

SEMANTIC GRAPH MODELLING: AGENTS THAT BUILD EXECUTABLE KNOWLEDGE GRAPHS FOR LONG-HORIZON WORK

Yanming Guo

The University of Sydney
AIDC

yguo0337@uni.sydney.edu.au

Haixin Wang

AIDC

nix@ai-dc.ai

Yanjun Lu

AIDC

lucas@ai-dc.ai

ABSTRACT

Language-model agents that work for long periods inside an organisation keep re-deriving the same knowledge from raw tables, policy documents and their own history. We study *semantic graph modelling*: before working, the agent compiles a domain’s sources and policies into an executable knowledge graph — typed objects and links mapped to the system of record, actions whose criteria encode the policies, functions, and automations that fire on change — and then works through a typed interface to it. We formalise the graph and the modelling operator, state what such a graph can and cannot save, and introduce OpsWEEK, a generator of simulated enterprise work weeks with a legacy ERP, policy prose, events and forty requests graded state-conditionally. With DeepSeek V4.1 Flash, the agent built an initial graph in 7.3 minutes for 0.82 CNY. Over six work weeks, request success was 99.5% with the graph and 97.7% for the same agent with SQL over the raw ERP; at this ceiling the difference is inconclusive. Long-context prompting (77.8%) and retrieval (24.1%), both single-call floors, were far behind. The graph used $1.66\times$ the tokens per request, in part because its modelling transcript was re-sent at every call (clearing the conversation halved its tokens and narrowed the gap to $1.27\times$), and did not repay its preparation. It changed how the work was done: automations met 78.6% of the standing-instruction obligations without a model call, and every write went through an engine-checked action. The raw agent, however, also compiled the policies, into a helper library of its own; what the graph added was enforcement and triggers rather than compiled knowledge.

1 INTRODUCTION

Language-model agents are moving from answering requests to doing work that lasts: they run an order desk, keep stock levels in check or reconcile accounts for days, through hundreds of model calls (Xu et al., 2025a; Backlund & Petersson, 2025). What makes such work hard is rarely a single step. It is that the knowledge each step needs — what a column means, how two tables join, which policy applies to a write, which events were already handled — is spread over raw tables, policy documents and the agent’s own history, and agents re-derive it from there again and again. Long contexts degrade (Liu et al., 2024a; Hsieh et al., 2024), retrieval returns fragments (Lewis et al., 2020), and an agent with database access recomputes the same joins at every request. In a production deployment that serves about three hundred agents for three companies, we observed all three effects¹: agents re-read raw ERP exports to answer recurring operational questions, one company’s agents ran some 1,800 scheduled jobs that re-checked data and were switched off for their cost, and long operational turns timed out midway.

Structured knowledge is the classical remedy. Knowledge graphs, ontology-based data access and semantic layers give data an explicit, typed meaning and let an engine, not the model, compute answers (Poggi et al., 2008; Hogan et al., 2021), and language models answer enterprise questions

¹Observed by the authors in their own deployment in September and October 2026; the counts are approximate.

more accurately through an ontology than through the raw schema (Sequeda et al., 2024). For long-horizon agents, however, these models fall short in three ways. They are *read* models: the rules that govern writes stay in the prompt. They are *passive*: nothing happens when the data change, so agents poll. And they are *built by others* — by people or by offline extraction pipelines (Zhang & Soh, 2024) — so their construction cost is never weighed against what they save at use time. Commercial data platforms offer operational ontologies that bind object types to data sources together with typed actions and automations. Among 475 candidate papers that we screened (Appendix H), the nearest work has an agent build a read-only ontology layer and evaluates it per question (Chong et al., 2026); we found no controlled study of whether an executable model helps a language-model agent over long, continuous work, or whether the agent can build one itself.

We study *semantic graph modelling* (SGM): before working, the agent compiles a domain’s sources and policies into an executable semantic graph — typed objects and links mapped to the sources, actions whose submission criteria encode the policies and whose effects write back to the system of record, functions for derived quantities, and automations that fire when data change — and it then works through a typed command-line interface to that graph. Three things can change: the context can hold views of the data rather than the data, the engine rather than the model checks policies on writes, and standing instructions become triggers rather than polling. We ask three questions. **RQ1**: on long, continuous work over evolving data, does operating on a semantic graph make an agent more accurate, cheaper and faster than long-context prompting, retrieval-augmented generation and a tool-using agent over the raw sources? **RQ2**: can the agent build the graph itself, how does a self-built graph compare with an expert-built one, and when does modelling pay for itself? **RQ3**: which parts of the graph carry the effect — the knowledge-graph part (typed objects, links, engine-side queries) or the executable part (actions, functions, automations)?

To answer them we built OpsWEEK, a generator of simulated work weeks of enterprise operations: a legacy ERP, policy prose, an event stream and forty requests per week, graded by a state-conditional oracle (Section 5). With DeepSeek V4.1 Flash, the answers differ from what the knowledge-graph view predicts. RQ1: the graph did not detectably change accuracy, since an agent with SQL over the raw ERP already succeeded on 97.7% of requests (99.5% with its own graph), and it did not save tokens: SQL already returns compact views, and the transcript of building the graph is re-sent with every later call ($1.66\times$ the tokens per request). RQ2: the agent built an initial graph by itself in 7.3 minutes for 0.82 CNY, but the raw agent also built a helper library of its own for the same policies, and the graph did not repay its preparation. RQ3: no rung of the ladder changed accuracy detectably; what the executable part changed was the form of the work, with writes through engine-checked actions and 78.6% of the standing-instruction obligations met by automations without a model call. Long-context prompting (77.8% at 1.003 CNY per request) and retrieval (24.1%) were far behind.

Contributions.

- A formulation of semantic graph modelling: an agent-built, executable knowledge graph $G = (\Sigma, \mu, D, \Lambda)$ with a modelling operator and a typed interface, and four propositions that state what such a graph can and cannot save (Section 3).
- OpsWEEK, a benchmark generator for long-horizon enterprise operations with evolving data, policy-constrained writes and standing instructions, graded state-conditionally, at three data scales (Section 5).
- A system that runs agents on a shared production runtime pool, isolates every execution, and connects them to an industrial ontology kernel through a typed command-line interface and a transactional outbox (Section 4).
- A study of seven conditions with an ablation ladder, six sessions for the main conditions and full cost accounting, which finds that a semantic graph changed how an agent writes and reacts rather than how accurately or cheaply it works, and that the raw agent compiled its own unenforced helpers for the same policies (Section 6).

2 RELATED WORK

Appendix H gives the full review behind this section.

Knowledge graphs and semantic layers for language models. Knowledge graphs give entities and relations an explicit, typed meaning (Hogan et al., 2021; Pan et al., 2024). Language models reason along graph paths (Sun et al., 2024) or retrieve over graph indexes built from text (Edge et al., 2024; Gutiérrez et al., 2024). Over relational data, ontology-based data access poses queries in domain terms (Poggi et al., 2008; Calvanese et al., 2016), and industrial semantic layers define measures and joins once for every consumer (dbt Labs, 2026; Google, 2026; Cube, 2026); text-to-SQL stays weak on enterprise schemas whose meaning lives in business knowledge (Lei et al., 2025; Chen et al., 2024). Querying a knowledge graph of an enterprise database instead of its schema raised GPT-4’s accuracy from 16% to 54% (self-reported; Sequeda et al., 2024), with similar gains reported for a semantic layer (Rumiantsev & Fokeev, 2026). ChatDB uses a database as an agent’s symbolic memory (Hu et al., 2023), and EvoOntology has a builder agent construct and refine an ontology layer for data agents (Chong et al., 2026). These are *read* models evaluated per question. We add the write path and triggers, let the working agent build them, and measure them over long, continuous sessions with full cost accounting.

Building and evaluating ontologies with language models. Language models extract entities, relations and schemas from text (Zhang & Soh, 2024; Bai et al., 2026) and help engineer ontologies (Lo et al., 2024; Zhang et al., 2025a); graphs are scored against references (Mihindukulasooriya et al., 2023), and ontology engineering evaluates a model by competency questions and pitfall catalogues rather than by one gold ontology (Grüninger & Fox, 1995; Poveda-Villalón et al., 2014). Our agent models structured sources together with policy prose, its model contains operations and rules besides classes and relations, and we judge it by the work it enables and by what it costs to build.

Context, memory and long-horizon agents. Long contexts degrade with length (Liu et al., 2024a; Hsieh et al., 2024; Hong et al., 2025), which motivates retrieval (Lewis et al., 2020; Li et al., 2024). Agent memories page, summarise or structure the agent’s own experience (Packer et al., 2023; Zhang et al., 2025c), also as temporal graphs (Rasmussen et al., 2025); for precise recall, agent-controlled search over the raw log can match structured memory (Li et al., 2026). Enterprise agent benchmarks test tool use over databases and applications (Styles et al., 2024; Yao et al., 2025; Xu et al., 2025a; Huang et al., 2026), long simulated runs derail (Backlund & Petersson, 2025), and cost is rarely reported with accuracy (Kapoor et al., 2024). A semantic graph models the domain’s state and rules rather than the agent’s experience, and OPSWEEK adds continuous sessions over evolving data with standing instructions.

Guards, compiled knowledge and automation. Agents act through tools and code (Yao et al., 2023; Wang et al., 2024b), write world models for a solver (Guan et al., 2023; Tang et al., 2024), and accumulate reusable skills (Wang et al., 2024a), also in a governed registry (Guo, 2026d). Policies can be compiled into guards that check tool calls at run time (Zwerdling et al., 2025; Wang et al., 2026a; Chen et al., 2025; Xiang et al., 2025; Rebedea et al., 2023; Moslemi et al., 2026). Business artifacts with guard–stage–milestone lifecycles, data-centric objects whose changes are guarded and driven by events, are the closest formal precedent (Nigam & Caswell, 2003; Hull et al., 2011); event–condition–action rules come from active databases (Widom & Ceri, 1995), and writing through a model to its sources is the view-update problem (Bancilhon & Spyrtos, 1981; De Giacomo et al., 2017), solved in practice by separating the write path from the read model (Fowler, 2011). A semantic graph combines these parts in one model that the agent itself defines, queries and changes through the same branch–validate–publish cycle.

3 SEMANTIC GRAPH MODELLING

3.1 SETTING: LONG-HORIZON WORK OVER A SYSTEM OF RECORD

An agent works for a long session in one domain. The domain has a system of record with relational sources $R = \{R_1, \dots, R_m\}$, written documents P (a data dictionary and policies), and a stream of external events that change R between requests (orders arrive, goods ship, invoices are paid). Work arrives as a sequence of dated requests $x_1, \dots, x_n$. A request asks for information (a *query*), asks for a change of R that must respect the policies (a *write*), or gives a standing instruction that applies to future events (a *rule*, e.g. “from now on, check every new EDI order on arrival”). We follow Guo (2026a): the agent is a harness $\mathcal{H}_{L,T}$ over a model L and tools T ; each request is one execution that

reads the agent’s record $d = (A, C \| s)$ — definition A , context C , workspace s — and commits a new record. A session is measured by the fraction of requests completed correctly, the fraction of rule-triggering events handled in time, and its cost: model calls, input and output tokens, and wall-clock time per request.

Without further structure, everything the agent learns about the domain — what a column means, how two tables join, which rule applies to a write, which events it already handled — lives in C or s and is re-read by the model whenever it is needed. Semantic graph modelling moves this knowledge into an external, typed and executable model that the engine, not the model, evaluates.

3.2 THE SEMANTIC GRAPH

Definition 1 (Semantic graph). *A semantic graph over sources R is $G = (\Sigma, \mu, D, \Lambda)$ with*

- *a schema $\Sigma = (O, E, \mathcal{A}, F, \Gamma)$: object types O , each with typed and described properties and one primary key; link types $E \subseteq O \times O$; action types $\mathcal{A}$, each a triple (Θ_a, c_a, ρ_a) of typed parameters, a submission criterion $c_a(\theta, D)$ that accepts or rejects with a reason, and an effect $\rho_a(\theta, D)$ that emits requests to the system of record (in our implementation c_a and ρ_a are one function rule that raises an error to reject); functions F , typed side-effect-free programs over object sets; automations Γ , each a pair $(\kappa_\gamma, \epsilon_\gamma)$ of a condition over object sets or change sets and an effect that applies an action, calls a function or sends a notification;*
- *a mapping μ that backs every object type with a source table or a read-only query and every property with a column or expression, as in ontology-based data access (Poggi et al., 2008); derived quantities are either such expressions or functions in F ;*
- *the data $D = \mu(R)$, a typed property graph that instantiates Σ ; it is re-synchronised from R after every action and every batch of external changes, and before every read when the agent can also write to R directly, so every read sees the current state of R ;*
- *a log Λ of every applied action with its parameters, actor and effects.*

The tuple (O, E, μ, D) is a knowledge graph in the usual sense (Hogan et al., 2021): typed entities and relations with meaning and provenance. Knowledge graphs, semantic layers and ontology-based data access are mostly used as *read* models. The second half, $(\mathcal{A}, F, \Gamma)$, makes the graph *executable*: it states how the domain may change and under which conditions, how derived quantities are computed, and what must happen when the data change. Operational ontologies in commercial data platforms have the same structure; we study it as a working substrate for language-model agents.

Interface. The agent touches G only through a typed command-line interface. A query $q \in Q$ (search with filters, traversal along links, aggregation, function call) returns a *view* $v = q(D)$, and only the view enters the context. An application $\text{apply}(a, \theta)$ commits if and only if $c_a(\theta, D)$ holds; its effect $\rho_a(\theta, D)$ is executed by the system of record, and D is re-synchronised from R . Automations run inside the engine: after every change set Δ of D , every γ whose condition $\kappa_\gamma(\Delta, D)$ holds applies ϵ_γ , without a model call.

Modelling. Semantic graph modelling (SGM) is the operator $\mathcal{M}_L : (R, P) \mapsto (\Sigma, \mu)$ executed by the agent itself, with the same model L that later uses the graph. The agent writes definitions on a branch of the schema, validates them (references resolve, keys are unique, mappings compile, actions are well typed) and publishes them, which re-synchronises D . Modelling is not a separate offline stage: whenever a request needs a concept that Σ lacks — typically an automation for a new standing instruction — the agent extends the schema through the same branch-validate-publish cycle.

Relation to previous objects. With a graph, the record becomes $d = (A, C \| s, G)$ with G external and shared. A policy gate of Guo (2026c), decided by a model for each request, becomes a criterion c_a that the engine checks; a region of a shared task space becomes an object set; the closure set of Guo (2026b), domain knowledge placed in context, becomes a schema whose instances are fetched on demand.

3.3 WHAT CHANGES FOR THE AGENT

Proofs are in Appendix A. The first statement locates where an agent’s input comes from; it predicts when a graph can save tokens and when it cannot.

Proposition 1 (Composition of the input). *For an agent that keeps its conversation, the input of a model call during request k is $|S| + |p| + \sum_{i < k} |h_i| + |o_k|$: system text S , the transcript p of its preparation (studying the sources or the graph, or building the graph), the history h_i of earlier requests and the tool outputs o_k of the current request so far. A graph changes o_k (views instead of raw rows), and through it the later history, and p . Hence a graph saves input tokens per call only through smaller tool outputs or a shorter preparation, and saves nothing on tool outputs against an agent that already computes views itself, for example with aggregate SQL. A policy that answers from data placed in its context needs $\Omega(n)$ input tokens for an aggregate to which n rows of D contribute, and a retriever that returns r chunks cannot answer exactly an aggregate over $m > r$ items stored in separate chunks.*

Proposition 2 (Compiled compliance). *If a policy $\pi(\theta, D)$ is implied by the acceptance of c_a , every committed application of π satisfies π . An agent that checks π in context with independent per-write error ε commits at least one violation in N writes with probability $1 - (1 - \varepsilon)^N$. If c_a is wrong on a set W of inputs, every write in W fails: compiled errors are systematic, as are the errors of any shared helper code.*

Proposition 3 (Event handling). *If a standing instruction is implemented by an automation whose condition and effect are correct, every triggering event is handled without a model call, so a request whose digest reports events costs the model no more calls than one without. An agent that handles events itself spends at least one model call on each digest that contains a trigger and misses each trigger at its per-event error rate.*

Proposition 4 (Break-even). *With preparation costs M_G and M_0 and per-request costs c_G and c_0 with and without the graph, cumulative costs cross after $n^* = (M_G - M_0)/(c_0 - c_G)$ requests if $c_0 > c_G$ and $M_G > M_0$, and never if $c_0 \leq c_G < \infty$ and $M_G > M_0$.*

Propositions 1 and 4 (an accounting identity) make the token benefit of a graph an empirical question about the composition of the context; Propositions 2 and 3 predict no policy errors on writes through correct actions and no extra model work for events. Section 6 measures each term.

4 SYSTEM

Figure 1 shows the three parts: a harness that runs agents on a shared runtime pool, a semantic-graph host that embeds an industrial ontology kernel, and the system of record.

Semantic-graph host. The host embeds the ontology kernel of a production data platform (version 0.4.1), which implements operational ontology services: schema branches and proposals, object sets with search, traversal and aggregation, actions with typed parameters, criteria and function-backed rules, isolated functions, and live and scheduled automations. The host adds three things (Appendix C). *Synchronisation*: object types are backed by ERP tables or read-only queries; after each event batch and each write the host applies the ERP change log to the mirrored, read-only objects and lets the automations evaluate the change set. *A transactional outbox*: an action never edits a mirrored object but emits `ErpRequest` objects, which the host executes in order as ERP transactions and synchronises back, so the ERP stays the single system of record. *Capability modes* for the ablation ladder. Each session has its own in-memory kernel.

Typed interface. The agent uses the graph through `sg`, a command-line client whose output the host renders and caps in size: read commands describe types, search, traverse, count and aggregate on the engine side and show edit history; write commands validate or apply an action; modelling commands define, map, validate and publish on a branch (Appendix B). Baseline agents use `erp` instead: read-only SQL on the live ERP and its transactions.

Harness on a shared runtime pool. Agents run on the harness of Guo (2026a): every request is one stateless execution of a general coding agent with a shell and an editor (Zechner, 2025), whose record (session log and workspace) is restored before and committed after it, and a metering proxy

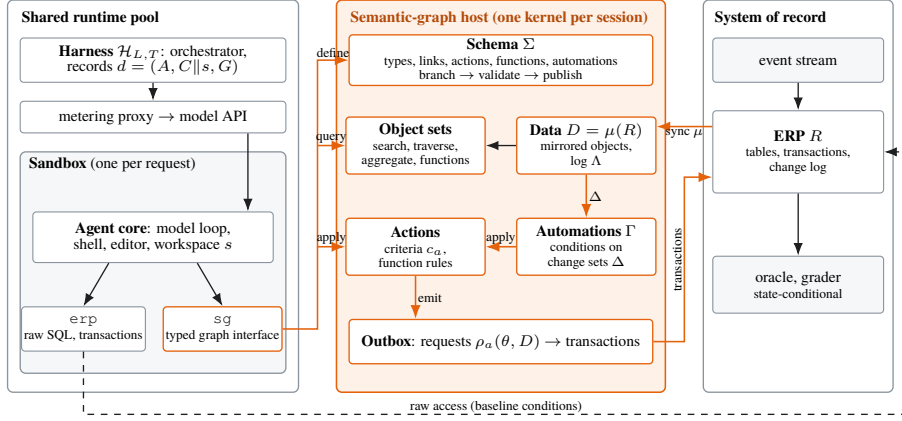


Figure 1: Architecture. The agent core runs in a fresh sandbox for every request and reaches the graph only through the typed `sg` interface (or, in baseline conditions, the raw `erp` interface). The host keeps one kernel per session: object sets answer queries, actions check their criteria and emit requests to an outbox, the outbox executes them as ERP transactions, and the resulting change set Δ is synchronised back through μ and evaluated by the automations. Orange: components introduced for semantic graph modelling.

records every model call and enforces a spending cap. We ran all experiments on a runtime pool that serves about 290 production agents, instead of on dedicated hardware; each execution runs in an isolated, resource-limited sandbox that sees only the language runtimes, the two clients and its own workspace (Appendix C).

5 OPSWEEK: A BENCHMARK OF CONTINUOUS ENTERPRISE OPERATIONS

Existing agent benchmarks for enterprise work either reset the world after every task (Styles et al., 2024; Yao et al., 2025) or evaluate a long task by its final outcome only (Xu et al., 2025a). Neither tests what SGM is for: many requests over one evolving system of record, under written policies, with standing instructions that apply to future events. OPSWEEK generates such sessions.² Every element is synthetic and seeded; the company is called Demo Company.

System of record. A legacy ERP with thirteen tables in SQLite (customers, materials, vendors, warehouses, inventory, sales orders and lines, shipments, invoices, purchase orders and lines, prices, notifications) plus a table of import junk, with ERP-style names (`SO_HDR.STAT`, `INV_BAL.QTY_RSV`) and coded values. Writes go through thirteen transactions (create a sales order, change its status or lines, allocate it, create a purchase order, change a price, set credit status, notify a team, ...) that enforce integrity but *not* business policy, as in real ERPs.

Documents. A data dictionary explains about 70% of the columns in terse ERP style. Six policy documents (credit, discounts and approvals, priority and allocation, replenishment, order changes, notifications) state the rules in business language, and some codes are explained only there (a *strategic account* is a Gold customer in two named regions). The rules interact: a credit hold takes precedence over an approval; an approval threshold applies to the discounted order value; available credit subtracts the open balance and the value of all open orders.

Events and sessions. A session is a simulated work week: a preparation request, then five days of eight requests. Before each day and after every second request an event batch changes the ERP: EDI orders arrive, released orders ship and are invoiced, purchase orders are received, invoices are paid, vendor costs and customer credit change. The generator guarantees that every day contains orders

²OPSWEEK is unrelated to OpsBench (Heyno, 2026) and Cloud-OpsBench (Wang et al., 2026c), which evaluate IT operations, and to the OpsWorld protocol (ColabHive Research, 2026).

that trigger the credit and approval rules and materials that fall below their reorder point. The agent sees a one-line digest of each batch (counts and identifiers) at the top of its next request.

Requests. Twenty-two templates in nine families (Appendix Table 6): lookups, multi-hop questions, aggregations over hundreds to thousands of rows, conditional analytics, policy-constrained writes (enter a phone order with a discount; change, cancel, approve and allocate orders), bulk updates, questions about the agent’s own earlier work, reports, and four standing instructions: replenish automatically when stock falls below the reorder point; check every new EDI order against the credit and approval rules on arrival; allocate priority-1 orders of strategic accounts as soon as they are open; and, on day four, lower the approval threshold from 50,000 to 40,000. Parameters are sampled on a reference trajectory so that every request is meaningful, and every reply ends with `ANSWER: <json>`.

Grading. Grading is *state-conditional*: an oracle computes the expected answer and the expected changes on the agent’s own pre-request state, so an early mistake is not counted again in every later answer. Writes are graded from the ERP change log of the request’s window by the net change of every row: every expected change must be present with the right fields, and no change may touch rows outside the request’s scope; free-text fields, the order of order lines and intermediate states that the agent corrects within the request do not matter. Numbers are accepted within 0.5% (counts exactly), sets must match exactly, and a code and its documented meaning are both accepted (Appendix G). Request success is computed over the 36 requests that are not standing instructions; an instruction itself only needs an acknowledgement and is graded on every later checkpoint (after each event batch and each request): each triggering event must have its effect by the end of the next request, and duplicates or effects without a trigger count as errors. A request whose premise the agent itself destroyed earlier (it cancelled the order it is now asked to approve) is marked *premise broken* and counted as failed; we report these separately as error propagation. A scripted agent that applies the oracle’s operations scores 100% on every generated session, and a second scripted agent that works only through the expert graph (Appendix E) solves the reference session; this checks that the generator, oracle and grader agree, not that every policy text is unambiguous to a reader.

Scales. Three scales vary the data while keeping the session fixed: S (24 customers, 60 materials, about 300 historical orders; 28k tokens of serialised data), M (120 customers, 400 materials, about 2,000 orders; 204k tokens) and L (600 customers, 2,000 materials, about 15,000 orders; 1454k tokens). The generator, the ERP server, the oracle and the grader are available from the authors.

6 EXPERIMENTS

6.1 SETUP

Conditions. Seven conditions share the model, the request texts, the event digests, the answer format and the ERP transactions; they differ in how the agent reads and writes (Table 1). *LLM* receives the full current ERP as CSV in every request; *RAG* receives the top 30 chunks of a hybrid retriever (BM25 and a dense encoder, Robertson & Zaragoza, 2009; Xiao et al., 2024, fused by reciprocal rank) over row, table and document chunks, re-indexed after every change. Both make one model call per request and are floors, not competitors with tools. *Agent* is a coding agent with a shell, a workspace that persists for the week and the raw `erp` interface (read-only SQL and transactions). *KG*, *+Actions* and *SGM-expert* give the same agent the expert graph with increasing capabilities and no SQL (RQ3); *SGM-self* gives it read-only SQL and the modelling commands, and no raw transactions, so every write it makes goes through an action it defined. Every agent condition starts with a preparation request under the same budget (120 model calls, 40 minutes): the raw agent prepares notes and helper scripts, the graph conditions study the graph, *SGM-self* builds its graph following a six-step recipe (Appendix D). The expert graph (Appendix E) was written by the authors from the policies and with knowledge of the request templates; it is an optimistic reference, not an upper bound, and *SGM-self* never sees it.

Design. The main study uses scale M and forty requests per session; each seed is a different company, and every condition run on a seed sees the same requests and events, so requests are paired by text (not by state, since grading is state-conditional). Agent, *SGM-expert* and *SGM-self* ran on 6

Table 1: Conditions. All use DeepSeek V4.1 Flash with low reasoning effort and the provider’s default sampling, and the same requests, digests and transactions.

Condition	Reads	Writes	Standing rules	State across requests
LLM	full ERP snapshot in prompt	transaction list	by the model	conversation
RAG	top-30 retrieved chunks	transaction list	by the model	conversation
Agent (L0, raw ERP)	SQL	raw transactions	by the agent	conversation + workspace
KG (L1)	graph (no SQL): search, traverse, aggregate	raw transactions	by the agent	conversation + workspace
+Actions (L2)	graph (no SQL)	actions	by the agent	conversation + workspace
SGM-expert (L3)	graph (no SQL) + functions	actions	automations or agent	conversation + workspace
SGM-self	graph it builds + SQL	actions it defines	automations or agent	conversation + workspace

Table 2: Main results at scale M. Success over the graded (non-instruction) requests, with 95% Wilson intervals; tokens, cost and time per request over all requests; SR recall: standing-rule obligations met in time (rescored, Appendix G); preparation: the first request of the agent conditions.

Condition	Seeds / graded	Success [95% CI] (%)	Tokens/request (k)	CNY/request	Wall time, median (s)	Model time, mean (s)	Calls/request	SR recall (%)	Preparation CNY
LLM (long context)	1 / 36	77.8 [61.9, 88.3]	446.4	1.003	20.4	41.3	1.00	63.3	0.00
RAG	3 / 108	24.1 [17.0, 32.9]	10.2	0.016	4.8	5.1	1.00	4.6	0.00
Agent (raw ERP)	6 / 216	97.7 [94.7, 99.0]	311.2	0.027	11.2	8.7	3.62	96.4	0.23
KG (L1)	3 / 108	100.0 [96.6, 100.0]	348.2	0.034	11.2	11.4	3.97	99.7	0.12
+Actions (L2)	3 / 108	98.1 [93.5, 99.5]	269.8	0.024	9.6	8.4	3.57	90.5	0.13
SGM-expert (L3)	6 / 215	99.1 [96.7, 99.7]	344.9	0.031	9.0	11.0	3.69	92.6	0.15
SGM-self	6 / 215	99.5 [97.4, 99.9]	514.4	0.037	9.9	10.7	3.21	95.7	0.82

seeds, RAG, KG and +Actions on three, and LLM, the most expensive condition, on one; contrasts use the seeds both conditions share. A request may use at most 30 model calls and 10 minutes. A request that exceeds the limit counts as failed with its cost, and the session goes on (3 executions: one preparation of KG and one request each of SGM-expert and SGM-self); requests hit by a documented infrastructure fault are excluded (Appendix G). Prompts and the interface help were tuned on a pilot session at scale S that is not part of the results.

Scoring and statistics. The primary outcome is request success. The three pre-planned contrasts are SGM-self versus Agent, SGM-expert versus Agent and SGM-self versus SGM-expert, tested with McNemar’s test on paired requests with Holm’s correction and with an exact sign-flip test on per-seed differences, because requests within a session are dependent. The scoring rules were corrected after the runs, for every condition alike: number formats and the documented meanings of codes are accepted, writes are compared by their final state, and standing-rule obligations are timed from their trigger (Appendix G); the contrasts are thus pre-planned but scored post hoc. Success intervals are Wilson intervals; differences and ratios are also given at the level of seeds. Costs are at list price (USD 0.30 per million uncached input tokens, 0.006 cached, 1.20 output; 7.2 CNY per USD) and with every input token at the uncached price. Speed is the median wall-clock time and the mean model time per request. A scaling study repeats the first twenty requests of one session at scale S (five conditions) and L (the raw agent and SGM-self), and a transfer study runs the raw agent and SGM-self on 60 WorkBench tasks (Styles et al., 2024).

6.2 RQ1: ACCURACY, TOKENS AND TIME

Accuracy. All agent conditions were near the ceiling (Table 2): SGM-self succeeded on 99.5% of requests, SGM-expert on 99.1%, the raw agent on 97.7%, the read-only graph and +Actions on 100.0 and 98.1%. SGM-self exceeded the raw agent by 1.9 percentage points (5 against 1 discordant requests; McNemar $p = 0.219$, Holm $p = 0.656$); at the level of seeds the difference is 1.8 points (95% CI $[-2.2, 5.8]$, sign-flip $p = 0.500$ over 6 seeds). The other two contrasts were not significant either (Holm $p = 0.906$ and 1.000). The result is inconclusive rather than null: at this ceiling the study can detect only large differences. The raw agent’s failures were all aggregations, conditional queries and reports (Appendix Table 7). The floors were far below: long-context prompting succeeded on 77.8% at 1.003 CNY per request, retrieval on 24.1%.

Tokens, cost and time. The graph did not save tokens. SGM-self used $1.66\times$ the raw agent’s tokens per request (seed level $1.58\times$, 95% CI $[0.99, 2.52]$, per seed 0.93–3.02) and SGM-expert $1.11\times$ (seed level $1.10\times$ $[0.89, 1.38]$). Because 99.7–99.8% of the agents’ input tokens were cache hits, SGM-self’s list cost was only $1.38\times$ (seed level $1.32\times$ $[0.88, 1.98]$), 0.010 CNY more per request; at the uncached price a request would cost 0.681 (Agent) and 1.121 CNY (SGM-self). The composition

of the input is consistent with Proposition 1 (Appendix Table 4): tool outputs are small in every agent condition (1203 characters per request for the raw agent, whose aggregate SQL already returns views), while every call re-sends the preparation transcript, 454k characters for SGM-self against 149k, so that its input per call was 159.6k tokens against 88.1k. Time did not differ consistently: SGM-self needed fewer calls (3.21 against 3.62) and a shorter median wall time (9.9 against 11.2 s), but $1.22\times$ the model time.

Standing rules and writes. Automations took over the standing-rule work in the graph conditions: SGM-self’s automations met 78.6% of its obligations, without a model call. Recall did not improve: the share of obligations met in time was 96.4% for the raw agent, 92.6% for SGM-expert and 95.7% for SGM-self (Appendix Table 5), and most misses concern one rule (allocate a strategic priority-one order as soon as it is open) and the orders already open when it was given, which agents read as inside or outside “from now on”. The digests push events to the agent, so no condition had to poll and immediacy was not rewarded. For the raw agent, a request whose digest reported events took 1.02 more model calls than a request without; the difference was -0.33 for SGM-expert and 0.24 for SGM-self (unadjusted for template), in the direction of Proposition 3. All agent conditions left a correct final state for almost every write request (Appendix Figure 2); the raw agent and the read-only graph, which also write raw transactions, made 0.33 and 0.67 transient policy violations per session (an order created with a status the policies forbid and corrected within the same request), the action conditions none. No request asked for a write that the policies forbid, so the criteria of Proposition 2 never had to reject one.

6.3 RQ2: SELF-BUILT GRAPHS

What the agent built. In its preparation request (7.3 minutes, range 4.7–11.3; 0.82 CNY, 0.59 more than the raw agent’s preparation) SGM-self built on average 14.7 object types, 13.8 link types, 15.3 actions, 18.2 functions and 4.2 automations. It mapped every source table that the expert graph maps but chose different designs: 48.3% of the expert’s mapped properties and 60.6% of its links have a counterpart (Appendix Table 8). It encoded the credit, discount, approval and allocation policies in the function rules of its actions, created automations for new EDI orders and for replenishment from the policy text alone, and kept modelling during the week (4.7 new model versions per session).

How it used the graph, and what the raw agent did instead. SGM-self wrote only through its own actions and let automations handle events, but it read with SQL: 75.9% of its read commands were raw SQL. The raw agent compiled the policies too, without a graph, as its preparation instructions suggested: in 5 of its 6 sessions it wrote a helper library during preparation (functions that evaluate an order against the credit and approval rules, process a new EDI order or select reorder candidates) and ran it 52 times per session. Both agents also acted on rules before being asked, because the policy text described them (28 effects for the raw agent, 16 for SGM-self). The difference between the two is therefore not whether the domain is compiled but how: as typed actions that the engine enforces and automations that it triggers, or as scripts that the agent remembers to run.

Cost and break-even. With the conversation kept, SGM-self cost more per request than the raw agent, so its extra preparation cost never paid back (Proposition 4). Because the extra input lies in the re-sent preparation transcript, we ran an exploratory variant, decided after seeing these results: the conversation is cleared after preparation, and the week starts with only the workspace and, for the graph conditions, the graph (seeds 1–3). Clearing cut SGM-self’s tokens per request to $0.53\times$ (95% CI [0.46, 0.62]) and the raw agent’s to $0.66\times$, with success unchanged (100.0 and 97.2%; Appendix Table 9). The gap between them shrank but did not close: cleared SGM-self still used $1.27\times$ [1.11, 1.46] the cleared raw agent’s tokens, with more calls (4.02 against 3.80) and a larger context per call, and its cumulative cost still did not fall below the raw agent’s.

6.4 RQ3: WHICH PART OF THE GRAPH MATTERS

The ladder rests on the expert graph and replaces SQL by graph reads; its last rung adds functions and automations together, so their effects cannot be separated. Query success was 96.7, 100.0, 98.7 and 99.3% for the raw agent, the read-only graph, +Actions and SGM-expert, and write success 100.0, 100.0, 97.0 and 98.5% (Appendix Figure 2); no rung differed from the raw agent beyond a

few requests. Costs were as uncertain: per request, the read-only graph cost $1.28\times$ the raw agent at the seed level (95% CI [0.99, 1.67]) and +Actions $0.92\times$ [0.64, 1.32], with as many calls per request as the raw agent (3.57 against 3.62). What the rungs changed was again the form of the work: from +Actions on, writes were applied through validated actions, and from SGM-expert on, automations met 74.1% of the standing-rule obligations.

6.5 SCALE AND TRANSFER

The raw agent’s cost per request did not grow with the data: over the first twenty requests of a session it used 261.3k tokens per request at scale S (28k tokens of data) and 241.5k at L (1454k), with 100.0 and 100.0% success. SGM-self succeeded on 100.0 and 100.0% with 406.0k and 884.1k tokens per request; its tokens followed its re-sent preparation transcript rather than the data, since its preparation at S exceeded the time limit while at L it reached 481k characters (Appendix Figure 4). Long-context prompting fell from 100.0% at S (0.158 CNY per request) to 77.8% at M and cannot hold the data of L. On WorkBench, where no policies exist and the data are reset before every task, the raw agent succeeded on 96.7% and SGM-self on 88.3% of 60 tasks, with 408.4k and 322.1k tokens and 28.0 and 11.4 s per task; SGM-self could also read the tables with SQL, which the raw agent’s WorkBench tools do not offer. Most of SGM-self’s failures were two-domain tasks in which it read “next Friday” as the coming Friday, a reading it kept from task to task.

7 DISCUSSION AND CONCLUSION

What the graph changed. In our setting the graph did not make the agent detectably more accurate, cheaper or faster. It changed the form of the work: writes went through actions that the engine checks, and standing instructions ran as automations without model calls. The raw agent compiled the same policies into a helper library that it ran itself, so the contribution specific to the graph is enforcement and triggers, not compiled knowledge. Proposition 1 says when a graph can save tokens: when tool outputs dominate the input, as for agents without a query language, and when the preparation transcript leaves the context (Section 6.3).

Self-modelling and governance. Agents of both kinds acted on rules that nobody had yet asked for, because the policy text described them; an organisation must decide whether agents may turn policy prose into executed rules, which a graph at least makes visible and versioned.

Representation or capability? Our ladder adds capabilities to one model of the domain; it does not separate the model from the capabilities. A raw agent with a policy-checked transaction wrapper (Zwerdling et al., 2025; Wang et al., 2026a) or with a trigger runtime over the database might obtain the same enforcement and event handling without a typed graph; our raw agent already wrote the policy code such a wrapper would run. Whether the typed model is what lets an agent author correct guards and triggers remains open.

Limitations. OPSWEEK is synthetic: one company type, clear policy texts, and an oracle that fixes one reading where several are possible. Sessions have forty requests, events are pushed in digests, no request must be refused, the day-four threshold change decides few orders, and the 0.5% tolerance can accept an aggregate that omits a small order; the benchmark under-exercises compiled criteria and rewards no immediacy. Requests are paired by text, not by state. The expert graph was written with knowledge of the request templates, so SGM-expert is an optimistic reference. We tested one inexpensive model at low reasoning effort; LLM and RAG are single-call floors. Inference rests on six sessions for the three main conditions, three for KG, +Actions and RAG, and one for LLM, so differences of a few points cannot be detected; scoring was corrected after the runs (Appendix G), and the cleared-conversation variant is exploratory. The ontology kernel is part of a commercial platform and is not released. These limits set the next steps: refusal requests, longer horizons and noisier data in OPSWEEK, and a comparison with guard and trigger runtimes that have no graph (Guo, 2026d).

Conclusion. An executable knowledge graph that an inexpensive model built in minutes did not detectably change accuracy and cost more tokens; it changed how a week of work was done, moving policy checks into engine-checked actions and standing instructions into automations.

REFERENCES

- Dean Allemang and Juan Sequeda. Increasing the Accuracy of LLM Question-Answering Systems with Ontologies. In *The Semantic Web – ISWC 2024*, pp. 324–339, 2024. doi: 10.1007/978-3-031-77847-6_18. URL https://link.springer.com/10.1007/978-3-031-77847-6_18.
- Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. AriGraph: Learning Knowledge Graph World Models with Episodic Memory for LLM Agents. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 12–20, 2025. doi: 10.24963/ijcai.2025/2. URL <https://www.ijcai.org/proceedings/2025/2>.
- Anthropic. Model Context Protocol Specification. <https://modelcontextprotocol.io>, 2024. Open protocol specification, accessed 2026-09-23.
- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://arxiv.org/abs/2310.11511>. arXiv:2310.11511.
- Hamed Babaei Giglou, Jennifer D’Souza, and Sören Auer. LLMs4OL: Large Language Models for Ontology Learning. In *The Semantic Web – ISWC 2023*, pp. 408–427, 2023. doi: 10.1007/978-3-031-47240-4_22. URL https://link.springer.com/10.1007/978-3-031-47240-4_22.
- Axel Backlund and Lukas Petersson. Vending-Bench: A Benchmark for Long-Term Coherence of Autonomous Agents. arXiv preprint arXiv:2502.15840, 2025. URL <https://arxiv.org/abs/2502.15840>.
- Jinheon Baek, Alham Aji, and Amir Saffari. Knowledge-Augmented Language Model Prompting for Zero-Shot Knowledge Graph Question Answering. In *Proceedings of the First Workshop on Matching From Unstructured and Structured Data (MATCHING 2023)*, pp. 70–98, 2023. doi: 10.18653/v1/2023.matching-1.7. URL <https://aclanthology.org/2023.matching-1.7>.
- Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, Feng Qin, Tianshi Zheng, Xi Peng, Xin Yao, Huiwen Yang, Leijie Wu, Ji Yi, Gong Zhang, Renhai Chen, and Yangqiu Song. AutoSchemaKG: Autonomous Knowledge Graph Construction through Dynamic Schema Induction from Web-Scale Corpora. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 20557–20584, 2026. doi: 10.18653/v1/2026.acl-long.942. URL <https://aclanthology.org/2026.acl-long.942>.
- François Bancilhon and Nicolas Spyrtas. Update Semantics of Relational Views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981. doi: 10.1145/319628.319634. URL <https://doi.org/10.1145/319628.319634>.
- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -Bench: Evaluating Conversational Agents in a Dual-Control Environment. arXiv preprint arXiv:2506.07982, 2025. URL <https://arxiv.org/abs/2506.07982>.
- Kamal Bhattacharya, Nathan S. Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y. Wu. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4):703–721, 2007. doi: 10.1147/SJ.464.0703. URL <https://doi.org/10.1147/SJ.464.0703>.
- Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. WorkArena++: Towards Compositional Planning and Reasoning-based Common Knowledge Work Tasks. In *Advances in Neural Information Processing Systems 37*, pp. 5996–6051, 2024. doi: 10.52202/079017-0195. URL <http://www.proceedings.com/079017-0195.html>.

- Pranav Bykampadi, Neel Mokaria, Vishesh Narayan, Faizan Wajid, and Ashok Agrawala. From Extraction to Governed Memory: Multi-Agent Knowledge Graph Construction with Domain-Expert Review. *arXiv preprint arXiv:2608.28642*, 2026. URL <https://arxiv.org/abs/2608.28642>.
- Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large Language Models as Tool Makers. In *International Conference on Learning Representations (ICLR)*, 2024. *arXiv:2305.17126*.
- Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2016. doi: 10.3233/sw-160217. URL <https://journals.sagepub.com/doi/full/10.3233/SW-160217>.
- Peter Baile Chen, Devin Yang, Weiyue Li, Fabian Wenz, Yi Zhang, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. BEAVER: An Enterprise Benchmark for Text-to-SQL. *arXiv preprint arXiv:2409.02038*, 2024. URL <https://arxiv.org/abs/2409.02038>.
- Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8313–8344. PMLR, 2025. URL <https://proceedings.mlr.press/v267/chen25ae.html>.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413*, 2025. URL <https://arxiv.org/abs/2504.19413>.
- Meiduo Chong, Shaolei Zhang, Ju Fan, and Xiaoyong Du. EvoOntology: A Self-Evolving Ontology Layer for Data Agents. *arXiv preprint arXiv:2609.15779*, 2026. URL <https://arxiv.org/abs/2609.15779>.
- ColabHive Research. Monolith, Hive, and Society: R6 Protocol v3. ColabHive Research protocol, September 2026, 2026. URL <https://colabhive.com/research/monolith-hive-society.html>. Design protocol with zero live R6 measurements; verified from indexed publisher page; accessed 2026-10-08.
- Cube. Cubes. Cube Documentation, 2026. URL <https://docs.cube.dev/reference/data-modeling/cube>. Undated living documentation; year denotes consulted version; accessed 2026-10-08.
- Das. R2RML: RDB to RDF Mapping Language. W3C Recommendation, 2012. URL <https://www.w3.org/TR/r2rml/>. Accessed 2026-10-08.
- Umeshwar Dayal and Philip A. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Transactions on Database Systems*, 7(3):381–416, 1982. doi: 10.1145/319732.319740. URL <https://doi.org/10.1145/319732.319740>.
- dbt Labs. dbt Semantic Layer. dbt Developer Hub documentation, 2026. URL <https://docs.getdbt.com/docs/use-dbt-semantic-layer/dbt-sl>. Last updated September 29, 2026; accessed 2026-10-08.
- Giuseppe De Giacomo, Domenico Lembo, Xavier Oriol, Domenico Fabio Savo, and Ernest Teniente. Practical Update Management in Ontology-Based Data Access. In *The Semantic Web – ISWC 2017*, volume 10587 of *Lecture Notes in Computer Science*, pp. 225–242. Springer, 2017. doi: 10.1007/978-3-319-68288-4_14. URL https://link.springer.com/chapter/10.1007/978-3-319-68288-4_14.
- Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How Capable Are Web Agents at Solving Common Knowledge Work Tasks? In *International Conference on Machine Learning (ICML)*, 2024. URL <https://arxiv.org/abs/2403.07718>. *arXiv:2403.07718*.

- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitan, Robert Osazuwa Ness, and Jonathan Larson. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv preprint arXiv:2404.16130, 2024. URL <https://arxiv.org/abs/2404.16130>.
- Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models. In *International Conference on Learning Representations*, pp. 24498–24525, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/3d7259031023c5aa463187c4a31c95c8-Abstract-Conference.html.
- Martin Fowler. Event Sourcing. MartinFowler.com, 12 December 2005, 2005. URL <https://martinfowler.com/eaDev/EventSourcing.html>. Accessed 2026-10-08.
- Martin Fowler. CQRS. MartinFowler.com, 14 July 2011, 2011. URL <https://martinfowler.com/bliki/CQRS.html>. Accessed 2026-10-08.
- Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024. doi: 10.14778/3641204.3641221. URL <https://dl.acm.org/doi/10.14778/3641204.3641221>.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10764–10799, 2023a. URL <https://proceedings.mlr.press/v202/gao23f.html>.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv preprint arXiv:2312.10997*, 2023b.
- Google. Introduction to LookML. Google Cloud Looker documentation, 2026. URL <https://docs.cloud.google.com/looker/docs/what-is-lookml>. Last updated October 6, 2026; accessed 2026-10-08.
- Benjamin Gruenbaum, Doron Porat, Assaf Natanzon, Roy Zavida, Chen Dinachi, and Or Itzhary. The Era by Eon Benchmark: A Generated Enterprise Estate with Exact Ground Truth for Benchmarking LLM Agents. arXiv preprint arXiv:2609.09853, 2026. URL <https://arxiv.org/abs/2609.09853>.
- Michael Grüninger and Mark S. Fox. Methodology for the Design and Evaluation of Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95*, Montreal, Canada, 1995. URL <https://www.eil.utoronto.ca/wp-content/uploads/enterprise-modelling/papers/gruninger-ijcai95.pdf>.
- Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *Advances in Neural Information Processing Systems 36*, pp. 79081–79094, 2023. doi: 10.52202/075280-3459. URL <http://www.proceedings.com/075280-3459.html>.
- Yanming Guo. AIDH: An Agent-Independent Distributed Harness for Persistent Agents. Manuscript, 2026a.
- Yanming Guo. Closure instead of training: Agentic domain modelling for life cycle assessment. Manuscript in preparation, The University of Sydney, 2026b.
- Yanming Guo. Neural orchestration: Spatio-temporal communication for large-scale multi-agent systems. Working paper, The University of Sydney, 2026c.
- Yanming Guo. Recursive Self-Improvement in Self-Organizing Multi-Agent Systems. Working paper, The University of Sydney, 2026d.

- Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and Fast Retrieval-Augmented Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10746–10761, 2025. doi: 10.18653/v1/2025.findings-emnlp.568. URL <https://aclanthology.org/2025.findings-emnlp.568>.
- Bernal Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. In *Advances in Neural Information Processing Systems 37*, pp. 59532–59569, 2024. doi: 10.52202/079017-1902. URL <http://www.proceedings.com/079017-1902.html>.
- Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to Memory: Non-Parametric Continual Learning for Large Language Models. In *International Conference on Machine Learning*, 2025. URL <https://arxiv.org/abs/2502.14802>.
- Kate Gwimm and Carson Eisenach. Beyond the Harness: End-to-End Optimization of Context Artifacts for Enterprise Text-to-SQL. arXiv preprint arXiv:2608.22830, 2026. URL <https://arxiv.org/abs/2608.22830>.
- Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann Lecun, Xavier Bresson, and Bryan Hooi. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. In *Advances in Neural Information Processing Systems 37*, pp. 132876–132907, 2024. doi: 10.52202/079017-4224. URL <http://www.proceedings.com/079017-4224.html>.
- Heyno. Introducing OpsBench. Heyno research announcement, 30 June 2026, 2026. URL <https://www.heyno.net/opsbench>. Official announcement; internal results and planned external availability; accessed 2026-10-08.
- Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D’amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge Graphs. *ACM Computing Surveys*, 54(4):1–37, 2021. doi: 10.1145/3447772. URL <https://dl.acm.org/doi/10.1145/3447772>.
- Kelly Hong, Anton Troynikov, and Jeff Huber. Context Rot: How Increasing Input Tokens Impacts LLM Performance. Technical report, Chroma, 2025. <https://research.trychroma.com/context-rot>.
- Cheng-Ping Hsieh, Simeng Sun, Samuel Krizan, Shantanu Acharya, Dima Rekeshe, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What’s the Real Context Size of Your Long-Context Language Models? In *Conference on Language Modeling*, 2024. URL <https://arxiv.org/abs/2404.06654>.
- Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory. arXiv preprint arXiv:2306.03901, 2023. URL <https://arxiv.org/abs/2306.03901>.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated Design of Agentic Systems. In *International Conference on Learning Representations (ICLR)*, 2025. URL <https://arxiv.org/abs/2408.08435>. arXiv:2408.08435.
- Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In *International Conference on Learning Representations*, pp. 156259–156291, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/fd1eff9dd295df50a41f2521942fa31d-Abstract-Conference.html.
- Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. CRM Arena: Understanding the Capacity of LLM Agents to Perform Professional CRM Tasks in Realistic Environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3830–3850, 2025. doi: 10.18653/v1/2025.naacl-long.194. URL <https://aclanthology.org/2025.naacl-long.194>.

- Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. CRMArena-Pro: Holistic Assessment of LLM Agents Across Diverse Business Scenarios and Interactions. *Transactions on Machine Learning Research*, 2026. URL <https://openreview.net/pdf/51fa206d5000500ead8a5b13b49b690c40616329.pdf>.
- Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno F. Terry Heath III, Stacy Hobson, Mark H. Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Noi Sukaviriya, and Roman Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proceedings of the 5th ACM International Conference on Distributed Event-Based Systems*, pp. 51–62, 2011. doi: 10.1145/2002259.2002270. URL <https://doi.org/10.1145/2002259.2002270>.
- Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. StructGPT: A General Framework for Large Language Model to Reason over Structured Data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9237–9251, 2023. doi: 10.18653/v1/2023.emnlp-main.574. URL <https://aclanthology.org/2023.emnlp-main.574>.
- Ziyan Jiang, Xueguang Ma, and Wenhui Chen. LongRAG: Enhancing Retrieval-Augmented Generation with Long-context LLMs. arXiv preprint arXiv:2406.15319, 2024. URL <https://arxiv.org/abs/2406.15319>.
- Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI Agents That Matter. arXiv preprint arXiv:2407.01502, 2024. URL <https://arxiv.org/abs/2407.01502>.
- Knublauch. Shapes Constraint Language (SHACL). W3C Recommendation, 20 July 2017, 2017. URL <https://www.w3.org/TR/2017/REC-shacl-20170720/>. Accessed 2026-10-08.
- Yassir Lairgi, Ludovic Moncla, Rémy Cazabet, Khalid Benabdeslem, and Pierre Cléau. iText2KG: Incremental Knowledge Graphs Construction Using Large Language Models. In *Web Information Systems Engineering – WISE 2024*, pp. 214–229, 2024. doi: 10.1007/978-981-96-0573-6_16. URL https://link.springer.com/10.1007/978-981-96-0573-6_16.
- Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *International Conference on Learning Representations*, 2025. URL <https://arxiv.org/abs/2411.07763>.
- Maurizio Lenzerini. Ontology-based data management. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 5–6, 2011. doi: 10.1145/2063576.2063582. URL <https://dl.acm.org/doi/10.1145/2063576.2063582>.
- 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. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL <https://arxiv.org/abs/2005.11401>. arXiv:2005.11401.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM Already Serve as A Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems 36*, pp. 42330–42357, 2023. doi: 10.52202/075280-1835. URL <http://www.proceedings.com/075280-1835.html>.
- Ruizhe Li, Licheng Zhang, Benfeng Xu, Mingxuan Du, Zheren Fu, and Weidong Chen. When Your Agent Opens the Chat App: Agent-Controlled Search over Raw Chat Logs Rivals Structured Memory. arXiv preprint arXiv:2608.12888, 2026. URL <https://arxiv.org/abs/2608.12888>.

- Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. StructRAG: Boosting Knowledge Intensive Reasoning of LLMs via Inference-time Hybrid Information Structurization. In *International Conference on Learning Representations*, pp. 36107–36124, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5975754c7650dfec0682e06elfec0522-Abstract-Conference.html.
- Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 881–893, 2024. doi: 10.18653/v1/2024.emnlp-industry.66. URL <https://aclanthology.org/2024.emnlp-industry.66>.
- Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023. doi: 10.1109/icra48891.2023.10160591. URL <https://ieeexplore.ieee.org/document/10160591/>.
- Lei Liang, Zhongpu Bo, Zhengke Gui, Zhongshu Zhu, Ling Zhong, Peilong Zhao, Mengshu Sun, Zhiqiang Zhang, Jun Zhou, Wenguang Chen, Wen Zhang, and Huajun Chen. KAG: Boosting LLMs in Professional Domains via Knowledge Augmented Generation. In *Companion Proceedings of the ACM on Web Conference 2025*, pp. 334–343, 2025. doi: 10.1145/3701716.3715240. URL <https://dl.acm.org/doi/10.1145/3701716.3715240>.
- Chengxi Liao, Tao Xu, Zulong Chen, Chuanfei Xu, Yiyan Wang, Xinyun Wang, Yanlong Zhang, Xiaojun Chen, Zhibo Yang, and Zeyi Wen. EntSQL: A Benchmark for Grounding Text-to-SQL in Long-Context Enterprise Knowledge. arXiv preprint arXiv:2606.03363, 2026. URL <https://arxiv.org/abs/2606.03363>.
- Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *arXiv preprint arXiv:2304.11477*, 2023. URL <https://arxiv.org/abs/2304.11477>.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. doi: 10.1162/tacl_a_00638. URL <https://arxiv.org/abs/2307.03172>. arXiv:2307.03172.
- Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, et al. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations (ICLR)*, 2024b. URL <https://arxiv.org/abs/2308.03688>. arXiv:2308.03688.
- Xiaoze Liu, Ruowang Zhang, Amir H. Abdi, Michel Galley, Zhikai Chen, Siheng Xiong, Xiaoqian Wang, and Jing Gao. Do Proactive Agents Need an LLM to Decide When to Act? arXiv preprint arXiv:2605.30152, 2026. URL <https://arxiv.org/abs/2605.30152>.
- Andy Lo, Albert Q. Jiang, Wenda Li, and Mateja Jamnik. End-to-End Ontology Learning with Large Language Models. In *Advances in Neural Information Processing Systems 37*, pp. 87184–87225, 2024. doi: 10.52202/079017-2767. URL <http://www.proceedings.com/079017-2767.html>.
- Yaxi Lu, Shenzi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Proactive Agent: Shifting LLM Agents from Reactive Responses to Active Assistance. In *International Conference on Learning Representations*, pp. 47431–47457, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/75c37811e830bf029584b1c6fac17726-Abstract-Conference.html.
- Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. In *International Conference on Learning Representations*, 2024. URL <https://arxiv.org/abs/2310.01061>.

- Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. MCP-Universe: Benchmarking Large Language Models with Real-World Model Context Protocol Servers. arXiv preprint arXiv:2508.14704, 2025. URL <https://arxiv.org/abs/2508.14704>.
- Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation. In *International Conference on Learning Representations*, pp. 52782–52806, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/830b1abc6d2da85f23d41169fa44d185-Abstract-Conference.html.
- Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating Very Long-Term Conversational Memory of LLM Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870, 2024. doi: 10.18653/v1/2024.acl-long.747. URL <https://aclanthology.org/2024.acl-long.747>.
- Costas Mavromatis and George Karypis. GNN-RAG: Graph Neural Retrieval for Efficient Large Language Model Reasoning on Knowledge Graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 16682–16699, 2025. doi: 10.18653/v1/2025.findings-acl.856. URL <https://aclanthology.org/2025.findings-acl.856>.
- Dennis McCarthy and Umeshwar Dayal. The architecture of an active database management system. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pp. 215–224, 1989. doi: 10.1145/67544.66946. URL <https://doi.org/10.1145/67544.66946>.
- Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants. In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://arxiv.org/abs/2311.12983>. arXiv:2311.12983.
- Nandana Mihindukulasooriya, Sanju Tiwari, Carlos F. Enguix, and Kusum Lata. Text2KGBench: A Benchmark for Ontology-Driven Knowledge Graph Generation from Text. In *The Semantic Web – ISWC 2023*, pp. 247–265, 2023. doi: 10.1007/978-3-031-47243-5_14. URL https://link.springer.com/10.1007/978-3-031-47243-5_14.
- Belinda Mo, Kyssen Yu, Joshua Kazdan, Proud Mpala, Lisa Yu, Charilaos Kanatsoulis, and Sanmi Koyejo. KGen: Extracting Knowledge Graphs from Plain Text with Language Models. In *Advances in Neural Information Processing Systems 38*, pp. 33962–33985, 2025. doi: 10.52202/085713-1010. URL <http://www.proceedings.com/085713-1010.html>.
- Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A. Rossi, Seunghyun Yoon, and Hinrich Schütze. NoLiMa: Long-Context Evaluation Beyond Literal Matching. In *International Conference on Machine Learning*, 2025. URL <https://arxiv.org/abs/2502.05167>.
- Zahra Moslemi, Keerthi Koneru, Yen-Ting Lee, Sheethal Kumar, and Ramesh Radhakrishnan. POLARIS: Typed Planning and Governed Execution for Agentic AI in Back-Office Automation. arXiv preprint arXiv:2601.11816, 2026. URL <https://arxiv.org/abs/2601.11816>.
- Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003. doi: 10.1147/SJ.423.0428. URL <https://doi.org/10.1147/SJ.423.0428>.
- Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs. *Communications of the ACM*, 62(8):36–43, 2019. doi: 10.1145/3331166. URL <https://dl.acm.org/doi/10.1145/3331166>.
- Abdulsohur Oyewale and Tommaso Soru. LLM-Driven Ontology Construction for Enterprise Knowledge Graphs. In *2026 International Conference on Semantic Computing (ICSC)*, pp. 267–270, 2026. doi: 10.1109/icsc67292.2026.00044. URL <https://ieeexplore.ieee.org/document/11486381/>.

- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023. URL <https://arxiv.org/abs/2310.08560>.
- Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599, 2024. doi: 10.1109/tkde.2024.3352100. URL <https://ieeexplore.ieee.org/document/10387715/>.
- Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2023. URL <https://arxiv.org/abs/2304.03442>. arXiv:2304.03442.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL <https://arxiv.org/abs/2305.15334>. arXiv:2305.15334.
- Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph Retrieval-Augmented Generation: A Survey. *ACM Transactions on Information Systems*, 44(2):1–52, 2025. doi: 10.1145/3777378. URL <https://dl.acm.org/doi/10.1145/3777378>.
- Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking Data to Ontologies. In *Journal on Data Semantics X*, pp. 133–173. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77688-8_5. URL http://link.springer.com/10.1007/978-3-540-77688-8_5.
- Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems 36*, pp. 36339–36348, 2023. doi: 10.52202/075280-1577. URL <http://www.proceedings.com/075280-1577.html>.
- María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. OOPS! (Ontology Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *International Journal on Semantic Web and Information Systems*, 10(2):7–34, 2014. doi: 10.4018/ijswis.2014040102. URL <https://doi.org/10.4018/ijswis.2014040102>.
- Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. CREATOR: Tool Creation for Disentangling Abstract and Concrete Reasoning of Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023. arXiv:2305.14318.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024. URL <https://arxiv.org/abs/2307.16789>. arXiv:2307.16789.
- Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A Temporal Knowledge Graph Architecture for Agent Memory. arXiv preprint arXiv:2501.13956, 2025. URL <https://arxiv.org/abs/2501.13956>.
- Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications with Programmable Rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 431–445, 2023. doi: 10.18653/v1/2023.emnlp-demo.40. URL <https://aclanthology.org/2023.emnlp-demo.40/>.
- Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2009. doi: 10.1561/15000000019. URL <https://www.emerald.com/ftinr/article/4/1-2/1/1326508/The-Probabilistic-Relevance-Framework-BM25-and>.

- Michael Rumiantsev and Ivan Fokeev. Semantic Layers for Reliable LLM-Powered Data Analytics: A Paired Benchmark of Accuracy and Hallucination Across Three Frontier Models. arXiv preprint arXiv:2604.25149, 2026. URL <https://arxiv.org/abs/2604.25149>.
- Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. In *International Conference on Learning Representations*, pp. 32628–32649, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8a2acd174940dbca361a6398a4f9df91-Abstract-Conference.html.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL <https://arxiv.org/abs/2302.04761>. arXiv:2302.04761.
- Juan Sequeda, Dean Allemang, and Bryon Jacob. A Benchmark to Understand the Role of Knowledge Graphs on Large Language Model’s Accuracy for Question Answering on Enterprise SQL Databases. In *Proceedings of the 7th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pp. 1–12, 2024. doi: 10.1145/3661304.3661901. URL <https://dl.acm.org/doi/10.1145/3661304.3661901>.
- Kartik Sharma, Peeyush Kumar, and Yunqing Li. OG-RAG: Ontology-grounded retrieval-augmented generation for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 32950–32969, 2025. doi: 10.18653/v1/2025.emnlp-main.1674. URL <https://aclanthology.org/2025.emnlp-main.1674>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. doi: 10.52202/075280-0377. URL <https://arxiv.org/abs/2303.11366>. arXiv:2303.11366.
- Gundeep Singh, Parsa Kavehzadeh, Jing Xia, Xue-Yong Fu, Julien Bouvier Tremblay, Md Tahmid Rahman Laskar, Vincent Lum, and Shashi Bhushan TN. Beyond Text-to-SQL: An Agentic LLM System for Governed Enterprise Analytics APIs. arXiv preprint arXiv:2605.21027, 2026. URL <https://arxiv.org/abs/2605.21027>.
- Ollie Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: a Benchmark Dataset for Agents in a Realistic Workplace Setting. arXiv preprint arXiv:2405.00823, 2024. URL <https://arxiv.org/abs/2405.00823>.
- Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive Architectures for Language Agents. *Transactions on Machine Learning Research*, 2024. URL <https://arxiv.org/abs/2309.02427>. arXiv:2309.02427.
- Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. In *International Conference on Learning Representations*, 2024. URL <https://arxiv.org/abs/2307.07697>.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv preprint arXiv:2405.16755, 2024. URL <https://arxiv.org/abs/2405.16755>.
- Hao Tang, Darren Key, and Kevin Ellis. WorldCoder, a Model-Based LLM Agent: Building World Models by Writing Code and Interacting with the Environment. In *Advances in Neural Information Processing Systems* 37, pp. 70148–70212, 2024. doi: 10.52202/079017-2243. URL <http://www.proceedings.com/079017-2243.html>.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL <https://arxiv.org/abs/2407.18901>. arXiv:2407.18901.

- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the International Conference on Computational Linguistics*, 2025a. URL <https://arxiv.org/abs/2312.11242>.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlkar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*, 2024a. URL <https://arxiv.org/abs/2305.16291>. arXiv:2305.16291.
- Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*, pp. 2938–2950, 2026a. doi: 10.1145/3744916.3764546. URL <https://doi.org/10.1145/3744916.3764546>.
- Ningning Wang, Xavier Hu, Pai Liu, He Zhu, Yue Hou, Heyuan Huang, Shengyu Zhang, Jian Yang, Jiaheng Liu, Ge Zhang, Changwang Zhang, Jun Wang, Yuchen Eleanor Jiang, and Wangchunshu Zhou. Efficient Agents: Building Effective Agents While Reducing Cost. arXiv preprint arXiv:2508.02694, 2025b. URL <https://arxiv.org/abs/2508.02694>.
- Ruiyang Wang, Hao-Lun Hsu, Swarajh Mehta, Jiwoo Kim, Zhihao Dou, and Miroslav Pajic. GAVEL: Graph World Models for Verified and Efficient Long-Horizon LLM Task Planning. arXiv preprint arXiv:2609.19315, 2026b. URL <https://arxiv.org/abs/2609.19315>.
- Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. OdysseyBench: Evaluating LLM Agents on Long-Horizon Complex Office Application Workflows. arXiv preprint arXiv:2508.09124, 2025c. URL <https://arxiv.org/abs/2508.09124>.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents. In *International Conference on Machine Learning (ICML)*, 2024b. URL <https://arxiv.org/abs/2402.01030>. arXiv:2402.01030.
- Yilun Wang, Guangba Yu, Haiyu Huang, Yujie Huang, Zirui Wang, Pengfei Chen, and Michael R. Lyu. Cloud-OpsBench: A Reproducible Benchmark for Agentic Root Cause Analysis in Cloud Systems. arXiv preprint arXiv:2603.00468, 2026c. URL <https://arxiv.org/abs/2603.00468>.
- Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, and Eugene Siow. MCP-Bench: Benchmarking Tool-Using LLM Agents with Complex Real-World Tasks via MCP Servers. In *International Conference on Learning Representations*, pp. 97377–97425, 2026d. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/9e4b14eb6f16fe7b5818a8d633a0606a-Abstract-Conference.html.
- Zhiruo Wang, Graham Neubig, and Daniel Fried. TroVE: Inducing Verifiable and Efficient Toolboxes for Solving Programmatic Tasks. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 51177–51191, 2024c. URL <https://proceedings.mlr.press/v235/wang24az.html>.
- Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. OfficeBench: Benchmarking Language Agents across Multiple Applications for Office Automation. arXiv preprint arXiv:2407.19056, 2024d. URL <https://arxiv.org/abs/2407.19056>.
- Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. In *International Conference on Learning Representations*, 2024e. URL <https://arxiv.org/abs/2401.04398>.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *International Conference on Machine Learning (ICML)*, 2025d. URL <https://arxiv.org/abs/2409.07429>. arXiv:2409.07429.

- Jennifer Widom and Stefano Ceri (eds.). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1995. URL <https://shop.elsevier.com/books/active-database-systems/widom/978-0-08-049856-0>.
- Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. In *International Conference on Learning Representations*, 2025. URL <https://arxiv.org/abs/2410.10813>.
- Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM Agents via Knowledge-Enabled Reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68316–68342. PMLR, 2025. URL <https://proceedings.mlr.press/v267/xiang25a.html>.
- Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed Resources For General Chinese Embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 641–649, 2024. doi: 10.1145/3626772.3657878. URL <https://dl.acm.org/doi/10.1145/3626772.3657878>.
- Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, et al. TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025a. URL <https://arxiv.org/abs/2412.14161>. arXiv:2412.14161.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents. *arXiv preprint arXiv:2502.12110*, 2025b. URL <https://arxiv.org/abs/2502.12110>.
- Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Ion Stoica, Joseph E. Gonzalez, Tianjun Zhang, and Shishir G. Patil. Berkeley Function-Calling Leaderboard. UC Berkeley Gorilla project, 2024. URL https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html. Accessed 2026-10-08.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*, 2023. URL <https://arxiv.org/abs/2210.03629>. arXiv:2210.03629.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *International Conference on Learning Representations*, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.
- Yining Ye, Xin Cong, Shizuo Tian, Jiannan Cao, Hao Wang, Yujia Qin, Yaxi Lu, Heyang Yu, Huadong Wang, Yankai Lin, Zhiyuan Liu, and Maosong Sun. ProAgent: From Robotic Process Automation to Agentic Process Automation. arXiv preprint arXiv:2311.10751, 2023. URL <https://arxiv.org/abs/2311.10751>.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018. doi: 10.18653/v1/d18-1425. URL <http://aclweb.org/anthology/D18-1425>.
- Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by Creating and Retrieving from Specialized Toolsets. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.17428.

- Mario Zechner. pi: a minimal agent harness and coding agent. <https://github.com/badlogic/pi-mono>, 2025.
- Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. FlowMind: Automatic Workflow Generation with LLMs. In *4th ACM International Conference on AI in Finance*, pp. 73–81, 2023. doi: 10.1145/3604237.3626908. URL <https://dl.acm.org/doi/10.1145/3604237.3626908>.
- Bohui Zhang, Valentina Anita Carriero, Katrin Schreiberhuber, Stefani Tsaneva, Lucía Sánchez González, Jongmo Kim, and Jacopo de Berardinis. OntoChat: A Framework for Conversational Ontology Engineering Using Language Models. In *The Semantic Web: ESWC 2024 Satellite Events*, pp. 102–121, 2025a. doi: 10.1007/978-3-031-78952-6_10. URL https://link.springer.com/10.1007/978-3-031-78952-6_10.
- Bowen Zhang and Harold Soh. Extract, Define, Canonicalize: An LLM-based Framework for Knowledge Graph Construction. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9820–9836, 2024. doi: 10.18653/v1/2024.emnlp-main.548. URL <https://aclanthology.org/2024.emnlp-main.548>.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, et al. AFlow: Automating Agentic Workflow Generation. In *International Conference on Learning Representations (ICLR)*, 2025b. URL <https://arxiv.org/abs/2410.10762>. arXiv:2410.10762.
- Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamunuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, et al. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. *arXiv preprint arXiv:2510.04618*, 2025c.
- Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A Survey on the Memory Mechanism of Large Language Model based Agents. *arXiv preprint arXiv:2404.13501*, 2024. URL <https://arxiv.org/abs/2404.13501>.
- Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv preprint arXiv:2504.07079*, 2025.
- Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. LLMs for knowledge graph construction and reasoning: recent capabilities and future opportunities. *World Wide Web*, 27(5), 2024. doi: 10.1007/s11280-024-01297-w. URL <https://link.springer.com/10.1007/s11280-024-01297-w>.
- Naama Zwerdling, David Boaz, Ella Rabinovich, Guy Uziel, David Amid, and Ateret Anaby Tavor. Towards Enforcing Company Policy Adherence in Agentic Workflows. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 595–606, 2025. doi: 10.18653/v1/2025.emnlp-industry.41. URL <https://aclanthology.org/2025.emnlp-industry.41/>.

A PROOFS

Proposition 1. The harness re-sends the conversation at every model call, so the input of a call is the system text, the preparation transcript, the earlier requests with their replies and tool outputs, and the current request with its tool outputs so far; this is the stated sum. The graph enters only through the commands the agent runs: their outputs are part of o_k (and of h_i once the request is over), and studying or building the graph is part of p . Replacing raw rows by views changes the size of tool outputs, and of the history that contains them, and nothing else; if the agent without a graph already writes queries whose results are views (aggregates, filtered rows), the tool-output term is of the same order in both cases. For the long-context policy, consider an aggregate $\sum_{x \in X} f(x)$ over the set $X \subseteq D$ of the n rows that contribute to it (e.g. the value of all open orders). Any policy that answers from its context must have every $x \in X$ in its context, because changing one omitted x changes the

true answer but not the context; hence the context contains $\Omega(n)$ tokens for such aggregates. For the retriever, if the m items that contribute to the aggregate are stored in m distinct chunks and the retriever returns $r < m$ chunks, at least one contributing item is missing from the context, and the same argument shows that the answer cannot be exact for all instances. $\square$

Proposition 2. The engine commits an application of a only if $c_a(\theta, D)$ holds, and $c_a \Rightarrow \pi$, so every committed application satisfies π . For the in-context policy, each of the N writes violates π independently with probability ε , so the probability of no violation is $(1 - \varepsilon)^N$. If c_a is wrong on a set W — it accepts some $(\theta, D) \in W$ with $\neg\pi(\theta, D)$ or rejects some with $\pi(\theta, D)$ — the engine’s decision on every input of W is the same deterministic wrong decision, so the errors are perfectly correlated across writes in W . $\square$

Proposition 3. The automation service evaluates κ_γ on every change set and applies ϵ_γ when it holds; no model is involved, so the handling cost in model calls is zero, a correct condition and effect handle every trigger, and the request that follows sees the events already handled. An agent that learns of events only from digests must act on them in a model call of the following request, after reading the digest; each trigger is acted upon only if the model recognises it, which happens with probability $1 - \varepsilon_e$. $\square$

Proposition 4. Cumulative cost after n requests is $M_0 + nc_0$ without and $M_G + nc_G$ with the graph; they are equal at $n^* = (M_G - M_0)/(c_0 - c_G)$, which is positive and finite if $c_0 > c_G$ and $M_G > M_0$; if $c_G \geq c_0$ and $M_G > M_0$ the graph’s cumulative cost stays above. $\square$

B THE TYPED GRAPH INTERFACE

Table 3 lists the commands of the graph interface. Output is rendered by the host as aligned text (at most 50 rows by default) or JSON; errors name the offending item and suggest one fix. Filters use a Mongo-style JSON ($\$eq, \$gt, \$in, \$contains, \$and, \$or, \dots$) translated to the kernel’s object-set filters; aggregation and traversal run in the engine. In the modelling mode the graph starts with one platform type (`ErpRequest`, the outbox) and the help text gives a six-step workflow, templates for every definition kind and a runnable example on two inventory tables that encode no policy.

Table 3: Commands of the typed graph interface and the modes that expose them (L1 read-only graph, L2 + actions, L3 + functions and automations, SELF + modelling).

Group	Commands	Modes
Read	<code>types, describe T, objects T --where --select --order-by, object T pk, links T pk L, traverse T --via L1, L2 --target-where, count T, aggregate T --metrics --group-by, history T</code>	all
Write	<code>actions, action A, apply A --param k=v [--validate-only], tx-catalog</code>	L2, L3, SELF
Compute and react	<code>functions, call F, automation list[show create update delete runs</code>	L3, SELF
Model	<code>template K, define FILE, map FILE, function publish FILE, validate, publish, model export</code>	SELF

C SYSTEM DETAILS

Host. Every object type is backed by an ERP table or a read-only query, with value transforms for codes. After each event batch and each write, the host reads the ERP change log, applies the changes to the mirrored objects as datasource edits, and asks the automation service to evaluate the change set; in the read-only mode, whose agent writes with raw transactions, it also pulls the log before every read. An action’s rule or function emits `ErpRequest` objects; the host executes them in order as ERP transactions under the action’s name, so a failed transaction fails the application with the ERP’s message, and synchronises the result back — the write path of production deployments. The capability modes expose read commands only, reads and actions, everything except modelling, or everything.

Sandbox and pool. Each execution runs in a transient system unit as an unprivileged user. It sees only the language runtimes, the two command-line clients and its own workspace (not the benchmark, the oracle, the reference graph, other sessions or other tenants), has no network beyond local

sockets to the proxy and the two interfaces, and is limited in memory and processes. All research processes share one cgroup of 2.5 GB and three cores, and a watchdog pauses the study when the pool’s free memory falls below 5 GB. Every execution sees its workspace at the same path, so consecutive requests of a session share a prompt prefix and the provider’s prefix cache applies (Appendix G). Long-context and retrieval baselines call the model through the same proxy.

D PREPARATION INSTRUCTIONS

Every agent condition received one of three preparation requests, each closed by “Reply READY when you are prepared” (SGM-self: “when the model is published”).

Raw agent. The week starts soon. Prepare yourself for the job; do not change the ERP. (1) Study the ERP (`erp tables`, `erp schema <TABLE>`, `erp sql`) and the documents in `docs/`: what each table and column means, the codes, how tables join, and every policy rule. (2) Write what you will need later into your workspace so that requests are fast and correct, for example `NOTES.md` (meanings, codes, joins, a checklist per policy) and helper scripts for computations you expect to repeat (order values, available credit, available stock, reorder needs, warehouse allocation). (3) Check your scripts against a few rows with `erp sql`.

Graph conditions (KG, +Actions, SGM-expert). The week starts soon. Prepare yourself for the job; do not change the ERP. (1) Study the semantic graph (`sg help`, `sg types`, `sg describe <Type>`, and in the conditions with actions `sg actions`) and the policies in `docs/`. (2) Write what you will need later into your workspace so that requests are fast and correct, for example `NOTES.md` with the types, properties, links (and actions) you will use for each kind of request and a checklist per policy. (3) Try a few queries to check that you read the graph correctly.

SGM-self. The week starts soon. Before it does, build the semantic graph that you will work with all week. A good model makes later requests fast and correct. Do not change the ERP. (1) Study the ERP and the documents in `docs/` (data dictionary, policies, transaction reference). (2) Define object types for the business entities, with clear names and descriptions (meaning, units, code values). Map each type to its table with `sg map`, translating codes into meaningful values. Define the links between the types. (3) Model the derived quantities that the policies and typical questions need (for example values, balances, availability), as properties of SQL-backed mappings or as functions. (4) Define actions for the changes that the policies govern. Each action checks its policy (criteria, or a function that throws an error) and emits the needed ERP transactions through the outbox. Read `sg help action` and test each action with `--validate-only`. (5) Run `sg validate`, then `sg publish`. Compare a few objects and counts with `erp sql`. (6) Write a short `NOTES.md` that describes your model.

The SGM-self instructions are more directive than the others, and the interface help adds templates and a runnable example on two inventory tables; this asymmetry favours SGM-self’s modelling and is part of what Section 6.3 measures.

E THE EXPERT GRAPH

The expert graph has 13 mirrored object types — Customer, Material, Vendor, Warehouse, InventoryBalance, SalesOrder, SalesOrderLine, Shipment, Invoice, PurchaseOrder, PurchaseOrderLine, PriceChange and Notification — plus a PolicySetting type that holds the approval threshold, so that the day-four change is one action. Codes are translated into labels (segment A → Gold, status H → Hold). Eight types are backed by read-only queries that add derived quantities: order value after discount, open status, a customer’s open order value and available credit, whether a customer is a strategic account and its maximum discount, a material’s available stock, open purchase orders and reorder need, and invoice days overdue. Twelve link types connect customers, orders, lines, materials, vendors, inventory, warehouses, invoices, shipments and purchase orders. Eleven actions implement the policies with function rules: entering a sales order (credit, discount, approval and priority rules), changing an order line (including the credit re-check), cancelling, approving, allocating (warehouse choice), checking a new EDI order, reordering a material (idempotent), changing a list price, putting a customer on credit hold, notifying, and changing the policy setting. Four query functions compute an order’s value, a customer’s available credit, the best warehouse for an order and a material’s re-

order need. A scripted agent that uses only this graph and three automations solves every request of the reference session and handles every standing-rule trigger (Section 5).

F ADDITIONAL RESULTS

Table 4: Where the input comes from (seeds as in Table 2). Preparation context: characters of the first request’s transcript, which every later call re-sends; tool output: characters returned by commands per request.

Condition	Tokens/call (k)	Calls/request	Preparation context (k chars)	Tool output/request (chars)
Agent (raw ERP)	88.1	3.62	149	1203
KG (L1)	90.8	3.97	120	2348
+Actions (L2)	76.2	3.57	134	1833
SGM-expert (L3)	99.2	3.69	147	2285
SGM-self	159.6	3.21	454	1072

Table 5: Standing-rule obligations met in time by rule, rescored from the change logs (Appendix G), with the live checkpoint values. SR1 replenish below reorder point; SR2 check new EDI orders on arrival; SR3 allocate strategic priority-one orders. By automations: share of the met obligations met by an automation.

Condition	SR1 (%)	SR2 (%)	SR3 (%)	Recall (%)	Precision (%)	By automations (%)	Live recall (%)	Live precision (%)
LLM (long context)	77.8	71.4	20.0	63.3	72.4	0.0	58.9	60.0
RAG	0.0	5.9	0.0	4.6	75.0	0.0	4.3	75.0
Agent (raw ERP)	90.0	100.0	85.3	96.4	98.6	0.0	96.3	80.7
KG (L1)	100.0	99.6	100.0	99.7	98.9	0.0	99.2	73.0
+Actions (L2)	96.7	100.0	49.2	90.5	98.4	0.0	89.0	82.0
SGM-expert (L3)	100.0	100.0	55.7	92.6	99.5	80.0	91.1	81.6
SGM-self	100.0	100.0	74.8	95.7	99.7	82.2	94.7	80.5

Table 6: Request families of a 40-request session.

Family	Example	#	Graded by
F1 lookup	payment terms and credit limit of a named customer	4	answer
F2 multi-hop	vendors of the materials in a customer’s latest open order	5	answer (set)
F3 aggregation	open order value per region for Gold customers	6	answer (tolerance 0.5%)
F4 conditional	customers with invoices >30 days overdue and open orders	4	answer (set)
F5 policy write	enter a phone order with a requested 12% discount	8	ERP diff + answer
F6 bulk update	a vendor raised prices by 4%: update its active materials	3	ERP diff
F7 standing rule	from now on, check every new EDI order on arrival	4	later events
F8 continuity	list the orders you entered today	2	answer
F9 report	end-of-day report with five figures	4	answer (per field)

The whole study, including the pilot, the aborted first main run, the cleared-conversation variant, the scaling study and the transfer study, made 9,167 model calls for 155.94 CNY at list price; most runs fell in the provider’s half-price hours, and the amount paid was 89.68 CNY (pilot 23.42, aborted run 8.84, main study 41.25, cleared-conversation variant 9.24, scaling 4.94 and transfer 1.98 CNY).

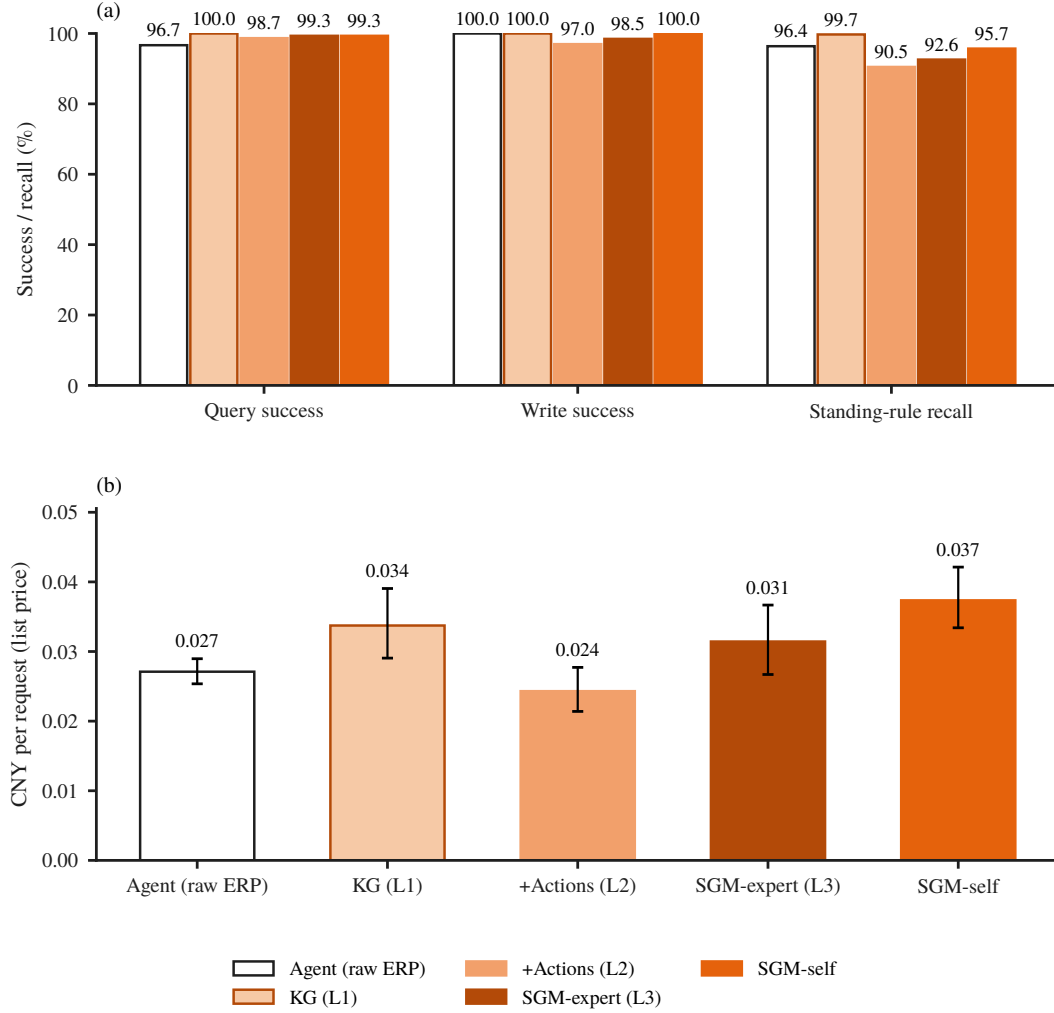


Figure 2: Ablation ladder at scale M, with SGM-self for comparison: (a) success on queries and on writes, and standing-rule obligations met in time; (b) list cost per request with 95% bootstrap intervals. Seeds as in Table 2.

Table 7: Request success by family (%; scale M; F7 standing instructions are graded through the rule checkpoints).

Condition	F1	F2	F3	F4	F5	F6	F7	F8	F9
LLM (long context)	100.0	80.0	50.0	50.0	100.0	33.3	–	100.0	100.0
RAG	66.7	0.0	16.7	0.0	37.5	0.0	–	50.0	25.0
Agent (raw ERP)	100.0	100.0	94.4	95.8	100.0	100.0	–	100.0	91.7
KG (L1)	100.0	100.0	100.0	100.0	100.0	100.0	–	100.0	100.0
+Actions (L2)	100.0	100.0	94.4	100.0	100.0	88.9	–	100.0	100.0
SGM-expert (L3)	100.0	100.0	100.0	95.8	97.9	100.0	–	100.0	100.0
SGM-self	100.0	100.0	100.0	95.8	100.0	100.0	–	100.0	100.0

Table 8: Graphs built by SGM-self (mean over sessions). Coverage and agreement are measured against the expert graph, which is one valid design among many; raw SQL requests: requests in which the agent used SQL at least once.

Metric	SGM-self (mean)
Types built	14.7
Links built	13.8
Actions built	15.3
Functions built	18.2
Automations built	4.2
Type coverage (%)	100.0
Property agreement (%)	48.3
Link recall (%)	60.6
Policy coverage (%)	—
Automation correctness (%)	—
Raw SQL requests (%)	86.0

Table 9: Exploratory variant: the conversation is cleared after the preparation request (cleared), against the main runs on the same seeds.

Condition	Seeds	Success (%)	Tokens/request (k)	Tokens/call (k)	CNY/request	Wall time, median (s)	Preparation CNY
Agent (raw ERP)	3	96.3	298.7	82.3	0.026	10.9	0.22
Agent (raw ERP) (cleared)	3	97.2	196.0	55.5	0.022	9.9	0.43
SGM-expert (L3)	3	99.1	347.1	94.4	0.031	9.6	0.15
SGM-expert (L3) (cleared)	3	99.1	284.2	71.4	0.029	8.9	0.24
SGM-self	3	100.0	467.3	154.3	0.032	8.9	0.83
SGM-self (cleared)	3	100.0	248.7	66.8	0.025	8.7	1.36

Table 10: Accounting of the model calls of the reported runs by stage, at list price and at the price paid (half price in the provider’s off-peak hours).

Stage	Sessions	Requests	Calls	Preparation CNY	List CNY	Paid CNY
main	37	1476	6279	14.07	95.04	47.52
pilot	5	69	253	0.53	9.41	9.41
scaling	7	140	716	1.74	9.89	4.94
transfer	2	120	437	0.22	3.95	1.98

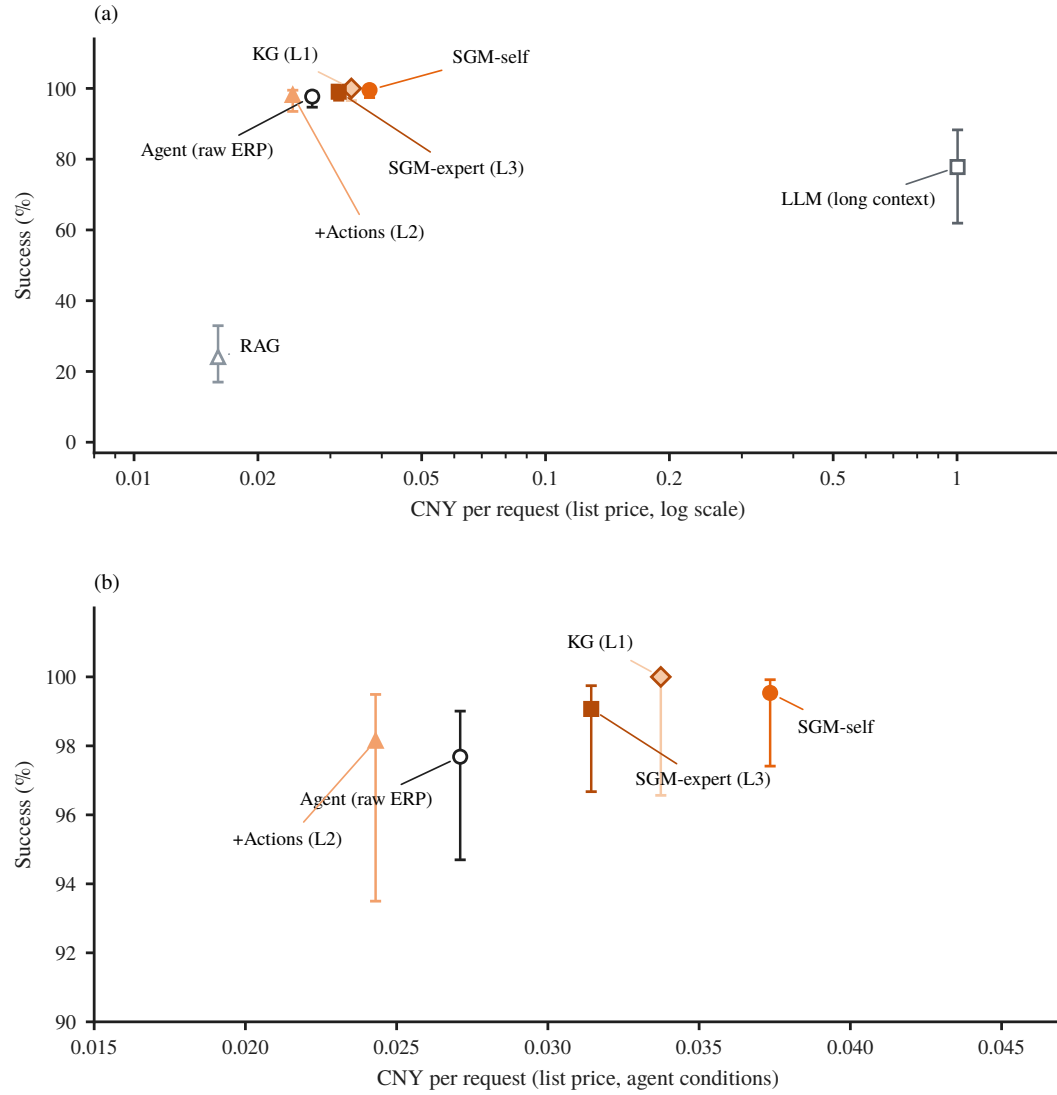


Figure 3: Success against list cost per request at scale M: (a) all conditions (log scale); (b) the agent conditions. Bars are 95% bootstrap intervals over requests; filled markers are graph conditions, open markers baselines.

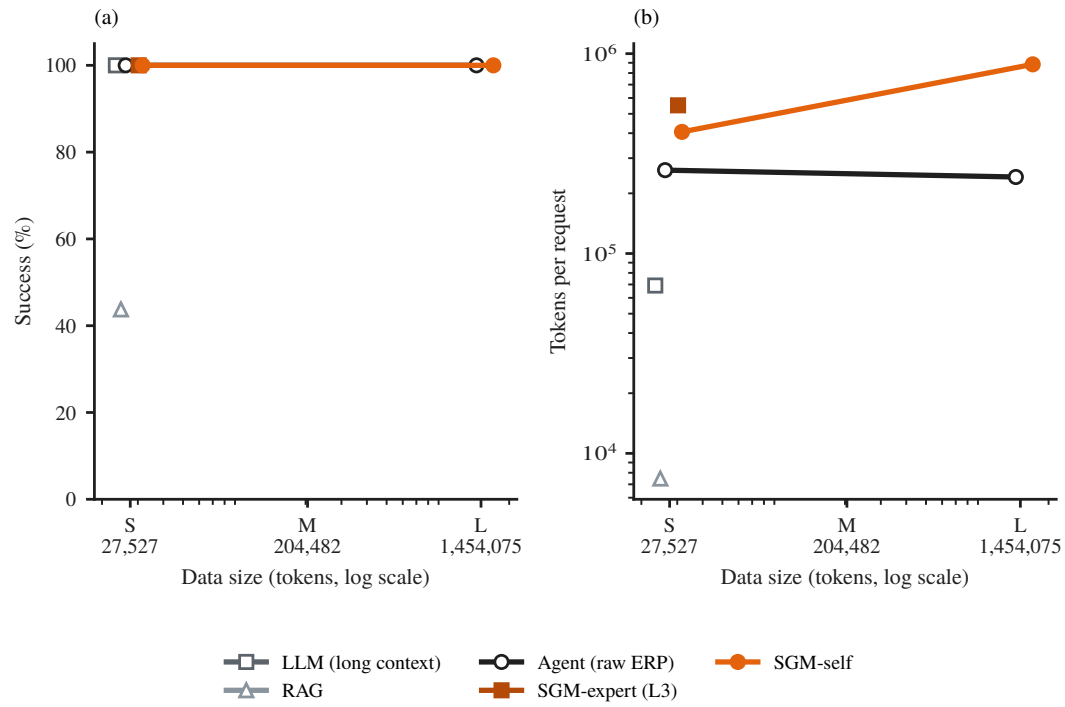


Figure 4: Scaling: success and tokens per request over the first 20 requests of one session at scales S and L (data size on a log axis; the main study at M has another request mix and is in Table 2). SGM-expert ran at S only; at L only the raw agent and SGM-self ran: the data exceed the context window of long-context prompting, and retrieval is a floor at M already.

G EVALUATION HYGIENE

We report every problem that affected the runs or their scoring, because they bear on any benchmark of agents with a shell. Items (i)–(iii) surfaced in the pilot and were fixed before the main study; items (iv)–(x) arose during or after it. (i) *Leaked reference*. The first sandbox exposed the research installation read-only. An SGM-self agent, inspecting the wrapper script of its command-line client, found the reference graph’s source files and published copies of its functions as its own. The final sandbox shows an execution only the language runtimes, the two clients and its own workspace. (ii) *Leaked instructions*. The first policy documents restated the standing instructions; an agent with automations then installed them before they were given. Standing instructions now appear only in the requests that give them. (iii) *Scoring*. A rule-triggering event that an agent ignored was re-counted at every later checkpoint, and rule effects made in another request’s window were scored as out-of-scope writes of that request. Each trigger is now counted once, rule work is scored only by the rule checkpoints, and an instruction is in force from the moment it is given, whatever the reply. Pilot data are used only for cost estimates and are not part of any reported result. (iv) *Prompt-prefix stability*. The first main run exposed each execution’s workspace under a fresh host path. The agent harness writes the working directory into the system prompt, so the prompt prefix changed at every request: the provider’s prefix cache missed on the first model call of each request, and agents tried to change into directories of earlier requests that no longer existed. The direct conditions, whose prompt is a fixed prefix followed by the current data, were not affected, so the artefact biased cost and time against the agent conditions. The workspace now appears at one fixed path in every execution, as in container sandboxes; the main study was restarted from scratch, and the partial first run (71 requests, stopped when the artefact was found) is not used. The driver itself had also run out of memory once in that run, because it buffered the agent’s streaming output; streaming updates are now dropped inside the sandbox. (v) *Answer formats and final-state writes*. The live grader compared answers with the oracle’s raw values, so an answer that gave a number as a string or the documented meaning of a code (“Shipped” for order status S, “Net 30” for payment terms N30) was scored wrong, although the request asks for the status, not its code; the graph conditions are affected most, because their mappings translate codes into meanings. It also judged writes on every intermediate change, so a phone order created with one status and corrected within the same request failed, as did a note in a free-text field or order lines in another order. We regraded every condition with one rule: numbers and codes are normalised, the code or its documented meaning is accepted, and writes are compared by the final state of each row in the request’s window, ignoring free-text fields and the order of lines; the expected values are those at the start of the request, reconstructed from the session’s change log. This changed 71 answers (15 for the raw agent, 10 for SGM-self, 12 for SGM-expert). Before the regrading, success was 69.4% (LLM), 13.0% (RAG), 90.7% (Agent), 90.7% (KG), 89.8% (+Actions), 93.5% (SGM-expert) and 94.9% (SGM-self). (vi) *Standing-rule timing*. The live checkpoints registered a rule’s obligations only at the first event batch after the rule was given. An agent that handled the existing backlog in the very request that gave the rule was therefore scored as making spurious effects, and an agent that ignored the backlog as missing it, so the two errors pointed in opposite directions for different conditions. We rescored every session from its change log: the log is replayed on the initial world, the rule’s trigger is evaluated after every change, and an obligation starts the moment its trigger holds (for the backlog, at the start of the request that gives the rule) and is due by the end of the next request; an effect satisfies it if it matches the policy decision on the state at the onset or at a later request start. Table 5 reports the live and the rescored values side by side; the main text uses the rescored ones. (vii) *Stale reads in the read-only graph*. The first KG sessions read a graph that the host synchronised only after event batches and actions; KG writes with raw transactions, so a read later in the same request could miss the agent’s own write. The host now pulls the change log before every read in that mode, and KG was re-run from scratch on all three seeds; the first KG runs are not used. (viii) *Time limits*. 3 executions exceeded their limit (10 minutes for a request, 40 for preparation): the preparation of one KG session, and one request each of SGM-expert and SGM-self; none of the raw agent. In the SGM-self request, which we inspected while it ran, the agent had started a recursive text search over the whole file system of its sandbox; the other two ended without a model call for the rest of the limit, and their cause is unknown. The first version of the driver aborted the session in that case; we changed it to score the execution as a failed request with its metered cost and to continue, and resumed the affected sessions from their last request. Without its time-out, SGM-self would have succeeded on every graded request. In the scaling study, the preparation of SGM-self at scale S also exceeded its limit; its commit then failed on the orchestrator’s 900-second lease, and the session was resumed with the preparation scored as timed out (the lease is now 3,000 seconds).

(ix) *Interface boundaries*. The sandbox hides the research installation and the other sessions, and the clients forward only the commands of the condition; we also audited every command of the 37 main sessions. No agent wrote to the ERP outside its interface (0 raw writes in the conditions without them) or ran SQL in a condition without SQL (0). Agents did look around: 79 commands read the source of the command-line clients (30 by SGM-self, 21 by SGM-expert), and SGM-self issued 17 commands that probed the interface sockets directly during preparation, none of which wrote to the ERP. (x) *Infrastructure faults*. Launching a later stage while sessions were running replaced the shared directory of the command-line clients, so six running executions lost their clients and their agents searched the file system for them. The rule for exclusion is fixed by time, not by outcome: an execution is excluded if it was running when the directory was replaced. We stopped these executions, excluded the four affected requests as interrupted (2 in the main conditions, 2 in the cleared-conversation variant), re-ran the two cleared-conversation sessions whose preparation was affected under a new identifier, and changed the launcher to update the clients in place.

H EXTENDED RELATED WORK

This section gives the full survey behind Section 2. *Search*. On 8 October 2026 we queried arXiv, Semantic Scholar and OpenAlex for ten topics (graph-based reasoning and retrieval; long context and retrieval; ontologies, semantic layers and text-to-SQL; knowledge-graph and ontology construction; agent memory; long-horizon and enterprise benchmarks; tools and executable world models; event-driven agents and active rules; agent cost; agent-built tools and skills), merged 4,239 records into 475 candidates, screened titles and abstracts with a typed classifier and by hand, kept 124 works and verified each against arXiv, Crossref, DBLP or the publisher. A design review added 24 targeted works (runtime policy guards, semantic layers, business artifacts, view update, ontology evaluation), verified the same way.

Knowledge graphs and semantic layers for language models. Knowledge graphs give entities and relations an explicit, typed meaning (Hogan et al., 2021; Noy et al., 2019), and a large body of work combines them with language models (Pan et al., 2024): models reason along graph paths (Sun et al., 2024; Ma et al., 2025; Luo et al., 2024; Jiang et al., 2023), retrieve subgraphs as context (Baek et al., 2023; He et al., 2024; Mavromatis & Karypis, 2025), or build graph indexes over text for retrieval (Edge et al., 2024; Gutiérrez et al., 2024; 2025; Guo et al., 2025; Liang et al., 2025; Li et al., 2025; Sarthi et al., 2024; Peng et al., 2025). Over relational data, ontology-based data access maps an ontology to tables so that queries are posed in domain terms (Poggi et al., 2008; Lenzerini, 2011; Calvanese et al., 2016; Das), and industrial semantic layers define measures, dimensions and joins once for every consumer (dbt Labs, 2026; Google, 2026; Cube, 2026); text-to-SQL methods instead generate queries against the raw schema (Yu et al., 2018; Li et al., 2023; Lei et al., 2025; Pourreza & Rafiei, 2023; Gao et al., 2024; Talaei et al., 2024; Wang et al., 2025a) and remain weak on enterprise schemas whose meaning lives in business knowledge (Chen et al., 2024; Liao et al., 2026; Gwimm & Eisenach, 2026). Posing enterprise questions over a knowledge graph of the database instead of its SQL schema raised GPT-4’s accuracy from 16% to 54% in one benchmark, and checking queries against the ontology raised it to 72% (both self-reported; Sequeda et al., 2024; Allemang & Sequeda, 2024); a paired benchmark reports fewer errors and hallucinations when three frontier models query through a semantic layer (self-reported; Rumiantsev & Fokeev, 2026); ontologies also ground retrieval (Sharma et al., 2025), governed analytics interfaces keep aggregation logic out of the model (Singh et al., 2026), and ChatDB gives an agent a database as symbolic memory that it reads and writes with SQL (Hu et al., 2023). Closest to us, EvoOntology lets a builder agent construct an ontology layer for data agents and refines it by evaluation (Chong et al., 2026). All of these are *read* models evaluated per question. We add the write path and triggers — actions whose criteria encode policies and automations that act on change — have the working agent build them, and evaluate them over long, continuous sessions with full cost accounting.

Building knowledge graphs and ontologies with language models. Language models extract entities, relations and schemas from text (Zhu et al., 2024; Zhang & Soh, 2024; Mo et al., 2025; Lairgi et al., 2024; Bai et al., 2026), learn and engineer ontologies (Babaei Giglou et al., 2023; Lo et al., 2024; Zhang et al., 2025a; Oyewale & Soru, 2026), and admit facts through review (Bykampadi et al., 2026); benchmarks score the graph itself against references (Mihindukulasooriya et al., 2023), while ontology engineering evaluates a model by the competency questions it answers and by catalogues

of modelling pitfalls, because a single gold ontology rarely exists (Grüninger & Fox, 1995; Poveda-Villalón et al., 2014). Our setting differs in input and in what counts as success: the agent models structured sources together with policy prose, the model includes operations and rules, not only classes and relations, and it is judged by the work it enables and by what it costs to build.

Context, memory and long-horizon agents. Long contexts degrade with length and position (Liu et al., 2024a; Hsieh et al., 2024; Modarressi et al., 2025; Hong et al., 2025), which motivates retrieval (Lewis et al., 2020; Gao et al., 2023b; Asai et al., 2024) and its combinations with long contexts (Li et al., 2024; Jiang et al., 2024). Agent memory systems page, summarise or structure the agent’s own experience (Packer et al., 2023; Park et al., 2023; Shinn et al., 2023; Xu et al., 2025b; Chhikara et al., 2025; Wang et al., 2025d; Zhang et al., 2025c; Summers et al., 2024; Zhang et al., 2024), including as temporal or world-model knowledge graphs (Rasmussen et al., 2025; Anokhin et al., 2025), and are evaluated on long conversations (Wu et al., 2025; Maharana et al., 2024; Hu et al., 2026); for precise recall, agent-controlled search over the raw log can match structured memory (Li et al., 2026). Agent benchmarks for enterprise and office work test tool use over databases and applications (Styles et al., 2024; Yao et al., 2025; Barres et al., 2025; Xu et al., 2025a; Wang et al., 2024d; Trivedi et al., 2024; Huang et al., 2025; 2026; Drouin et al., 2024; Boisvert et al., 2024; Wang et al., 2025c; Gruenbaum et al., 2026; Luo et al., 2025; Wang et al., 2026d; Liu et al., 2024b; Mialon et al., 2024), and in long simulated business runs agents derail with high variance, not obviously because the context fills (Backlund & Petersson, 2025). Cost is rarely reported alongside accuracy (Kapoor et al., 2024; Wang et al., 2025b). A semantic graph is not memory of experience: it models the state and rules of the domain and is kept in sync with the system of record. OPSWEEK complements existing benchmarks with continuous sessions over evolving data, standing instructions and state-conditional grading.

Tools, compiled knowledge and automation. Agents act through tools and APIs (Yao et al., 2023; Schick et al., 2023; Patil et al., 2024; Qin et al., 2024; Yan et al., 2024; Anthropic, 2024) or code (Wang et al., 2024b; Gao et al., 2023a; Wang et al., 2024e; Liang et al., 2023). Several lines compile knowledge into executable artefacts: language models write planning domains or world models that a solver or simulator then uses (Liu et al., 2023; Guan et al., 2023; Tang et al., 2024; Wang et al., 2026b), and agents accumulate reusable tools and skills (Wang et al., 2024a; Cai et al., 2024; Qian et al., 2023; Yuan et al., 2024; Wang et al., 2024c; Zheng et al., 2025; Hu et al., 2025), also as a governed organisational registry (Guo, 2026d). Policies can also be compiled into guards that check an agent’s tool calls at run time, from policy documents (Zwerdling et al., 2025), from a rule language (Wang et al., 2026a; Rebedea et al., 2023) or from verified policy models (Chen et al., 2025; Xiang et al., 2025). Business artifacts describe operations as data-centric objects with lifecycles (Nigam & Caswell, 2003; Bhattacharya et al., 2007), and their guard–stage–milestone form drives these lifecycles by guarded transitions and events (Hull et al., 2011), the closest formal precedent of actions with criteria and automations. Event–condition–action rules and triggers are classical in active databases (McCarthy & Dayal, 1989; Widom & Ceri, 1995), and constraint languages validate graph data (Knublauch). Writing through a model to its sources is the view-update problem (Bancilhon & Spyrtatos, 1981; Dayal & Bernstein, 1982), also studied for ontology-based data access (De Giacomo et al., 2017); practice separates a write path of commands from the read models that are rebuilt from events (Fowler, 2011; 2005), which is how our host writes through an outbox and re-synchronises. language models now generate workflows and process automations (Ye et al., 2023; Zeng et al., 2023; Zhang et al., 2025b; Fan et al., 2025), decide when to act proactively (Lu et al., 2025; Liu et al., 2026), and run governed back-office plans (Moslemi et al., 2026). In a semantic graph, operations, guards and triggers are not free-standing code: they are typed against the same object model the agent queries, the engine enforces their criteria, and the agent authors them in the same branch–validate–publish cycle as the schema.

I STATEMENTS

Reproducibility. Every result number in this paper is generated by the analysis scripts from the run records (request records, metering logs and ERP change logs); each run records the hashes of its data bundle, prompts and system prompt. Prompts and the interface help were tuned on one pilot session at scale S before the main study; the scoring corrections of Appendix G came after it. Sections 4–6 and Appendices B–G describe the conditions, prompts, limits and scoring. The benchmark generator,

ERP server, oracle, grader, drivers and analysis code are available from the authors; the ontology kernel is part of a commercial platform and is not released.

Ethics. The study involves no human participants. All benchmark data are synthetic, and the company in them is fictitious. The production observations in Section 1 are aggregate counts from the authors’ own deployment and contain no customer data. Agents ran in sandboxes with memory, process and network limits on a pool shared with production agents, under a watchdog that paused the study when the pool’s free memory fell.

Use of language models. DeepSeek V4.1 Flash is the subject of the experiments. Coding and writing assistants helped implement the experiment code, screen the literature and translate the Chinese version; the authors checked every claim, number and reference against the run records and the sources.

Competing interests and funding. The authors are employed by AIDC, which develops the ontology kernel used by the graph conditions and funded this work. We cite three earlier working papers of our group (Guo, 2026a;c;b) and one of our studies on governed self-improvement (Guo, 2026d).