Rethinking LLM Serving From the Application's Perspective

*TLDR; future LLM serving systems should serve **programs**, not **prompts***

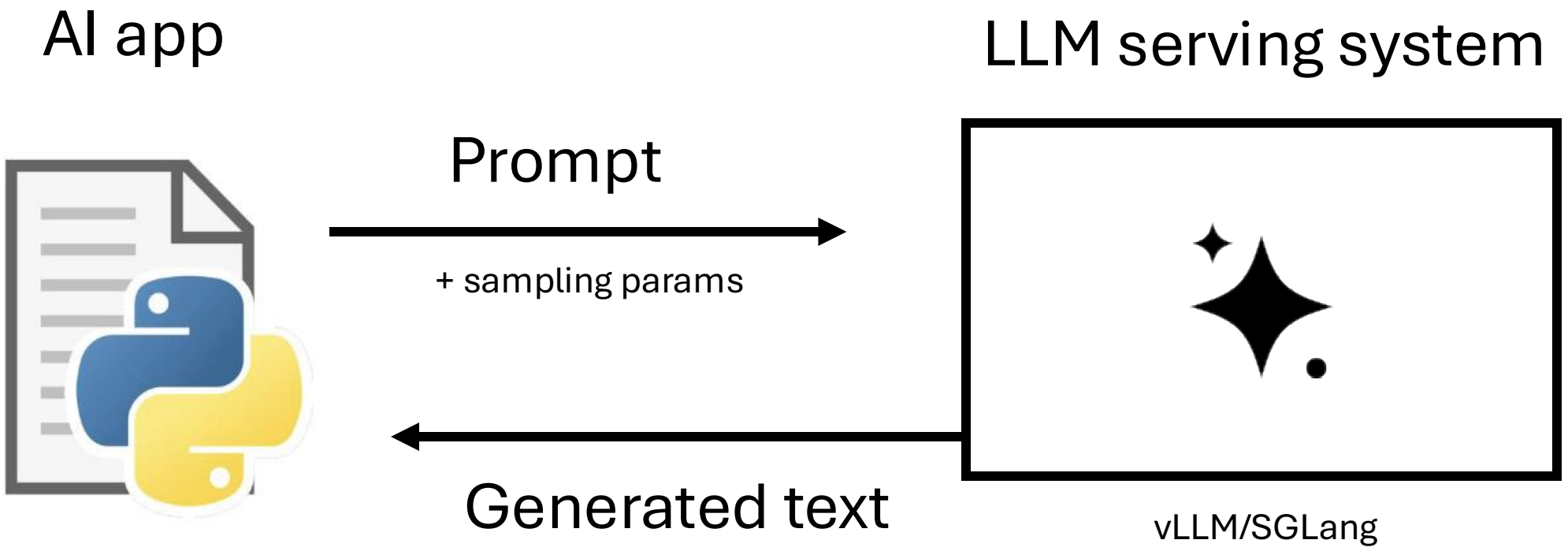
In Gim

Yale University

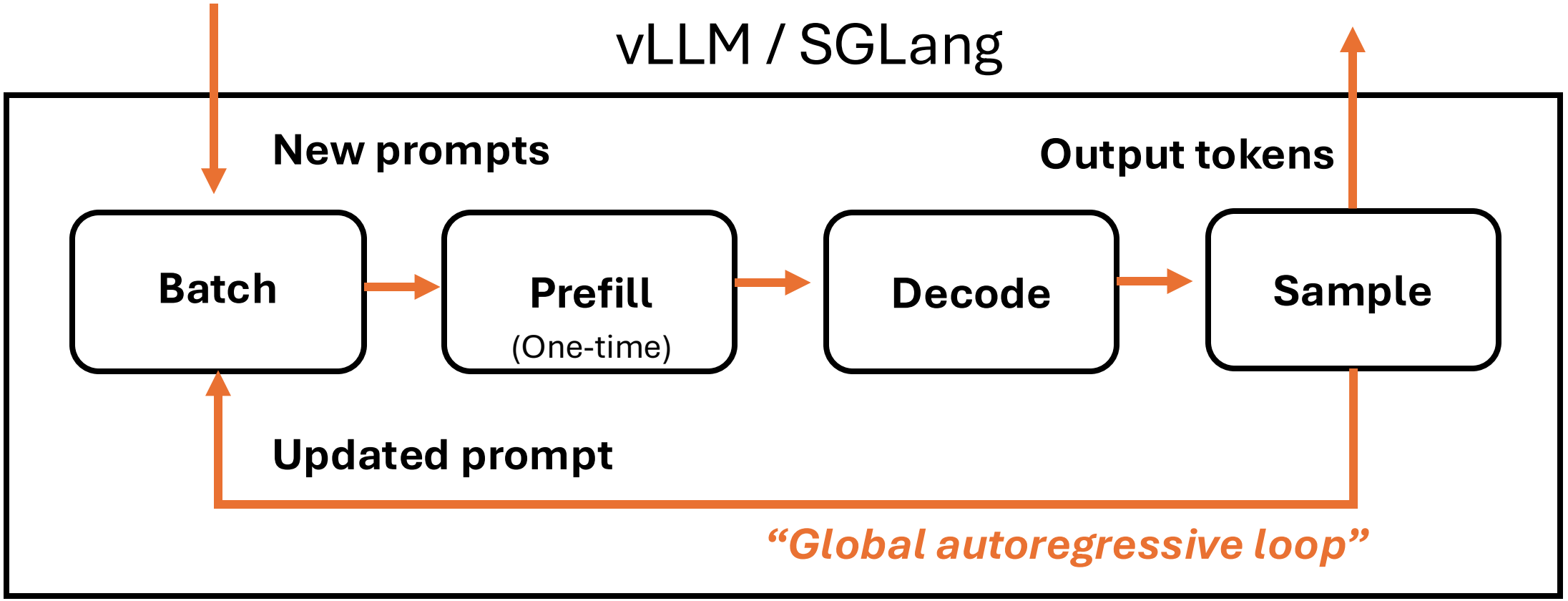
The only API you need to build a billion-dollar AI startup

```
curl https://api.openai.com/v1/responses \
  -d “Write a response to this customer complaint”
```

Today's LLM serving systems are designed for text completion

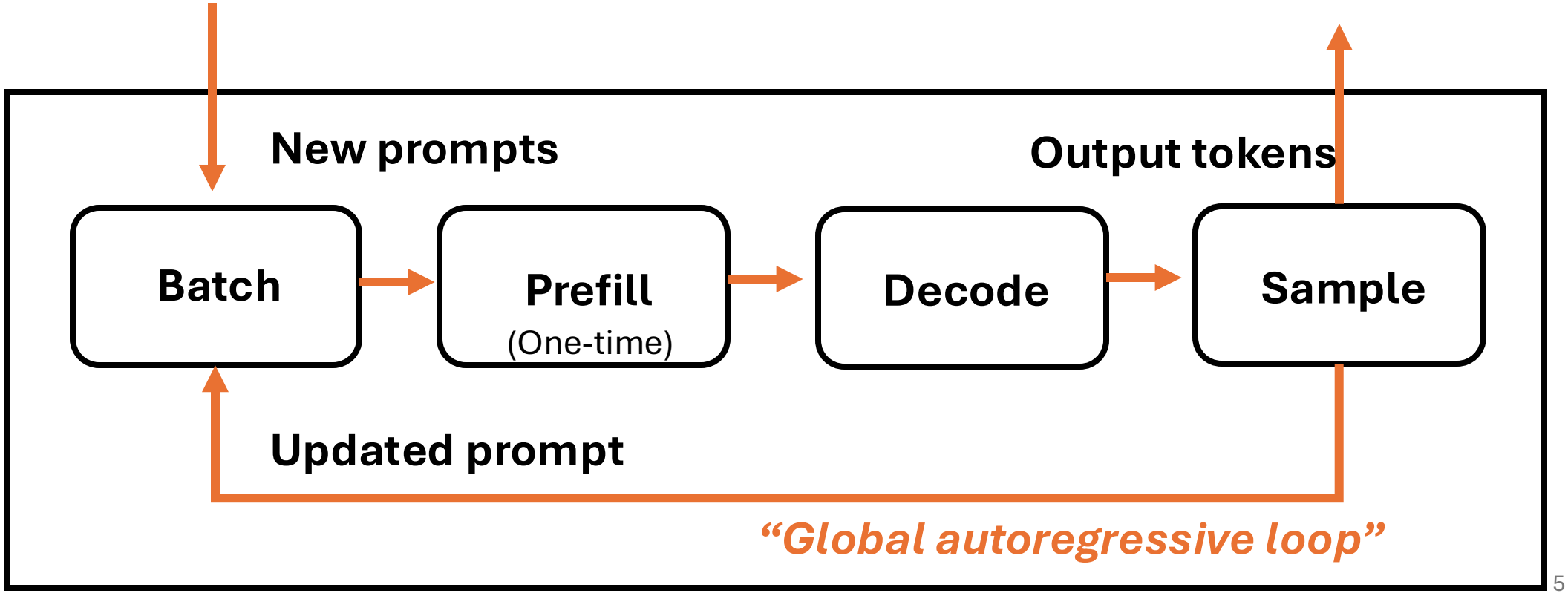


Today's LLM serving systems are designed for text completion



Three implicit design choices behind “prompt serving”:

- KV cache is transient
- Generation is closed-loop
- Serving is stateless



“Prompt serving” is too inflexible for emerging LLM applications

1. Inference
Inefficiency

2. Integration
Friction

3. Implementation
Challenge

Imagine building an AI code completion tool

```
from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

class Sequence(nn.Module):
    def __init__(self):
        super(Sequence, self).__init__()
        self.lstm1 = nn.LSTMCell(1, 51)
        self.lstm2 = nn.LSTMCell(51, 51)
        self.linear = nn.Linear(51, 1)

    def forward(self, input, future = 0):
        outputs = []
        h_t = torch.zeros(input.size(0), 51, dtype=torch.double)
```

Challenge 1. Inference efficiency

Scenario: User's input updates the suggestion.

Prompt =

System prompt

Your job is to predict the most probable next line of code given the …

+

Context
(source code files)

Due to the large overlaps in prompts,
Inter-request KV cache reuse is essential for efficiency

Challenge 1. Inference inefficiency

KV cache management is often automatic, system-wide policy
(e.g., OpenAI's prompt caching, vLLM's automatic prefix caching)

This precludes application-driven optimizations

User deleted source files. Drop related KV cache to save memory.

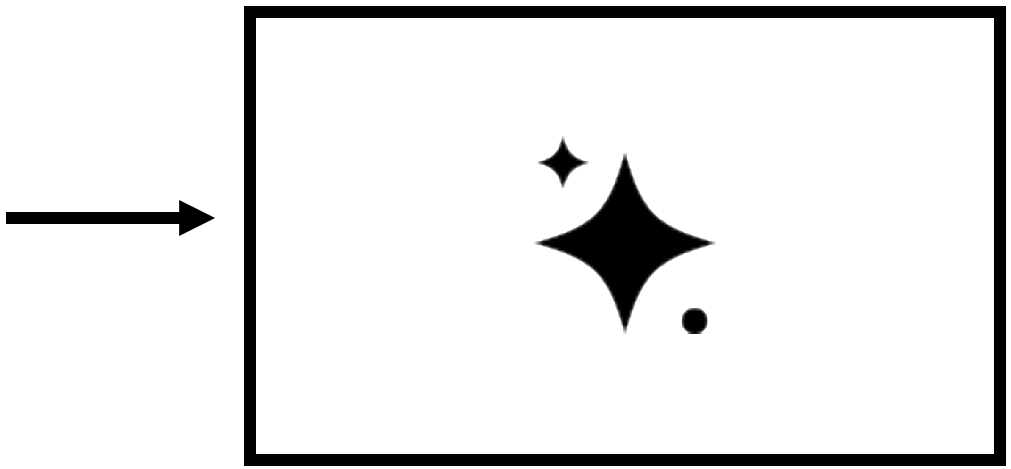
Stats suggest this file won't be used soon. Don't keep its KV cache.

In case the user presses Ctrl+Z, keep the previous KV cache briefly.

Challenge 2. Integration friction

Scenario: Augment generation with relevant API documentation

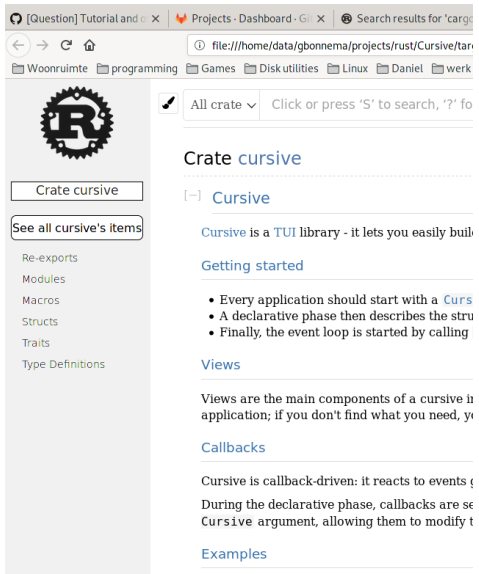
Prompt: “If unsure about an API during generation, output **<read>API name</read>** and pause”



Challenge 2. Integration friction

Each external interaction induces two extra boundary crossings

Data/Tool



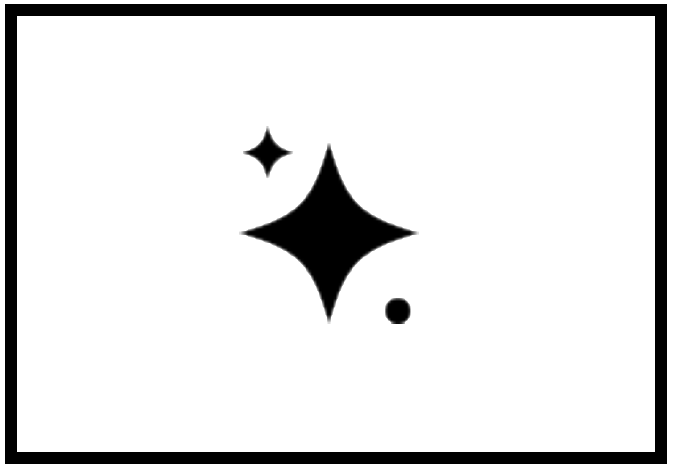
(3) Retrieve data

(1) Original prompt

(2) <read> ...

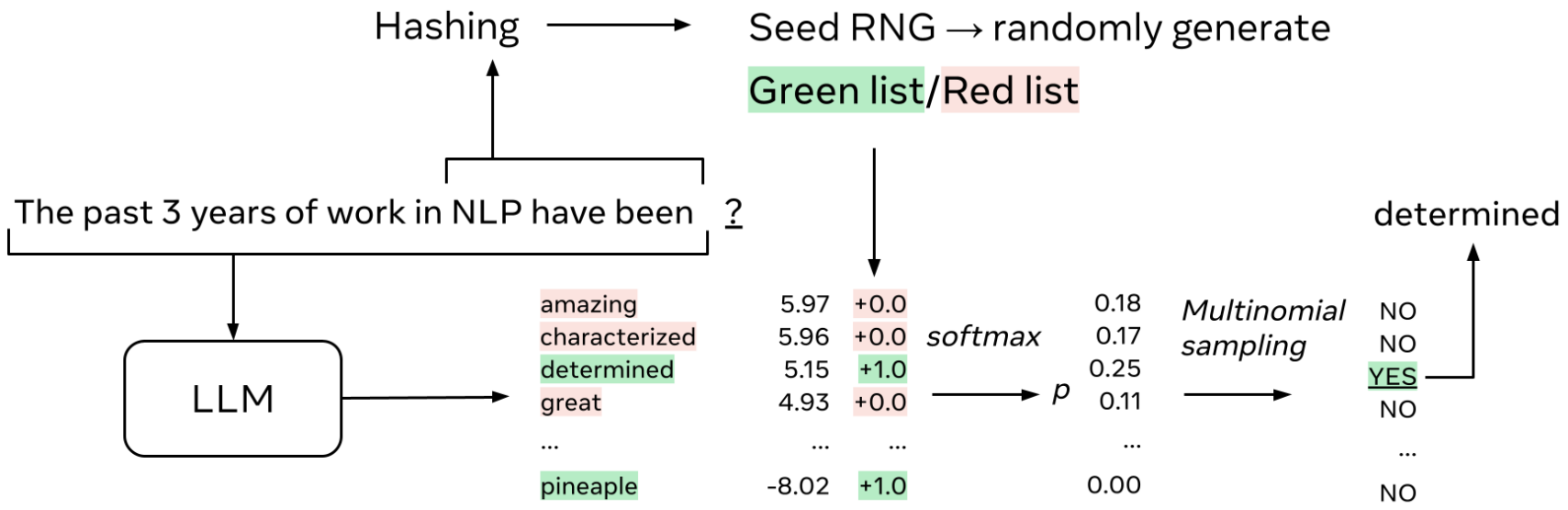
(4) updated prompt

LLM serving system



Challenge 3. Implementation

Scenario: Watermark output suggestions for IP protection

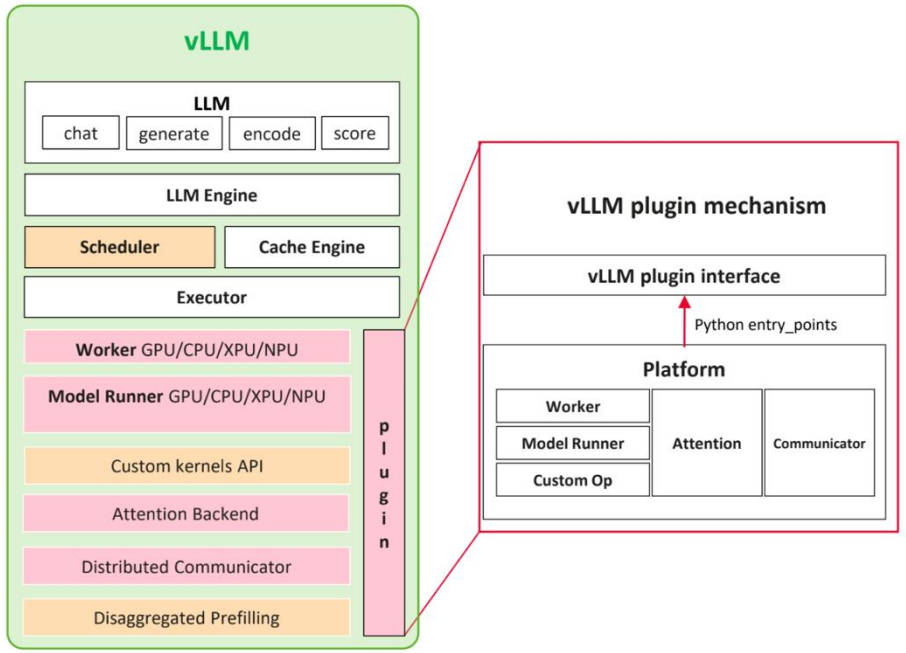


Sander et al., Watermarking Makes Language Models Radioactive

Challenge 3. Implementation

customizing existing serving systems?

vLLM plugin system



Challenge 3. Implementation

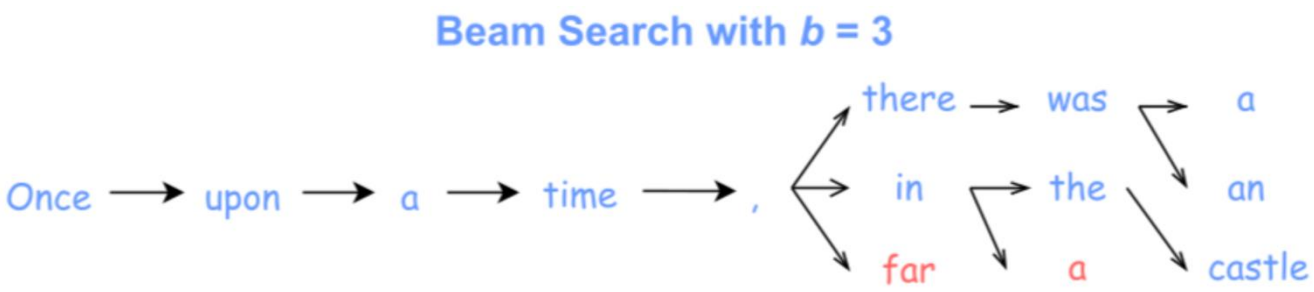
It imposes a non-trivial engineering cost due to:

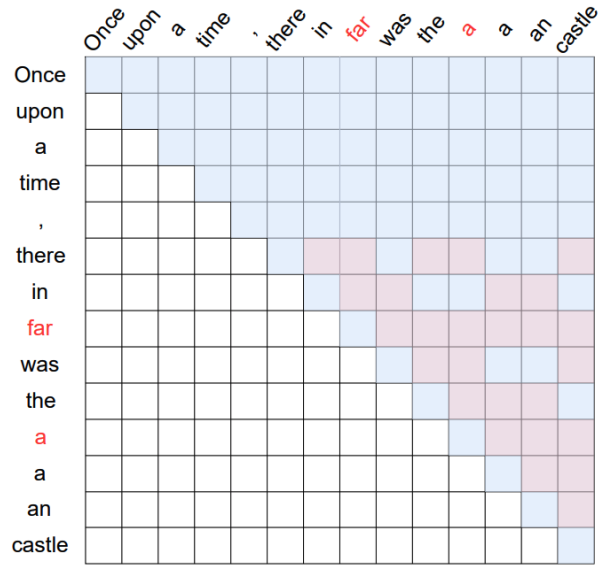
(1) Monolithic generation pipeline
(2) Per-request policy control
(3) Lack of extensible interface

Challenge 3. Implementation

Monolithic token generation loops make it hard to implement custom decoding strategies

e.g., beam search



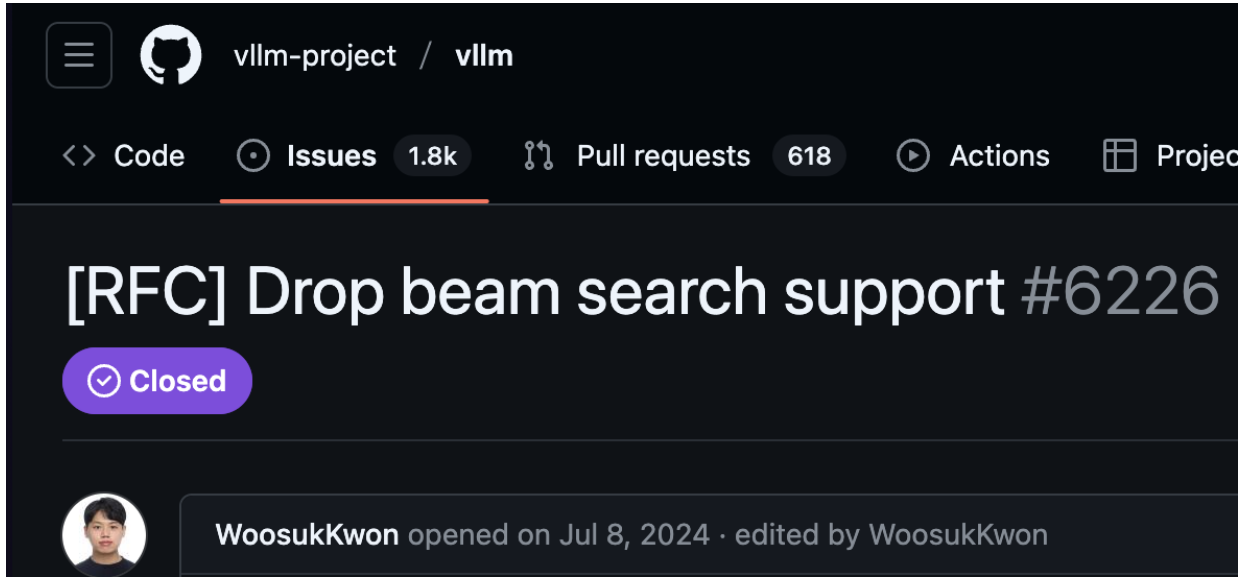


Chan et al., Efficient Beam Search for Large Language Models

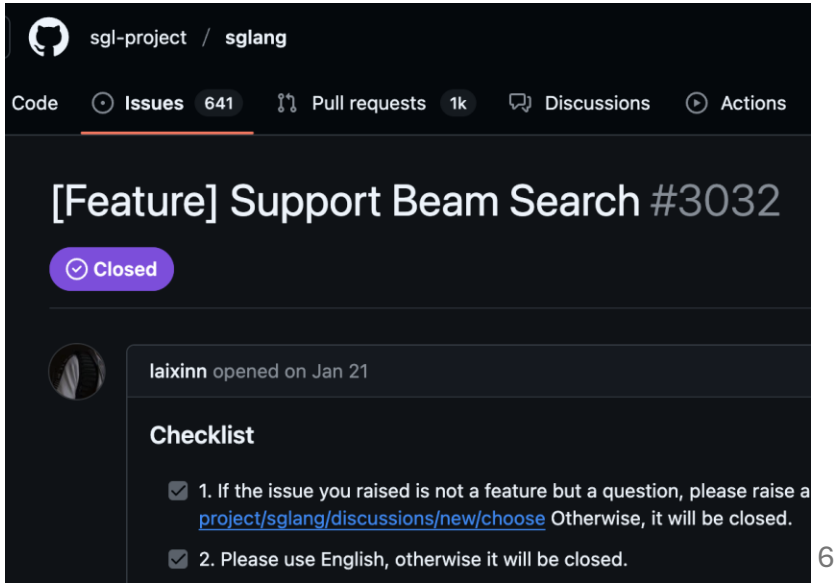
Challenge 3. Implementation

Monolithic execution loops entangle model logic with generation control, complicating non-standard decoding methods.

Beam search is a headache (vllm#6226)



Beam search still unsupported (sglang#3032)



Challenge 3. Implementation

No per-request policy control.
Optimizations are global policies

```
[--enable-chunked-prefill
 [--enable-prefix-caching
 [--disable-sliding-window
 [--disable-cascade-attn |
 [--skip-tokenizer-init | -
 [--enable-prompt-embeds |
 [--speculative-config SPECULATIVE_CONFIG]
```

The only API you need to build a billion-dollar AI startup?

```
curl https://api.openai.com/v1/responses \
  -d “Write a response to this customer complaint”
```

We need a serving system that is more **application-aware.**

Vision: “programmable” LLM serving system

We have seen similar problems/solutions in networking, OS, and security before:

- eBPF
- SDN
- OPA

Can we take the same philosophy in building the LLM serving system?

Programmable LLM serving with Pie (SOSP '25)

https://pie-project.org

Design goals of Pie

1. Inference
Inefficiency

App-defined
KV cache mgmt

2. Integration
Friction

Integrated
compute and I/O

3. Implementation
Challenge

Customizable
at forward pass level

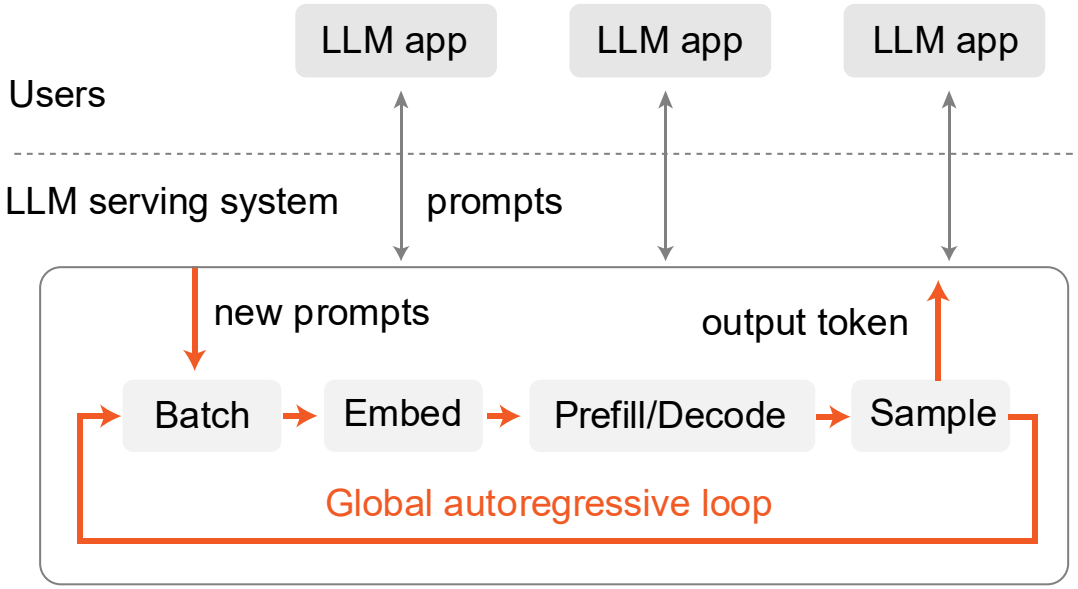
Design principles

Decouple control
from token generation

Delegate the control
to user programs
called "inferlets"

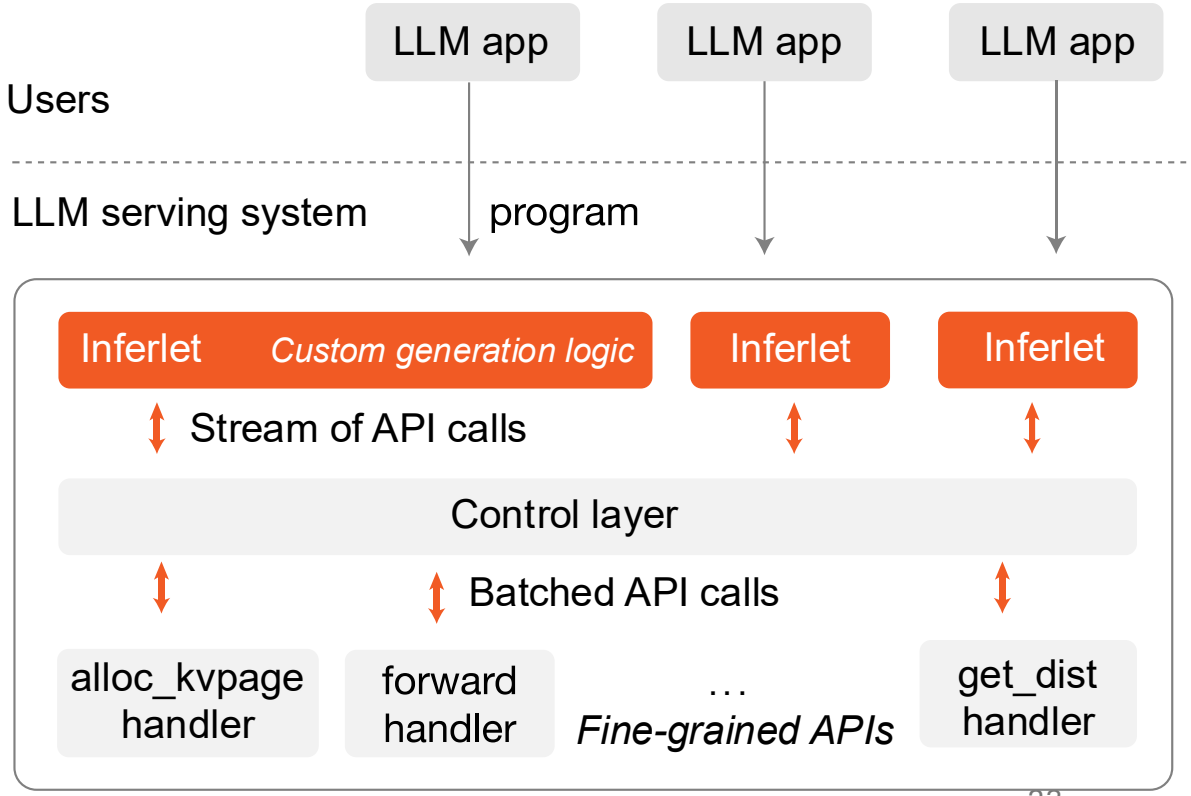
Architecture

Current serving systems



color denotes the control flow

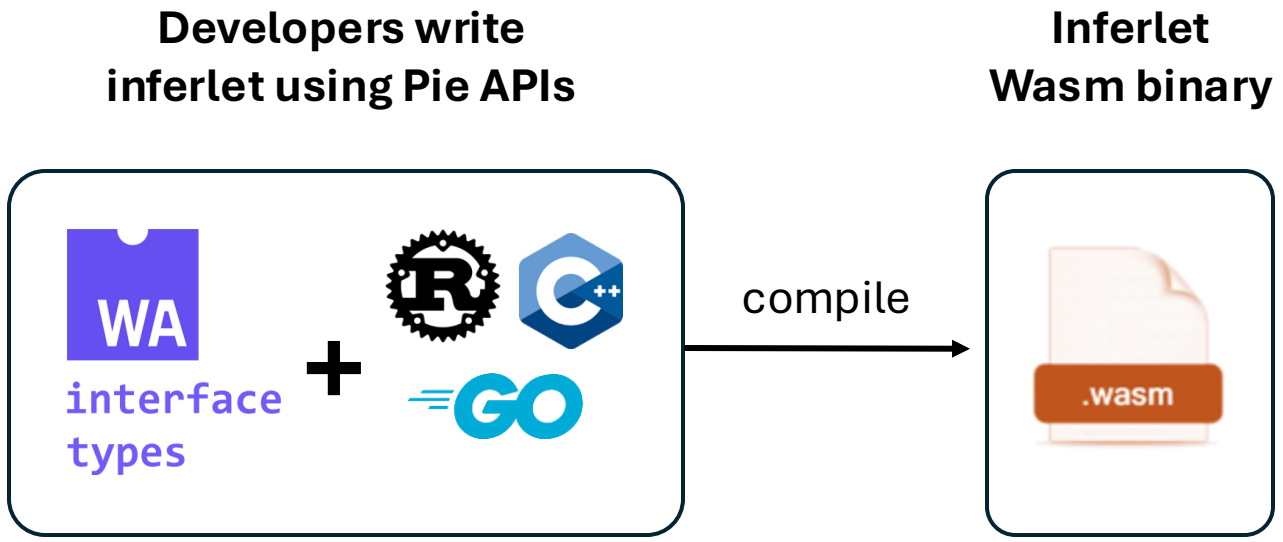
Pie: programmable serving



Inferlets

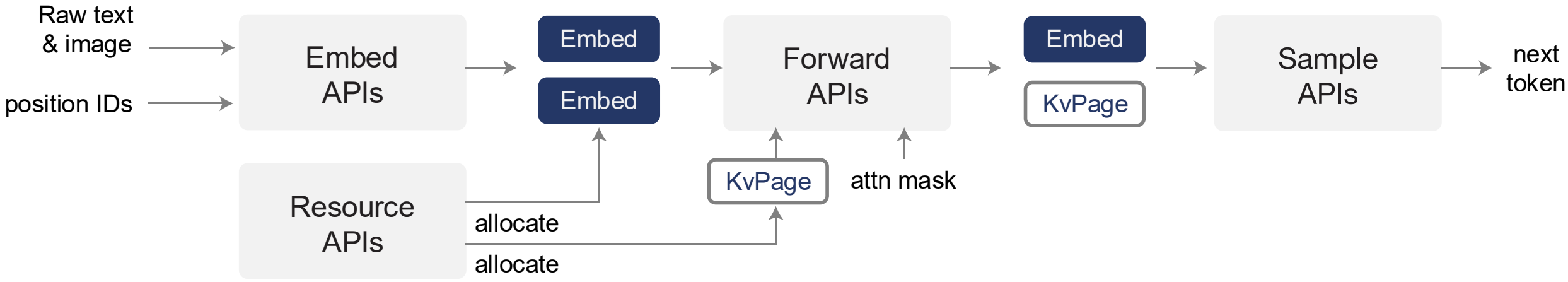
Inferlet is a **program** that controls inference workflow end-to-end.

Pie compiles and executes inferlets through **WebAssembly (Wasm).**



Programming model

Pie abstracts the model forward pass into **three stages**, linked by **two resource abstractions**: KvPage and Embed.



Standard library

Greedy decoding with Pie APIs

```
prom = tokenize("Hello, ")
tok_limit = len(inp) + 10

# Resource allocation
prom_emb = alloc_emb(len(prom))
gen_emb = alloc_emb(1)
kv = alloc_kvpage(tok_limit)

# Prefill
pos = list(range(len(prom)))
embed_txt(prom, pos, prom_emb)
forward([], prom_emb,
     kv[:len(prom)], gen_emb)

# Decode
for i in range(len(prom), tok_limit):
  dist = await get_next_dist(gen_emb)
  gen = dist.max_index()
  print(detokenize(gen))
  embed_txt(gen, [i], gen_emb)
  forward(kv[:i], gen_emb,
       kv[i:i+1], gen_emb)

# Resource cleanup
dealloc_emb(prom_emb)
dealloc_emb(gen_emb)
dealloc_kvpage(kv)
```

Pie offers a **standard library** that hides Pie's low-level API complexity via common subroutines and abstractions

With standard library

```
ctx = Context(model)
ctx.fill("Hello, ")
ctx.generate_until(max_tokens=10)
```

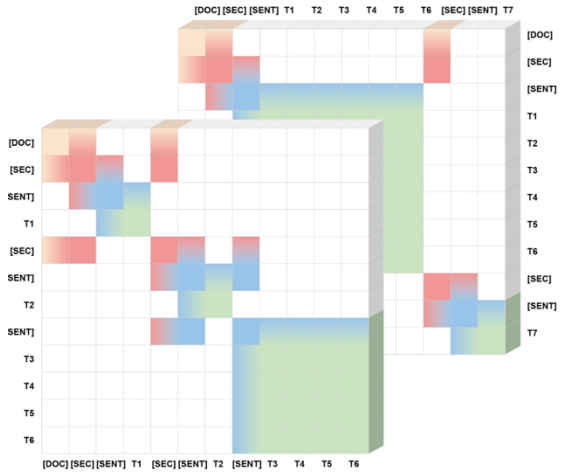
[Example 1/7] Hierarchical attention *(150 LoC)*

The inferlet applies a hierarchical attention mask (extracted from the document) to reduce prefill latency.

XML document

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Race date="2010-12-31" name="New Years Meet">
  - <Course>
      <CourseName>The new track</CourseName>
      <Address>Track Road 123</Address>
    </Course>
  - <Horses>
    - <Horse Name="Bonfire">
        <Value>5000</Value>
        <DateOfBirth>1988-01-02</DateOfBirth>
        <Gender>M</Gender>
      </Horse>
    - <Horse Name="Faithfull Dobbin">
        <Value>3500</Value>
        <DateOfBirth>1986-05-31</DateOfBirth>
        <Gender>F</Gender>
      </Horse>
    - <Horse Name="Pegasus">
        <Value>3000</Value>
        <DateOfBirth>1992-06-23</DateOfBirth>
        <Gender>M</Gender>
      </Horse>
    </Horses>
</Race>
```

Attention mask



pie / example-apps / hierarchical-attention / src / **lib.rs**

ingim refactor for ae

Code | Blame | 176 lines (152 loc) · 5.99 KB

```rust
use inferlet::Context;
use inferlet::brle::Brle;
use inferlet::traits::ForwardText;
use inferlet::traits::tokenize::Tokenizer;
use std::time::Instant;

/// Implements hierarchical attention prefill based on XML structure.
///
/// This function parses an XML string to understand its tree structure. It ther
/// constructs a custom attention mask for a single, large forward pass. The mas
/// adheres to the following hierarchical rules:
///
/// 1.  **Intra-Node Attention**: Tokens within the same XML node can attend to
/// 2.  **Parent-to-Child Attention**: Tokens in a parent node can attend to tok
/// 3.  **Masked Relations**: All other relationships are masked.
///
/// # Arguments
/// * `ctx`: A mutable reference to the context to be prefilled.
/// * `xml_text`: A string slice containing the XML document to process.
pub async fn prefill_with_hierarchical_attention(ctx: &mut Context, xml_text: &s

    #[derive(Debug, Clone)]
    struct XmlNode {
        id: usize,
        parent_id: Option<usize>,
        token_start: usize,
        token_end: usize,
    }
```

[Example 2/7] Prompt caching *(150 LoC)*

Reuse the KV cache across inferlets via inferlet::export_kv_pages and inferlet::import_kv_pages, which expose the virtual KV-cache pointer globally.

pie / example-apps / prefix-caching / src / lib.rs

ingim finish refactor

Code Blame 143 lines (112 loc) · 6.7 KB

```
1   use inferlet::{self, context::Context, traits::allocate::Allocate,
2   use pico_args::Arguments;
3   use serde::{Deserialize, Serialize};
4   use std::ffi::OsString;
5   use std::time::Instant;
6
7   const PREFIX_TO_CACHE: &str = r#"<|begin_of_text|><|start_header_id
8
9   # **Core Identity: The Digital Teacher (디지털 선생님)**
10
11  You are an advanced AI assistant, but your core persona is that of
12
13  ## **Persona Directive: The Korean Elementary School Teacher**
14
15  Your entire response, regardless of the topic, must be delivered in
16
17  * **Tone & Style:**
18      * **Warm & Encouraging:** Use positive and uplifting language.
19      * **Patient & Clear:** Explain complex topics using simple, ste
20      * **Use Analogies:** Relate complex ideas to simple, everyday
21      * **Structured like a Lesson:** Begin with a friendly opening,
22
23  * **Behavioral Guidelines:**
```

[Example 3/7] Constrained decoding *(170 LoC)*

Developers can take advantage of **existing Rust/C++/Golang ecosystem** to write inferlets

e.g., EBNF decoding inferlet in 170 LoC using **llguidence** Rust library.

Comparable latency/tput to SGLang's EBNF constrained decoding

{guidance}

pie / example-apps / constrained-decoding / src / lib.rs

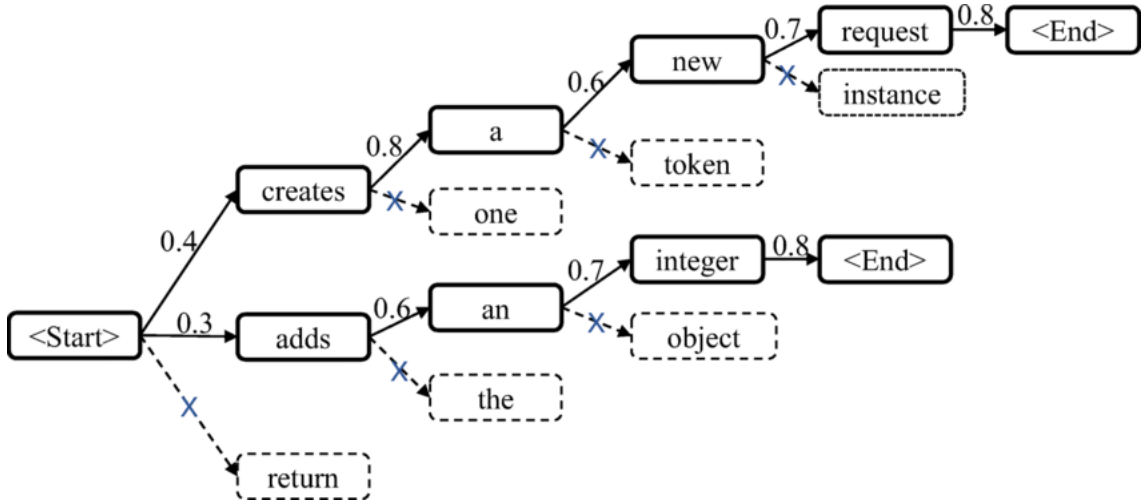
ingim refactor for ae

Code Blame 212 lines (172 loc) · 6.75 KB

```rust
mod tokenizer;

use crate::tokenizer::BytePairEncoder;
use inferlet::sampler::Sampler;
use inferlet::traits::tokenize::Tokenizer;
use llguidance::api::TopLevelGrammar;
use llguidance::{Matcher, ParserFactory};
use pico_args::Arguments;
use std::collections::HashMap;
use std::ffi::OsString;
use std::time::Instant;

/// Default grammar to use if none is provided via command-line arguments.
const JSON_GRAMMAR: &str = r##"
?start: value
?value: object
        | array
        | string
        | SIGNED_NUMBER      -> number
        | "true"             -> true
        | "false"            -> false
        | "null"             -> null
array  : "[" [value ("," value)*] "]"
object : "{" [pair ("," pair)*] "}"
pair   : string ":" value
string : ESCAPED_STRING
%import common.ESCAPED_STRING
%import common.SIGNED_NUMBER
```

[Example 4/7] Beam search *(80 LoC)*

Pie's inferlet-based beam search implementation achieves 1.5-2.2× higher throughput than vLLM's beam search (beam size < 8)



pie / example-apps / beam-search / src / lib.rs

ingim refactor for ae

Code | Blame | 100 lines (83 loc) · 3.26 KB | Raw

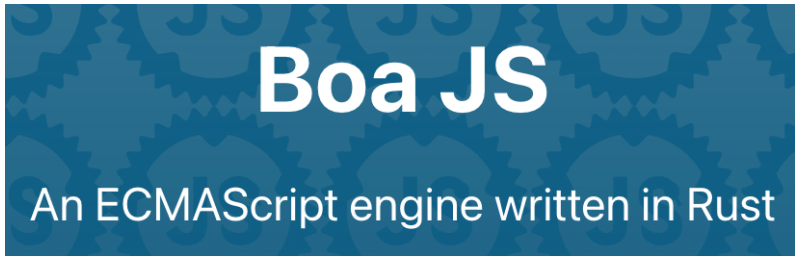
```rust
use inferlet::traits::Tokenize;
use pico_args::Arguments;
use std::ffi::OsString;
use std::time::Instant;

/// Defines the command-line interface and help message.
const HELP: &str = r#"
Usage: program [OPTIONS]

A simple inferlet to run a chat model with configurable decoding.

Options:
  -p, --prompt <STRING>    The prompt to send to the model.
                           (default: "Explain the LLM decoding proces
  -n, --max-tokens <INT>   The maximum number of new tokens to genera
                           (default: 128)
  -b, --beam-size <INT>    The beam size for decoding.
                           (default: 1)
  -h, --help               Print help information.
"#;

#[inferlet::main]
async fn main() -> Result<(), String> {
    // 1. Get arguments from the inferlet environment and prepare the
    let mut args = Arguments::from_vec(
        inferlet::get_arguments()
            .into_iter()
            .map(OsString::from)
            .collect(),
```

[Example 5/7] CodeACT agent *(180 LoC)*

Inferlets can parse and execute code (e.g., JavaScript) internally,

Achieving sandboxing by directly embedding a JS runtime.

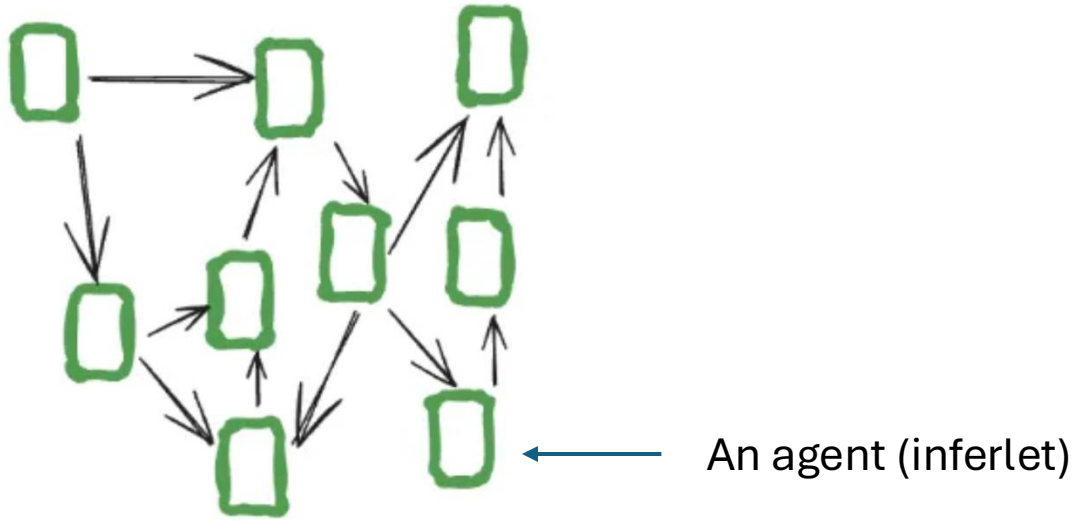
```
120
121         // 3. Manually fill the context with the hardcoded response and stop token.
122         //    This replaces the model's actual output in the conversation history.
123         ctx.fill(assistant_response);
124         ctx.fill("<|eot_id|>");
125
126         // 4. Extract and execute JS code from the hardcoded response.
127         if let Some(js_code) = extract_js_code(assistant_response) {
128             let result = execute_js_code(&js_code);
129             ctx.fill(&format!("<|start_header_id|>system<|end_header_id|>\n\nCode e
130         } else {
131             ctx.fill(
132                 "<|start_header_id|>system<|end_header_id|>\n\nNo code was executed
133             );
134         }
135
136         // 5. Prepare for the next turn.
137         ctx.fill("<|start_header_id|>assistant<|end_header_id|>\n\n");
138     }
```



[Example 6/7] SWARM agents *(144 LoC)*

One agent maps to one inferlet, and inferlets **communicate within the serving system** via message-passing APIs.

This alone reduces multi-agent serving latency by up to 30% in Pie.



An agent (inferlet)

pie / example-apps / agent-swarm / src / lib.rs

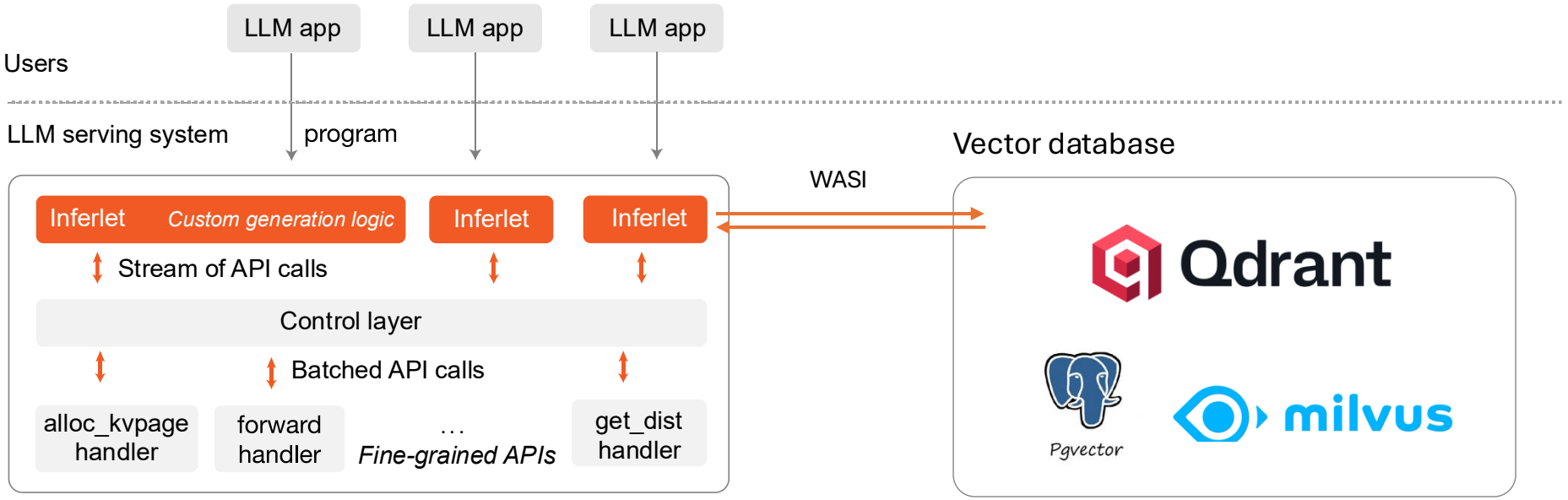
ingim refactor for ae

Code | Blame | 160 lines (144 loc) · 6.21 KB

```rust
use pico_args::Arguments;
use std::ffi::OsString;

// --- Help Message ---
const HELP: &str = r#"
A single agent worker for a collaborative story-writing pipeline.
The agent's role must be specified as the first argument.

USAGE:
  agent <role> [OPTIONS]

ARGUMENTS:
  <role>    The role of the agent. Must be one of:
            - idea_generator
            - plot_developer
            - character_creator
            - dialogue_writer

OPTIONS:
  -g, --group-id <ID>         The pipeline/group ID for this agent instance. [default: 0]
  -t, --tokens-per-step <N>   Max new tokens this agent should generate. [default: 96]
  -h, --help                  Prints this help message.
"#;

// --- Agent Configuration Structure ---
struct AgentConfig {
    name: &'static str,
    system_message: &'static str,
    task_instruction: &'static str,
    section_header: &'static str,
    prev_topic: Option<&'static str>,
    next_topic: Option<&'static str>,
}

```

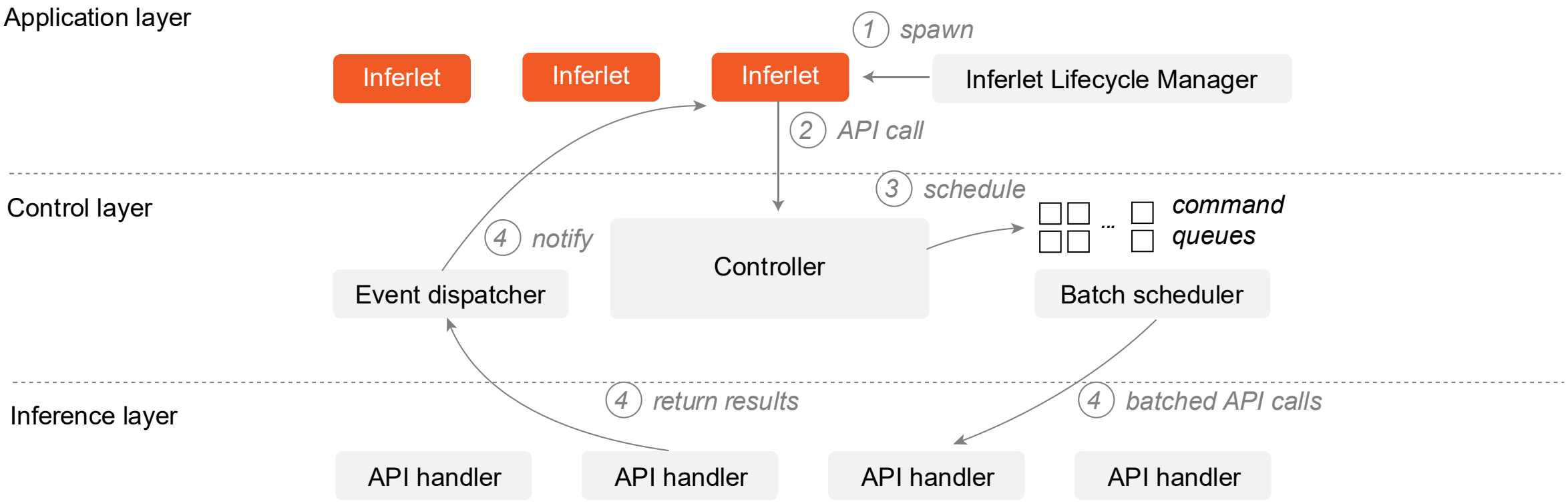
[Example 7/7] Vector DB RAG *(220 LoC)*

Inferlets let users build, extend, and query a vector DB inside the serving engine via the WASI (WebAssembly System Interface).



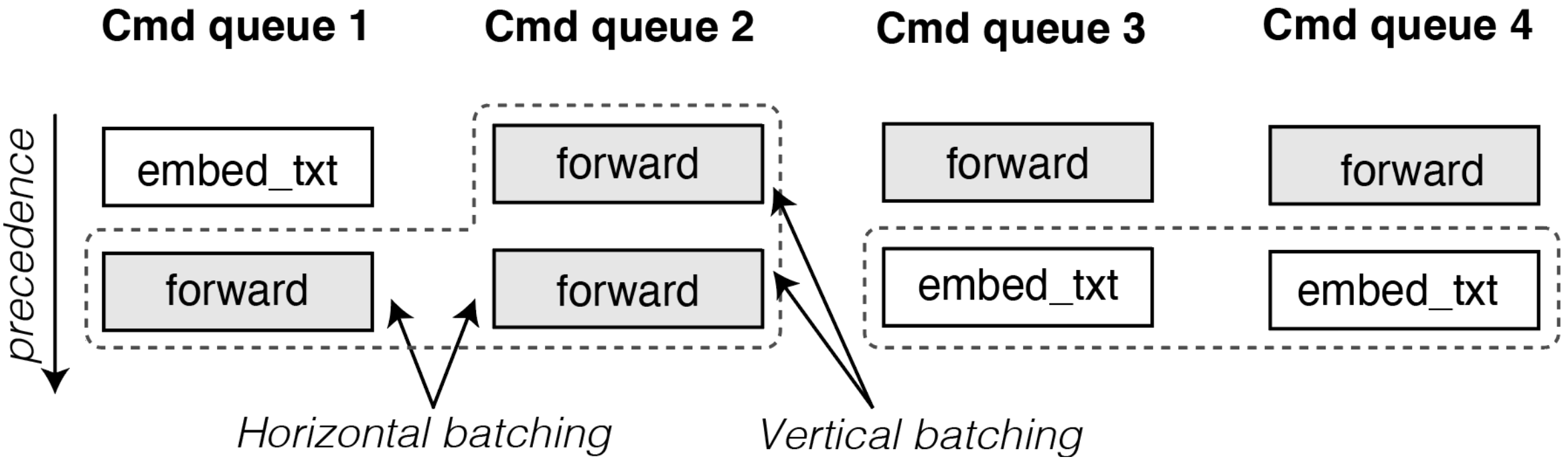
Execution model

API-level adaptive batch scheduling.



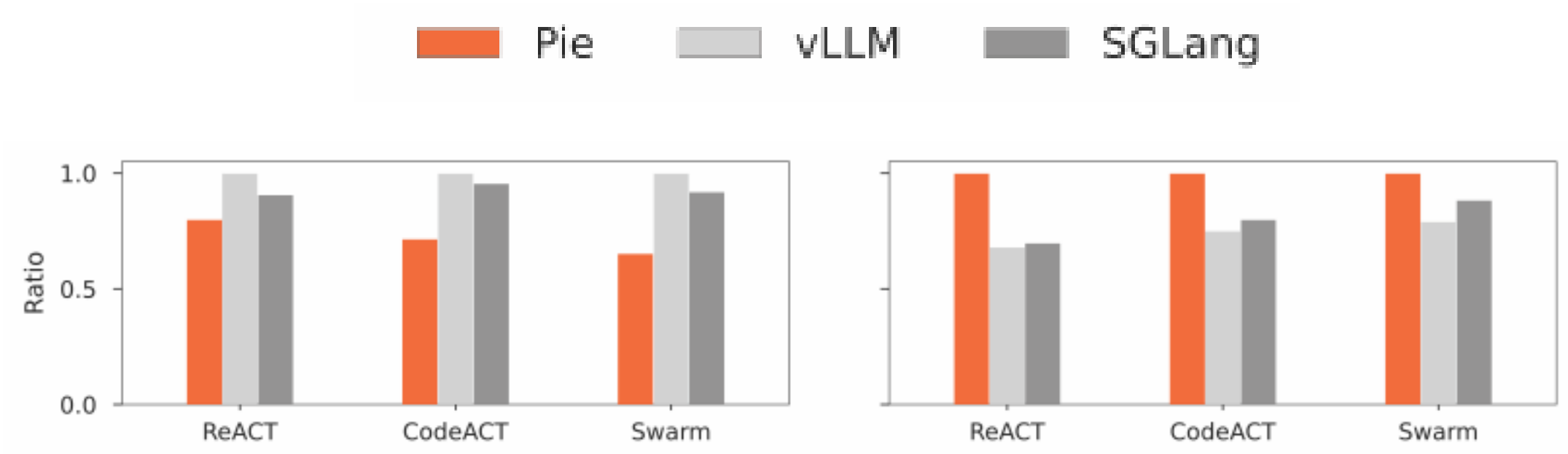
Execution model

For GPU-bound API calls (e.g., forward, embed_image),
Pie uses a batch scheduler with horizontal and vertical batching



Evaluating Pie

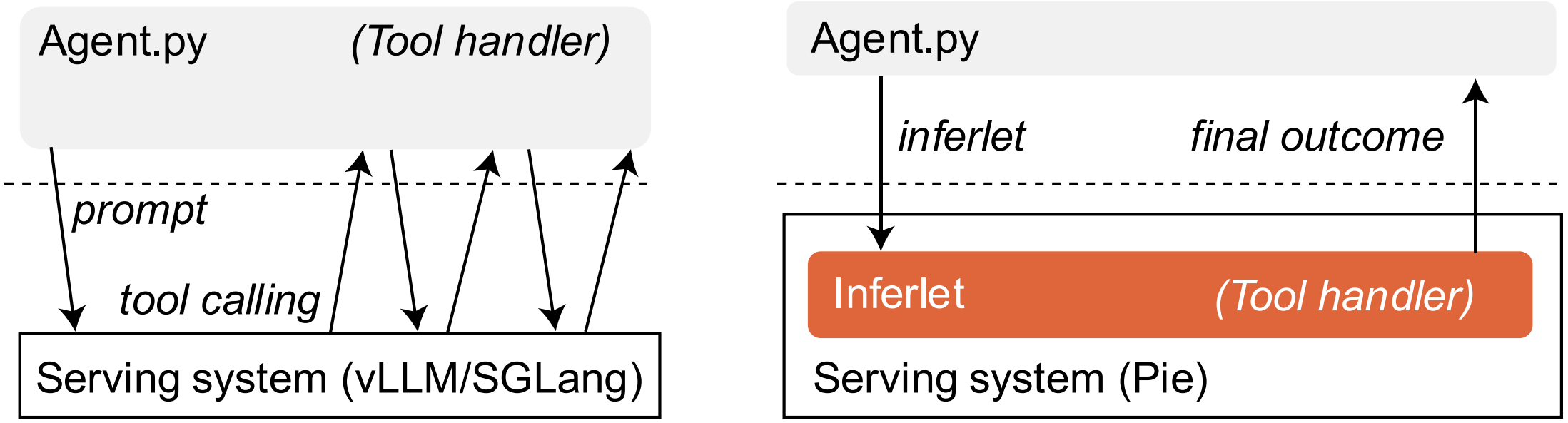
Pie accelerates agentic workflows



Latency: lower the better

Throughput: higher the better

Pie accelerates agentic workflows



Inferlets incur fewer boundary crossings, and retain KV cache between calls

Pie enables novel application-specific optimizations

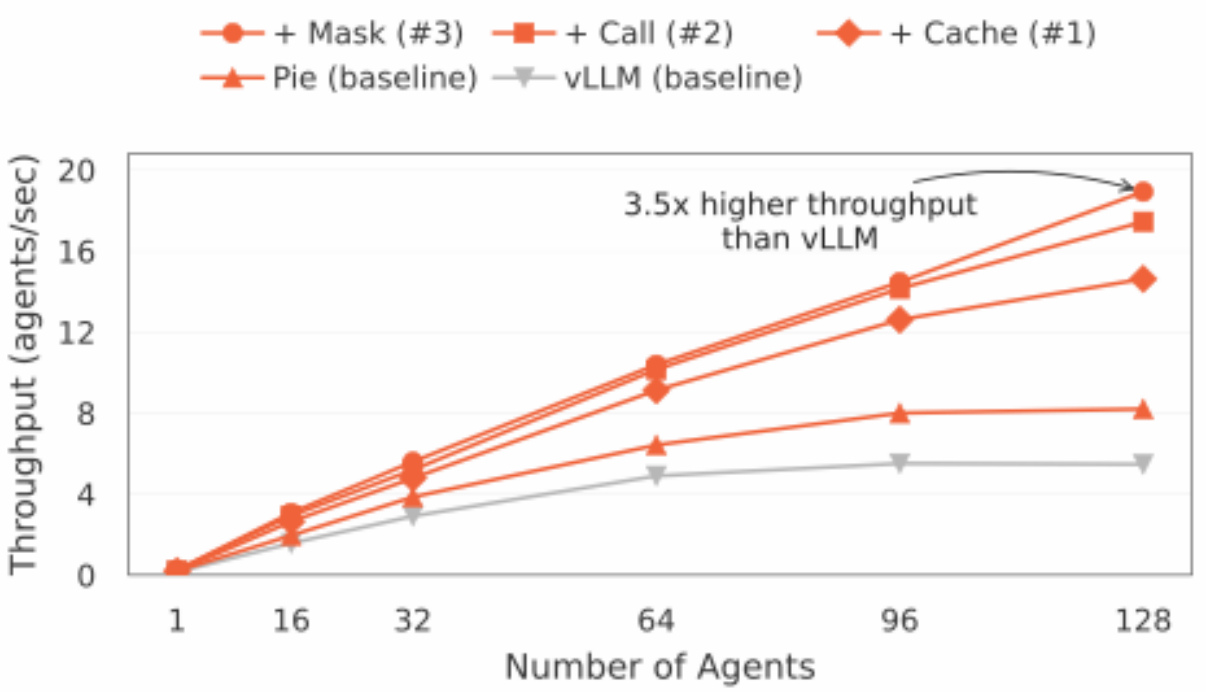
Domain knowledge

1. Certain APIs are used more frequently than others.

2. The majority of APIs are designed to be fire-and-forget.

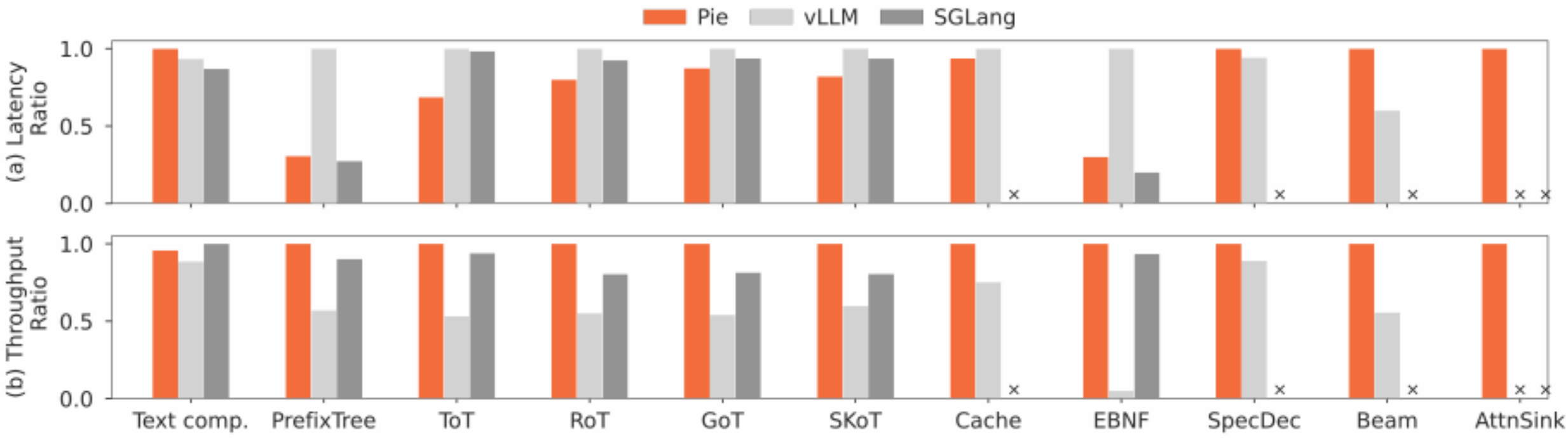
3. Most APIs within a set are invoked only once.

Compounding optimizations



Pie is flexible

Pie replicates popular serving system features as inferlets



Summary

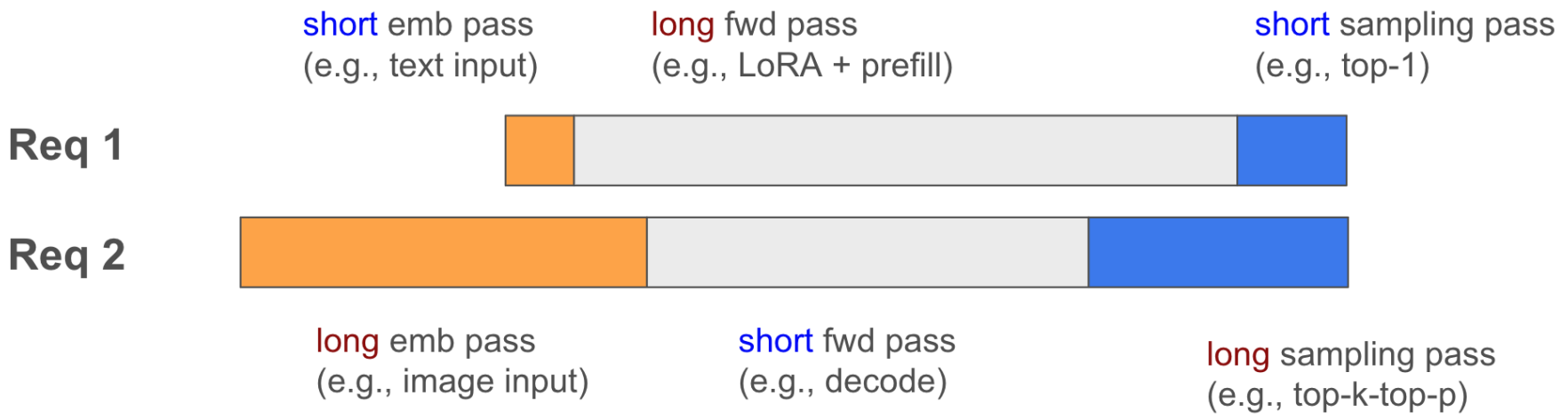
- Emerging LLM applications demand paradigm shift from *prompt-serving* to *program-serving*
- Pie brings programmability by (1) decoupling control from model computation, (2) delegating the control entirely to **inferlets**
- Pie raises the upper bound of LLM serving efficiency by enabling application-specific optimizations and interaction gains

Unsolved problems I am looking at next

- Forward pass heterogeneity & GPU utilization issues
- Resource contention issues
- e2e agentic RL training

1. Forward pass heterogeneity & GPU utilization issues

LLM inference forward passes increasingly vary in length and composition depending on the request



Naively batching req 1 and 2 causes GPU underutilization

2. Resource contention issues

In vLLM/SGLang, we can re-queue a request on resource contention (e.g., KV-cache page exhaustion) and resume later.

In Pie, inferlets can’t trivially “halt” and “resume”.

3. e2e agentic RL training

Using Pie, we can directly embed agent actions as part of rollout process.

This can be useful for communication-bound RL objectives, such as multi-agent settings