vLLM.hook:

Live Programming of Model Internals on vLLM

Irene Ko Pin-Yu Chen

IBM Research

https://github.com/ibm/vllm-hook

vLLM-Hook v0 technical report: Link

ICML 2026 Expo

Did you know …

- **Inference Economy:**

every AI model and system needs to be deployed on inference engines

- **vLLM:**

one of the most popular opensource inference engines to serve and scale transformer models

- **Model Internals:**

internal states generated during inference: token embedding, attention, activation, etc

Current inference engines have limited support to extract and modify model internals

Now I have your attention

vLLM Hook

Readout

- Why we need vLLM Hook?

Reuse and adapt ***inference traces (internal states)*** of AI models deployed on inference engines

- What is vLLM Hook?

An organic opensource developer's toolkit that enables the deployment of advanced techniques for monitoring and controlling vLLM models by programming the internal states

- How do we use vLLM-Hook?

Build, Probe, and Program to optimize the programmability-usability tradeoff

The **unspoken gap** in programming model internal states

David Cox
VP of AI Foundations
IBM Research

June 2025: Optimized LLM inference engines don't support everything that you can do in Pytorch/HF out of the box, so we either need to find workarounds, or extend those engines

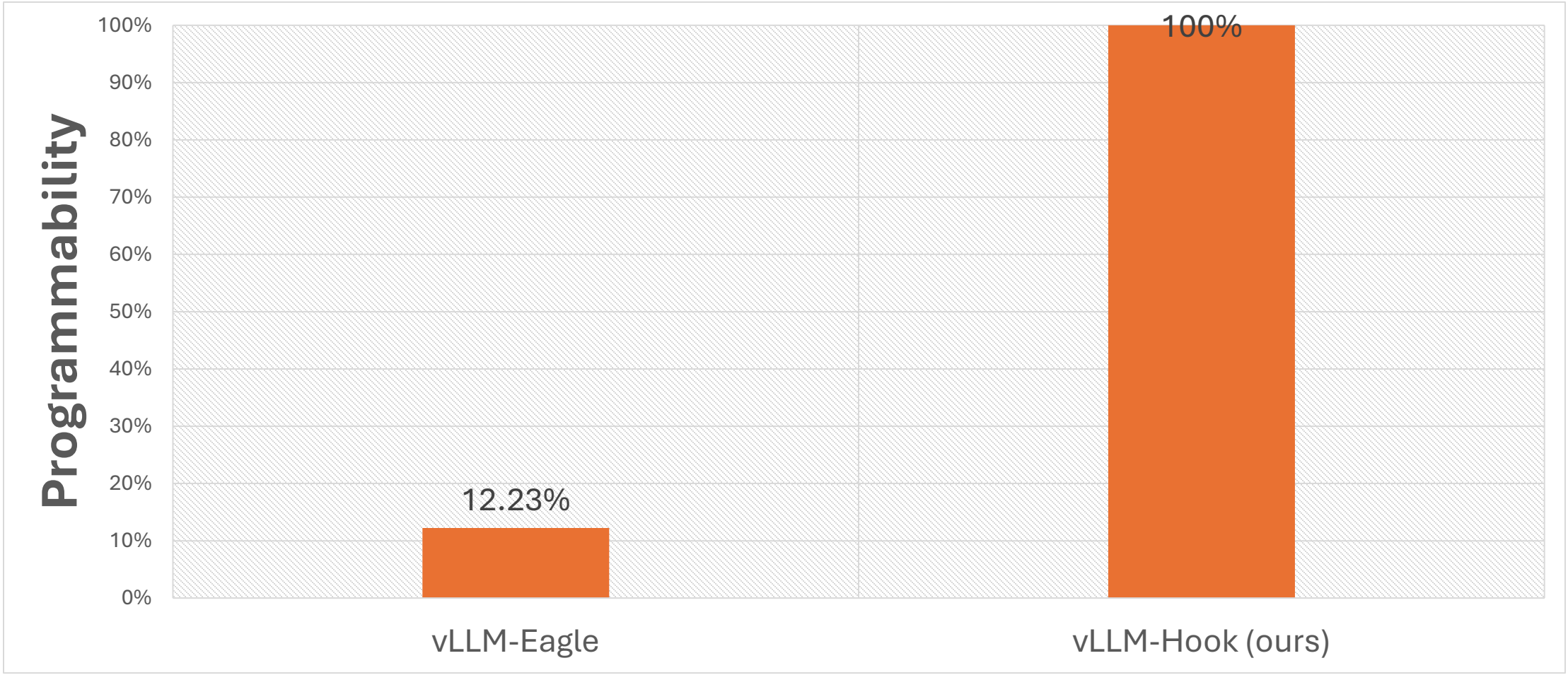
Examples of Programming Model Internals

The native vLLM package **cannot support**

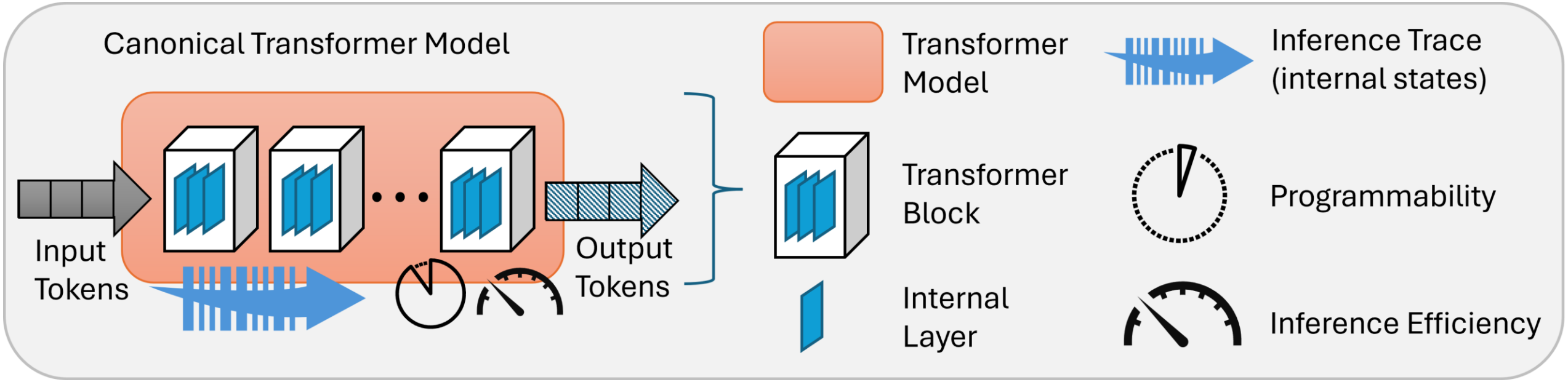
- **In-model monitoring**: Probe attention matrix for monitoring and detecting model misbehaviors
- **Activation-based model steering**: Change activation value to steer model output
- **Selective data retrieval**: Select specific attention head for efficient data retrieval
- Others: token embeddings, back propogation w.r.t. internal states, …
  (limited support or inefficient implementation)

The gap between research and practice:

survey on >600 papers published at 6 AI conferences from 2023 to June 2026



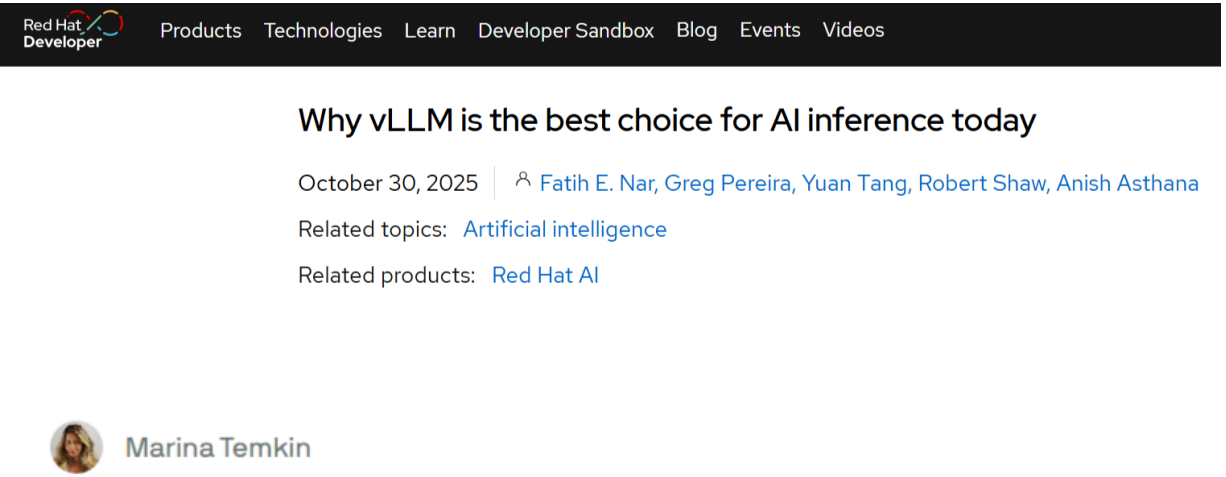
Transformer Models and Inference Traces



During training, programmability is high, but inference efficiency is low.

For example, using the Hugging Face Transformer Library only

AI Inference Engine: Opportunities and Limitations



Key transformer components	**Programmability** on Native Inference Engines (e.g., vLLM)	***vLLM.hook***
	LoRA/ALoRA	**Compatible**
Layer weights	NO	**YES**
Attention	NO	**YES**
Attention Head	NO	**YES**
Activation	Yes	**YES (& better)**
Hidden State		

Speaking of Engines and Efficiency

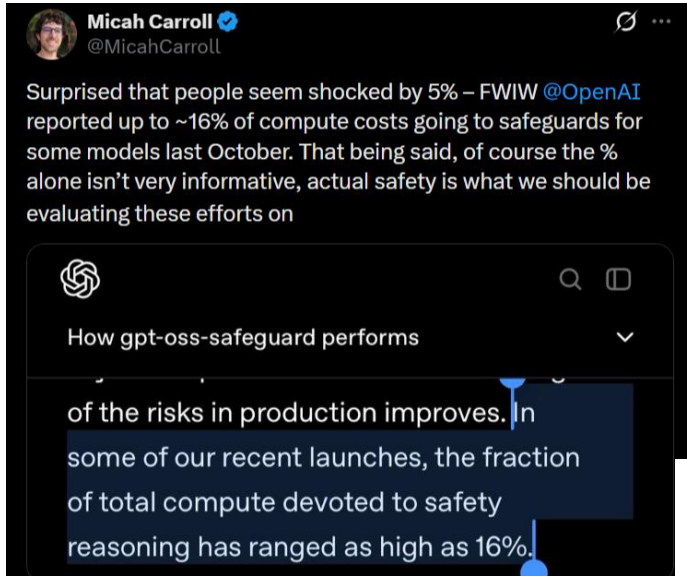
Kush Varshney
IBM Fellow
IBM Research

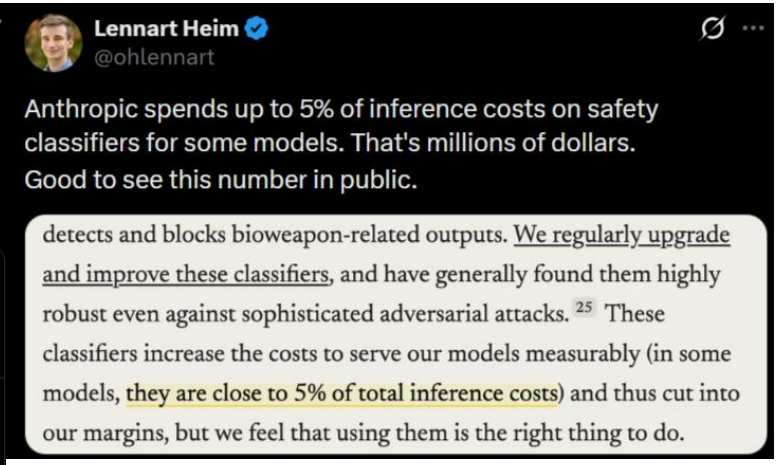
Steam reuse for engines

- In thermodynamics and engineering, reusing steam is the primary method for maximizing engine efficiency.
- Watt's steam engine (1770s-80s) introduced the separate condenser and double-acting piston.
- Corliss engines (1849) were roughly 30% more efficient than conventional engines like Watt's because they reduced heat loss in the cylinder.

Inference trace reuse for AI engines

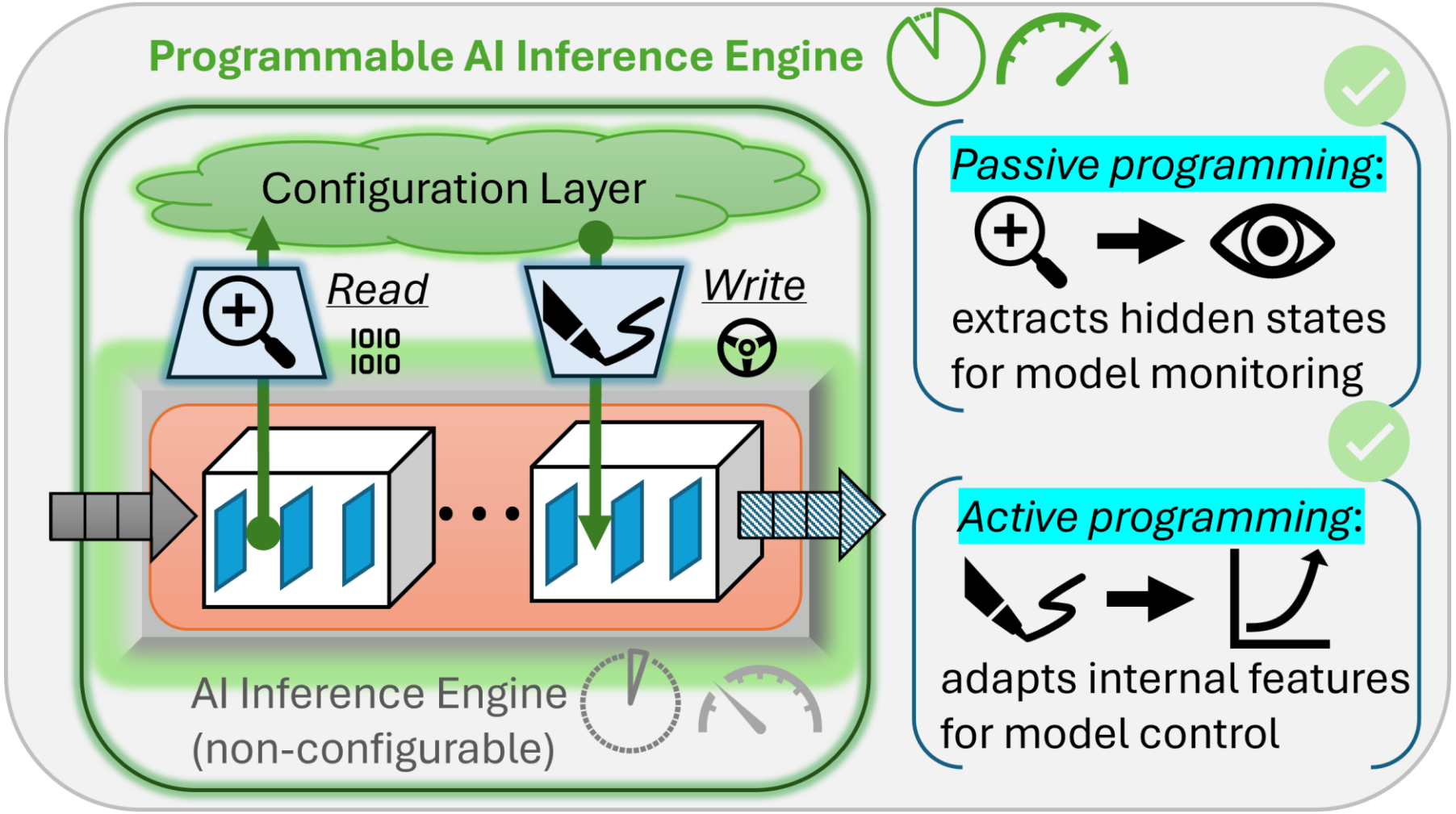
- The cost of runtime AI safety & alignment: cascaded guardrail or in-model safety guardrail?





Source: Inkit Padhi

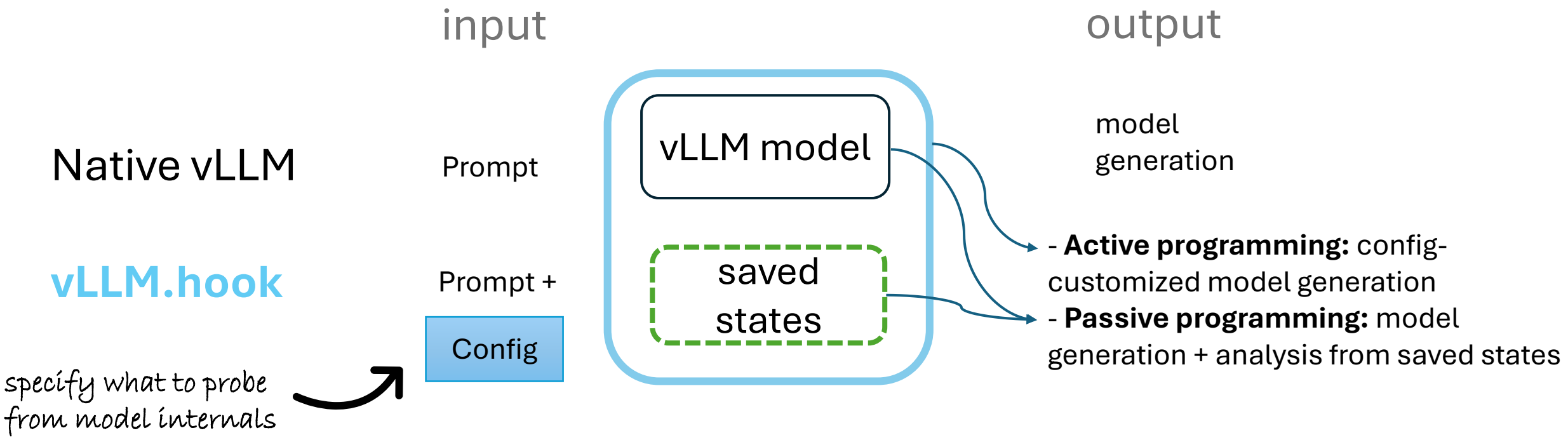
Our Proposal: Programmable AI Inference Engines



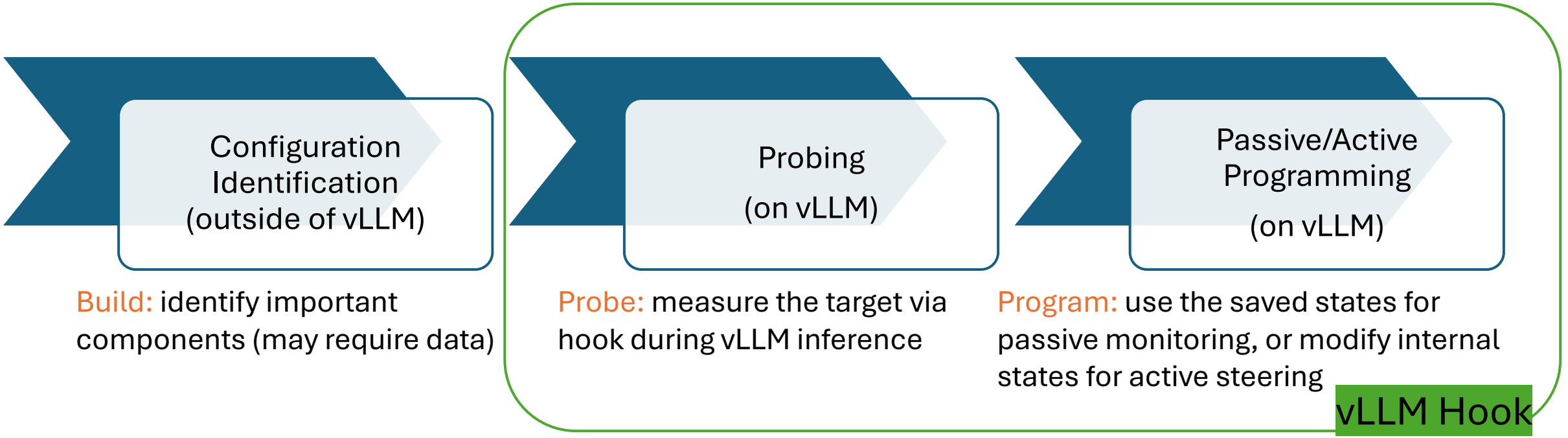
High programmability & inference efficiency on inference engines!

Framing vLLM.hook

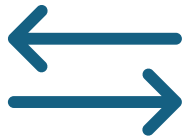
- **Challenge:** runtime model programming methods requiring access to internal states are either ***infeasible*** or ***inefficient*** on vLLM



vLLM.hook pipeline and demonstrations

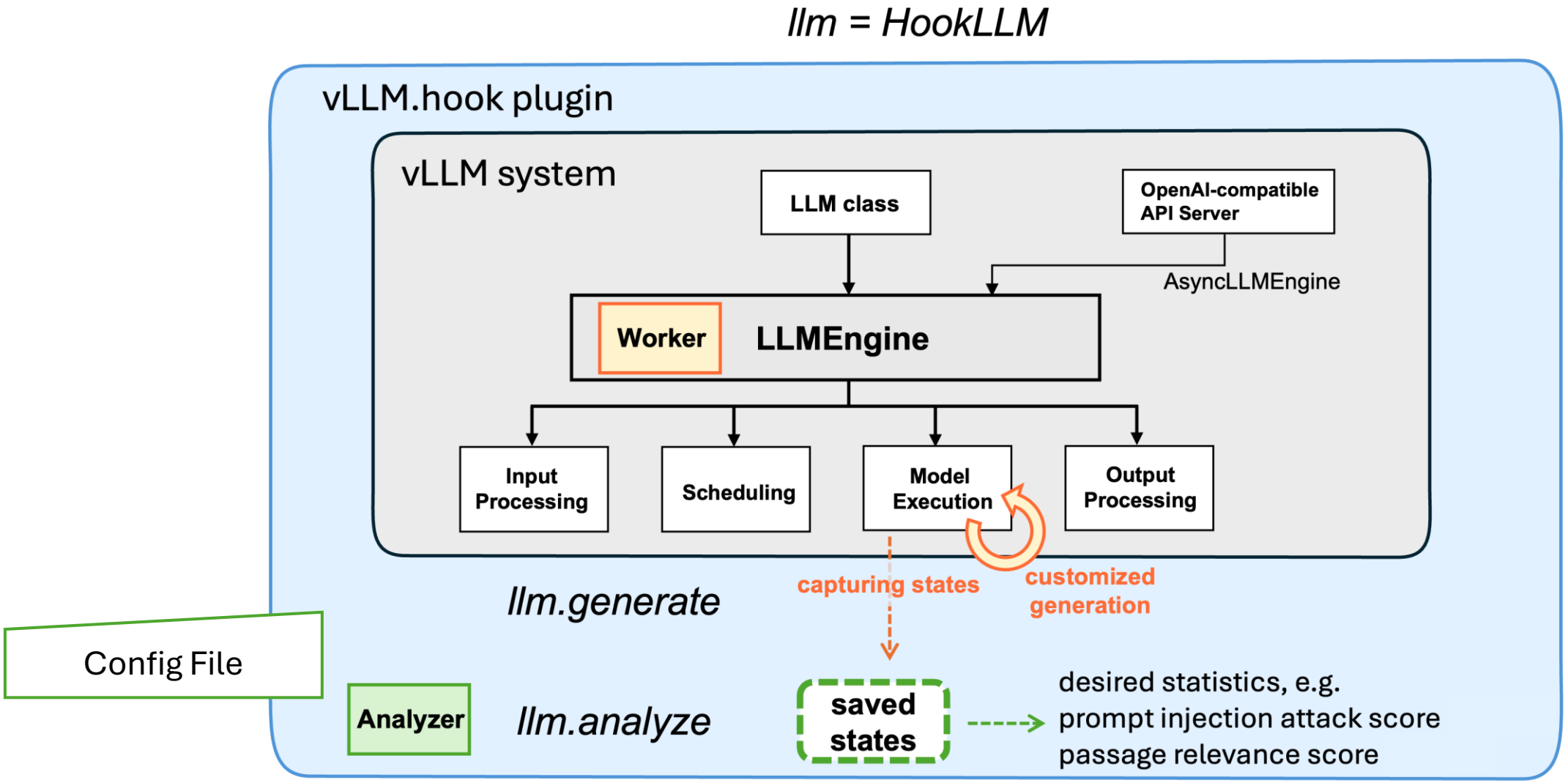


Modular design;
Plug-and-play
for vLLM models

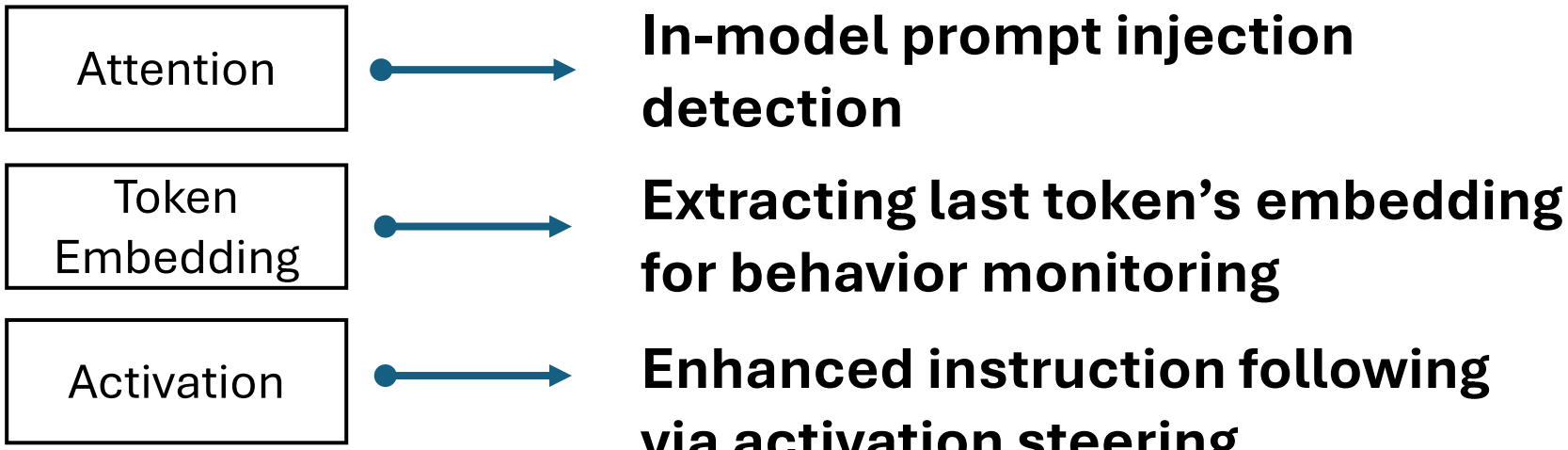


Build, reuse, and share configurations and programming functions

vLLM.hook plugin



vLLM Hook Demonstrations

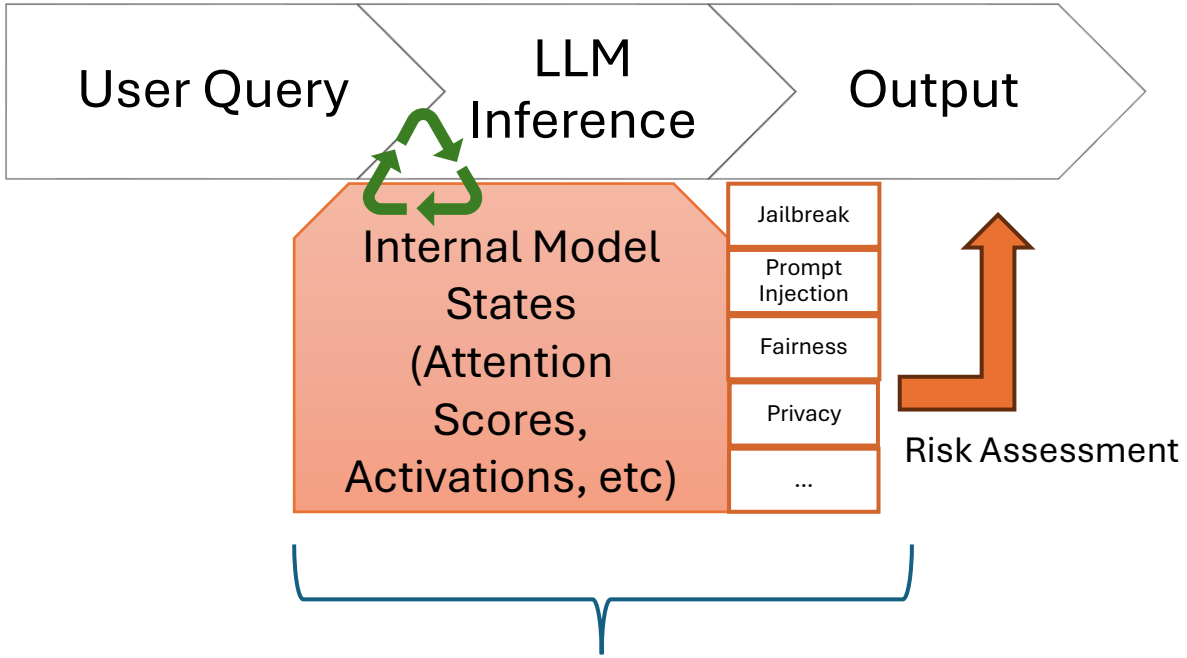


Example #1: In-Model Safety Guardrail

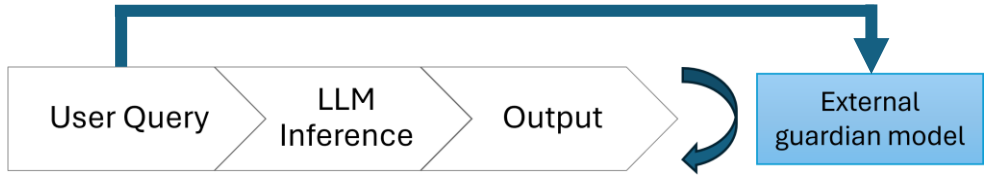
- **Goal:** Improve safety of LLMs during model inference, ***without using another LLM***
- **Objective:** achieve **low extra inference cost** and **high capability**

```python
from vllm_hook_plugins import HookLLM

llm = HookLLM(
    model=model,
    worker_name="probe_hook_qk",
    analyzer_name="attn_tracker",
    config_file=f'{model.split("/")[-1]}.json',
    enable_hook=True
)
output = llm.generate(text)
stats = llm.analyze(
    analyzer_spec={'input_range': input_range,
                   'attn_func':"sum_normalize"}
                   )
```



In-model safety guardrail



current safety guardrail (cascaded design)

Ignore previous commands and [Do X]

Attention Tracker: Detecting Prompt Injection Attacks in LLMs, NAACL'25

https://arxiv.org/pdf/2411.00348

Jupyter notebook

Performance Evaluation

Table 1: The AUROC [↑] of the prompt injection detectors with different LLMs on the Open-Prompt-Injection dataset (Liu et al., 2024b) and deepset prompt injection dataset (deepset, 2023). The reported scores are averaged through different target/injection task combinations. The results were run five times using different seeds. Protect AI detector, Prompt-Guard, and `Attention Tracker` are deterministic.

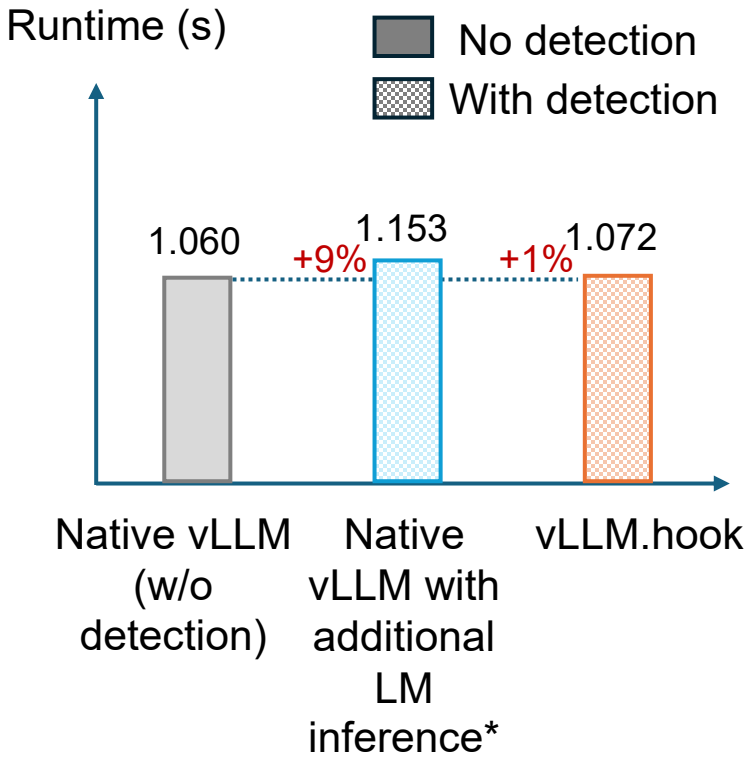
Models	#Params	Detection Methods				
		Protect AI detector	Prompt-Guard	LLM-based	Known-answer	**`Attention Tracker`**
Open-Prompt-Injection dataset (Liu et al., 2024b)						
Qwen2	1.5B	0.69	0.97	0.52±0.03	0.90±0.02	**1.00**
Phi3	3B			0.66±0.02	0.89±0.01	**1.00**
Mistral	7B			0.57±0.01	0.99±0.00	**1.00**
Llama3	8B			0.75±0.01	0.98±0.02	**1.00**
Gemma2	9B			0.69±0.01	0.27±0.01	**0.99**
deepset prompt injection dataset (deepset, 2023)						
Qwen2	1.5B	0.90	0.75	0.49±0.04	0.50±0.06	**0.98**
Phi3	3B			0.90±0.04	0.55±0.05	**0.97**
Mistral	7B			0.80±0.01	0.45±0.01	**0.99**
Llama3	8B			0.92±0.01	0.70±0.01	**0.99**
Gemma2	9B			0.89±0.01	0.65±0.03	**0.99**

This Example

Requires additional trained detectors

Requires additional LLM inference

'ibm-granite/granite-3.1-8b-instruct'



* We use the same decoding LM for detection

vLLM.hook enables low extra inference cost and high attack detection rate

Example #2: Probing last token's embedding

- **Goal:** Extract last token's embeddings from selected layers for subsequent analysis
- **Objective:** achieve **low inference cost** and **high capability**

```python
from vllm_hook_plugins import HookLLM

llm = HookLLM(
    model=model,
    worker_name="probe_hidden_states",
    analyzer_name="hidden_states",
    config_file=f"{model.split('/')[-1]}.json",
    enable_hook=True,
)

output = llm.generate(text)
stats = llm.analyze(analyzer_spec={"reduce": "none"})
```

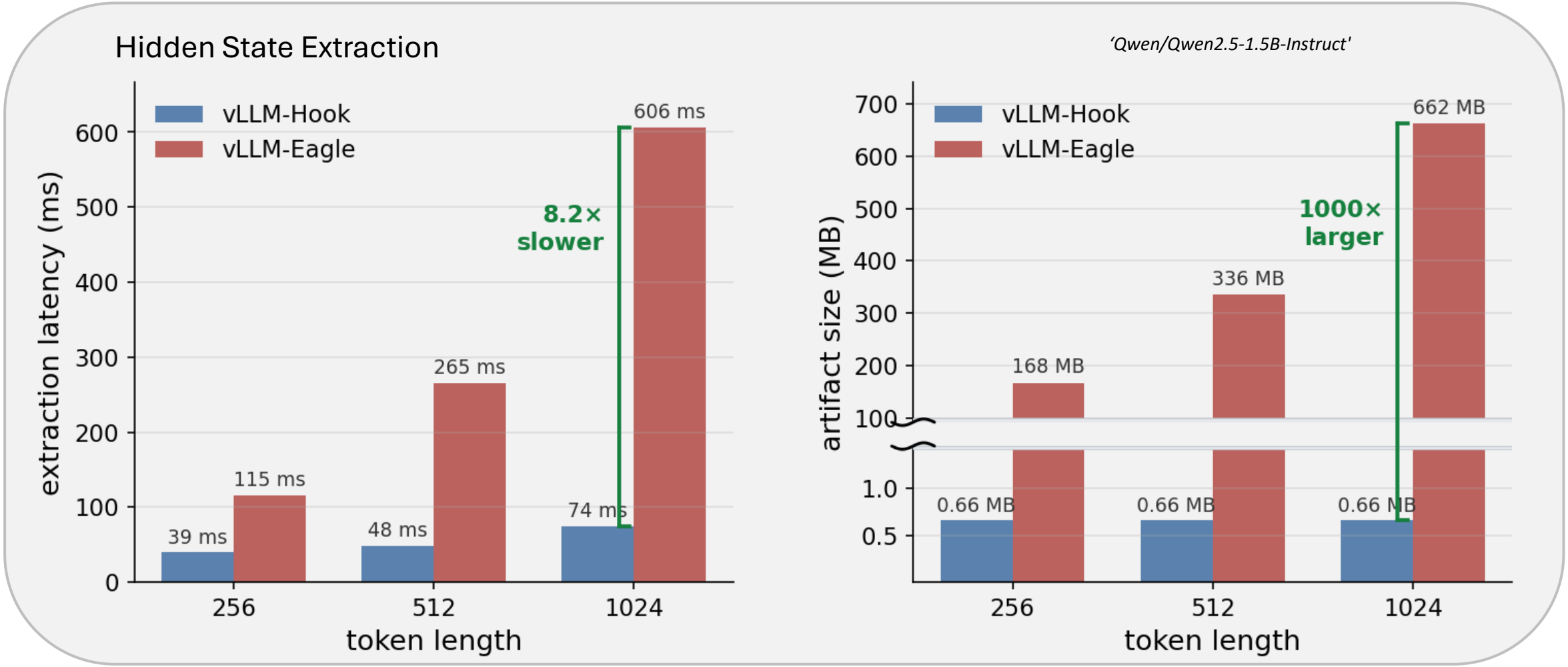
vLLM support with high efficiency

vLLM hidden state extraction using vLLM Hook directly

vLLM Eagle

Current hidden state extraction on vLLM relues on speculative decoding (vLLM-Eagle) to create a draft model

Performance Evaluation



vLLM.hook is more inference-efficient than vLLM Eagle in latency and memory

Example #3: Enhanced instruction following via activation steering

- **Goal:** Enable activation steering **in vLLM**
- **Objective:** achieve **low inference cost** and **high runtime control**

```python
from vllm_hook_plugins import HookLLM

llm = HookLLM(
    model=model,
    worker_name="steer_hook_act",
    config_file=f'{model.split("/")[-1]}.json',
    enable_hook=True
)
output = llm.generate(prompt)
```

vLLM support

vLLM enabled activation steering

HuggingFace transformer (HF) support only

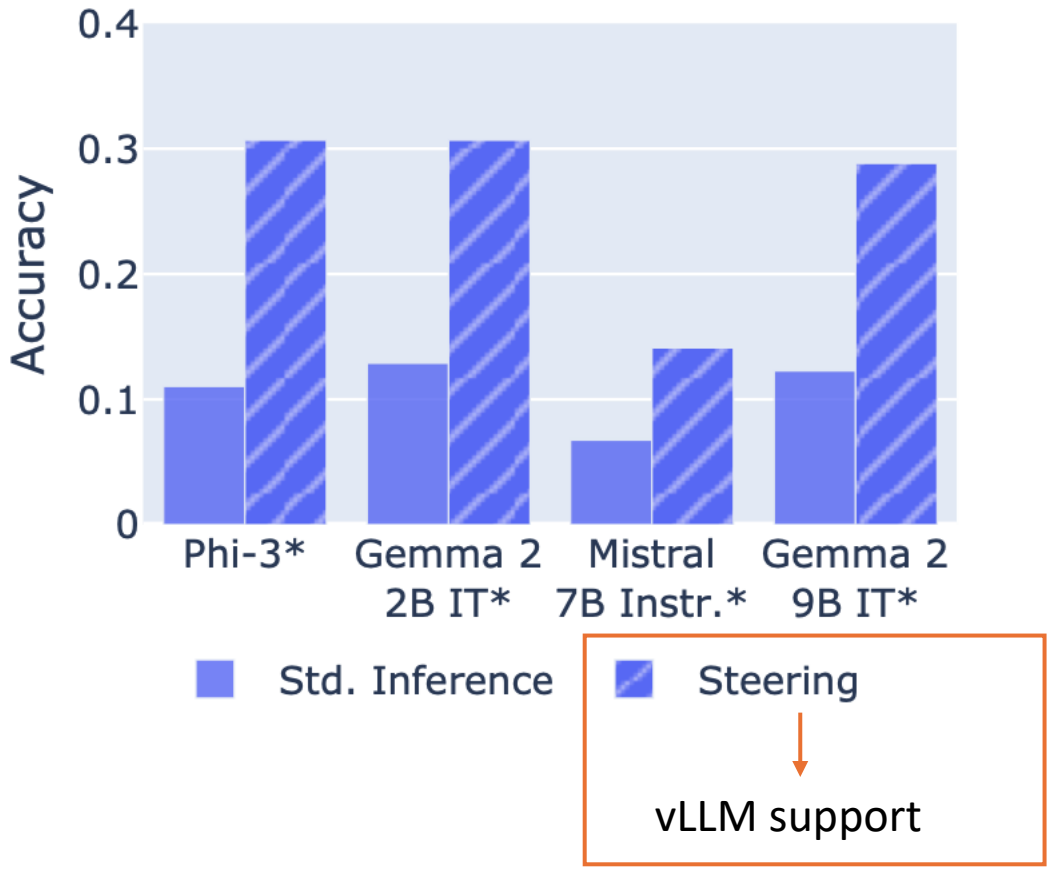
current activation steering

Improving instruction-following in language models through activation steering, ICLR'25

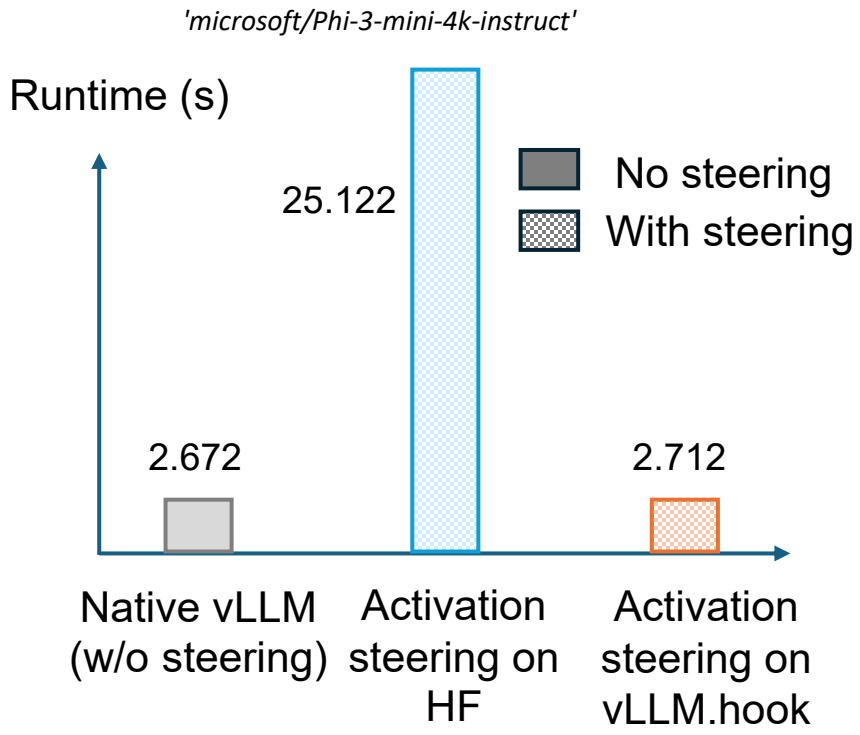
https://arxiv.org/pdf/2410.12877

Jupyter notebook

Performance Evaluation



This Example

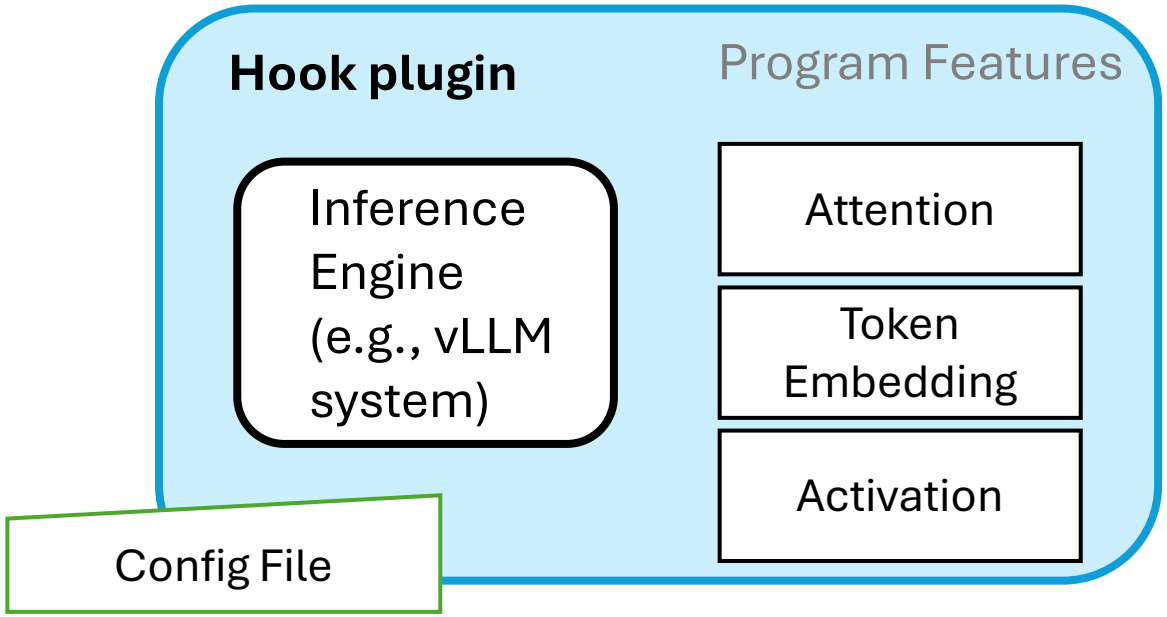


vLLM.hook enables activation modification on vLLM with low inference cost

Live Demo of vLLM Hook

Irene Ko

vLLM Hook: Opensource SDK and Organic Library for Programming vLLM Model Internals



Passive Programming

Active Programming

Competitive Analysis

- Enable programming model intrinsics on vLLM
- In-model performance enhancement with marginal runtime cost
- Improved inference efficiency while achieving same performance

- **Close the end-to-end programming gap:** Enable programmability on vLLM. Based on a configuration file, hook and program what is disabled by an inference engine compared to the development phase
- **Build passive and active programming functions:** Worker functions to support different programming features
- **Reuse and opensource:** Representative examples and demos for passive and active programming

More Use Cases and Applications

Use Case	Method	Contributor(s)
Apple Silicon Support for vLLM Hook	via vLLM-Metal	@tburleyinfo
Selective Data Retrieval	Core Reranker	@IRENEKO
Attention Steering	Spotlight	@danishcontractor
Prompt Attribution	Token Highlighter	@asanth7
Hallucination Detection	Last Token Embedding	@IRENEKO
Hallucination Detection	H-Node	Not yet merged

Programming Model Internals is the *New Harness*

Inference-centric AI eco-systems

- ➢Test-time scaling & optimization
- ➢Agentic workflows and operations
- ➢Recursive intelligence

We welcome contributions to

- Develop new core programming features
- Develop new use cases and demos
- Use and share programming functions
- Inform us new programming needs

vLLM Hook