

Hermes: Verified Query Planning over the SPARQL Algebra with Language Agents

Lam Nguyen

Case Western Reserve University
Cleveland, Ohio, USA
ltn18@case.edu

ABSTRACT

Translating natural-language questions into SPARQL lets users query knowledge graphs without learning their schema, and is a basic step in question answering over knowledge graphs. Large language models (LLMs) now write plausible SPARQL, but they write the whole query at once. A query that fails to parse, joins values of incompatible types, or returns nothing is detected only after it is complete, and the repair is another guess. This paper introduces HERMES, a query-planning agent that builds a SPARQL query one operator of the SPARQL algebra at a time, over a typed formalization of the language. (1) We formalize SPARQL query plans as terms of a typed algebra: an RDF schema induces a type for every query variable, and well-formedness and typing rules decide, without evaluating the query, whether a plan is valid. (2) We formulate natural language to SPARQL as verified query planning. Actions are algebra operators; after each one, the agent observes its effect on the graph (the number of solutions, how it changed, a sample of rows, and any error); and a plan is accepted only when it is well typed and satisfies a goal derived from the question. (3) We implement a verifier that rejects ill-typed steps before they reach the graph store, and an LLM planner that drives it. (4) We normalize six benchmarks into one format and implement a scorer that reproduces every published Text2SPARQL 2026 score. We evaluate with Qwen3 models of 4B, 8B, and 14B parameters. On the Text2SPARQL 2026 corporate benchmark with the 14B model, HERMES reaches a set F_1 of 0.282, against 0.231 without the verifier, 0.197 for constraint-guided search, and 0.089 for one-shot generation, at a higher model-call budget; the verifier cuts the items without a runnable query from 26 to 9.

KEYWORDS

SPARQL, knowledge graphs, natural language interfaces, query algebra, query planning, language agents

PVLDB Reference Format:

Lam Nguyen. Hermes: Verified Query Planning over the SPARQL Algebra with Language Agents. PVLDB, 20(1): XXX-XXX, 2027.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 20, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

The source code, data, and/or other artifacts have been made available at <https://github.com/lamng3/hermes>.

1 INTRODUCTION

Knowledge graphs such as DBpedia [2] and enterprise graphs hold facts that users want to query in their own words. Natural language interfaces translate a question into a structured query that the graph can execute. For relational data this has a long history, from interactive interfaces that map parse trees onto SQL [12] to recent LLM-based systems [6, 11, 13, 19]. For RDF, question answering has been cast as subgraph matching over the graph [29] and as template decomposition of the question [28], and LLMs now generate SPARQL directly [3, 21, 26].

Existing approaches leave three problems open. **C1. Whole-query generation.** Most LLM-based generators emit a complete query string. A query that fails to parse, joins values of incompatible types, or returns nothing is detected only after it is complete, and the repair is another sample from the same model. **C2. No static guarantees.** A query can parse and still be wrong in ways that the schema already rules out: a literal where the property expects an entity, a filter on a variable that nothing binds, or a projected variable that is neither grouped nor aggregated. Such queries run and quietly return nothing, or an error, and the generator learns little from either. **C3. Search needs a reference.** Constraint-guided search [26] refines a query by adding and removing constraints with Chase and Backchase [4], which gives soundness and completeness guarantees, but relative to a reference answer set that must be supplied or guessed, and over monotone conjunctive constraints, with aggregation and ordering deferred.

Example 1.1. Consider the question “Which department does Ms. Brant belong to?” over the Text2SPARQL corporate graph [1], asked in German. With Qwen3-14B [27], a one-shot generator wrote a query over `ecc:Employee` and `foaf:name` “Frau Brant”, terms the graph does not use, so the query ran and returned nothing. Constraint-guided search kept only the constraint `?e pv:memberOf ?d` and returned all six departments that have members. Planning one operator at a time, the agent of Section 4 first stated the goal: one answer of type `pv:Department`. Looking up “Frau Brant” returned two employees, Karen and Sylvester Brant; the agent bound Karen Brant, joined `?e pv:memberOf ?d`, and saw one solution of the stated type: the correct department. Each step was a single algebra operator, and each observation told the agent what that operator did to the answer.

Database systems have handled a similar problem for decades. A query optimizer does not guess a whole execution plan; it applies one algebraic rewrite at a time and estimates its effect before keeping it [20], and the quality of those estimates largely decides the

Table 1: Key notation.

Symbol	Meaning
$\mathcal{G}; \llbracket P \rrbracket_{\mathcal{G}}$	RDF graph; answers of pattern P on $\mathcal{G}$
I, L, V	IRIs, literals, variables ($?x$)
$O = (C, E)$	ontology: classes C , properties E
$dom, rng; \sqsubseteq$	domain and range of a property; subclass order
$D; \text{Lit}$	datatypes; literal types (D and rdfs:Literal)
Q	plan $(T, F, J, X, K, A, H, R, \Pi, \delta, \ell, f)$ (Definition 3.1)
$\bowtie, \sigma, \pi, \gamma, \tau, \lambda$	join, filter, project, group, order, slice (Table 2)
$\beta(Q), \beta^+(Q)$	variables bound by Q ; with aggregate aliases
Γ_Q	typing context of Q
$O \models Q$	Q is valid against O
$g = (\vec{a}, \kappa, c, m); Q \models g$	goal; Q satisfies g
$q; a, o, n$	question; action, observation, solution count
M, ρ, B	language model, planning policy, step budget

quality of the plan [10]. The SPARQL algebra has a compositional semantics [18, 25], so a query can be built and checked the same way. This paper takes that view of natural language to SPARQL.

Contributions. This paper introduces HERMES, a query-planning agent over a typed formalization of the SPARQL algebra.

- (1) **A typed SPARQL algebra** (Section 3). We treat query plans as terms of the SPARQL algebra and derive a typing context for their variables from the RDF schema [24]. Well-formedness and typing rules decide, without evaluating a plan, whether it is valid.
- (2) **Verified query planning** (Section 3). We formulate natural language to SPARQL as planning: actions are algebra operators, observations report each operator’s effect on the graph, and a plan is accepted only when it is well typed and satisfies a goal derived from the question.
- (3) **The HERMES system** (Section 4). A verifier rejects ill-typed steps before they reach the graph store, and an LLM planner drives it. Existing generators are policies over the same action space.
- (4) **A reproducible evaluation** (Section 5). We normalize six benchmarks and implement a scorer that reproduces all 5,024 published Text2SPARQL 2026 scores. With Qwen3 models of 4B to 14B parameters on the corporate benchmark, HERMES is the most accurate method at 14B, its accuracy grows fastest with model size, and the verifier cuts the items without a runnable query at every size.

The code, data, configurations, and run outputs are available at the artifact URL above.

2 PRELIMINARIES

Table 1 summarizes the notation used throughout the paper.

RDF graphs and ontologies. Let I and L be disjoint sets of IRIs and literals; blank nodes are treated as IRIs. An RDF graph $\mathcal{G}$ is a finite set of triples $(s, p, o) \in I \times I \times (I \cup L)$. An ontology $O = (C, E)$ over $\mathcal{G}$, its schema, has classes $C \subseteq I$ and properties $E \subseteq I$, a domain $dom(p) \subseteq C$ and a range $rng(p) \subseteq C \cup D$ for each property $p \in E$, where D is a set of datatypes, and the reflexive–transitive closure $\sqsubseteq$ of rdfs:subClassOf [24]. A *type* is a class or a datatype; Lit denotes the datatypes and rdfs:Literal .

Table 2: SPARQL algebra operators, the relational-algebra symbols we write them with, and the HERMES actions that apply them.

Operator	Symbol	Meaning	Action
BGP, Join	$\bowtie$	join triple patterns	triple
Filter	σ_e	keep mappings satisfying e	filter
LeftJoin	$\Join$	optional extension	optional
Union	$\cup$	either branch	union
Minus	$-$	drop compatible mappings	minus
Extend	$\varepsilon_{?v \leftarrow e}$	bind $?v$ to e	bind
Group, Aggregate	$\gamma_{K;A}$	group and aggregate	group, aggregate
Filter on groups	σ_H	HAVING	having
OrderBy	τ_R	sort	order
Project	π_{Π}	keep variables	project
Distinct	δ	deduplicate	distinct
Slice	λ_{ℓ}	LIMIT, OFFSET	limit, offset
ASK form	$\exists$	existence	ask

The SPARQL algebra. Let V be a set of variables, disjoint from $I \cup L$. A triple pattern is a triple over $I \cup L \cup V$. Following Pérez et al. [18] and the SPARQL 1.1 specification [25], a solution mapping μ is a partial function from V to $I \cup L$, two mappings are compatible when they agree on their shared variables, and a graph pattern P evaluates on $\mathcal{G}$ to a multiset of mappings $\llbracket P \rrbracket_{\mathcal{G}}$. Patterns are built from basic graph patterns (BGPs) by the operators in Table 2. Evaluating SPARQL graph patterns is PSPACE-complete in general [18], so running a plan to inspect it is not free, which motivates checking what can be checked without running it.

Natural language to SPARQL. Given a question q , a graph $\mathcal{G}$, and its ontology O , the task is to produce a query whose answers on $\mathcal{G}$ match the intended answers of q . Following the Text2SPARQL evaluation [1], answers are compared as sets of values: every value bound to every projected variable counts once, and quality is measured by set precision, recall, and F_1 against the answers of a gold query.

3 TYPED PLANS AND VERIFIED QUERY PLANNING

This section formalizes query plans as typed terms of the SPARQL algebra and states the planning problem that HERMES solves.

Plans. A plan is the logical form of a query, built one operator at a time.

Definition 3.1 (Plan). A plan is a tuple $Q = (T, F, J, X, K, A, H, R, \Pi, \delta, \ell, f)$: a set T of triple patterns; a set F of filter expressions; a set J of groups (ω, T') that combine a set T' of triple patterns with the plan by $\omega \in \{\text{optional, union, minus}\}$; extensions X , pairs $(e, ?v)$ of an expression and a variable; group keys $K \subseteq V$; aggregates A , pairs $(\phi, ?v)$ of an aggregate expression and a variable; group filters H ; order keys R ; projected variables $\Pi \subseteq V$; a distinct flag δ ; a slice ℓ ; and a form $f \in \{\text{SELECT, ASK}\}$.

A plan denotes an algebra term with a fixed operator order, written as in relational algebra and applied right to left:

$$Q = \lambda_{\ell} \delta \pi_{\Pi} \tau_R \sigma_H \gamma_{K;A} \varepsilon_X (\bowtie T \odot J), \quad (1)$$

where $\bowtie T$ joins the triple patterns of T , $\odot J$ applies each group of J with its operator ($\bowtie$ for optional, $\cup$ for union, $-$ for minus), an operator with an empty component is the identity, and δ applies only when the distinct flag is set; an ASK plan replaces the top by $\exists$. This is the order of the SPARQL 1.1 algebra [25]. Rendering is deterministic, so a plan and its query are interchangeable, and we write $\llbracket Q \rrbracket_{\mathcal{G}}$ for its answers. An *action* a is a total function from plans to plans that applies one operator of Table 2; the action *remove* retracts a triple pattern or a filter. Plans are immutable, so a sequence of actions is a path $Q_0 \rightarrow Q_1 \rightarrow \dots \rightarrow Q_k$ from the empty plan Q_0 .

Typing. Let $\text{vars}(\cdot)$ denote the variables of a pattern or expression. A *typing context* Γ maps each variable to a set of types. The typing context Γ_Q of a plan Q over an ontology O is the least context closed under the following rules, applied to every triple pattern of T and J :

$$\frac{(\text{?}x, \text{rdf:type}, c) \quad c \in C}{c \in \Gamma(\text{?}x)} \text{ (T-TYPE)}$$

$$\frac{(\text{?}x, p, o) \quad d \in \text{dom}(p)}{d \in \Gamma(\text{?}x)} \text{ (T-DOM)}$$

$$\frac{(s, p, \text{?}y) \quad r \in \text{rng}(p)}{r \in \Gamma(\text{?}y)} \text{ (T-RNG)}$$

Types are compared up to $\sqsubseteq$.

Definition 3.2 (Well-typed plan). A plan Q is *ill typed* over O if (1) a literal occurs in subject position; (2) a literal is the object of a property p whose range $\text{rng}(p)$ contains a class; (3) for some variable, Γ_Q holds both a type in Lit and a class; or (4) a class or property in a namespace that O describes is not in C or E . Otherwise Q is *well typed*.

Two classes in $\Gamma_Q(\text{?}x)$ that are incomparable under $\sqsubseteq$ are reported as a warning, not an error, because RDF makes the open-world assumption and O need not declare disjointness.

Definition 3.3 (Well-formed plan). Let $\beta(Q)$ be the variables bound by Q : the variables of T , of the optional and union groups of J , and of the extensions X ; the variables of minus groups are not bound. Let $\beta^+(Q) = \beta(Q) \cup \{\text{?}v \mid (\varphi, \text{?}v) \in A\}$. A plan is *well formed* if $\text{vars}(e) \subseteq \beta(Q)$ for every filter and extension e ; no extension binds a variable of T or J ; $K \subseteq \beta(Q)$; the variables of H , R , and Π are in $\beta^+(Q)$; and, when A or K is non-empty, $\Pi \cap \beta(Q) \subseteq K$.

A plan that is both well typed and well formed over O is *valid*, written $O \vdash Q$. Validity is decided from Q and O alone, without evaluating Q on $\mathcal{G}$.

Goals. A question constrains the shape of its answer before any query exists: “which department” asks for entities of one class, “how many” for a number, “is there” for a Boolean.

Definition 3.4 (Goal). A goal is $g = (\vec{a}, \kappa, c, m)$: answer variables $\vec{a}$, a kind $\kappa \in \{\text{entity, literal, number, boolean}\}$, an optional type c , and a cardinality $m \in \{\text{one, many}\}$. A valid plan Q *satisfies* g , written $Q \models g$, if: for $\kappa = \text{boolean}$, $f = \text{ASK}$; otherwise $f = \text{SELECT}$, $\vec{a} \subseteq \beta^+(Q)$, the projection is exactly $\vec{a}$, a number is witnessed by an aggregate or a numeric datatype in Γ_Q , an entity has no type in Lit ,

and, if c is given, some type in Γ_Q is subsumed by c . The cardinality $m = \text{one}$ is checked after execution: $|\llbracket Q \rrbracket_{\mathcal{G}}| = 1$.

The goal is derived from the question by the planner, so it can be wrong; satisfaction guarantees that the plan has the shape its goal asks for, not that the goal reflects the question.

The planning problem.

Definition 3.5 (Verified query planning). Given a question q , a graph $\mathcal{G}$, its ontology O , and a step budget B , a policy ρ states a goal g and applies actions $a_1, \dots, a_k$ with $k \leq B$, producing plans $Q_1, \dots, Q_k$. After each action it receives an observation $o_i = (ok_i, n_i, n_{i-1}, \text{rows}_i, \text{err}_i)$: whether Q_i is valid and runs, its number of solutions $n_i = |\llbracket Q_i \rrbracket_{\mathcal{G}}|$, the previous count, a sample of rows, and an error message. An action whose plan is not valid is rejected and not executed. The output Q_k must satisfy $O \vdash Q_k$ and $Q_k \models g$; among such outputs, the objective is the set F_1 of $\llbracket Q_k \rrbracket_{\mathcal{G}}$ against the intended answers of q .

Example 3.6. Take the question “Which US city hosted the most Winter Olympics?” over a small graph of Olympic host cities. The planner first states the goal $g = (\text{?city}, \text{entity}, \text{ex:City}, \text{one})$. The query it builds is

```
SELECT ?city WHERE {
  ?city ex:hosted ?e . ?city a ex:City .
  ?city ex:country ex:USA .
  ?e ex:season "Winter" . }
GROUP BY ?city ORDER BY DESC(COUNT(?e))
LIMIT 1
```

and, as a term of Equation (1),

$$\lambda_1 \pi_{\text{?city}} \tau_{\downarrow n} \gamma_{\text{?city}; n \leftarrow \text{COUNT}(\text{?e})} (t_1 \bowtie t_2 \bowtie t_3 \bowtie t_4),$$

where

$$\begin{aligned} t_1 &= (\text{?city}, \text{ex:hosted}, \text{?e}), \\ t_2 &= (\text{?city}, \text{a}, \text{ex:City}), \\ t_3 &= (\text{?city}, \text{ex:country}, \text{ex:USA}), \\ t_4 &= (\text{?e}, \text{ex:season}, \text{"Winter"}). \end{aligned}$$

Figure 1 shows the plan as an operator tree, built bottom up by ten actions, with the number of solutions the explorer observed after each one. Joining t_3 and t_4 cuts 19 solutions to 8 and then 4; grouping leaves 3 cities; the slice leaves 1, Lake Placid. The verifier keeps two goals open until the end: $\text{?city} : \text{ex:City}$ until t_2 states the type, since this graph declares no domain for ex:hosted , and a single projected answer until the slice and the projection. finish is accepted only after the last action.

The intended answers are unknown during planning, so the objective cannot be optimized directly; the observations are the evidence the policy has. Three properties make this action space suitable for planning.

PROPOSITION 3.7 (MONOTONE REFINEMENT). Let Q be a plan with $J = A = \emptyset$ and no slice, and let Q' apply triple or filter to Q . Then every mapping in $\llbracket Q' \rrbracket_{\mathcal{G}}$, restricted to $\beta(Q)$, is a mapping in $\llbracket Q \rrbracket_{\mathcal{G}}$.

PROOF SKETCH. A mapping of the join of Q with a triple pattern is the union of compatible mappings of Q and of the pattern, so its restriction to $\beta(Q)$ is a mapping of Q . Filtering only removes mappings. $\square$

Table 3: Normalized benchmarks. Items are question-language pairs.

Benchmark	Questions	Items	Graph
CK26 [1]	50	100	corporate, local
DB26 [1]	50	100	DBpedia, endpoint
CK25 [22]	50	50	corporate, local
DB25 [22]	100	200	DBpedia, endpoint
QALD-10 [23]	394	1,564	Wikidata, live
QALD-9-plus [17]	150	450	DBpedia, live

locally; DB26 asks 50 questions in English and Spanish over DBpedia [2], queried at the organizers’ endpoint. Each track has 100 items.

Data caveats. Three properties of the data shape what a number means. (1) Forty-nine of the fifty CK26 gold queries appear verbatim in CK25, on the same graph, so neither is a development set for the other; no method here uses either for examples or tuning. (2) The CK gold answers were computed on the Tentris engine. Rerunning the gold queries on Oxigraph reproduces 43 of the 50; the other seven differ in tie-breaking under LIMIT, decimal formatting, or the non-standard cast `xsd:int()`. We report results on all items and on the 86 items whose gold reproduces. All 50 DB26 gold queries reproduce on the organizers’ endpoint. (3) The official scorer has two quirks for ASK and ordered questions, described in Appendix C; we report the official metric and a strict variant that corrects them.

Metrics. We report set precision (P), recall (R), and F_1 , and the challenge’s ranking metric F_{ndcg} , which uses nDCG on questions whose order matters and set F_1 elsewhere. Our scorer reproduces all 5,024 per-question and average scores published for the 2026 challenge (Appendix C), so our numbers are on the leaderboard’s scale.

Methods. *OneShot* prompts a single query from the schema vocabulary, declares any known prefix the draft omits, and makes one repair call if the draft does not parse. *CGS* is our reimplementaion of constraint-guided search with Chase and Backchase [26]. It follows the published three stages, with two documented deviations: queries are rendered from constraints deterministically rather than by a second model, and candidates are scored by comparing answer sets with the reference answers rather than by a model. It uses two model calls per question, one for the constraint table and one for the reference answers, beam width 5, and depth 5. *HERMES*[−] is the planner of Section 4 with the verifier disabled: execution feedback, no typing, no goal. *HERMES* is the full verified planner. Both planners have a step budget of $B = 15$, and their prompt examples use an unrelated domain (Olympic host cities, Figure 4); no prompt names CK26 content.

Models. We run every method with Qwen3 [27] at three sizes, 4B, 8B, and 14B parameters, with thinking disabled and temperature 0. Each model is served in bf16 by vLLM [8] on one NVIDIA L40S GPU, and each run sends 16 items at a time so that the server batches their requests. Replies are capped at 2,048 tokens. Small open models keep every run reproducible at low cost and keep differences between methods from being masked by a model that

Table 4: Effectiveness on CK26 (100 items). Official metrics; Strict is set F_1 under the corrected scorer, and Repro. is set F_1 on the 86 items whose gold reproduces. Best per model in bold.

Model	Method	P	R	F_1	F_{ndcg}	Strict	Repro.
Qwen3-4B	OneShot	0.001	0.000	0.000	0.000	0.000	0.000
	CGS	0.094	0.169	0.084	0.085	0.084	0.082
	HERMES [−]	0.099	0.167	0.103	0.102	0.103	0.114
	HERMES	0.064	0.207	0.070	0.071	0.070	0.080
Qwen3-8B	OneShot	0.093	0.068	0.067	0.068	0.067	0.078
	CGS	0.155	0.262	0.121	0.121	0.121	0.119
	HERMES [−]	0.168	0.372	0.178	0.178	0.178	0.182
	HERMES	0.148	0.412	0.158	0.159	0.158	0.172
Qwen3-14B	OneShot	0.111	0.091	0.089	0.090	0.089	0.103
	CGS	0.209	0.399	0.197	0.197	0.197	0.210
	HERMES [−]	0.254	0.277	0.231	0.231	0.231	0.251
	HERMES	0.290	0.414	0.282	0.282	0.282	0.297

answers from memory; the published Text2SPARQL entries use larger models, so our numbers are a lower bound for each method. HERMES takes the model as a parameter, so the same runs extend to other open models such as Llama 3.1 [7] and to API models such as GPT-4 [15]. **[Pending: API models and Llama 3.1 8B.]**

Implementation. All methods share the graph store (Oxigraph, in memory), the entity index, and the scorer. Every graph query is limited to 30 s and every item to 300 s; an item that exceeds its limit counts as unanswered. Runs record the source commit, and each table and figure in this section is generated from the committed run outputs.

5.2 RQ1: Effectiveness

Table 4 reports all twelve runs. HERMES is the most accurate method at 14B: F_1 0.282, against 0.231 for HERMES[−], 0.197 for CGS, and 0.089 for OneShot, with the highest precision (0.290), recall (0.414), and F_{ndcg} . It answers 16 items exactly, as many as HERMES[−], against 11 for CGS and 4 for OneShot. The lead holds in both languages (F_1 0.297 in English and 0.268 in German, against 0.274 and 0.187 for HERMES[−]) and on the 86 items whose gold reproduces (0.297). Strict scoring changes no F_1 at any size, because no method answers an ASK question.

Accuracy grows with model size for every method (Figure 2), but not at the same rate. From 4B to 14B, F_1 rises by 0.212 for HERMES, 0.128 for HERMES[−], 0.113 for CGS, and 0.089 for OneShot. At 4B, CGS is the most accurate method (0.084 F_1 against 0.070 for HERMES): the planners need a model that can follow a multi-step protocol, and the 4B model often does not (Section 5.3). OneShot at 4B scores 0: on 79 items its query does not parse, most often because it omits the query form (SELECT).

For scale, Table 5 places the 14B runs among the published 2026 entries on CK26, which range from 0.157 to 0.832 in F_{ndcg} [1, 21]. This is not a paired comparison: those systems were scored by the organizers on Tentris, and their models are not controlled here. With a 14B open model, HERMES places above three of the eight entries.

Table 6 breaks the 14B runs down by the feature tags of the gold queries. HERMES leads on plain selections, filters, and nested

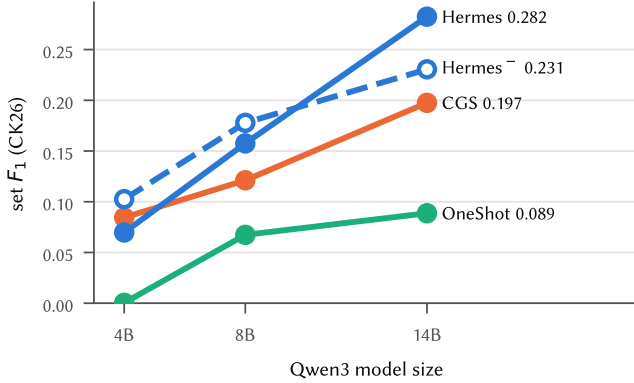


Figure 2: Set F_1 on CK26 against Qwen3 model size. Dashed, hollow: HERMES without the verifier.

Table 5: CK26 F_{ndcg} of the published Text2SPARQL 2026 entries, scored by the organizers, and of our runs with Qwen3-14B (shaded).

Entry	F_{ndcg}	Entry	F_{ndcg}
SPARQL-LLM [21]	0.832	HERMES ⁻	0.231
INFAI-ETI-A	0.633	IRIS	0.217
ADFR	0.513	LIBER-AI-QWEN	0.206
INFAI-ETI-C	0.452	CGS	0.197
LIBER-AI-CLAUDE	0.335	INFAI-ETI-B	0.157
HERMES	0.282	OneShot	0.090

Table 6: CK26 set F_1 by feature tag of the gold query (Qwen3-14B); a question can carry several tags, and n counts items. Best per row in bold.

Tag	n	OneShot	CGS	HERMES ⁻	HERMES
SELECT	94	0.094	0.210	0.245	0.300
FILTER	10	0.096	0.391	0.285	0.395
ORDER	30	0.030	0.092	0.174	0.131
SUBSELECT	8	0.047	0	0.001	0.094
GROUP	18	0	0.081	0.168	0.156
COUNT	16	0	0.089	0.244	0.165
LIMIT	26	0	0.063	0.115	0.065
ASK	6	0	0	0	0

queries. HERMES⁻ leads on ordering, grouping, counting, and slicing. No method answers any of the six ASK items.

5.3 RQ2: Components

Verifier. Table 7 compares HERMES with HERMES⁻, which differ only in the verifier. The verifier sharply reduces the items that end with no runnable query at every size: from 57 to 35 at 4B, from 15 to 2 at 8B, and from 26 to 9 at 14B. It rejects 37, 48, and 29 steps per run at 4B, 8B, and 14B, and each rejection names the violated rule (for example, a projected variable that no pattern binds), so the planner repairs the step instead of running it. Accuracy follows at 14B, where F_1 rises from 0.231 to 0.282. At 4B and 8B it does not: F_1 falls from 0.103 to 0.070 and from 0.178 to 0.158. At 4B, HERMES returns a query on 30 items where HERMES⁻ returned none, but only 10 of them score above zero, and it scores zero on 11 items

Table 7: The verifier: HERMES against HERMES⁻ on CK26. No answer counts items that end without a runnable query; Steps is the mean number of actions per item.

Model	F_1		No answer		Steps	
	HERMES ⁻	HERMES	HERMES ⁻	HERMES	HERMES ⁻	HERMES
Qwen3-4B	0.103	0.070	57	35	5.6	9.3
Qwen3-8B	0.178	0.158	15	2	11.2	13.7
Qwen3-14B	0.231	0.282	26	9	10.3	11.3

that HERMES⁻ answered in part; at 8B the counts are 14, 6, and 16. The verifier turns failures into answers; whether the answers are right depends on the model.

Reference answers. CGS ranks candidates by their agreement with reference answers that the model predicts. [Pending: CGS with gold reference answers on Qwen3.]

Feedback signals. [Pending: ablation of the observation, removing in turn the count, the change in count, the row sample, and the error message.]

5.4 RQ3: Time and Cost

Table 8 reports the cost of every run; Figure 3(a) sets accuracy against model calls, and Figure 3(b) shows the time of every item. All twelve runs take 27.6 minutes of L40S time in total.

Model calls and tokens. The planners spend 12.7 to 14.6 model calls per item, against about 2 for CGS and 1 to 2 for OneShot, and read about 28k input tokens per item, 7 times as many as CGS: every step resends the plan and its observations. Output is small for all methods (11k to 52k tokens per run), so input dominates the cost. The comparison in Table 4 is therefore not at equal model budget. [Pending: budget-matched comparison, e.g., CGS or OneShot with self-consistency at 15 calls per item.]

Time. Because the server batches 16 items at a time, every run of 100 items finishes in 1.0 to 3.6 minutes. Per item, the planners are slower and more predictable: at 14B the median item takes 25.5 s for HERMES and 21.7 s for HERMES⁻, against 9.9 s for CGS and 3.1 s for OneShot, and the 95th percentile is 32.7 s for HERMES against 110.5 s for CGS. The step budget bounds each planner item, while the time of CGS depends on the size of its search: at 14B it runs 2,132 graph executions, more than twice as many as HERMES (902).

5.5 Case Study

Table 9 is the trace of HERMES with Qwen3-14B on the German version of Example 1.1. The first action states the goal, one answer of type pv: Department, before any query exists. The lookup returns two employees named Brant; the planner binds Karen Brant, and the join with pv: memberOf yields one solution of the goal’s type, the gold answer. The verifier matters on other items. On “What’s Baldwin Dirksen’s phone number?”, HERMES⁻ found the employee but then tried to project the property pv: phone as if it were a variable, failed, and repeated the step until its budget ran out; HERMES stated a literal goal, joined pv: phone ?p, and answered in four steps. On a German request for every supplier’s name and address, the verifier rejected a projection of ?addressText, a variable whose pattern the planner had removed because it matched nothing; the

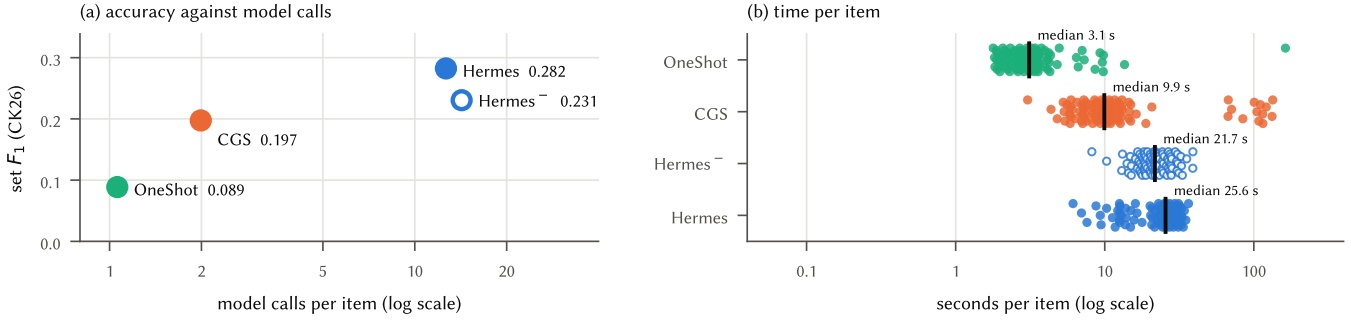


Figure 3: Cost on CK26 with Qwen3-14B. (a) Set F_1 against model calls per item. (b) Seconds per item, one dot per item, with the median as a bar. Hollow markers: HERMES without the verifier.

Table 8: Cost per run on CK26 (100 items). Calls are model calls; In and Out are input and output tokens in thousands; Exec. counts graph executions; Wall is the whole run in minutes; p50 and p95 are seconds per item.

Model	Method	Calls	In (k)	Out (k)	Exec.	Wall	p50	p95
4B	OneShot	198	91	16.8	21	1.1	2.7	3.6
	CGS	180	419	39.9	995	1.6	3.2	54.0
	HERMES ⁻	1,417	2,569	23.4	452	3.6	7.3	177.3
	HERMES	1,455	2,841	34.5	482	1.0	9.5	12.5
8B	OneShot	111	76	12.8	87	1.8	2.3	4.8
	CGS	196	430	51.3	635	2.8	7.0	80.9
	HERMES ⁻	1,342	2,806	29.0	830	2.1	18.3	24.9
	HERMES	1,395	3,196	30.5	873	2.5	22.4	27.5
14B	OneShot	106	73	11.3	93	3.0	3.1	9.3
	CGS	199	392	27.3	2,132	2.7	9.9	110.5
	HERMES ⁻	1,424	2,945	28.0	833	2.5	21.7	32.4
	HERMES	1,265	2,856	24.5	902	2.6	25.5	32.7

Table 9: Trace of HERMES (Qwen3-14B) on “Zu welcher Abteilung gehört Frau Brant?” (CK26, item 1, German).

Action	Observation
goal ?d : pv : Department	one answer
lookup “Frau Brant”	2 entities: Karen, Sylvester Brant
lookup “Karen Brant”	1 entity
triple ?e pv : name “Karen Brant”	1 solution
triple ?e pv : memberOf ?d	1 solution
project ?d	1 solution
finish	accepted

planner projected the bound address fields instead and reached F_1 0.84, where HERMES⁻ scored 0.

6 RELATED WORK

Natural language interfaces to data. Natural language interfaces to relational databases map a parsed question onto SQL, and NaLIR made the mapping interactive by asking the user to resolve ambiguities [12]. For RDF, gAnswer casts question answering as subgraph matching between a semantic query graph and the knowledge graph [29], and template decomposition answers complex questions by composing binary templates mined from the graph [28]. GraphLingo explores domain knowledge graphs in natural language by synchronizing them with an LLM [9]. Benchmarks for question answering over RDF include LC-QuAD

2.0 [5], QALD-9-plus [17], QALD-10 [23], and the Text2SPARQL challenges [1, 22].

LLM-based text-to-SQL. With LLMs, text-to-SQL has moved to prompting and fine-tuning. DAIL-SQL studies how question representation and example selection shape LLM prompts [6], PURPLE retrieves demonstrations that match the logical operators a question needs [19], CodeS pre-trains open models for SQL generation [13], and NL2SQL360 evaluates these systems along many dimensions [11]. These methods emit a whole query and improve it by better context or better models; none types the query against the schema before it runs.

LLM-based text-to-SPARQL. SPARQL-LLM, the strongest entry of the 2026 Text2SPARQL challenge, generates SPARQL from natural-language questions in real time [21]. ARUQUA explores the graph with tools in a ReAct loop before writing a query [3]. UniQGen extracts a table of constraints from the question and the graph and refines a query with Chase and Backchase against reference answers from an LLM, rendering it to SPARQL or Cypher [26]. HERMES differs from these in its unit of generation: an operator of the SPARQL algebra rather than a query string or a constraint table, checked against the schema before execution and observed after it.

Query optimization and SPARQL semantics. Cost-based optimizers choose among algebraically equivalent plans by estimating the effect of each operator [20], and cardinality estimates dominate the quality of the chosen plan [10]. Chase and Backchase reformulates queries under constraints and minimizes them [4]. Pérez et al. gave SPARQL a compositional semantics and characterized the complexity of its evaluation [18]. HERMES borrows the optimizer’s step, one operator with an observed effect, and the algebra’s semantics, and uses them to generate a query rather than to rewrite one.

7 CONCLUSION

We have presented HERMES, which treats natural language to SPARQL as verified query planning over a typed SPARQL algebra. Plans are typed against the RDF schema and checked without execution; actions are algebra operators whose effect on the graph is observed; and a plan is accepted only when it is valid and satisfies a goal derived from the question. On the Text2SPARQL 2026 corporate benchmark with Qwen3 models from 4B to 14B

parameters, HERMES is the most accurate method at 14B (F_1 0.282), and its accuracy grows fastest with model size. The verifier cuts the items without a runnable query at every size, but raises accuracy only at 14B: most of the queries it recovers from the smaller models score zero. No method answers ASK questions, whose answer shape a goal can state in advance. Next steps are API and larger open models, a budget-matched comparison, ablations of the observation, the DBpedia and Wikidata benchmarks, and the proofs in Appendix A.

REFERENCES

- [1] AKSW. 2026. The Text2SPARQL Challenge 2026: Datasets, Results, and Evaluation Client. Challenge site and result repository. <https://github.com/AKSW/text2sparqlaksw.org>
- [2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference (ISWC) (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 722–735. https://doi.org/10.1007/978-3-540-76298-0_52
- [3] Felix Brei, Lorenz Bühlmann, Johannes Frey, Daniel Gerber, Lars-Peter Meyer, Claus Stadler, and Kirill Bulert. 2025. ARUQULA – An LLM based Text2SPARQL Approach using ReAct and Knowledge Graph Exploration Utilities. In *Proceedings of the First International TEXT2SPARQL Challenge 2025, co-located with Text2KG at ESWC 2025 (CEUR Workshop Proceedings)*, Vol. 4094. CEUR-WS.org, Aachen, Germany, 49–63. <https://ceur-ws.org/Vol-4094/>
- [4] Alin Deutsch, Lucian Popa, and Val Tannen. 1999. Physical Data Independence, Constraints, and Optimization with Universal Plans. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, San Francisco, CA, USA, 459–470.
- [5] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In *The Semantic Web – ISWC 2019 (Lecture Notes in Computer Science)*. Springer, Cham, 69–78. https://doi.org/10.1007/978-3-030-30796-7_5
- [6] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1132–1145. <https://doi.org/10.14778/3641204.3641221>
- [7] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783. <https://arxiv.org/abs/2407.21783>
- [8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 611–626. <https://doi.org/10.1145/3600006.3613165>
- [9] Duy Le, Kris Zhao, Mengying Wang, and Yinghui Wu. 2024. GraphLingo: Domain Knowledge Exploration by Synchronizing Knowledge Graphs and Large Language Models. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, Piscataway, NJ, USA, 5477–5480. <https://doi.org/10.1109/ICDE60146.2024.00432>
- [10] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proceedings of the VLDB Endowment* 9, 3 (2015), 204–215. <https://doi.org/10.14778/2850583.2850594>
- [11] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proceedings of the VLDB Endowment* 17, 11 (2024), 3318–3331. <https://doi.org/10.14778/3681954.3682003>
- [12] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84. <https://doi.org/10.14778/2735461.2735468>
- [13] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proceedings of the ACM on Management of Data (SIGMOD)* 2, 3 (2024), 1–28. <https://doi.org/10.1145/3654930>
- [14] National Institute of Standards and Technology. 2024. trec_eval. Information retrieval evaluation software. https://github.com/usnistgov/trec_eval
- [15] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774. <https://arxiv.org/abs/2303.08774>
- [16] Oxigraph contributors. 2026. Oxigraph. RDF graph database. <https://github.com/oxigraph/oxigraph>
- [17] Aleksandr Perevalov, Dennis Diefenbach, Ricardo Usbeck, and Andreas Both. 2022. QALD-9-plus: A Multilingual Dataset for Question Answering over DBpedia and Wikidata Translated by Native Speakers. In *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*. IEEE, Piscataway, NJ, USA, 229–234. <https://doi.org/10.1109/ICSC52841.2022.00045>
- [18] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems* 34, 3 (2009), 16:1–16:45. <https://doi.org/10.1145/1567274.1567278>
- [19] Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: Making a Large Language Model a Better SQL Writer. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, Piscataway, NJ, USA, 15–28. <https://doi.org/10.1109/ICDE60146.2024.00009>
- [20] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 23–34. <https://doi.org/10.1145/582095.582099>
- [21] Panayiotis Smeros, Vincent Emonet, Ruijie Wang, Ana-Claudia Sima, and Tarcisio Mendes de Farias. 2026. SPARQL-LLM: Real-Time SPARQL Query Generation from Natural Language Questions. *ACM Transactions on the Web* (2026). <https://doi.org/10.1145/3847195>
- [22] Sebastian Tramp, Marcos Gólo, Edgard Marx, and Paulo Vivivurka do Carmo (Eds.). 2025. *Proceedings of the First International TEXT2SPARQL Challenge 2025, co-located with Text2KG at ESWC 2025*. CEUR Workshop Proceedings, Vol. 4094. CEUR-WS.org, Aachen, Germany. <https://ceur-ws.org/Vol-4094/>
- [23] Ricardo Usbeck, Xi Yan, Aleksandr Perevalov, Longquan Jiang, Julius Schulz, Angelie Kraft, Cedric Moeller, Junbo Huang, Jan Reineke, Axel-Cyrille Ngonga Ngomo, Muhammad Saleem, and Andreas Both. 2023. QALD-10 – The 10th Challenge on Question Answering over Linked Data. *Semantic Web journal*, submission 3357; dataset at <https://github.com/KGQA/QALD-10>. <https://www.semantic-web-journal.net/node/3357>
- [24] W3C RDF Working Group. 2014. RDF Schema 1.1. W3C Recommendation, 25 February 2014. <https://www.w3.org/TR/rdf-schema/>
- [25] W3C SPARQL Working Group. 2013. SPARQL 1.1 Query Language. W3C Recommendation, 21 March 2013. <https://www.w3.org/TR/sparql11-query/>
- [26] Mengying Wang, Nicolaas Jedema, Rahul Pandey, RaviKiran Krishnan, Jens Lehmann, and Yinghui Wu. 2026. Graph Query Generation with Constraint-guided Large Language Agents. arXiv:2605.00845. <https://arxiv.org/abs/2605.00845>
- [27] An Yang, Anfeng Li, Baosong Yang, et al. 2025. Qwen3 Technical Report. arXiv:2505.09388. <https://arxiv.org/abs/2505.09388>
- [28] Weiguo Zheng, Jeffrey Xu Yu, Lei Zou, and Hong Cheng. 2018. Question Answering over Knowledge Graphs: Question Understanding via Template Decomposition. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1373–1386. <https://doi.org/10.14778/3236187.3236192>
- [29] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural Language Question Answering over RDF: A Graph Data Driven Approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 313–324. <https://doi.org/10.1145/2588555.2610525>

Algorithm 1: Verified query planning

Input: question q , graph $\mathcal{G}$, ontology O , policy ρ , step budget B
Output: a plan Q

```

1  $Q \leftarrow Q_0$ ;  $g \leftarrow \perp$ ;  $h \leftarrow []$ ;
2 for  $i \leftarrow 1$  to  $B$  do
3    $a \leftarrow \rho(q, O, Q, g, h)$ ;
4   if  $a$  is goal then
5      $g \leftarrow a.g$ ;  $o \leftarrow \text{VERIFY}(Q, g)$ ;
6   else if  $a$  is lookup or inspect then
7      $o \leftarrow \text{read } \mathcal{G}$ ;
8   else if  $a$  is finish then
9     if  $\text{VERIFY}(Q, g)$  has no errors and no open goals then
10      return  $Q$ ;
10     $o \leftarrow$  refusal with the open goals;
11  else
12     $Q' \leftarrow a(Q)$ ;  $o \leftarrow \text{VERIFY}(Q', g)$ ;
13    if  $o$  has no errors then
14       $o \leftarrow o \cup \{\llbracket Q' \rrbracket_{\mathcal{G}}, \text{sample}\}$ ;
15      if  $Q'$  runs then  $Q \leftarrow Q'$ ;
16  append  $(a, o)$  to  $h$ ;
17 return the last plan in  $h$  that runs;
```

APPENDIX

A. Theorems and Proofs

Proposition 3.7 (Monotone refinement).

PROOF. Let Q be a plan with $J = A = \emptyset$ and no slice, and let $\mu' \in \llbracket Q' \rrbracket_{\mathcal{G}}$. If Q' adds a triple pattern t , then by the semantics of join $\mu' = \mu \cup \nu$ for some compatible $\mu \in \llbracket Q \rrbracket_{\mathcal{G}}$ and $\nu \in \llbracket t \rrbracket_{\mathcal{G}}$; μ is defined exactly on $\beta(Q)$, so the restriction of μ' to $\beta(Q)$ is μ . If Q' adds a filter e , then $\llbracket Q' \rrbracket_{\mathcal{G}}$ is the set of mappings of $\llbracket Q \rrbracket_{\mathcal{G}}$ that satisfy e . In both cases the restriction of μ' to $\beta(Q)$ is in $\llbracket Q \rrbracket_{\mathcal{G}}$. $\square$

Proposition 3.8 (Static soundness). [Proof to be written: by induction on the structure of the plan, following Algorithm 2, showing that each condition of the definitions of well-typed and well-formed plans is checked.]

Proposition 3.9 (Checking cost). [Proof to be written: one pass over the triple patterns with constant-time schema lookups, one pass over expressions, and a precomputed subclass closure.]

B. Algorithms and Analysis

Algorithm 1 is the planning loop of Section 4; Algorithm 2 is the verifier.

[Complexity analysis of Algorithms 1 and 2: to be written with Proposition 3.9.]

C. Evaluation Details

Scorer parity. The official Text2SPARQL scorer flattens every value of every variable into one set and scores it with `trec_eval` [14]. We rescored the predicted result sets of all eight published 2026 entries on CK26 and DB26 and compared every per-question score

Algorithm 2: $\text{VERIFY}(Q, g)$

Input: plan Q , goal g (possibly $\perp$), ontology O
Output: errors Err , typing context Γ_Q , open goals $Open$

```

1  $Err \leftarrow \emptyset$ ;  $\Gamma_Q \leftarrow \emptyset$ ;
2 foreach triple pattern  $(s, p, o)$  of  $T$  and  $J$  do
3   if  $p \notin E$ , or a class not in  $C$ , lies in a namespace  $O$ 
4     describes then add to  $Err$ ;
5   if  $s$  is a literal, or  $o$  is a literal and  $\text{rng}(p)$  has a class
6     then add to  $Err$ ;
7   apply T-TYPE, T-DOM, and T-RNG to extend  $\Gamma_Q$ ;
8 foreach variable with a type in  $Lit$  and a class in  $\Gamma_Q$  do add
9   to  $Err$ ;
10 check the conditions of Definition 3.3 on  $\beta(Q)$  and  $\beta^+(Q)$ ,
11   adding violations to  $Err$ ;
12  $Open \leftarrow$  the conditions of  $Q \models g$  (Definition 3.4) that fail;
13 return  $(Err, \Gamma_Q, Open)$ ;
```

and every average with the official files: all 5,024 numbers agree exactly. The check is a script in the artifact.

Quirks of the official metric. (1) An ASK answer is scored as one document named true with relevance 1 or 0. A gold true therefore scores 1.0 whatever the system answers, and a gold false scores 0.0 for every system. (2) Every returned value has the same score, so `trec_eval` orders values by their text, and the official $nDCG$ ignores the order the system returned. (3) The official F_{ndcg} average counts the mean $nDCG$ of the scope once more as an extra item. Our official column reproduces all three; the strict column scores Booleans by exact match, uses the returned order for $nDCG$, and takes a plain mean.

Engine dependence of the CK gold answers. Of the seven CK26 gold queries that do not reproduce on Oxigraph, two break ties under `LIMIT` differently, two use `xsd:int()`, which is not a SPARQL 1.1 function, two format computed numbers differently (e.g., 3 against 3.0), and one computes a different value for an arithmetic expression. On `rdflib`, 42 of 50 reproduce.

D. Prompts

Figure 4 is the action catalogue and the rules shown to the planner at every turn, followed by the vocabulary, the linked entities, the history of actions and observations, the plan state, and the current query. HERMES⁺ uses the same catalogue without the goal action, no plan state, and shorter rules: use only the terms shown or found, look entities up before using them, project only the answer variables, and finish when the result looks right.

State the goal first:

```
{
  "op": "goal",
  "answer": ["?c"],
  "kind": "entity",
  "type": "ex:City",
  "many": true
}
```

kind is entity, literal, number or boolean; type is optional; many is false for one answer.

Actions. Reply with exactly one JSON object per turn.

Read the graph (the plan does not change):

```
{
  "op": "lookup",
  "text": "Lake Placid"
}
```

entities whose label contains the text

```
{
  "op": "inspect",
  "iri": "ex:City"
}
```

predicates around an entity or a class

Change the plan (SPARQL algebra):

```
{
  "op": "triple",
  "pattern": "?c ex:hosted ?g"
}
```

add a triple pattern (join)

```
{
  "op": "filter",
  "expr": "?year > 1900"
}
```

```
{
  "op": "optional",
  "patterns": ["?c ex:country ?k"]
}
```

```
{
  "op": "union",
  "branches": [{"?x a ex:A"}, [{"?x a ex:B"}]]
}
```

```
{
  "op": "minus",
  "patterns": ["?g ex:season \"Summer\""]
}
```

```
{
  "op": "bind",
  "expr": "YEAR(?date)",
  "var": "?y"
}
```

```
{
  "op": "group",
  "vars": ["?c"]
}
```

```
{
  "op": "aggregate",
  "expr": "COUNT(DISTINCT ?g)",
  "alias": "?n"
}
```

```
{
  "op": "having",
  "expr": "?n > 2"
}
```

```
{
  "op": "order",
  "expr": "DESC(?n)"
}
```

```
{
  "op": "limit",
  "n": 1
}
```

```
{
  "op": "offset",
  "n": 1
}
```

```
{
  "op": "project",
  "vars": ["?c"]
}
```

```
{
  "op": "distinct"
}
```

```
{
  "op": "ask"
}
```

yes/no questions

```
{
  "op": "remove",
  "text": "?c ex:country ?k"
}
```

drop a triple pattern or filter

```
{
  "op": "undo"
}
```

```
{
  "op": "finish"
}
```

when the query answers the question

Rules: state the goal first. A verifier checks every step before it runs and rejects ill-typed ones with a message; read it and fix the step. Use only the prefixes and terms shown or found with lookup and inspect. Look entities up before using them. Finish only when the plan state says goals accomplished.

Figure 4: Planner prompt: action catalogue and rules.