

Eluna: An Agentic LLM System for Automating Warehouse Operations with Reasoning and Task Execution

Ning Liu* P Aditya Sreekar* Kalle Kujanpää* Zhaoxuan Zhu* Kaiwen Liu
Chuanneng Sun Jorge Marchena Menendez Matthew Bales Tianyu Yang
Shahnawaz Alam Rose Yu Baoyuan Liu Kristina Klinkner Shervin Malmasi

Amazon.com, Inc. Fulfillment Technologies and Robotics
{ningliun,malmasi}@amazon.com

Abstract

Warehouse operations are governed by Standard Operating Procedures (SOPs) that encode complex, multi-system decision logic, which must be executed reliably under strict time constraints, yet LLM agents lack mechanisms to enforce procedural compliance and degrade under the context overload full SOP specifications introduce. We present Eluna, a production-deployed agentic system for reliable SOP execution. Eluna is a graph-guided, multi-agent framework that encodes SOPs as directed acyclic graphs with progressive disclosure and delegates independent tasks to parallel sub-agents, each with persistent code execution and live data access. To meet production latency and accuracy needs, we use asymmetric episodic distillation where a strong teacher is improved through episodic error memories, then a smaller student is fine-tuned on the corrected trajectories with memory stripped, internalizing corrections without inference-time overhead. On a 13-task benchmark and two production applications, our fine-tuned models match or exceed their teacher, beat all larger off-the-shelf baselines, and reach 94% expert agreement on the ticket processing application.

1 Introduction

Modern warehouse operations require continuous monitoring, multi-step diagnosis, and timely intervention across dozens of systems, ranging from robotic conveyance operators tracing threshold breaches to root causes within minutes, to inventory ticket processing that queries state, validates constraints, and submits physical item picks across many systems per ticket. Robotic conveyance diagnosis and ticket processing are two among many such workflows, each governed by Standard Operating Procedures (SOPs): directed decision pathways with prescribed steps, dependencies, quantitative thresholds, and constraints that must be followed faithfully. Today much of their execution is manual,

slow, and error-prone, and delays in diagnosis directly degrade throughput, making automated SOP execution a pressing business need.

Large language models reason well (Wei et al., 2022; Kojima et al., 2022), but agentic frameworks such as ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2024) rely on loosely structured prompts with no mechanism to strongly guide SOP compliance, so a capable model can still deviate from prescribed decision paths. Recent benchmarks confirm this gap on workflow-guided tasks (Xiao et al., 2024; Wang et al., 2025), industrial SOP-following (Nandi et al., 2025), and operational diagnostics where even SOP-enhanced multi-agent systems plateau in performance (Pei et al., 2025). A core bottleneck is context overload: with the full SOP in view, performance degrades as complex workflows overwhelm the model’s ability to select relevant actions (Xiao et al., 2024; Pei et al., 2025). This plateau holds regardless of model scale, and prompting alone is insufficient without domain-specific training (Nandi et al., 2025).

We present a graph-guided agent framework that addresses these challenges through a unified execution model. SOPs are encoded as directed acyclic graphs, and progressive disclosure surfaces only the reachable subgraph and node-level specifications on demand. A main agent orchestrates traversal and delegates independent node evaluations to parallel sub-agents in isolated contexts, each with a persistent code interpreter (the CodeAct paradigm of Wang et al. (2024)) and access to real-time data via the Model Context Protocol (Anthropic, 2024), providing context isolation and parallelism while guiding the agent toward procedural compliance. Together, progressive disclosure and sub-agent isolation address context overload. Each operational use case is packaged as a self-contained *Skill*¹ loaded on demand, so new workflows need no be-

¹<https://agentskills.io/>: a bundle comprising the decision graph, use-case-specific tools, node specifications, and execution instructions.

* Equal contribution.

spoke systems. Since prompting alone is insufficient (Nandi et al., 2025), we pair the framework with a trajectory-centric training pipeline where episodic learning corrects a strong teacher’s trajectories without weight updates, and a smaller, low-latency student is fine-tuned without episodic memory (asymmetric episodic distillation), internalizing the episodic corrections in its weights rather than depending on them at inference. We make three contributions:

- A **graph-guided skill-based agent framework** that encodes SOPs as directed acyclic graphs with progressive disclosure, and uses hierarchical multi-agent execution with parallel sub-agent delegation for scalable, procedurally compliant reasoning across multiple operational use cases.
- A **trajectory-centric training pipeline with asymmetric episodic distillation**: episodic learning iteratively improves teacher trajectory quality, and the student is fine-tuned on per-turn decomposed trajectories filtered by rejection sampling, with episodic memory stripped, internalizing the episodic corrections in its weights and eliminating inference-time memory dependence.
- **Real-world performance** on a 13-task operational reasoning benchmark and two production warehouse applications (robotic conveyance and ticket processing), demonstrating that a fine-tuned 32B model outperforms all baselines including its teacher and larger off-the-shelf models. A fine-tuned 355B model achieves a 94% human-match on the ticket processing application.

2 Related Work

Tool-Augmented Agents and Multi-Step Reasoning. LLM agents have been equipped with code execution (Gao et al., 2023; Wang et al., 2024), APIs (Patil et al., 2023; Qin et al., 2024), and standardized tool protocols (Anthropic, 2024). Multi-step reasoning methods range from chain-of-thought (Wei et al., 2022) and tree-of-thoughts (Yao et al., 2024) to interleaved reasoning and action (Yao et al., 2023) and hierarchical decomposition (Khot et al., 2023; Besta et al., 2024). These approaches improve general reasoning but rely on loosely structured prompts without mechanisms to enforce deterministic procedural constraints.

Structured Reasoning and Operational AI.

Multi-agent frameworks such as AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024) enable task decomposition across specialized agents. SOPStruct (Garg et al., 2025) converts unstructured SOP text into DAG representations, but is an upstream *structuring* system: its output is a structured graph, and its evaluation measures structuring quality (completeness and soundness of the DAG). It has no execution engine, tool use, or assessment of task completion, so it occupies a different pipeline stage than Eluna, which addresses the downstream problem of faithfully *executing* a structured SOP graph in production; a direct empirical comparison is therefore not meaningful, and the two are complementary. Agent-S (Kulkarni, 2025) navigates SOPs via prompt-driven state machines but lacks graph-guided progressive disclosure, code execution, and post-training; we compare empirically against this paradigm in Section 5.3. Our work integrates SOP graphs into both inference and training.

Trajectory-Centric Training. Reflexion (Shinn et al., 2024) and Self-Refine (Madaan et al., 2024) use iterative feedback to improve trajectories at inference time. FireAct (Chen et al., 2023) and AgentTuning (Zeng et al., 2024) show that fine-tuning on agent trajectories improves generalization. Our pipeline differs in two ways: episodic learning improves the *teacher* trajectories used for distillation rather than the deployed model, and training is aligned with graph-structured procedural logic rather than free-form tasks.

3 A Graph-Guided Operational Agent

Eluna is a hierarchical multi-agent framework that addresses the two challenges above (context overload and use-case generality) through graph-structured progressive disclosure, parallel sub-agent delegation, and skill packaging. Figure 1 provides an overview.

3.1 Graph-Structured SOP Representation

Each SOP is a directed acyclic graph $G = (V, E)$ with five node types: *observations* (detectable conditions), *root causes* (diagnosed causes), *calculations* (metric analysis), *constraints* (preconditions), and *actions* (remediation). Each node has a parameterized specification defining its evaluation procedure, which produces a structured outcome: a boolean, a classification (e.g., a category that routes downstream traversal), or a computed value. Edges

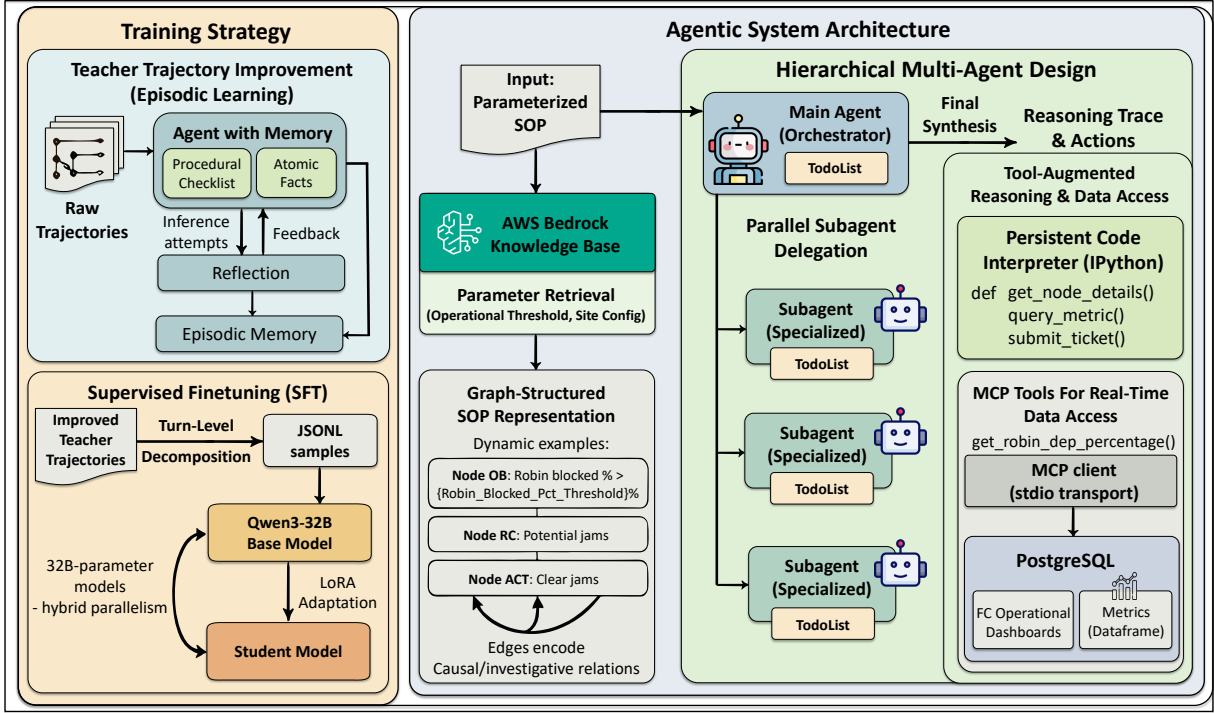


Figure 1: Overview of the Eluna framework. **Left:** Training strategy combining episodic learning for teacher trajectory improvement with turn-level decomposition for supervised fine-tuning via LoRA. **Right:** system architecture featuring graph-structured SOP representation, hierarchical multi-agent design with parallel sub-agent delegation, and tool-augmented reasoning through persistent code execution and MCP-based data access.

encode directed dependencies: a single-node edge activates the child on the boolean outcome of one parent (holding, or not holding for fallback paths), while a multi-node edge requires several parents to hold simultaneously. A node may have multiple outgoing edges.

Progressive Disclosure. Presenting the full graph degrades traversal accuracy (Liu et al., 2024), especially with parallel paths, where the agent conflates branches, omits nodes, or explores irrelevant subtrees. We disclose the graph at two levels. First, a retrieval tool runs BFS from the triggered node and returns only the reachable subgraph (node types, edge relationships, brief descriptions). Second, each node’s full procedural specification (evaluation logic, required tool calls, thresholds) is fetched only when the agent begins evaluating it, bounding working context to one node at a time.

3.2 Hierarchical Multi-Agent Architecture

Eluna uses a two-level hierarchy: a main agent orchestrates traversal while sub-agents evaluate individual nodes in parallel. The main agent holds the retrieved subgraph, selects the next nodes to evaluate from edge dependencies, and issues delegation calls; each sub-agent receives a fresh context with

only the assigned node’s procedural specification, role instructions, and tools. Action nodes remain with the main agent because they require graph-level state for sequencing and constraint checking.

Multiple delegation calls issued in one turn execute in parallel, reducing wall-clock time to roughly the slowest single evaluation. Only each sub-agent’s structured conclusion is returned, so the main agent’s context grows by one summary per node rather than the full investigation history; sub-agent sessions persist so the main agent can query one for detail without re-running it, and sub-agents cannot spawn further sub-agents. This isolates tool outputs and intermediate computation within sub-agent sessions, addressing context overload by construction while the main agent holds only graph structure and node conclusions. In a parallel effort, we reduce the per-node sub-agent overhead by compiling repeated SOP steps into reusable tools at build time (Kujanpää et al., 2026).

3.3 Skill Packaging

Each warehouse workflow needs its own decision logic, prompts, data sources, and actions. A custom agent per use case duplicates the execution engine and training pipeline, while a single agent

loaded with all use cases suffers context overload and tool-space conflicts. The workflow is a more natural packaging unit: we bundle per-workflow artifacts into a self-contained *skill* loaded at runtime, following the standardized skill format models are increasingly trained to use. The framework stays use-case-agnostic so the same agent and execution logic serve any loaded skill, and adding a workflow means authoring a skill, not modifying the agent.

A skill comprises five components: (1) the decision graph encoding the SOP as a DAG, (2) use-case-specific tools accessed via the Model Context Protocol (Anthropic, 2024) for querying data and executing actions, (3) node detail specifications containing per-node evaluation logic, (4) CodeAct functions (Wang et al., 2024), and (5) execution instructions directing traversal strategy (e.g., when to delegate, how to handle constraints); reference documents for knowledge retrieval (Schick et al., 2024) are bundled as readable files. When a trigger fires, a skill-loading tool registers the skill’s tools under a namespace prefix, injects its CodeAct functions into the persistent interpreter as importable modules, and appends its execution instructions, after which the agent has exactly the tools and instructions for the current use case.

Automated SOP Authoring. Skills do not require manual graph authoring. A separate LLM-based authoring agent, developed in parallel, extracts operational knowledge from existing documents and subject-matter-expert (SME) interviews, converts it into the formal graph representation (identifying node types, edge dependencies, and evaluation logic), and compiles the result into an executable skill package. A validation loop closes the process: Eluna generates reasoning traces against the compiled skill, disagreements with expected outcomes are surfaced to the SME for feedback that drives targeted graph refinements, and the loop iterates until agreement metrics meet a predefined threshold, at which point the skill is deployed. This authoring pipeline is orthogonal to the execution framework and training strategy that are the focus of this paper, so we do not detail it further, but it makes the approach scalable to new workflows.

3.4 Tool Ecosystem

The agent operates over five tools: a persistent code interpreter for programmatic data manipulation and metric computation; a task-tracking list that doubles as a steering mechanism to keep traversal on

the intended path; MCP-based data and action access (Anthropic, 2024) to operational dashboards; agentic RAG (Schick et al., 2024) for on-demand knowledge retrieval; and an operative memory that persists across invocations, accumulating knowledge and enabling deduplication of redundant investigations. Appendix A details each one.

4 Training Strategy

The graph-guided framework (Section 3) is model-agnostic, but even the largest available models fail to reach reliable accuracy on graph-structured procedural reasoning (Nandi et al., 2025), motivating domain-specific training. Our trajectory-centric pipeline has three stages. First, a strong teacher generates trajectories within the full framework, and episodic learning (EL) (Shinn et al., 2024). EL is a text memory of past errors injected into the prompt, which iteratively corrects the teacher’s mistakes without weight updates. Second, rejection sampling (Yuan et al., 2023) with LLM-judge evaluation (Zheng et al., 2024a) retains only correct trajectories. Third, a smaller student that meets the production latency SLA is fine-tuned on these trajectories *with the EL memory stripped*. We call this *asymmetric episodic distillation* because EL is applied only on the teacher side and the student never sees the memory during training or inference, so it must internalize the corrections in its weights, and no runtime memory store is needed. Appendix B details each stage.

5 Experiments

5.1 Experimental Setup

The agent framework is implemented using the Strands Agents SDK (Strands Agents, 2025), following the design in Section 3. We fine-tune Qwen3-32B (Yang et al., 2025) and GLM-4.5-Air (Zeng et al., 2025) with LoRA (Hu et al., 2022) on 8,525 training samples (expanded from trajectories via turn-level decomposition, Appendix B). Training and inference details are in Appendix E.

Datasets We evaluate on three tasks: an **operational reasoning benchmark** of 13 task types testing capabilities required for warehouse operational reasoning (1,400 evaluation samples; Table 1), a **robotic conveyance application** covering end-to-end SOP execution over 1,500 real scenarios, and a **ticket processing application** executing a 46-node inventory consolidation SOP on 410 real

Task	GLM-4.7	Qwen3-235B	Qwen3-32B	Eluna-Q	Eluna-G
1 Metric Reading	+6.6	-1.1	-57.1	+5.2	+6.6
2 State Classification	+4.3	-22.3	-34.0	+2.4	+2.1
3 Operational Language Understanding	+4.4	-12.1	-7.7	+4.4	+1.1
4 Metric Trend Detection	+11.9	+9.5	-36.9	+10.7	+14.3
5 Causal Chain Analysis	+25.0	-22.4	-38.2	+13.2	+23.7
6 Statistical Pattern Analysis	-5.0	+1.2	-80.0	+9.1	0.0
7 Knowledge Base Lookup	+3.4	-20.7	-8.0	+4.6	+3.4
8 Query Understanding	+11.1	-13.9	-31.9	+19.0	+12.5
9 Cross-System Knowledge Synthesis	+6.3	+15.9	-12.7	+7.9	0.0
10 Complex Query Decomposition	-2.2	-21.3	-37.1	+1.1	-1.1
11 Temporal Comparison	+14.3	-8.6	-77.1	+2.9	+14.3
12 Multi-Factor Context Awareness	+2.2	-5.6	-31.5	-3.4	+7.9
13 Knowledge Base Context Filtering	+3.3	-20.9	-45.1	+4.7	+3.3
AVG	+6.1	-10.0	-38.0	+6.0	+6.5

Table 1: Operational benchmark: % improvement over the GLM-4.5-Air baseline across 13 tasks.

tickets evaluated against expert annotations. For the first two, ground truth is generated from real operational data by sampling timestamps, checking root-node conditions, and running a deterministic SOP implementation where they hold, with temporal partitioning separating training from evaluation. Ticket processing has no deterministic ground truth, so correctness is estimated from expert review of sampled agent traces.

Evaluation Claude Sonnet 4 is used as the automated judge of answer correctness for the first two tasks. On the benchmark, the judge scores model outputs against reference answers. On the robotic conveyance application, node classifications are extracted from agent trajectories and compared against ground-truth labels, reporting binary correctness (per-node accuracy) and exact match (all nodes of a type correct within a scenario). Because ground truth is deterministic, judging reduces to structured matching rather than open-ended assessment; the LLM judge was validated against human judgements and achieves full agreement, confirming its reliability for this structured evaluation setting. For ticket processing, correctness is estimated from expert review of agent traces (Appendix F). Results are reported as improvement over the GLM-4.5-Air off-the-shelf (OTS) baseline.

Models All models are evaluated within the same framework, with identical tools and skills and with episodic memory stripped at inference (Section 3). The **baseline** is OTS GLM-4.5-Air, a 106B parameter model that meets the production latency SLA,

hence the reference point for what is achievable under operational constraints. GLM-4.7 (355B), the **teacher** that generated training trajectories, is too large to meet that SLA, so it serves as a trajectory generator and upper reference rather than a deployable baseline. GLM-4.7 was selected as teacher because it achieved the highest off-the-shelf accuracy on our benchmark among the open-weight models we evaluated, and its reasoning patterns align with our target student (GLM-4.5-Air), facilitating knowledge transfer; distillation from closed commercial models is not permitted under their terms of service, constraining teacher selection to open weights. The distillation framework itself is model-agnostic, and any sufficiently capable open-weight teacher can be substituted. We additionally compare OTS Qwen3-235B-A22B-Thinking and OTS Qwen3-32B, against our two fine-tuned models Eluna-Q (Qwen3-32B) and Eluna-G (GLM-4.5-Air). For the ticket task, which runs on a longer interval, we additionally fine-tune the teacher itself into Eluna-G-XL, whose latency meets that SLA.

5.2 Results

Operational Benchmark. Table 1 reports per-task results. Eluna-G improves 6.5% and Eluna-Q improves 6.0% over the baseline on average. Eluna-Q, trained on GLM-4.7 trajectories, matches the teacher (+6.0% vs. +6.1%) despite being a different and smaller architecture, showing that our pipeline transfers operational reasoning across model families, while OTS Qwen3-32B averages 38.0% below the baseline, showing the gap that domain-specific

Model	Binary Correctness			Exact Match		
	OB	RC	ACT	OB	RC	ACT
GLM-4.7 (teacher)	+50.2	+73.7	+8.5	+93.7	+84.0	+164.8
Eluna-Q	+48.9	+73.9	+8.5	+93.7	+84.4	+165.3
Eluna-G	+49.5	+73.9	+8.5	+93.9	+84.4	+164.8

Table 2: Robotic conveyance application: % improvement over GLM-4.5-Air baseline, averaged over 4 runs.

training closes. The largest gains concentrate on complex multi-step tasks like Causal Chain Analysis (Task 5), Query Understanding (Task 8), and Query Decomposition (Task 10). These require chaining tool calls and synthesizing across data sources, indicating that training instills operational reasoning rather than single-step retrieval.

Robotic Conveyance Application. End-to-end SOP execution is far harder than isolated benchmark tasks: even with full framework access, the OTS baseline compounds per-node errors across the traversal and degrades sharply. We report binary correctness and exact match (Section 5.1) per node type (observations, root causes, and actions) to reflect the distinct sub-tasks of detection, diagnosis, and remediation. Fine-tuning on complete trajectories closes the gap (Table 2): all trained models improve observation binary correctness by over +49%, root-cause binary correctness by over +73%, and exact match by +84% on root causes and +165% on actions, with our 32B model matching the GLM-4.7 teacher across all metrics, which shows that trajectory distillation transfers full SOP traversal behavior, not just isolated skills.

Median end-to-end execution latency (wall-clock from query to final action) is also reduced by 54.3% (Eluna-Q) and 54.8% (Eluna-G) relative to GLM-4.7, meeting the latency constraints for real-time diagnostic support.

5.3 Ablation Studies

Architectural Ablation. Table 3 isolates the contribution of the two architectural components of Section 3, namely progressive disclosure and multi-agent delegation, by running GLM-4.7 on the robotic conveyance application under all four configurations. The single-agent, full-SOP configuration doubles as a paradigm comparison: it simulates Agent-S (Kulkarni, 2025), where a single LLM receives the entire SOP as a prompt-driven state machine without progressive disclosure or parallel delegation.

Configuration	OB	RC	ACT
Single-agent, full SOP (Agent-S style)	+76.1	+73.3	+129.5
Multi-agent, full SOP	+81.0	+77.6	+137.1
Single-agent, progressive disclosure	+91.6	+77.7	+158.0
Multi-agent, prog. disclosure (Eluna)	+93.9	+84.4	+164.8

Table 3: Architectural ablation on the robotic conveyance application (All rows use GLM-4.7): exact-match % improvement over the GLM-4.5-Air baseline, as in Table 2.

Setting	Task 5 (%)	Task 10 (%)
No EL anywhere (reference)	0.0	0.0
EL in teacher + student (train & eval)	+16.6	+25.3
EL in teacher + student (train only)	-52.7	-40.0
EL in teacher only (Eluna)	+16.2	+28.6

Table 4: Episodic learning (EL) ablation: % change relative to the no-EL variant.

Both components contribute meaningfully. Progressive disclosure provides the larger individual gain (rows 1 vs. 3: +76.1 to +91.6 on observations, +129.5 to +158.0 on actions), consistent with the context-overload hypothesis of Section 3.1. Sub-agent delegation adds complementary improvements, particularly on actions (+158.0 to +164.8), where isolated node evaluation benefits graph-level state management. The full Eluna architecture achieves the best results across all metrics and outperforms the Agent-S-style configuration by +17.8 on observations, +11.1 on root causes, and +35.3 on actions, confirming that the gains come from how the SOP graph is integrated into inference rather than from model strength alone.

Episodic Learning Ablation. Table 4 isolates the contribution of EL by varying where the EL memory is applied. We use Tasks 5 and 10 from the benchmark as a deliberate proof of concept before full-scale training: they are the hardest tasks for GLM-4.7 (Table 1), maximizing the room for episodic correction to demonstrate measurable gains; the full pipeline (Tables 1 and 2) then applies the validated design across all 13 tasks and both production applications. Results are reported as percentage change relative to the no-EL variant.

Applying EL only to the teacher (our approach) yields +16.2% on Task 5 and +28.6% on Task 10, matching the variant that also feeds EL to the student at inference (Task 5: +16.6%, Task 10: +25.3%) but without its runtime memory dependence. Training the student with EL yet removing it at inference degrades catastrophically (Task 5: -52.7%, Task 10: -40.0%), confirming that our

Metric	Value
Overall estimated correctness	94.4%
Consolidation decision precision	96.7%
Consolidation decision recall	99.2%
Consolidation decision F1	98.0%
Terminal distribution	
Proceed to consolidation	84.1%
Rejected (no eligible inventory)	13.7%
Manual handling (outside SOP scope)	2.2%

Table 5: Ticket processing: Eluna-G-XL (open-weight 355B fine-tune) correctness (Appendix F).

asymmetric design must distill episodic knowledge into parameters rather than expose it as a brittle inference-time dependency. Appendix C details the four variants.

6 Deployment and Business Impact

Our framework is deployed in production across multiple operational applications, served on Amazon Bedrock AgentCore (Amazon Web Services, 2026). The two applications below apply it in contrasting regimes: robotic conveyance is latency-critical metric analysis, while ticket processing is latency-insensitive but needs complex reasoning.

6.1 Robotic Conveyance

This metric-analysis task queries live dashboards to compute metrics, and compares them against thresholds over a 63-node SOP graph whose high branching factor drives parallel sub-agent delegation. A degraded station can back up the main conveyor and erode facility-wide throughput in minutes, so diagnosis must be fast and correct.

In production deployment at a warehouse, the agent has processed roughly 8,000 triggers, producing 3,435 process alerts and 1,078 maintenance tickets. SOP adherence is near-perfect: SME review of agent trajectories finds the execution correct. The remaining errors trace to gaps in the SOP specification and upstream data quality rather than to agent reasoning. Prior to deployment, operators manually interpreted dashboards, navigated SOPs, and coordinated corrective actions. Early deployment feedback indicates a 300% speedup in resolution time while reducing cognitive load and improving SOP adherence consistency.

6.2 Ticket Processing

This is a language-reasoning task. The agent interprets free-text ticket requests (expiration, dam-

age, recalls, audits) and performs multi-step reasoning over structured inventory records (validating constraints, resolving eligibility, and deciding consolidation) rather than merely parsing the request. Each request type routes differently, and eligibility depends jointly on item condition and storage-location type. The SOP graph contains 46 nodes and is predominantly sequential, with mutually exclusive conditional paths. The network processes tens of thousands of these tickets daily. Approximately 80% are handled through an existing scanning workflow. The remaining 20% require manual desk processing at ≈ 6 minutes per ticket. We deploy Eluna-G-XL, an open-weight 355B fine-tune from the same pipeline. The agent’s median generation duration is 187s (p90: 257s, mean: 210s), below the manual rate.

The agent automates the digital triage preceding physical inspection: interpreting the request, querying inventory, validating constraints, and submitting the pick, leaving the associate only the inspection. Precision (Table 5) is the operationally critical metric, as a false consolidation submits a pick that cannot be processed, wasting labor and blocking the queue. Only a small fraction of tickets need manual handling, freeing capacity for substantially larger volumes.

7 Conclusion

We presented Eluna, a production-deployed agentic system for reliable SOP execution that combines graph-guided progressive disclosure, hierarchical multi-agent delegation, and skill packaging with a trajectory-centric training pipeline built on asymmetric episodic distillation. Deploying Eluna in production surfaced two lessons that generalize beyond our setting. First, within the graph-guided framework, distillation beats scale for procedural compliance: a fine-tuned 32B model matches a far larger teacher and surpasses every OTS model tested, which plateau regardless of size. Second, the operational constraint of low latency under a production SLA is what forced the asymmetric distillation design, since exposing episodic memory at inference proved a brittle dependency rather than just an accuracy gain. The residual errors we observe trace to SOP-specification gaps and upstream data quality rather than agent reasoning, pointing to the SOP authoring and data-integrity pipeline as the next bottleneck for reliable operational automation.

Limitations

Our system is currently deployed on two operational domains; while the skill-based architecture is designed for generality, transfer to substantially different operational environments remains to be validated empirically. The episodic learning pipeline assumes access to a capable teacher model and sufficient compute for multiple trajectory generation rounds, which may limit adoption in resource-constrained settings. Finally, both episodic learning and fine-tuning rely on ground-truth labels, which are expensive and time-consuming to obtain at scale; reducing this dependence is important future work.

References

- Amazon Web Services. 2026. *Amazon Bedrock Agent-Core Developer Guide*. Accessed: 2026-06-04.
- Anthropic. 2024. Model context protocol. <https://modelcontextprotocol.io>.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefer. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence*.
- Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *International Conference on Machine Learning*.
- Deepeka Garg, Sihan Zeng, Sumitra Ganesh, and Leo Ardon. 2025. Generating structured plan representation of procedures with LLMs. *arXiv preprint arXiv:2504.00029*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.
- Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *International Conference on Learning Representations*.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35.
- Kalle Kujanpää, Ning Liu, Shah Nawaz Alam, Yeshwanth Reddy Sura, Tianyu Yang, Kristina Klinkner, and Shervin Malmasi. 2026. Tool-making and self-evolving LLM agents in low-latency systems. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Budapest (Hungary). Association for Computational Linguistics.
- Mandar Kulkarni. 2025. Agent-S: LLM agentic workflow to automate standard operating procedures. *arXiv preprint arXiv:2503.15520*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 12:157–173.
- Ning Liu, Siavash Jafarzadeh, Brian Y Lattimer, Shuna Ni, Jim Lua, and Yue Yu. 2025. Harnessing large language models for data-scarce learning of polymer properties. *Nature Computational Science*, 5(3):245–254.
- Ning Liu, Chuanneng Sun, Kristina Klinkner, and Shervin Malmasi. 2026. Beyond pairs: Your language model is secretly optimizing a preference graph. *arXiv preprint arXiv:2605.08037*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2024. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.
- Subhrangshu Nandi, Arghya Datta, Nikhil Vichare, Indranil Bhattacharya, Huzefa Raja, Jing Xu, Shayan

- Ray, Giuseppe Carenini, Abhi Srivastava, Aaron Chan, and 1 others. 2025. Sop-bench: Complex industrial sops for evaluating llm agents. *arXiv preprint arXiv:2506.08119*.
- Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.
- Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, and 1 others. 2025. Flow-of-action: Sop enhanced llm-based multi-agent system for root cause analysis. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 422–431.
- 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. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.
- Strands Agents. 2025. Strands agents sdk. <https://github.com/strands-agents/harness-sdk>. Accessed: 2026-06-04.
- Jiaming Wang, Zhe Tang, Zehao Jin, Hefei Chen, Yilin Jin, Peng Ding, Xiaoyu Li, and Xuezhi Cao. 2025. Sop-maze: Evaluating large language models on complicated business standard operating procedures. *arXiv preprint arXiv:2510.08942*.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.
- Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. 2024. Flowbench: Revisiting and benchmarking workflow-guided planning for llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10883–10900.
- An Yang, Baosong Yang, Beichen Zhang, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.
- Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.
- Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. *arXiv preprint arXiv:2310.12823*.
- Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.
- Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. 2024. *Swift: a scalable lightweight infrastructure for fine-tuning*. Preprint, arXiv:2408.05517.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2024a. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36.
- Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024b. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

Appendix

A Tool Ecosystem Details

Code Interpreter. The agent performs data manipulation and metric computation through a persistent Python interpreter (CodeAct (Wang et al., 2024)) whose state (variables, imports, and intermediate results) is retained across tool invocations within an investigation. Operational data retrieved through MCP tools is analyzed programmatically to perform operations such as filtering, aggregating, and computing derived quantities, so that only the resulting view enters the model’s input context rather than the full table. Executing comparisons and threshold checks in code eliminates arithmetic errors that arise when models perform these operations in natural language, and loops over CodeAct functions (programmatic wrappers exposed within the interpreter) let the agent batch repeated queries without issuing a separate tool call per iteration.

Task Tracking. A `ToDoList` tool, exposed within the interpreter, lets the agent record and update the steps of an investigation as it traverses the decision graph. Beyond bookkeeping, this serves as a steering mechanism: maintaining an explicit plan keeps the agent on the intended path through parallel branches, surfaces which nodes remain to be evaluated, and reduces premature termination on multi-step procedures.

MCP Tools. Data retrieval and actions are implemented through the Model Context Protocol (Anthropic, 2024), providing a standardized interface to warehouse operational dashboards. A `query_metric` interface abstracts tool invocation, accepting a tool name, warehouse identifier, and time range, and returning results as pandas DataFrames for the interpreter to consume. Decoupling the agent’s reasoning from data-source implementation lets dashboard backends evolve independently, and thread-safe connection management enables concurrent queries from parallel sub-agents.

Agentic RAG. The agent augments its decision-making with domain knowledge retrieved on demand. Unlike static pipelines that prepend fixed context (Lewis et al., 2020), the agentic RAG component operates via tool-calling (Schick et al., 2024): when the agent detects a knowledge gap, it formulates a query and invokes a knowledge-base tool, receiving relevant passages as a tool response.

This grounds reasoning in operational definitions, site-specific configurations, and historical incident documentation, and lets these definitions be updated without modifying the core SOP structure.

Memory. The framework includes an operative memory that persists across agent invocations, accumulating knowledge over time rather than treating each investigation in isolation. The agent queries memory for prior findings on an entity before investigating and writes its outcome afterward, so observations build into patterns that are not visible within a single run. Deduplication follows as a direct consequence: when a prior invocation already resolved the same condition or determined that no remediation is possible, the current agent omits the redundant investigation. Memory entries expire after a configurable window so that stale findings do not suppress legitimate re-evaluation.

B Training Pipeline Details

Episodic Learning for Teacher Improvement.

Initial teacher trajectories contain errors such as missed branches and threshold misinterpretations. Rather than manually correcting these, we iteratively improve the teacher through episodic learning (Shinn et al., 2024) without weight updates. The episodic memory is used only during teacher trajectory generation and is excluded from student training and inference. After each batch of teacher runs, trajectories are evaluated against ground truth. For failures, a separate LLM analyzes the error and generates candidate memory entries that would prevent recurrence. New entries are added, and a consolidation step merges redundant entries to keep the memory compact. The teacher is re-run with updated memory injected into its prompt, iterating for N rounds until convergence. Providing the memory at inference would yield equivalent accuracy (our ablation in Section 5.3 confirms this), but adds prompt overhead and requires maintaining a growing memory store. Training on improved trajectories without the memory forces the student to internalize the episodic corrections in its weights, eliminating runtime context dependence.

Trajectory Decomposition and Fine-Tuning.

Training trajectories are generated by running the teacher on labeled scenarios within the full framework (skills, MCP tools, code execution). For each scenario, the teacher queries operational data through the same tools used at inference time, pro-

ducing a multi-turn trajectory of reasoning, tool calls, and tool outputs. Scenarios are sampled with balanced representation across outcome categories, and temporal partitioning enforces train-test separation. We decompose each trajectory into T training samples (one per agent turn): the t -th sample takes as input the conversation history up to turn t (system prompt, all prior tool calls and outputs, with the model’s own reasoning from prior turns stripped) and targets the complete reasoning and action for turn t . Retaining tool outputs is necessary: omitting them would desynchronize the model’s state from the actual execution trace. Stripping prior reasoning forces independent generation at each turn, focusing the loss on single-turn decision quality. Not all trajectories are correct even after episodic improvement. We apply rejection sampling (Yuan et al., 2023) with LLM-judge evaluation (Zheng et al., 2024a): numerical outputs are compared with 2% tolerance, and categorical outputs require exact match. Only fully correct trajectories are retained for fine-tuning; exploiting the rejected trajectories through preference-based objectives (Liu et al., 2026) is a natural extension.

C Episodic Learning Ablation Details

To isolate the impact of episodic learning and understand whether asymmetric distillation is necessary, an ablation is performed on Tasks 5 and 10 from the benchmark (Table 4). Four variants are tested: (1) no EL anywhere, (2) EL in teacher only (our approach), (3) EL memory provided to the student during both training and inference, and (4) EL memory provided during student training only, removed at inference. Applying EL only to the teacher (our approach) yields +16.2% on Task 5 and +28.6% on Task 10 over the no-EL variant. When EL is given to the student at both training and evaluation, comparable accuracy is reached (+16.6%, +25.3%), but an inference-time dependence on the growing EL context is introduced, adding latency for no accuracy gain. Training with EL but evaluating without it degrades catastrophically (−52.7%, −40.0%), falling below even the no-EL variant. This shows that naively exposing the student to episodic context creates a brittle dependency. Our asymmetric design distills the episodic corrections into the student’s weights, removing any inference-time memory requirement.

D Cost Analysis

Training and deploying a capable model is an unavoidable investment when the goal is faithful execution of complex, multi-step SOPs with strict correctness requirements: off-the-shelf models plateau regardless of scale (OTS Qwen3-235B averages −10% versus baseline, and even the 355B teacher reaches only +6.1%; Table 1), consistent with the broader observation that general-purpose LLMs require domain-specific adaptation to perform reliably in specialized domains (Liu et al., 2025), while the alternative of manual execution at ≈6 minutes per ticket carries far higher ongoing operational cost. We nonetheless take deliberate steps to minimize compute requirements. First, LoRA fine-tuning (Hu et al., 2022) rather than full-parameter training reduces trainable parameters by orders of magnitude, completes in at most 2 epochs, and its rank can be adjusted to available resources; even a rank-32 fine-tune (Eluna-G) outperforms all OTS baselines. Second, asymmetric distillation compresses the 355B teacher into production-viable students (32B/106B) that match or exceed teacher accuracy, and the teacher is used only offline for trajectory generation, never at serving time. Third, at inference the fine-tuned 32B model is served on 8 H100 GPUs via vLLM with a 54% latency reduction versus the teacher at equivalent accuracy, and merged LoRA weights add zero inference overhead. At production scale these recurring savings, e.g. ≈8,000 autonomously processed conveyance triggers with a 300% resolution speedup or ticket processing below the manual rate (median 187s vs. ≈360s), substantially exceed the one-time training cost.

E Training and Inference Details

Models are fine-tuned using MS-SWIFT (Zhao et al., 2024) with LoRA (Hu et al., 2022). For Qwen3-32B, we use rank and alpha of 256, learning rate 2×10^{-4} , and train on 64 H100 GPUs. For GLM-4.5-Air, we use rank and alpha of 32, learning rate 1×10^{-4} , and train on 128 H100 GPUs. Both use a batch size of 64, 5% linear warmup followed by cosine annealing to zero, and train for 2 epochs. The ticket-processing model Eluna-G-XL fine-tunes GLM-4.7 (355B) with the same MS-SWIFT/LoRA pipeline (rank and alpha 64, learning rate 5×10^{-5} , batch size 64, 1 epoch) on 128 H100 GPUs. LoRA weights are merged after training, eliminating inference overhead. At inference,

Eluna-G and Eluna-Q are served via vLLM (Kwon et al., 2023) on 8 H100 GPUs, and Eluna-G-XL via SGLang (Zheng et al., 2024b) on 8 H200 GPUs.

F Ticket Processing Evaluation

Model. The deployed ticket-processing agent is Eluna-G-XL, an open-weight GLM-4.7 (355B) fine-tuned with the same trajectory-centric pipeline as Eluna-G and Eluna-Q (Appendix B, config in Appendix E). Ticket processing runs on a longer interval than robotic conveyance, so a larger fine-tune is admissible: Eluna-G-XL’s median generation duration (Section 6.2) meets that requirement. All results in Table 5 are from this single fine-tuned model.

Evaluation setup. The 410 tickets are an unbiased sample of open consolidation tickets at one fulfillment center on a single day, processed by the agent in the production framework with live inventory access. The task has no deterministic ground truth (the correct solution depends on live inventory state at processing time), so correctness is estimated by expert review of agent traces: three experienced operations associates per batch examined each original ticket and its correspondence alongside the agent’s reasoning trace, and judged the solution as correct or incorrect. Tickets were distributed across reviewers to maximize coverage rather than labeled redundantly, so by design no inter-rater agreement statistic exists for this review.