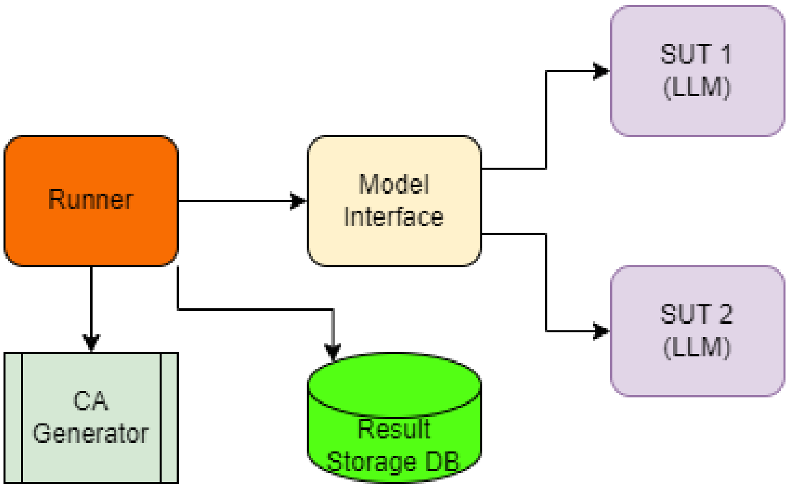
Selected Technologies

This document serves to describe the selected technological components of the envisioned LLM testing framework. The diagram visualizes a draft of the overall technical architecture (note that due to the early stage of methodological development, further changes to this architecture may be required). For each of the components, we detail the selected library/language/paradigm and the reasons for our choice. Because our methodology is not fully developed yet, slight changes to our technology selection are possible in the future.



Criteria

We selected technologies based on the following criteria:

1. Popular with developers. To facilitate the adoption and further development of our project, we selected technologies that are widely used (based on GitHub's evaluation and our own experience).
2. Actively maintained. To mitigate the risk of our underlying technologies becoming obsolete, we impose the requirements that there must be commit or issue tracking activity by the developers of each of our dependencies.
3. License compatibility. Selected technologies may not impose additional licensing restrictions on our own project.

Programming Language

We opted for Python 3[1] as our development language of choice. Together with Rust, it seems to be the most widely used systems and scripting language to date. Unlike Rust, its simplified typing system, accessible syntax and widespread IDE support make it an attractive basis for our project that should be easily adoptable for third party developers.

A second close contender is JavaScript/node.js, which is similarly popular and relatively quick to learn. However, the sanity of the JavaScript ecosystem is questionable, with many projects adding dozens of dependencies for trivial functionality, imposing additional burden on maintainers in the face of breaking changes due to changes in

[1] https://www.python.org/

underlying libraries. Additionally, the lack of true concurrency in JavaScript may be problematic for our purposes of parallel evaluation of test cases.

Orchestration/Virtualization

For the purposes of setting up a reproducible architecture, we opted to use Docker Compose[2], a Docker plugin that allows users to set up entire infrastructures of interdependent containers with a single text file.

Persistence

We decided to utilize a PostgreSQL[3] database as the DBMS to store LLM evaluation results in. To connect to it using Python, we chose the Pyscopg[4] database adapter. We will additionally utilize pandas[5] for importing/exporting data frames to/from the database and numpy[6] for numeric conversions.

Inter-Process Communication

IPC is facilitated through HTTP, specifically using REST APIs. For servers, we decided to utilize the Flask[7] microframework due to its simplicity and popularity. Client-side communication is facilitated through the requests[8] library.

Natural Language Processing

Two libraries are set to be used for NLP: NLTK[9], the Natural Language Toolkit, which contains a host of functionality such as tokenization, word type detection (tagging), stemming, etc. Because we plan to use synonyms to generate additional test cases, we are also evaluating the use of the spaCy[10] NLP library. We further plan to use scikit-learn[11], one of the most popular machine learning libraries for Python, for additional learning tasks such as more flexible tokenization and train/test data splitting.

LLM connectivity

Depending on the LLM, there will be slight additions or changes to our selected technologies. It is highly likely that the popular Transformers library by HuggingFace will be used to implement necessary transformations[12] to queries. The developers of this

[2] https://docs.docker.com/compose/
[3] https://www.postgresql.org
[4] https://www.psycopg.org/
[5] https://pandas.pydata.org/
[6] https://numpy.org/
[7] https://palletsprojects.com/projects/flask/
[8] https://github.com/psf/requests
[9] https://www.nltk.org/
[10] https://spacy.io/
[11] https://scikit-learn.org/
[12] https://github.com/huggingface/transformers

library also commonly recommend the PyTorch[13] framework to train tokenizers and encode queries. Finally, some LLMs (such as Meta's Llama) primarily support the Socket.IO protocol, which we plan to implement using the python-socketio[14] library.

[13] https://pytorch.org/
[14] https://python-socketio.readthedocs.io/en/stable/