Expansion-Contraction: A Multi-Agent Graph Traversal Pattern for Compound AI Systems

Aiham Taleb
Amazon Web Services Inc.
Berlin, Germany
aitaleb@amazon.de

Zainab Afolabi
Amazon Web Services Inc.
London, United Kingdom
zafolabi@amazon.co.uk

Joao Sousa
Amazon Web Services Inc.
Lisbon, Portugal
joaosous@amazon.pt

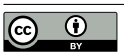
Mathias Seidel
Continental AG
Hannover, Germany
mathias.seidel@conti.de

Abstract

Compound AI systems that coordinate multiple specialized agents offer a promising path for complex reasoning tasks, yet principled architectural patterns for multi-agent coordination over structured data remain under-explored. We introduce *Expansion-Contraction*, a multi-agent graph traversal pattern in which an expansion phase walks a domain graph outward from a query origin, dynamically spawning ephemeral specialist agents at each node, and a contraction phase aggregates their findings inward to produce a verdict. Agent topology emerges isomorphically from the data graph rather than being hand-designed, and each agent operates on a small local context—avoiding the context-window saturation that degrades single-agent approaches on large graphs. We instantiate the pattern for supply chain root cause analysis, integrating domain-specific tools with temporal lead-time propagation. Across eight datasets (three real-world, five synthetic with controlled depth and width), Expansion-Contraction achieves 98.2% accuracy on a production supply chain (624 cases) and 100% on public benchmarks, outperforming single-agent baselines by 14+ percentage points while degrading gracefully as graph complexity increases. A deterministic depth-priority disambiguation heuristic, motivated by our failure analysis, further improves Dataset A accuracy to 99.5% (621/624, 95% CI [98.6%, 99.9%]). To assess transfer, we evaluate the pattern on a second domain—microservice dependency tracing over a 17-service DAG (100 scenarios)—where Expansion-Contraction reaches 88% overall accuracy and 85% on NLP-complex cases (vs. 55% for the next-best baseline). Investigation caching reduces token usage by up to 93.9%, concurrent path analysis yields up to 1.43× speedup, and a production deployment demonstrates the pattern's viability for enterprise-scale agentic systems.

CCS Concepts

• **Computing methodologies** → **Multi-agent systems**; *Natural language processing*; • **Software and its engineering** → *Software architectures*; • **Applied computing** → *Operations research.*



CAIS '26, San Jose, CA, USA

ACM ISBN 979-8-4007-2415-2/2026/05
https://doi.org/10.1145/3786335.3813170

Keywords

Compound AI Systems, Agentic Systems, Multi-Agent Coordination, Graph Traversal, Tool-Augmented Reasoning, Architectural Patterns, Supply Chain Root Cause Analysis

ACM Reference Format:
Aiham Taleb, Zainab Afolabi, Joao Sousa, and Mathias Seidel. 2026. Expansion-Contraction: A Multi-Agent Graph Traversal Pattern for Compound AI Systems. In *ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3786335.3813170

1 Introduction

The compound AI systems paradigm—orchestrating multiple specialized components rather than relying on a single monolithic model—has emerged as the dominant architecture for complex reasoning tasks [19]. A critical open question is how to *coordinate* these components: existing frameworks offer conversational patterns [16], role-based crews [12], and static state machines [9], but principled patterns for multi-agent coordination over *structured, graph-shaped data* remain scarce. Many real-world domains—supply chains, knowledge graphs, infrastructure dependency maps, organizational hierarchies—are naturally represented as directed graphs, yet current agentic architectures either serialize the entire graph into a single prompt (causing context-window saturation [11]) or assign agents via hand-designed topologies that do not mirror the underlying data.

Problem. Given a query on a directed acyclic domain graph—"why can't we fulfill this order?" or "which service caused this incident?"—identify the node whose local state is the root cause of a condition observed at a query origin, where causality propagates along edges with potential temporal offsets. Single-agent approaches must serialize the full graph into one prompt, incurring "lost-in-the-middle" degradation on realistic topologies; hand-designed multi-agent topologies require per-domain engineering and cannot react to graph instances that change between queries.

Approach. We introduce **Expansion-Contraction**, a multi-agent graph traversal pattern that operates in two phases: an *expansion* phase walks the domain graph outward from a query origin, dynamically spawning ephemeral specialist agents at each node; a *contraction* phase aggregates their findings inward to produce a verdict. Agent topology emerges *isomorphically* from the data graph—no hand-designed wiring—and each agent operates on a

small local context. Three graph-aware optimization techniques (investigation caching, concurrent path analysis, recursive lead-time propagation) exploit graph structure for efficiency (Section 4; ablated in Section 6.5).

We instantiate the pattern for supply chain root cause analysis—a representative graph-structured task where disruptions propagate along directed edges with temporal lead-time offsets—and probe transfer on microservice dependency tracing (Section 6.8).

Our contributions are: (1) the Expansion-Contraction pattern (Sections 3–4); (2) optimization techniques achieving up to 93.9% token savings and 1.43× speedup (Section 4); (3) evaluation across eight supply chain datasets and six baselines, with a deterministic depth-priority heuristic that lifts Dataset A accuracy to 99.5% (Sections 5–7); (4) cross-domain validation on microservice tracing (Section 6.8); and (5) a production deployment demonstrating enterprise viability (Section 6.7).

2 Related Work

2.1 Architectural Patterns for Compound AI Systems

The shift from monolithic models to compound AI systems—architectures combining multiple AI components with programmatic orchestration—represents a fundamental change in system design [19]. DSPy [7] introduced declarative programming for LLM pipelines, enabling systematic optimization of prompts and module composition.

Recent multi-agent frameworks offer diverse coordination patterns. AutoGen [16] provides conversational interaction patterns. CrewAI [12] organizes role-based crews with defined workflows. LangGraph [9] implements graph-based state machines for agent orchestration. Guo et al. [5] synthesize these into a unified framework of profile, perception, action, interaction, and evolution. A common limitation is that agent topology is *statically designed*—the number of agents and their communication graph are fixed at design time, not derived from the problem instance.

Agent-as-graph paradigms explore treating agents as computational graphs where prompts and connectivity are optimized [22]. Expansion-Contraction differs fundamentally: agent topology emerges isomorphically from the *domain graph* itself, not from optimization or learned selection. Each domain node spawns a specialist agent during expansion; contraction aggregates results along the graph's edges.

Concurrent execution in multi-agent systems presents unique challenges; DynTaskMAS [21] introduces dynamic task graphs for asynchronous coordination, while Wang et al. [14] propose dual-thread architectures with interruptible execution. Our pattern achieves parallelism through the natural structure of the domain graph—sibling branches are investigated concurrently, with synchronization at contraction boundaries.

A key limitation of single-agent approaches is context-window saturation: when a large graph is serialized into a single prompt, LLMs exhibit degraded reasoning over long contexts [11]. Multi-agent decomposition avoids this by giving each agent only its local subgraph, keeping per-agent context small and focused.

2.2 Tool-Augmented Reasoning

ReAct [18] established the reasoning-action paradigm for tool use. Toolformer [13] demonstrated self-supervised tool learning. Retrieval-Augmented Generation (RAG) [10] combines parametric models with non-parametric retrieval, enabling grounded generation from external knowledge. Tool-augmented agents extend this paradigm by grounding LLM reasoning in structured external data, mitigating hallucination—a critical requirement for enterprise systems where incorrect root cause attribution has direct business impact. Recent work on inference-time scaling explores cost/quality/latency tradeoffs through techniques like self-reflection [2]. Our system integrates domain-specific tools within coordinated graph traversal, extending tool use from single-agent to multi-agent orchestration where each agent accesses only the tools relevant to its site type.

2.3 Graph-Based Reasoning and Supply Chain AI

GNN approaches to supply chains [3, 8] are the dominant alternative for graph-structured root cause analysis, but require labeled training data, pre-defined schemas, and retraining when topology changes—assumptions violated in real supply chains that evolve with new suppliers and routes. Expansion-Contraction is zero-shot and generalizes across eight datasets with diverse topologies (Section 6) without retraining. Unlike agentic graph *construction* approaches [1] where agents iteratively build knowledge graphs, our pattern addresses agentic graph *traversal*—dynamically spawning topologies that mirror pre-existing data structures with temporal constraint propagation. For supply chain disruption detection specifically, Zhang et al. [20] combine causal discovery with RL for delivery risk attribution, and Heus et al. [6] use knowledge graphs for agentic risk analysis; we combine LLM reasoning with explicit graph traversal and temporal propagation from both structured and unstructured sources, without pre-trained causal models.

2.4 Temporal Reasoning in LLM Agents

Temporal reasoning remains challenging for LLMs, though recent work demonstrates learnable temporal capabilities [17]. Supply chain causality is inherently temporal: a block at an upstream plant on day $t-\delta$ causes a shortfall at a downstream site on day t, where δ is the cumulative lead time. Our system addresses temporal reasoning through explicit lead time propagation during graph traversal—for each upstream edge, the investigation date is adjusted by the supplier-to-customer lead time, expressed as $d_{\text{supplier}} = d_{\text{customer}} - \Delta t_{\text{lead}}$. This ensures constraint analysis occurs at the correct temporal context for each supply chain tier rather than relying on implicit LLM temporal understanding. We evaluate the impact of this design choice in Section 6.5: disabling temporal propagation reduces accuracy by 9.9 percentage points on Dataset A and 22.0 pp on DataCo (Table 2).

3 System Architecture

3.1 Running Example

To ground the discussion for readers outside supply chain, we refer to a concrete scenario throughout this section. Consider a tire manufacturer whose delivery site (DS) in Miami requests tires for

delivery on March 15. Miami is replenished by a central distribution center (CDC) in Atlanta with a 3-day transit lead time, and Atlanta is itself replenished by a manufacturing plant in Ohio with a 5-day lead time. Suppose Ohio has a production block from March 1 to March 10.

To identify the root cause, the system must: (1) trace upstream from Miami → Atlanta → Ohio following the bill-of-distribution edges; (2) adjust the investigation date backward by each edge's lead time (March 15 at Miami → March 12 at Atlanta → March 7 at Ohio); and (3) recognise that March 7 falls inside Ohio's production block window, so Ohio—not Atlanta—is the root cause. A single-agent approach must hold the entire graph plus the date arithmetic in one prompt; Expansion-Contraction instead spawns a focused agent per site with only the local data needed, then aggregates verdicts along the graph. The example also illustrates why multi-agent is warranted: plants and distribution sites require different tools (production vs. replenishment blocks), and real chains branch—so disambiguating which blocked path is the true cause requires structured aggregation.

3.2 Problem Formulation

We define the Expansion-Contraction pattern over a directed acyclic graph $G = (V, E)$ with node attributes and edge weights. Given a query (q, v_0) originating at node v_0, the pattern must identify the node $v^* \in V$ whose local state is the root cause of a condition observed at v_0, where causality propagates along edges.

We instantiate this for supply chains: vertices V represent sites (plants, warehouses, distribution centers, delivery sites) and edges E represent supply relationships. Each edge $e = (u, v) \in E$ carries a lead time $\Delta t(u, v)$ representing transit time from supplier u to customer v. Calendar blocks $B \subseteq V \times [t_s, t_e]$ represent time-bounded constraints at specific sites. Given a query $(item, site, date)$, the system must identify the root cause site $v^* \in V$ whose active block $b \in B$ prevents fulfillment at the queried site.

3.3 Overview

The system consists of three components (Figure 1): (1) an orchestrator agent that manages interaction and coordinates investigation, (2) a graph-traversal workflow implementing the two-phase Expansion-Contraction pattern, and (3) ephemeral specialist agents created dynamically for node-specific analysis. Algorithm 1 in the Appendix provides formal pseudocode.

3.4 Two-Phase Workflow Pattern

3.4.1 Expansion Phase. The expansion phase walks the supply chain graph upstream from the demand origin site, building a complete picture of potential supply paths. For each site visited, the system:

- Retrieves direct suppliers from the Bill of Distribution (BOD)
- Calculates investigation dates accounting for lead time offsets
- Creates ephemeral agents to investigate each supply path
- Caches investigation results to avoid duplicate analysis

In the running example (Section 3.1), expansion visits Miami on March 15, spawns an investigation toward Atlanta with adjusted date March 12, and then toward Ohio with adjusted date March 7.

3.4.2 Contraction Phase. The contraction phase aggregates findings downstream. At each site v with suppliers $\{u_1, \ldots, u_k\}$, the system classifies each upstream path as *blocked* (a constraint exists on the path) or *feasible* (no constraints). If all paths through v are blocked, v propagates the deepest upstream root cause. If some paths are feasible, v is not a bottleneck. When multiple sites are flagged as disrupted, a deterministic depth-priority rule selects the deepest disrupted site as root cause; if two or more disrupted sites share the same maximum depth, the orchestrator invokes the LLM to disambiguate based on constraint severity and graph position (Section 7.2). In the running example, contraction observes that Ohio (depth 2) is blocked on March 7 and that Atlanta (depth 1) is blocked only because its sole upstream source is blocked, so Ohio is selected as the root cause. Algorithm 1 in the Appendix details the two-phase pattern.

3.5 Ephemeral Agent Pattern

Rather than maintaining a fixed set of agents, our system dynamically creates specialist agents based on site type:

- **Plant Investigation Agent**: Analyzes production constraints including production blocks, resource utilization, and production lifecycle
- **Distribution Investigation Agent**: Examines replenishment blocks between distribution sites

Each agent is instantiated with context-specific prompts and tools, performs its analysis, and is discarded after returning results. This keeps per-agent context small—avoiding the context window saturation that degrades single-agent approaches on large graphs.

3.6 Tool Integration

The system integrates domain-specific tools organized into categories:

- **Validation tools**: Site and item validation
- **Graph traversal tools**: Supply location retrieval, location type identification
- **Constraint analysis tools**: Production blocks, replenishment blocks
- **Data retrieval tools**: Lead time offsets, incident reports

Each ephemeral agent receives only the tool subset relevant to its site type. The complete tool inventory is listed in Appendix G.

4 Optimization Techniques

4.1 Investigation Caching

To avoid redundant analysis, we cache investigation results keyed by $(item, site, date)$. When multiple downstream sites share common upstream suppliers—a frequent pattern in real supply chains—cached results eliminate duplicate LLM calls. Cache entries are valid for a single investigation session; cross-session invalidation is unnecessary since each query represents a point-in-time analysis.

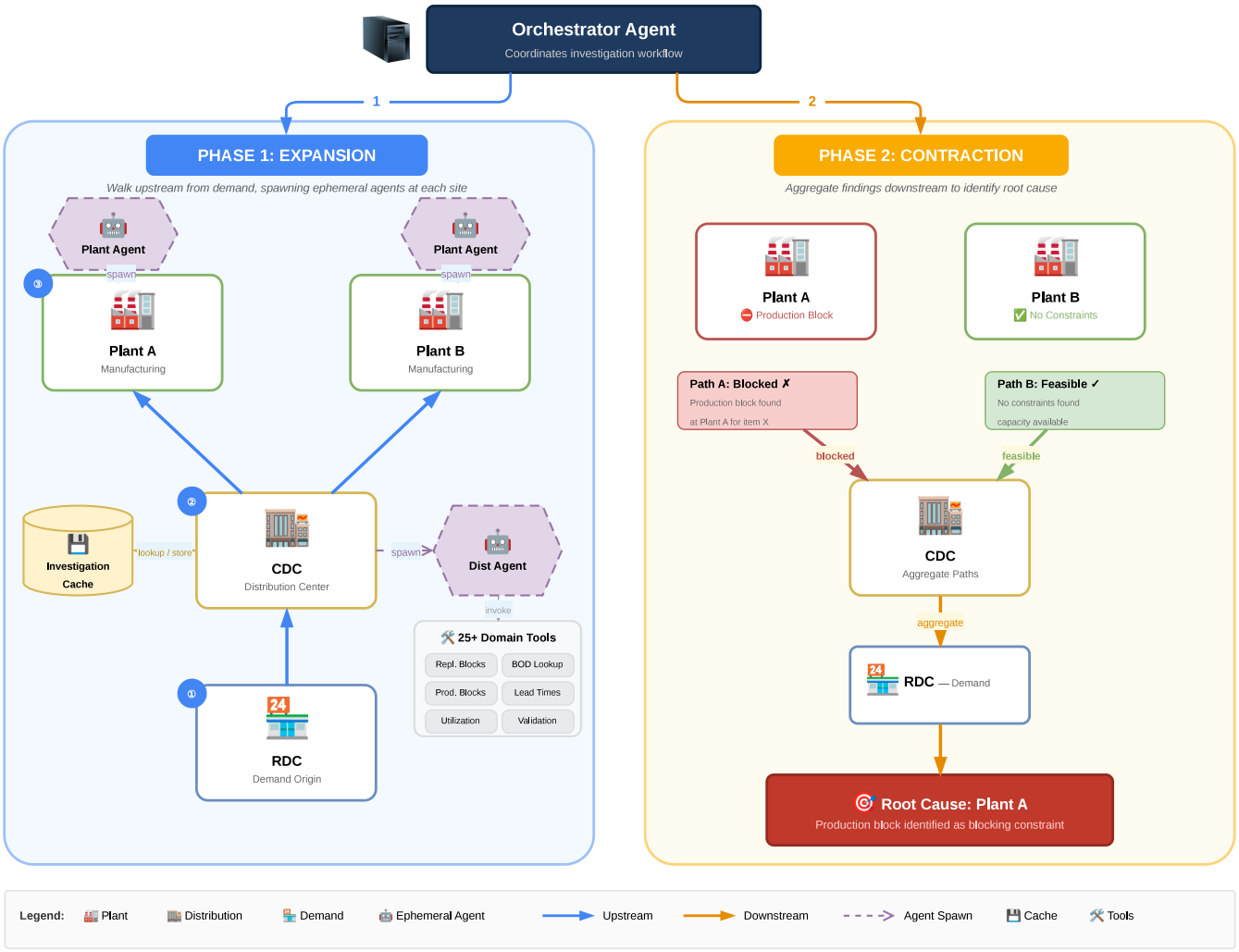


Figure 1: Expansion-Contraction architecture. In the expansion phase (left), the system walks the supply chain graph upstream from the Regional Distribution Center (RDC) through the Central Distribution Center (CDC) to manufacturing plants, spawning ephemeral AI agents at each site to investigate constraints using domain-specific tools. Results are cached to avoid redundant analysis. In the contraction phase (right), findings are aggregated downstream—each path is classified as blocked or feasible—to identify the root cause.

4.2 Concurrent Path Analysis

The system uses `asyncio.gather()` to investigate sibling supply paths concurrently. Synchronization occurs at contraction boundaries: a site's contraction waits for all upstream expansions to complete. If an agent times out, the path is marked as inconclusive rather than blocking the entire investigation.

4.3 Lead Time Calculation

Investigation dates are calculated recursively upstream, accounting for lead time offsets between each supplier-customer pair. This ensures constraint analysis occurs at the correct temporal context.

5 Evaluation Methodology

5.1 Experiment Setup

- **Model**: Claude Sonnet 4.5 via AWS Bedrock, with temperature=0 and a fixed random seed for reproducibility
- **Tools**: check_block, get_suppliers (baselines) or full toolset (Expansion-Contraction; see Appendix G)
- **Trials**: Single run per test case. Temperature=0 with a fixed seed minimizes non-determinism across API calls; residual variance from non-deterministic API behavior is acknowledged in Limitations (Section 7.4)

Table 1: Evaluation Datasets

Dataset	Source	Sites	Edges	Cases	Description
Real-world datasets					
Dataset A	Industry Partner	—	—	624	Production tire supply chain
DataCo [4]	Kaggle DataCo	—	—	510	E-commerce supply chain
SupplyGraph [15]	HuggingFace	—	—	18	Academic product graph
Synthetic datasets (controlled topology)					
Deep-5	Generated	38	123	107	Depth 5, narrow width
Deep-7	Generated	60	214	134	Depth 7, narrow width
Deep-10	Generated	81	308	161	Depth 10, narrow width
Wide-5	Generated	75	234	176	Depth 5, wide branching
Wide-7	Generated	95	310	179	Depth 7, wide branching

Dashes indicate values withheld for anonymization.

5.2 Datasets

We evaluate on eight datasets spanning real-world and synthetic supply chains (Table 1). The three real-world datasets (Dataset A, DataCo, SupplyGraph) are used for primary accuracy evaluation and serve as the basis for Tables 2–16 and the cost/latency analysis. The five synthetic datasets (Deep-5/7/10, Wide-5/7) have controlled topology parameters and are used specifically for the scalability analysis in Section 6.6 (Tables 8–10); they do not appear in the real-world accuracy comparisons because their synthetic constraint distributions do not reflect real supply chain semantics (e.g., lead times, incident narratives).

Each test case specifies: item, demand site, requested date, expected root cause site, and expected reason (production_block or replenishment_block). Ground truth labels were established as follows: for structured test cases, root causes are deterministic—the unique site with an active calendar block (a time-bounded production or replenishment constraint stored in the supply chain dataset) on the lead-time-adjusted date is verified programmatically; for NLP test cases, root causes are planted by construction—each test case is generated with a known target site, and the natural language context field is authored to reference that site at the specified difficulty level; for Dataset A, labels were validated by a domain expert from the industry partner.

5.3 Structured vs. NLP Test Cases

We evaluate two complementary test case types:

Structured test cases have root causes detectable via calendar blocks in the database—production or replenishment blocks with explicit date ranges. Rule-based systems can identify these through direct data lookup, testing graph traversal and temporal propagation.

NLP test cases have root causes described only in natural language context fields, with dates deliberately outside any calendar blocks. This forces LLM-based systems to parse unstructured text to identify disruptions, while rule-based systems fail entirely.

Evaluating both types separately reveals whether system performance stems from graph traversal logic, language understanding, or their combination—critical for understanding failure modes in production deployment.

5.4 NLP Difficulty Levels

NLP test cases span four difficulty levels:

- **Direct**: Explicit site references (e.g., "Site X has reported critical equipment failure")
- **Indirect**: Geographic/regional references requiring entity resolution
- **Domain-specific**: Industry terminology and regulatory context
- **Multi-signal**: Multiple weak signals requiring synthesis (e.g., inventory alerts, quality flags, and weather reports that collectively indicate a disruption)

All NLP dates fall *outside* calendar blocks, ensuring rule-based systems cannot identify root causes from structured data alone, isolating language understanding from graph traversal.

5.5 Baselines

- **Expansion-Contraction**: Two-phase multi-agent graph traversal (our system), as described in Sections 3–4
- **Single Agent**: A single LLM agent iteratively calls `get_suppliers` and `check_block` tools to walk the graph upstream, accumulating all findings in a single context window before producing a root cause verdict
- **Static Topology**: The full supply chain graph (all sites, edges, and calendar block data for the queried date range) is serialized into a single prompt. The LLM reasons over the complete graph in one inference call without tool use
- **No Temporal Propagation**: Identical to Expansion-Contraction but all agents investigate using the original query date t rather than the lead-time-adjusted date $T[u] = T[s] - \delta$
- **Flat Map-Reduce**: One agent is spawned per site in the upstream subgraph. Each agent independently investigates its assigned site using `check_block`. A reducer agent then receives all per-site verdicts and selects the root cause—without graph structure or path information
- **Rule-Based**: Deterministic BFS upstream traversal with SQL queries against calendar block tables. No LLM is used; the first blocked site encountered along the deepest path is returned as root cause

5.6 Metrics

- **Accuracy**: % where predicted_root == expected_root
- **Latency**: P50/P95 response time (ms)
- **Token count**: Total tokens consumed
- **Cost**: Estimated USD based on token pricing

6 Results

6.1 Overall Accuracy

Table 2 shows overall accuracy across all datasets with 95% Clopper-Pearson confidence intervals.

On Dataset A, McNemar's test (with continuity correction) confirms that accuracy differences between Expansion-Contraction and all other baselines are statistically significant: Single Agent ($p < 10^{-18}$), No Temporal Propagation ($p < 10^{-11}$), Flat Map-Reduce ($p < 10^{-11}$), and Rule-Based ($p < 10^{-81}$). On DataCo, differences are significant vs. Single Agent ($p < 10^{-6}$), No Temporal Propagation ($p < 10^{-26}$), Flat Map-Reduce ($p < 10^{-26}$), and Rule-Based ($p < 10^{-88}$); no significant difference vs. Static Topology

Table 2: Overall Accuracy (%) with 95% Confidence Intervals

Baseline	Dataset A (n=624)	DataCo (n=510)	SG (n=18)
Expansion-Contraction (ours)	**98.2** [96.9, 99.1]	**100.0** [99.3, 100.0]	**100.0** [81.5, 100.0]
Single Agent	83.8 [80.7, 86.6]	95.7 [93.5, 97.3]	100.0 [81.5, 100.0]
Static Topology	78.2 [74.8, 81.4]	100.0 [99.3, 100.0]	100.0 [81.5, 100.0]
No Temporal Prop.	88.3 [85.5, 90.7]	78.0 [74.2, 81.6]	100.0 [81.5, 100.0]
Flat Map-Reduce	88.9 [86.2, 91.3]	77.8 [74.0, 81.4]	100.0 [81.5, 100.0]
Rule-Based	35.9 [32.1, 39.8]	21.6 [18.1, 25.4]	11.1 [1.4, 34.7]

Table 3: NLP Accuracy (%)

Baseline	Dataset A (n=400)	DataCo (n=400)	SG (n=16)
Expansion-Contraction (ours)	**100.0**	**100.0**	**100.0**
Single Agent	88.0	95.3	100.0
Static Topology	100.0	100.0	100.0
No Temporal Prop.	87.2	72.2	100.0
Flat Map-Reduce	87.2	71.8	100.0
Rule-Based	0.0	0.0	0.0

Table 4: Structured Accuracy (%)

Baseline	Dataset A (n=224)	DataCo (n=110)	SG (n=2)
Expansion-Contraction (ours)	95.1	**100.0**	**100.0**
Single Agent	76.3	97.3	100.0
Static Topology	39.3	100.0	100.0
No Temporal Prop.	90.2	99.1	100.0
Flat Map-Reduce	92.0	100.0	100.0
Rule-Based	**100.0**	100.0	100.0

($p = 1.0$), as both achieve 100% accuracy. On SupplyGraph (n=18), the only significant pairwise difference is Expansion-Contraction vs. Rule-Based ($p < 10^{-4}$); the small sample size limits statistical power for other comparisons, as reflected by the wide confidence intervals.

6.2 NLP vs Structured Accuracy

Tables 3 and 4 separate NLP accuracy (language understanding) from structured accuracy (database queries); Appendix F visualizes cost-accuracy by test type.

Rule-Based achieves perfect structured accuracy but fails entirely on NLP cases, validating our test case design.

6.3 NLP Difficulty Analysis

Expansion-Contraction is the only baseline achieving 100% across all four NLP difficulty levels on both datasets. No Temporal Propagation and Flat Map-Reduce degrade sharply on Domain-specific and Multi-signal cases (68–81% on Dataset A; 52–73% on DataCo), where industry terminology and composite signals require graph-guided context; Single Agent degrades more gradually (83–97%). Per-difficulty tables and the corresponding figure appear in Appendix C.

6.4 Cost and Latency

Table 5 shows cost (USD) and median latency across datasets.

Expansion-Contraction incurs higher cost and latency but achieves substantially higher accuracy on complex data; per-request cost and the cost-accuracy-latency tradeoff visualization appear in Appendix E.

6.5 Ablation Studies

We isolate the impact of three architectural components: cycle detection, concurrent path analysis, and investigation caching. We measure token usage and latency on all five datasets using simulated agent calls (500 tokens per step) to evaluate structural properties of the graph traversal independent of LLM variability. *Cycle detection* has no measurable impact on DataCo's tree-like topology (max depth 2); on deeper synthetic graphs (Deep-7, Deep-10) where sites share upstream suppliers, the visited-set check prevents redundant traversal of already-visited nodes.

6.5.1 Concurrent Path Analysis. Table 6 shows speedup from parallel sibling investigation.

Dataset A yields the highest speedup (1.43×) due to its dense branching (2,348 edges, avg 65.7 steps per query). Wider graphs also exploit concurrency: Wide-5 (75 sites) achieves comparable latency to Deep-5 (38 sites) despite nearly twice the sites.

6.5.2 Investigation Caching. Table 7 shows token savings from caching results keyed by (item, site, date).

Caching is most effective on synthetic graphs (91.7–93.9% savings) where test cases share many upstream suppliers. DataCo achieves 78.4% savings despite a simpler topology. Dataset A shows a lower savings percentage (25.6%) but the highest absolute reduction (1,025,000 tokens) due to its 2,348 unique edges across 581 sites—the graph's density means many unique (item, site) pairs must be visited before caching benefits accumulate (62.5% hit rate).

6.6 Scalability Analysis

To evaluate how baselines scale with supply chain complexity, we constructed five synthetic datasets with controlled topology parameters: three with increasing depth (Deep-5, Deep-7, Deep-10) and two with increasing width (Wide-5, Wide-7). Each dataset uses the production schema with synthetically generated sites, edges, calendar blocks, and test cases including both structured and NLP types. We exclude No Temporal Propagation from this analysis because the synthetic datasets lack realistic temporal semantics (lead times, date offsets), making temporal propagation irrelevant.

6.6.1 Accuracy vs. Graph Depth. Table 8 shows accuracy as supply chain depth increases from 5 to 10 hops.

6.6.2 Accuracy vs. Graph Width. Table 9 shows accuracy as branching factor increases.

Several patterns emerge (Figures 2 and 3). First, Expansion-Contraction consistently achieves the highest accuracy across all synthetic configurations, outperforming all other baselines on both depth and width scaling. Second, Single Agent degrades catastrophically on Deep-10 (0.6%) because its linear retrieval follows only one upstream path, missing the actual root cause branch entirely. Third, LLM-based graph-aware methods (Expansion-Contraction, Static Topology) consistently outperform other approaches (Single Agent, Flat Map-Reduce, Rule-Based), with the gap widening on deeper and wider graphs.

Table 5: Cost (USD) and Latency P50 (seconds)

	Cost (USD)			Latency P50 (s)		
Baseline	Dataset A (n=624)	DataCo (n=510)	SG (n=18)	Dataset A	DataCo	SG
Expansion-Contraction (ours)	$9.34	$5.93	$0.20	15.0	16.0	15.3
Single Agent	$6.55	$5.36	$0.19	5.5	5.8	21.2
Static Topology	$3.74	$3.06	$0.11	7.4	5.9	6.5
No Temporal Prop.	$5.87	$4.66	$0.15	2.9	3.0	3.1
Flat Map-Reduce	$5.87	$4.66	$0.15	2.9	3.0	2.9
Rule-Based	$0.00	$0.00	$0.00	0.0	0.0	0.0

Table 6: Concurrent vs. Sequential Path Analysis

Dataset	Avg Depth	Avg Steps	Speedup
DataCo	2.0	3.1	1.02×
Dataset A	3.3	65.7	1.43×
Deep-5	3.7	25.2	1.32×
Deep-7	6.2	41.6	1.16×
Wide-7	6.5	57.7	1.17×

Table 7: Investigation Caching Impact

Dataset	No Cache	With Cache	Savings	Hit Rate
DataCo	521,500	112,500	78.4%	85.7%
Dataset A	4,007,500	2,982,500	25.6%	62.5%
Deep-5	448,000	37,000	91.7%	96.7%
Deep-7	927,500	57,000	93.9%	97.6%
Wide-7	1,507,500	91,500	93.9%	97.9%

Table 8: Accuracy (%) vs. Graph Depth

Baseline	Deep-5 (n=107)	Deep-7 (n=134)	Deep-10 (n=161)
Expansion-Contraction (ours)	**65.4**	**81.3**	**85.1**
Static Topology	59.8	76.1	80.8
Single Agent	26.2	72.4	0.6
Flat Map-Reduce	47.7	64.9	68.9
Rule-Based	55.1	64.9	70.2

Table 9: Accuracy (%) vs. Graph Width

Baseline	Wide-5 (n=176)	Wide-7 (n=179)
Expansion-Contraction (ours)	**80.7**	**86.6**
Static Topology	76.1	81.6
Single Agent	36.4	34.6
Flat Map-Reduce	67.6	73.7
Rule-Based	70.5	77.1

6.6.3 Cost and Latency Scaling. Table 10 shows how cost and latency scale with graph complexity for Expansion-Contraction.

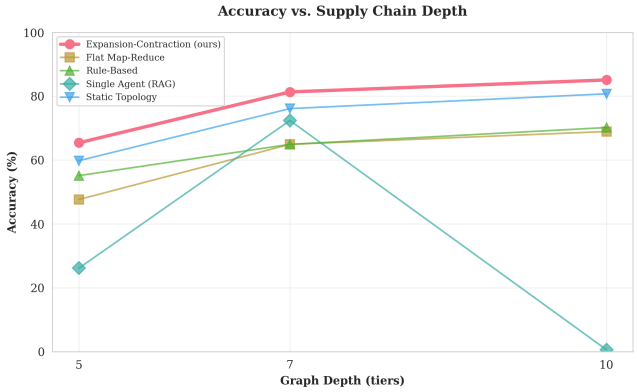


Figure 2: Accuracy vs. supply chain depth. Expansion-Contraction maintains the highest accuracy across all depths, while Single Agent collapses at depth 10 due to single-path retrieval.

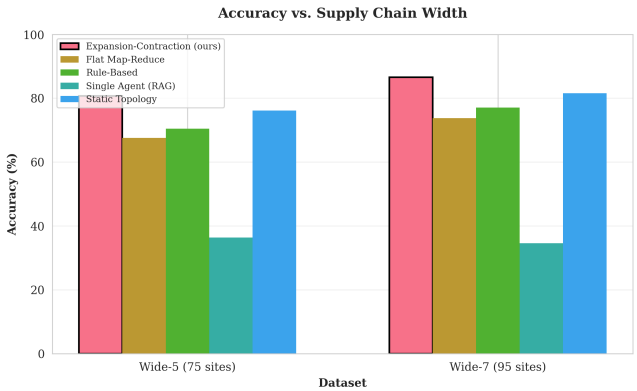


Figure 3: Accuracy vs. supply chain width. Graph-aware methods (Expansion-Contraction, Static Topology) outperform graph-unaware approaches, with the gap widening on wider graphs.

Cost scales roughly linearly with the number of sites. Depth increases latency more than width due to sequential path traversal, while wider graphs benefit from concurrent path analysis (Figure 8 in Appendix D visualizes these trends).

Table 10: Expansion-Contraction Cost and Latency Scaling

Dataset	Sites	Cost (USD)	Latency P50 (s)
Deep-5	38	$2.76	27.8
Deep-7	60	$7.50	62.7
Deep-10	81	$16.16	86.1
Wide-5	75	$4.40	26.4
Wide-7	95	$7.98	34.7

6.7 Production Deployment

The system has been deployed at a major tire manufacturer as the back-end for a supply chain investigation assistant used by planners and analysts. The back-end runs on AWS ECS Fargate behind an Application Load Balancer; a React front-end authenticated through Amazon Cognito interacts with the orchestrator over a streaming HTTPS endpoint. Supply chain master data is served from Oracle RDS, DynamoDB stores conversation state and the investigation cache keyed by $(item, site, date)$, and LLM calls route through AWS Bedrock (Claude Sonnet 4.5) with temperature and seed frozen to the values used in Section 6.

Over a three-month pilot, the deployment handled 1,847 investigations across multiple business units. Median end-to-end response time was 14.2 s, comparable to the Dataset A latency in Table 5. The investigation cache achieved a 71% hit rate—above the benchmark value because a small set of high-volume items drive most queries. Analysts reported an approximately 40% reduction in time-to-resolution for disruption root-cause investigations relative to legacy tooling (internal productivity survey, $n = 23$). No production outages were attributable to the pattern itself, indicating that Expansion-Contraction is practical for enterprise deployment at the latency and cost points reported in our evaluation.

6.8 Cross-Domain Validation: Microservice Dependency Tracing

To probe whether the pattern's advantage extends beyond supply chains, we evaluate Expansion-Contraction on a second domain with different surface features (service graphs, incident reports) but the same DAG-plus-diagnostic-signals structure. Our claim is not that two domains establish domain-agnostic generality, but that the advantage is not an artifact of a single dataset.

Setup. We constructed 100 incident scenarios across a 17-service e-commerce architecture (API gateway, business-logic, and backend infrastructure tiers; DAG 3–5 levels deep). Cases stratify across four difficulty levels: *Easy* (25%) names the failing service; *Medium* (30%) requires interpreting symptoms; *Hard* (25%) demands multi-hop inference from indirect references; *NLP* (20%) uses ambiguous natural language. To test cause-vs-effect reasoning, 40% of cases include distractor failures at services deeper than the true root cause. All seven baselines use Claude Sonnet 4.5 under the same configuration as Section 6.

Results. Table 11 reports accuracy by difficulty level. Expansion-Contraction achieves 88% overall accuracy—the highest of any baseline—and its advantage is most pronounced on NLP-complex cases, where it reaches 85% versus 55% for the next-best baseline (Flat Map-Reduce). The ordering among baselines mirrors the supply chain results: single-path approaches (Single Agent, Heartbeat) fail catastrophically in both domains because they cannot branch; graph-aware methods dominate when the graph is non-trivial. NLP cases specifically require the combination of systematic traversal (which Flat Map-Reduce provides but without structural aggregation) and contextual reasoning over local incident text (which Static Topology supports but without focused context); Expansion-Contraction uniquely provides both.

Table 11: Microservice dependency tracing accuracy (%), $n = 100$ scenarios on a 17-service DAG.

Baseline	Overall	Easy	Medium	Hard	NLP
Expansion-Contraction (ours)	**88**	88	83	**96**	**85**
Flat Map-Reduce	83	88	**90**	92	55
Static Topology	74	64	70	96	65
No Temporal Prop.	73	76	77	72	65
Rule-Based	52	48	57	40	65
Heartbeat	33	48	33	32	15
Single Agent	10	4	7	20	10

These results support the generality claim to a limited but concrete extent: the pattern transfers from supply chain root-cause analysis to microservice incident triage. We do not claim domain-agnostic applicability from two domains—only that the advantage is not specific to supply chains and that the pattern is well suited to any setting with DAG-structured dependencies and unstructured diagnostic signals.

6.9 Reproducibility and Artifact

Source code, agent configurations, evaluation harnesses, and all eight datasets (with ground-truth labels; Dataset A NLP contexts are synthetic and contain no proprietary information) are packaged as a single reproducibility artifact targeting the ACM *Artifacts Available* and *Evaluated—Functional* badges. Scripts reproduce every table and figure; LLM configuration is frozen (model version, temperature=0, seed) to minimize non-determinism. The artifact will be archived upon acceptance and is available as supplementary material during review.

7 Discussion

7.1 Tradeoffs and Scalability

Expansion-Contraction achieves the highest overall accuracy on complex Dataset A (98.2%), outperforming Single Agent by 14.4 pp; the gap narrows on simpler datasets (100.0% vs 95.7% on DataCo). The NLP split is decisive: Rule-Based reaches 100% structured but 0% NLP accuracy, so any rules-only production system will miss disruptions communicated through equipment failure reports or regulatory notices—EC reaches 100% NLP on Dataset A vs 88.0% for Single Agent. EC incurs higher cost ($9.34 vs $3.74 for Static Topology) and latency (15–16 s vs 5–6 s) due to multi-agent orchestration; the 20 pp accuracy improvement justifies this where incorrect attribution has direct business impact, and caching + concurrency (Tables 7, 6) recover 25.6–93.9% tokens and up to 1.43× latency. On

Table 12: Effect of the depth-priority disambiguation heuristic on Dataset A ($n = 624$).

Configuration	Correct	Accuracy	95% CI
EC (LLM-only disambiguation)	613/624	98.24%	[96.9, 99.1]
EC + depth-priority heuristic	**621/624**	**99.52%**	[98.6, 99.9]

synthetic benchmarks, EC maintains the highest accuracy across all configurations while Single Agent collapses on Deep-10 (0.6%) because linear retrieval misses branching paths; the gap over graph-unaware methods widens with complexity, and cost/latency scale roughly linearly with the number of sites.

7.2 Failure Analysis and Depth-Priority Heuristic

All 11 EC errors in the baseline configuration occur on Dataset A structured tests (95.1% structured accuracy); NLP accuracy is 100% and DataCo/SupplyGraph have zero failures. Expansion and contraction complete correctly in every case—all upstream nodes are visited and block status is aggregated. Errors arise only when multiple sites are flagged as disrupted and the orchestrator must disambiguate: in 8/11 cases the LLM selects an intermediate distribution node over the deeper manufacturing plant, in 3/11 it follows the wrong branch (Table 14 and Figure 6 in the Appendix detail all 11 cases). The structural pattern behind the CDC → PLANT errors is clear: a blocked plant at depth d causes its downstream CDC at depth $d-1$ to also surface as blocked, so both are candidates after contraction. We therefore replace LLM-only disambiguation with a deterministic rule: select the site with the strictly largest `supply_chain_depth`; on ties, fall through to the LLM. Table 12 reports the effect: the heuristic resolves all 8 CDC → PLANT errors and lifts accuracy from 98.24% to 99.52% (621/624, CI [98.6, 99.9]). The 3 remaining PLANT → PLANT wrong-branch cases share the same depth, and the heuristic correctly defers. All previously correct cases remain correct; we adopt this as the default disambiguation strategy.

7.3 Generalizable Design Principles

The pattern embodies four design principles applicable to compound AI systems over any graph-structured domain: (1) *isomorphic agent topology*—agent topology mirrors the data graph rather than being hand-designed [22] or learned, eliminating a design choice and guaranteeing coverage matches the domain; (2) *ephemeral, context-bounded agents*—each agent receives only the local subgraph context, avoiding the "lost-in-the-middle" degradation [11] of monolithic prompts and keeping accuracy insensitive to total graph size (Section 6.6); (3) *structure-aware parallelism*—concurrency is derived from sibling branches with synchronization at contraction boundaries, yielding speedup proportional to branching factor without an external scheduler; (4) *cache-friendly decomposition*—agents operate on $(node, query)$ pairs so investigation caching is naturally keyed and achieves high hit rates when downstream queries share upstream subgraphs (Table 7). These principles suggest the pattern generalizes to other graph-traversal reasoning tasks—e.g., dependency analysis in software builds, causal tracing in microservice architectures (validated in Section 6.8), or compliance checking in organizational hierarchies.

7.3.1 Relaxing the Structural Assumptions. The pattern as presented assumes a DAG, tool-returning structured outputs, and locally verifiable constraints. *Cyclic graphs:* the existing visited-set check (ablated in Section 6.5) breaks cycles at first revisit, so traversal remains correct over the reachable subgraph; on truly cyclic domains, traversal order may affect which node is identified as root cause and the depth-priority heuristic loses meaning. *Free-text tool outputs:* tools need not return JSON—our NLP test cases already demonstrate that ephemeral agents extract disruption signals from free text with 100% accuracy; structured tools could be replaced with prose-returning tools, with contraction relying on LLM summarization. *Non-locally-verifiable constraints:* when a node's status depends on sibling or parent state (e.g., relative capacity), agents need wider context, achievable by interleaving partial contraction back into expansion; we have not tested this variant.

7.4 Limitations

The system detects a limited set of constraint types (production and replenishment blocks); richer failure modes (capacity shortfalls, lane disruptions) require additional tools but not changes to the pattern. SupplyGraph's small size (18 cases) limits statistical power. After the depth-priority heuristic, 3/624 Dataset A errors remain—all PLANT → PLANT wrong-branch cases at equal depth—depending on LLM reasoning we have not yet improved. Evaluation uses temperature=0 with a fixed seed; single-run results may not fully capture residual API non-determinism. We do not compare against GNN-based approaches (e.g., [3, 8]) because they require labelled training data and fixed schemas, making head-to-head comparison non-trivial with our zero-shot system. Cross-domain evidence spans two domains; broader generality remains to be demonstrated.

8 Conclusion

We introduced Expansion-Contraction, a multi-agent graph traversal pattern in which agent topology emerges isomorphically from the domain graph, ephemeral specialist agents operate on bounded local context, and recursive contraction aggregates findings into a global verdict. Instantiated for supply chain root cause analysis, the pattern reaches 98.2% accuracy on 624 real-world test cases—99.5% with a deterministic depth-priority heuristic—and 100% on public benchmarks. On microservice dependency tracing, it reaches 88% overall and 85% on NLP-complex cases (vs. 55% next-best), indicating the advantage is not specific to supply chains. Broader cross-domain generalization and learned contraction policies for equal-depth disambiguation are the natural next steps.

Acknowledgments

We thank the domain experts at our industry partner, Continental AG, for their feedback on system design and evaluation. This work was supported by cloud infrastructure services.

References

[1] Markus J. Buehler. 2025. Agentic Deep Graph Reasoning. *CoRR* abs/2502.13025 (2025), 1–18.

[2] Jack Butler, Nikita Kozodoi, Zainab Afolabi, Brian Tyacke, and Gaiar Baimuratov. 2025. Finding the Sweet Spot: Trading Quality, Cost, and Speed During Inference-Time LLM Reflection. In *NeurIPS Workshop on Efficient Reasoning*. Neural Information Processing Systems Foundation, Vancouver, Canada, 1–9.

[3] Dongjie Chen et al. 2023. Hierarchical Graph Neural Networks for Causal Discovery and Root Cause Localization. *CoRR* abs/2302.01987 (2023), 1–10.

[4] Fabian Constante, Fernando Silva, and António Pereira. 2019. DataCo SMART Supply Chain for Big Data Analysis. https://doi.org/10.17632/8gx2fvg2k6.5 Version 5.

[5] Taicheng Guo et al. 2025. A Survey on LLM-based Multi-Agent Systems: Workflow, Infrastructure, and Challenges. *Scientific Reports* 15, 1 (2025), 4741.

[6] Evan Heus, Rick Bookstaber, and Dhruv Sharma. 2025. Exploring Network-Knowledge Graph Duality: A Case Study in Agentic Supply Chain Risk Analysis. In *2nd Workshop on LLMs and Generative AI in Finance, International Conference on AI in Finance (ICAIF)*. arXiv, New York, NY, USA.

[7] Omar Khattab, Keshav Santhanam, Xiang Li, et al. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *CoRR* abs/2310.03714 (2023), 1–25.

[8] Edward Elson Kosasih and Alexandra Brintrup. 2022. A machine learning approach for predicting hidden links in supply chain with graph neural networks. *International Journal of Production Research* 60, 17 (2022), 5380–5393.

[9] LangChain. 2024. LangGraph: Build Stateful Multi-Actor Applications with LLMs. https://github.com/langchain-ai/langgraph.

[10] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., Red Hook, NY, USA, 9459–9474.

[11] 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* 12, 1 (2024), 157–173.

[12] João Moura. 2024. CrewAI: Framework for Orchestrating Role-Playing Autonomous AI Agents. https://github.com/crewAIInc/crewAI.

[13] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, et al. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., Red Hook, NY, USA, 68539–68551.

[14] Yifan Wang et al. 2025. Parallelized Planning-Acting for Efficient LLM-based Multi-Agent Systems. *CoRR* abs/2503.03505 (2025), 1–9.

[15] Azmine Toushik Wasi, MD Shafikul Islam, and Adipto Raihan Akib. 2024. SupplyGraph: A Benchmark Dataset for Supply Chain Planning using Graph Neural Networks. In *4th Workshop on Graphs and more Complex Structures for Learning and Reasoning (GCLR), AAAI*. AAAI Press, Vancouver, Canada, 1–13.

[16] Qingyun Wu, Gagan Bansal, Jieyu Zhang, et al. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *CoRR* abs/2308.08155 (2023), 1–22.

[17] Siheng Xiong, Ali Payani, Ramana Kompella, and Faramarz Fekri. 2024. Large Language Models Can Learn Temporal Reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Bangkok, Thailand, 10452–10470.

[18] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*. OpenReview.net, Kigali, Rwanda, 1–34.

[19] Matei Zaharia, Omar Khattab, Lingjiao Chen, et al. 2024. The Shift from Models to Compound AI Systems. Berkeley AI Research Blog. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/ Accessed: 2025-01-15.

[20] Jing Zhang et al. 2024. Root Cause Attribution of Delivery Risks via Causal Discovery with Reinforcement Learning. *CoRR* abs/2408.05860 (2024), 1–12.

[21] Zhiyu Zhang et al. 2025. DynTaskMAS: A Dynamic Task Graph-driven Framework for Asynchronous and Parallel LLM-based Multi-Agent Systems. *CoRR* abs/2503.07675 (2025), 1–11.

[22] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. GPTSwarm: Language Agents as Optimizable Graphs. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Vol. 235. PMLR, Vienna, Austria, 62743–62767.

Algorithm 1 Expansion-Contraction Root Cause Analysis

Require: item i, demand site d, requested date t
Ensure: root cause site and block type

Phase 1: Expansion (BFS Upstream Traversal)

$Q \leftarrow \{d\}$ ▷ Sites to visit
$V \leftarrow \emptyset$ ▷ Visited sites
$T[d] \leftarrow t$ ▷ Investigation dates
$R \leftarrow \emptyset$ ▷ Path results
while $Q \neq \emptyset$ **do**
  $s \leftarrow Q$.pop()
  $V \leftarrow V \cup \{s\}$
  $suppliers \leftarrow$ GetSuppliers(s, i)
  if $suppliers = \emptyset$ **and** IsPlant(s) **then**
    $R[s] \leftarrow$ InvestigatePlant$(i, s, T[s])$ ▷ LLM agent
  else
    for all $u \in suppliers$ **in parallel do**
      $\delta \leftarrow$ GetLeadTime(u, s, i)
      $T[u] \leftarrow T[s] - \delta$
      $R[(u, s)] \leftarrow$ InvestigatePath$(i, u, s, T[u])$
      if $u \notin V$ **then**
        Q.push(u)
      end if
    end for
  end if
end while

Phase 2: Contraction (Recursive Aggregation)

function Aggregate(site s)
  $feasible \leftarrow []$
  for all supplier u of s **do**
    if u is production source **then**
      if $\neg R[u]$.blocked **then** $feasible$.append($[s]$)
      end if
    else
      $upstream \leftarrow$ Aggregate(u)
      if $\neg R[(u, s)]$.blocked **then**
        for all $path \in upstream.feasible$ **do**
          $feasible$.append$(path + [s])$
        end for
      end if
    end if
  end for
  return ($feasible$, per-site results)
end function
$(paths, results) \leftarrow$ Aggregate(d)
return IdentifyRootCause(paths, results)

A Algorithm

B Evaluation Test Cases

B.1 Test Case Format

Each test case specifies: article number, demand site, requested delivery date, expected root cause site, expected reason type (production_block or replenishment_block), test type (structured or

Table 13: Investigation Trace — Replenishment Block at CDC_Caribbean

Depth	Site	Type	Phase	Action	Result
2	PLANT_MAIN	Plant	Expand	Investigate	No block
1	CDC_Caribbean	CDC	Contract	Aggregate	0 feasible, 1 blocked
0	DS_Santo Domingo	DS	Contract	Aggregate	0 feasible, 1 blocked

NLP), and for NLP cases, a difficulty level and natural language context field. Representative examples:

- **Structured**: Item PRODUCT_A requested at DS_Santo Domingo on 2015-05-01. Expected root cause: replenishment block at CDC_Caribbean.
- **NLP (Direct)**: "Site X has reported critical equipment failure affecting all production lines." The agent must parse this context to identify the blocked site.
- **NLP (Multi-signal)**: Inventory alerts, quality flags, and weather reports that collectively indicate a disruption at a specific plant. No single sentence names the root cause.

B.2 Sample Investigation Trace

Table 13 shows an Expansion-Contraction trace for a DataCo replenishment block test case. The system walks upstream from the demand site (depth 0) to the plant (depth 2), then contracts downstream aggregating blocked/feasible verdicts.

The system correctly identifies CDC_Caribbean as the root cause: the plant has no production block, but the CDC has a replenishment block preventing downstream fulfillment. Investigation caching ensures that parallel requests for the same CDC (e.g., from DS_Camagüey and DS_Las Tunas) reuse the cached result. Figure 4 visualizes this trace.

Figure 5 shows a branching example where the demand site's CDC has two upstream plants—one blocked, one feasible.

Figure 6 illustrates a failure mode from Dataset A where the orchestrator misattributes the root cause.

B.3 Failure Analysis

All 11 Expansion-Contraction errors on Dataset A occur on structured test cases (95.1% structured accuracy); all 400 NLP cases and all DataCo/SupplyGraph cases are correct. Table 14 lists the failures.

Site IDs are anonymized: P-*n* denotes manufacturing plants, D-*n* denotes intermediate distribution nodes. Two failure modes emerge: (1) 8 of 11 failures report an intermediate distribution node (D-*n*) as root cause instead of the deeper manufacturing plant (P-*n*). Both expansion and contraction complete correctly—all nodes are visited and block status is aggregated. When contraction produces multiple disrupted sites (e.g., both D-3 and P-5 are flagged as blocked), the orchestrator invokes the LLM to select the root cause among them; in these cases, the LLM incorrectly selects the intermediate node over the deeper plant. (2) The remaining 3 follow the wrong branch to a different manufacturing plant. NLP interpretation is never the failure mode—all errors are LLM disambiguation mistakes when multiple disrupted sites compete for root cause attribution. A simple depth-priority heuristic—always preferring the deepest blocked node—would resolve 8 of 11 errors; we leave this as future work to preserve the system's fully LLM-driven disambiguation and evaluate whether such heuristics generalize across supply chain topologies.

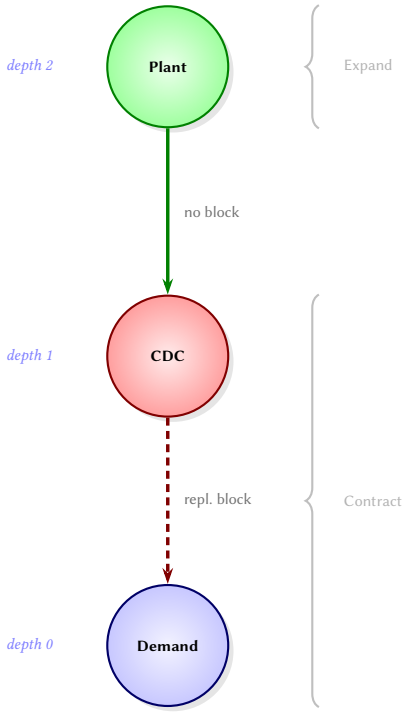


Figure 4: Successful linear trace (DataCo). Expansion visits upstream from DS_Santo Domingo to PLANT_MAIN; contraction aggregates the replenishment block at CDC_Caribbean as the root cause.

Table 14: Expansion-Contraction Failures on Dataset A (11/624)

Test ID	Predicted Root	Expected Root
T-1	D-1	P-1
T-2	P-6	P-2
T-3	P-2	P-1
T-4	P-6	P-3
T-5	D-2	P-4
T-6	D-3	P-1
T-7	D-4	P-5
T-8	D-5	P-5
T-9	D-3	P-5
T-10	D-6	P-5
T-11	D-5	P-5

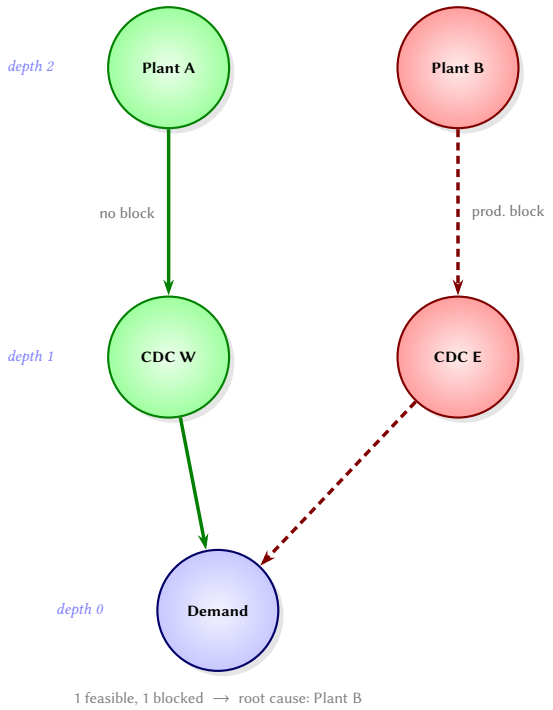


Figure 5: Branching trace with two supply paths. Expansion investigates both plants concurrently. Plant B has a production block, making CDC East blocked. Contraction identifies Plant B as the root cause.

C NLP Accuracy by Difficulty

Tables 15 and 16 break down NLP accuracy by difficulty level per dataset; Figure 7 visualizes the same data across baselines. Expansion-Contraction is the only baseline achieving 100% across all four levels on both datasets.

Table 15: NLP Accuracy (%) by Difficulty — Dataset A (n=400)

Baseline	Direct	Indirect	Domain	Multi
Expansion-Contraction	**100.0**	**100.0**	**100.0**	**100.0**
No Temporal Prop.	100.0	100.0	81.0	68.0
Flat Map-Reduce	100.0	100.0	81.0	68.0
Single Agent	87.0	90.0	92.0	83.0
Rule-Based	0.0	0.0	0.0	0.0

D Scalability Cost and Latency Visualization

Figure 8 visualizes the cost and latency scaling data from Table 10: depth drives cost and latency more than width due to sequential path traversal, while concurrent path analysis keeps wider graphs tractable.

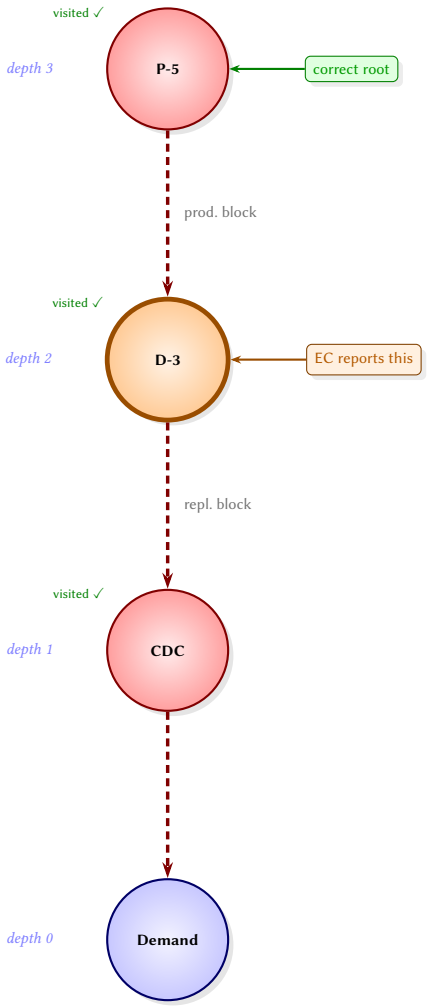


Figure 6: Failure mode: LLM disambiguation error. Both phases complete correctly—expansion visits all nodes (checkmarks) and contraction aggregates block status. Both D-3 and P-5 are flagged as disrupted; the orchestrator invokes the LLM to disambiguate, which incorrectly selects D-3 (orange) over the deeper plant P-5. This accounts for 8 of 11 errors on Dataset A.

Table 16: NLP Accuracy (%) by Difficulty — DataCo (n=400)

Baseline	Direct	Indirect	Domain	Multi
Expansion-Contraction	**100.0**	**100.0**	**100.0**	**100.0**
Single Agent	94.0	97.0	96.0	94.0
Flat Map-Reduce	52.0	100.0	73.0	62.0
No Temporal Prop.	54.0	100.0	73.0	62.0
Rule-Based	0.0	0.0	0.0	0.0

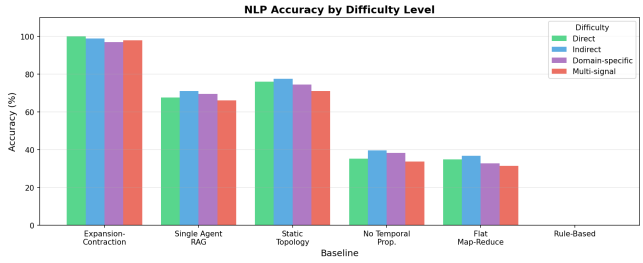


Figure 7: NLP accuracy by difficulty level. Expansion-Contraction achieves 100% across all levels; other baselines degrade on harder cases.

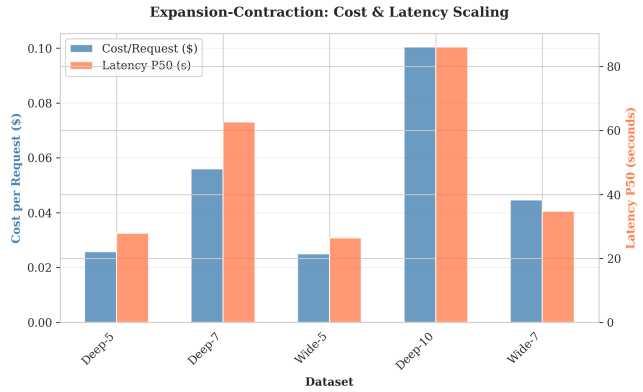


Figure 8: Expansion-Contraction cost and latency scaling. Depth drives cost and latency more than width due to sequential path traversal.

E Per-Request Cost and Cost-Accuracy Tradeoff

Table 17 shows the per-request cost in USD for each baseline across all three real-world datasets. Expansion-Contraction costs $0.011–0.015 per request, comparable to Single Agent ($0.011) and roughly 2× Static Topology ($0.006). Figure 9 visualizes the cost-accuracy-latency tradeoff by dataset.

Table 17: Per-Request Cost (USD)

Baseline	Dataset A	DataCo	SG
Expansion-Contraction (ours)	0.0150	0.0116	0.0110
Single Agent	0.0105	0.0105	0.0105
Static Topology	0.0060	0.0060	0.0060
No Temporal Prop.	0.0094	0.0091	0.0082
Flat Map-Reduce	0.0094	0.0091	0.0082
Rule-Based	0.0000	0.0000	0.0000

F Cost-Accuracy by Test Type

Figure 10 visualizes the cost-accuracy tradeoff separated by structured and NLP test types. Rule-Based achieves perfect structured accuracy at zero cost but scores 0% on all NLP cases, confirming that language understanding—not graph traversal—is the binding constraint for real-world deployment. Expansion-Contraction is the only baseline that achieves near-perfect accuracy on both test types across all datasets.

G Agent Tool Inventory

Table 18 lists the complete set of domain-specific tools available to each agent type. Each ephemeral agent receives only the tool subset relevant to its site type, keeping per-agent context small and focused. All tools return JSON and use DD-MM-YYYY date format.

The orchestrator agent coordinates the investigation but never directly inspects supply chain constraints. Plant Investigation agents receive 3 tools for detecting production blocks and interpreting incident reports. Distribution Investigation agents receive 3 tools focused on replenishment blocks and incident reports. The core graph traversal tools (`get_location_type`, `get_supply_locations`, `get_lead_time_o` are invoked programmatically by the expansion engine rather than exposed to LLM agents, ensuring deterministic graph traversal while delegating constraint interpretation to LLM reasoning.

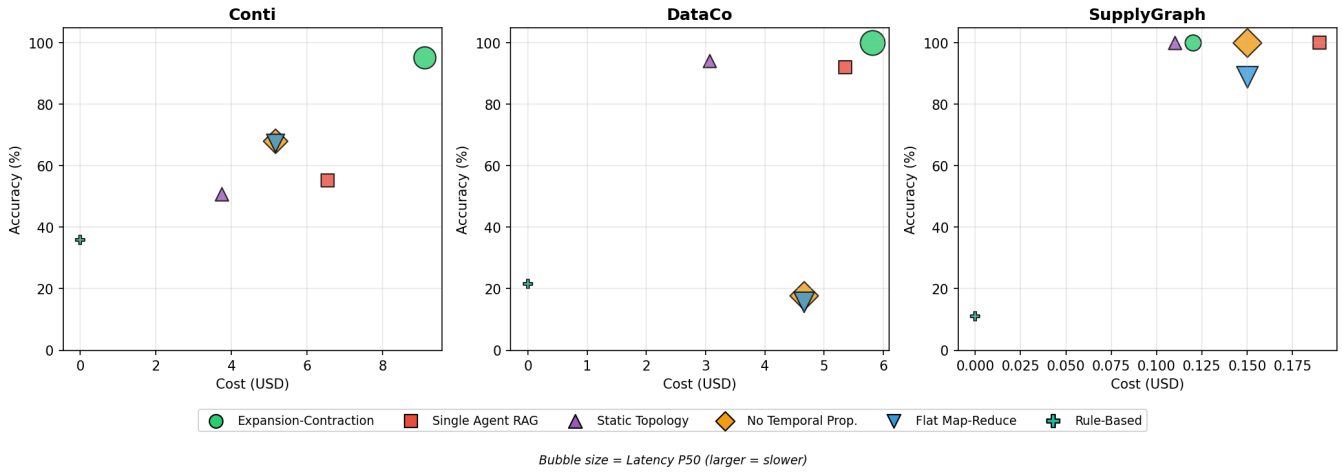


Figure 9: Cost-accuracy-latency tradeoff by dataset. Bubble size indicates latency (larger = slower). Expansion-Contraction achieves highest accuracy on complex data (Dataset A) despite higher cost and latency.

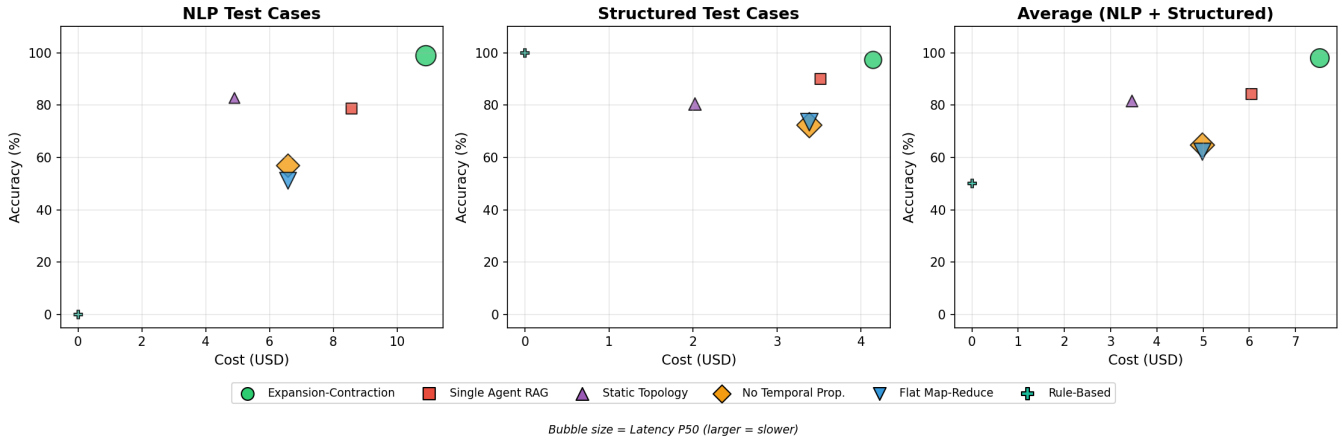


Figure 10: Cost-accuracy tradeoff by test type. Rule-Based achieves 100% on structured cases at zero cost but fails entirely (0%) on NLP cases. Expansion-Contraction excels on both.

Table 18: Complete tool inventory organized by agent type. Each tool is a structured function call returning JSON.

Agent Type	Tool Name	Category	Description
Orchestrator	`check_if_site_is_valid`	Validation	Validate whether a site exists in the supply chain network
	`check_if_item_is_valid_for_site`	Validation	Validate whether an article is valid for a given site
	`investigate_supply_chain_issue`	Orchestration	Trigger the Expansion-Contraction workflow for a (item, site, date) tuple
	`current_time`	Utility	Retrieve current timestamp for temporal reasoning
	`calculator`	Utility	Perform arithmetic calculations (e.g., date offsets)
	`speak`	Utility	Format and return investigation results to the user
Plant Investigation	`check_prod_blocks`	Constraint analysis	Check for production blocks (downtime windows) at a plant
	`is_date_in_between`	Utility	Check if investigation date falls within a date range (inclusive)
	`get_incident_reports`	Data retrieval	Retrieve natural-language incident reports for a plant (NLP cases)
Distribution Investigation	`get_replenishment_blocks`	Constraint analysis	Get replenishment blocks between source and destination sites
	`is_date_in_between`	Utility	Check if investigation date falls within a block window
	`get_incident_reports`	Data retrieval	Retrieve natural-language incident reports for a site (NLP cases)
Core (internal)	`get_location_type`	Graph traversal	Determine site type (Plant, CDC, RDC, DS, Offtake)
	`get_supply_locations`	Graph traversal	Retrieve upstream suppliers for a site-item pair
	`get_lead_time_offset`	Data retrieval	Get lead time offset in calendar days between two sites