



Enabling Knowledge Graph Understanding at Scale with EXYGEN

The **EX**plore **Y**our **G**raphs **EN**gine

Harshdeep Singh · Yurui Zhu · Giovanni Colavizza · Matteo Romanello

Odoma · RAGE-KG 2026, Bari, Italy · October 25–26, 2026

LLMs meet knowledge graphs

LLMs let natural language act as an interface to KGs without learning SPARQL; KGs in turn ground LLM output and reduce hallucinations. Two limitations hold LLM-based KGQA back:

LIMITATION 1

Dependence on KG-specific supervision

Fine-tuning needs costly graph-specific data and generalize poorly across schemas.

Fine-tuned Llama models [\[Ref 2\]](#):

60.68% Macro F1 on DBpedia, only **13.36%** on Wikidata.

LIMITATION 2

Scalability of metadata generation

Context-based methods rely on endpoint metadata (VoID, ShEx), produced with heavy SPARQL aggregations and Cartesian products.

On large KGs these hit prohibitive runtimes and frequent timeouts.

Research questions and contributions

RQ1

To what extent can existing LLMs perform text-to-SPARQL generation **without task-specific fine-tuning**, given only automatically extracted, KG-derived context?

RQ2

How can the structured metadata (VoID, ShEx) this method relies on be **generated efficiently** from KGs beyond the scalability limits of current techniques?

1

Ablation of KG-derived context (VoID, ShEx, triples, question-query pairs) in a RAG pipeline, benchmarking open-weight LLMs on SciQA [[Ref 1](#)]

2

A predicate-coverage-aware parallel graph sampling strategy that extends VoID and ShEx generation to billion-triple scale

VoID and ShEx as LLM context

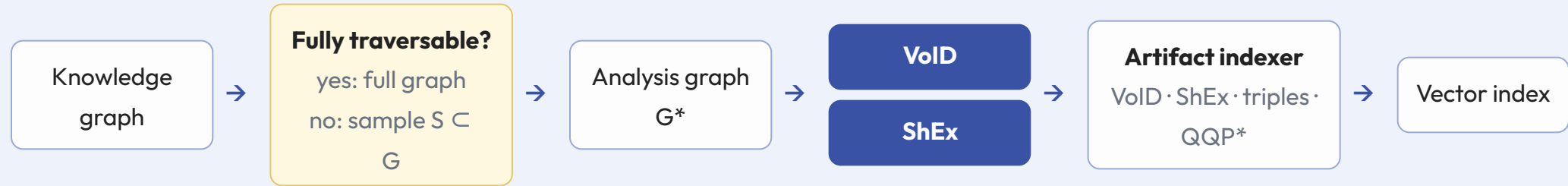
	VoID	ShEx
Describes	What the dataset contains, and how much	What a node of each class looks like
Content	Counts per dataset, class and predicate	Properties, value types, cardinalities
Readability	Verbose; class structure is convoluted	Concise, explicit schema
Retrieval unit	Chunks split by blank lines	One chunk per class shape
In EXYGEN	Generated from the endpoint via SPARQL	Derived from the VoID output



Together they give the LLM structured, class-specific constraints on the KG's schema, with no task-specific supervision.

The EXYGEN-RAG pipeline

EXYGEN · METADATA GENERATION



RAG · QUERY TIME



*QQP = optional example question–query pairs.

Metadata generation and sampling are exposed through the EXYGEN API (exygen.graphia-ssh.eu/docs).

Generating VoID and ShEx

VoID description

Built on [void-generator](#) (Swiss Institute of Bioinformatics), extended with:

Backend-aware queries — light probes detect the triple store (Virtuoso, QLever, ...) and pick specific queries

Two-stage named-graph selection — top- k graphs by triple count ($k = 10$), plus k random graphs from the tail to cover sparse partitions

ShEx schema

A VoID-to-ShEx conversion pipeline applies regular expressions and SPARQL queries to the VoID output to:

Identify class definitions and the properties used with each class

Constrain each property with a value type and cardinality derived from the VoID statistics

Full-scan aggregations are the bottleneck everywhere: across 618 public endpoints, list-only queries succeed on up to 94%, aggregation queries on as few as 25% ([SPORTAL](#) [Ref 3]).

Four retrievable context types

VoID

Split by blank lines (prefixes dropped); dataset-level and per-class/predicate statistics

ShEx

One chunk per class, describing that class's structural constraints

Triples

Sample RDF triples from KG maintainers or representative sampling, embedded per triple

QQP

Example question-SPARQL pairs, provided manually or generated synthetically

Embedding · store · orchestration

Qwen3-Embedding-0.6B · ChromaDB · LangChain

Per component, top-*k*

25 chunks

QQP from SciQA train

top-3 based on similarity

Each type has its **own index** and top-*k*. Results are concatenated into the prompt as context using the following strategies:

- “Base” context: give only one type of information at a time, for e.g. just ShEx or QQP.
- “Pairwise” context: mix two or three context types to see if they help each other, for e.g. VoID + ShEx.
- “Few-shot” enhanced: add QQPs on top of other contexts, for e.g. VoID + ShEx + QQP.

Experimental setup (RQ1)

Benchmark

SciQA: 2,565 question-SPARQL pairs over the ORKG

1,795 train · 257 val · 513 test

Domain-specific, unlikely to be in LLM pretraining

Models

Qwen2.5-Coder-3B

Dense, 3B, 32k context window

Qwen3-Coder-30B-A3B

MoE, 3B active, 128k window

DeepSeek-V3.1

MoE, frontier scale, 128k window

Metrics

Executability (Exec) — does the query run?

Exact Match-RelEx — do the answers match? (Levenshtein ≥ 0.9)

F1 — query token overlap, read with caution

PrefMis — prefix mismatches

Model inference configuration: vLLM on A100 80GB · temperature 0.1 · top-p 0.9 · seed 42.

Results: text-to-SPARQL on SciQA

Context	Exec Qwen-3B	Exec Qwen-30B	Exec DeepSeek	EM-RelEx Qwen-3B	EM-RelEx Qwen-30B	EM-RelEx DeepSeek
No context	.460	.925	.974	.000	.000	.000
ShEx	.508	.976	.994	.000	.000	.000
QQP	.000	.000	.953	.000	.000	.097
VoID + ShEx + Triples	.834	.935	.986	.000	.000	.000
ShEx + QQP	.001	.684	.265	.000	.062	.107
VoID + ShEx + QQP	.001	.382	.980	.000	.011	<u>.329</u>
VoID + Triples + QQP (no ShEx)	.000	.001	.000	.000	.000	.000
ShEx + Triples + QQP	.007	.844	.986	.000	.037	.272
All four (VoID + ShEx + Triples + QQP)	.003	.604	<u>.978</u>	.000	.046	.419

Selected rows from the 14-configuration ablation. Qwen-3B = Qwen2.5-Coder-3B; Qwen-30B = Qwen3-Coder-30B-A3B; DeepSeek = DeepSeek-V3.1.

What the ablation tells us

0.419

Best EM-RelEx: DeepSeek-V3.1 with all four contexts, **no fine-tuning** — about 7× Qwen3-30B's best

0

Correct results from any configuration **without example pairs** — even at 0.98+ executability

≈ 0

Executability for **all models** once ShEx is removed under QQP — ShEx, not scale, decides whether a query runs

0.886

Top F1, from Qwen2.5-Coder-3B — whose queries **almost never execute** (≤ 0.007). Lexical F1 is not correctness

Anatomy of a failure: prefix mismatch

“Can you provide the highest benchmark result, including the metric and score, for the Ball in cup, catch (DM Control 100k) dataset?”

Ground truth

```
PREFIX orkgp: <.../predicate/>
PREFIX orkgc: <.../class/>
PREFIX orkgr: <.../resource/>
SELECT DISTINCT ?metric ?metric_lbl
  (MAX(?value) AS ?score)
WHERE {
  ?dataset a orkgc:Dataset ;
    rdfs:label ?dataset_lbl .
  ?benchmark orkgp:HAS_DATASET ?dataset ;
    orkgp:HAS_EVALUATION ?eval .
```

Generated (F1 = 0.9464, fails to execute)

```
PREFIX orkg0P: <.../predicate/>
PREFIX orkg0C: <.../class/>
PREFIX orkg0R: <.../resource/>
SELECT DISTINCT ?metric ?metric_lbl
  (MAX(?value) AS ?score)
WHERE {
  ?dataset a orkgc:Dataset ;
    rdfs:label ?dataset_lbl .
  ?benchmark orkgp:HAS_DATASET ?dataset ;
    orkgp:HAS_EVALUATION ?eval .
```

Declared longhand prefixes are never used; the generated query uses **undeclared** shorthand ones. Nearly identical text, zero results.

Metadata generation breaks on large KGs

VoID and ShEx need aggregations and Cartesian products over the endpoint. Instead of tolerating incomplete metadata, we **shrink the input**: sample the KG first, prioritizing coverage of every predicate — the graph's full relational vocabulary.

4.94B

OpenCitations Meta

Bibliographic metadata: titles, authors, venues, identifiers. Flat schema.

97.52M

GESIS KG

Social-science datasets, publications, survey instruments. Flat schema.

1.13M

ORKG

Fine-grained ontology of contributions and claims: many class-predicate combinations.

Predicate-coverage-aware parallel sampling

Parallel random walks from **N** start nodes for **H** hops. At each hop, one of three neighbor-selection strategies:

A

Brute force + pruning

Consider all neighbors; weight by predicate novelty (rarer = higher) and keep the top α

B

High degree

Keep the top-D neighbors by degree; a high+low variant adds the lowest-degree ones so sparse nodes are not lost

C

Semantic similarity

Embed nodes with their neighbors (BAAI/bge-m3); keep only nodes below a similarity threshold

Grid

$N \in \{10, 20, 40\} \cdot H \in \{1, 2, 3\} \cdot \alpha \in \{25, 50, 75\}\%$

Metrics

- **Number of Triples (Triples)**: how big the sampled subgraph is.
- **Predicate coverage (%) (PredCov)**: how many of the KG's unique predicates show up at least once in the sample.
- **Missed Triplets (%) (MissTrip)**: how many of the full KG's triples use a predicate that never appears in the sample.
- **Time taken (minutes)**: how long the sampling takes to run.

Sampling results

KG	Strategy (N, H, α)	PredCov	MissTrip	Time
GESIS	Brute force (20, 2, 50%)	30.62%	0.82%	1.67 min
	Top-40 degree (40, 2, 50%)	35.79%	0.71%	7.90 min
	Similarity < 0.8 (40, 3, 25%)	57.86%	0.42%	95.82 min
OCM	Brute force (20, 2, 50%)	76.64%	0.0049%	5.02 min
	Top-10 degree (20, 2, 50%)	76.64%	0.0049%	36.78 min
	Similarity < 0.6 (20, 2, 50%)	76.64%	0.0049%	2.41 min

OCM coverage plateaus at 76.64% because the missing predicates are mostly administrative (e.g. dct:modified, rdfs:comment). Selected rows of 16.

Full graph vs. sampled metadata generation

GESIS KG

19m → 14s

Sampled 0.42% of 97M triples. >80x faster.

OPENCITATIONS META

64m → 11s

Sampled <0.01% of 4.94B triples.
331 queries instead of 604.

ORKG

∞ → 44m

Sampled 18% of triples. Full graph needs ~18M queries: rate-limited, infeasible.

Schema complexity, not triple count, governs tractability. Queries scale with classes, predicates and their pairings — so for schema-rich graphs, sampling is the only path to complete metadata.

ShEx + VoID generation on live endpoints vs. sampled subgraphs (5 attempts, concurrency 5).

Conclusions and limitations

What we showed

- ✓ **ShEx drives executability**; exemplars drive correctness. Both are needed
- ✓ **0.419 EM-RelEx without fine-tuning**; fine-tuned FIRE-SPARQL (0.85) shows the gap
- ✓ **Model scale beats code specialization** once exemplars are in context
- ✓ **Lexical F1 cannot replace** execution-based evaluation
- ✓ **Sampling makes metadata tractable** from million- to billion-triple KGs

Limitations

Retrieval relies on descriptive labels; opaque URIs (e.g. Wikidata) may impact performance

Fine-tuning-free, but not yet exemplar-free

Sample quality measured by proxies, not downstream SPARQL accuracy

One benchmark (SciQA) and three KGs

Next steps · thank you

01

Evaluate generalizability

More KGQA benchmarks and knowledge graphs beyond SciQA

02

Execution feedback

An agentic loop that refines failed or empty queries, e.g. prefix mismatches

03

Synthetic exemplars

Generate question–query pairs for KGs that have none

EXYGEN API docs · exygen.graphia-ssh.eu/docs

Contact · harshdeep.singh@odoma.ch

Funded by the European Union (GRAPHIA, grant 101188018) and the Swiss State Secretariat for Education, Research and Innovation (SERI).



References

- [1] S. Auer, D. A. C. Barone, C. Bartz, E. G. Cortes, M. Y. Jaradeh, O. Karras, M. Koubarakis, D. Mouromtsev, D. Pliukhin, D. Radyush, I. Shilin, M. Stocker, E. Tsalapati, The SciQA Scientific Question Answering Benchmark for Scholarly Knowledge, Scientific Reports 13 (2023) 7240. URL: <https://www.nature.com/articles/s41598-023-33607-z>. DOI:10.1038/s41598- 023-33607- z.
- [2] T. Mecharnia, M. d'Aquin, Performance and Limitations of Fine-Tuned LLMs in SPARQL Query Generation, in: G. A. Gesese, H. Sack, H. Paulheim, A. Merono-Penuela, L. Chen (Eds.), Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK), International Committee on Computational Linguistics, Abu Dhabi, UAE, 2025, pp. 69–77. URL: <https://aclanthology.org/2025.genaik-1.8/>.
- [3] A. Hasnain, Q. Mehmood, S. Sana E Zainab, A. Hogan, SPOTAL: Profiling the Content of Public SPARQL Endpoints, International Journal on Semantic Web and Information Systems 12 (2016) 134–163. URL: <https://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/IJSWIS.2016070105> DOI:10.4018/IJSWIS.2016070105

What is VoID?

The **Vocabulary of Interlinked Datasets** is a W3C RDF vocabulary for describing a linked dataset: *what is in it* and *how much*.

Dataset level — total triples, classes, properties, distinct subjects and objects

Class partitions — per class: how many entities are typed with it

Property partitions — per predicate: how many triples use it, and between which classes

Computed by running aggregation queries over the SPARQL endpoint — which is exactly what gets expensive at scale.

```
:kg a void:Dataset ;  
  void:triples    N ;  
  void:classes    N ;  
  void:classPartition [  
    void:class    orkgc:Paper ;  
    void:entities N ] ;  
  void:propertyPartition [  
    void:property orkgp:P31 ;  
    void:triples  N ] .
```

Schematic VoID description (counts shown as N)

What is ShEx?

Shape Expressions is a schema language for RDF: a concise, human-readable way to say what a node of a given class *should look like*.

Each **shape** lists the expected properties of a class, with their **value types** and **cardinalities**. Originally built for validating RDF data.

*	zero or more
+	one or more
.	any value
[...]	value must be one of the listed terms

```
orkgOC:Introduction {  
  a [ orkgOC:Introduction ]* ;  
  a . + ;  
  orkgOP:hasContent xsd:string* ;  
  rdfs:label xsd:string*  
}
```

ShEx shape EXYGEN generated for the ORKG class orkgOC:Introduction: typed as such (other types allowed), with string content and labels.