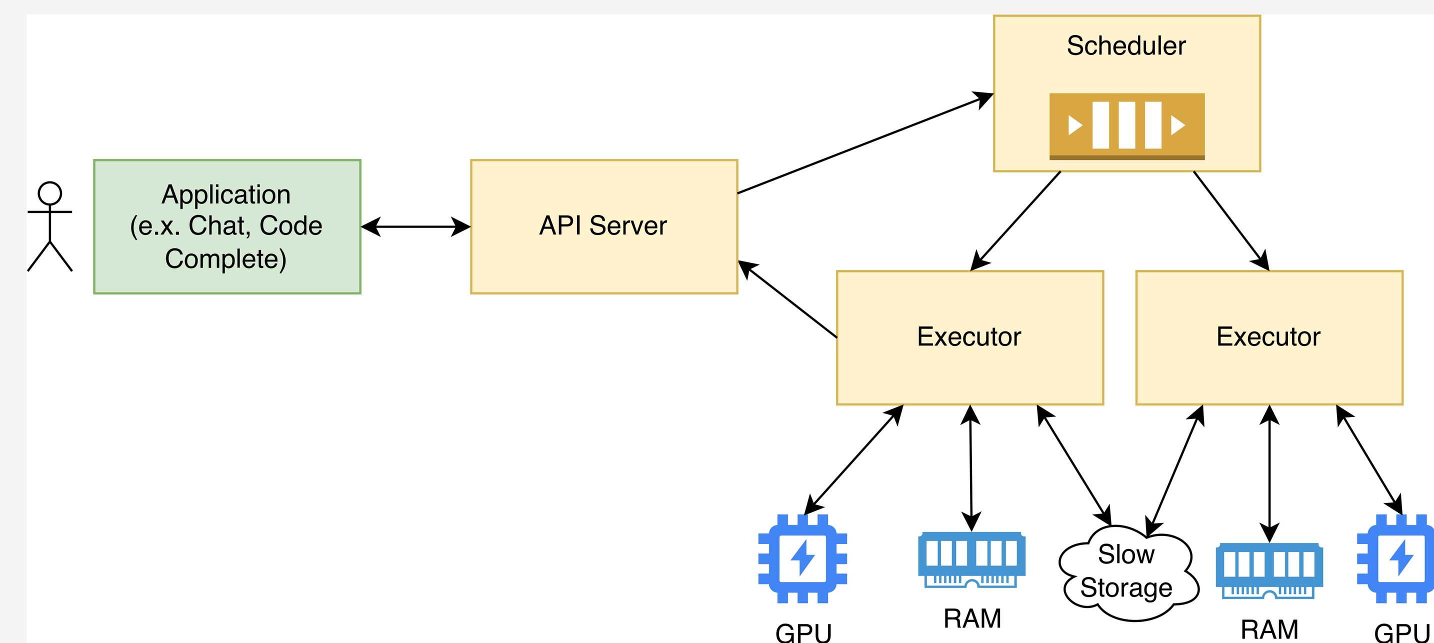
# prefix_vLLM: Optimizing KV Cache Locality for Faster Parallel Language Model Inference

Nikhil Jha, Kevin Wang, Shiyi Cao, William Brandon, Simon Mo, Joseph E Gonzalez, John Kubiatowicz
UC Berkeley

## Motivation

- The use of language models for chatbots, code completion, and data analytics has become extremely popular.
- Existing techniques to accelerate language model inference focus on persisting computed attention values (a "KV Cache"), which speeds up inference of completion requests that share a common prefix.
- These methods persist these caches on a per-request basis, and are not shared between requests.
- However, many common application use cases often involve issuing requests that share a common prefix over a longer time horizon, such data analytics or chat conversations.
- GPU computation is often the bottleneck for language models, so we would like to trade it for other resources to avoid recomputation when it is possible.

## Our Contributions




- We add support for persisting K/V caches to any arbitrary storage medium in vLLM[1]. This involves a new prefix-aware CUDA kernel.
- We implement and benchmark different caching policies to swap these K/V caches between GPU RAM, CPU RAM, and local NVMe SSDs.
- We implement two new performance benchmarks for testing the throughput and latency of LLM serving systems: a realistic chat benchmark and a realistic data analytics benchmark.
- We receive hints from applications where appropriate, and use them to inform the caching policy. For example, if the application is chat, we should expect that the entire context window will be sent back when the user replies, and we should attempt to persist the cache for the whole window.

## Code Completion Technique

```
 7  fn thing() {
 8      FooBar::doesnt_exist();
 9      let x: &str = "hello";
10      String::clear(self: 23);
11  }
```

Hallucinations

Type Hints

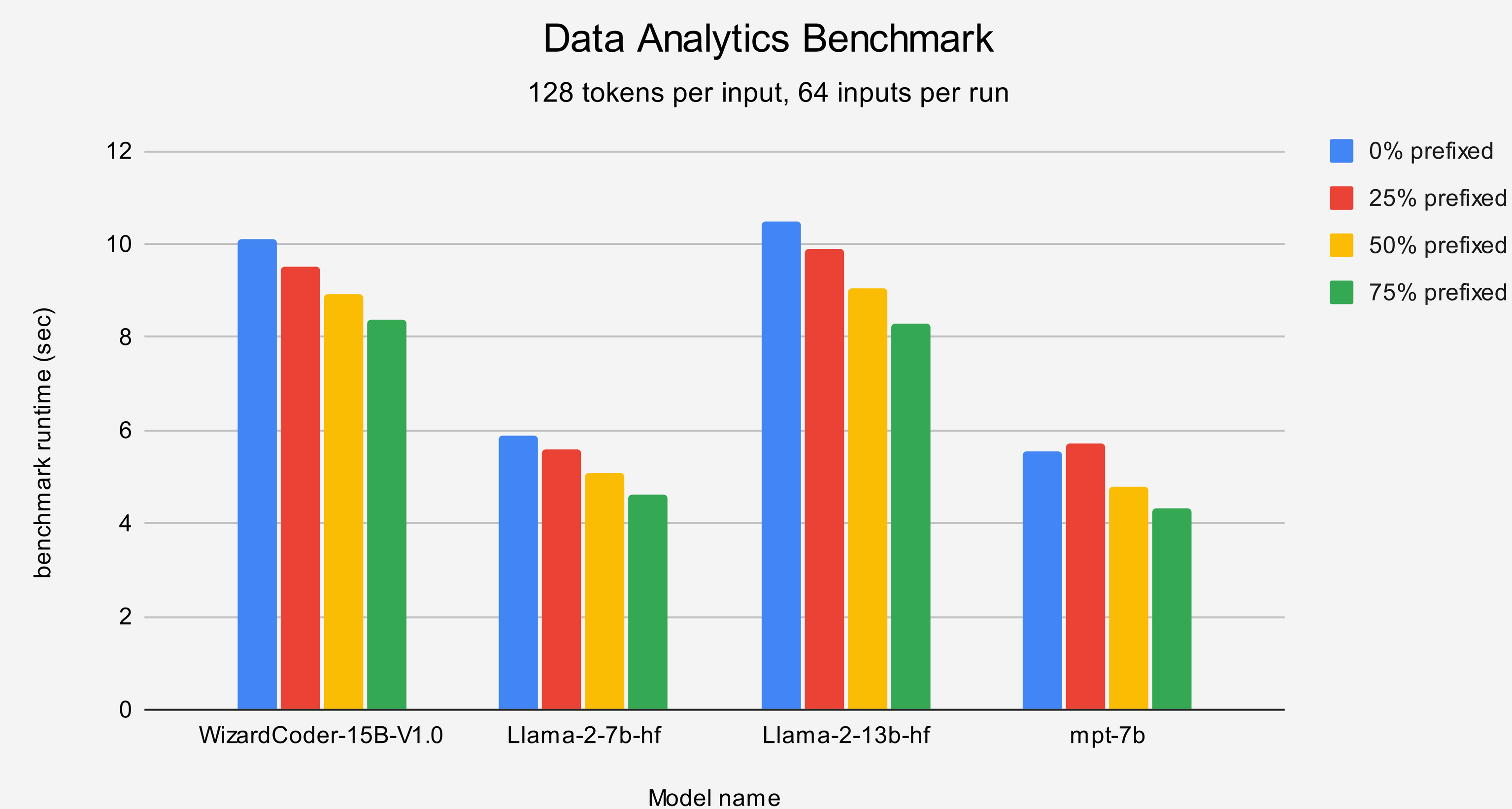
Incorrect Types

- Work around improving code completion techniques often revolves around providing more useful information to the language model. For programming, humans use a language server in their IDE to get instant feedback about what they are typing. However, this is very expensive to give to the LLM because of constant backtracking and re-evaluation.
- Since caching prefixes can cache a lot of existing LLM state, we hypothesized that this method could become feasible as a result.
- We implemented a robust code completion technique that uses the language server to provide context to the LLM by injecting information about types, and masking out tokens that aren't valid via syntactic analysis.

| HumanEval Benchmark Speed | |
|---|---|
| No Prefix Cache | Prefix Cache |
| 30.2s | 26.1s |

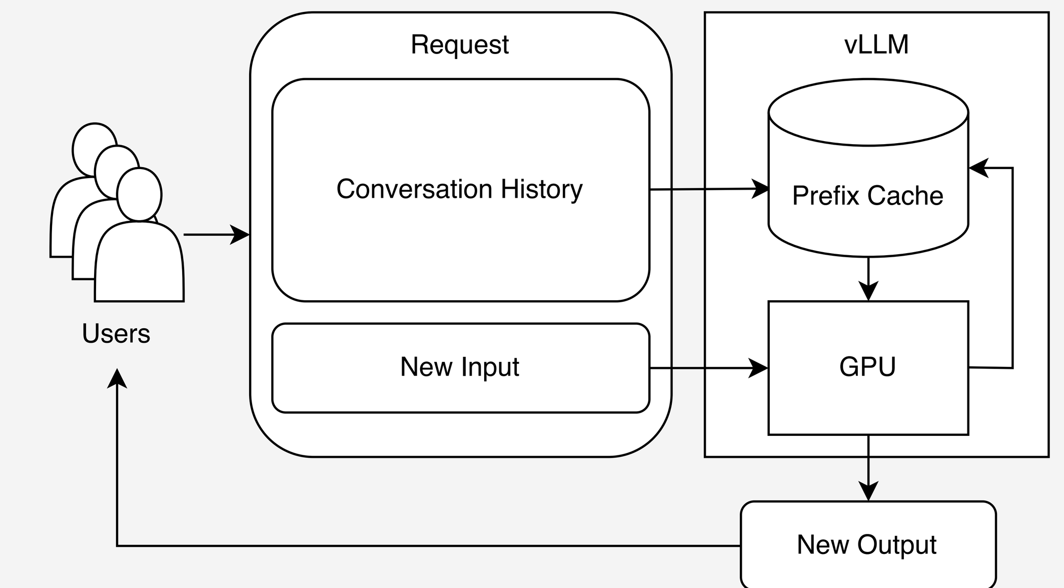
## Data Analytics Benchmark

- The data analytics benchmark is a fairly simple benchmark that takes some data from Twitter, and prefixes it with a prompt that asks the language model to determine if the tweet has a Positive, Neutral, or Negative sentiment. The LLM then outputs two tokens representing this decision.
- The prefix for this workload remains resident entirely in GPU memory for the entire duration of the benchmark, and no swapping is used. As such, this is a good benchmark as a baseline to make sure that our method does not degrade performance of simpler tasks.

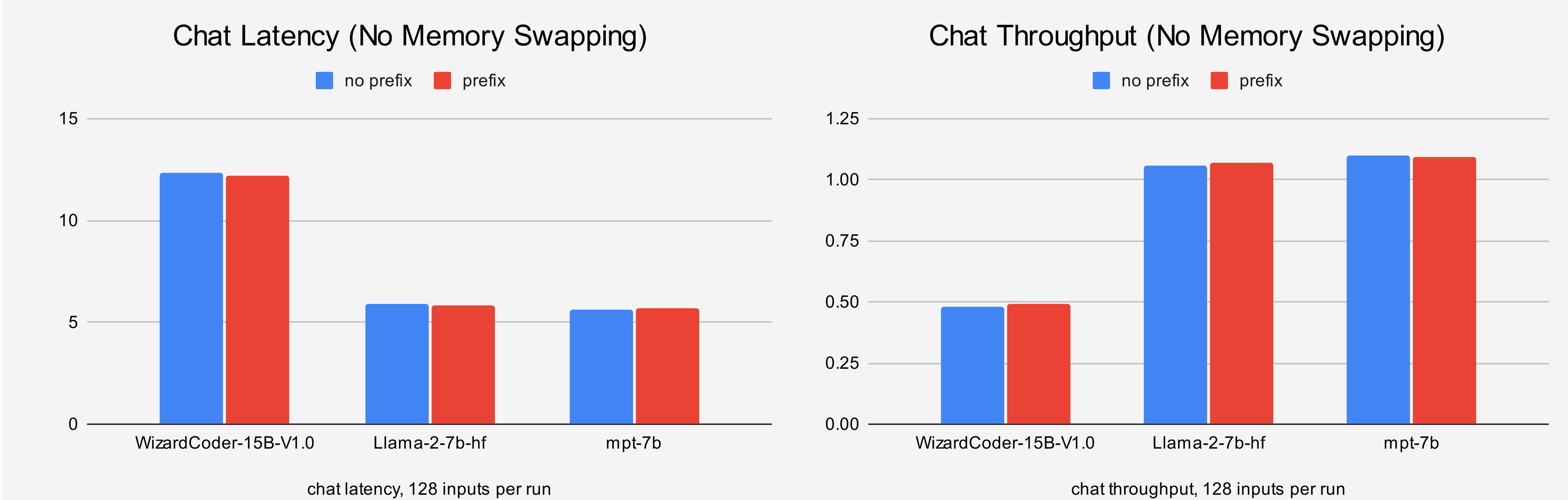


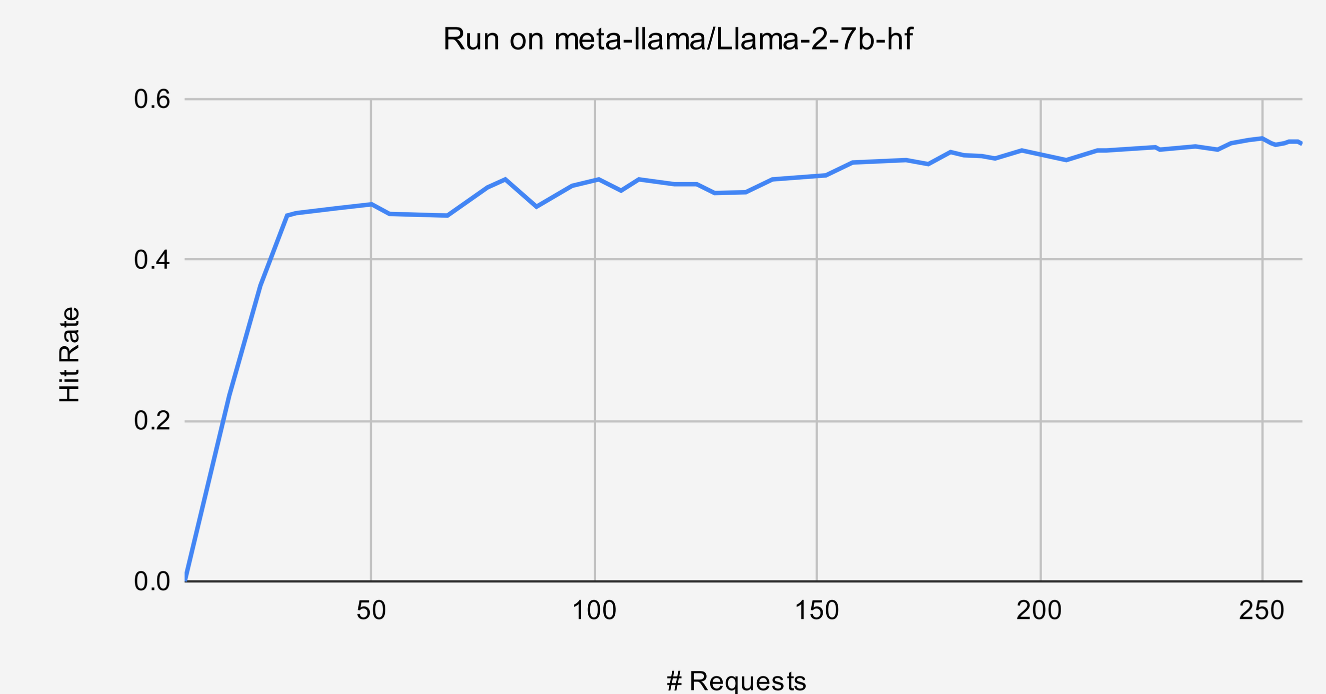

We benchmark various prefix sizes and language models. All benchmarks were run on an NVidia A6000 with 48GB of VRAM.

## Chat Benchmark




- The first benchmark that uses scraped ChatGPT conversations (via the ShareGPT[2] dataset). We split this dataset up among a varying number of concurrent "chatters" who each make requests to the vLLM API server. We record average end-to-end latency for a response to each request, time to first byte, and overall throughput in terms of messages per second.




## Limitations

- We accept an increased latency penalty for the cache with the expectation that this will increase overall throughput. We argue that for many popular LLM applications, the additional latency of a disk read or even a network call is acceptable.
- For very small language models (smaller than 7B parameters), the increased scheduling overhead seems to make the requests slower in the serial case. We are planning on benchmarking the parallel case in our final paper.
- Our implementation is very buggy, and often crashes. Part of this is because it is built on an existing research-quality platform with small bugs that show themselves when it is modified. We are working on making it more robust so we can collect more benchmark data.

https://github.com/nikhiljha/vllm

{nikhiljha, kevwang, shicao}@berkeley.edu, wbrandon@csail.mit.edu, {xmo, jegonzal, kubitron}@berkeley.edu

CS262 Project, Fall 2023