Evaluation Event II - *RedFlickeringCandle*

Haizhou Zheng (*haizhou.zheng@uzh.ch*)
Zhenmei Hao (*zhenmei.hao@uzh.ch*)

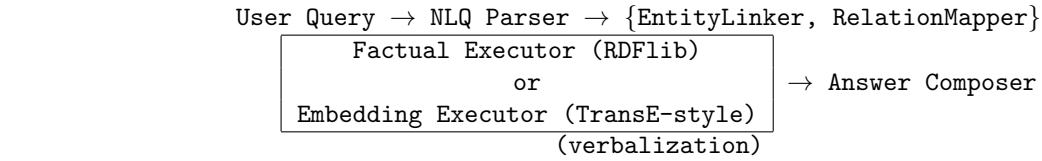
October 29, 2025

1 Introduction

We build a lightweight but reliable QA agent over a given knowledge graph (KG) that supports two answering routes: a **factual** route that compiles a SPARQL-style lookup, and an **embedding** route that retrieves answers in the vector space. If a question explicitly requires a factual (or embedding) approach, the agent only executes that route; otherwise it returns both. The pipeline consists of five modules: NLQ parsing, entity linking, relation mapping, factual/embedding executors, and answer composition. Key features include (i) quote- and case-robust NLQ parsing, (ii) fast and cached entity linking, (iii) **multi-property** label and type resolution (RDFS/SKOS/Schema.org/Wikidata) from the provided KG file, (iv) a subject-first candidate set for embeddings, and (v) concise natural-language verbalization with de-duplication.

2 Capabilities

Overview.



Factual questions. (1) The NLQ parser normalizes quotes and extracts entity mentions and relation triggers. (2) The EntityLinker builds a label index from the KG and uses RapidFuzz to return top-k entity candidates. (3) The RelationMapper maps trigger tokens to a KG predicate IRI. (4) The GraphExecutor then reads $(s, p, ?o)$ triples via RDFlib and resolves objects to readable labels from `rdfs:label`, `skos:prefLabel/altLabel`, `schema:name`, and `wdt:P1476`, followed by de-duplication and verbalization.

Embedding (tail prediction). This path answers queries of the form $(subject, predicate, ?object)$. We pick the first subject candidate that has an embedding vector. Candidate

tails are *first* collected from the 1-hop neighborhood $(subject, predicate, ?o)$; if empty, we *fallback* to all tails of that predicate across the graph with downsampling (`MAX_TAILS`). We compute a TransE-style target $t = \text{norm}(s + r)$ and score candidates by cosine similarity. For the top hit we attach a short type from `rdf:type`/`wdt:P31`; we also estimate the predicate's majority object type as an *expected type* for display consistency.

Embedding (head prediction). We additionally implement the reverse direction, answering $(?subject, predicate, object)$. Given a tail entity that has an embedding vector, we first collect head candidates *specific to that object*, i.e., $(?s, predicate, object)$, and fallback to all heads of the predicate if needed. We use a TransE target $t = \text{norm}(o - r)$ and cosine scoring to retrieve the top-k heads. As of this submission, we attach per-entity types like in tail prediction; *we do not yet compute an "expected subject type" for this direction.*

3 Adopted Methods

- **RDFlib**[1]: used for parsing the KG and querying triples (s, p, o) in Python with stable APIs and serializers.
- **RapidFuzz**[2]: fast fuzzy string matching for entity labels to construct top-k candidates robust to typos and variants.
- **FastAPI**[3]: lightweight service wrapper for a clean `/ask` endpoint and health checks.
- **NumPy**[4]: efficient vector math and L2 normalization for embedding scoring.
- **Translation-based scoring** [1]: we adopt $s + r \approx o$ and $o - r \approx s$ in a TransE-style setup, using cosine similarity.

4 Examples

- **Factual.** Given "*Please answer this question with a factual approach: Who is the director of 'Good Will Hunting'?* " the parser extracts the title and relation trigger. The linker returns the film entity; the mapper maps to the "director" predicate. The factual executor retrieves the objects and resolves labels, yielding *Gus Van Sant*, which matches the KG.
- **Embedding (tail).** Given "*What is the genre of 'Shoplifters'?*" we select the film as subject and collect 1-hop tails for the "genre" predicate as candidates. After TransE-style scoring, the top hit is *drama film* with type *film genre* (e.g., Q201658).

[1] `https://rdflib.readthedocs.io/`
[2] `https://maxbachmann.github.io/RapidFuzz/`
[3] `https://fastapi.tiangolo.com/`
[4] `https://numpy.org/`

5 Additional Features

- **Intent routing.** We strictly obey the question instruction (factual / embedding / both), ensuring only the requested route runs and preventing unintended mixing.
- **Direction-aware candidates for embeddings.** For tail prediction we first gather 1-hop tails $(s, p, ?o)$ via graph lookup and only then fall back to global predicate tails; for head prediction we symmetrically use $(?s, p, o)$. This subject-/object-first policy keeps candidates semantically tight.
- **Efficient retrieval.** We downsample large global pools with `MAX_TAILS` and compute Top-k using vectorized `argpartition`, yielding low latency even on bigger KGs.
- **Readable labels and types.** We resolve labels from multiple RDF properties (RDFS/SKOS/Schema.org/Wikidata title) and extract types from `rdf:type`/`wdt:P31`. For tail prediction, we also estimate a predicate-level *expected object type* to satisfy the task's "return the entity type" requirement.
- **Clean verbalization.** Answers are de-duplicated and rendered as concise sentences ("A and B and C"), avoiding repeated or list-like artifacts.
- **Caching and observability.** We maintain small persistent caches (label index, predicate tails/heads, majority types) and expose a `/health` endpoint with structured logging for monitoring.
- **Fail-safe behavior.** All computations are offline and deterministic (no external I/O). If a relation vector, subject/tail vector, or candidate pool is missing, the executor returns no embedding answer rather than a speculative guess—favoring precision over noise.

6 Conclusions

Our agent reliably answers KG questions via two complementary routes and obeys the task's instruction about which route to use. For embeddings, we support both tail and head prediction. Next steps include multimedia, recommendation question types.

Contributions. H. Zheng implemented the service skeleton, entity linker and NLQ parsing; Z. Hao implemented the intent routing and report. Both authors co-designed the relation mapper, factual executor, embedding executor and debugging.

References

[1] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.