Building Questions and Queries Datasets for Knowledge Graphs: a Demo of Q²Forge

Yousouf Taghzouti[1], Franck Michel[2], Tao Jiang[3], Louis-Felix Nothias[3] and Fabien Gandon[4]

[1]Univ. Côte d'Azur, Inria, ICN, I3S, France
[2]Univ. Côte d'Azur, CNRS, Inria, I3S, France
[3]Univ. Côte d'Azur, CNRS, ICN, France
[4]Inria, Univ. Côte d'Azur, CNRS, I3S, France

Abstract

In this paper, we present a demo of how Q²Forge addresses the challenge of generating competency questions and corresponding SPARQL queries for any target Knowledge Graph. It iteratively validates those queries with human feedback and LLM as a judge. Q²Forge is open source, generic, extensible and modular. The demo shows the complete pipeline from competency question formulation to query evaluation, supporting the creation of reference question-query sets.

Demonstrated Resource	Reusable Software and Services
License	GNU Affero General Public License v3.0 or later (AGPL-3.0-or-later)
Online prototype	https://www.w3id.org/q2forge/
Q²Forge	**Repo** https://github.com/Wimmics/q2forge **DOI** 10.5281/zenodo.15388693
Gen²KGBot	**Repo** https://github.com/Wimmics/gen2kgbot **DOI** 10.5281/zenodo.15388687
Demo video	**Teaser** https://youtu.be/E9rgCZzWH4k **Full** https://youtu.be/I3w-jmZRJII

1. Introduction

Datasets of question-query pairs (**Q²sets**) are used to train, test and benchmark question-answering methods, and to document systems and knowledge graphs (KG). To support their creation, we identified the need to provide tools to understand existing KGs and generate or refine corresponding Q²sets. Various methods and tools exist to help and create competency questions (CQ) and equivalent queries [1, 2, 3, 4, 5, 6, 7]. However, to the best of our knowledge, these tools are either domain-specific, extensively manual, or address only specific steps, but do not provide an end-to-end, integrated pipeline. This demo will present the methods, tools and services implemented in **Q²Forge, a web application guiding the user through the steps of a generic, extensible, end-to-end pipeline to generate a reference Q²set i.e. a dataset of (NL question, SPARQL query) pairs tailored to a specific KG.**

2. From a Knowledge Graph to Q²Forge Pipeline

Through an interactive and iterative process, Q²Forge helps users to carry out three main tasks: generate CQs in NL for a target KG, based on information about the KG and its domain; propose SPARQL query counterparts of the CQs, given the KG and its schemata; **test** the SPARQL queries, judge the relevance of question-query pairs, and recommend refinements. To do so, Q²Forge orchestrates the use of

ISWC 2025 Companion Volume, November 2–6, 2025, Nara, Japan
yousouf.taghzouti@univ-cotedazur.fr (Y. Taghzouti); franck.michel@inria.fr (F. Michel); tao.jiang@cnrs.fr (T. Jiang); louis-felix.nothias@cnrs.fr (L. Nothias); fabien.gandon@inria.fr (F. Gandon)
https://youctagh.github.io/ (Y. Taghzouti); https://w3id.org/people/franckmichel (F. Michel); https://orcid.org/0000-0002-5293-3916 (T. Jiang); https://lfnothias.github.io/ (L. Nothias); http://fabien.info/ (F. Gandon)
0000-0003-4509-9537 (Y. Taghzouti); 0000-0001-9064-0463 (F. Michel); 0000-0002-5293-3916 (T. Jiang); 0000-0001-6711-6719 (L. Nothias); 0000-0003-0543-1232 (F. Gandon)

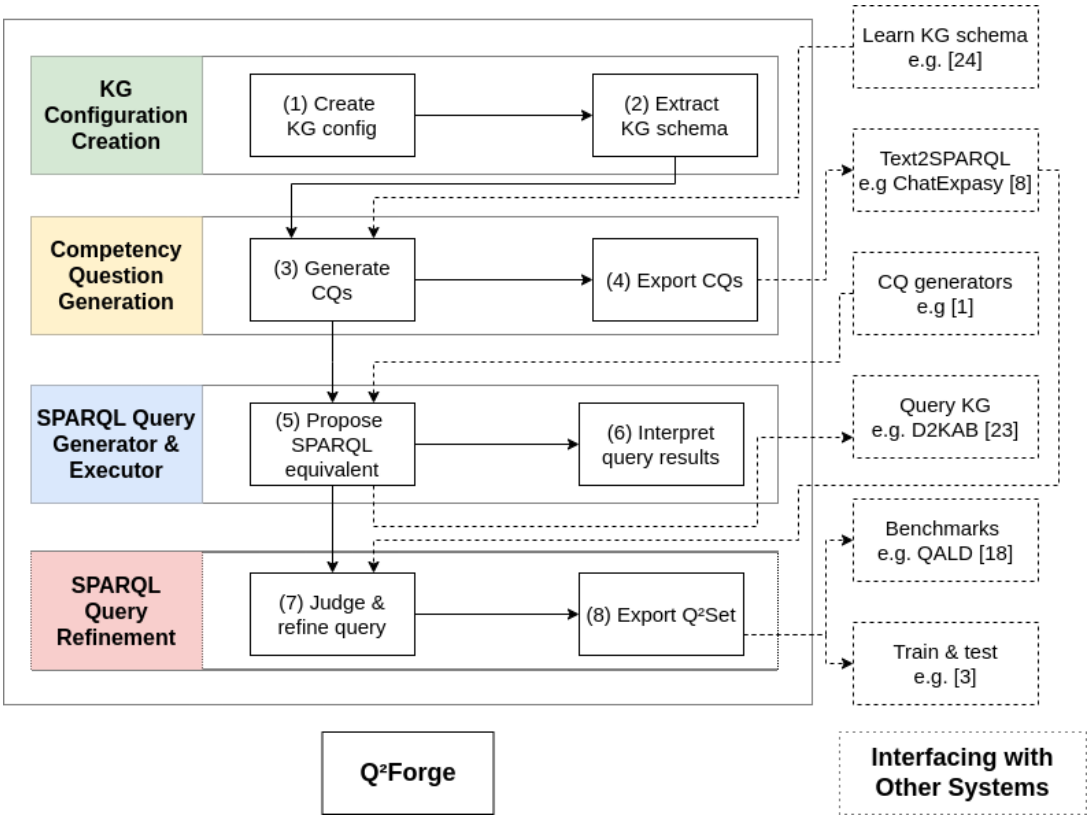


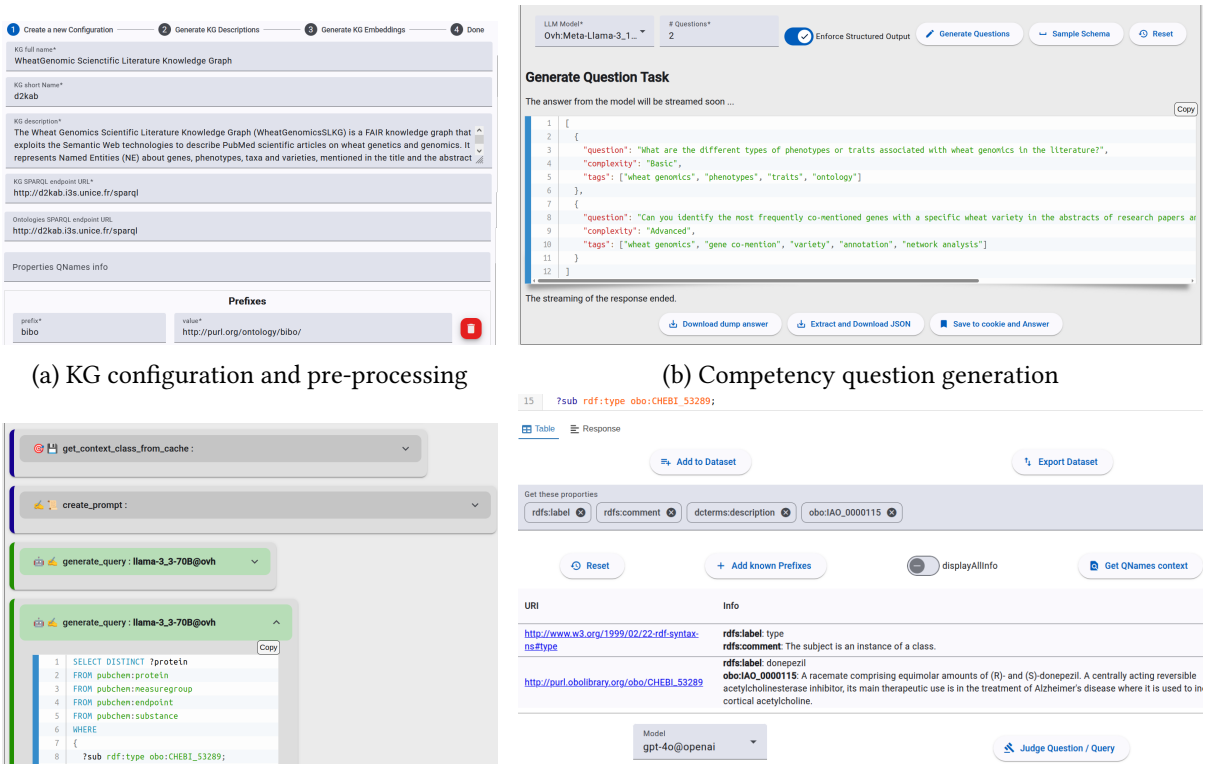
Figure 1: Q^{2}Forge pipeline: resources and services.

various services to manage multiple per-KG configurations, extract the schema of a KG, invoke various language models depending on the task to achieve at each step of the pipeline, etc. The services are invoked through a documented Web API implemented by a back-end server. We provide a prototype implementation of the back-end server called Gen^{2}KGBot. A community may reuse Gen^{2}KGBot as-is, or customize or extend its services to meet their specific needs.

Figure 1 describes the pipeline of Q^{2}Forge: (1) create the configuration for a KG and (2) extract its schema; (3) generate CQs and (4) optionally export them for reuse with another application or for documenting purpose; (5) translate a CQ into SPARQL; (6) execute the query and propose an interpretation of the results; (7) judge the relevance of the question-query pair and allow the user to iteratively refine the query; (8) export the Q^{2}set for reuse with other systems. Note that Q^{2}Forge remains very flexible: a user may follow the whole pipeline, but may also run each task independently by simply importing/-pasting input data and exporting/copying the outputs. The rest of this section further describes the steps depicted in Figure 1.

2.1. KG Configuration and Pre-processing

The pipeline starts with creating a KG configuration (depicted in Figure 2a) where the user provides minimal information about the target KG: a name, a short name used later as an identifier, a textual description, a SPARQL endpoint URL, and the namespaces and prefixes to be used in the SPARQL queries and Turtle descriptions. Optionally, the user may fill in the URL of a SPARQL endpoint hosting the ontologies in case they are not on the same endpoint as the KG itself. The configuration is stored on the back-end server and additional parameters can be edited manually to configure the available language models (seq-to-seq and embedding), where they are hosted (e.g. local vs. cloud resources, vector database etc.), and how they are assigned to each step of the pipeline. For instance, one may choose to use a large model with reasoning capabilities for generating a SPARQL query, but use a smaller model to interpret SPARQL results. Other parameters configure the strategy adopted to serialize ontology



(a) KG configuration and pre-processing

(b) Competency question generation

(c) SPARQL Query Generator/Executor

(d) Query refinement

Figure 2: Screenshots at different stages of the pipeline

classes in the prompts submitted to seq-to-seq models, such as the number of ontology classes to describe and the linearization format used to describe them. Multiple formats are supported (currently Turtle, tuples or a NL format), since different language models may behave differently depending on the selected format.

We then extract from the KG various types of information that will be helpful to carry out the downstream text-to-SPARQL task. This can be ontology classes that are relevant with respect to a NL question, example SPARQL queries, etc. In our implementation, this step first creates a textual description of the classes from the labels and descriptions available in the ontologies, and computes text embeddings thereof. In Figure 2a, this is achieved in steps 2 and 3. Furthermore, there is usually a gap between how an ontology defines classes and how instances of these classes are concretely represented in the KG. Typically, instances may use properties and resources from additional vocabularies that are not explicitly mentioned in the ontology. Therefore, the text-to-SPARQL task requires not only a textual description of the classes, but also a description of how instances of these classes are represented. Gen^{2}KGBot addresses this need by sampling class instances and analyzing the properties and value types they use. Lastly, the user may provide existing examples of NL question and associated SPARQL query. The pre-processing includes computing embeddings of these question-query pairs.

2.2. Competency Question Generation

This step invokes a language model to generates CQs based on various information about the KG: name and description, endpoint URL, list of the used ontologies. This information is either taken from the KG configuration (created in the previous step) or manually entered in a form. The user may also provide any other relevant information, e.g. the abstract of an article describing the KG.

Figure 2b depicts the competency question generation interface. The user can select the language model to be used for the generation of the CQs and the number of CQs to be generated. The model is instructed to return each question with an evaluation of its complexity (Basic, Intermediate or Advanced)

and a set of tags. The *Enforce Structured Output* toggle can be used to compel the model to return the CQs as a JSON-formatted document. Upon completion of the process, the user may download the output as a JSON document, and save it in a browser's cookie for reuse in the next step.

2.3. SPARQL Query Generator/Executor

In this step, the user has the ability to generate a SPARQL query counterpart of a NL question, execute it against a KG, and get an interpretation of the results. The question may originate from the preceding task, or the user may paste a question either hand-crafted or generated by another system.

Q^{2}Forge relies on various strategies provided by Gen^{2}KGBot to accomplish this task, which we refer to as "scenarios". The demo will focus on Scenario 5, depicted in Figure 3. When running a scenario, the steps of that scenario are progressively rendered on the interface, and for the ones that make an LLM call, the response is dynamically streamed to ensure a good user experience. Figure 2c is a snapshot of the interface of the SPARQL Query Generator/Executor. The steps are as follows:

1. **Initial question:** the workflow is initiated by the user posing a NL question.
2. **Question validation:** the question is evaluated to ensure its relevance to the context of the KG. If it is deemed invalid, the workflow stops.
3. **Question pre-processing:** named entities (NEs) are extracted from the question.
4. **Select similar classes:** similarity search between the question and the ontology class descriptions (computed in the KG pre-processing step) selects relevant classes.
5. **Get context information about the classes:** retrieve a description of the properties and value types used with instances of the selected classes.
6. **Generate query:** generate a prompt from a template[1] using the KG configuration and the inputs from the previous steps, and submit it to the configured LLM.
7. **Verify query and retry:** check if a SPARQL query was generated and if it is syntactically correct. If not, generate a retry prompt that includes the last generated answer and the reason for the retry, e.g. syntax errors, and submit this retry prompt to the configured LLM.
8. **Execute the SPARQL query:** if a valid SPARQL query was generated, submit it to the KG endpoint and get the results.
9. Use the configured LLM to **interpret the SPARQL results**.

Scenario 5 is useful as a starting point when no prior question-query pair exists. However, once some pairs have been validated or if some pairs were hand-crafted, they can be added to the context and serve as examples. Scenario 6 can then be applied instead, as it provides the model with relevant example SPARQL queries that can help in generating more accurate queries with fewer refinement iterations.

2.4. SPARQL Query Refinement

In this step, the user can incrementally refine a SPARQL query so that it reflects precisely the question. Figure 2d is a snapshot of the interface, and the process is as follows:

1. First, the query is displayed in a SPARQL editor that highlights potential syntactic errors and can be used to submit the query to the endpoint.
2. To help the user understand the query, Q^{2}Forge can extract the qualified (prefixed) names (QNs) and fully qualified names (FQNs) from the query and get their labels and descriptions. For instance, the label of http://purl.obolibrary.org/obo/CHEBI_53289 is "donepezil", and its description is "a racemate comprising equimolar amounts of (R)- and (S)-donepezil (...)".
3. Then the LLM is asked to judge whether the query matches the given question. It is requested to provide a grade between 0 and 10 along with explanations justifying the grade.

[1] https://github.com/Wimmics/gen2kgbot/blob/master/app/scenarios/scenario_5/prompt.py#L3

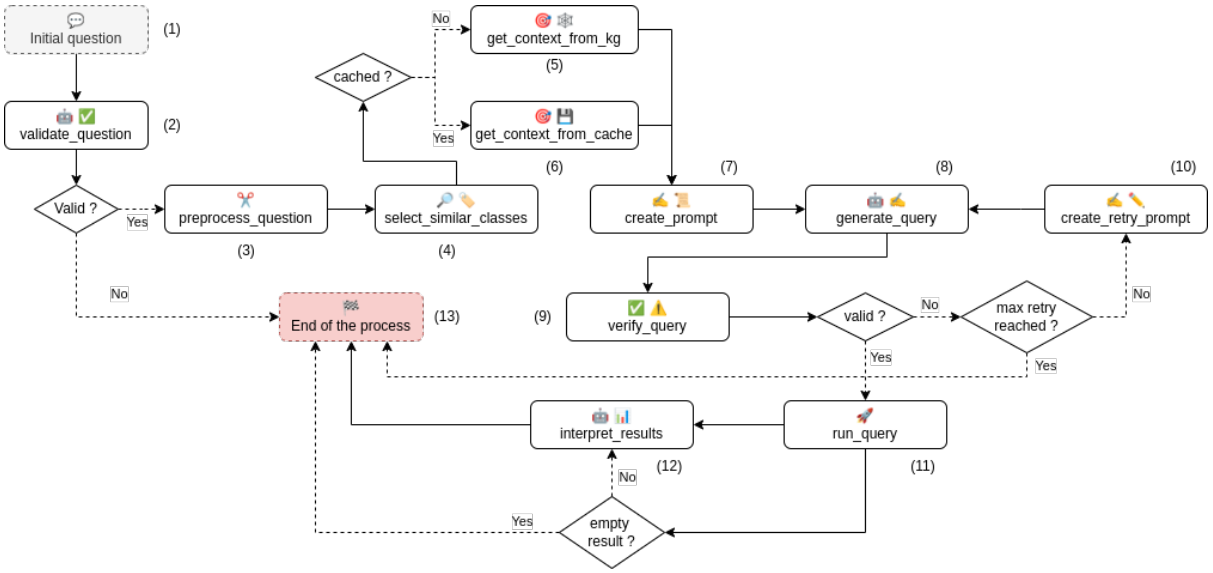


Figure 3: Gen^{2}KGBot SPARQL Query Generator/Executor: workflow of Scenario 5

The user may then iterate as needed: amend the query based on the grade and insights from the model, test it, have the model judge it, etc. Once a satisfying query is reached, the user can add the question-query pair to a dataset and export it in a variety of formats, catering to different use cases.

3. Conclusion and Availability

The article proposes to demonstrate Q^{2}Forge designed to address the generation of competency questions (CQ) in NL, translate them into SPARQL queries, help users to refine those queries, and export high-quality QALD-like Q^{2}sets that can be used for benchmarking, training and evaluating text-to-SPARQL models. Q^{2}Forge is provided under the GNU Affero General Public License v3.0 or later (AGPL-3.0-or-later) license. The code is published on public Github repositories, and the versions used at the time of writing are identified by DOIs to ensure the long-term preservation and citability. A prototype is available for public access and has been assigned a W3ID. The API provided by Gen^{2}KGBot is documented according to the OpenAPI format.[2]

Acknowledgments

This work was supported by the French government through the France 2030 investment plan managed by the National Research Agency (ANR), as part of the Initiative of Excellence Université Côte d'Azur (ANR-15-IDEX-01). Additional support came from French Government's France 2030 investment plan (ANR-22-CPJ2-0048-01), through 3IA Cote d'Azur (ANR-23-IACL-0001) as well as the MetaboLinkAI bilateral project (ANR-24-CE93-0012-01 and SNSF 10002786).

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT and DeepL for the following: Grammar and spelling checks. After using these tools/services, the author(s) reviewed and edited the content as needed, taking full responsibility for the publication's content.

[2] http://w3id.org/q2forge/api/docs/

References

[1] R. Alharbi, V. Tamma, F. Grasso, T. R. Payne, The Role of Generative AI in Competency Question Retrofitting, in: The Semantic Web: ESWC 2024 Satellite Events: Hersonissos, Crete, Greece, May 26–30, 2024, Proceedings, Part I, Springer-Verlag, 2025, pp. 3–13. doi:10.1007/978-3-031-78952-6_1.

[2] F. Ciroku, J. de Berardinis, J. Kim, A. Meroño-Peñuela, V. Presutti, E. Simperl, RevOnt: Reverse Engineering of Competency Questions from Knowledge Graphs via Language Models, Web Semant. 82 (2024). doi:10.1016/j.websem.2024.100822.

[3] X. Pan, J. van Ossenbruggen, V. de Boer, Z. Huang, A RAG Approach for Generating Competency Questions in Ontology Engineering, in: M. Sfakakis, E. Garoufallou, M. Damigos, A. Salaba, C. Papatheodorou (Eds.), Metadata and Semantic Research, Springer Nature Switzerland, 2025, pp. 70–81. doi:10.1007/978-3-031-81974-2_6.

[4] A. Zouaq, F. Martel, What Is the Schema of Your Knowledge Graph? Leveraging Knowledge Graph Embeddings and Clustering for Expressive Taxonomy Learning, in: Proceedings of The International Workshop on Semantic Big Data, SBD '20, Association for Computing Machinery, 2020, pp. 1–6. doi:10.1145/3391274.3393637.

[5] Y. Rebboud, L. Tailhardat, P. Lisena, R. Troncy, Can LLMs Generate Competency Questions?, in: A. Meroño Peñuela, O. Corcho, P. Groth, E. Simperl, V. Tamma, A. G. Nuzzolese, M. Poveda-Villalón, M. Sabou, V. Presutti, I. Celino, A. Revenko, J. Raad, B. Sartini, P. Lisena (Eds.), The Semantic Web: ESWC 2024 Satellite Events, Springer Nature Switzerland, 2025, pp. 71–80. doi:10.1007/978-3-031-78952-6_7.

[6] K. B. Cohen, J.-D. Kim, Evaluation of SPARQL Query Generation from Natural Language Questions, Proceedings of the Joint Workshop on NLP&LOD and SWAIE: Semantic Web, Linked Open Data and Information Extraction 2013 (2013) 3–7.

[7] V. Emonet, J. Bolleman, S. Duvaud, T. M. de Farias, A. C. Sima, LLM-based SPARQL Query Generation from Natural Language over Federated Knowledge Graphs, 2024. URL: https://arxiv.org/abs/2410.06062. arXiv:2410.06062.