IEL: INTENT-LEVEL EXPERIENCE LEARNING FOR LLM-BASED CUSTOMER SERVICE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents increasingly operate in interactive service environments where a single dialogue may contain multiple user intents with different outcomes. This creates a granularity mismatch for experience learning: episode-level success or failure cannot determine which intent-specific behaviors should be retained, discarded, or reused. We introduce Intent-level Experience Learning (IEL), which treats a cross-turn intent instance as the unit of experience attribution and procedural learning. IEL first reconstructs intent instances from multi-turn interactions, attributes dialogue and tool evidence to individual instances, and verifies their outcomes using intent-specific goals and completion criteria. It then induces reusable procedures from verified successful instances, while using verified failures, when available, to identify recurring error patterns. Taxonomy-absent intents are accumulated as candidates and may subsequently be merged with existing intents or promoted to new canonical intents as evidence grows. This design implements the principle of credit assignment before procedural abstraction, where credit assignment refers to associating interaction evidence and outcomes with individual intent instances rather than with the entire episode. We evaluate IEL on ECom-Bench and a cross-category extension covering previously unseen product categories, using three executor LLMs. IEL obtains the highest mean score in all evaluated executor–metric combinations, with improvements of up to 7.58 percentage points in pass^1 and 13.18 points in pass^3 in the in-domain setting. These results suggest that intent-level experience attribution provides a useful basis for learning reusable procedures from mixed-outcome service interactions.

1 INTRODUCTION

Large language model (LLM) agents are increasingly studied in interactive service scenarios, where they must communicate with users, invoke tools, and satisfy domain-specific operational requirements (Yao et al., 2023; 2025; Wang et al., 2025a). Task-oriented dialogue research has explicitly modeled multi-intent user goals and dialogues that require multiple subgoals (Moghe et al., 2023; Kaiser et al., 2024). In tool-mediated service interactions, these goals need not share an outcome: a user may ask to track an order, modify an address, and inquire about a return policy, while some goals succeed, others fail, and still others remain unresolved. This raises a fundamental question for experience learning: how can an agent extract reusable experience from multi-turn dialogues when different parts of the same episode have different outcomes? We study this problem in tool-using service agents, where intent boundaries and intent-level outcomes are not annotated and the executor model remains fixed throughout experience accumulation and evaluation.

Existing approaches to agent experience reuse either retain or distill prior interactions into reusable lessons and guidelines (Shinn et al., 2023; Zhao et al., 2024; Fu et al., 2024), or abstract interaction trajectories into reusable workflows and procedural memory (Wang et al., 2025b; Fang et al., 2026). Despite differences in representation, these approaches commonly derive reusable experience from task- or episode-level trajectories and feedback. This creates a supervision-granularity mismatch in mixed-outcome dialogues: a successful episode does not imply that every intent was correctly handled, while a failed episode may still contain reusable successful behavior. Figure 1 illustrates this distinction: episode-level methods retain or abstract a mixed-outcome dialogue as a single experience unit, whereas IEL first separates cross-turn intent instances and verifies their outcomes before

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.

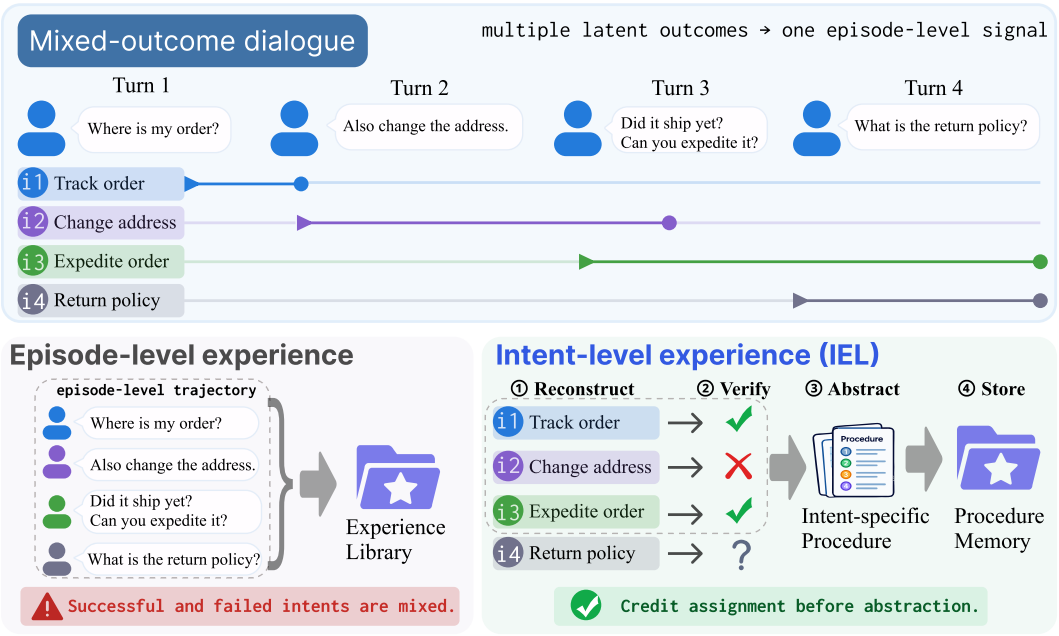

Figure 1: Episode-level versus intent-level experience learning in mixed-outcome dialogues. Episode-level approaches derive retained or abstracted experience from trajectory-level signals without first resolving intent-specific outcomes, which can conflate independently successful and failed goals. IEL instead reconstructs cross-turn intent instances, verifies their outcomes separately, and performs procedural abstraction only after intent-level credit assignment.

procedural abstraction. Formally, consider an episode containing intent instances $\mathcal{I}_e = \{i_1, \ldots, i_m\}$ with unobserved intent-level outcomes y_j. The benchmark provides only an episode-level outcome r_e that aggregates these outcomes. When $m > 1$, the same episode-level outcome can correspond to multiple configurations of successful and failed intents, making it insufficient to determine which parts of the trajectory should become positive or negative experience. Directly abstracting the full trajectory can therefore conflate behaviors associated with independently successful and unsuccessful goals.

To address this mismatch, we propose Intent-level Experience Learning (IEL), which treats each cross-turn intent instance as an independent unit of experience. IEL reconstructs intent instances and attributes interaction evidence to them, verifies their outcomes with explicit uncertainty, and only then induces reusable intent-specific procedures. This ordering—credit assignment before procedural abstraction—separates independently successful and failed goals before they are incorporated into reusable experience. IEL additionally accumulates recurring taxonomy-absent goals as candidates that can be incorporated into the evolving intent taxonomy.

Our contributions are threefold:

- We identify a supervision-granularity mismatch in experience learning for multi-intent service agents: episode-level outcomes can conflate independently successful and unsuccessful goals. We formulate intent instances as the unit of experience attribution and procedural reuse.
- We introduce IEL, a non-parametric framework that performs intent-level credit assignment through cross-turn intent reconstruction, evidence-based outcome verification, and intent-specific procedure induction, while supporting recurring taxonomy-absent goals.
- We evaluate IEL on ECOM-BENCH and ECOM-BENCH-XCAT with three executor LLMs. IEL obtains the highest mean score across all evaluated executor–metric combinations, while ablations and cross-category experiments support the value of intent-level organization and transfer under a shared service and tool interface.

2 Related Work

Tool-using agents for interactive service tasks. Large language models are increasingly studied as agents that interleave reasoning, dialogue, and external tool use. ReAct (Yao et al., 2023) established a general reasoning-and-acting paradigm, while subsequent benchmarks and frameworks have examined agent behavior in interactive environments and large-scale tool-use settings. AgentBench (Liu et al., 2024) evaluates LLM agents across diverse interactive environments, while API-Bank (Li et al., 2023) and ToolLLM/ToolBench (Qin et al., 2024) study planning, retrieval, and invocation over external APIs. Service-oriented benchmarks such as τ-bench (Yao et al., 2025) and ECom-Bench (Wang et al., 2025a) further evaluate agents in realistic multi-turn settings involving tool calls and domain-specific constraints.

Experience learning and procedural memory. Prior work has explored external memory for LLM agents through retained interaction memories, structured long-term memory, and reusable skill libraries (Park et al., 2023; Zhong et al., 2024; Xu et al., 2025; Jiang et al., 2026; Wang et al., 2024). More directly, Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and AutoGuide (Fu et al., 2024) distill feedback from prior trajectories into reusable reflections, lessons, or guidelines, while Agent Workflow Memory (Wang et al., 2025b) and MemP (Fang et al., 2026) abstract trajectories into reusable workflows or procedural memory. SUIT (Kaiser et al., 2024) similarly identifies useful subgoals from successful task-oriented dialogues for subsequent learning. IEL differs primarily in the unit at which experience receives supervision: it reconstructs cross-turn intent instances and verifies their outcomes before procedural abstraction, allowing successful and failed goals within the same dialogue to be separated.

Intent modeling and open-world experience organization. Intent modeling is widely studied in task-oriented dialogue, including multi-intent settings (Louvan & Magnini, 2020; Moghe et al., 2023). Prior work has further studied new-intent discovery and continual open-world intent recognition (Lin et al., 2020; Zhang et al., 2021; Mou et al., 2022; Song et al., 2023; Zhou et al., 2023). These methods primarily treat intents as semantic categories to recognize or discover. IEL instead uses intent instances as the supervision and organization units of reusable procedural experience, while incorporating recurring taxonomy-absent goals as the memory evolves.

3 Method

We consider a tool-using service agent interacting with a user over a multi-turn episode, where intent-instance boundaries and intent-level outcomes are unannotated. IEL treats a cross-turn realization of a business goal, together with its attributed interaction evidence, as the basic unit of experience. We use *intent-level credit assignment* to denote assigning evidence and outcomes to these instances before reusable procedures are abstracted, rather than estimating the causal contribution of individual actions to a scalar return.

3.1 Framework Overview

Figure 2 summarizes three stages of IEL: (1) reconstruct cross-turn intent instances and attribute interaction evidence, (2) verify their outcomes and organize previously unseen intents, and (3) induce reusable procedures from verified intent-specific experience. At the beginning of episode e, IEL maintains a versioned intent-level memory

$$\mathcal{M}_e = (\mathcal{V}_e, \mathcal{L}_e, \mathcal{X}_e, \mathcal{E}_e)\,, \tag{1}$$

where $\mathcal{V}_e$ is the Intent Taxonomy, $\mathcal{L}_e$ the Candidate Ledger, $\mathcal{X}_e$ the Intent Record Store, and $\mathcal{E}_e$ the Procedure Memory. The first two components organize known and emerging intent semantics, while $\mathcal{X}_e$ retains reconstructed evidence and $\mathcal{E}_e$ stores reusable procedures indexed by canonical intents. Thus, $\mathcal{V}_e$ specifies *what* goal an intent represents, whereas $\mathcal{E}_e(v)$, when available, specifies *how* to pursue it; an intent may exist without yet having a learned procedure.

Experience accumulation starts from an empty memory, i.e., $\mathcal{M}_0 = (\varnothing, \varnothing, \varnothing, \varnothing)$, whose components are populated solely from growth interaction experience. During interaction, user mentions

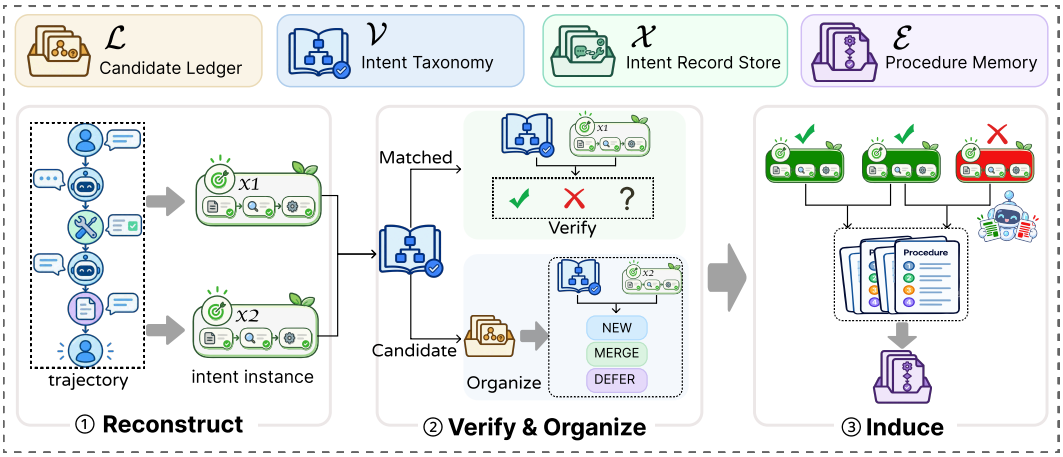

Figure 2: Overview of IEL. Cross-turn intent instances are reconstructed from a mixed-outcome dialogue and associated with their interaction evidence. The resulting records are verified and organized using intent-specific semantics, and verified experience is subsequently used to induce reusable procedures.

are matched against $\mathcal{V}_e$, and an available $\mathcal{E}_e(v)$ is retrieved as guidance for the fixed executor. Unmatched mentions and matched intents without procedures can still be handled using dialogue context and tools. Memory is read-only within an episode; post-episode updates become visible only from the next episode. Detailed online inputs and memory fields are given in Appendix B.1.

3.2 Intent Reconstruction and Evidence Attribution

After an episode ends, IEL links related intent mentions across turns using goal semantics, business objects, arguments, references, and completion cues. Clarifications and revisions remain within the same instance when they refine the same underlying goal, while ambiguous boundaries are explicitly retained. Dialogue and tool evidence are then attributed to reconstructed instances; a tool call is assigned only when its intended instance is unambiguous. Attribution determines which intent a call targets, not whether the call itself is correct, allowing failed calls to serve as negative evidence when their target is clear. Ambiguous or unassigned events are retained but cannot independently establish an intent-level outcome. Detailed rules are given in Appendix B.2.

3.3 Experience Verification and Intent Evolution

Reconstruction and evidence attribution establish what happened for each intent instance; verification determines whether the attributed evidence supports success or failure, or remains insufficient for either conclusion.

Known intents. For a record x associated with a canonical intent $v \in \mathcal{V}$, a shared verifier assesses the attributed evidence in light of the intent's goal and completion type, when known. The resulting label is

$$y(x) = \begin{cases} \text{POSITIVE}, & \text{verified completion}, \\ \text{NEGATIVE}, & \text{verified failure}, \\ \text{UNCERTAIN}, & \text{otherwise}. \end{cases} \tag{2}$$

Positive and negative labels require unambiguous attribution and sufficient outcome evidence; otherwise the record remains uncertain. Completion-type, temporal, and repair rules are detailed in Appendix B.3.

Candidate resolution. Algorithm 1 in Appendix B.4 summarizes the verification and intent-evolution procedure. Records corresponding to taxonomy-absent intent candidates remain UNCERTAIN and are organized in the Candidate Ledger. For a candidate $c \in \mathcal{L}$, let $\mathcal{X}_c \subseteq \mathcal{X}$

denote the set of intent records supporting c. Once $|\mathcal{X}_c| \geq n_{\min}$, where $n_{\min}$ is the minimum number of distinct intent instances required before semantic resolution, the resolver compares c with the current Intent Taxonomy and returns

$$\text{Resolve}(c) \in \{\text{MERGE}(v), \text{NEW}(\hat{v}_c), \text{DEFER}\}, \tag{3}$$

where $v \in \mathcal{V}$ denotes an existing canonical intent and $\hat{v}_c$ denotes a proposed new canonical intent derived from c. $\text{MERGE}(v)$ associates the candidate with an existing goal; $\text{NEW}(\hat{v}_c)$ defines a distinct goal and, when identifiable, whether completing it requires a state-changing action; and DEFER retains the candidate when its semantic evidence remains insufficient.

Resolution determines *what the candidate means*; it does not by itself establish whether its historical executions succeeded. The resolved goal and completion type, when known, guide the shared verifier when it assesses the candidate's attributed evidence. Let

$$\mathcal{X}_c^+ = \{x \in \mathcal{X}_c : y(x) = \text{POSITIVE}\}$$

denote the verified positive records of candidate c. The resolved candidate can affect subsequent memory updates only when

$$|\mathcal{X}_c| \geq n_{\min}, \qquad |\mathcal{X}_c^+| \geq s_{\min}, \tag{4}$$

where $s_{\min}$ is the minimum number of verified positive records required for an update.

We set $n_{\min} = 2$ and $s_{\min} = 1$ in all IEL runs. Additional implementation details are reported in Appendix B.6.

For NEW, passing the gate adds $\hat{v}_c$ to the Intent Taxonomy; for MERGE, the verified records are associated with the existing intent v. In both cases, the resulting verified positive and negative records become available for subsequent procedure induction. Candidates that are deferred or fail the evidence gate remain in the Candidate Ledger for future reconsideration. Further resolution and update rules are provided in Appendix B.4.

3.4 Evidence-Contrastive Procedure Induction

For each canonical intent v, IEL selects historical records with unambiguous attribution and verified positive or negative outcomes, and passes their user text, slots, outcomes, and tool calls to the procedure inducer. Successful records provide observed execution paths, while failed records, when available, inform warnings about errors. When both are available, their contrast helps distinguish reusable execution structure from failure patterns:

$$\mathcal{X}_v^+, \mathcal{X}_v^- \longrightarrow \mathcal{E}(v). \tag{5}$$

One longest common subsequence of tool names across successful records is used as a structural hint, while nonshared steps are represented as conditional branches. No procedure is generated without a successful record. The record selection and write rule are detailed in Appendix B.4, and the induction prompt in Appendix C.4.

4 Experiments

We evaluate IEL on in-domain held-out tasks and cross-category transfer, followed by ablations and analyses of the learned experience.

4.1 Experimental Setup

Benchmarks. For in-domain evaluation, we use the 53-task ECom-Bench (Wang et al., 2025a), which focuses on customer-service scenarios involving home-appliance products. Each task consists of an interactive customer-service episode with an isolated database and a simulated user. To evaluate cross-category transfer, we construct ECom-Bench-XCat, a 60-task extension of ECom-Bench comprising 30 clothing tasks and 30 food tasks. ECom-Bench-XCat follows the same customer-service interaction and tool-use setting as ECom-Bench while introducing previously unseen product categories. Appendix A details its task composition, simulator protocol, environment state, and evaluation annotations.

Baselines. We compare IEL against four representative baselines. ReAct interleaves reasoning and tool use without maintaining an external experience library (Yao et al., 2023). Trajectory-RAG directly reuses successful interaction history through retrieval. Specifically, we retain episodes with successful outcomes, segment each episode into chunks consisting of one user turn and all subsequent assistant and tool interactions before the next user turn, and retrieve relevant chunks by embedding similarity at inference time. ExpeL distills reusable natural-language lessons from prior trajectories (Zhao et al., 2024), while AWM induces reusable workflows from agent experience and retrieves them for subsequent tasks (Wang et al., 2025b). For Trajectory-RAG, ExpeL, AWM, and IEL, we use the same growth tasks as the experience source and freeze the resulting experience before held-out evaluation.

Evaluation Metrics. Following τ-bench (Yao et al., 2025) and its adoption in ECom-Bench (Wang et al., 2025a), we report pass^k to measure task success consistency across repeated executions. Under the ECom-Bench evaluator, an episode is counted as successful only when the required benchmark checks for Action, Search, and Output are satisfied. For task i, let $c_i \in \{0, 1, 2, 3\}$ denote the number of successful executions among three runs. Over N tasks, we compute

$$\text{pass}\hat{}k = \frac{1}{N} \sum_{i=1}^{N} \frac{\binom{c_i}{k}}{\binom{3}{k}}, \qquad k \in \{1, 2, 3\}.$$

Thus, pass^1 is the average per-run success rate across tasks, whereas pass^3 is the fraction of tasks for which all three executions succeed. All results are reported as percentages, and higher values indicate better performance.

Implementation Details. Growth episodes are processed sequentially to construct the experience memory. Before evaluation on the held-out tasks, all four memory components are frozen, such that evaluation performs only intent matching and procedure retrieval without further memory updates. We instantiate all methods with three executor backbones: DS-V4.1-FLASH (DeepSeek, 2026), DS-V3.2 (DeepSeek-AI et al., 2025), and GLM-5.3-FLASH (Z.ai, 2026), while keeping the user-simulator model fixed across all comparisons. Unless otherwise specified, IEL uses DS-V4.1-FLASH with thinking disabled for all intent-system LLM calls. The sampling temperature is set to 0.3, and each episode is limited to 20 dialogue turns. For every method–backbone configuration, each test task is executed three times to compute the pass^k metrics. Prompt templates and input-output formats are provided in Appendix C. The three-fold split and task shuffling use random seed 10. The complete growth and evaluation protocol and the intent-system hyperparameters are provided in Appendices B.5 and B.6, respectively.

4.2 IN-DOMAIN GENERALIZATION

We first investigate whether experience acquired from tasks within the same service domain generalizes to unseen tasks. We conduct three-fold cross-validation on ECOM-BENCH. In each round, the experience-based methods construct their experience libraries using two folds and evaluate the frozen libraries on the remaining fold. Each cross-validation round starts from an empty IEL memory. Thus, every task serves as a held-out test task exactly once, and no test trajectory is exposed during experience construction. We report the mean and standard deviation of the fold-level pass^k scores across the three rounds.

Results. Table 1 shows that IEL obtains the highest mean score for all nine executor–metric combinations. Averaged across executor backbones, IEL reaches 66.52, 55.21, and 48.47 on pass^1, pass^2, and pass^3, respectively. Relative to the strongest baseline within each executor–metric configuration, the average improvements are 4.22 percentage points in pass^1 and 5.70 points in pass^3. The gains are largest for DS-V4.1-FLASH and smaller for the other executor backbones, indicating that the magnitude of the benefit varies across models. Overall, the results provide evidence that intent-organized experience can improve held-out performance within the evaluated service domain.

4.3 CROSS-CATEGORY GENERALIZATION

We evaluate transfer from ECOM-BENCH to ECOM-BENCH-XCAT under a shared operational interface: the two benchmarks retain the same platform, dialogue protocol, tool interface, and eval-

Table 1: In-domain generalization on ECOM-BENCH. Results are mean ± standard deviation across three held-out folds. All values are percentages. Bold and underlined values denote the best and second-best results, respectively, among methods using the same executor LLM.

Executor LLM	Method	pass^1↑	pass^2↑	pass^3↑
DS-V4.1-FLASH	ReAct	<u>59.30 ± 14.50</u>	<u>46.70 ± 16.43</u>	<u>39.76 ± 15.26</u>
	Trajectory-RAG	37.91 ± 9.68	20.26 ± 7.25	11.44 ± 6.05
	ExpeL	54.25 ± 8.24	39.76 ± 6.39	30.28 ± 4.34
	AWM	34.75 ± 7.32	21.53 ± 6.88	15.25 ± 7.17
	IEL (ours)	**66.88 ± 14.95**	**58.03 ± 16.60**	**52.94 ± 17.43**
DS-V3.2	ReAct	44.70 ± 10.39	30.24 ± 5.35	20.81 ± 3.64
	Trajectory-RAG	56.54 ± 3.25	40.85 ± 1.69	30.17 ± 2.85
	ExpeL	<u>61.54 ± 4.16</u>	<u>46.48 ± 2.89</u>	<u>37.69 ± 2.08</u>
	AWM	54.03 ± 3.23	41.50 ± 0.97	33.98 ± 1.13
	IEL (ours)	**62.85 ± 2.02**	**48.44 ± 3.74**	**39.65 ± 5.71**
GLM-5.3-FLASH	ReAct	60.31 ± 3.13	43.25 ± 6.95	31.81 ± 12.26
	Trajectory-RAG	65.98 ± 3.94	54.07 ± 2.95	47.17 ± 2.78
	ExpeL	<u>66.05 ± 2.83</u>	<u>55.95 ± 2.93</u>	<u>50.87 ± 4.32</u>
	AWM	60.97 ± 4.99	47.17 ± 8.34	39.65 ± 11.19
	IEL (ours)	**69.83 ± 2.85**	**59.15 ± 3.11**	**52.83 ± 2.78**

uation structure, while differing in product categories, entities, and category-specific constraints. For each experience-based method, the experience library is constructed only from ECOM-BENCH trajectories generated by DS-V4.1-FLASH, frozen, and reused across all executor backbones; no ECOM-BENCH-XCAT trajectory or feedback is used for memory construction. ReAct is evaluated without an external experience library.

Table 2: Cross-category generalization from ECOM-BENCH to ECOM-BENCH-XCAT. Experience is constructed only from source-category trajectories and remains frozen throughout target-category evaluation. All values are percentages. Bold and underlined values denote the best and second-best results, respectively, among methods using the same executor LLM.

Executor LLM	Method	pass^1↑	pass^2↑	pass^3↑
DS-V4.1-FLASH	ReAct	<u>48.89</u>	<u>42.22</u>	<u>38.33</u>
	Trajectory-RAG	33.33	27.22	21.67
	ExpeL	46.67	41.67	<u>38.33</u>
	AWM	35.00	28.33	25.00
	IEL (ours)	**53.89**	**46.67**	**41.67**
DS-V3.2	ReAct	38.89	35.00	33.33
	Trajectory-RAG	<u>42.78</u>	<u>38.89</u>	<u>36.67</u>
	ExpeL	39.44	34.44	31.67
	AWM	37.22	32.22	30.00
	IEL (ours)	**46.11**	**41.11**	**38.33**
GLM-5.3-FLASH	ReAct	49.44	40.56	36.67
	Trajectory-RAG	<u>53.33</u>	<u>47.22</u>	<u>45.00</u>
	ExpeL	51.11	43.33	40.00
	AWM	50.00	43.89	40.00
	IEL (ours)	**58.89**	**52.22**	**48.33**

Results. As shown in Table 2, IEL obtains the highest mean score across all nine executor–metric combinations. Averaged across the three executor backbones, it obtains pass^1, pass^2, and pass^3 scores of 52.96, 46.67, and 42.78, respectively. Compared with the strongest baseline for each executor, the average improvements are 4.63, 3.89, and 2.78 percentage points. The same source-trained library remains useful with DS-V3.2 and GLM-5.3-FLASH, providing evidence of cross-category and cross-executor reuse within the shared operational interface.

4.4 Ablation Study

We next examine the contributions of the main design choices in IEL. All ablations are conducted on ECom-Bench with DS-V4.1-Flash under the same three-fold held-out protocol as Sec. 4.2. We report the mean and standard deviation of fold-level passˆk scores across the three folds.

We compare the full IEL against four variants. *w/o Intent Decomposition* constructs reusable experience directly at the episode level rather than reconstructing cross-turn intent instances, testing the contribution of intent-level experience granularity. *w/o Procedure Induction* directly reuses the corresponding interaction trajectories instead of inducing reusable procedures from verified intent-level experience. *w/o Procedure Refinement* disables post-episode refinement, preventing previously constructed experience from being updated with subsequent interactions. Finally, *Embedding Retrieval* replaces intent-based experience access with embedding-similarity retrieval over the same experience library, testing the contribution of explicit intent-level indexing.

Table 3: Ablation study of IEL on ECom-Bench using DS-V4.1-Flash. Results are mean ± standard deviation across three held-out folds. All values are percentages. Bold and underlined values denote the best and second-best results, respectively.

Method	passˆ1↑	passˆ2↑	passˆ3↑
IEL (Full)	$\mathbf{66.88 \pm 14.95}$	$\mathbf{58.03 \pm 16.60}$	$\mathbf{52.94 \pm 17.43}$
w/o Intent Decomposition	50.22 ± 9.44	38.85 ± 11.30	31.92 ± 11.05
w/o Procedure Induction	57.30 ± 6.32	49.75 ± 6.21	$\underline{45.32 \pm 5.76}$
w/o Procedure Refinement	$\underline{61.80 \pm 13.47}$	$\underline{51.05 \pm 13.90}$	43.47 ± 14.23
Embedding Retrieval	61.08 ± 8.05	50.29 ± 8.41	45.21 ± 9.04

Results. As shown in Table 3, all four variants achieve lower mean scores than the full IEL across the three metrics. The largest performance degradation occurs when intent decomposition is removed: passˆ3 drops from 52.94 to 31.92, a decrease of 21.02 percentage points. The larger drop in passˆ3 than in passˆ1 suggests that intent-level organization is particularly important for consistent success across repeated executions. Replacing induced procedures with raw interaction trajectories reduces passˆ3 by 7.62 points, while disabling procedure refinement and replacing intent-based access with embedding retrieval lead to decreases of 9.47 and 7.73 points, respectively.

4.5 Analysis

We further analyze two questions: whether IEL's performance gains can be explained simply by retrieving more experience context, and how the learned intent-level procedures support reuse across product categories.

Retrieved-Experience Context Efficiency. We next examine whether IEL's performance advantage can be explained simply by providing the executor with a larger retrieved experience context. Figure 3 compares mean retrieved experience tokens per task with mean passˆ3 under the DS-V4.1-Flash in-domain setting. ReAct provides the zero-experience reference point because it does not use an external experience library. Among experience-based methods, IEL uses fewer retrieved tokens than ExpeL and AWM while achieving a higher mean passˆ3. Trajectory-RAG uses fewer tokens than IEL but attains a lower passˆ3. These comparisons indicate that the observed performance gains are not simply associated with providing a larger retrieved experience context, and suggest that how experience is organized and abstracted also matters.

Case Study: Cross-Category Procedure Reuse. Figure 4 presents a case study of intent-level procedure reuse from a home-appliance interaction to a food-category task. The source and target interactions instantiate the same canonical intent, `order.expedite`, and use the same tool interface, while differing in product category, entity values, and surface realization. From the source interaction, IEL abstracts a procedure that retains the operational structure needed to handle the intent: locating the target order, checking whether it can be expedited, submitting the request, and verifying the resulting completion evidence. At the same time, source-specific details such as the original order identifier and product information are omitted from the stored procedure. When applied to the

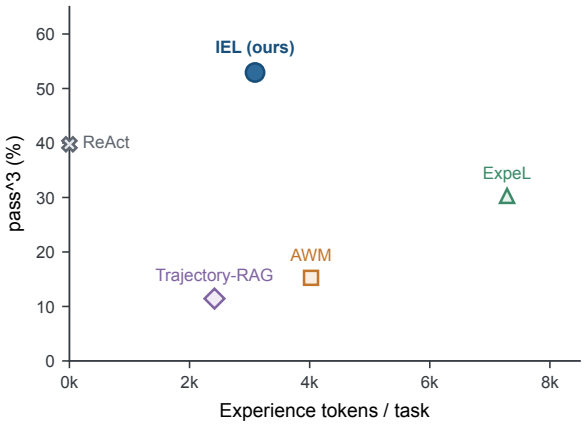

Figure 3: Performance–context trade-off under the DS-V4.1-FLASH in-domain setting. Each marker shows mean experience tokens per task and mean pass^3 across three held-out folds. ReAct is shown at zero experience tokens because it does not use an external experience library.

target interaction, the procedure is reused with a different product and order while preserving the same intent-level handling structure. This case illustrates how intent-level procedural abstraction can retain reusable operational structure while discarding instance-specific details under a shared service and tool interface.

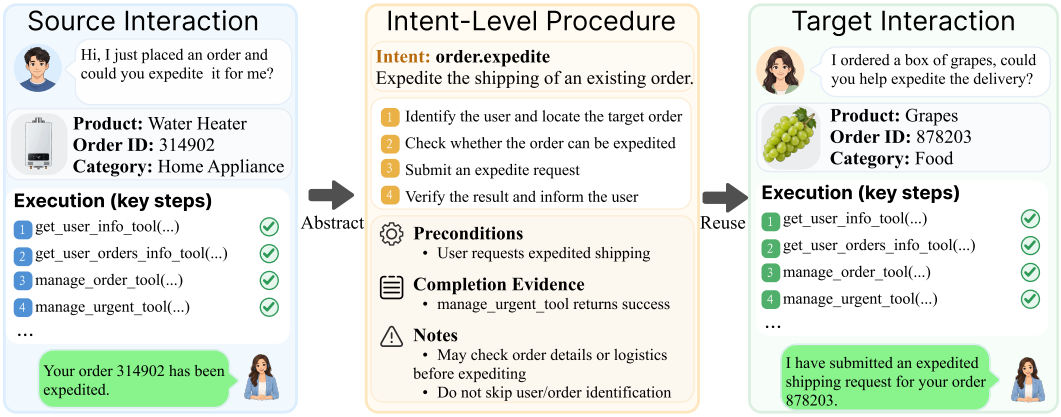

Figure 4: Case study of cross-category intent-level procedure reuse. The source and target interactions instantiate the same canonical intent, `order.expedite`, and share the same tool interface, while differing in product category, entity values, and surface realization. The induced procedure retains reusable operational structure and completion checks while omitting source-specific details.

5 CONCLUSION

We studied experience learning for multi-intent service agents under mixed-outcome interactions, where episode-level feedback provides ambiguous supervision for intent-specific procedural reuse. We introduced IEL, which reconstructs cross-turn intent instances, attributes interaction evidence and outcomes at the intent level, and induces reusable procedures only after this intent-level credit assignment. Across ECOM-BENCH and the ECOM-BENCH-XCAT cross-category setting, IEL obtains the highest mean score in all evaluated executor–metric combinations. Ablation results further show that removing intent decomposition and the resulting intent-specific organization substantially reduces repeated-execution success. Overall, these findings support experience granularity as an important design dimension for agents that learn reusable procedures from mixed-outcome interaction histories.

REPRODUCIBILITY STATEMENT

Appendix A describes the construction and evaluation setting of ECom-Bench-XCat. Appendices B and C document the memory-growth and held-out evaluation procedures, intent reconstruction and verification rules, implementation constants, and prompt templates. Section 4.1 reports the random seed, executor backbones, and repeated-execution protocol used in the experiments. An anonymized implementation of the proposed method is available at this anonymous repository.

AI USE STATEMENT

Generative AI tools were used to assist with drafting and editing manuscript text, consistency and reference checks, figure preparation, and LaTeX diagnostics. All AI-assisted suggestions were reviewed by the authors, who take responsibility for the final content of the manuscript.

REFERENCES

DeepSeek. DeepSeek-V4.1-Flash release. DeepSeek API Change Log, 2026. URL https://api-docs.deepseek.com/updates/. Release entry dated September 10, 2026. Accessed September 25, 2026.

DeepSeek-AI et al. DeepSeek-V3.2: Pushing the frontier of open large language models. arXiv preprint arXiv:2512.02556, 2025. URL https://arxiv.org/abs/2512.02556.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. MemP: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026. doi: 10.18653/v1/2026.findings-acl.866. URL https://aclanthology.org/2026.findings-acl.866/.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 119919–119948, 2024. doi: 10.52202/079017-3811. URL https://papers.nips.cc/paper_files/paper/2024/hash/d8efbb5dd415974eb095c3f06bff1f48-Abstract-Conference.html.

Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. MAGMA: A multi-graph based agentic memory architecture for AI agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36848–36865. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1709. URL https://aclanthology.org/2026.acl-long.1709/.

Magdalena Kaiser, Patrick Ernst, and György Szarvas. Learning from relevant subgoals in successful dialogs using iterative training for task-oriented dialog systems. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6236–6246. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.362. URL https://aclanthology.org/2024.findings-emnlp.362/.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Ting-En Lin, Hua Xu, and Hanlei Zhang. Discovering new intents via constrained deep adaptive clustering with cluster refinement. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(5):8360–8367, 2020. doi: 10.1609/aaai.v34i05.6353. URL https://ojs.aaai.org/index.php/AAAI/article/view/6353.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie

Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, pp. 52989–53046, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html.

Samuel Louvan and Bernardo Magnini. Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 480–496. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.42. URL https://aclanthology.org/2020.coling-main.42/.

Nikita Moghe, Evgeniia Razumovskaia, Liane Guillou, Ivan Vulić, Anna Korhonen, and Alexandra Birch. Multi3NLU++: A multilingual, multi-intent, multi-domain dataset for natural language understanding in task-oriented dialogue. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 3732–3755. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-acl.230. URL https://aclanthology.org/2023.findings-acl.230/.

Yutao Mou, Keqing He, Yanan Wu, Pei Wang, Jingang Wang, Wei Wu, Yi Huang, Junlan Feng, and Weiran Xu. Generalized intent discovery: Learning from open world dialogue system. In *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 707–720. International Committee on Computational Linguistics, 2022. URL https://aclanthology.org/2022.coling-1.59/.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22. Association for Computing Machinery, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763. Article 2.

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. In *The Twelfth International Conference on Learning Representations*, pp. 9695–9717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://papers.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Xiaoshuai Song, Yutao Mou, Keqing He, Yueyan Qiu, Jinxu Zhao, Pei Wang, and Weiran Xu. Continual generalized intent discovery: Marching towards dynamic and open-world intent recognition. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 4370–4382. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.289. URL https://aclanthology.org/2023.findings-emnlp.289/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Haoxin Wang, Xianhan Peng, Xucheng Huang, Yizhe Huang, Ming Gong, Chenghan Yang, Yang Liu, and Ling Jiang. ECom-Bench: Can LLM agent resolve real-world E-commerce customer support issues? In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 276–284, Suzhou (China), November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-333-3. doi: 10.18653/v1/2025.emnlp-industry.19. URL https://aclanthology.org/2025.emnlp-industry.19/.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 17577–17604, 2025. doi: 10.52202/085713-0593. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Z.ai. GLM-5.3-Flash. Model card, 2026. URL https://huggingface.co/zai-org/GLM-5.3-Flash. Technical report: GLM-5: from Vibe Coding to Agentic Engineering, arXiv:2602.15763. Accessed September 24, 2026.

Hanlei Zhang, Hua Xu, Ting-En Lin, and Rui Lyu. Discovering new intents with deep aligned clustering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14365–14373, 2021. doi: 10.1609/aaai.v35i16.17689. URL https://ojs.aaai.org/index.php/AAAI/article/view/17689.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946. URL https://ojs.aaai.org/index.php/AAAI/article/view/29946.

Yunhua Zhou, Guofeng Quan, and Xipeng Qiu. A probabilistic framework for discovering new intents. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3771–3784. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.209. URL https://aclanthology.org/2023.acl-long.209/.

A ECOM-BENCH-XCAT DATASET

Scope and transfer setting. We construct ECom-Bench-XCat (XCat) to evaluate whether experience acquired from the home-appliance tasks in ECom-Bench transfers to previously unseen clothing and food categories. XCat contains 60 interactive customer-service tasks, split evenly between the two target categories. It preserves the platform, dialogue protocol, tool interface, and evaluation structure of ECom-Bench while changing the product entities and category-specific constraints. The benchmark therefore measures cross-category transfer within a shared operational setting. No XCat trajectory or feedback is used to construct the experience libraries evaluated on XCat.

Task and user-simulator design. Each task specifies a simulated customer identity, a behavioral profile, and an ordered set of two to six private goals. These goals are provided to the user simulator but are not exposed directly to the customer-service agent. The simulator is instructed to remain in character, reveal goals incrementally rather than in a single turn, provide task-specified identifiers

when needed, avoid inventing unavailable information, and emit a stop marker after all goals have been addressed. The current task set contains 252 goal specifications, with 126 in each category. Within each category, 4 tasks contain two goals, 2 contain three goals, 16 contain four goals, and 8 contain six goals. The tasks instantiate four customer profiles: rational comparison (8 tasks), time-sensitive (22), evidence- and compliance-oriented (10), and after-sales rights-oriented (20).

Environment state and interaction scope. XCat provides an offline store state with 36 products (18 clothing and 18 food), 20 customer records, 60 orders, and 60 corresponding logistics records. Additional state covers invoices, discounts, category-specific policies, after-sales cases, and specialist service tickets. Clothing scenarios include size, color, material, tag, return, and exchange constraints. Food scenarios include shelf life, cold-chain delivery, leakage, damage, allergens, discounts, invoicing, logistics, and after-sales handling.

B Additional Method Details

B.1 Interaction Records and Online Decomposition

Trajectory and episode-level evaluation. Let episode e contain T_e user turns. Its interaction trajectory is

$$\tau_e = \langle (u_{e,t}, \xi_{e,t}) \rangle_{t=1}^{T_e}, \tag{6}$$

where $u_{e,t}$ is the user message and $\xi_{e,t}$ is the ordered interaction segment following that message until the next user turn. Each $\xi_{e,t}$ contains the assistant's tool calls and their arguments, the corresponding tool observations, and assistant replies. Thus, τ_e preserves the dialogue and tool-use evidence required for post-episode intent reconstruction and outcome verification.

The episode-level benchmark outcome is

$$r_e = \mathrm{Eval}_e(\tau_e) \in \{0, 1\}. \tag{7}$$

This outcome is reserved for evaluation and is not used as supervision for memory updates. Intent-instance boundaries and intent-level outcomes are not annotated.

Intent representations and record fields. An *intent mention* is a turn-level semantic unit extracted during online interaction. An *intent instance* links mentions associated with a specific cross-turn business goal. A *candidate intent* groups recurring taxonomy-absent instances with similar semantics. A *taxonomy intent* is a canonical intent definition used for matching and verification. The memory snapshot at episode e is

$$\mathcal{M}_e = (\mathcal{V}_e, \mathcal{L}_e, \mathcal{X}_e, \mathcal{E}_e). \tag{8}$$

Each record $x \in \mathcal{X}_e$ retains the supporting user spans, instance-specific arguments and business objects, attributed tool events, outcome evidence, the current learning label, ambiguity information, and provenance references to the original interaction events. The Candidate Ledger stores candidate semantics together with references to supporting records rather than replacing their original evidence. Each $v \in \mathcal{V}_e$ specifies a canonical goal and, when identifiable, its completion type. $\mathcal{E}_e(v)$ stores the corresponding procedure when one is available.

Online decomposition. Given the current user message $u_{e,t}$, short dialogue history $h_{e,t}^{\mathrm{short}}$, current taxonomy $\mathcal{V}_e$, and available tool signatures $\mathcal{A}$, the decomposer returns

$$D(u_{e,t}, h_{e,t}^{\mathrm{short}}, \mathcal{V}_e, \mathcal{A}) = \left(I_{e,t}^{\mathrm{match}}, I_{e,t}^{\mathrm{cand}}\right). \tag{9}$$

A mention belongs to $I_{e,t}^{\mathrm{match}}$ only when it can be confidently matched to an existing taxonomy definition. It belongs to $I_{e,t}^{\mathrm{cand}}$ only when it expresses a concrete goal judged absent from the current taxonomy. Ambiguous matches to existing intents are omitted from both sets. Each mention retains its supporting user span, available arguments, and business-object identifiers. For a matched intent v, the router retrieves $\mathcal{E}_e(v)$ when available. The executor can still act using dialogue context and tools when no procedure exists or when a mention remains unmatched. Both the interaction trajectory and online decompositions are retained for post-episode reconstruction.

B.2 Intent-Instance Boundaries and Tool Attribution

Lifecycle reconstruction. After the episode ends, IEL links intent mentions using goal semantics, business objects, arguments, references, and completion cues. Clarifications and parameter revisions update the same active instance when they continue the same underlying business goal. The record preserves successive specifications so that actions can later be evaluated against the specification applicable at the time they occurred.

A change in the primary goal or business object starts a new instance. A renewed request after a previous instance has completed or been withdrawn also starts a new instance. Withdrawal closes an instance without deleting its evidence. When the boundary between two possible instances cannot be determined reliably, the ambiguity is recorded rather than resolved optimistically. Online decomposition identifies turn-level intent mentions for routing, whereas post-episode reconstruction links these mentions into cross-turn intent instances for experience learning.

Dependency-based tool attribution. Consider a tool call g. Let C_g contain g together with its predecessor calls linked through output-to-input dependencies. The set of compatible intent instances is

$$J_g = \{i : \mathrm{Open}(i, g) \wedge \mathrm{Compatible}(i, C_g)\}, \tag{10}$$

where $\mathrm{Open}(i, g)$ checks whether instance i is active immediately before g, and $\mathrm{Compatible}(i, C_g)$ uses dialogue context, business-object references, and argument provenance to determine which instance the call attempts to address.

Compatibility concerns the *intended goal* of the call rather than its correctness. A call with incorrect arguments or an incorrect target object may therefore still be attributed to an instance if the intended goal is unambiguous. This separation allows execution mistakes to provide negative evidence without confusing attribution with outcome judgment.

A call is exclusively attributed only when

$$|J_g| = 1.$$

If multiple instances are compatible, the call remains ambiguous. If none are compatible, it remains unassigned. Both ambiguous and unassigned calls are retained in the underlying trajectory, but they cannot independently establish an intent-level outcome. Dependency predecessors that provide shared context may support multiple instances without being exclusively assigned.

Candidate-intent attribution can be revisited after semantic resolution identifies a canonical goal and, when available, its completion type.

B.3 Completion Types and Outcome Verification

Completion types. When identifiable, a canonical intent $v \in \mathcal{V}$ has a completion type

$$q_v \in \{\mathrm{INFORMATION}, \mathrm{ACTION}\}.$$

For an INFORMATION intent, the verifier assesses whether the available observations and response address the requested information.

For an ACTION intent, a successful state-changing tool call provides evidence of completion. A handoff is treated separately; ambiguous or unsupported outcomes remain uncertain.

Temporal scope and revisions. Actions are evaluated against the intent specification that was active when they occurred. When a user revises arguments or constraints, completion is determined using the latest applicable specification.

Verification considers evidence up to instance closure or, for an instance still active at the end of the dialogue, up to episode termination. An unsuccessful intermediate call followed by a successful retry does not prevent a positive label when the later evidence supports completion. Similarly, a temporary mismatch that is later repaired does not by itself establish failure.

A negative label can arise from failed tool actions, a missing required state change, or an unresolved negative signal. Insufficient evidence yields an uncertain label. Withdrawal or supersession does not by itself imply failure.

Uncertainty and provenance. Both positive and negative labels require unambiguous intent attribution and sufficient outcome evidence. Otherwise, the record receives the label UNCERTAIN.

Each evidence item retains references to the underlying trajectory events from which it was derived. This provenance allows verification decisions to be traced back to dialogue turns, tool calls, arguments, observations, and assistant replies.

Semantic resolution identifies the canonical goal under which candidate evidence is interpreted. Available completion-type information then informs outcome assessment by the shared verifier.

B.4 Candidate Decisions and Procedure Updates

Algorithm 1 Experience Verification and Intent Evolution

Require: Newly reconstructed intent records $\Delta\mathcal{X}_e$, memory $(\mathcal{V}_e, \mathcal{L}_e, \mathcal{X}_e)$, thresholds $n_{\min}, s_{\min}$
Ensure: Updated $(\mathcal{V}_{e+1}, \mathcal{L}_{e+1}, \mathcal{X}_{e+1})$

$\mathcal{V} \leftarrow \mathcal{V}_e,\ \mathcal{L} \leftarrow \mathcal{L}_e,\ \mathcal{X} \leftarrow \mathcal{X}_e \cup \Delta\mathcal{X}_e$
for all $x \in \Delta\mathcal{X}_e$ **do**
  $r \leftarrow \textsc{RouteIntent}(x, \mathcal{V})$ ▷ $r \in \{\text{MATCH}(v), \text{CANDIDATE}, \text{AMBIGUOUS}\}$
  if $r = \text{MATCH}(v)$ **then**
    $y(x) \leftarrow \textsc{Verify}(x, v)$ ▷ shared evidence rules, informed by the type of v
    $\textsc{Associate}(x, v)$
  else if $r = \text{CANDIDATE}$ **then**
    $y(x) \leftarrow \text{UNCERTAIN}$
    $c \leftarrow \textsc{GroupCandidate}(x, \mathcal{L})$ ▷ equivalent candidate or new entry
    add x to c
  else ▷ $r = \text{AMBIGUOUS}$
    $y(x) \leftarrow \text{UNCERTAIN}$
  end if
end for
for all $c \in \mathcal{L}$ with $|\mathcal{X}_c| \geq n_{\min}$ **do**
  $d \leftarrow \textsc{Resolve}(c, \mathcal{V})$
  if $d \neq \text{DEFER}$ **then**
    $y(x) \leftarrow \textsc{Verify}(x, d)$ for all $x \in \mathcal{X}_c$
    $\mathcal{X}_c^+ \leftarrow \{x \in \mathcal{X}_c : y(x) = \text{POSITIVE}\}$
    if $|\mathcal{X}_c^+| \geq s_{\min}$ **then**
      $v^\star \leftarrow \textsc{ApplyResolution}(d, \mathcal{V})$ ▷ add NEW; return MERGE match
      associate all verified positive/negative records in $\mathcal{X}_c$ with $v^\star$
      mark c as resolved
    end if
  end if
end for
return $\mathcal{V}, \mathcal{L}, \mathcal{X}$

Candidate semantics. A candidate $c \in \mathcal{L}$ is considered for semantic resolution after it contains at least $n_{\min}$ distinct intent-instance records.

The resolver compares the candidate with the current Intent Taxonomy:

- $\text{MERGE}(v)$ associates the candidate with an existing canonical goal and its available completion type;
- $\text{NEW}(\hat{v}_c)$ proposes a distinct canonical goal and, when identifiable, whether completion requires a state-changing action;
- DEFER retains the candidate when the available semantic evidence remains insufficient.

A non-deferred decision identifies the goal under which candidate evidence is assessed. The shared verifier uses this goal and its available completion type to assign outcome labels. The evidence gate for a memory update is

$$|\mathcal{X}_c| \geq n_{\min}, \qquad |\mathcal{X}_c^+| \geq s_{\min}. \tag{11}$$

For NEW, passing the gate adds $\hat{v}_c$ to the Intent Taxonomy. For MERGE, the verified candidate records are associated with the existing canonical intent. Deferred candidates and candidates that fail the evidence gate remain in the Candidate Ledger for future reconsideration.

Record selection and presentation. For procedure induction, IEL excludes records with ambiguous attribution or uncertain outcomes. Each remaining record is supplied with its outcome label and reason, user text, slots, and tool calls, including each call's name, action, and status.

Evidence-contrastive procedure induction. The program computes one longest common subsequence of tool names across all successful records for the canonical intent and supplies it as a structural hint alongside the full records. The prompt asks the inducer to use this subsequence as a backbone and express differences as optional branches with trigger conditions. When successful paths differ substantially, the subsequence can be short or empty; the full records remain available to the inducer. Failed records are supplied to help the model describe divergences associated with errors.

The induced procedure $\hat{\mathcal{E}}(v)$ may contain:

- supported execution steps;
- preconditions and branch conditions;
- completion evidence;
- failure warnings.

The prompt restricts the common-errors section to cases where both successful and failed records are available.

Procedure write rule. The inducer is called only when at least one successful record remains. A nonempty text response is written as the procedure for the canonical intent, replacing any previous procedure; otherwise the previous procedure is retained. References to the selected records are stored with the procedure.

The update is

$$\mathcal{E}_{e+1}(v) = \begin{cases} \hat{\mathcal{E}}(v), & \text{if the inducer returns nonempty text,} \\ \mathcal{E}_e(v), & \text{otherwise.} \end{cases} \tag{12}$$

An empty or invalid response therefore leaves the previous procedure unchanged.

B.5 Memory Growth and Held-Out Evaluation

Growth episodes are processed sequentially. Episode e reads only the memory snapshot $\mathcal{M}_e$ available at its start. During the episode, the memory is read-only and the executor parameters remain fixed.

After the episode ends, intent reconstruction and evidence attribution may add records to the Intent Record Store. Verified records may refine existing procedures. Candidate records may update the Candidate Ledger, and sufficiently supported resolved candidates may modify the Intent Taxonomy. All resulting changes become visible only through the next memory snapshot $\mathcal{M}_{e+1}$.

Before held-out evaluation, all four memory components are frozen. Evaluation uses the same online intent decomposition and procedure retrieval mechanism as during growth, but it performs no:

- intent-record accumulation,
- candidate-semantic update or resolution,
- taxonomy modification, or
- procedure induction or replacement.

The executor parameters also remain fixed. Consequently, held-out performance measures reuse of experience learned from the designated growth episodes rather than test-time adaptation.

B.6 Implementation Hyperparameters

Table 4 reports the fixed intent-system constants.

Implementation quantity	Value	Role
Candidate records, $n_{\min}$	2	Minimum number of distinct records before adjudication.
Verified positives, $s_{\min}$	1	Minimum positive evidence required for a resolved candidate to update memory.
Taxonomy neighbors	5	Maximum number of active intents considered during candidate resolution.
Recent dialogue context	3	Maximum number of preceding messages supplied to online intent decomposition.
Terminal-action inference	2	Require successful state-changing calls in two unambiguous records before marking an intent as action-requiring.

Table 4: Fixed intent-system constants for growth and experience retrieval.

C Prompt Templates

This section documents the main prompt templates and input-output formats used by IEL. Each intent-side call is sent as a single user message. Runtime values represented by brace-delimited placeholders are serialized by the program and are not generated by the template.

C.1 Online Intent Decomposition

The decomposer jointly segments the current user message and matches each segment against the current taxonomy. The taxonomy view contains each active intent's identifier, description, discriminators, negative triggers, and slot schema. The complete instruction and few-shot portion is:

```
You are processing an e-commerce customer-service dialogue between a buyer
and a service agent. A business intent is a concrete request that the buyer
wants the agent to help complete, such as returning an item or changing a
delivery address.

Decompose the user message into atomic business intents.
1. One intent is one independently closable business goal: it could form a
   separate ticket and has its own terminal criterion.
2. Use an object.action intent id. Reuse the same wording for the same meaning.
3. A slot is a parameter difference within one intent. If the difference
   changes which tools are called or their order, it may be a new intent. If
   it changes only parameters, call count, or a trigger condition, use a slot.
4. If an item cannot be named without ambiguity, emit no intent for it and
   place it under non_transactional.
5. Reuse keys and values in the matched intent's existing slot schema whenever
   possible; add a new slot only when necessary.
6. Put an item in matched only when the message contains a supporting span,
   the item satisfies the intent's discriminators, and it triggers none of the
   intent's negative_triggers. Do not infer a match from vague relatedness.
7. matched may be empty or contain multiple items. Do not force an uncertain
   existing-intent match, and do not move such uncertainty to candidates.
   candidates is only for a genuinely new intent absent from the taxonomy.
   When uncertain, omit the item rather than guess.
8. Every matched item is used directly to retrieve and inject experience.

Non-exhaustive examples:
A. "Please change my delivery address, and I also want to schedule
   installation for order 1234567890." Emit order.change_address and
   service.schedule_install because they have independently testable outcomes.
B. "Your delivery is far too slow!" is emotion without an independently
   closable goal; emit no intent and mark it non_transactional.
C. "This seems expensive; I should have waited for a sale." does not directly
   request a coupon lookup or price protection. Do not force a match or create
   a candidate merely to avoid an empty output.

Output JSON only:
{
```

```
  "matched": [
    {"id": "existing taxonomy id", "slots": {"key": "value"},
     "span": "verbatim supporting user span"}
  ],
  "candidates": [
    {"proposed_id": "object.action", "desc": "one-sentence description",
     "slots": {"key": "value"}, "span": "supporting user span"}
  ],
  "non_transactional": ["description of an omitted span"]
}
matched may contain only existing taxonomy ids that are certain matches.
Put taxonomy-absent intents in candidates. Put uncertain existing-intent
matches in neither list.

Current intent taxonomy:
<taxonomy>{taxonomy_json}</taxonomy>
Available tools:
<tools>{tool_names}</tools>
Recent dialogue context:
<context>{up_to_three_recent_non_system_messages}</context>
Current user message:
<message>{user_message}</message>
```

C.2 SHARED OUTCOME VERIFICATION

The following template specifies the shared verifier described in Section 3.3. Records for unresolved candidates remain uncertain until semantic resolution supplies a canonical goal.

```
Review the outcome of each reconstructed intent instance in this e-commerce customer-service
     episode. Judge each instance separately under its canonical goal and completion type.
     Use only evidence attributed to that instance; shared contextual observations may be
     used only when their provenance and relevance to the instance are explicit. Do not use
     benchmark rewards or private task annotations.

For each instance:
1. Read the current user goal, its applicable arguments and business object, and any later
     clarification or revision. Evaluate an action against the specification active when it
     occurred, using the latest applicable specification to assess final completion.
2. For an INFORMATION goal, determine whether the available tool observations
   and the assistant's response address the information requested by the user.
   A successful query alone does not establish that the response is supported.
3. For an ACTION goal, look for an attributed, successful state-changing call
   supporting the requested effect on the intended business object. A claim
   of completion in the assistant's reply is not evidence that the action
   occurred. Treat a handoff separately from completed execution.
4. Identify failed calls, a required state change that was not performed,
   repeated requests, explicit user dissatisfaction, and rejection of the
   assistant's conclusion. Assess whether later evidence repairs the problem.
   An unsuccessful intermediate call does not establish final failure after
   a supported successful retry.
5. Return POSITIVE only when unambiguous attributed evidence supports
   completion of the applicable goal. Return NEGATIVE when attributed evidence
   supports an unresolved failure. Otherwise return UNCERTAIN. In particular,
   ambiguous attribution, an unresolved instance boundary, missing completion
   semantics, or insufficient outcome evidence must not produce a positive
   or negative label. Withdrawal or supersession alone is not failure.
6. Cite the event IDs supporting the decision. Do not invent events, tool
   effects, object identifiers, or missing arguments.

Episode dialogue and event index:
{dialogue_and_event_index}

Intent instances (one record per reconstructed goal):
{instances_json_with_intent_id_user_spans_slots_business_object_
 attributed_event_ids_tool_calls_arguments_status_observations_replies_
 ambiguity_and_revision_history}

Canonical intent definitions from taxonomy memory:
{intent_definitions_json_with_canonical_goal_and_completion_type}

Output JSON only:
{"instances":[{"intent_id":"id","label":"POSITIVE|NEGATIVE|UNCERTAIN",
 "reason":"concise evidence-based explanation",
 "evidence_event_ids":["event id"],
 "negative_signal":true,"repeated_query":false,"claimed":false}]}
```

The three Boolean fields preserve the episode review's negative-signal, repeated-query, and completion-claim assessments. The outcome label follows the completion-type, temporal, repair, and uncertainty rules in Appendix B.3; the event IDs make its evidential basis auditable.

C.3 Candidate Grouping and Resolution

Candidate grouping is semantic rather than string based. Exact duplicate descriptions are grouped without an LLM call; otherwise the following compact template is used:

```
Determine whether the new candidate and one of the existing Candidate Ledger
descriptions denote the same business intent.
New candidate: {candidate_description}
Existing candidate descriptions: {ledger_descriptions_json}
Output JSON only: {"match_index": integer index or null}
```

Once a candidate has sufficient supporting records, the system compares it with up to five semantic neighbors from the active taxonomy. The following schematic English template summarizes the semantic resolution task and its output. Resolution identifies the canonical goal and, when available, its completion type; outcome labels are assigned from attributed evidence by the shared verifier.

```
Resolve how an e-commerce candidate intent relates to the current taxonomy.
The candidate comes from buyer messages. Decide whether it matches an existing
intent or represents a distinct request that the buyer wants the agent to complete.

There are three outputs:
- new: the candidate expresses a distinct, independently closable goal.
  State how dialogue evidence distinguishes it from each relevant neighbor
  and whether completion requires a state-changing action, if identifiable.
- merge: the candidate has the same canonical goal as an existing intent and
  inherits its available completion type. Differences in parameters or
  optional steps belong to that intent rather than defining a new one.
- defer: the available evidence is insufficient to establish a distinct goal,
  or identify an equivalent neighbor.

Every discriminator must be answerable directly from dialogue content. When
the decision is uncertain, choose defer rather than forcing new or merge.

For new, decide whether completion necessarily requires a system state change,
such as create, modify, cancel, or submit. Return false for information-only
intents; leave the type unspecified if this cannot be determined reliably.
For merge, inherit the completion type from the target intent.

Candidate description: {candidate_description}
Previously proposed ids: {candidate_aliases}
Nearest taxonomy intents: {neighbor_records_json}

Output JSON only:
{"outlet":"new|merge|defer",
 "target_id":"required for merge",
 "new_id":"object.action; required for new",
 "description":"required for new",
 "discriminators":{"neighbor id":"question; required for new"},
 "requires_terminal_action":true|false|null,
 "reason":"rationale"}
Include only fields applicable to the selected outlet. For defer, return
only outlet and reason.
```

Neighbor selection asks the model to return the five semantically closest identifiers from the serialized taxonomy. In this method-level template, `defer` denotes an unresolved candidate. The canonical goal and available terminal-action requirement are used to inform the shared outcome assessment. Unresolved candidates remain in the Candidate Ledger for later reconsideration.

C.4 Evidence-Contrastive Procedure Induction

Before prompting, records with ambiguous attribution or uncertain outcomes are removed. If no positive record remains, no procedure is generated. For the remaining records, the program computes the longest common subsequence of tool names across all positive trajectories and supplies it as a structural hint. An English rendering of the induction template and input fields follows:

```
Distill experience for an e-commerce service agent from multiple execution
trajectories of the same business intent. Each instance contains its tool
sequence and outcome.
```

```
The output must use one skill structure with progressive disclosure:
1. The first line must be exactly "description: <one-sentence summary>".
   It summarizes the handling path and is visible during intent matching; it
   must be understandable without reading the body.
2. After one blank line, write Markdown using only the following level-two
   headings, omitting any empty section:
   - "## Applicable scenarios": one or two sentences describing triggers or
     typical user expressions.
   - "## Handling steps": use the common subsequence across successful
     instances as the backbone; express differences as optional branches with
     their trigger conditions.
   - "## Common errors": include only when both positive and negative records
     exist; locate the divergence at a concrete step and state what omission
     causes what failure.

If all instances failed, return an empty output rather than inventing
experience. Tool names must exactly match names observed in the input; never
invent tools. Make the procedure quickly scannable and omit non-actionable
restatement.

Intent id: {canonical_intent_id}
Common tool subsequence across positive instances (reference only):
{tool_1 -> tool_2 -> ...}
All instances:
{instances_json_with_outcome_reason_slots_user_text_tools_and_guard_flags}
```

C.5 Executor and Procedure Injection

All methods use the unmodified ECom-Bench customer-service system prompt, including the platform, shop, user, confirmation, and tool-policy fields. IEL adds only the following method-specific injection block:

```
The following reference experience was returned by the intent router. Use it
only when it fits the current request and current tool evidence; otherwise
ignore it.
- For policies or rules about a specific product, rely on the actual information returned by
      tools for that product.
- If the tool cannot find the product, or its result clearly conflicts with the
  user's product description, do not fill in product-specific policy using
  general knowledge, industry conventions, or the stored experience.

### Intent experience: {canonical_intent_id}
{stored_procedure_markdown}
```

This block is inserted as a separate user-role message immediately before the current user message, not as a system instruction. It is inserted only on the first model call of a turn and is not repeated during the subsequent tool loop. Retrieval is an exact canonical-key lookup and has no Top-k truncation.