LLM-QPP Code Documentation

[**https://github.com/D3Mlab/llm-qpp**](https://github.com/D3Mlab/llm-qpp)

# **1. Overview**

**1.1 Experimentation**

* **Experiment start:** Create an ExperimentManager instance (e.g. python run\_experiment.py) and call run\_experiment(self, config\_dir)to pass: 1) config.yaml defining the experiment parameters, and 2) queries (one at a time) to a search agent.
* Search agent:
  + Receives an observation (i.e. a query), initializes the agent state (state\_hist), and executes actions determined by a self.policy instance given state\_hist[-1].
  + At each step, the policy specifies a component method to execute (e.g. DenseRetriever.rank(state). The method's result appends a new state to state\_hist.
  + Results (i.e. state\_hist) are returned from the agent to the experiment manager when the policy decides to terminate the experiment.
* **Experiment end:** maps state\_hist to a .json and TREC results file

**1.2 Evaluation**

* Completely separate from experimentation. Handled by eval\_manager.py to use the pytrec\_eval API to compute retrieval metrics, etc.

# 2. Input Data Form

Each experiment gets its own directory where two files are expected: config.yaml to define the experiment parameters and eval\_config.yaml to define the evaluation parameters (e.g. which metrics to evaluate). Here is an example config.yaml:

logging:

level: INFO

data\_paths:

corpus\_text\_path: data/ROEGEN/q50\_d570/collection.jsonl

emb\_path: data/ROEGEN/wikivoyage\_embed\_trunc/collection\_...\_L6-v2.pkl

queries\_path: data/ROEGEN/q50\_d570/queries.tsv

agent:

agent\_class: GeneralAgent

policy: PipelinePolicy

policy\_steps:

- component: DenseRetriever

method: rank

- component: Prompter

method: rerank\_best\_and\_latest

- component: AgentLogic

method: check\_max\_q\_reforms

- component: Prompter

method: decide\_termination\_best\_docs

- component: Prompter

method: reform\_q\_post\_retr

max\_queries: 3

embedding:

embedder\_class: HuggingFaceEmbedder

model\_name: sentence-transformers/all-MiniLM-L6-v2 #Alibaba-NLP/gte-large-en-v1.5

knn:

knn\_class: ExactKNN

sim\_f: cosine

k: 30

implementation: load\_all\_in\_batches

rerank:

k: 30

llm:

model\_class: GeminiLLM # OpenAILLM #GeminiLLM

model\_name: gemini-1.5-flash #gpt-4o-mini #gemini-1.5-flash #gpt-4o

template\_dir: templates

templates:

post\_retrieval\_reformulation: post\_r\_reform/elaborative\_expansion.jinja2

pre\_retrieval\_reformulation: pre\_r\_reform/uninformed\_q2d.jinja2

reranking: reranking/listwise\_rerank\_w\_city\_format.jinja2

termination: termination/termination.jinja2

As per the data\_paths: field, data for retrieval is defined using :

* corpus\_text\_path: a .jsonl with “docID” and “text” fields such as

{"docID": "0", "text": "solid wood platform bed. good … }

{"docID": "1", "text": "all-clad 7 qt . slow cooker. create delicious slow …}

…

* emb\_path:path to embeddings pkl file
* queries\_path: a .tsv of the queries in the form: <qID> /t <query>

# 3. Agent actions (i.e. policy components)

* Each agent action is defined by a dictionary {component: \_#class name, method: \_#method\_name}, for example, for the pipeline policy, these components are given as a list in config fields [‘agent’][‘policy\_steps’].
* Component text is mapped to classes via the dictionaries in registry.py in the search\_agent package (similar dictionaries are used in the \_\_init\_\_.py files of other packages to map from strings in the config.yaml to class names).

# 4. Writing experiment results

* Handled by ExperimentManager if the search agent returns successfully.
* In the per\_query\_results directory, each query gets a directory named as the query id which contains a detailed\_results.json and a trec\_results\_raw.txt file (which may contain duplicates that need to be removed if LLMs are used).

# 5. Evaluation

• **QRELs**: Relevance judgments are loaded from the path specified in eval\_config.yaml and used to evaluate retrieved results.  
• **Metrics**: Evaluation metrics are defined in eval\_config.yaml under the measures field.   
• **Deduplication**: Ensures the removal of duplicate documents from the raw TREC result files, saving a deduplicated version as trec\_results\_deduplicated.txt.  
• **Per-query Results**: Writes evaluation results to eval\_results.jsonl in the respective query directories.

* A **PDF report** (detailed\_results.pdf) is also generated with retrieved and relevant documents for each query.  
  • **Aggregation**: Computes mean and confidence intervals for each metric across all queries. Results are saved in all\_queries\_eval\_results.jsonl.