ONTOS: CANDIDATE-FIRST KNOWLEDGE GRAPH QUESTION ANSWERING WITH ONTOLOGY-CONSTRAINED PROGRAM SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge graph question answering requires translating a question's intent into executable programs over a large graph schema. Direct query generation must resolve schema identifiers and query structure, while LLM-guided exploration repeatedly interleaves planning with graph traversal. We present ONTOS, an ontology-driven, candidate-first framework that separates conceptual planning from initial program construction. An LLM produces an abstract derivation sketch without graph identifiers. A dual-track synthesizer then combines sketch-guided, schema-filtered search with type-checked instantiation of program structures from annotated training questions. The resulting candidates are executed and presented to the LLM for selection, with low-confidence selections routed to a bounded, ontology-grounded program-repair stage. Across 6,159 evaluation questions from CWQ, WebQSP and GrailQA, ONTOS achieves pooled Hits@1 of 83.3%, 87.6% and 89.1% with Qwen2.5-7B, Qwen3.5-9B and Qwen3.5-27B, respectively, without task-specific LLM fine-tuning; on CWQ, all three exceed the results reported for the baselines we compare with, including GPT-4-based agents. With the 27B backbone, repair increases pooled Macro-F1 from 79.2% to 81.6% while processing 26.7% of questions. Stage-level ablations show complementary contributions from the two synthesis tracks. These results support ontology-constrained candidate construction and LLM-based selection as an effective division of work for knowledge graph question answering.

1 INTRODUCTION

Knowledge graph question answering (KGQA) requires translating natural-language questions into executable programs that combine relations, entities and constraints over a knowledge graph such as Freebase (Bollacker et al., 2008). A question may involve several reasoning steps, secondary entities, or conditions on dates and numbers, so both its intended meaning and its realization in the graph schema must be recovered (Berant et al., 2013; Yih et al., 2015). LLM-based approaches either generate logical forms or SPARQL queries (Li et al., 2023a; Yu et al., 2023), or guide graph exploration through successive relation choices (Sun et al., 2024; Chen et al., 2024; Liu et al., 2026); Figure 1 contrasts their decision points.

Query generation must resolve concrete schema identifiers and join structure among tens of thousands of relations, whereas hop-by-hop exploration couples semantic planning with local search: at the topic entities of our evaluation questions the LLM faces 28 relation types at the median and at least 97 for one entity in ten, and a useful path may be pruned before its relevance becomes apparent. The exploration agents we compare with report 5.5–22.6 LLM calls per question (Liu et al., 2026). We investigate a candidate-first alternative, in which *the LLM describes the intended reasoning and evaluates executed candidates, while ontology-constrained synthesis handles the initial search over the graph*.

The knowledge graph's ontology makes this division possible. It supplies the relation vocabulary, relation domains and ranges, inverse pairs and the roles of compound value type (CVT) nodes; together with namespace and system-relation cleanup, these rules reduce the incident relation candidates at the median topic entity by 65%, a descriptive reduction that does not isolate type constraints.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

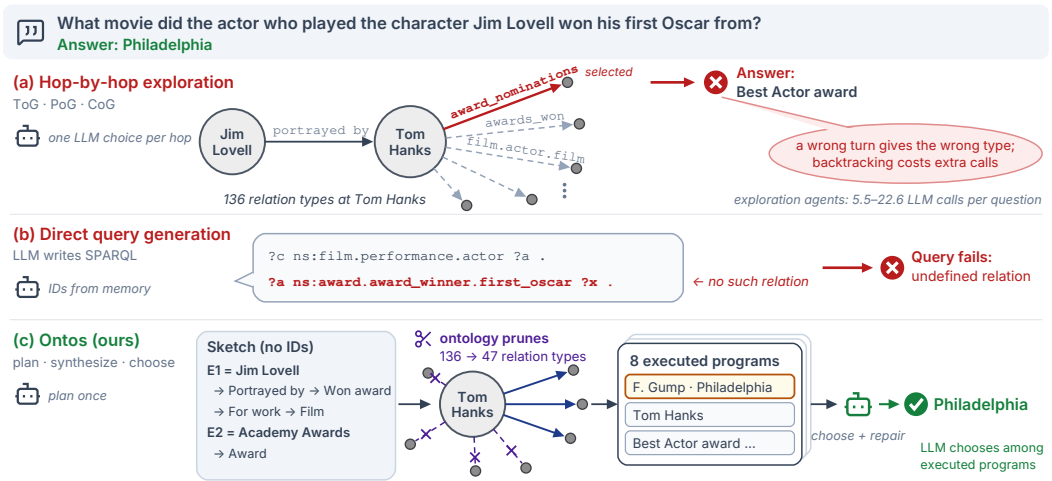


Figure 1: Illustrative decision points for a CWQ question. (a) In hop-by-hop exploration, early pruning can discard a useful path; the example node *Tom Hanks* has 136 incident relation types. (b) Direct query generation requires exact schema identifiers; an invented relation makes the query invalid, which differs from a valid query that returns no answer. (c) ONTOS writes a conceptual sketch, applies schema filtering and inverse folding (16 to 4 relation candidates at *Jim Lovell*, 136 to 47 at *Tom Hanks*), then chooses among eight executed programs. This example is routed to repair at confidence 3, where a missing constraint for "first" is added. The first two panels illustrate possible failure modes, not an inability of prior methods to revise earlier decisions.

We build on this observation ONTOS, an ontology-driven, candidate-first framework (Figure 2). The LLM first writes an abstract derivation sketch—the answer concept, conceptual relation paths and constraints—without graph identifiers. A dual-track synthesizer then constructs candidate programs. The micro-track searches schema-filtered relations guided by the sketch and can build paths that no training template covers; the macro-track instantiates type-compatible program structures retrieved from annotated training questions. ONTOS uses annotated training programs but does not fine-tune task-specific LLM parameters.

The LLM selects among the executed candidates and reports a confidence level; only low-confidence selections enter program repair. An editing agent proposes alternatives through ontology-grounded tools, and a pairwise gate decides whether an alternative replaces the default program. Repair may revise the sketch: the initial synthesis avoids stepwise LLM-guided exploration rather than prohibiting later replanning. With Qwen3.5-27B, the eight-program selection window contains a candidate whose answers are Hits@1-correct for 96.6% of questions; the highest-ranked candidate meets that metric for 76.8%, and the LLM's choice for 88.5%. These are answer-hit rates, not rates of programs that faithfully express the question.

On 6,159 questions from CWQ, WebQSP and GrailQA, ONTOS achieves pooled Hits@1 of 83.3%, 87.6% and 89.1% with Qwen2.5-7B, Qwen3.5-9B and Qwen3.5-27B. On CWQ, the 7B system obtains 82.4%, compared with the 54.0% reported for CoG with the same backbone; this is a cross-paper comparison rather than a controlled rerun. Within-system ablations show that both synthesis tracks contribute, and with the 27B backbone repair raises pooled Macro-F1 from 79.2% to 81.6% while processing 26.7% of questions. Our contributions are threefold:

- A candidate-first framework for KGQA that separates conceptual planning, ontology-constrained initial program construction and LLM selection over executed candidates.
- A dual-track program synthesizer that combines schema-filtered search with type-checked reuse of annotated training programs.
- A confidence-routed program-repair stage, together with an empirical analysis of when repair improves answer quality across backbones.

2 Related Work

Generating Logical Forms. Semantic parsing maps questions to executable logical forms or SPARQL (Berant et al., 2013; Yih et al., 2015), by ranking candidate paths or subgraphs (Miller et al., 2016; Zhang et al., 2022; Cao et al., 2023; Jiang et al., 2023b), fine-tuned generation (Yu et al., 2023; Luo et al., 2024), discriminative grounding against the schema (Gu et al., 2023), or in-context generation followed by binding (Li et al., 2023a;b). Interactive agents interleave query writing with access to the knowledge base: Interactive-KBQA (Xiong et al., 2024), SPINACH (Liu et al., 2024) and GRASP (Walter & Bast, 2025) search nodes, inspect neighborhoods and revise partial queries from execution feedback in a thought–action–observation loop (Yao et al., 2023). In ONTOS the model writes a sketch without identifiers; executable identifiers and joins are produced by program synthesis and compilation against the graph and its ontology.

Exploring the Graph. ToG (Sun et al., 2024) performs LLM-guided beam search over triples, PoG (Chen et al., 2024) adds adaptive planning and self-correction, and other work combines exploration with structured APIs (Jiang et al., 2023a), graph neural retrieval (Mavromatis & Karypis, 2025), symbolic agents (Liu et al., 2025a) or tools (Jiang et al., 2025); GCR (Luo et al., 2025) constrains the decoding of relation paths with a trie built from the graph. CoG (Liu et al., 2026) guides exploration with relation-only blueprints retrieved from similar training questions and refines the blueprint when search fails; ONTOS instead reuses complete executable programs of training questions, including variables, directions and aggregations. It lets the LLM plan at the conceptual level and choose among executed programs, and its micro-track can construct paths without a matching training template. Low-confidence selections may then be edited, with proposed replacements compared in two presentation orders (Zheng et al., 2023).

Using the Ontology. ORT (Liu et al., 2025b) builds label paths on the ontology from the question's purpose back to its conditions to guide retrieval, Allemang and Sequeda (Allemang & Sequeda, 2024) check generated queries against the ontology and ask the model to repair violations, and OntGQA (Shan et al., 2026), closest to our work, builds a relation-centric ontology graph from the Freebase schema, predicts the entity types of both endpoints of the reasoning path with a fine-tuned planner, and verifies retrieved paths with a judge. ONTOS uses schema information during program construction: admissible relations, CVT roles, secondary-entity attachment and scalar-constraint grounding. It does not fine-tune task-specific model parameters, but its macro-track does use annotated training programs.

3 Method

ONTOS answers a question in four stages (Figure 2). An LLM first plans the question at the conceptual level, as an abstract derivation sketch without graph identifiers (Section 3.2). The sketch is then turned into a set of executed candidate programs by a dual-track synthesis constrained by the ontology (Section 3.3). Finally, the LLM selects one candidate and reports its confidence, and only low-confidence selections are repaired by editing the selected program (Section 3.4).

3.1 Problem Setup

Knowledge Graph and Ontology. A knowledge graph $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \mathcal{L})$ is a set of triples over entities $\mathcal{E}$, relations $\mathcal{R}$ and literals $\mathcal{L}$. Its schema is an ontology $\mathcal{O} = (\mathcal{C}, \leq_{\mathcal{C}}, \mathrm{dom}, \mathrm{rng}, \mathrm{inv}, \mathcal{C}_{\mathrm{CVT}})$: a type hierarchy $(\mathcal{C}, \leq_{\mathcal{C}})$, the domain and range type of every relation, inverse relation pairs, and the Compound Value Types $\mathcal{C}_{\mathrm{CVT}} \subset \mathcal{C}$ whose nodes represent n-ary facts (e.g., an office term with holder, position, start and end), each with declared role relations.

Task. Given a question q and its topic entities $E_q \subseteq \mathcal{E}$, the goal is an executable program $\mathcal{P}$ over $\mathcal{G}$ whose denotation $[\![\mathcal{P}]\!]_{\mathcal{G}}$ is the answer set. A program is a SPARQL query consisting of triple patterns, filters on literal values, and modifiers for ordering and aggregation.

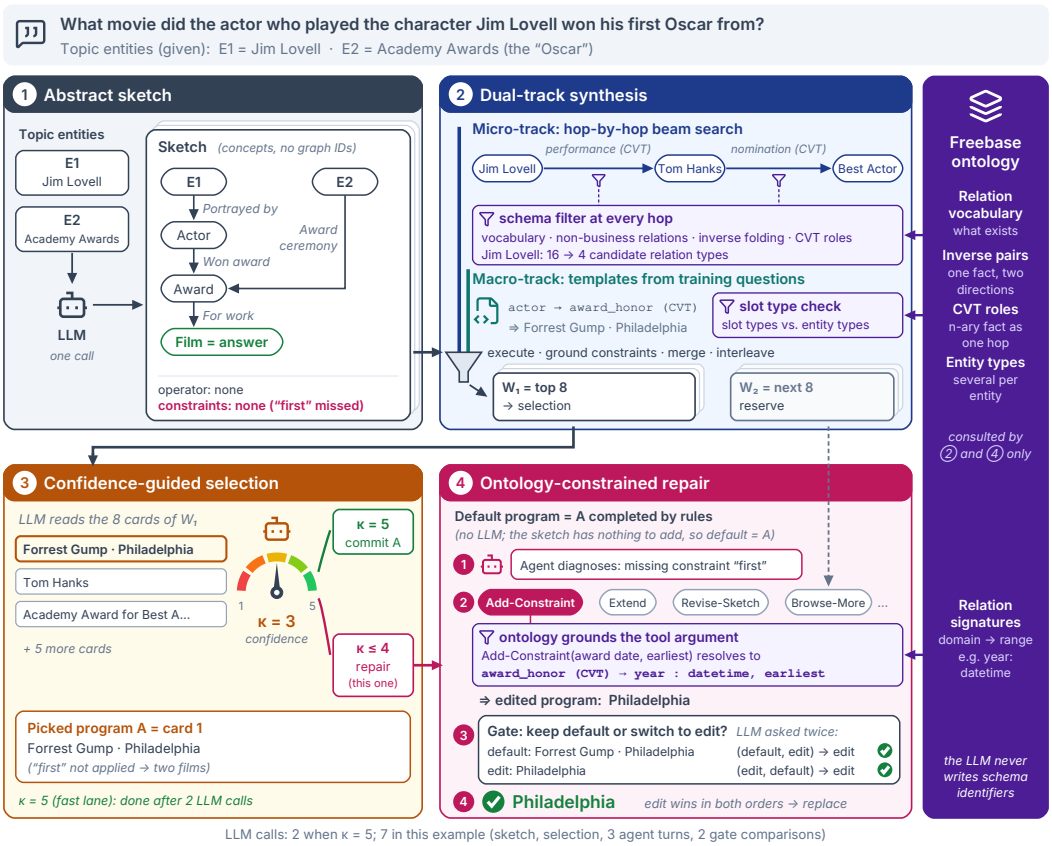


Figure 2: Overview of ONTOS on the question of Figure 1. Left: LLM steps, which write no schema identifiers; right: steps constrained by the Freebase ontology (purple). (1) The sketch misses "first". (2) Two tracks produce executed programs, merged into the selection window W_1 (eight) and a reserve W_2. (3) The LLM selects A with confidence 3, so the question enters repair; $\kappa = 5$ commits directly. (4) The default $\mathcal{P}_{\text{def}}$ is A completed by rules (here $\mathcal{P}_{\text{def}} = A$); the agent's edited program $\mathcal{P}_{\text{edit}}$ adds the missing ordering and replaces it only if the gate prefers it in both presentation orders.

3.2 ABSTRACT DERIVATION SKETCH

The LLM receives q and the names of E_q, each bound to a slot $E_1, E_2, \ldots$, and writes a sketch

$$S = \big(c_{\text{ans}},\ \{b_i\},\ \Phi,\ \text{op}\big) \tag{1}$$

containing no graph identifier: the expected answer concept c_{ans}; one branch b_i per topic entity, each a sequence of conceptual hops $(\text{relation label}, \text{target concept}, \text{intent})$ starting at its slot, with branches of secondary entities converging on the answer; scalar constraints $\Phi = \{(a, \circ, x)\}$ whose value x is a number or a date; and an operator $\text{op} \in \{\text{none}, \text{count}, \text{argmax}, \text{argmin}\}$. It is produced in one call under a JSON schema (prompt in Appendix B) and names concepts rather than schema items.

3.3 ONTOLOGY-CONSTRAINED DUAL-TRACK PROGRAM SYNTHESIS

The sketch is grounded by two tracks that produce executed programs independently.

Micro-Track: Hop-by-Hop Search under the Ontology. Starting from the anchor of each branch, the micro-track expands one conceptual hop of the sketch at a time (Figure 3). At a frontier node, relations outside the schema vocabulary, system relations and the inverse of the previous step are removed, each inverse pair is folded into one candidate, and a relation into a CVT node is expanded with one of its declared roles, so that an n-ary fact counts as one hop. Each step r is scored

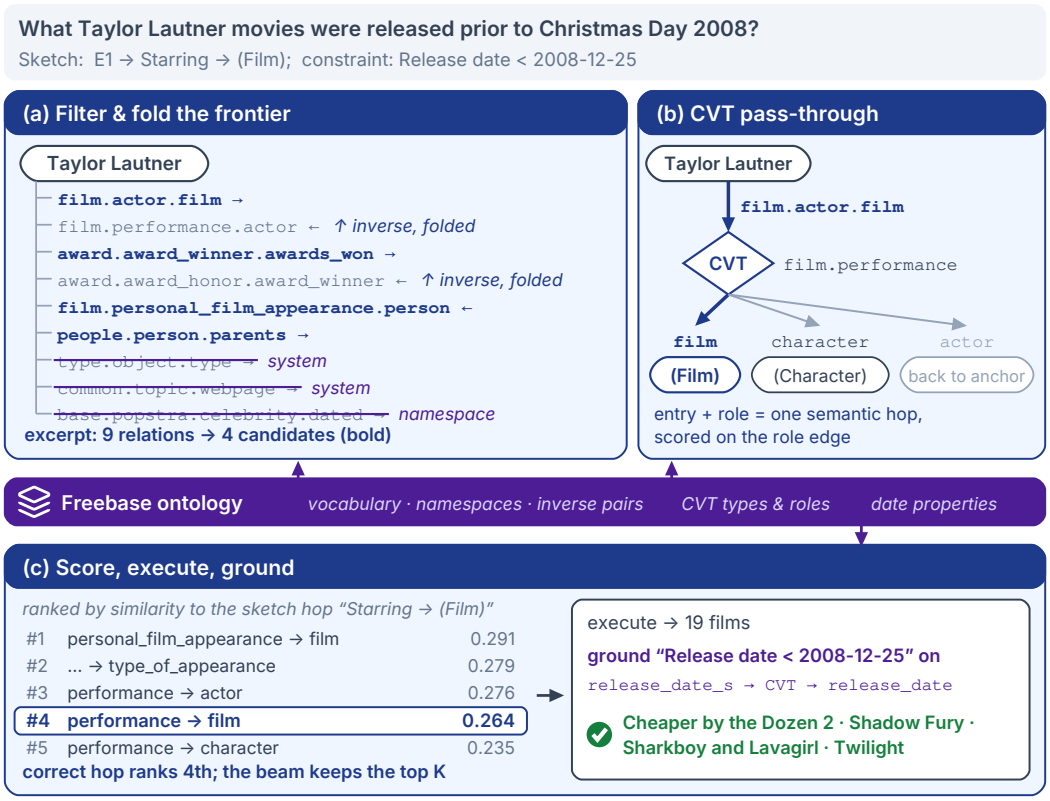


Figure 3: One micro-track hop from the anchor *Taylor Lautner* (an excerpt of its relations). (a) Filtering and inverse folding reduce nine relations to four candidates. (b) A relation into the CVT type `film.performance` and one of its roles form one hop, scored on the role edge. (c) The target hop ranks fourth but stays in the beam; its program is executed, and the date constraint is grounded on the release-date CVT.

by the cosine similarity $s(r, p_j)$ between embeddings of r (the role relation for a CVT step) and of the sketch hop p_j; a path is scored by its mean hop score, and a global beam keeps the $K = 128$ best paths. Secondary topic entities are attached by a reverse join, the sketch operator decides counting or ordering, and surviving paths are compiled into SPARQL and executed; empty results are discarded and identical ones merged (Appendix A.2).

Macro-Track: Structures of Training Questions. Programs of training questions with program annotations are abstracted into templates by replacing their entities with slots. For an evaluation question, templates are retrieved by the embedding similarity between the question and the training questions that use them. Before any query is sent, templates whose slots are type-incompatible with the topic entities under $\mathcal{O}$ are discarded (49% of the checked templates); the remaining ones are instantiated and executed in retrieval order until eight have a non-empty denotation. These eight are ranked by a score that favors templates used by several training questions close to q and templates whose relations match the sketch hops (Appendix A.3).

Constraint Grounding and Fusion. On both tracks, scalar constraints of the sketch that a candidate does not express are attached to the node whose type carries the constrained property, which is often a CVT node or a node two hops from the answer (Appendix A.4). The two candidate lists are then fused: programs with identical denotations are merged into one card marked as produced by both tracks, the remaining candidates of the two tracks are interleaved, and the first 16 programs are kept. The first eight form the candidate window W_1 shown to the selection; the next eight form a reserve W_2 that the repair stage can browse.

3.4 CONFIDENCE-GUIDED SELECTION AND ONTOLOGY-CONSTRAINED PROGRAM REPAIR

Cards. Each candidate is shown to the LLM as a card: its triple patterns with topic entities as their slots and names, its filters and modifiers in SPARQL syntax, and a preview of its answers (Appendix A.5).

Confidence-Guided Selection. The LLM reads q, the topic-entity slots and the cards of W_1, and returns a short justification, the chosen card $A \in W_1$ and a self-reported confidence level $\kappa \in \{1, \ldots, 5\}$ defined by a rubric (5: the program fully expresses the question and the answers have the requested type; 1: no card fits, A is only the closest). The output schema enumerates exactly the cards of W_1, so the selection can neither refuse nor name a program outside the pool. Questions with $\kappa < 5$ are routed to repair; the others are committed. The level is used for routing, not interpreted as a calibrated probability of correctness.

Deterministic Sketch Completion. Before any further LLM call, entity branches and scalar constraints of S that A does not realize are grounded on A. A secondary entity e is attached to a variable v of A either directly or through a two-hop connection whose mediator is a CVT node, since a two-hop connection through an ordinary entity (e.g., a shared country) filters answers without expressing the question. If the completed program A' is non-empty and differs from every card, it becomes the default program $\mathcal{P}_{\text{def}} = A'$; otherwise $\mathcal{P}_{\text{def}} = A$.

Program Editing Agent. The agent first diagnoses why A may be wrong (*missing constraint*, *one hop short*, *wrong relation chain*, *wrong anchor entity* or *answer not in pool*) and then edits programs for at most six turns, one action and one observation per turn (Yao et al., 2023). Its tools operate on existing programs, and their arguments are resolved against the ontology rather than written by the model: ADD-CONSTRAINT grounds a comparison or an ordering on a program as above; APPLY-SKETCH-CONSTRAINTS grounds all constraints of S; EXTEND appends one hop at the answer node using the micro-track's admissible relations; REVISE-SKETCH reruns the micro-track with a corrected sketch; BROWSE-MORE reveals cards from W_2; and COMMIT ends the episode. Produced programs are executed and enter the pool only if their denotation is non-empty and new (budgets in Appendix A.6). The program the agent commits is the edited program $\mathcal{P}_{\text{edit}}$.

Pairwise Replacement Gate. The gate controls only the agent's change: $\mathcal{P}_{\text{edit}} \neq \mathcal{P}_{\text{def}}$ replaces the default only if it wins two independent comparisons, one in each presentation order:

$$\hat{\mathcal{P}} = \begin{cases} \mathcal{P}_{\text{edit}}, & \text{if } \mathrm{Cmp}(\mathcal{P}_{\text{def}}, \mathcal{P}_{\text{edit}}) = \mathcal{P}_{\text{edit}} \ \wedge \ \mathrm{Cmp}(\mathcal{P}_{\text{edit}}, \mathcal{P}_{\text{def}}) = \mathcal{P}_{\text{edit}}, \\ \mathcal{P}_{\text{def}}, & \text{otherwise,} \end{cases} \tag{2}$$

where Cmp is an LLM call that sees only the question, its topic entities and the two cards; an invalid output counts as a vote for $\mathcal{P}_{\text{def}}$. The gate does not guarantee preservation of a correct original choice A, since $A \to \mathcal{P}_{\text{def}}$ is outside the pairwise comparison.

4 EXPERIMENTS

We evaluate ONTOS on the 6,159 questions in our CWQ, WebQSP and GrailQA evaluation sets to answer: **(RQ1)** How does it compare with parsing, retrieval and exploration methods, with the same and with stronger backbones? **(RQ2)** What do the two synthesis tracks and the repair stage contribute, and which repairs help? **(RQ3)** How do accuracy, repair and cost scale with the backbone?

4.1 EXPERIMENTAL SETUP

Datasets and Metrics. We use 3,531 CWQ (Talmor & Berant, 2018) questions, 1,628 WebQSP (Yih et al., 2016) questions and the 1,000 GrailQA (Gu et al., 2021) questions in our evaluation set, divided into I.I.D., compositional and zero-shot levels, for a total of 6,159 questions over Freebase (Bollacker et al., 2008) (Appendix Table 4). We compute Hits@1 with the same evaluation code and answer lists as ToG, PoG, CoG and RJE (Lin et al., 2025), so our Hits@1 is on the same scale as their reported results, and additionally report a set-level Macro-F1; all decisions of the system are taken without access to gold answers (definitions and matching limits in Appendix C).

Method	Backbone	CWQ		WebQSP		GrailQA
		Hits@1	F1	Hits@1	F1	Hits@1
LLM only (no knowledge graph)						
CoT (Wei et al., 2022)	GPT-3.5	38.8	–	62.2	–	28.1
Fine-tuned KG-augmented methods						
DECAF (Yu et al., 2023)	–	70.4	–	82.1	–	–
KG-Agent (Jiang et al., 2025)	–	72.2	–	83.3	–	–
OntGQA (Shan et al., 2026)	Llama-2-7B	74.6	56.8	**91.5**	77.1	–
Training-free prompting, closed-source LLMs						
KB-BINDER (Li et al., 2023a)	Codex	–	–	–	74.4	–
ToG (Sun et al., 2024)	GPT-3.5	57.1	41.9	76.2	50.9	68.7
PoG (Chen et al., 2024)	GPT-3.5	63.2	43.7	82.0	58.1	76.5
CoG (Liu et al., 2026)	GPT-3.5	66.9	59.9	86.8	74.3	79.2
ToG (Sun et al., 2024)	GPT-4	67.6	47.6	82.6	58.9	81.4
PoG (Chen et al., 2024)	GPT-4	75.0	42.1	87.3	59.8	84.7
ORT (Liu et al., 2025b)	GPT-4o	65.4	58.7	87.7	71.8	–
CoG (Liu et al., 2026)	GPT-4	77.8	69.2	89.7	75.5	86.4
Training-free prompting, open-source LLMs						
ToG (Sun et al., 2024)	Qwen2.5-7B	42.5	28.7	56.0	37.3	62.6
PoG (Chen et al., 2024)	Qwen2.5-7B	46.0	31.4	58.5	40.4	68.9
CoG (Liu et al., 2026)	Qwen2.5-7B	54.0	48.5	74.5	63.9	72.0
ONTOS (ours)	Qwen2.5-7B	82.4	73.3	87.0	75.9	80.5
ONTOS (ours)	Qwen3.5-9B	88.4	80.4	87.5	78.3	85.1
ONTOS (ours)	Qwen3.5-27B	**89.2**	**81.2**	90.1	**80.4**	**87.1**

Table 1: Performance (%) on CWQ, WebQSP and GrailQA (GrailQA: overall Hits@1; levels in Appendix Table 6). Baseline values are as reported by their authors (ToG, PoG and CoG with Qwen2.5-7B are not our reruns); Hits@1 uses the same evaluation code as these methods. Best value per column in bold; more baselines in Appendix Table 5.

Topic Entities. Like the agentic baselines, we start from the topic entities released with each question and perform no entity linking; a missing entity is recovered only as a constraint on a candidate program (Appendix C).

Implementation. Every stage uses the same LLM, served by vLLM (Kwon et al., 2023) with greedy decoding under JSON schemas: Qwen2.5-7B-Instruct (Yang et al., 2024), Qwen3.5-9B or Qwen3.5-27B (Qwen Team, 2026) for the main results, and Qwen3.5 with 0.8B, 2B and 4B parameters for scaling, all with an identical pipeline and configuration. Embeddings come from jina-embeddings-v5 (Akram et al., 2026), and programs run on a Virtuoso endpoint (Appendix C).

4.2 MAIN COMPARATIVE RESULTS (RQ1)

Table 1 compares ONTOS with prior methods on the three benchmarks; results by GrailQA generalization level are in Appendix Table 6.

Same Backbone. With Qwen2.5-7B, a backbone also reported for ToG, PoG and CoG, ONTOS obtains 82.4% CWQ Hits@1 against CoG's reported 54.0%, a 28.4-point difference; the corresponding reported differences are 12.5 points on WebQSP and 8.5 on GrailQA.

Stronger Backbones. On CWQ, our three results are 4.6 to 11.4 points above the reported 77.8% of CoG with GPT-4. On WebQSP, the 27B system (90.1%) is above CoG with GPT-4 (89.7%) but below OntGQA (Shan et al., 2026) (91.5%), which fine-tunes Llama-2-7B as type planner and judge. On GrailQA, the 27B system reaches 87.1% against CoG with GPT-4's 86.4%; zero-shot questions are answered almost as well as I.I.D. ones (87.5% against 90.0% with 9B), consistent with the micro-track not requiring a matching training structure, while compositional questions remain the hardest (Appendix Tables 6 and 8).

Configuration	CWQ		WebQSP		GrailQA		Overall	
	H@1	F1	H@1	F1	H@1	F1	H@1	F1
Full system (Qwen3.5-27B)	89.2	81.2	90.1	80.4	87.1	84.6	89.1	81.6
w/o micro-track†	88.7	81.3	88.8	79.1	75.7	72.5	86.6	79.3
w/o macro-track	84.1	72.3	87.7	75.8	86.0	82.9	85.3	75.0
w/o repair agent	88.8	78.6	88.8	78.0	86.8	83.4	88.5	79.2
w/o micro-track and repair	85.7	76.1	86.6	76.3	32.1	26.3	77.2	68.1
w/o macro-track and repair	81.7	62.5	87.9	71.9	85.2	82.0	83.9	68.1

Table 2: Stage-level ablations on Qwen3.5-27B (Hits@1 and Macro-F1, %). Without repair, the selection is committed for every question; without a track, the other fills both windows (macro-track budget raised from 8 to 16). †Removed from initial generation only; repair can reuse its search. The rows do not isolate individual ontology rules.

Backbone	Selection only		Repair		Full system		Cost per question	
	H@1	F1	Routed	Fix / Brk	H@1	F1	Calls	Tokens
Qwen3.5-0.8B	76.5	65.8	5.6%	0 / 8	76.4	65.8	2.31	13,927 (1,425)
Qwen3.5-2B	80.4	71.0	10.2%	4 / 17	80.2	71.1	2.75	14,326 (1,806)
Qwen3.5-4B	85.2	76.2	24.6%	58 / 54	85.2	77.5	3.59	15,178 (2,244)
Qwen3.5-9B	87.0	78.0	23.6%	87 / 50	87.6	80.0	3.55	15,238 (2,191)
Qwen3.5-27B	88.5	79.2	26.7%	92 / 53	**89.1**	**81.6**	3.38	15,224 (2,080)
Qwen2.5-7B	83.6	74.1	8.1%	15 / 35	83.3	74.7	2.63	13,391 (1,585)

Table 3: Scaling across backbones on all 6,159 questions (Hits@1 and Macro-F1, %; same pipeline for every backbone). *Selection only* commits the selected program; *Repair* gives the share of questions routed ($\kappa \leq 4$) and the questions whose Hits@1 repair fixes or breaks. Calls include the sketch; tokens are counted once per question (re-sent history is not recounted), and parentheses exclude the fixed prompt prefix.

4.3 Component Ablation Studies (RQ2)

Table 2 removes component packages from the full system on Qwen3.5-27B: the micro-track, the macro-track, the repair agent, or a track together with repair.

The Two Tracks Are Complementary. Removing either track lowers accuracy, and the two tracks fail on different questions. Without the macro-track, Hits@1 drops from 89.1% to 85.3% and Macro-F1 from 81.6% to 75.0%, mostly on CWQ (-5.2 Hits@1) and on I.I.D. questions of GrailQA (-7.5), whose structures occur in training; zero-shot questions improve slightly (92.0% against 90.4%). Without both the micro-track and repair, zero-shot accuracy falls to 8.4% and compositional accuracy to 27.3%. With repair but without initial micro-track generation, zero-shot accuracy recovers to 74.2% because Revise-Sketch reruns that search: it supplies 513 of the 650 programs that repair turns correct.

Repair. Committing the selection for every question gives 88.5% Hits@1 and 79.2% Macro-F1; repair, invoked on 26.7% of the questions, raises them to 89.1% and 81.6%. With 27B, Revise-Sketch fixes 22 questions and breaks 3; Add-Constraint contributes most of the Macro-F1 gain but has a mixed effect on Hits@1 (31 fixed, 23 broken), since a constraint grounded on the wrong property can remove a credited answer. Replacements accepted by the gate raise F1 relative to the default in 261 cases and lower it in 78, while blocked ones are balanced (63 against 73); removing the gate lowers Hits@1 and Macro-F1 by only 0.1 and 0.2 points (Appendix Tables 9 and 10).

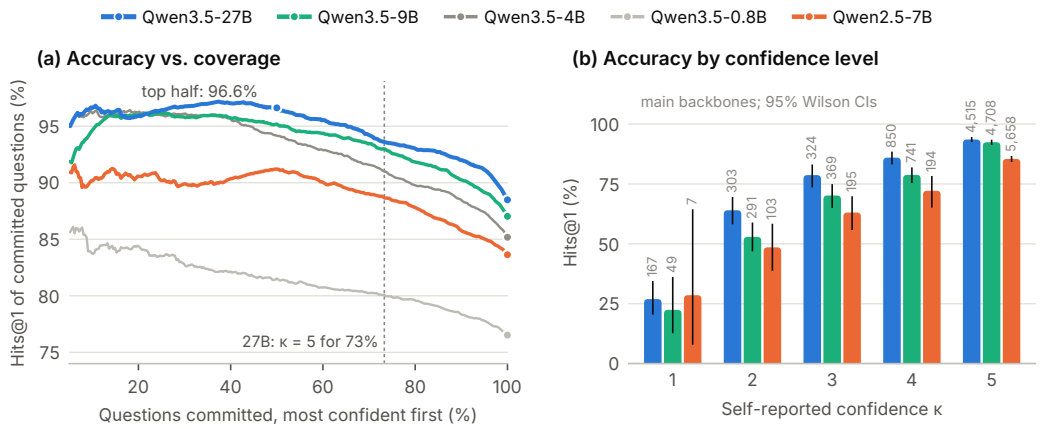


Figure 4: Confidence and selective accuracy of the selection stage (Hits@1). (a) Hits@1 of the top-ranked questions at each coverage, sorted by the expected level from the confidence-token probabilities (a plotting score, distinct from the routing threshold $\kappa = 5$). (b) Hits@1 per self-reported level, with sample counts above the bars.

4.4 Scaling and Cost (RQ3)

Scaling. Table 3 runs the identical pipeline with five Qwen3.5 backbones and Qwen2.5-7B. Among the Qwen3.5 backbones, selection accuracy grows monotonically with capacity, from 76.5% to 88.5% Hits@1. Repair is slightly harmful for the 0.8B and 2B backbones and for Qwen2.5-7B (0 fixed against 8 broken, 4 against 17, and 15 against 35) and net positive for Qwen3.5 from 4B on, raising Macro-F1 on the routed questions by 5.3, 8.6 and 8.8 points for 4B, 9B and 27B. Routing requires confidence that separates correct from incorrect selections: the 0.8B backbone reports the highest level on 94% of questions, so its routed set does not isolate many errors; from 4B on about a quarter of the questions are routed. Figure 4 shows that the reported level separates selections usefully: with Qwen3.5-27B, Hits@1 rises from 27.0% at $\kappa = 1$ to 93.6% at $\kappa = 5$ (Appendix D).

Cost. Every question incurs one sketch call and one selection call, while routed questions also incur repair turns and possibly pairwise comparisons: 2.8–3.7 LLM calls per question for the 9B and 27B backbones, against the 5.5–22.6 reported for exploration agents (Appendix Table 7). Under our accounting rule, which counts repeated conversation history only once per question, the totals are 14.4k–15.6k tokens, 2.2 to 4.4 times CoG with GPT-3.5; 1.7k–2.4k exclude the fixed prompt and demonstrations. Runtime is not measured; Appendix E gives case studies and errors.

5 Conclusion

We presented Ontos, an ontology-driven, candidate-first approach to KGQA: the LLM describes the intended reasoning at the conceptual level and evaluates executed candidates, while ontology-constrained synthesis handles the initial search over the graph. With 7B to 27B open-weight backbones, its results exceed the reported results of prior open-weight prompting methods on CWQ, WebQSP and GrailQA, and those of GPT-4-based agents on CWQ; within-system ablations show that the two synthesis tracks contribute complementary candidates.

Limitations and Future Work. High answer-hit coverage of the candidate set does not by itself ensure reliable repair of the selected program. Our repair stage relies on a fixed set of ontology-grounded tools and on the LLM's decisions about error diagnosis, tool use, and termination. Under the current configuration, repair has a negative net effect on Hits@1 for Qwen3.5-0.8B, Qwen3.5-2B, and Qwen2.5-7B. Future work could investigate reinforcement learning on training questions to improve the repair policy, with objectives that reward answer-set improvements, penalize harmful replacements, and account for tool-use costs.

AI USE STATEMENT

We used generative AI tools to assist with literature retrieval, reference checking and language editing. Under the authors' direction, they also assisted with parts of the implementation, the experimental design and analysis, and the annotation of error cases in Appendix E. The research questions, the method and all experimental decisions are the authors'. Generative AI tools were not used to create data sets or to formulate or prove mathematical claims. The authors checked reported numbers against run result files, assessed code changes by rerunning affected stages, checked references against publishers' records, and labelled nine calibration cases of the error audit in Appendix E and spot-checked the labels that LLM agents assigned to the remaining cases under a fixed taxonomy. The Qwen models used as backbones of ONTOS are components of the studied method, not writing aids. The authors take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

The backbones are open-weight Qwen models served with vLLM, embeddings come from jina-embeddings-v5, and the knowledge graph is a public Freebase dump queried through Virtuoso. Evaluation covers the 6,159 questions in our CWQ, WebQSP and GrailQA evaluation sets. Section 3 and Appendix A describe the stages and hyperparameters; Appendix B shows excerpts of the system prompts, while Appendix C defines the decoding settings, query budgets, metrics and token accounting. Algorithms 1 and 2 give pseudocode. Every LLM call uses greedy decoding under a JSON schema. The planned supplementary material contains per-question predictions of the backbones and ablations, from which the reported metrics can be recomputed against the evaluation gold answers. The current manuscript does not include the complete prompts, few-shot demonstrations, JSON schemas or implementation code; these are planned for release upon publication.

REFERENCES

Mohammad Kalim Akram, Saba Sturua, Nastia Havriushenko, Quentin Herreros, Michael Günther, Maximilian Werk, and Han Xiao. jina-embeddings-v5-text: Task-targeted embedding distillation. *arXiv preprint arXiv:2602.15547*, 2026.

Dean Allemang and Juan Sequeda. Increasing the accuracy of LLM question-answering systems with ontologies. In *The Semantic Web – ISWC 2024*, volume 15233 of *Lecture Notes in Computer Science*, pp. 324–339. Springer, 2024.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, 2013.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1247–1250, 2008.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Yong Cao, Xianzhi Li, Huiwen Liu, Wen Dai, Shuai Chen, Bin Wang, Min Chen, and Daniel Hershcovich. Pay more attention to relation exploration for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 2119–2136, 2023.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. In *Advances in Neural Information Processing Systems*, 2024.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. XGrammar: Flexible and efficient structured generation engine for large language models. In *Proceedings of Machine Learning and Systems*, volume 7, 2025.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. Beyond IID: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference*, pp. 3477–3488, 2021.

Yu Gu, Xiang Deng, and Yu Su. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 4928–4949, 2023.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Jinhao Jiang, Kun Zhou, 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, 2023a.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations*, 2023b.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. KG-Agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9505–9523, 2025. doi: 10.18653/v1/2025.acl-long.468.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 6966–6980, 2023a.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 6966–6980, 2023b.

Can Lin, Zhengwang Jiang, Ling Zheng, Qi Zhao, Yuhang Zhang, Qi Song, and Wangqiu Zhou. RJE: A retrieval-judgment-exploration framework for efficient knowledge graph question answering with LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17277–17294, 2025.

Ben Liu, Jihai Zhang, Fangquan Lin, Cheng Yang, Min Peng, and Wotao Yin. SymAgent: A neural-symbolic self-learning agent framework for complex reasoning over knowledge graphs. In *Proceedings of the ACM on Web Conference 2025*, pp. 98–108, 2025a.

Runxuan Liu, Bei Luo, Jiaqi Li, Baoxin Wang, Ming Liu, Dayong Wu, Shijin Wang, and Bing Qin. Ontology-guided reverse thinking makes large language models stronger on knowledge graph question answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15269–15284, 2025b.

Shicheng Liu, Sina J. Semnani, Harold Triedman, Jialiang Xu, Isaac Dan Zhao, and Monica S. Lam. SPINACH: SPARQL-based information navigation for challenging real-world questions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15977–16001, 2024.

Yuanxiang Liu, Songze Li, Xiaoke Guo, Zhaoyan Gong, Qifei Zhang, Huajun Chen, and Wen Zhang. CoG: Controllable graph reasoning via relational blueprints and failure-aware refinement over knowledge graphs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15547–15568, 2026. URL https://aclanthology.org/2026.acl-long.708/.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. *International Conference on Learning Representations*, 2024.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Yuan-Fang Li, Chen Gong, and Shirui Pan. Graph-constrained reasoning: Faithful reasoning on knowledge graphs with large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 41540–41565, 2025.

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.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1400–1409, 2016. doi: 10.18653/v1/D16-1147.

Qwen Team. Qwen3.5. https://huggingface.co/Qwen/Qwen3.5-9B, 2026.

Yongxue Shan, Jie Peng, Zixuan Dong, Fei Hu, and Xiaodong Wang. Reasoning with ontology graph: Toward type-constrained knowledge graph question answering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10684–10697, 2026. URL https://aclanthology.org/2026.acl-long.489/.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 641–651, 2018.

Sebastian Walter and Hannah Bast. GRASP: Generic reasoning and SPARQL generation across knowledge graphs. In *The Semantic Web – ISWC 2025*, volume 16140 of *Lecture Notes in Computer Science*, pp. 271–289. Springer, 2025.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. Knowledge-driven CoT: Exploring faithful reasoning in LLMs for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Guanming Xiong, Junwei Bao, and Wen Zhao. Interactive-KBQA: Multi-turn interactions for knowledge base question answering with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10561–10582, 2024.

An Yang, Baosong Yang, Beichen Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Ruize Men, Ruobin Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Yang Zhang, Yichang Wan, Yudong Liu, Zeyi Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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*, 2023.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1321–1331, 2015. doi: 10.3115/v1/P15-1128.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 201–206, 2016.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Yang Wang, Zhiguo Wang, and Bing Xiang. DECAF: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*, 2023.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2023.

A Method Details

A.1 Algorithms

Algorithm 1 summarizes the pipeline of Section 3, and Algorithm 2 details the micro-track. LLM calls are marked with the subscript LLM; all other steps are deterministic and query the graph or the ontology.

A.2 Micro-Track

Anchors. The search starts from the topic entity of each sketch branch; topic entities are the identifiers released with each question. When the sketch refers to a concept rather than an entity (e.g., "*Male*" or "*Degree*"), the concept is not used as a root, since starting a search from a type node would enumerate all of its instances. Numeric values of the question are bound as literals.

Admissible Relations. At a frontier node, the incident relations are retrieved in both directions and filtered by four rules derived from the ontology: relations outside the schema vocabulary or in non-business namespaces (`base.`, `user.`, `key.`) are removed; system relations (`type.`, `common.topic.`, `kg.object_profile.`) are removed; from the second hop on, the previous relation and its inverse are removed; and each inverse pair $(r, \mathrm{inv}(r))$ is folded into one candidate, keeping the direction with the higher score.

Algorithm 1 ONTOS

Require: question q, topic entities E_q, graph $\mathcal{G}$, ontology $\mathcal{O}$, training templates $\mathcal{T}$
$S = (c_{\text{ans}}, \{b_i\}, \Phi, \text{op}) \leftarrow \textsc{Sketch}_{\text{LLM}}(q, E_q)$ ▷ Section 3.2
$C_{\text{mi}} \leftarrow \textsc{MicroTrack}(S, E_q)$ ▷ Algorithm 2
$C_{\text{ma}} \leftarrow$ templates of $\mathcal{T}$ retrieved by question similarity, type-compatible with E_q under $\mathcal{O}$, executed in order until 8 are non-empty, ranked by $\text{Score}(T)$
for each program $\mathcal{P} \in C_{\text{mi}} \cup C_{\text{ma}}$ **do** ▷ Appendix A.4
    ground the constraints of Φ not expressed by $\mathcal{P}$ on the node whose type carries the property
end for
$C \leftarrow$ merge programs with identical denotations, interleave C_{mi} and C_{ma}; keep the first 16
$W_1 \leftarrow C[1..8]$; $W_2 \leftarrow C[9..16]$
$(A, \kappa) \leftarrow \textsc{Select}_{\text{LLM}}(q, E_q, W_1)$ ▷ $A \in W_1, \kappa \in \{1, \ldots, 5\}$
if $\kappa = 5$ **then return** $[\![A]\!]_{\mathcal{G}}$
$A' \leftarrow A$ completed with the branches and constraints of S it does not realize
$\mathcal{P}_{\text{def}} \leftarrow A'$ if $[\![A']\!]_{\mathcal{G}} \neq \emptyset$ and A' differs from every card, else A
$\mathcal{P}_{\text{edit}} \leftarrow \textsc{EditAgent}_{\text{LLM}}(q, S, \mathcal{P}_{\text{def}}, W_1, W_2)$ ▷ diagnosis, ≤ 6 turns of ontology-grounded tools
if $\mathcal{P}_{\text{edit}} \neq \mathcal{P}_{\text{def}}$ **and** $\textsc{Cmp}_{\text{LLM}}(\mathcal{P}_{\text{def}}, \mathcal{P}_{\text{edit}}) = \mathcal{P}_{\text{edit}}$ **and** $\textsc{Cmp}_{\text{LLM}}(\mathcal{P}_{\text{edit}}, \mathcal{P}_{\text{def}}) = \mathcal{P}_{\text{edit}}$ **then return** $[\![\mathcal{P}_{\text{edit}}]\!]_{\mathcal{G}}$ ▷ pairwise gate
return $[\![\mathcal{P}_{\text{def}}]\!]_{\mathcal{G}}$

Algorithm 2 MICROTRACK: hop-by-hop search under the ontology

Require: sketch S, topic entities E_q, beam width K ($= 128$)
for each branch $b_i = (p_1, \ldots, p_m)$ of S with anchor $e_i \in E_q$ **do**
    $B \leftarrow \{$the empty path at $e_i\}$
    for $j = 1, \ldots, m$ **do**
        $B' \leftarrow \emptyset$
        for each path $\pi \in B$ with frontier node v **do**
            $R \leftarrow$ relations incident to v in both directions, without relations outside the schema vocabulary, system relations and the inverse of the last step of π; each inverse pair folded into one
            for each $r \in R$ **do**
                if $\text{rng}(r) \in \mathcal{C}_{\text{CVT}}$ **then**
                    for each declared role ρ of $\text{rng}(r)$ **do**
                        add $\pi \cdot (r, \rho)$ to B' with hop score $s(\rho, p_j)$
                    end for
                else
                    add $\pi \cdot r$ to B' with hop score $s(r, p_j)$
                end if
            end for
        end for
        $B \leftarrow$ the K paths of B' with the highest mean hop score
    end for
    $P_i \leftarrow B$
end for
join secondary branches onto the paths of the main branch by a reverse join from each secondary entity; apply op
compile each path into SPARQL and execute it on $\mathcal{G}$; drop empty results and merge identical denotations
return the executed programs, in score order

CVT Expansion and Scoring. If a relation leads into a CVT type, the CVT's declared role relations are expanded, and the entry relation together with one role forms a single semantic hop whose score is that of the role relation. A path's score is the mean of its semantic-hop scores. The beam keeps at most $K = 128$ paths globally, with at most eight children per parent at later hops; candidates are ranked by path score, and branches are pruned by score before any materialization query is issued.

Secondary Anchors and Operators. A secondary topic entity is attached to a path by querying the relations incident to the entity itself and intersecting the resulting nodes with the path's frontier; candidates that close such a join with a non-empty denotation receive a score bonus of 0.08. The sketch operator is applied according to the answer concept: a count is projected only when the

answer concept is numeric, so that a comparative question mistakenly labelled as *count* keeps its entity answers; *argmax* and *argmin* order the answers by the grounded literal.

Execution Budget. Every query carries a server-side execution budget of 10 seconds. A query that exceeds it is recorded as a timeout for that query and budget, and lower-ranked variants of the same branch are skipped.

A.3 MACRO-TRACK

Templates are the programs of training questions with their entities replaced by slots. Retrieval compares the embedding of the test question with those of the training questions. Before execution, a template is checked against the ontology: the declared domain and range of every relation next to a slot must be compatible with the types of the entity that fills it, including supertypes, and slots whose relation has an enumeration or type range are treated as schema constants rather than entities. Templates are then instantiated and executed in retrieval order, stopping when eight have a non-empty denotation or after 300 templates. With Qwen3.5-27B, 49.4% of the 518k checked templates are removed by the ontology check before any query is issued. The eight executed templates are ranked by

$$\mathrm{Score}(T) = \tau \log \sum_{q' \in \mathcal{N}_T(q)} \exp\big(\mathrm{sim}(q, q')/\tau\big) + \lambda\, \mathrm{Align}(T, S), \tag{3}$$

where $\mathcal{N}_T(q)$ are those of the 300 training questions most similar to q that use T, so that a template supported by several close questions ranks high, and $\mathrm{Align}(T, S)$ is the fraction of sketch hops matched by a relation of T ($\tau = 0.05$, $\lambda = 0.15$).

A.4 CONSTRAINT GROUNDING

A scalar constraint is frequently not a property of the answer node itself: the year of an office term is recorded on a CVT node, and a population figure is reached from a city through a supertype and a further CVT node. For a candidate program $\mathcal{P}$ that contains no comparison, every variable v of $\mathcal{P}$ is assigned the types $\tau(v)$ implied by the schema domains and ranges of its incident relations, their supertypes, and the types v carries in the graph. For a constraint $\phi = (a, \circ, x)$ we enumerate the literal-valued property paths π reachable from v in one hop, or in two hops through a CVT node, whose range matches the datatype of x; rank the pairs (v, π) by lexical overlap with the attribute a and with the question, then by embedding similarity; and select the first pair whose path is non-empty under $\mathcal{P}$'s own constraints:

$$\mathcal{P}' = \mathcal{P} \wedge \big(v^*\ \pi^*\ ?z\big) \wedge \mathrm{FILTER}\big(?z \circ x\big). \tag{4}$$

Equality on the year of an interval-valued property is compiled with interval semantics (the term starts no later than the end of year x and ends no earlier than its start, or has no end). Grounding issues type and existence probes whose cost grows with the size of $\mathcal{P}$'s intermediate results, so on the micro-track it is applied only to the 16 top-ranked candidates, the number consumed by fusion; the grounded program is inserted before its unconstrained parent, which is kept. On the macro-track, each instantiated template with a non-empty denotation is grounded in the same way and replaced when the grounded program remains non-empty. Entity-valued constraints are attached directly or through a CVT mediator, and only to variables whose range is not a literal type.

A.5 CARDS

A card renders a program as (i) its triple patterns in execution order, with the answer variable `?answer`, CVT nodes on the searched path `?cvt_node_X`, other intermediate nodes (including CVT nodes added by constraint grounding) `?node_X` and literal values `?value_X`, and with topic entities shown as `[E1: name]`; (ii) its visible filters, orderings and aggregates in SPARQL syntax; and (iii) the names of up to five returned answers with the count returned by the executed query (Figure 5). Each card is tagged with its origin (macro-track, micro-track, or both). Programs outside the normalizable fragment, such as `UNION` or nested sub-queries of some templates, are shown as the executed SPARQL with entity identifiers replaced by their slots and names.

Executed SPARQL (merged program of both tracks)

```
SELECT DISTINCT ?x2 WHERE {
  VALUES ?x0 { ns:m.07ldhs }
  ?x0 ns:film.actor.film ?x1 .
  ?x1 ns:film.performance.film ?x2 .

  ?x2 ns:film.film.release_date_s ?scm0 .
  ?scm0 ns:film.film_regional_release_date.release_date
        ?sc0 .
  FILTER(((STRLEN(STR(?sc0)) >= 10 &&
    SUBSTR(STR(?sc0), 1, 10) < "2008-12-25") ||
    (STRLEN(STR(?sc0)) < 10 &&
    xsd:integer(SUBSTR(STR(?sc0), 1, 4)) <= 2008)))
} LIMIT 128
```

render

Card shown to the LLM at selection

```
Option 5 [CONSENSUS]:
- Graph Pattern (W3C Triples):
  * [E1: Taylor Lautner] ns:film.actor.film ?cvt_node_1 .
  * ?cvt_node_1 ns:film.performance.film ?answer .
  * ?answer ns:film.film.release_date_s ?node_1 .
  * ?node_1 ns:film.film_regional_release_date.release_date
      ?value_1 .
- Modifiers & Filters (W3C SPARQL Syntax):
  * FILTER(((STRLEN(STR(?value_1)) >= 10 &&
      SUBSTR(STR(?value_1), 1, 10) < "2008-12-25") ||
      (STRLEN(STR(?value_1)) < 10 &&
      xsd:integer(SUBSTR(STR(?value_1), 1, 4)) <= 2008)))
- Execution Preview:
  * ['Cheaper by the Dozen 2', 'Shadow Fury',
     'The Adventures of Sharkboy and Lavagirl', 'Twilight']
     (Total: 4 items)
```

Renaming
same colour on both sides

- `?x0` → `[E1: Taylor Lautner]` — *topic entity slot*
- `?x1` → `?cvt_node_1` — *CVT node*
- `?x2` → `?answer` — *answer*
- `?scm0` → `?node_1` — *intermediate node*
- `?sc0` → `?value_1` — *literal value*

Figure 5: An executed SPARQL candidate (top) and its normalized selection card (bottom) for the question of Figure 3. Variables are renamed by role, as listed on the right; the card adds an origin tag and an answer preview. The execution query's LIMIT 128 is not shown among the card modifiers, so the displayed Total is the count returned by that execution rather than a claim about the unlimited answer set. Literals with only a year are compared by year (year $\leq$ 2008).

A.6 Repair Agent

Per question, the tool budgets are Add-Constraint 4, Apply-Sketch-Constraints 2, Extend 3, Revise-Sketch 1 and Browse-More 2 (four cards per call); a tool whose budget is exhausted is removed from the output schema, and the sixth turn allows only Commit. Add-Constraint accepts the operators $=, <, \leq, >, \geq$ and current, latest, earliest. Extend runs a one-hop beam of width 16 from the answer node and recompiles the extended program so that anchor joins and constraints are re-attached; Revise-Sketch reruns the micro-track with a beam of 32. Within one tool call, once three probes have been reported by the endpoint as exceeding their budget, the remaining probes are skipped. The comparator of the pairwise gate uses a short system prompt (Appendix B) and sees only the question, the topic entities and the two cards.

B Prompts

All LLM calls use a system prompt followed by few-shot demonstrations, and their outputs are constrained by JSON schemas. We show excerpts of the system prompts below; the text is reproduced as used, except for light reformatting and for internal card-tag and tool names, which are replaced by the names used in this paper; omissions are marked [. . .]. The complete prompts and demonstrations will be released with the code. The tag [CONSENSUS] marks a card produced by both tracks.

Abstract Derivation Sketch.

```
# Task: Question -> Typed Derivation Sketch
You translate a natural-language question and its candidate topic entities into a
typed derivation sketch over a Freebase-style knowledge graph. [...]

## Output Schema
{ "answer":   {"concept": "<target entity type or concept>",
               "operator": "none | count | argmax | argmin"},
  "branches": [{"entity": "E1",
                "hops": [{"relation": "<relation phrase>",
                          "target_concept": "<concept reached by this hop>",
                          "intent": "<role connected by this hop>"}]}],
  "filters":  [{"attribute": "<scalar property>", "operator": "= | != | < | <= | > | >=",
                "value": "<literal date or number string>"}] }
[...]
* entity: Reference the entity alias strictly as "E1", "E2", etc. NEVER copy internal
  Freebase MIDs. [...]
* target_concept: The concept reached after this hop. All branches must converge to the
  same final target concept!
* Directionless Principle: DO NOT specify physical traversal directions (such as forward
  or reverse). [...]
* Universal Semantic Folding (CVT Events): Compound Value Type (CVT) intermediary
  structures (e.g. marriage events, sports team rosters, employment tenures) represent a
  single real-world business action. [...]
* filters: Used ONLY for strict numerical, date, or scalar value restrictions explicitly
  mentioned in the question. [...]
```

Confidence-Guided Selection.

```
# Task: Select the Best Query Program and State Your Confidence
You are an expert verifier choosing the single best executable query program for a
question over a knowledge graph. You are given a natural language question, the topic
entities with fixed slots ([E1], [E2], ...), and up to eight candidate programs [...].
You must always choose exactly one candidate, and state how confident you are that it
answers the question.

## 1. Candidate Card Structure
- Graph Pattern: the executed program as RDF triples; ?answer is the returned variable,
  intermediate entities are ?node_X, reified event nodes are ?cvt_node_X [...]
- Modifiers & Filters: FILTER, ORDER BY ... LIMIT, and aggregates in SPARQL syntax [...]
- Execution Preview: real entity names returned by executing the program [...]
Each option is tagged with where it came from: [MACRO_TRACK], [MICRO_TRACK] or
[CONSENSUS]. [...]

## 2. Graph Pattern Verification Protocol
- Trace the variable bindings from the topic entities through intermediate nodes to the
  final target variable ?answer, and check that ?answer has the type the question
  asks for. [...]
- When a question requires temporal boundaries or extremum semantics, check that the
  corresponding FILTER or ORDER BY clause is present. [...]

## 3. Choose One Option and Rate Your Confidence
Always choose the option most likely to return the correct answer, even when every option
has flaws: an imperfect option whose answers include the right entity is better than none.
- "5": the graph pattern, constraints and modifiers fully express the question, and the
  returned answers have the requested type and fit the question.
- "4": the answer type and main relation chain are right; a minor doubt remains [...]
- "3": plausible but with a notable doubt [...]
- "2": the best available, but probably wrong or incomplete [...]
- "1": no option fits the question; this is only the closest.
Output: {"reason": "...", "choice": "Option k", "confidence": "1" | ... | "5"}
```

Program Editing Agent.

```
# Task: Knowledge-Graph Program Repair Agent
You answer a question over Freebase by repairing a query program. A single-pass reviewer
already chose an option but was not confident, so the question was sent to you. [...]
Your goal is the program whose answer set is exactly what the question asks: every asked
entity, and nothing else. [...]
Phase 1 -- Diagnose (one reply, no tools). [...]
Phase 2 -- Repair (a few turns). Use the tools to build a better program, then commit the
best option in the pool. [...]

| diagnosis            | Tool                                              |
| missing_constraint   | Add-Constraint (or Apply-Sketch-Constraints)      |
```

```
| one_hop_short        | Extend                                                |
| wrong_relation_chain | Revise-Sketch                                         |
| wrong_anchor_entity  | Revise-Sketch                                         |
| answer_not_in_pool   | Browse-More                                           |

You never write predicate URIs. You describe the intent in plain words; the system grounds
it on the real graph using the Freebase ontology, executes the program, and adds any new
non-empty program to the pool as a new option. [...]
Output: {"thought": "...", "action": {"name": "<action>", "arguments": {...}}}
```

Pairwise Replacement Gate.

```
You compare two query programs that were executed on Freebase for the same question. Each
card shows the program's graph pattern (W3C triples), its filters, and the answers it
actually returned with the total count.
Pick the program whose answer set is exactly what the question asks: every asked entity
and nothing else. Check that '?answer' is the kind of thing asked, that every constraint
in the question (other entities, years, 'current', 'first/last', counts) is expressed, and
that no constraint was added that the question does not state. If both programs are
equally good, pick Program 1.
Output one JSON object: {"thought": "<at most 2 sentences, single quotes only>", "choice":
"Program 1" or "Program 2"}.
```

C Experimental Details

Dataset	Test	Official groups (questions)
CWQ (Talmor & Berant, 2018)	3,531	composition (1,546), conjunction (1,575), comparative (213), superlative (197)
WebQSP (Yih et al., 2016)	1,628	–
GrailQA (Gu et al., 2021)	1,000	I.I.D. (240), compositional (198), zero-shot (562)
Total	6,159	

Table 4: Evaluation sets used in this paper (6,159 questions total). All are answered over Freebase (Bollacker et al., 2008) using the topic entities supplied with the questions. The GrailQA row denotes the 1,000-question evaluation set used here and its three generalization groups.

Baselines. Baseline results are taken from the original papers, grouped as in CoG (Liu et al., 2026); the Qwen2.5-7B results of ToG, PoG and CoG are those reported by CoG. The direct-answer results of GPT-4o, DeepSeek-V3 and Qwen-Max and the results of SymAgent, GNN-RAG, GCR, ORT and OntGQA are taken from OntGQA (Shan et al., 2026) and ORT (Liu et al., 2025b). These are literature-reported values, not baselines rerun under our exact graph snapshot, prompts and decoding setup.

Topic Entities. Every benchmark question is released with its topic entities as Freebase identifiers, and the agentic baselines we compare with start their exploration from these given entities rather than from an entity linker of their own. We follow the same protocol: the given entities fill the entity slots of the sketch, and no entity linking is performed. The given anchors are not always complete; a missing entity is recovered only as a constraint on a candidate program (Sections 3.3 and 3.4): an exact-name lookup proposes nodes carrying the name the sketch gives it, and a node is admitted only if it is connected to a variable of the program directly or through a CVT mediator. A recovered entity thus only narrows answers the program already reaches and never serves as a new starting point.

Implementation. All stages use the same LLM, served by vLLM (Kwon et al., 2023) on one H200 GPU with structured decoding by XGrammar (Dong et al., 2025) and greedy decoding (temperature 0); outputs are streamed and accepted only when generation stops normally and the output satisfies its schema. Relation and question embeddings are produced by jina-embeddings-v5-text-small-retrieval (Akram et al., 2026). The graph is a Freebase (Bollacker et al., 2008) dump served by Virtuoso; every query carries a server-side execution budget (10 s in the micro-track, 30 s for repair

tools, 60 s for deterministic sketch completion), and a query reported as exceeding it is treated as a timeout for that query and budget, not as an empty result.

Metrics. *Hits@1* is computed with the same evaluation code as ToG, PoG, CoG and RJE (Lin et al., 2025). Gold answers are represented as in CoG and RJE (answers with aliases for CWQ and WebQSP; entity names for GrailQA). A prediction counts as a hit when its answer string—a single answer, or the string form of the answer list when several answers are returned—matches a gold answer under case- and whitespace-insensitive bidirectional containment. *Macro-F1* is computed at the set level: predictions and gold answer groups are matched one-to-one by maximum bipartite matching under that same containment rule, giving precision over predictions and recall over answer groups, averaged over questions.

Cost. LLM calls include the sketch, all selection attempts (including re-asks after an overlong output), every agent turn and every comparison of the gate. Our token figure counts conversation history only once per question even when later turns resend it. *Unique* tokens additionally exclude the fixed system prompt and demonstrations. These are content-accounting measures; they do not measure all transmitted tokens, cache savings, graph-query costs or end-to-end latency.

D Additional Results

Confidence and Selective Accuracy. Routing uses the self-reported integer κ, whereas Figure 4(a) orders questions by a finer score. For the first token of the confidence field, let ℓ_j be the recorded log probability of level $j \in \{1, \ldots, 5\}$; over the available level tokens, set $p_j = \exp(\ell_j) / \sum_k \exp(\ell_k)$ and $s = \sum_j j p_j$. The curve sorts by decreasing s, then decreasing κ; exact ties retain input order, and missing token probabilities receive the lowest plotting score. Thus the top half can be selected within the large $\kappa = 5$ group. With Qwen3.5-27B, the top-ranked half reaches 96.6% Hits@1 and the top 90% reaches 92.0%; accuracy by discrete level rises from 27.0% at $\kappa = 1$ to 93.6% at $\kappa = 5$. These results show useful ranking for selective prediction, not probability calibration. Qwen2.5-7B reports $\kappa = 5$ for 5,658 of the 6,159 questions, leaving few errors in its routed set (Table 3).

Question Types. Table 8 breaks down the ONTOS results by the official question groups. Conjunctions, which require joining a second entity, are answered best (91.6% Hits@1 with 9B), and repair adds most here: 32.1% of these questions are routed, and Macro-F1 rises from 80.6% to 85.1%. The gain comes mostly from adding what the selected program lacks rather than from choosing another candidate: of the 71 question-equivalents of F1 gained, 51 come from programs produced by ADD-CONSTRAINT (32.0) or by deterministic sketch completion (18.6).

Inside the Repair Stage. With Qwen3.5-9B, 1,451 questions (23.6%) are routed to repair. Deterministic sketch completion yields a new default program for only 48 of them, because most sketch constraints are already grounded during candidate generation. The agent commits voluntarily on 582 questions and at the turn limit on 627; on 27 its output is invalid and the default is kept. It proposes to replace the default on 543 questions, and the pairwise gate accepts 329 of these replacements. Relative to selection alone, repair turns 87 questions correct and 50 incorrect under Hits@1 (Table 3).

Table 9 gives the full breakdown for the 9B and 27B backbones. Edits that change the structure of the program or bring in new programs are clearly net positive: extending by one hop fixes 24 questions and breaks 7, revising the sketch fixes 23 and breaks 8, and browsing further candidates fixes 11 and breaks 2. Two kinds of change are not: re-choosing a card from the initial pool (11 fixed, 13 broken), which repeats the decision of the selection stage, and adding a constraint (16 fixed, 19 broken). A constraint narrows the answer set; it raises Macro-F1 when it is grounded on the right property (it is the largest source of the F1 gain on conjunctions, Appendix D), but it loses a Hits@1 that the wider answer list was credited with when it is grounded on the wrong one. The agent's own diagnoses agree: questions diagnosed as *one hop short* are fixed 25 times and broken 3 times, *wrong relation chain* 34 and 17, and *missing constraint* 27 and 30.

Table 10 varies the pairwise gate and the routing threshold on Qwen3.5-27B. Removing the pairwise gate lowers both metrics (89.0% Hits@1, 81.4% Macro-F1, against 89.1% and 81.6%). Repairing

Method	Backbone	CWQ		WebQSP		GrailQA
		Hits@1	F1	Hits@1	F1	Hits@1
LLM only (no knowledge graph)						
IO prompt (Brown et al., 2020)	GPT-3.5	37.6	–	63.3	–	29.4
CoT (Wei et al., 2022)	GPT-3.5	38.8	–	62.2	–	28.1
Self-consistency (Wang et al., 2023b)	GPT-3.5	45.4	–	61.1	–	29.6
Direct answer	GPT-4o	38.2	32.9	61.8	43.6	–
Direct answer	DeepSeek-V3	41.1	33.8	64.0	43.9	–
Direct answer	Qwen-Max	36.4	29.5	59.0	40.0	–
Fine-tuned KG-augmented methods						
RE-KBQA (Cao et al., 2023)	–	50.3	–	74.6	–	–
UniKGQA (Jiang et al., 2023b)	–	51.2	48.0	77.2	72.2	–
DECAF (Yu et al., 2023)	–	70.4	–	82.1	–	–
KG-Agent (Jiang et al., 2025)	–	72.2	–	83.3	–	–
RoG (Luo et al., 2024)	Llama-2-7B	62.6	56.2	85.7	70.8	–
GNN-RAG (Mavromatis & Karypis, 2025)	Llama-2-7B	66.8	59.4	85.7	71.3	–
SymAgent (Liu et al., 2025a)	Llama-2-7B	35.1	31.2	55.5	41.3	–
GCR (Luo et al., 2025)	Llama-2-7B+GPT-4o	66.0	58.5	87.5	71.5	–
OntGQA (Shan et al., 2026)	Llama-2-7B	74.6	56.8	**91.5**	77.1	–
OntGQA (Shan et al., 2026)	Llama-2-7B+GPT-4o	73.8	55.8	**91.5**	76.6	–
Training-free prompting, closed-source LLMs						
KB-BINDER (Li et al., 2023a)	Codex	–	–	–	74.4	–
StructGPT (Jiang et al., 2023a)	GPT-3	54.3	49.6	72.6	63.7	–
KD-CoT (Wang et al., 2023a)	GPT-3.5	50.5	–	73.7	50.2	–
ToG (Sun et al., 2024)	GPT-3.5	57.1	41.9	76.2	50.9	68.7
PoG (Chen et al., 2024)	GPT-3.5	63.2	43.7	82.0	58.1	76.5
CoG (Liu et al., 2026)	GPT-3.5	66.9	59.9	86.8	74.3	79.2
ToG (Sun et al., 2024)	GPT-4	67.6	47.6	82.6	58.9	81.4
PoG (Chen et al., 2024)	GPT-4	75.0	42.1	87.3	59.8	84.7
ORT (Liu et al., 2025b)	GPT-4o	65.4	58.7	87.7	71.8	–
CoG (Liu et al., 2026)	GPT-4	77.8	69.2	89.7	75.5	86.4
Training-free prompting, open-source LLMs						
ToG (Sun et al., 2024)	Qwen2.5-7B	42.5	28.7	56.0	37.3	62.6
PoG (Chen et al., 2024)	Qwen2.5-7B	46.0	31.4	58.5	40.4	68.9
CoG (Liu et al., 2026)	Qwen2.5-7B	54.0	48.5	74.5	63.9	72.0
ONTOS (ours)	Qwen2.5-7B	82.4	73.3	87.0	75.9	80.5
ONTOS (ours)	Qwen3.5-9B	88.4	80.4	87.5	78.3	85.1
ONTOS (ours)	Qwen3.5-27B	**89.2**	**81.2**	90.1	**80.4**	**87.1**

Table 5: Reported performance (%) on CWQ, WebQSP and GrailQA with all baselines (Table 1 shows a selection). Baseline values come from prior papers; Hits@1 uses the same evaluation code as these methods; our rows additionally report our own Macro-F1. Direct answer queries GPT-4o (Hurst et al., 2024), DeepSeek-V3 (DeepSeek-AI et al., 2024) or Qwen-Max (Yang et al., 2024) without the knowledge graph.

only questions with confidence $\kappa \leq 3$ routes 12.9% of the questions instead of 26.7% and gives slightly higher Hits@1 (89.3%) but lower Macro-F1 (81.2%): on questions with $\kappa = 4$, repair mainly improves the answer set of selections whose Hits@1 is already correct.

Of the 329 replacements the gate accepts, 206 raise the question's F1 and 72 lower it. Of the 214 it rejects, 89 would have raised it and 81 lowered it. The gate thus lets through replacements that are mostly improvements and blocks a set that is close to a coin flip; the price is 43 Hits@1 fixes blocked alongside 47 breaks prevented.

Method	Backbone	Overall	I.I.D.	Comp.	Zero-shot
LLM only (no knowledge graph)					
IO prompt (Brown et al., 2020)	GPT-3.5	29.4	–	–	–
CoT (Wei et al., 2022)	GPT-3.5	28.1	–	–	–
Self-consistency (Wang et al., 2023b)	GPT-3.5	29.6	–	–	–
Training-free prompting, closed-source LLMs					
ToG (Sun et al., 2024)	GPT-3.5	68.7	70.1	56.1	72.7
PoG (Chen et al., 2024)	GPT-3.5	76.5	76.3	62.1	81.7
CoG (Liu et al., 2026)	GPT-3.5	79.2	80.4	65.2	83.6
ToG (Sun et al., 2024)	GPT-4	81.4	79.4	67.3	86.5
PoG (Chen et al., 2024)	GPT-4	84.7	87.9	69.7	88.6
CoG (Liu et al., 2026)	GPT-4	86.4	88.3	**76.3**	89.1
Training-free prompting, open-source LLMs					
ToG (Sun et al., 2024)	Qwen2.5-7B	62.6	60.8	47.0	68.9
PoG (Chen et al., 2024)	Qwen2.5-7B	68.9	67.9	51.5	75.4
CoG (Liu et al., 2026)	Qwen2.5-7B	72.0	72.5	57.6	76.9
ONTOS (ours)	Qwen2.5-7B	80.5	**92.1**	60.6	82.6
ONTOS (ours)	Qwen3.5-9B	85.1	90.0	72.2	87.5
ONTOS (ours)	Qwen3.5-27B	**87.1**	90.8	73.2	**90.4**

Table 6: Hits@1 (%) on the GrailQA evaluation set by generalization level, with all baselines and the grouping of Table 1. Baseline values follow CoG (Liu et al., 2026); our rows report Hits@1 of the full system with the same evaluation code (overall Macro-F1: 77.3, 81.6, 84.6 for the 7B, 9B and 27B backbones). These are cross-paper reported values. Best value per column in bold.

Data	Method (backbone)	Calls	Input	Output	Total	Unique	H@1
CWQ	ToG (GPT-3.5)†	22.6	8,183	1,486	9,669	–	57.1
	PoG (GPT-3.5)†	13.3	7,803	353	8,156	–	63.2
	CoG (GPT-3.5)†	11.7	6,589	487	7,076	–	66.9
	Ours (Qwen3.5-9B)	3.73	14,912	637	15,549	2,380	88.4
	Ours (Qwen3.5-27B)	3.56	14,986	577	15,563	2,244	89.2
WebQSP	ToG (GPT-3.5)†	15.9	6,031	988	7,019	–	76.2
	PoG (GPT-3.5)†	9.0	5,235	283	5,518	–	82.0
	CoG (GPT-3.5)†	8.3	4,694	206	4,900	–	86.8
	Ours (Qwen3.5-9B)	3.47	14,458	503	14,961	2,016	87.5
	Ours (Qwen3.5-27B)	3.34	14,514	460	14,974	1,984	90.1
GrailQA	ToG (GPT-3.5)†	11.1	4,066	775	4,841	–	68.7
	PoG (GPT-3.5)†	6.5	3,373	203	3,576	–	76.5
	CoG (GPT-3.5)†	5.5	3,122	166	3,288	–	79.2
	Ours (Qwen3.5-9B)	3.08	14,129	463	14,592	1,810	85.1
	Ours (Qwen3.5-27B)	2.80	14,034	399	14,433	1,656	87.1

Table 7: Per-question cost by dataset. †Numbers as reported by Liu et al. (2026) (Table 4), with GPT-3.5 Turbo. Our tokens include the sketch call and count each token once per question, so the conversation history re-sent in every repair turn is not counted again; *Unique* further excludes the fixed prompt prefix (system prompts and demonstrations) that is identical across questions. H@1 of the baselines follows their papers.

		Qwen2.5-7B		Qwen3.5-9B		Qwen3.5-27B	
Group	n	H@1	F1	H@1	F1	H@1	F1
Full system							
CWQ	3,531	82.4	73.3	88.4	80.4	89.2	81.2
composition	1,546	81.8	72.6	85.9	77.0	87.1	77.8
conjunction	1,575	84.1	75.6	91.6	85.1	92.3	86.1
comparative	213	81.2	67.5	88.7	75.5	88.3	75.6
superlative	197	75.1	67.2	81.7	73.8	82.7	75.4
WebQSP	1,628	87.0	75.9	87.5	78.3	90.1	80.4
GrailQA	1,000	80.5	77.3	85.1	81.6	87.1	84.6
I.I.D.	240	92.1	90.0	90.0	88.7	90.8	88.7
compositional	198	60.6	51.8	72.2	63.0	73.2	67.6
zero-shot	562	82.6	80.9	87.5	85.2	90.4	88.8
All	6,159	83.3	74.7	87.6	80.0	89.1	81.6
Selection only (no repair)							
CWQ	3,531	82.7	72.7	87.9	77.9	88.8	78.6
composition	1,546	82.0	72.4	85.6	76.0	87.1	76.7
conjunction	1,575	84.3	74.4	90.2	80.6	91.2	81.7
comparative	213	81.2	67.5	90.1	76.0	86.4	72.5
superlative	197	76.7	67.1	84.3	73.2	86.3	74.5
WebQSP	1,628	87.8	75.5	87.2	77.0	88.8	78.0
GrailQA	1,000	80.3	76.7	83.8	80.1	86.8	83.4
I.I.D.	240	92.1	90.0	90.4	88.3	92.1	88.7
compositional	198	61.6	51.1	71.7	61.9	71.7	63.5
zero-shot	562	81.9	80.1	85.2	83.0	89.9	88.2
All	6,159	83.6	74.1	87.0	78.0	88.5	79.2
Share of questions routed to repair (%)							
CWQ	3,531	8.3		26.7		30.8	
composition	1,546	6.0		18.8		21.3	
conjunction	1,575	11.0		32.1		38.1	
comparative	213	5.2		37.6		39.0	
superlative	197	7.1		33.5		38.6	
WebQSP	1,628	7.5		21.3		24.4	
GrailQA	1,000	8.7		16.1		15.8	
I.I.D.	240	2.9		10.0		10.8	
compositional	198	11.1		27.3		32.3	
zero-shot	562	10.3		14.8		12.1	
All	6,159	8.1		23.6		26.7	

Table 8: Results by official question group (Hits@1 and Macro-F1, %).

	Qwen3.5-9B		Qwen3.5-27B	
	Fixed / Broken	ΔF1	Fixed / Broken	ΔF1
By the origin of the committed program				
Sketch completion (A')	1 / 1	+21.4	2 / 1	+24.5
ADD-CONSTRAINT	16 / 19	+55.1	31 / 23	+86.0
APPLY-SKETCH-CONSTRAINTS	1 / 0	+0.6	0 / 0	-0.3
EXTEND	24 / 7	+16.4	19 / 16	+1.7
REVISE-SKETCH	23 / 8	+18.0	22 / 3	+20.4
BROWSE-MORE	11 / 2	+10.4	9 / 6	+6.5
Re-selection from the initial pool	11 / 13	+3.3	9 / 4	+6.2
By the agent's diagnosis				
missing constraint	27 / 30	+87.1	34 / 30	+111.3
one hop short	25 / 3	+21.0	21 / 10	+9.5
wrong relation chain	34 / 17	+15.5	36 / 12	+24.8
wrong anchor entity	0 / 0	+1.0	1 / 0	+1.3
answer not in pool	1 / 0	+0.5	0 / 1	-2.0
All routed questions	87 / 50	+125.1	92 / 53	+145.0
	Better / Worse	Tie	Better / Worse	Tie
Pairwise gate (F1 of the agent's choice vs. the default)				
Accepted replacements	206 / 72	51	261 / 78	39
Blocked replacements	89 / 81	44	63 / 73	18

Table 9: Inside the repair stage. Upper part: questions whose Hits@1 the repair stage turns correct (fixed) or incorrect (broken) relative to the selection, and the change in summed F1 (question equivalents), by the origin of the committed program and by the diagnosis. Lower part: replacements proposed by the agent, split by whether the pairwise gate accepted them, and whether they raise or lower F1 relative to the default program.

	CWQ		WebQSP		GrailQA		Overall	
Configuration	H@1	F1	H@1	F1	H@1	F1	H@1	F1
Full system ($\kappa \leq 4$ routed, with gate)	89.2	81.2	90.1	80.4	87.1	84.6	89.1	81.6
Selection only (w/o repair)	88.8	78.6	88.8	78.0	86.8	83.4	88.5	79.2
w/o pairwise gate	89.3	81.2	89.6	79.9	87.1	84.4	89.0	81.4
Repair only if $\kappa \leq 3$	89.6	80.8	90.1	80.0	87.1	84.4	89.3	81.2
Repair only if $\kappa \leq 2$	89.6	80.1	89.9	79.4	87.4	84.5	89.3	80.6

Table 10: Design choices of the repair stage on Qwen3.5-27B (Hits@1 and Macro-F1, %). The full system routes questions with confidence $\kappa \leq 4$ (26.7%); routing only $\kappa \leq 3$ or $\kappa \leq 2$ routes 12.9% or 7.6%.

E CASE STUDIES AND ERROR ANALYSIS

Constraint Grounding During Synthesis. For "*What Taylor Lautner movies were released prior to Christmas Day 2008?*" the sketch contains one hop, *Starring* $\rightarrow$ (Film), and the constraint *release date* $<$ 2008-12-25. Around the anchor, the micro-track discards system and non-schema relations, folds each inverse pair into one candidate, and passes through the CVT `film.performance` so that entry edge and role edge form one hop. The correct hop, `performance` $\rightarrow$ `film`, ranks only fourth by semantic similarity (0.264, behind personal appearances and the actor role), but the beam keeps it. The date constraint is grounded on the film node through a second CVT, `film.film.release_date_s` $\rightarrow$ `film_regional_release_date.release_date`: a date-valued property path that the ontology makes reachable from the film type and that is non-empty under the program. The grounded program is produced by both tracks, selected with confidence 5, and returns exactly the four films released before that date.

Repair by Editing the Selected Program. For "*What movie did the actor who played the character Jim Lovell won his first Oscar from?*" the macro-track provides a program returning the two films for which Tom Hanks won an Academy Award, *Forrest Gump* and *Philadelphia*. The selection picks it with confidence 3, noting that nothing selects the *first* win, so the question is routed. The agent diagnoses a missing constraint and calls ADD-CONSTRAINT with attribute "award date" and operator EARLIEST; the attribute is grounded on the year of the award-honor CVT, and the edited program returns *Philadelphia*. The comparator prefers the edited program in both presentation orders, and it is committed. The question costs seven LLM calls; an unrouted question costs two.

Error Analysis. Table 11 partitions wrong final answers into metric-defined categories, judging candidates by Hits@1. For all backbones, 65%–74% of errors have a Hits@1-correct candidate in W_1 that is not finally chosen; for the 9B and 27B backbones this is about two thirds of errors, while about one fifth have no Hits@1-correct program among the 16 fused candidates. These categories do not identify the causal stage of failure: Hits@1 can credit a superset that merely includes a gold answer. The audit below examines this distinction in a sample. A qualitative grounding failure is a generic attribute such as *year* attached to a property of the right datatype but the wrong meaning, such as the year of a census record.

	Qwen3.5-9B		Qwen3.5-27B	
Hits@1 category	#	% of errors	#	% of errors
Hits@1-correct candidate in W_1, not selected	494	64.8	436	65.1
No Hits@1-correct program among the 16 fused programs	159	20.9	136	20.3
Repair breaks a Hits@1-correct selection	50	6.6	53	7.9
Hits@1-correct candidate only in reserve W_2	59	7.7	45	6.7
All errors (of 6,159 questions)	762	100.0	670	100.0

Table 11: Metric-defined categories of wrong final answers (Hits@1). Each wrong answer is assigned to the first applicable category, checked from repair back to generation. A Hits@1-correct program can return a superset containing a gold answer without faithfully expressing the question. These categories are descriptive, not a causal decomposition; the sampled audit in Table 12 adds a semantic assessment.

Error Audit. Following the audit categories of OntGQA (Shan et al., 2026), we sampled 100 Qwen3.5-27B wrong answers in proportion to the categories of Table 11 (Table 12). Each case was labelled under a fixed taxonomy using the question, gold query, candidate window and repair record. The resulting sample labels 40 cases as benchmark-confounded: 24 where the gold query does not express the question, 15 with more than one plausible property for the asked relation, and one string-matching failure. Five cases need an entity absent from the given topic entities. Of 55 labelled method errors, 20 follow a wrong relation or miss a hop, 15 lack a stated constraint, 10 select a wrong program despite a faithful one in W_1, 5 lose an answer in repair and 5 start from a misread sketch. Of 65 sampled cases whose W_1 contains a Hits@1-correct candidate, 26 have a program judged to express the question. These labels are diagnostic for the sample; they are not a label set for all errors.

	Category	CWQ	WebQSP	GrailQA	All
Benchmark	Gold query does not express the question	15	2	7	24
	Another valid property than the gold query	12	3	0	15
	Alias or format mismatch	0	0	1	1
Input	Needed entity not among the topic entities	3	2	0	5
System	Wrong relation or missing hop	3	5	12	20
	Constraint of the question missing	11	4	0	15
	Faithful program in W_1 not selected	3	6	1	10
	Correct answer lost in repair	3	0	2	5
	Sketch misreads the question	3	1	1	5
	Total	53	23	24	100

Table 12: Audit of 100 wrong answers of Qwen3.5-27B, sampled in proportion to the categories of Table 11. Each answer is assigned to the first applicable category from top to bottom.