Name: Jiankun Wang
Email: jiankun7@illinois.edu

Paper title: SGLang: Efficient Execution of Structured Language Model Programs

1. The problem the paper is trying to tackle.
The paper wants to optimize the execution of LM programs, not just the inference of LLM bare bones. In the frontend, optimize the workflow of prompting and parallelization control. In the runtime, enable the KV cache reuse for prefix tokens.

2. What's the impact of the work, e.g., why is it an important problem to solve?
To complete complex tasks, LM programs require multiple generation calls, advanced prompting techniques, control flow, and structured inputs/outputs. Therefore, it is not only important to optimize the execution of LLM bare bones, but also to enable user-friendly

3. The main proposed idea(s).
1) "language primitives". Simplify the programming workflow of LM programs by introducing primitives in python for controlling prompt state, generation, and parallelism.
2) "prefix sharing". Design a radix tree to retain prefix's KV cache, enabling prefix search, reuse, insertion, and eviction.
3) While using LLMs to follow specific formats with regex, recognize and allow multiple tokens to be decoded in one forward pass.

4. **A summary of your understanding of different components of the proposed technique, e.g., the purpose of critical design choices.**
- The idea of "language primitives" is inspired by the strong instruction-following ability of LLMs. For example, "regex" primitive instructs LLM to output in a specified JSON schema. It simplifies LM programs. Instead of manual string manipulation by programmers, SGlang systems do it behind the scene.
- The idea of "prefix sharing", i.e. radix attention, relies on the point of view of "LM programs". That's because in the scenario where we submit questions to the local LLM model, there are few prefixes to be shared among questions. The prefix sharing technique is not profitable. However, in "LM programs", system prompts required by complex overflow become important, accounting for a significant shared portion of users' input. Therefore, the optimization of "prefix sharing" becomes profitable.
- To increase cache hit rate, 1) page size, the smallest unit in prefix matching, is set as one token. 2) the scheduler prioritizes requests in the waiting queue with longer matched prefixes.
- The frontend-runtime codesign enables optimizations like "frontend hint" while using fork primitive. Basically, the frontend can hint the processing of radix-trees, ensuring the correctness of tree maintenance.
- In summary, they observe and verify structures in both frontend and runtime. So it is named as a *Structured* Generation Language for LLMs.

5. Your perceived strengths and weaknesses of the work, e.g., novelty, significance of improvements, quality of the evaluation, easy-to-use.

Strengths:

1) An efficient frontend and runtime codesign for LM programs. Improved the throughput and latency.
2) The idea of “prefix sharing” can naturally support multi-modal models with images and videos.

Limitation:

1) Focus on prefix sharing on one LLM instance.

Is there room for improvement? If so, which directions you may want to explore or idea you have for improving the techniques?

I would like to work on “prefix sharing” among multiple LLM instances. Radix attention only allows the prefix sharing for the requests processed on one LLM instance. However, in the scenario of multiple LLM instances, one request’s longest cached prefix may exist in another remote LLM instance.

1) Should we schedule requests across LLM instances based on the location of the longest cached prefix? If so, compared with round-robins, will the balance of workload across instances become a new issue?
2) If we still schedule requests across LLM instances in round-robins, should we migrate the remote KV cache to the local? If so, how can we determine and reduce the migration cost?