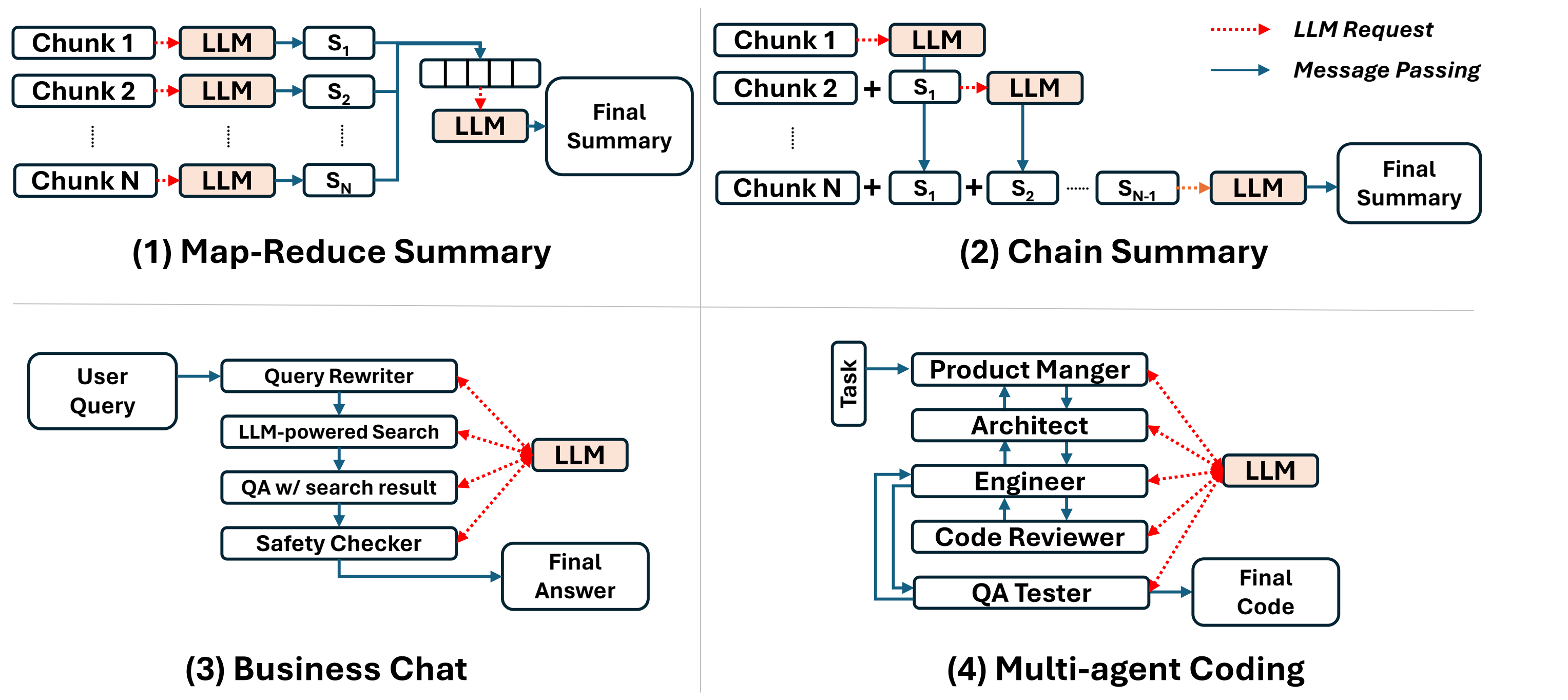
Parrot: Efficient Serving of LLM-based Applications with Semantic Variable

Chaofan Lin [1] Zhenhua Han [2] Chengruidong Zhang [2] Yuqing Yang [2] Fan Yang [2] Chen Chen [1] Lili Qiu [2]

[1]Shanghai Jiao Tong University [2]Microsoft Research Asia

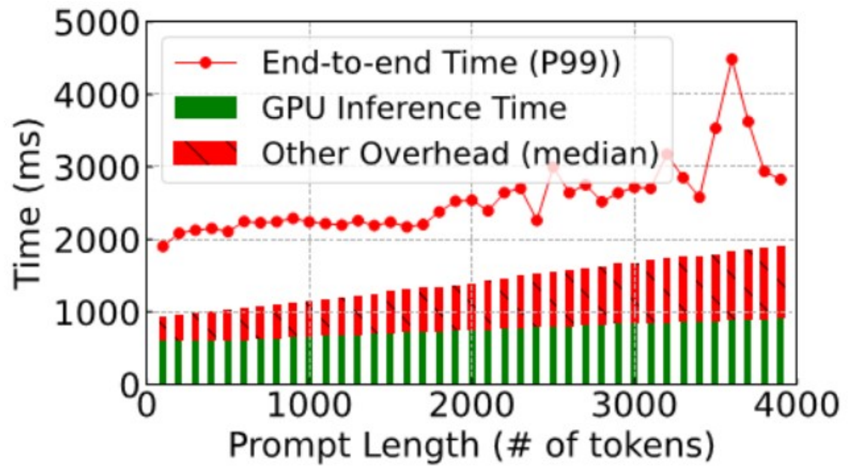
Multi-tenant LLM Services face Diverse Applications (LLM Apps)

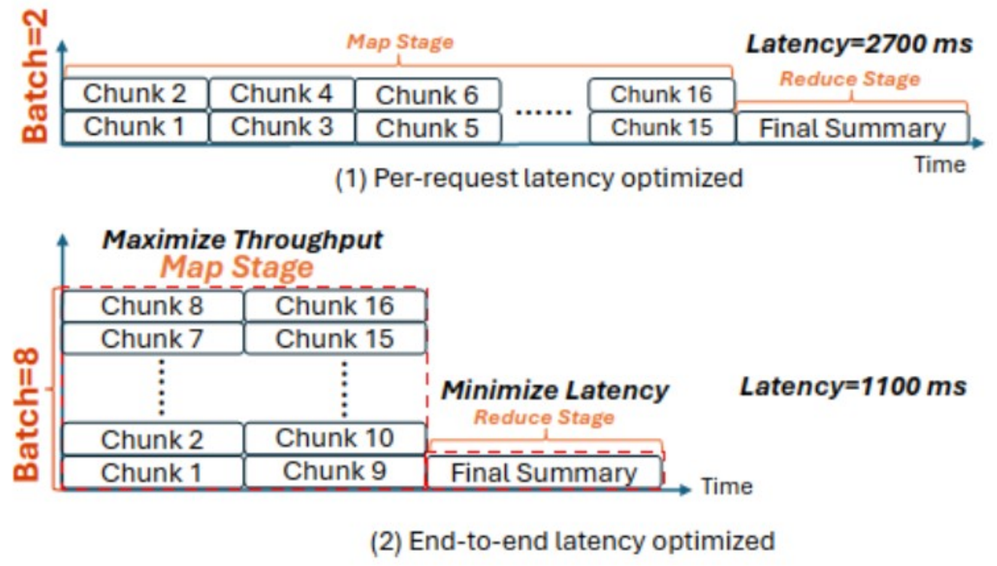


(1) Map-Reduce Summary (2) Chain Summary (3) Business Chat (4) Multi-agent Coding

Public multi-tenant LLM services use too over-simplified request-level API, which *loses the essential application-level information*, leading to sub-optimal *end-to-end performance*.

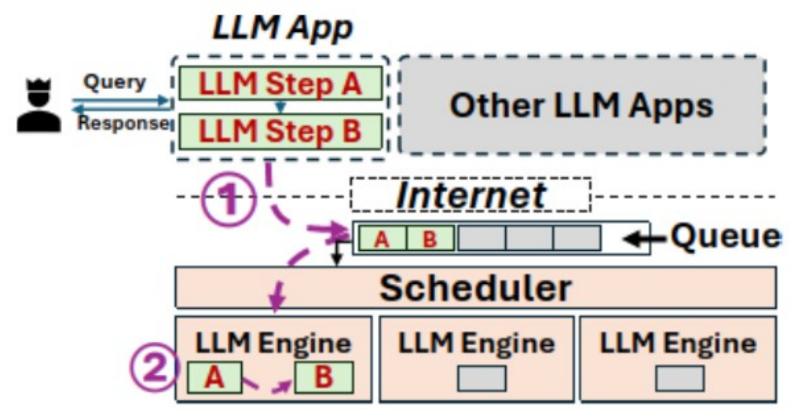
Problems of Multi-tenant Serving

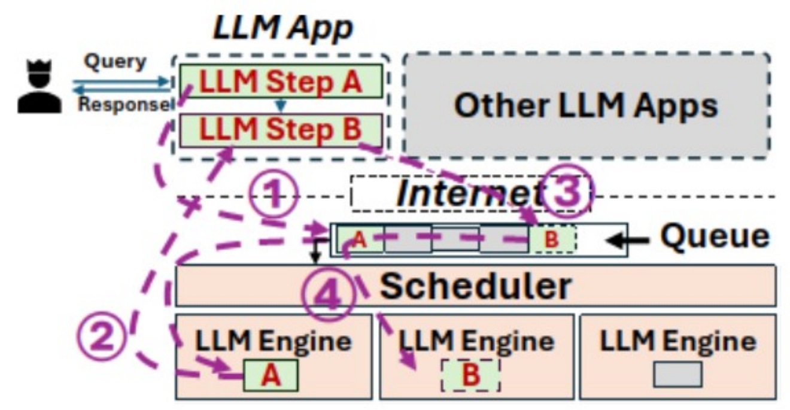




(1) Per-request latency optimized

(2) End-to-end latency optimized





(1) Excessive Overhead of Consecutive Requests;
(2) Misaligned Scheduling Objectives;
(3) Redundant Computations

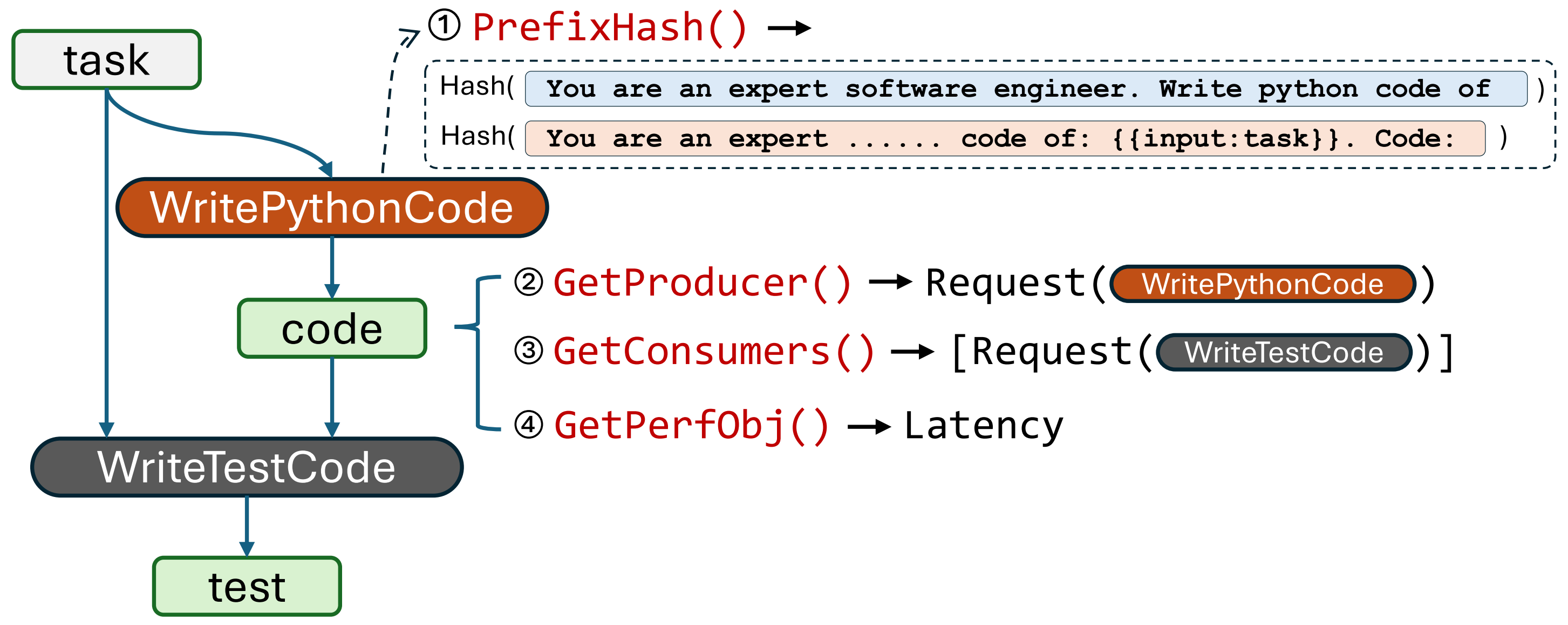
Parrot Design with Semantic Variable as Core Abstraction

```
import Parrot as P
from Parrot.PerformanceCriteria import LATENCY

@P.SemanticFunction
def WritePythonCode(task: P.SemanticVariable):
""" You are an expert software engineer.
    Write python code of {{input:task}}.
    Code: {{output:code}}
"""

@P.SemanticFunction
def WriteTestCode(
    task: P.SemanticVariable,
    code: P.SemanticVariable):
""" You are an experienced QA engineer.
    You write test code for {{input:task}}.
    Code: {{input:code}}.
    Your test code: {{output:test}}
"""

def WriteSnakeGame():
  task = P.SemanticVariable("a snake game")
  code = WritePythonCode(task)
  test = WriteTestCode(task, code)
  return code.get(perf=LATENCY), test.get(perf=LATENCY)
```

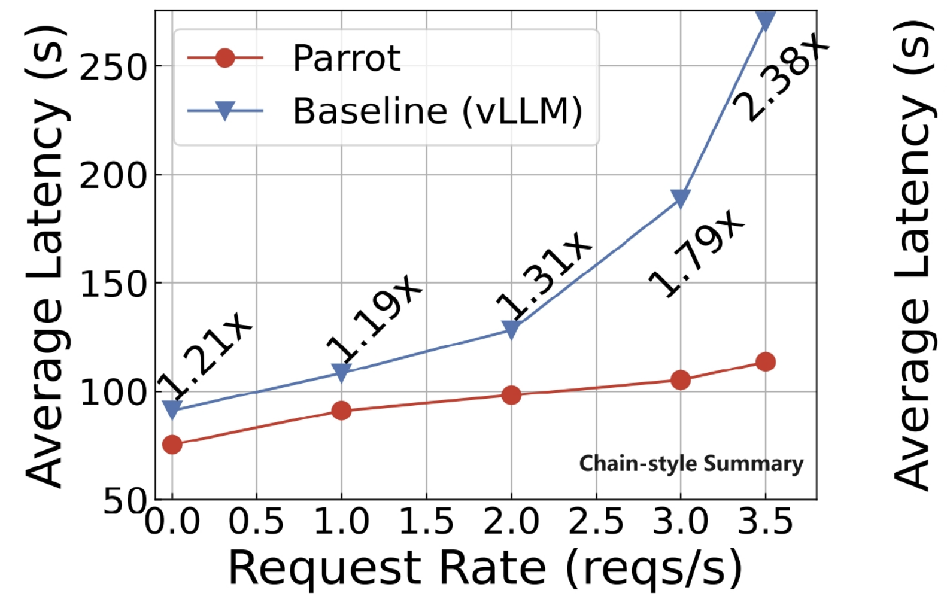

task
WritePythonCode
code
WriteTestCode
test
① PrefixHash() →
Hash(You are an expert software engineer. Write python code of)
Hash(You are an expert ...... code of: {{input:task}}. Code:)
② GetProducer() → Request(WritePythonCode)
③ GetConsumers() → [Request(WriteTestCode)]
④ GetPerfObj() → Latency

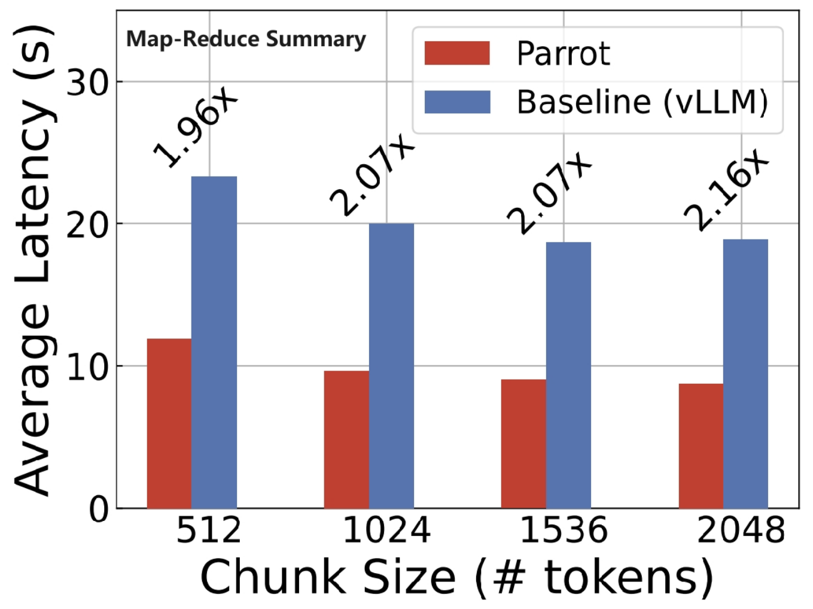
Parrot provides a natural way of *programming LLM applications with Semantic Variable annotations*, enabling DAG analysis and optimizations.

Optimizations with Semantic Variable

1. Serving Dependent Requests.

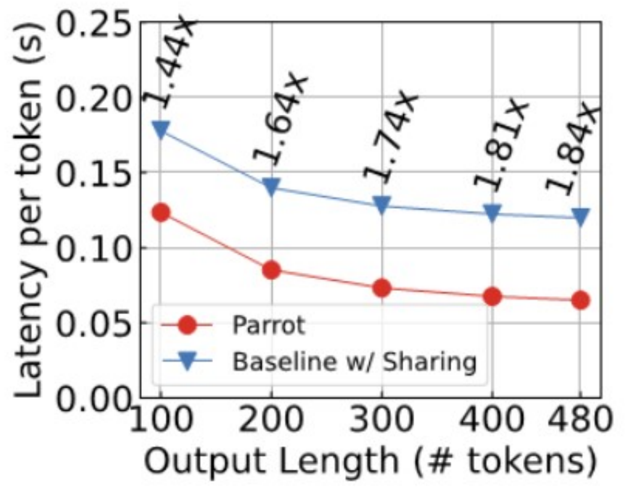
Serving dependent requests. Avoid unnecessary communication. Optimized scheduling.

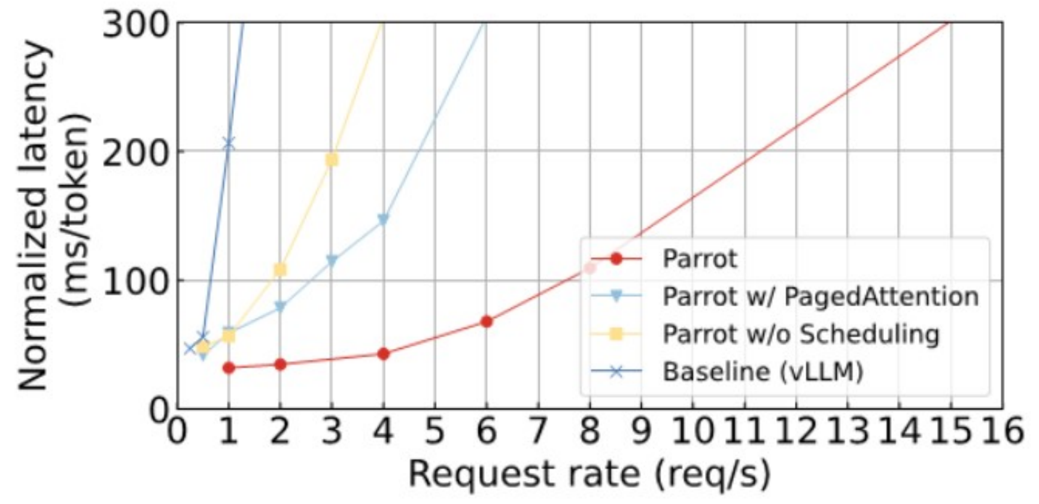




2. Sharing Prompt Prefix.

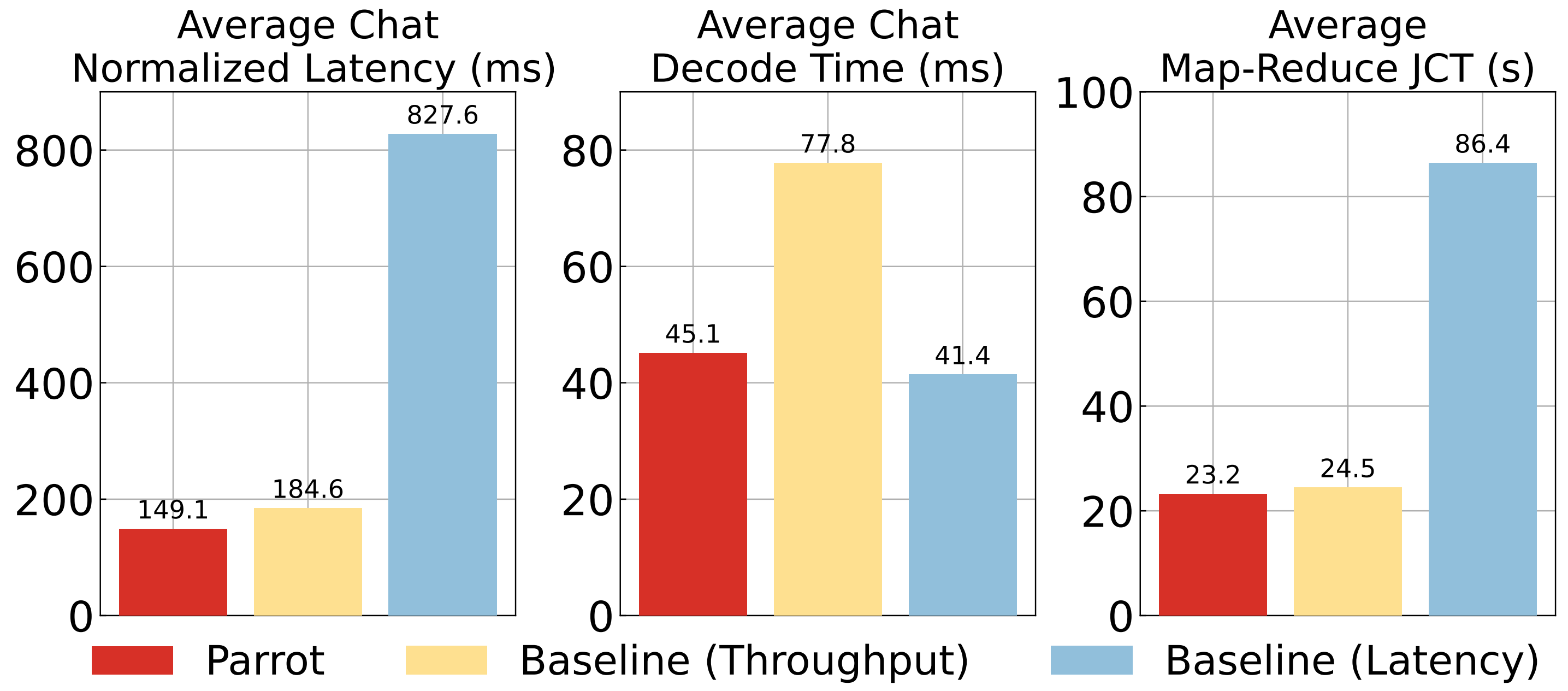
Quickly detect (static & dynamic) sharing, maximize them by scheduling, run them with efficient kernels.





3. App-Centric Scheduling.

Mixed workloads of latency and throughput preference. Parrot achieves the best of both worlds.



https://github.com/microsoft/ParrotServe 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI'24) chaofansys@outlook.com