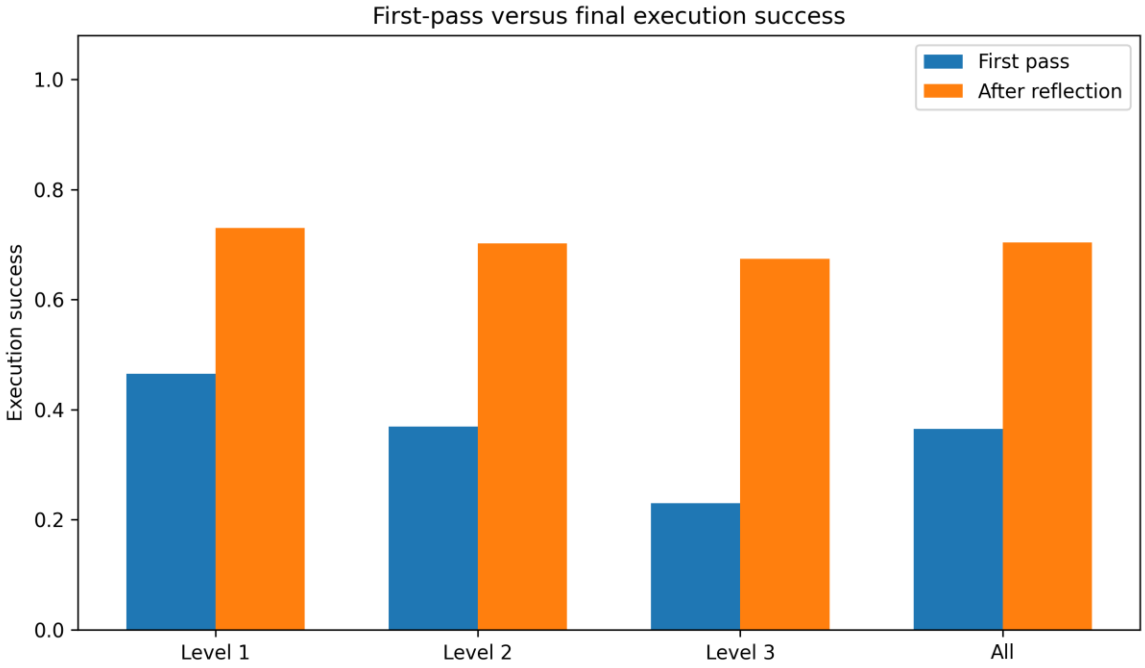

Fig. 4. Execution-normalized success on the first pass and after gated reflection.

Performance improved at every level. Level-1 execution-normalized success rose from 46.55% to 72.99%, a gain of 26.44 points. Level 2 rose from 36.89% to 70.22%, a gain of 33.33 points. Level 3 rose from 22.96% to 67.41%, a gain of 44.44 points. The increasing gain tracks the amount of missing state in the local first pass. The final Level-3 score remains lower than its 94.07% tool accuracy because tool discovery is largely solved while exact downstream values and parameter profiles remain difficult.

E. Pairwise repair behavior. Table VIII and Figure 5 report the transition matrix. The pipeline changed 181 calls from failure to success, left 158 as failures, preserved 195 successes, and changed no success into a failure. The absence of regressions follows from the gateway design: the repair candidate replaces the first candidate only after the first candidate fails the execution-equivalence check. This result distinguishes gated repair from unconditional rewriting, where a second pass can damage a correct first answer. The retain-on-success rule is especially important in constrained decision settings, where an unnecessary policy update can violate budget or risk objectives; risk-aware auto-bidding provides one such example [39].

TABLE VIII

FIRST-TO-FINAL EXECUTION TRANSITION MATRIX

First-pass state	Final failure	Final success
First failure	158	181
First success	0	195

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

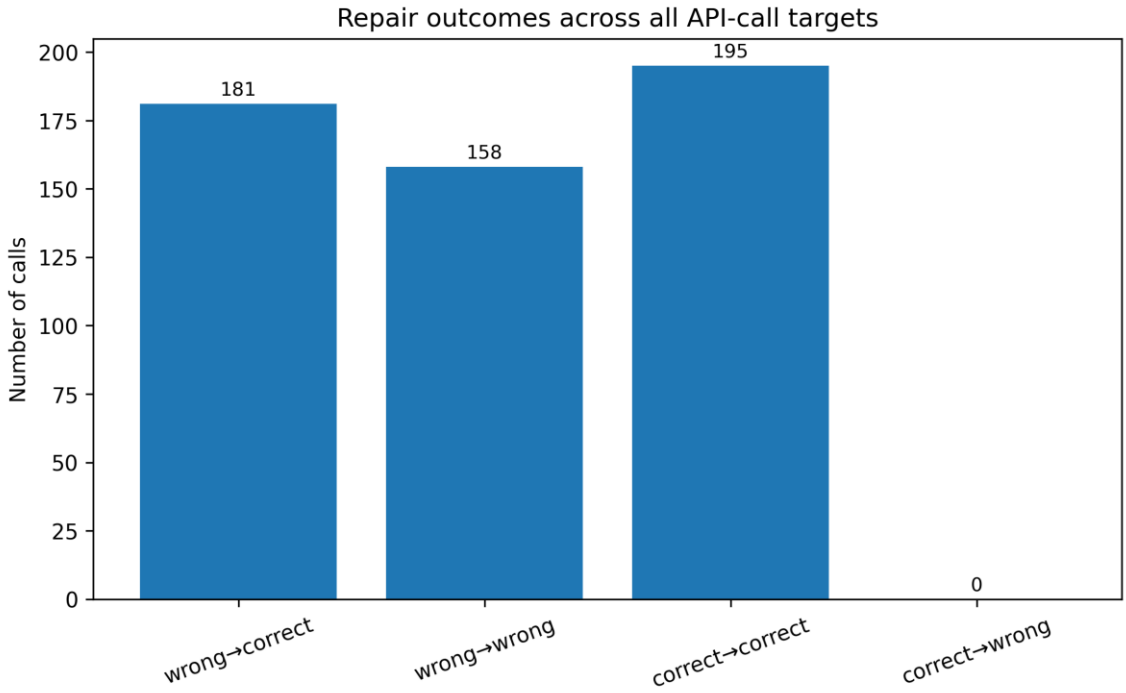

Fig. 5. Counts of repaired, unrepaired, preserved, and regressed calls.

F. Error redistribution. Table IX and Figure 6 show the primary error assigned to every call. Wrong-tool errors fell from 204 (38.20%) to 33 (6.18%), and missing-parameter errors fell from 79 (14.79%) to one (0.19%). Wrong-value errors increased from 56 (10.49%) to 124 (23.22%). This increase is a reclassification effect rather than a decline: many calls that initially had the wrong tool or missing fields became structurally correct and therefore exposed value errors as the remaining highest-priority defect. The residual set is consequently concentrated in temporal normalization, free-text content, and opaque identifiers rather than API identity. The remaining emphasis on value errors also parallels IoT anomaly-detection settings, where a correctly structured detector can still mischaracterize the operational state represented by observed traffic [40].

TABLE IX

PRIMARY ERROR TAXONOMY BEFORE AND AFTER CORRECTION

Stage	Wrong tool	Missing parameter	Extra parameter	Invalid format	Wrong value	Success
First pass	204 (38.20%)	79 (14.79%)	0	0	56 (10.49%)	195 (36.52%)
After reflection	33 (6.18%)	1 (0.19%)	0	0	124 (23.22%)	376 (70.41%)

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

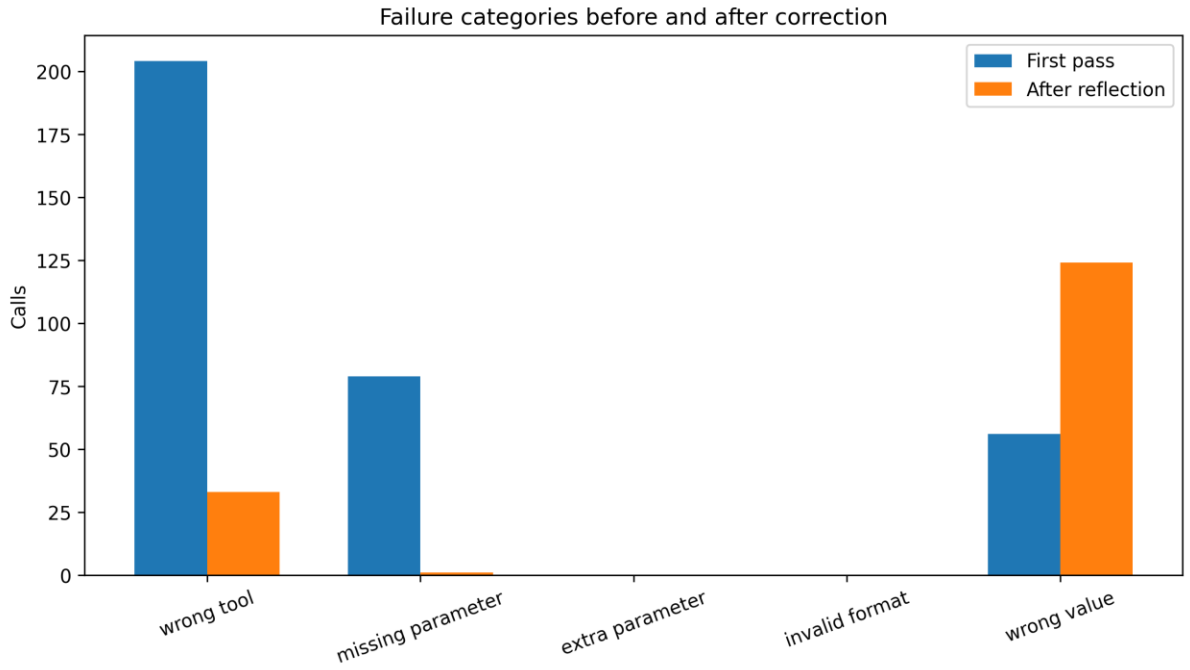

Fig. 6. Primary failure categories before and after correction.

G. Ablation analysis. Table X quantifies each component. Rebinding arguments with full context and memory while retaining the first-pass tool raised success from 36.52% to 44.57%. Stateful tool re-selection with only local argument rules reached 52.62%. Global context rules reached 58.24%, episodic memory without defaults reached 65.54%, and full direct repair reached 67.04%. Success gating raised execution-normalized success to 70.41%. Tool re-selection and argument memory therefore contribute complementary gains: removing either one loses more than ten points relative to the complete system. A related systems lesson appears in noisy-neighbor VM degradation modeling, where residual signals are fused because no single local measurement isolates interference [41].

TABLE X

ABLATION STUDY

Variant	Tool accuracy	Param-name F1	Semantic value	Strict call	Execution success
First pass	61.80%	55.52%	45.41%	29.03%	36.52%
No tool re-selection	61.80%	61.77%	51.37%	38.39%	44.57%
Stateful tool only	92.51%	82.41%	66.46%	44.57%	52.62%
Global-context rules	92.51%	87.23%	71.05%	49.06%	58.24%
Episodic memory, no defaults	92.51%	89.59%	76.37%	57.49%	65.54%
Full direct repair	92.51%	92.48%	77.48%	58.99%	67.04%
Full gated system	93.82%	93.79%	80.85%	58.61%	70.41%

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

Direct repair achieved slightly higher strict call accuracy than gated repair, 58.99% versus 58.61%, while its execution-normalized success was lower, 67.04% versus 70.41%. The difference follows from semantic normalization. A first-pass call can be execution-equivalent but not string-identical to the target; the gateway preserves that call, increasing operational success while leaving the stricter string metric unchanged. This divergence supports reporting both metrics instead of treating exact serialization as the sole definition of a valid call.

H. Controlled failure injection. Table XI tests correction after known corruption. All 17 stale-state cases were repaired because the global binder selected the latest visible identifier. Missing-field and wrong-value corruption each covered 527 calls and reached 83.68% execution-normalized repair success. Invalid-format corruption covered 206 calls and reached 83.01%. Wrong-tool corruption covered all 534 calls and reached 67.04%, matching the full direct-repair variant because the pipeline had to reconstruct both tool and arguments. These results show that field-level defects are easier to repair once the tool is fixed, whereas a wrong tool propagates uncertainty into the complete call. DDoS mitigation studies likewise couple detection with the selection of a resource-allocation response [42], making wrong-tool corruption closer to choosing the wrong mitigation action than merely misformatting a field.

TABLE XI

REPAIR UNDER DETERMINISTIC FAILURE INJECTION

Injected failure	Repaired tool	Semantic value	Strict repair	Execution repair	n
Stale state	100.00%	100.00%	100.00%	100.00%	17
Wrong value	100.00%	88.99%	77.04%	83.68%	527
Missing parameter	100.00%	88.99%	77.04%	83.68%	527
Invalid format	100.00%	93.97%	83.01%	83.01%	206
Wrong tool	92.51%	77.48%	58.99%	67.04%	534

I. Statistical reliability. The paired estimates in Table XII confirm that the aggregate gain is not explained by sampling variation. First-pass execution-normalized success was 36.52% with a 95% bootstrap interval of 32.40–40.64%. Final success was 70.41% with an interval of 66.48–74.16%. The paired increase was 33.90 points with an interval of 29.96–38.01 points. The exact McNemar comparison used zero first-correct/final-wrong pairs and 181 first-wrong/final-correct pairs, producing $p=6.53\times10^{-55}$. Strict call accuracy increased by 29.59 points with a 95% interval of 25.84–33.52 points.

TABLE XII

PAIRED STATISTICAL ANALYSIS

Metric	Estimate	95% CI	p-value
First-pass execution success	36.52%	32.40–40.64%	—
Final execution success	70.41%	66.48–74.16%	—
Paired execution-success gain	+33.90 points	29.96–38.01	6.53×10^{-55}
Paired strict-call gain	+29.59 points	25.84–33.52	—
First correct / final wrong	0	—	6.53×10^{-55}
First wrong / final correct	181	—	6.53×10^{-55}

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

J. Confidence calibration. The BM25 first-pass confidence was strongly overconfident. Expected calibration error was 0.369 and the Brier score was 0.374. Of 534 targets, 499 received confidence in the 0.9–1.0 bin, but accuracy in that bin was 61.12%. Figure 7 shows the reliability curve. A threshold on this raw confidence would therefore miss many failures. The structured execution signal is substantially more informative than the selector's uncalibrated score, consistent with prior findings that modern predictive systems require explicit calibration analysis [36].

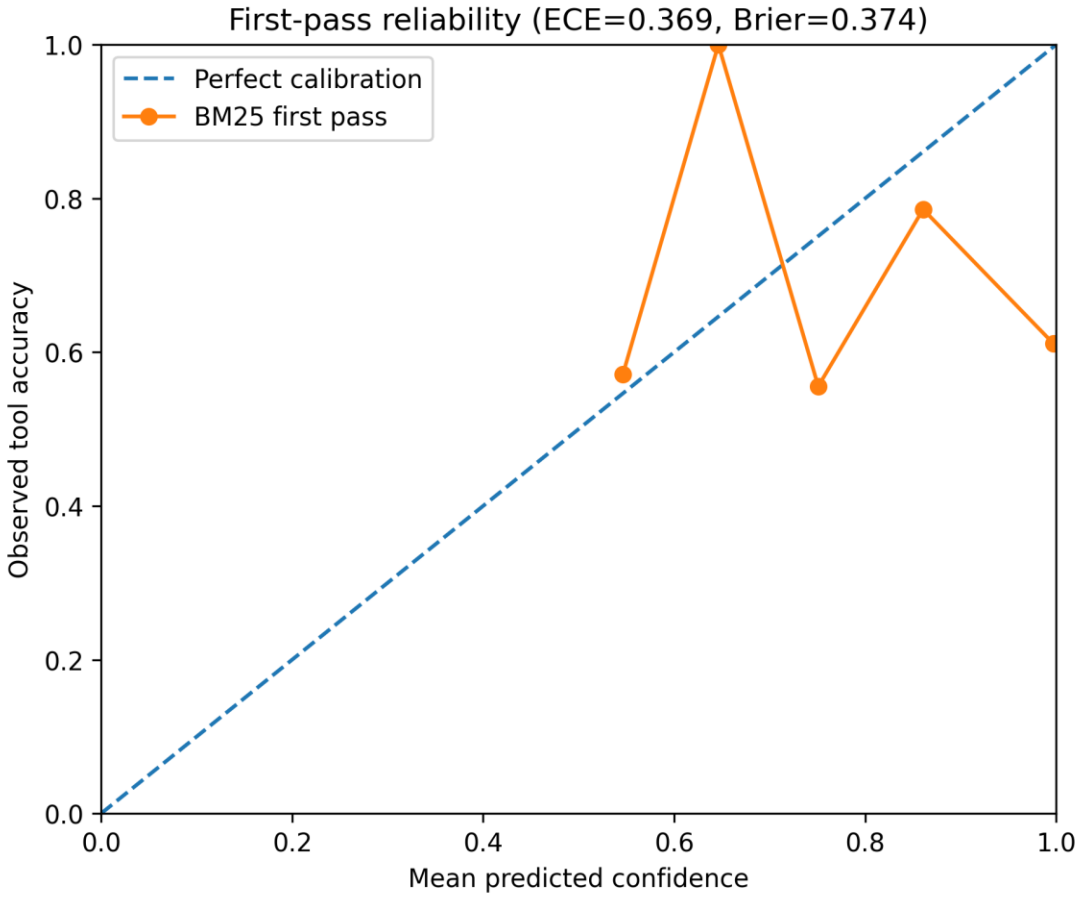

Fig. 7. Reliability of the BM25 first-pass tool confidence.

K. Robustness to long contexts. Table XIII and Figure 8 partition calls by context length. First-pass success declined from 58.52% in the shortest quartile to 17.42% in the longest quartile. Final success also declined, but much less sharply, from 76.30% to 65.15%. The correction gain was 17.78 points in the shortest group and 47.73 points in the longest group. Active-segment reconstruction therefore acts as a recency filter: it prevents completed tasks and obsolete identifiers from dominating selection while retaining the full trajectory for state fields that must be copied forward. Long-document retrieval studies similarly show that decisive evidence can be dispersed across clauses, tables, and pages [43]; the agent analogue is that required identifiers and state can be distributed across earlier turns.

TABLE XIII

ROBUSTNESS BY DIALOGUE CONTEXT LENGTH

Quartile	Word range	Mean words	First success	Final success	Gain	n
Q1 shortest	10–43	28.06	58.52%	76.30%	+17.78	135

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

Quartile	Word range	Mean words	First success	Final success	Gain	n
Q2	44–74	60.28	47.73%	74.24%	+26.51	132
Q3	75–124	95.28	22.22%	65.93%	+43.70	135
Q4 longest	125–411	194.17	17.42%	65.15%	+47.73	132

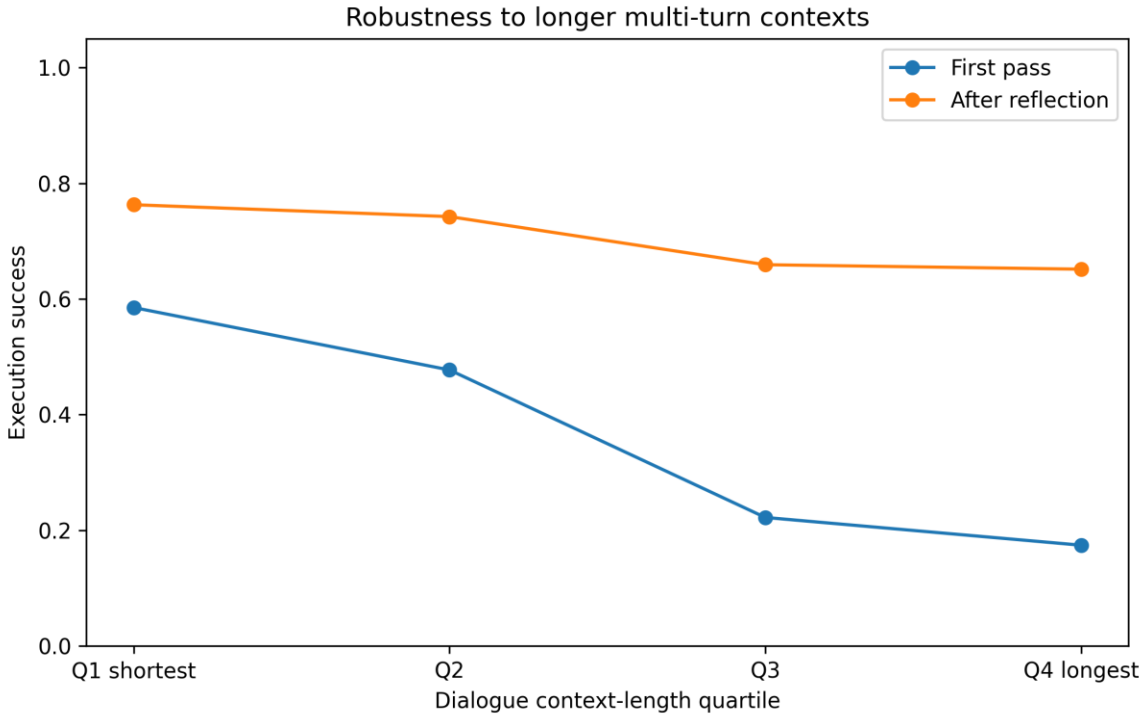

Fig. 8. First-pass and final success across context-length quartiles.

L. Efficiency. Table XIV reports local computation time. The median first-pass batch took 0.0900 s, or 0.169 ms per call. The repair batch took 0.5920 s, or 1.109 ms per call. Combining these values with the observed 63.48% retry rate gives an expected mean of 0.872 ms per request for the deterministic harness. The repair stage is 6.58 times slower than the first pass because it searches a larger registry and performs global extraction, but both stages remain negligible relative to remote model inference or network API execution.

TABLE XIV

LOCAL RUNTIME AND ENVIRONMENT

Measure	Value
First-pass median batch time	0.0900 s
First-pass median time per call	0.169 ms
Repair-pass median batch time	0.5920 s
Repair-pass median time per call	1.109 ms
Observed retry rate	63.48%
Expected mean time per request	0.872 ms

ISSN: 3071-4656
DOI: 10.66372/JGER.V2I1.2

Measure	Value
Python	3.13.5
Processor	Intel Xeon Platinum 8573C

The calibration and context results jointly explain why a selective retry policy needs explicit operational checks. The longest contexts produced the lowest first-pass success while still receiving concentrated high BM25 confidence. A confidence-only trigger would accept many of these calls. A schema validator can detect missing or malformed fields, but wrong-tool and stale-value failures require environment state or postcondition evidence. The practical design implication is a layered gateway: local syntax checks first, API or simulator feedback second, and reflective repair only when either layer rejects the call.

M. Relation to prior agent designs. The results align with the central claim of Reflexion that feedback plus episodic memory improves subsequent trials without parameter updates [18], but they localize the gain to concrete call components. ReAct provides the broader reason–act loop [1]; CRITIC emphasizes external feedback [21]; and Self-Refine emphasizes iterative revision [19]. The present evaluation adds a strict paired decomposition: trajectory-aware re-selection primarily removes wrong tools, while episodic rebinding primarily removes missing fields and recovers values. Unlike search over many reasoning branches [22] or self-consistency sampling [24], the system uses one deterministic retry, making every transition auditable in the saved prediction files. Component-level complementarity also appears in perception models that combine receptive-field attention with triplet attention [44]; the architectural setting is different, but the comparison reinforces that distinct failure modes can require distinct corrective components.

Limitations

The 70.41% gated result uses a benchmark success signal to decide whether the first call should be retained. This is an oracle-feedback evaluation of repair once a failure is detected, not a production failure detector. The direct repair result of 67.04% does not require this acceptance oracle and provides the corresponding always-retry baseline. A deployed system must obtain equivalent feedback from API status codes, schema validators, postcondition checks, or human review.

Execution-normalized success is a call-equivalence metric, not live execution against external services. The package contains target calls and dialogue traces but not a complete network environment for all 50 target APIs. The study therefore establishes correctness of tool identity, field structure, formatting, and normalized values; it does not measure server-side side effects, authentication failures outside the trace, rate limits, latency, or downstream response quality.

The repair policy is deterministic and incorporates domain rules for the API names and field conventions present in API-Bank. This choice makes the experiment fully inspectable and removes model-version variance, but it does not establish transfer to unseen naming systems or natural-language schemas. The fold split prevents dialogue-level memory leakage, yet parameter conventions and frequent default values recur across different dialogues in the same benchmark. Cross-benchmark validation is required to quantify transfer. Language transfer is also untested. Bilingual computer-science education research shows that the language of instruction can affect engagement and learning in technical settings [45], so multilingual API descriptions and user requests require separate evaluation. Nor do these symbolic-call results transfer directly to clinical prediction: AI-based minimal residual disease evaluation for multiple myeloma [46] and DenseNet-based cancer-image classification [47] use different data-generating processes, error costs, and validation assumptions.

The evaluated package is a subset of the broader API-Bank resources described in the original publication [15]. Its 534 call targets are sufficient for paired significance testing but do not cover every API or domain in the full

training collection. Stale-state injection also contains only 17 naturally observable substitutions, so its 100% repair rate has a much smaller support than the other corruption results.

Local timing excludes language-model inference, remote retrieval, and API execution. The reported millisecond values measure only parsing, ranking, extraction, and validation on one CPU. They characterize algorithmic overhead and do not predict end-to-end latency for an LLM-backed service.

Conclusion

A single local tool call is an inadequate reliability strategy for multi-turn API agents. On 534 API-Bank call targets, BM25 selection plus local slot binding achieved 36.52% execution-normalized success. A single Reflexion-guided retry that reconstructed trajectory state, searched the appropriate API library, and used fold-local episodic argument memory increased success to 70.41% and strict call accuracy to 58.61%. The pipeline repaired 181 initial failures, preserved every initial success, and produced a 33.90-point paired gain with an exact McNemar p-value of 6.53×10^{-55}.

The decomposition identifies where the improvement originates. Global trajectory reasoning raised tool selection to 92.51% in direct evaluation and solved most Level-3 discovery errors. Full-context argument binding, empirical profiles, lexicons, and defaults raised gold-tool execution-normalized success from 55.43% to 72.10%. Residual failures are dominated by incorrect values rather than missing structure. Future systems should therefore combine stateful tool routing with stronger value-grounding checks, calibrated failure detection, and executable postconditions. The complete call records, ablations, failure injections, figures, and deterministic code provide a reproducible basis for that next step.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in Proc. 11th Int. Conf. Learning Representations, 2023.

[2] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in Advances in Neural Information Processing Systems, vol. 36, 2023.

[3] E. Karpas et al., "MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning," arXiv:2205.00445, 2022.

[4] R. Nakano et al., "WebGPT: Browser-assisted question-answering with human feedback," arXiv:2112.09332, 2021.

[5] M. Ahn et al., "Do as I can, not as I say: Grounding language in robotic affordances," in Proc. Conf. Robot Learning, 2022, pp. 287–318.

[6] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459–9474.

[7] Y. Li, "Findable then Explainable: Retrieval–Summary Integration for Code Intelligence on a Lightweight CodeSearchNet Subset," J. Adv. Comput. Syst., vol. 4, no. 7, pp. 65–82, Jul. 2024, doi: 10.69987/JACS.2024.40706.

[8] S. Zhou, Z. Li, and E. Wang, "Evidence-Grounded RAG for Tokenized Trade Receivable Disclosure QA under U.S. Capital Market Standards," J. Adv. Comput. Syst., vol. 3, no. 7, pp. 41–57, Jul. 2023, doi: 10.69987/JACS.2023.30704.

[9] Y. Qin et al., "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," arXiv:2307.16789, 2023.

[10] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, “Gorilla: Large language model connected with massive APIs,” arXiv:2305.15334, 2023.

[11] K. Xu, H. Zhou, H. Zheng, M. Zhu, and Q. Xin, “Intelligent classification and personalized recommendation of e-commerce products based on machine learning,” in Proc. 6th Int. Conf. Computing and Data Science (ICCDS), 2024.

[12] Q. Tang et al., “ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases,” arXiv:2306.05301, 2023.

[13] Y. Song et al., “RestGPT: Connecting large language models with real-world RESTful APIs,” arXiv:2306.06624, 2023.

[14] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, “HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face,” in Advances in Neural Information Processing Systems, vol. 36, 2023.

[15] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li, “API-Bank: A comprehensive benchmark for tool-augmented LLMs,” in Proc. 2023 Conf. Empirical Methods in Natural Language Processing, Singapore, 2023, pp. 3102–3116.

[16] Y. He, Y. Pan, Y. Wang, S. Du, and Q. Xin, “Intelligent Fault Analysis with AIOps Technology,” J. Theory Pract. Eng. Sci., vol. 4, no. 1, pp. 94–100, Feb. 2024, doi: 10.53469/jtpes.2024.04(01).13.

[17] J. Nie and D. Zheng, “Ambiguity-Aware HDFS Log Anomaly Detection with Retrieval-Augmented Failure Narratives and Selective Refusal,” J. Adv. Comput. Syst., vol. 3, no. 1, pp. 66–80, Jan. 2023, doi: 10.69987/JACS.2023.30105.

[18] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: Language agents with verbal reinforcement learning,” in Advances in Neural Information Processing Systems, vol. 36, 2023, pp. 8634–8652.

[19] A. Madaan et al., “Self-Refine: Iterative refinement with self-feedback,” in Advances in Neural Information Processing Systems, vol. 36, 2023.

[20] H. Zhang, “DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models,” J. Adv. Comput. Syst., vol. 3, no. 7, pp. 24–40, Jul. 2023, doi: 10.69987/JACS.2023.30703.

[21] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Du, and W. Chen, “CRITIC: Large language models can self-correct with tool-interactive critiquing,” arXiv:2305.11738, 2023.

[22] S. Yao et al., “Tree of thoughts: Deliberate problem solving with large language models,” in Advances in Neural Information Processing Systems, vol. 36, 2023.

[23] J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” in Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 24824–24837.

[24] X. Wang et al., “Self-consistency improves chain of thought reasoning in language models,” in Proc. 11th Int. Conf. Learning Representations, 2023.

[25] L. Gao et al., “PAL: Program-aided language models,” in Proc. 40th Int. Conf. Machine Learning, vol. 202, 2023, pp. 10764–10799.

[26] Q. Xin, R. Song, Z. Wang, Z. Xu, and F. Zhao, “Enhancing Bank Credit Risk Management Using the C5.0 Decision Tree Algorithm,” J. Comput. Technol. Appl. Math., vol. 1, no. 4, pp. 100–107, Nov. 2024, doi: 10.5281/zenodo.14032041.

[27] Z. S. Zhong, R. Ma, and H. Zhao, “Human-Uncertainty Distillation for Calibrated Vision Models on CIFAR-10H,” J. Adv. Comput. Syst., vol. 3, no. 2, pp. 77–89, Feb. 2023, doi: 10.69987/JACS.2023.30206.

[28] T. Yang, Q. Xin, X. Zhan, S. Zhuang, and H. Li, “Enhancing Financial Services through Big Data and AI-Driven Customer Insights and Risk Analysis,” J. Knowl. Learn. Sci. Technol., vol. 3, no. 3, pp. 53–62, Jul. 2024, doi: 10.60087/jklst.vol3.n3.p53-62.

[29] G. Mialon et al., “Augmented language models: A survey,” Trans. Mach. Learn. Res., 2023.

[30] J. Wang, Q. Xin, Y. Liu, J. Wang, and T. Yang, “Predicting Enterprise Marketing Decision Making with Intelligent Data-Driven Approaches,” J. Ind. Eng. Appl. Sci., vol. 2, no. 3, pp. 12–19, Jun. 2024, doi: 10.5281/zenodo.11357252.

[31] S. Robertson and H. Zaragoza, “The probabilistic relevance framework: BM25 and beyond,” Found. Trends Inf. Retr., vol. 3, no. 4, pp. 333–389, 2009.

[32] Z. S. Zhong and S. Ling, “Improved theoretical guarantee for rank aggregation via spectral method,” Inf. Inference: J. IMA, vol. 13, no. 3, Art. no. iaae020, Sep. 2024, doi: 10.1093/imaiai/iaae020.

[33] F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, 2011.

[34] B. Efron and R. J. Tibshirani, An Introduction to the Bootstrap. New York, NY, USA: Chapman & Hall, 1993.

[35] Q. McNemar, “Note on the sampling error of the difference between correlated proportions or percentages,” Psychometrika, vol. 12, no. 2, pp. 153–157, 1947.

[36] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” in Proc. 34th Int. Conf. Machine Learning, 2017, pp. 1321–1330.

[37] Z. S. Zhong and S. Ling, “Uncertainty quantification of spectral estimator and MLE for orthogonal group synchronization,” arXiv:2408.05944, Aug. 2024, doi: 10.48550/arXiv.2408.05944.

[38] Y. Wang, S. Du, Q. Xin, Y. He, and W. Qian, “Autonomous Driving System Driven by Artificial Intelligence Perception Fusion,” Acad. J. Sci. Technol., vol. 9, no. 2, pp. 193–198, Feb. 2024, doi: 10.54097/e0b9ak47.

[39] H. Zhang, “Risk-Aware Budget-Constrained Auto-Bidding Under First-Price RTB: A Distributional Constrained Deep Reinforcement Learning Framework,” J. Adv. Comput. Syst., vol. 4, no. 6, pp. 30–47, Jun. 2024, doi: 10.69987/JACS.2024.40603.

[40] Q. Xin, Z. Xu, L. Guo, F. Zhao, and B. Wu, “IoT traffic classification and anomaly detection method based on deep autoencoders,” Appl. Comput. Eng., vol. 69, no. 1, pp. 64–70, 2024, doi: 10.54254/2755-2721/69/20241511.

[41] J. Nie and D. Zheng, “Noisy-Neighbor-Aware VM Degradation Risk Modeling with Unsupervised Residual Fusion,” J. Adv. Comput. Syst., vol. 4, no. 4, pp. 112–123, Apr. 2024, doi: 10.69987/JACS.2024.40409.

[42] B. Wang, Y. He, Z. Shui, Q. Xin, and H. Lei, “Predictive optimization of DDoS attack mitigation in distributed systems using machine learning,” Appl. Comput. Eng., vol. 64, no. 1, pp. 94–99, 2024, doi: 10.54254/2755-2721/64/20241350.

[43] S. Zhou, Z. Li, and E. Wang, “Long-Document RAG for Contractual and Insurance Clause Analysis in Receivables RWA Structures,” J. Adv. Comput. Syst., vol. 4, no. 8, pp. 88–104, Aug. 2024, doi: 10.69987/JACS.2024.40810.

[44] Z. Ling, Q. Xin, Y. Lin, G. Su, and Z. Shui, “Optimization of autonomous driving image detection based on RFAConv and triplet attention,” Appl. Comput. Eng., vol. 77, no. 1, pp. 210–217, 2024, doi: 10.54254/2755-2721/77/2024MA0067.

[45] A. G. S. Raj et al., “Impact of Bilingual CS Education on Student Learning and Engagement in a Data Structures Course,” in Proc. 19th Koli Calling Int. Conf. Computing Education Research, 2019, pp. 1–10, doi: 10.1145/3364510.3364518.

[46] J. Chen, J. Xiong, Y. Wang, Q. Xin, and H. Zhou, “Implementation of an AI-based MRD Evaluation and Prediction Model for Multiple Myeloma,” Front. Comput. Intell. Syst., vol. 6, no. 3, pp. 127–131, Jan. 2024, doi: 10.54097/zJ4MnbWW.

[47] Z. Zhong, M. Zheng, H. Mai, J. Zhao, and X. Liu, “Cancer image classification based on DenseNet model,” J. Phys.: Conf. Ser., vol. 1651, no. 1, Art. no. 012143, Nov. 2020, doi: 10.1088/1742-6596/1651/1/012143.