QueryGym: A Toolkit for Reproducible LLM-Based Query Reformulation

Amin Bigdeli*
University of Waterloo
Waterloo, Ontario, Canada
abigdeli@uwaterloo.ca

Radin Hamidi Rad*
Mila – Quebec AI Institute
Montreal, Quebec, Canada
radin.hamidi-rad@mila.quebec

Mert Incesu
University of Toronto
Toronto, Ontario, Canada
mert.incesu03@gmail.com

Negar Arabzadeh
University of California, Berkeley
Berkeley, California, USA
negara@berkeley.edu

Ebrahim Bagheri
University of Toronto
Toronto, Ontario, Canada
ebrahim.bagheri@utoronto.ca

Charles L. A. Clarke
University of Waterloo
Waterloo, Ontario, Canada
claclark@gmail.com

Abstract

We present QueryGym, a lightweight, extensible Python toolkit that supports large language model (LLM)-based query reformulation. This is an important tool development since recent work on llm-based query reformulation has shown notable increase in retrieval effectiveness. However, while different authors have sporadically shared the implementation of their methods, there is no unified toolkit that provides a consistent implementation of such methods, which hinders fair comparison, rapid experimentation, consistent benchmarking and reliable deployment. QueryGym addresses this gap by providing a unified framework for implementing, executing, and comparing llm-based reformulation methods. The toolkit offers: (1) a Python API for applying diverse LLM-based methods, (2) a retrieval-agnostic interface supporting integration with backends such as Pyserini and PyTerrier, (3) a centralized prompt management system with versioning and metadata tracking, (4) built-in support for benchmarks like BEIR and MS MARCO, and (5) a completely open-source extensible implementation available to all researchers. QueryGym is publicly available at https://github.com/radinhamidi/QueryGym.

1 Introduction

Query reformulation and expansion play a central role in Information Retrieval (IR), particularly in scenarios where the initial user query is underspecified, ambiguous, or contextually sparse [1, 4, 9, 10]. Building on recent advances in large language models (LLMs), a growing body of work has introduced techniques that employ LLMs to generate enriched or contextualized variants of user queries, with the goal of improving alignment between query intent and relevant documents [2, 12, 14, 15]. These approaches frequently demonstrate strong retrieval gains in zero-shot and few-shot settings, largely due to their reliance on prompt-based generation rather than supervised training [3, 5, 11, 16].

Despite increasing interest in LLM-driven query expansion, progress in this area is constrained by the absence of a dedicated, reusable software framework that enables systematic development and reproducible experimentation. Existing methods [5, 12, 14, 16] present three recurring limitations. First, many approaches lack publicly released implementations [3, 12, 14, 15], and the limited codebases that are available are often tightly bound to specific datasets, prompt templates, or retrieval backends, reducing their applicability across benchmarks and domains [5, 16]. Second, current implementations typically do not provide standardized interfaces, modular components, or extensible system abstractions. As a result, adapting these methods to new datasets, modifying prompting strategies, or integrating with alternative retrieval pipelines requires substantial engineering overhead. Third, reproducibility remains difficult due to undocumented dependencies, hardcoded configurations and prompts, ad hoc scripts, and inconsistent output formats. These challenges, among others, necessitate a unified and extensible toolkit that would facilitate rapid experimentation, systematic assessment of prompt and model variations, and ensures reproducible work in LLM-based query reformulation.

To address these challenges, we propose QueryGym[1], a well-structured, extensible, and publicly available toolkit designed to support research on LLM-based query reformulation. QueryGym is designed to facilitate the development of LLM-based query expansion strategies within a unified software framework. At its core, the toolkit is built around four key capabilities: (1) *a unified reformulation framework* for standardizing the implementation of methods; (2) *a retrieval-agnostic interface* for seamless integration with diverse IR retrieval libraries; (3) *a centralized prompt bank* for reproducible prompt engineering and template management; and (4) *LLM compatibility and reproducibility support* to enable implementation across diverse LLMs and prompting strategies.

At the center of QueryGym is the unified reformulation framework, which provides a standardized execution flow for implementing reformulation methods, managing prompts, interacting with LLMs, and formatting outputs. The toolkit supports batch reformulation with flexible concatenation strategies and robust handling of inputs and outputs. It also offers native support for popular IR datasets, including MS MARCO and BEIR, while also supporting custom and local formats through flexible loaders.

QueryGym also leverages a retrieval-agnostic interface that enables seamless integration with diverse IR pipelines, such as Pyserini [6] and PyTerrier [7]. This design ensures that query reformulation process can be performed through a standardized retrieval setting without requiring any pipeline reimplementation.

To support prompt experimentation, QueryGym introduces a centralized Prompt Bank, which manages versioned templates along with structured metadata. This enables prompt sharing and reuse

*Both authors contributed equally to this research.

[1] https://querygym.readthedocs.io/

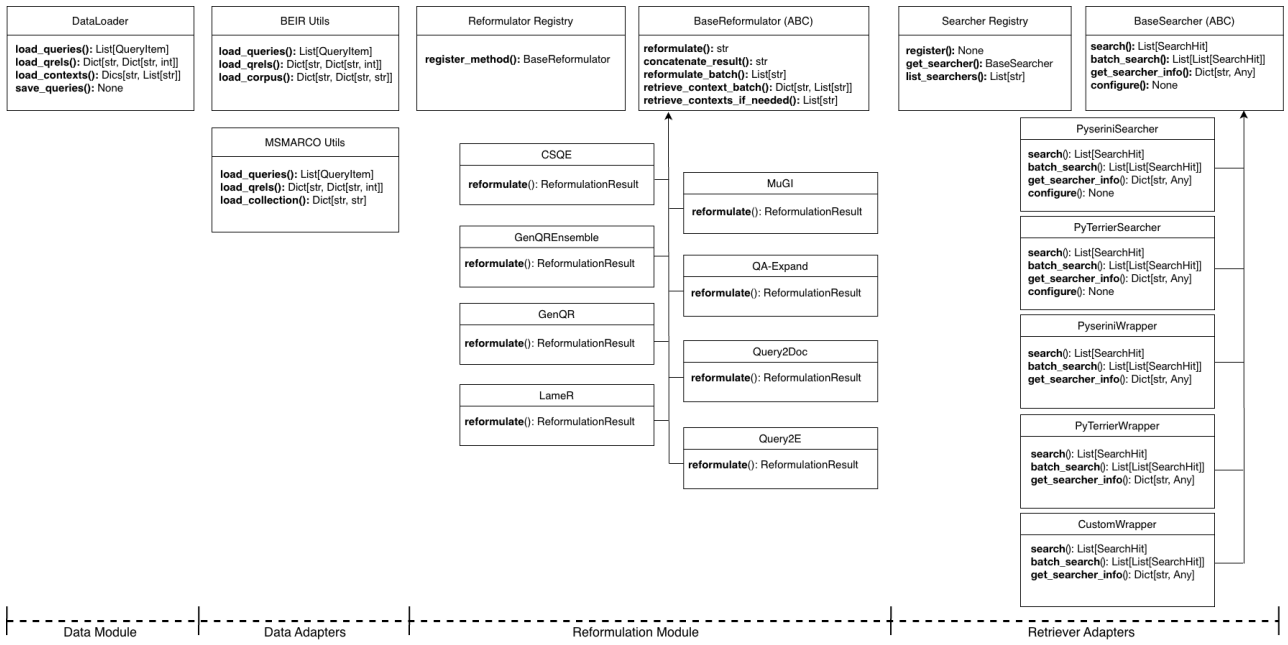

Figure 1: Inheritance hierarchy for the main classes in the `QueryGym` Python package.

across models and datasets and ensures reproducibility and transparency in prompt design. Importantly, the toolkit is fully LLM-compatible and supports both open-source models and API-based LLMs that can be accessed through OpenAI-compatible endpoints. These makes `QueryGym` a suitable toolkit to assess the impact of various LLMs and prompt variations.

Finally, the toolkit is built to help with reproducibility and scale. All experiments are driven by structured configuration files that control prompt version, model parameters, and retrieval settings. Reformulated queries are saved in both retrieval-ready and structured formats, while a CLI and high-level Python API provide flexible entry points for rapid prototyping and large-scale batch runs. Metadata, including prompt identifiers, LLM generation traces, and configuration parameters, is automatically logged to ensure that experimental results can be reliably audited and reproduced.

In our demo presentation, we will present `QueryGym` as a principled, flexible, and reproducible toolkit for LLM-based query reformulation. We (1) introduce the reformulation methods supported by `QueryGym` and illustrate how different methods can be employed; (2) demonstrate how new prompting methods and LLMs can be seamlessly integrated into the framework; (3) show how the unified interface enables fair, reproducible experimentation across datasets and retrieval backends; and (4) walk through real-world usage scenarios using the provided CLI and API, highlighting `QueryGym`'s value for both fast experimentation and scalable pipeline deployment.

2 Toolkit Overview

`QueryGym` is a modular and extensible Python toolkit designed to facilitate systematic research and development in LLM-based query reformulation. It can be easily installed via "`pip install querygym`". The toolkit is organized into four loosely coupled modules including *Data Module and Data Adapters*, *Reformulation Module*, *Retriever Adapters*, and *Configuration Utilities*. It exposes a consistent object-oriented interface for defining reformulation strategies, loading benchmark datasets, integrating with retrieval engines, and managing input/output workflows. Figure 1 presents an overview of `QueryGym` core classes and their relationships.

Data Module and Data Adapters. The data module provides a lightweight, dependency-free interface for ingesting benchmark datasets. At its core is the `DataLoader` class, which handles the loading and saving of queries, qrels, and context passages in TSV or JSONL formats. To streamline experimentation on widely used benchmarks, `QueryGym` includes specialized adapters for datasets like BEIR [13] and MS MARCO [8], with utilities tailored to their respective file structures. This abstraction simplifies switching between datasets and ensures compatibility with retrieval pipelines and experimentation frameworks. The modular design also allows additional dataset adapters to be added with minimal effort, enabling broader applicability across IR benchmarks.

Reformulation Module. At the core of `QueryGym` is a unified reformulation framework centered on an abstract base class that standardizes method execution and encapsulates shared functionality for prompt rendering, context retrieval, and output formatting. The toolkit includes implementations of recent LLM-based reformulation methods drawn from literature, including Query2Doc [14], GenQR [15], GenQREnsemble [2], MuGI [16], QA-Expand [11], LameR [12], Query2E [3], and CSQE [5]. Each method is implemented as a modular subclass, adhering to a common interface, enabling consistent usage, extensibility, and integration.

To support extensibility, `QueryGym` adopts a lightweight decorator-based registration mechanism that allows new methods to be seamlessly added without altering the core framework. Once registered,

these methods are immediately accessible through the toolkit's Python API and CLI. This design facilitates rapid experimentation with novel prompting techniques while ensuring compatibility with the surrounding infrastructure for dataset management and retrieval integration.

Retriever Integration and Wrappers. For query reformulation methods that rely on retrieval context, such as those requiring passages or pseudo-relevance feedback, QueryGym defines a dedicated retriever abstraction through the `BaseSearcher` interface. This interface decouples reformulation logic from backend retrieval, supporting both single-query and batch retrieval workflows. It is compatible with widely-used IR toolkits such as Pyserini [6] and PyTerrier [7], which are integrated via dedicated wrappers.

Searchers can be instantiated dynamically through a registry-based mechanism, allowing flexible configuration at runtime. Provided wrappers, including support for Pyserini, PyTerrier, and custom search engines, enable seamless interoperability with existing IR tools. This retrieval-agnostic architecture ensures that reformulation methods can operate consistently across different retrieval backends without requiring changes to their core implementation.

Prompt Management and Configuration. Prompting logic is centralized through a YAML-based Prompt Bank, which stores template definitions along with metadata, role formatting (system/user/assistant), and version identifiers. Prompts are rendered dynamically with variable and are logged for full traceability. This enables systematic prompt tuning and sharing across experiments.

Overall, QueryGym has a clear separation between data handling, reformulation logic, retrieval integration, and prompt configuration, which allows researchers to experiment with new methods, prompts, and retrieval strategies without entangling components. By providing a consistent API, structured configuration, and interoperability with widely used IR toolkits, QueryGym lowers the barrier to reproducible experimentation and scalable deployment.

3 Demonstrating Use Cases

To illustrate the utility of QueryGym, we present several representative use cases that demonstrate how the toolkit facilitates query reformulation across different levels of complexity and integration. These examples highlight QueryGym's suitability for both rapid prototyping and scalable experimentation in realistic setting.

Basic Query Reformulation. Figure 2 shows a basic example in which a set of user queries is reformulated using a single method and a specified LLM. This simple workflow is representative of lightweight experimentation, where researchers can iterate over prompting strategies and inspect reformulation outputs with minimal setup. The toolkit automatically handles batch processing, progress tracking, and result formatting. Results include both reformulated queries and method-specific metadata, enabling comprehensive downstream analysis and evaluation.

Context-Based Reformulation with Retrieval. Query reformulation methods that rely on external context such as top-ranked passages can be directly integrated with retrieval engines using QueryGym's retriever abstraction. Figure 3 illustrates an end-to-end pipeline where a context-aware method is applied to a benchmark dataset using a prebuilt Pyserini index. The retrieval system is seamlessly wrapped using QueryGym's utility functions, allowing

```python
import querygym as qg
# Load queries
queries = qg.load_queries("examples/tiny_queries.tsv")
# Create reformulator
reformulator = qg.create_reformulator("query2doc", model="gpt-4")
# Reformulate
results = reformulator.reformulate_batch(queries)
# Show results
for r in results:
    print(f{r.qid}: {r.original}  {r.reformulated})
```

Figure 2: Example usage of `QueryGym` for query reformulation.

reformulation strategies to incorporate retrieved content without altering core logic or re-implementing IR components.

This retrieval-agnostic integration allows researchers to run retrieval-augmented prompting methods in realistic scenarios using widely adopted toolkits. QueryGym's interface ensures compatibility with both batch and single-query workflows and supports configuration of retrieval parameters such as ranking models and passage cutoffs. By bridging LLM-based reformulation with established IR infrastructure, QueryGym enables reproducible experimentation within modular and extensible pipelines.

```python
import querygym as qg
from pyserini.search.lucene import LuceneSearcher
# User configuration
DATA_DIR = "data"
# Load queries
queries = qg.loaders.msmarco.load_queries(f"{DATA_DIR}/queries.tsv")
# Initialize searcher
searcher = LuceneSearcher.from_prebuilt_index('msmarco-v1-passage')
searcher.set_bm25(k1=0.9, b=0.4)
# Wrap searcher and create reformulator
wrapped_searcher = qg.wrap_pyserini_searcher(searcher, answer_key="contents")
reformulator = qg.create_reformulator(
    method_name="csqe",
    model="gpt-4",
    params={
        "searcher": wrapped_searcher,
        "retrieval_k": 10,
        "gen_passages": 5,
    },
    llm_config={
        "temperature": 1.0,
        "max_tokens": 128,
    }
)
# Reformulate
results = reformulator.reformulate_batch(queries)
```

Figure 3: Integrated pipeline for Pyserini retrieval and `QueryGym` reformulation.

Leaderboard and Benchmarking. QueryGym facilitates reproducible method comparison across datasets under controlled experimental conditions. Figure 4 demonstrates a systematic experimentation pipeline that benchmarks six reformulation methods on three MS MARCO datasets with identical LLM configurations.

The pipeline enforces identical experimental configurations across all runs through centralized parameter management. Method-specific requirements are handled programmatically, while outputs are organized hierarchically for immediate downstream evaluation with standard IR tools.

This workflow illustrates how QueryGym facilitates systematic IR research, enabling researchers to scale from single-method to multi-method, multi-dataset experimentation while preserving reproducibility and methodological consistency.

```python
import querygym as qg
from pyserini.search.lucene import LuceneSearcher
# Benchmark configuration
DATASETS = ["dev_small", "trecdl2019", "trecdl2020"]
METHODS = ["genqr", "genqr_ensemble", "query2doc", "qa_expand", "lamer", "csqe"]
CONTEXT_METHODS = {"lamer", "csqe"}
DATA_DIR = "data"
LLM_CONFIG = {
    "model": "gpt-4o",
    "temperature": 0.8,
    "max_tokens": 256,
    "seed": 42
}
# Initialize shared index for all MS MARCO datasets
searcher = LuceneSearcher.from_prebuilt_index('msmarco-v1-passage')
searcher.set_bm25(k1=0.9, b=0.4)
wrapped_searcher = qg.wrap_pyserini_searcher(searcher, answer_key="contents")
for dataset in DATASETS:
    queries = qg.loaders.msmarco.load_queries(f"{DATA_DIR}/{dataset}/queries.tsv")
    for method in METHODS:
        method_params = (
            {"searcher": wrapped_searcher, "retrieval_k": 10}
            if method in CONTEXT_METHODS else {}
        )
        reformulator = qg.create_reformulator(
            method,
            model=LLM_CONFIG["model"],
            params=method_params,
            llm_config={k: v for k, v in LLM_CONFIG.items()
            if k != "model"},
            seed=LLM_CONFIG["seed"]
        )
        reformulated = reformulator.reformulate_batch(queries)
        # Export reformulated queries
        qg.DataLoader.save_queries(
            [qg.QueryItem(r.qid, r.reformulated) for r in reformulated],
            f"benchmark/{dataset}/{method}.tsv"
        )
```

Figure 4: Multi-method benchmarking pipeline across datasets under controlled conditions.

4 Concluding Remarks

QueryGym provides a cohesive and practical environment for investigating LLM-based query reformulation, designed to facilitate extensibility, controlled experimentation, and seamless interoperability with modern retrieval pipelines. The demo will guide attendees through the core capabilities of the toolkit and illustrate how its unified abstractions simplify the development of reformulation strategies. Our demonstration will showcase the following:

- interactive execution of reformulation methods via both the Python API and command-line interface, highlighting the ease of iterating over models, prompts, and reformulation strategies;
- end-to-end workflows that integrate reformulation with retrieval tools such as Pyserini and PyTerrier, illustrating retrieval-agnostic design choices and straightforward backend substitution;
- prompt selection, versioning, and metadata inspection enabled by the centralized prompt bank, demonstrating how prompt management supports transparent and reproducible experimentation;
- benchmarking pipelines that compare multiple reformulation methods across datasets under consistent settings, enabling fair and repeatable experimentation;
- incorporation of a new reformulation method or LLM backend to demonstrate the lightweight extensibility mechanism and the clarity of the underlying abstractions;

Through these examples, the demonstration aims to show how QueryGym enables both rapid prototyping and systematic experimentation of LLM-driven query reformulation, providing a reproducible, modular, and extensible foundation for future research. The complete open-source implementation of QueryGym is available at https://github.com/radinhamidi/QueryGym.

5 Ethical Use of Data and Informed Consent

All datasets used in this study are publicly available and intended for academic research. They contain no personally identifiable information (PII), and no new human subject data was collected. All data was used transparently, responsibly, and in compliance with the terms set by the original providers.

References

[1] Nasreen Abdul-Jaleel, James Allan, W Bruce Croft, Fernando Diaz, Leah Larkey, Xiaoyan Li, Mark D Smucker, and Courtney Wade. 2004. UMass at TREC 2004: Novelty and HARD. (2004).

[2] Kaustubh D Dhole and Eugene Agichtein. 2024. Genqrensemble: Zero-shot llm ensemble prompting for generative query reformulation. In *European Conference on Information Retrieval*. Springer, 326–335.

[3] Rolf Jagerman, Honglei Zhuang, Zhen Qin, Xuanhui Wang, and Michael Bendersky. 2023. Query expansion by prompting large language models. *arXiv preprint arXiv:2305.03653* (2023).

[4] Victor Lavrenko and W Bruce Croft. 2017. Relevance-based language models. In *ACM SIGIR Forum*, Vol. 51. ACM New York, NY, USA, 260–267.

[5] Yibin Lei, Yu Cao, Tianyi Zhou, Tao Shen, and Andrew Yates. 2024. Corpus-Steered Query Expansion with Large Language Models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, St. Julian's, Malta, 393–401. doi:10.18653/v1/2024.eacl-short.34

[6] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*. 2356–2362.

[7] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation inInformation Retrieval using PyTerrier. In *Proceedings of ICTIR 2020*.

[8] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human-generated machine reading comprehension dataset. (2016).

[9] Gustavo Penha, Arthur Câmara, and Claudia Hauff. 2022. Evaluating the robustness of retrieval pipelines with query variation generators. In *European conference on information retrieval*. Springer, 397–412.

[10] Joseph John Rocchio Jr. 1971. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing* (1971).

[11] Wonduk Seo and Seunghyun Lee. 2025. QA-Expand: Multi-Question Answer Generation for Enhanced Query Expansion in Information Retrieval. *arXiv preprint arXiv:2502.08557* (2025).

[12] Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Yibin Lei, Tianyi Zhou, Michael Blumenstein, and Daxin Jiang. 2024. Retrieval-Augmented Retrieval: Large Language Models are Strong Zero-Shot Retriever. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 15933–15946. doi:10.18653/v1/2024.findings-acl.943

[13] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=wCu6T5xFjeJ

[14] Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. *arXiv preprint arXiv:2303.07678* (2023).

[15] Xiao Wang, Sean MacAvaney, Craig Macdonald, and Iadh Ounis. 2023. Generative query reformulation for effective adhoc search. *arXiv preprint arXiv:2308.00415* (2023).

[16] Le Zhang, Yihong Wu, Qian Yang, and Jian-Yun Nie. 2024. Exploring the Best Practices of Query Expansion with Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1872–1883. doi:10.18653/v1/2024.findings-emnlp.103