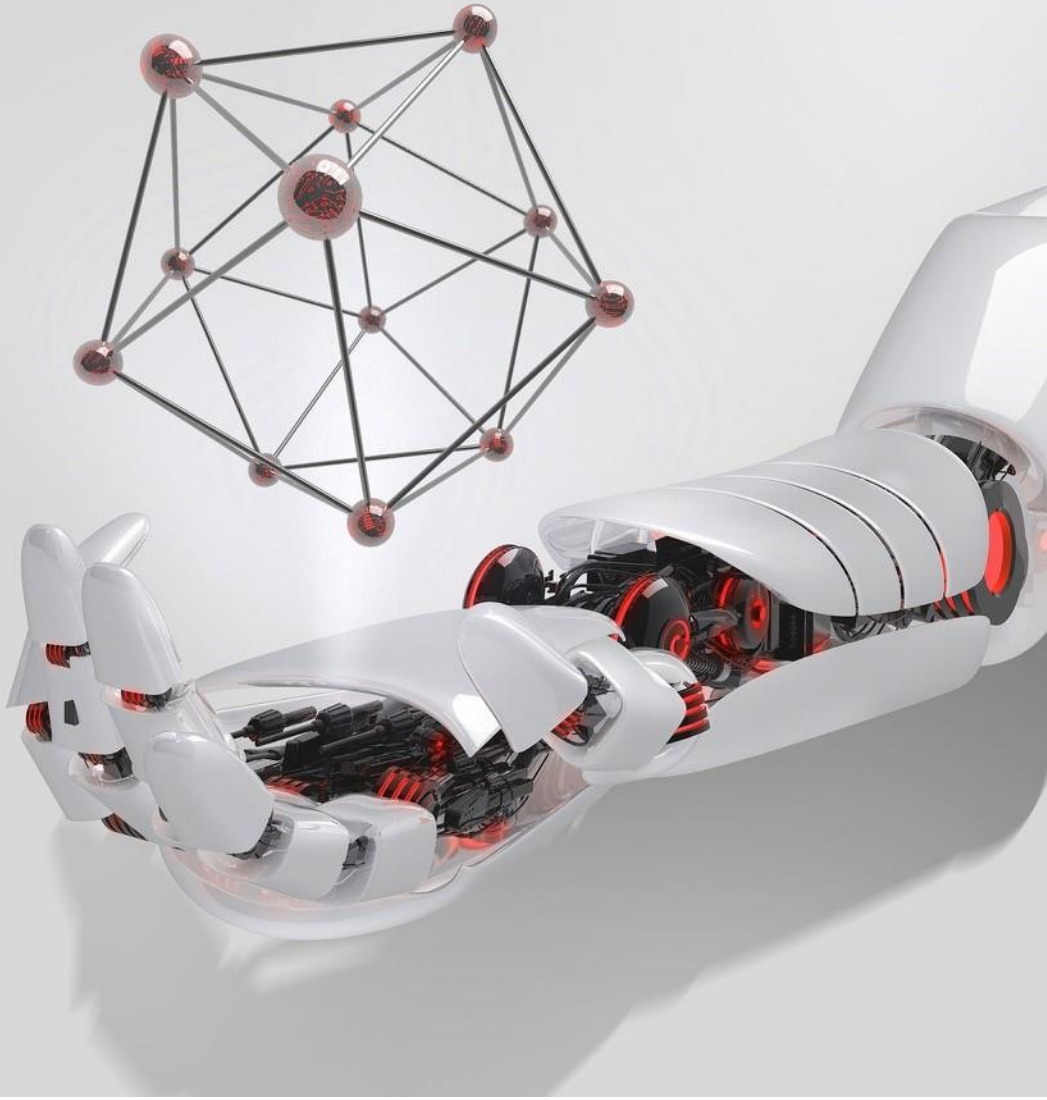
ProfInfer: An eBPF-based Fine-Grained LLM Inference Profiler

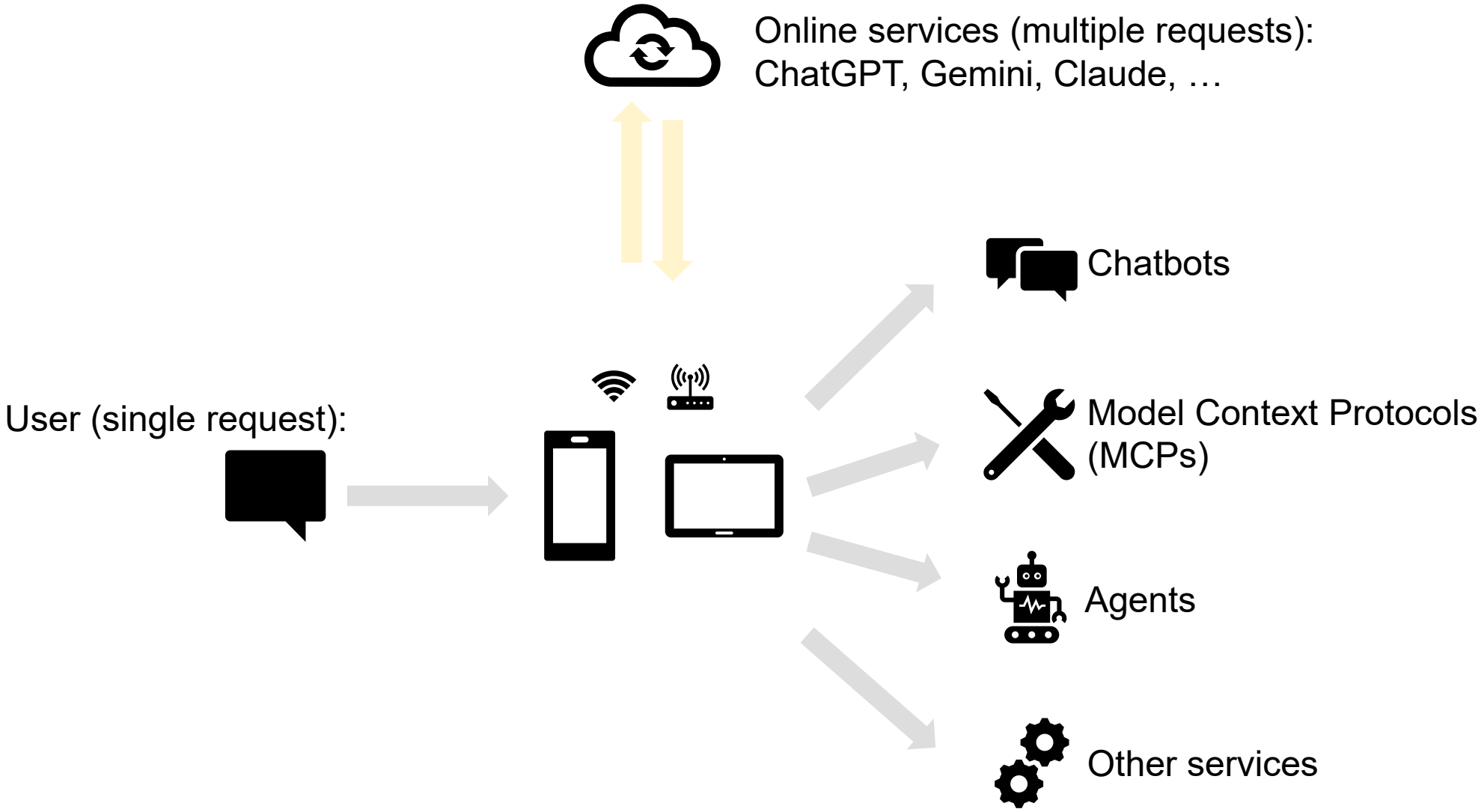
Bohua Zou[1,2], Debayan Roy[1], Dhimankumar Yogesh Airao[1], Weihao Xu[2], Binqi Sun[2], Yutao Liu[1], Haibo Chen[3,4]

1 Huawei Hilbert Research Center (Dresden),
2 Technical University of Munich,
3 Huawei Center Software Institute,
4 Shanghai Jiao Tong University

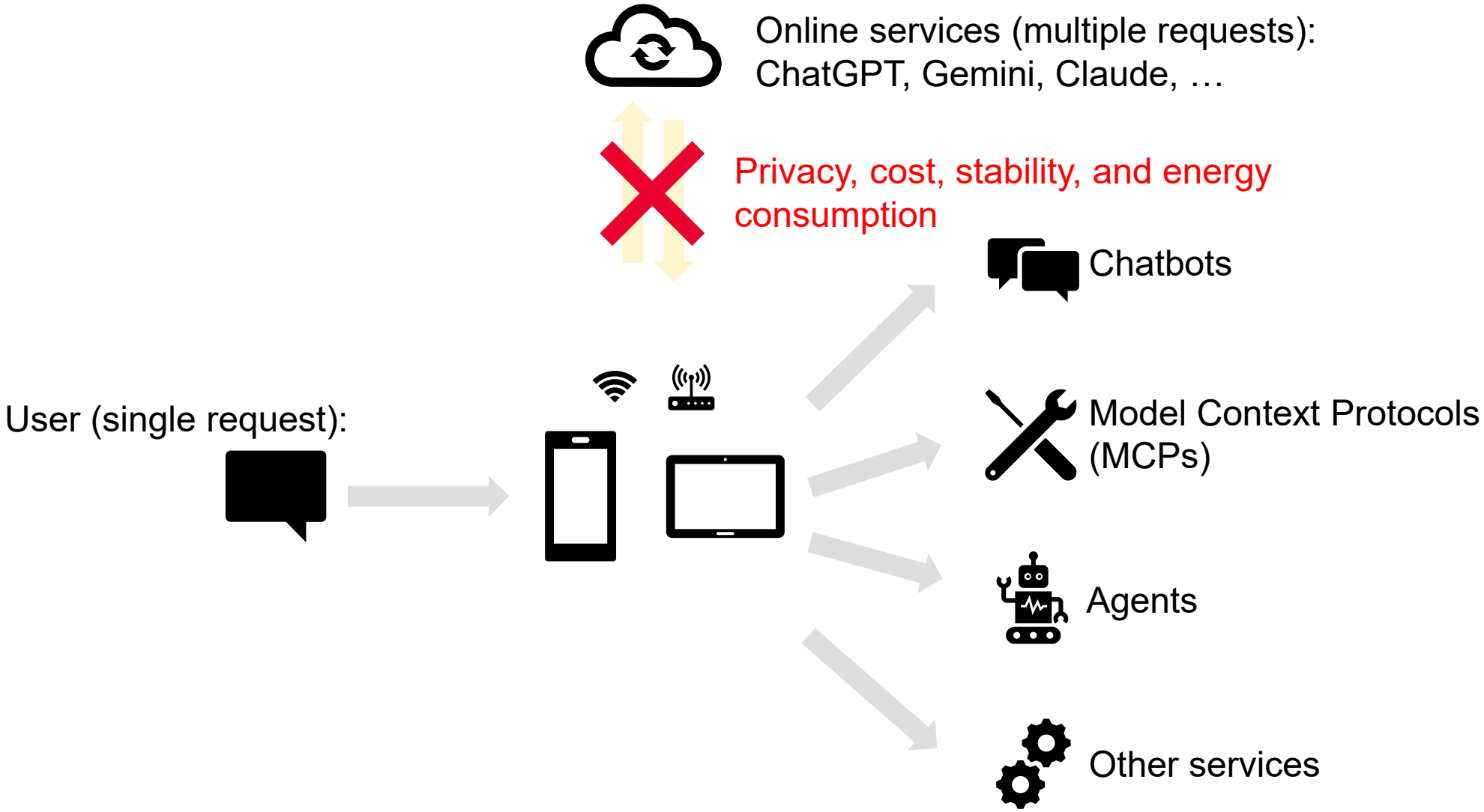
May 21st, 2026,
Bellevue, WA, USA
MLSys 2026 Industry Track



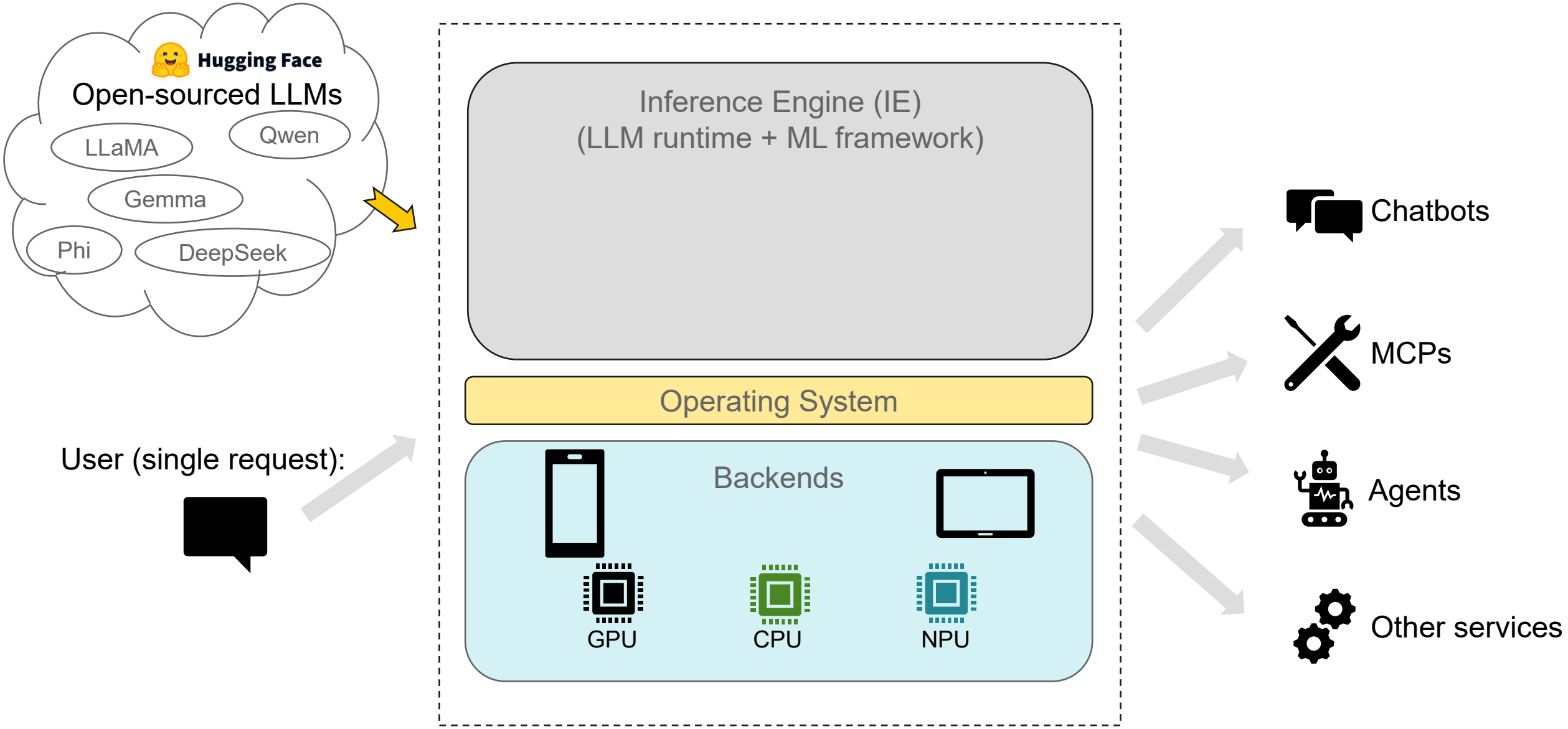
LLM inference: From cloud to mobile and edge devices



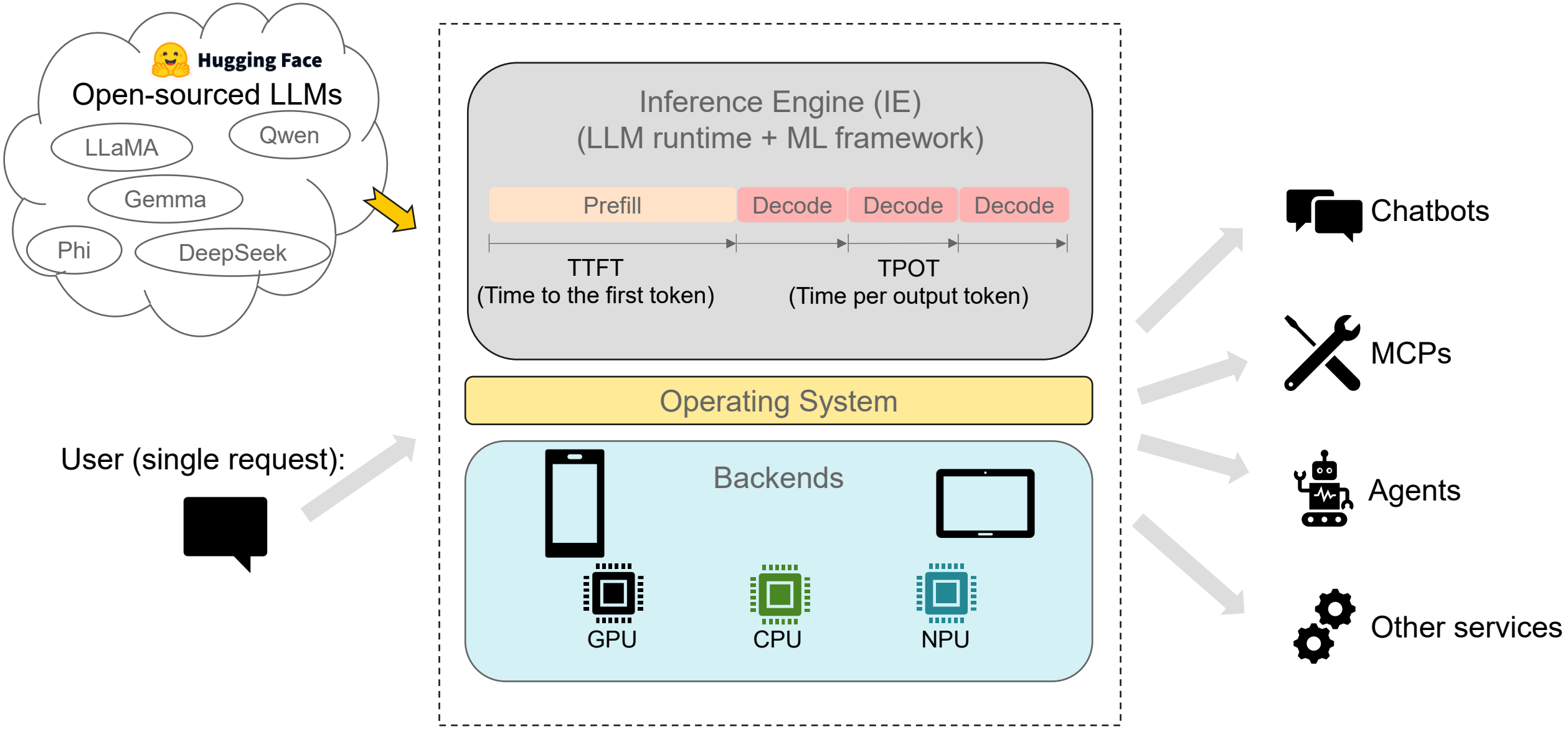
LLM inference: From cloud to mobile and edge devices



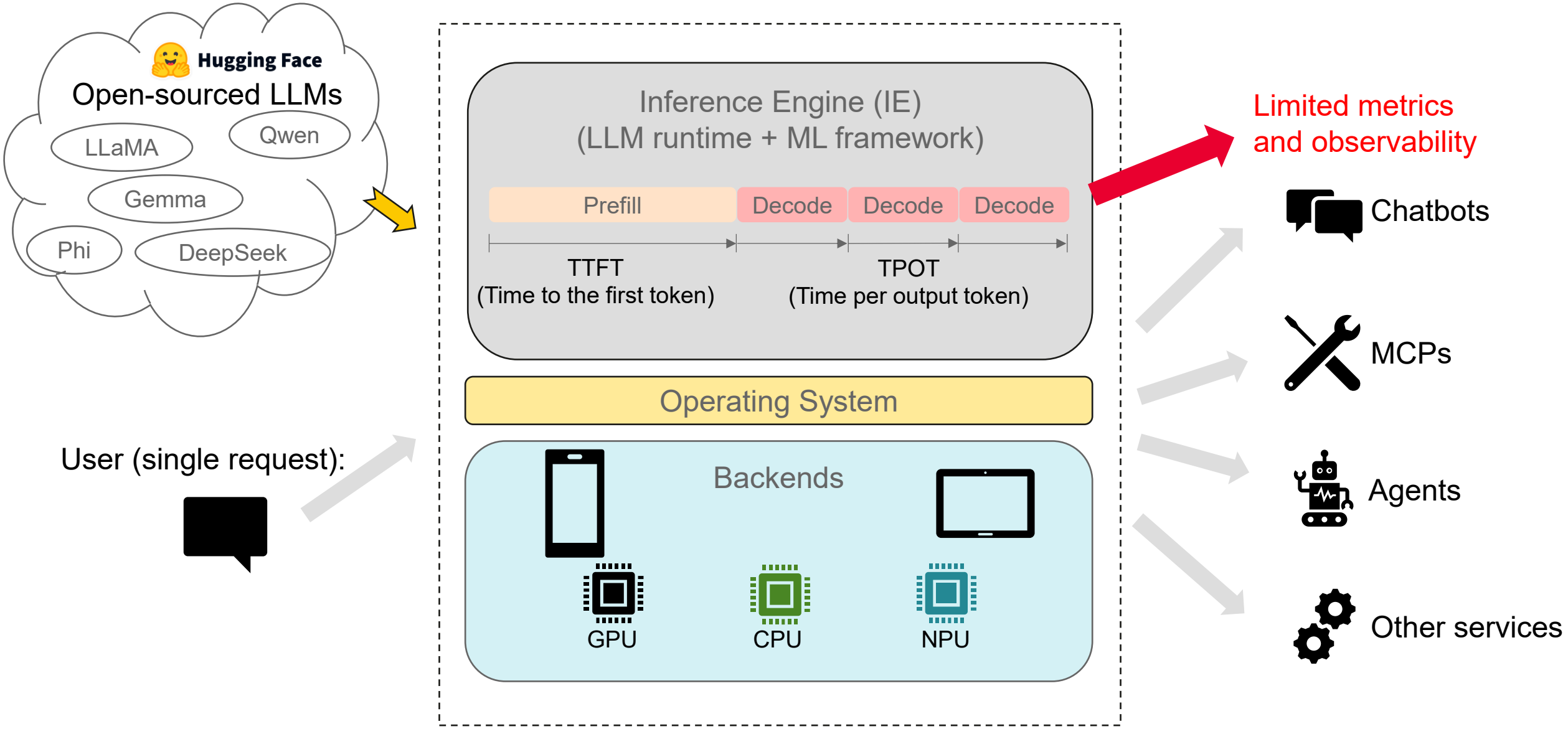
On-device LLM inference: limited metrics and observability



On-device LLM inference: limited metrics and observability

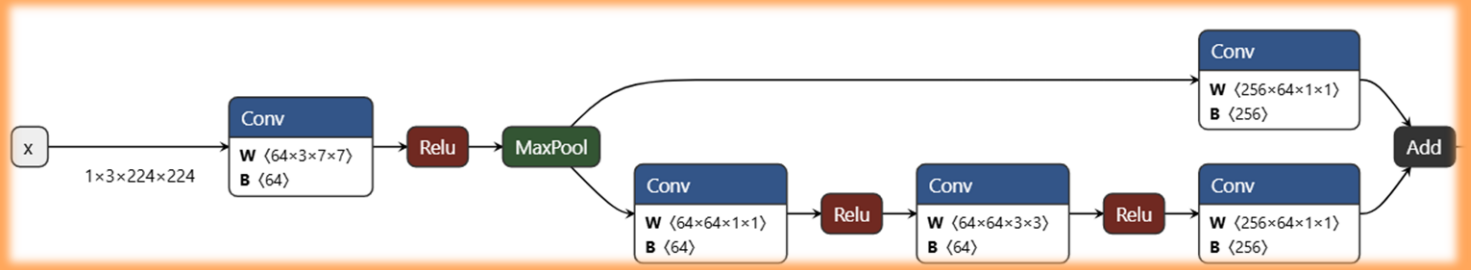


On-device LLM inference: limited metrics and observability



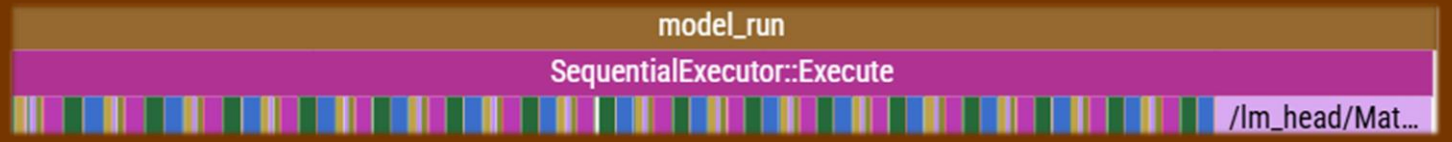
Existing profiling tools

Netron: ***Static*** model structural analysis.



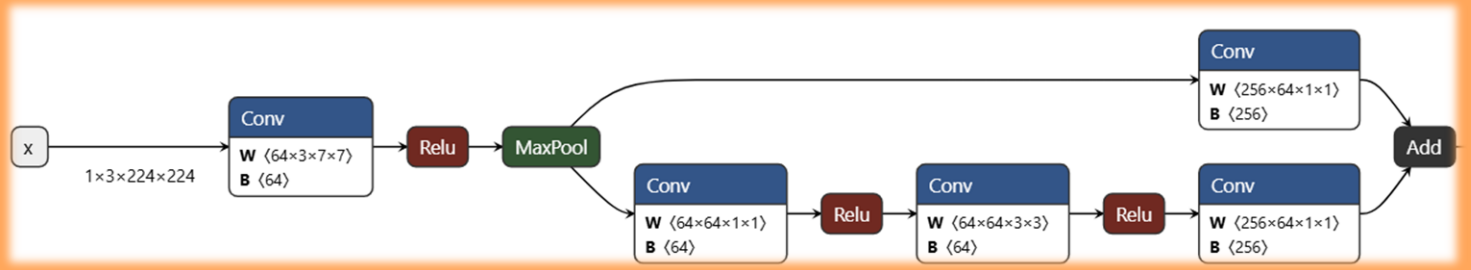
llama-bench: ***Coarse-grained*** metrics (TTFT, TPOT).

ONNX Runtime profiler: Pre-defined ***intrusive*** tracing.



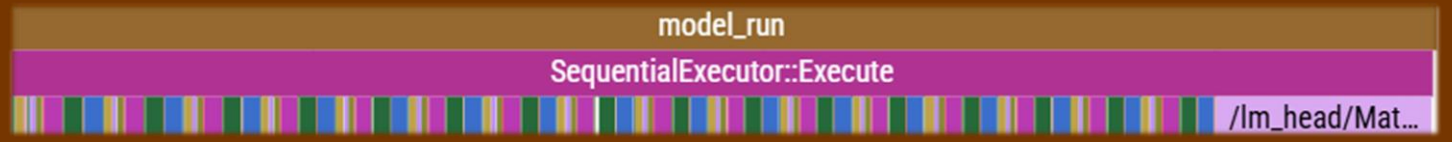
Existing profiling tools

Netron: ***Static*** model structural analysis.



llama-bench: ***Coarse-grained*** metrics (TTFT, TPOT).

ONNX Runtime profiler: Pre-defined ***intrusive*** tracing.



Dynamic

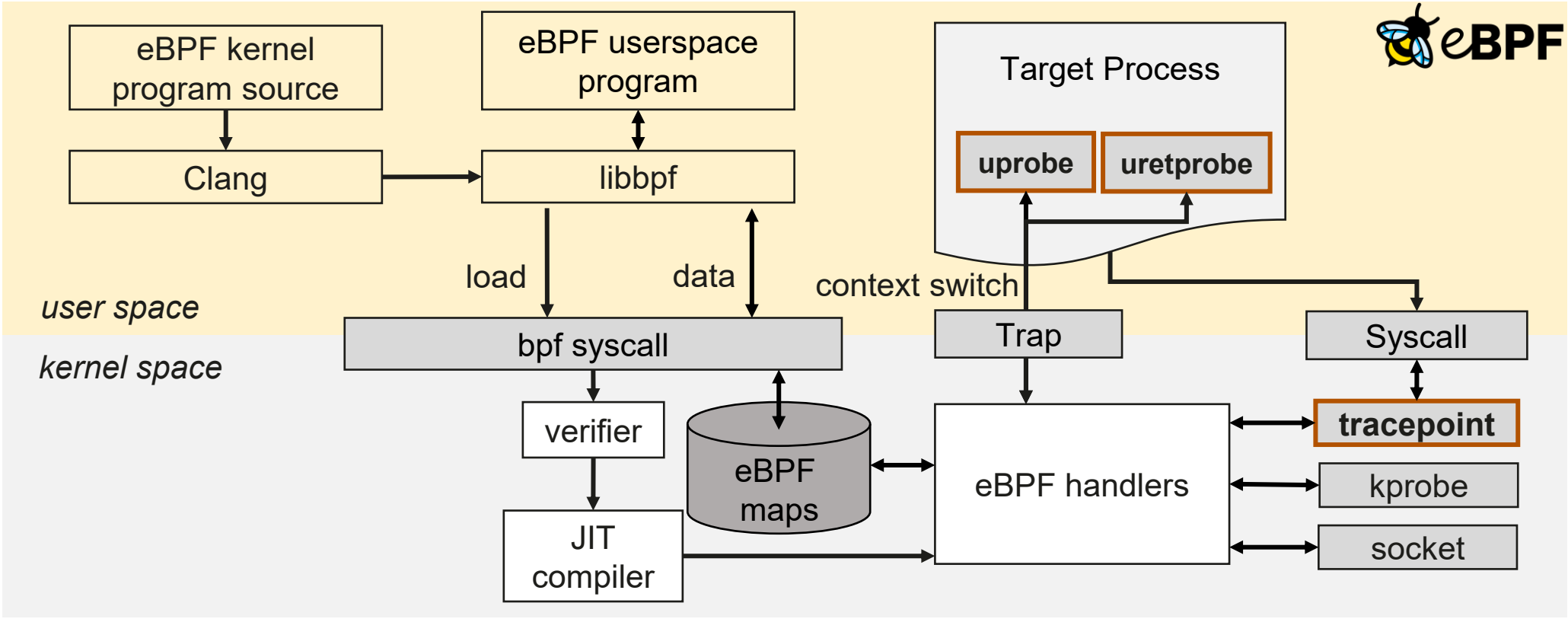
Fine-grained

Non-intrusive

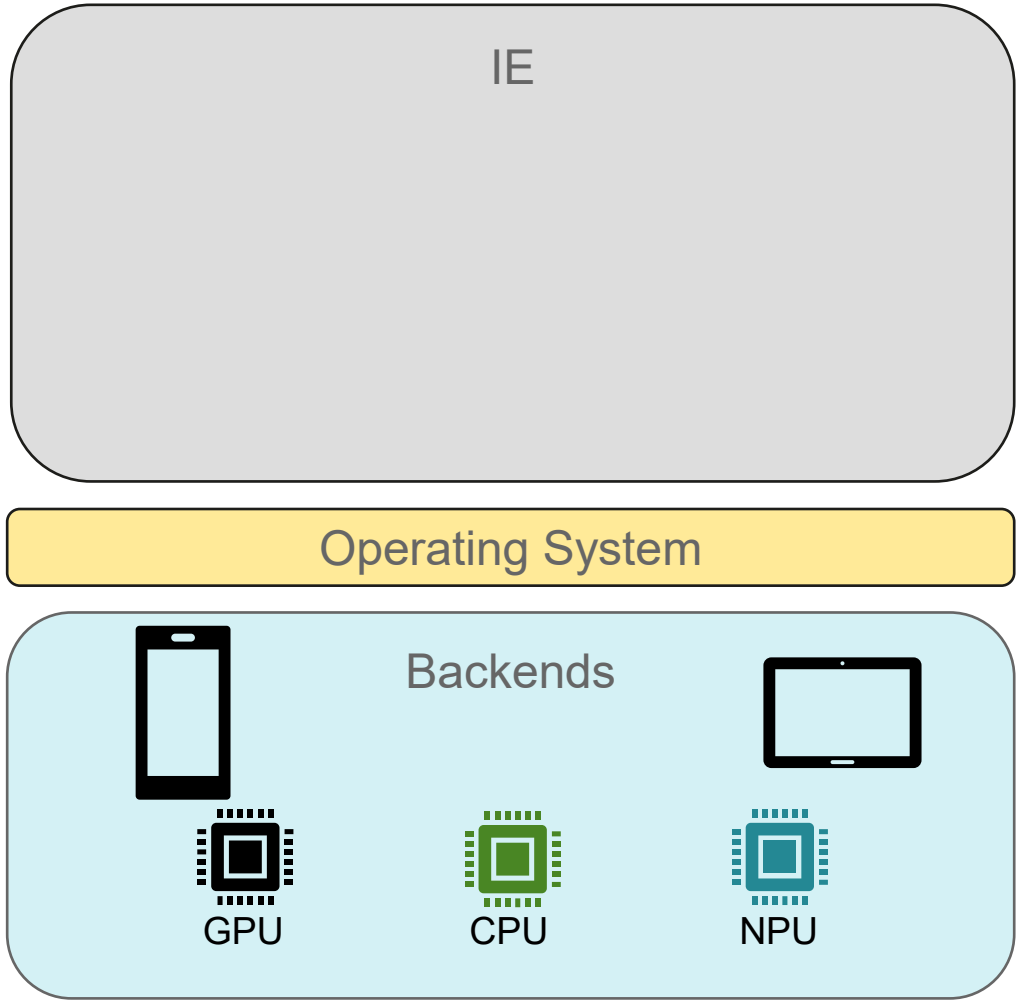
ProfInfer

Background: **e**xtended **B**erkeley **P**ackage **F**ilter (eBPF)

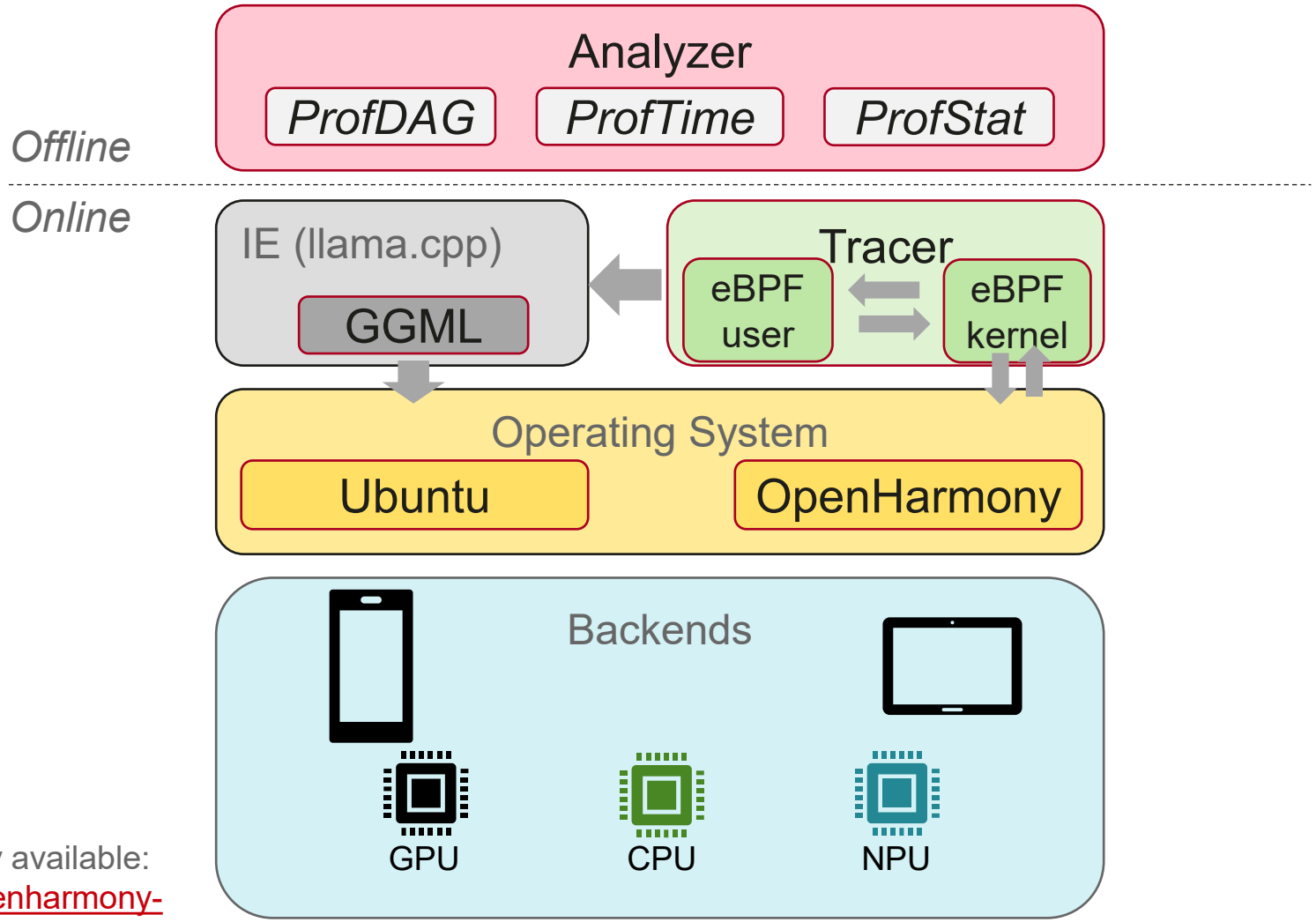
- **Flexibility:** non-intrusive, programmable, without changing kernel or loading kernel module.
- **Safety:** sandboxed program within OS kernel.



Overview design of ProfInfer

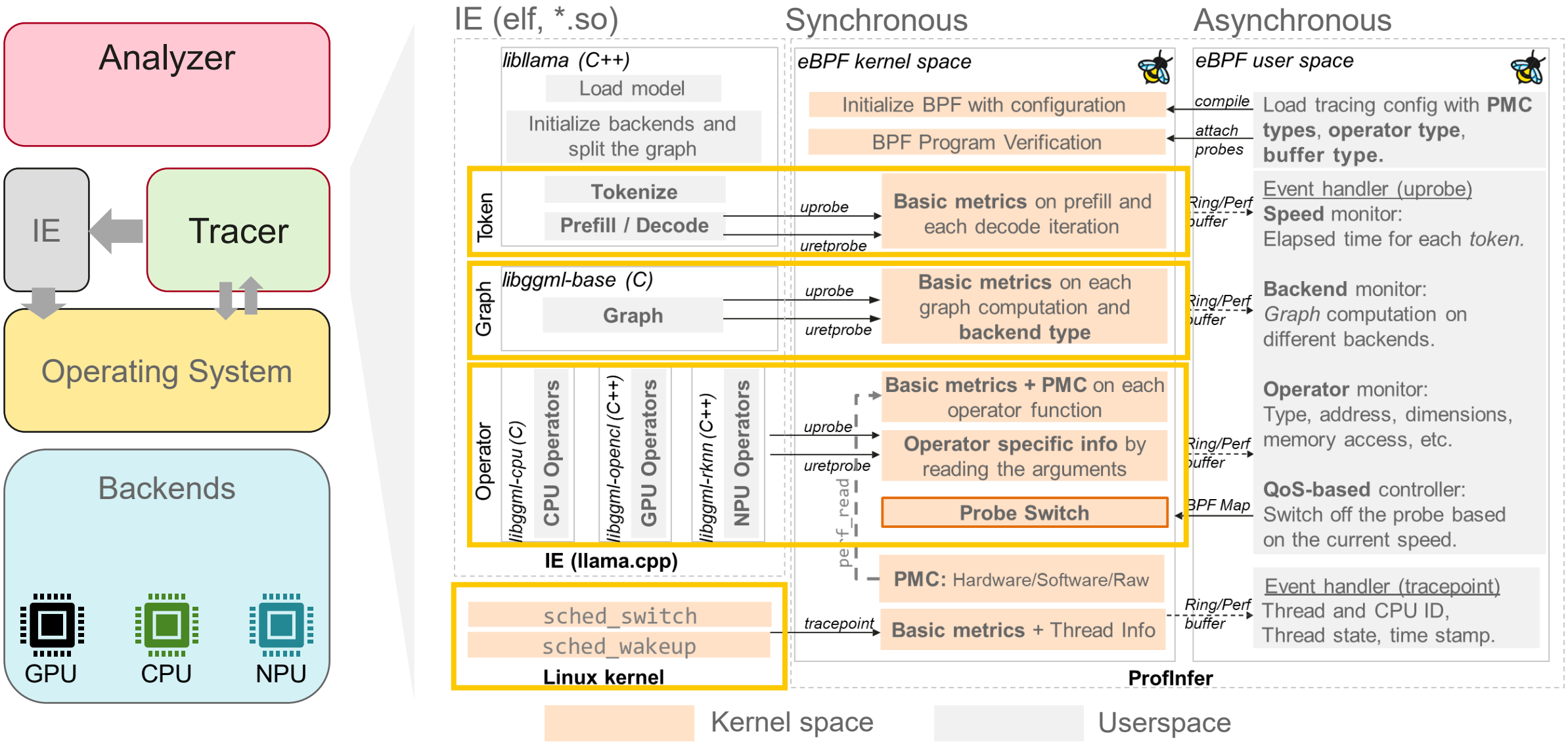


Overview design of ProfInfer

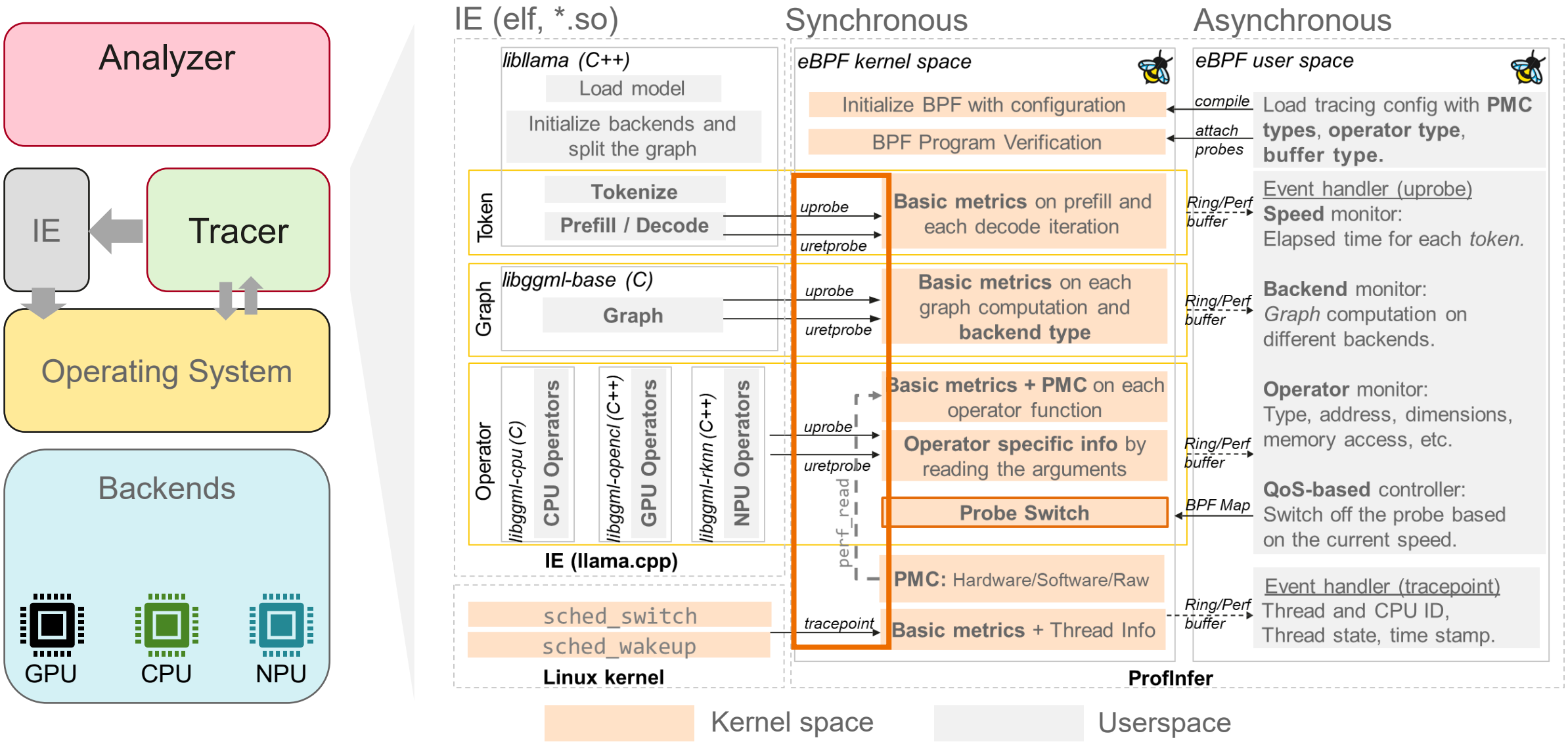


Source code is publicly available:
https://gitcode.com/openharmony-robot/oh-llama.cpp/tree/main/profinfer

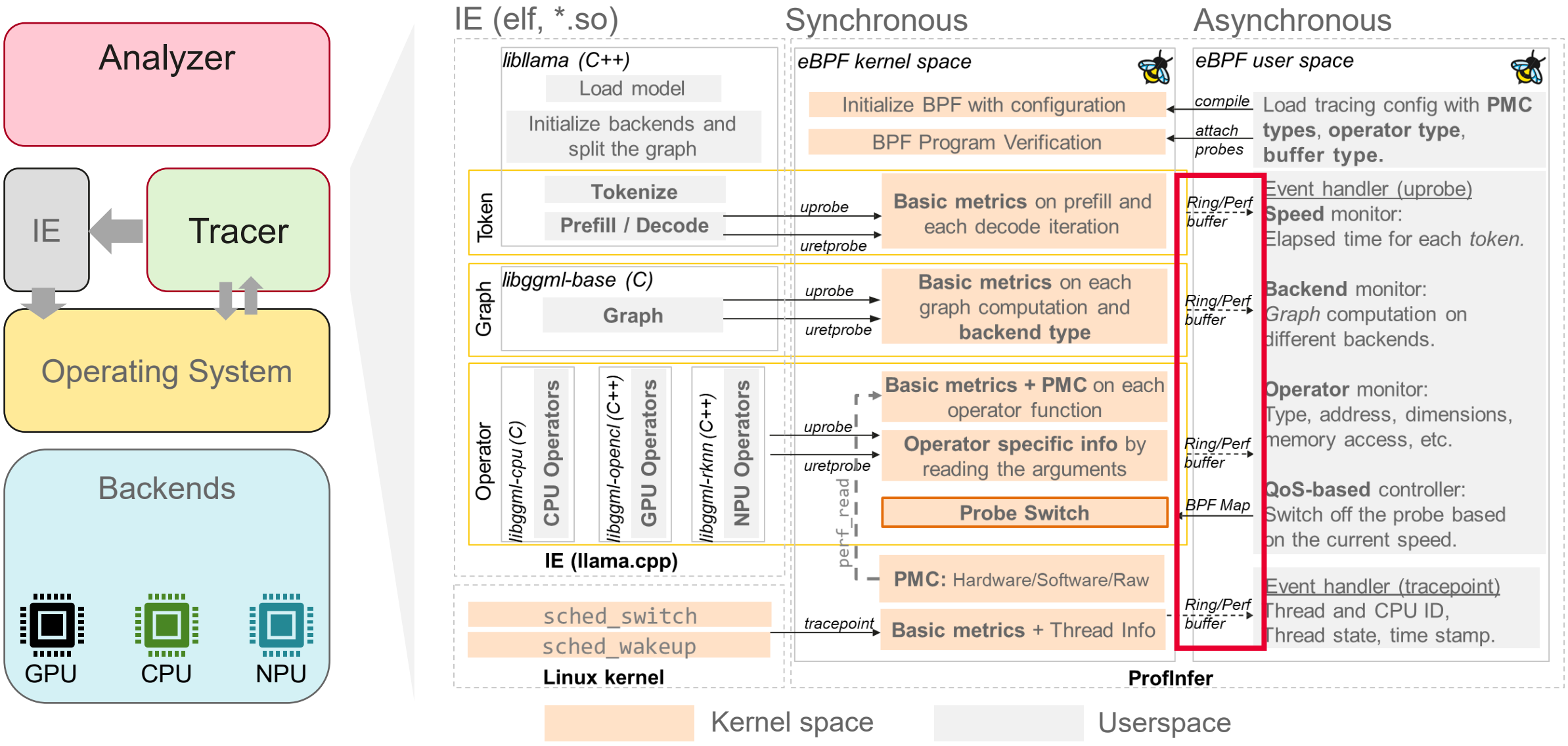
Lightweight eBPF-based tracing (online)



Lightweight eBPF-based tracing (online)



Lightweight eBPF-based tracing (online)



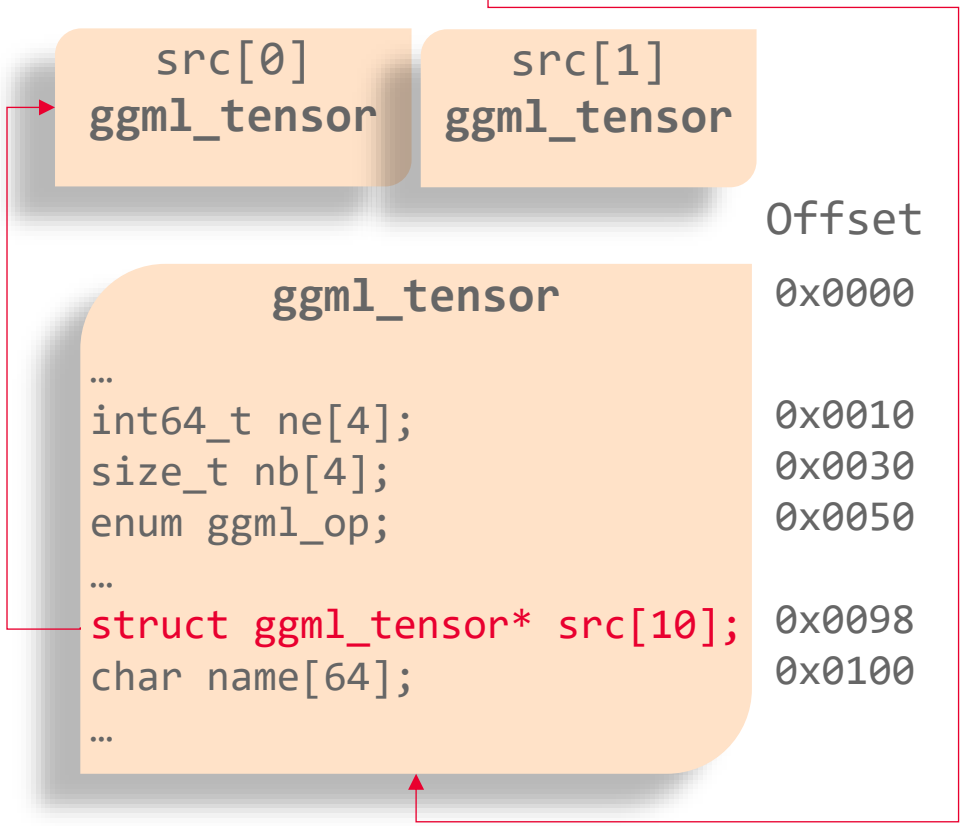
Lightweight eBPF-based tracing (online)

E.g.: Operator-level function:

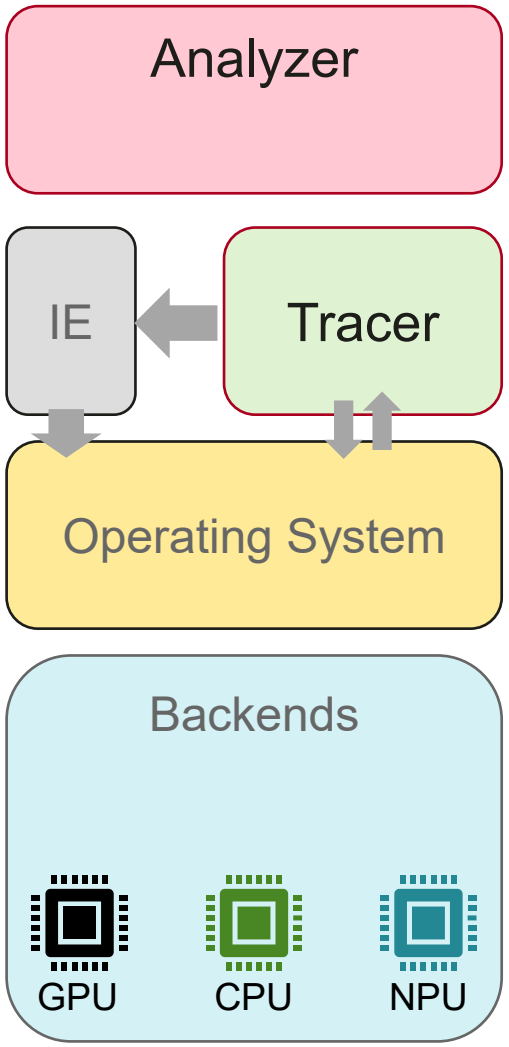
```
void ggml_compute_forward(…, struct ggml_tensor * tensor)
```

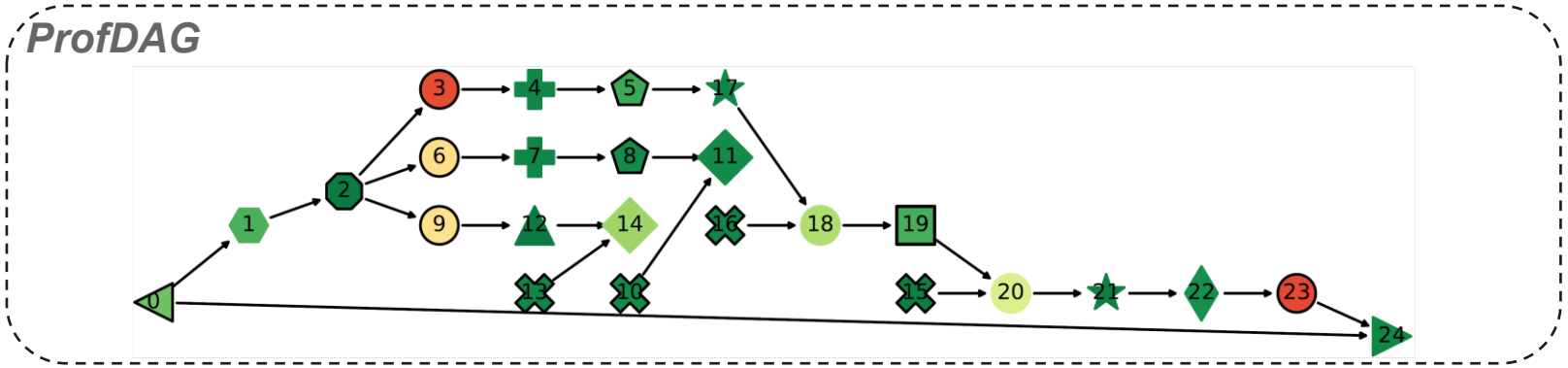
BPF Helpers	Features
bpf_ktime_get_ns	Timestamp
bpf_get_current_pid_tgid	PID/TID
bpf_get_smp_processor_id	CPU ID
map.perf_read	PMC
bpf_probe_read_user	Read userspace address

PMC: Performance Monitoring Counters

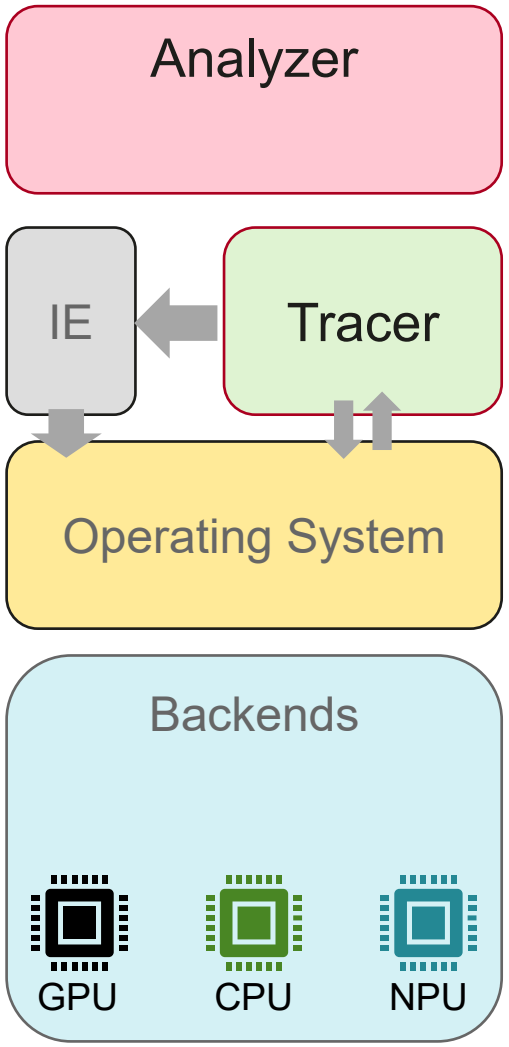


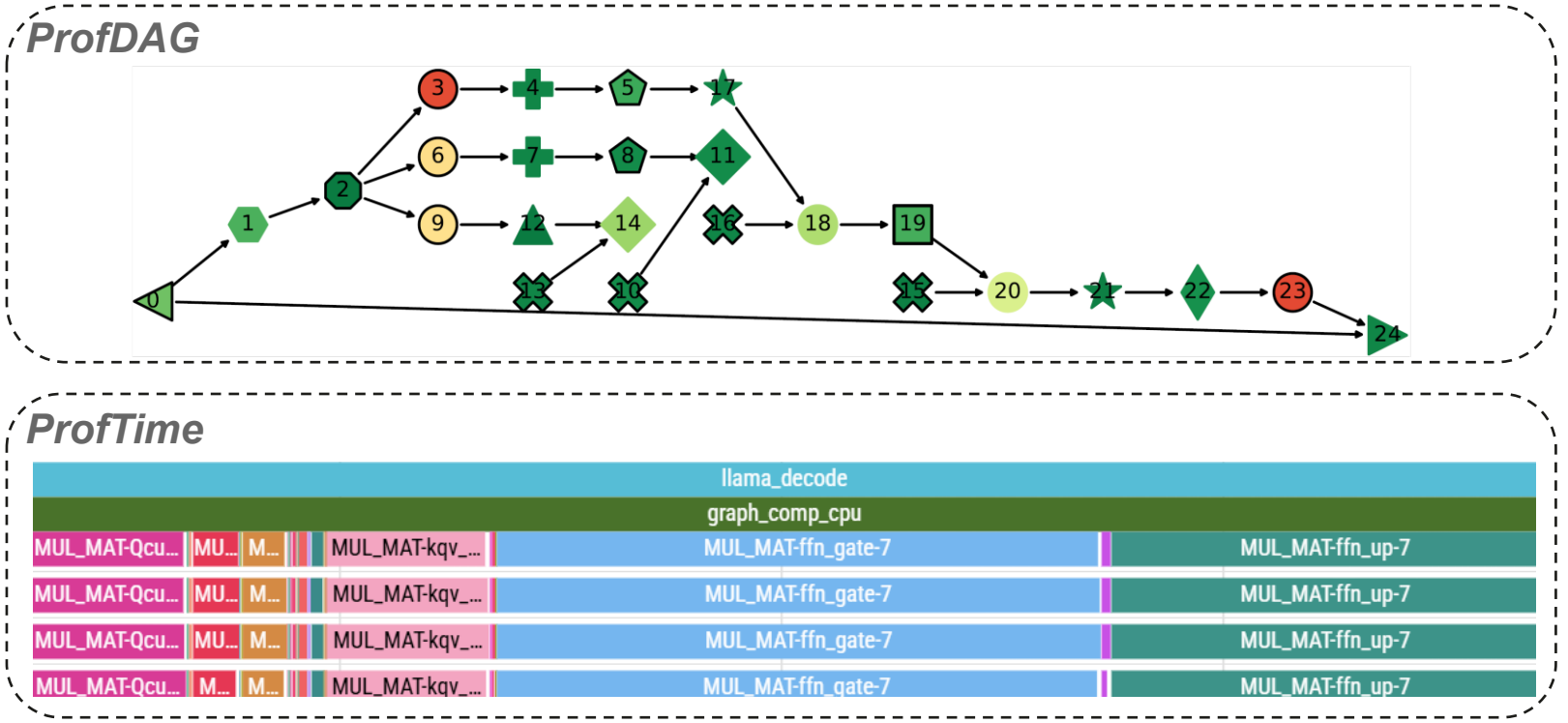
Operator-level fine-grained profiling (offline)



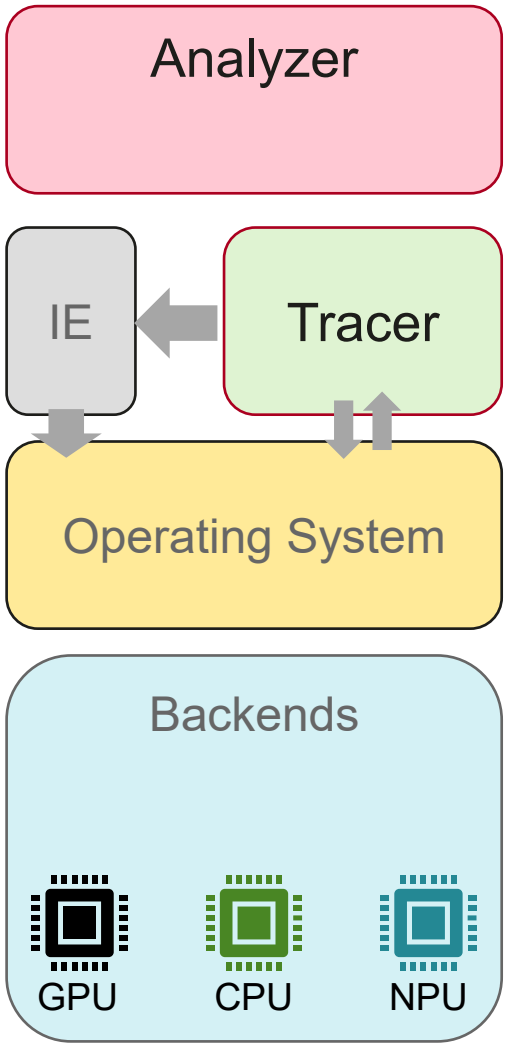


Operator-level fine-grained profiling (offline)

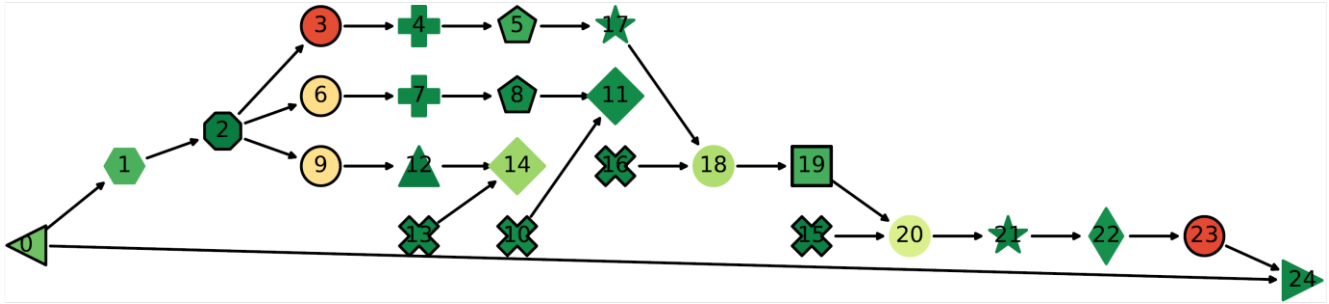




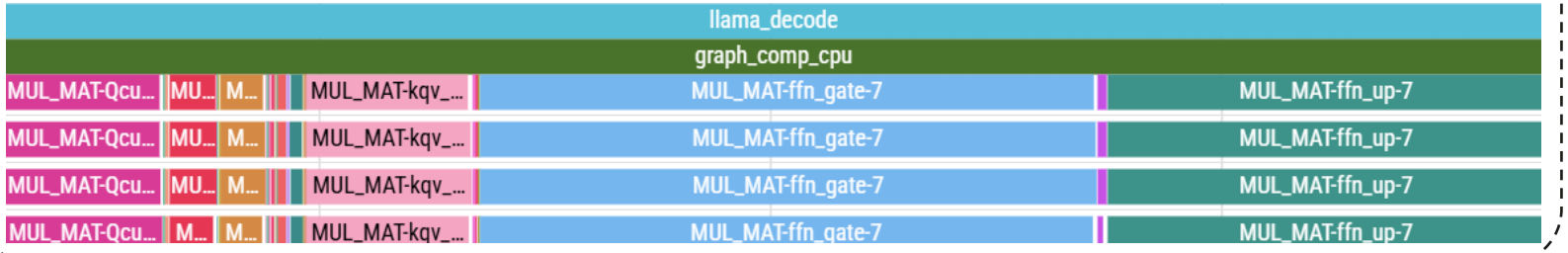
Operator-level fine-grained profiling (offline)



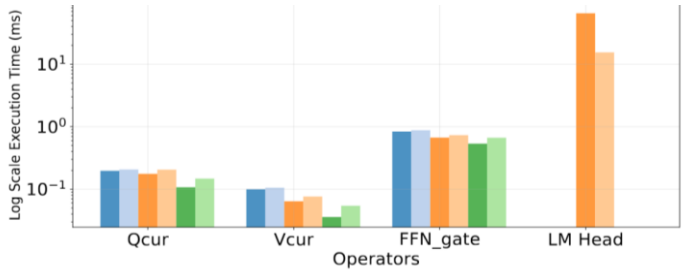
ProfDAG

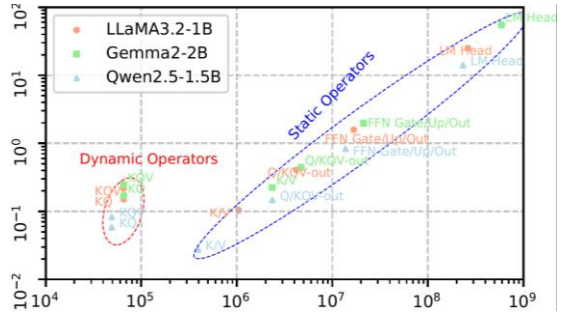


ProfTime

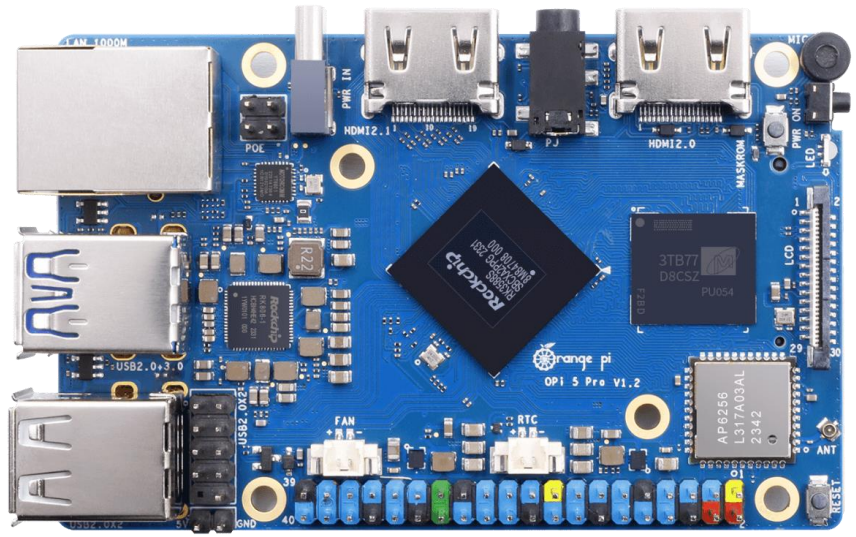


ProfStat

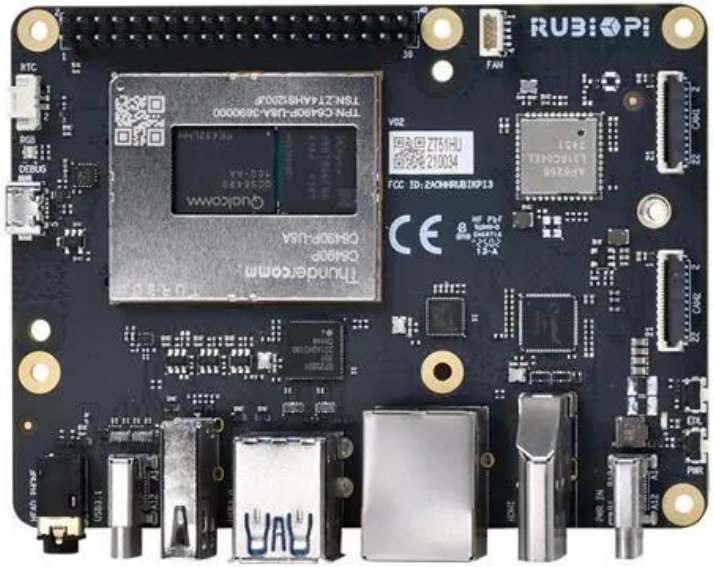




Evaluations: System setup



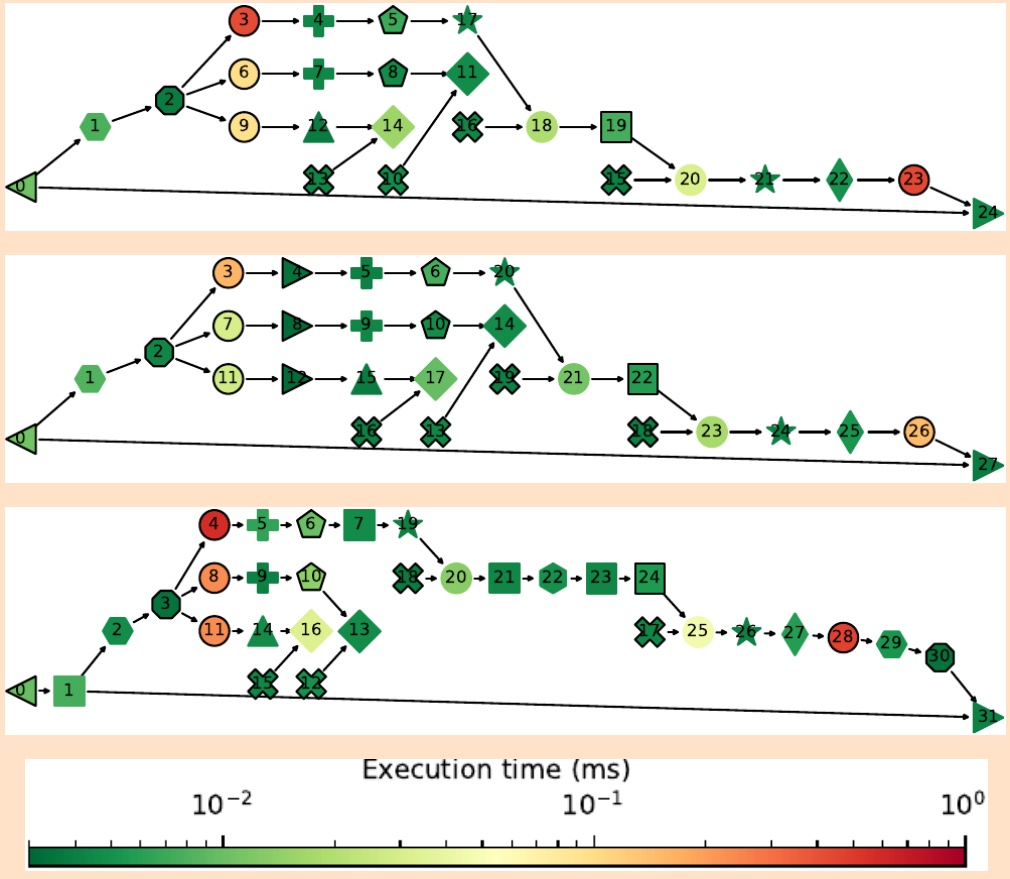
Orange Pi 5 Pro/Ultra/Plus
SoC: RK3588/RK3588s
CPU: Cortex-A76x4 + Cortex-A55x4
GPU: ARM Mali-G610
NPU: Rockchip NPU 6 TOPS
Memory: LPDDR4/4X/5 8/16GB
OS: Ubuntu 22.04 / OpenHarmony 5.1.0



Rubik Pi 3
SoC: QCS6490
CPU: Cortex-A78x4 + Cortex-A55x4
GPU: Adreno GPU 643
NPU: 12 TOPS
Memory: LPDDR4 8GB
OS: Ubuntu 24.04

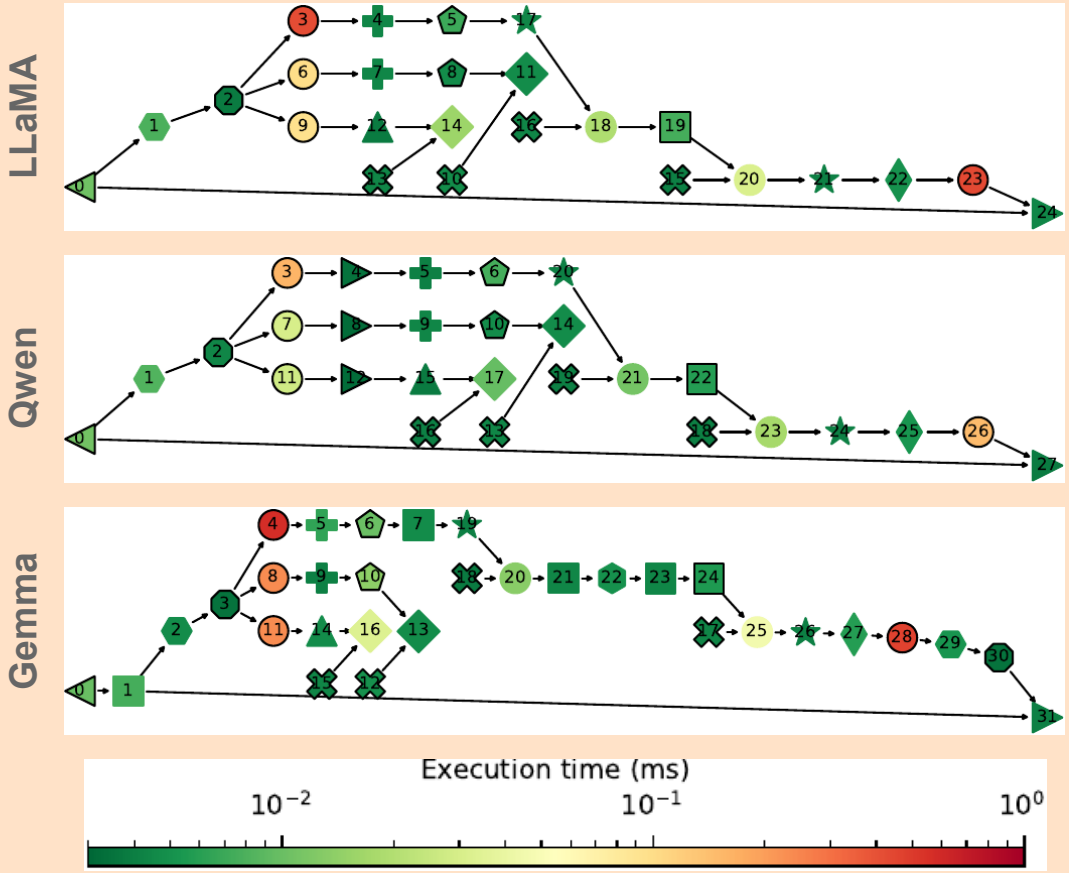
Evaluations: Profile-driven insights—ProfDAG

- **Different topologies across LLM families**



Evaluations: Profile-driven insights—ProfDAG

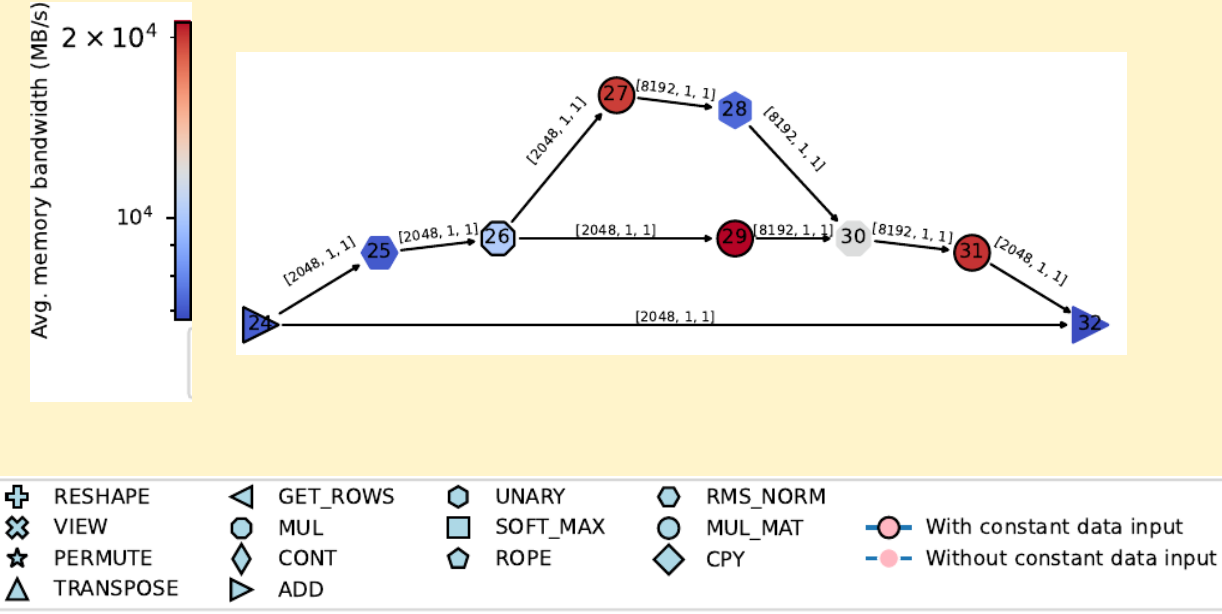
- **Different topologies across LLM families**



- **Memory bandwidth usage varies across operators**

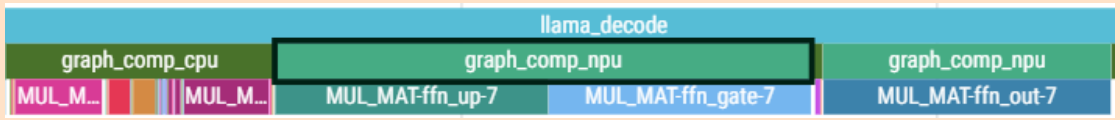
Performance Monitoring Counter (PMC)

Name	Description
l3d_cache_refill	Read from memory
mem_access_wr	Write into memory



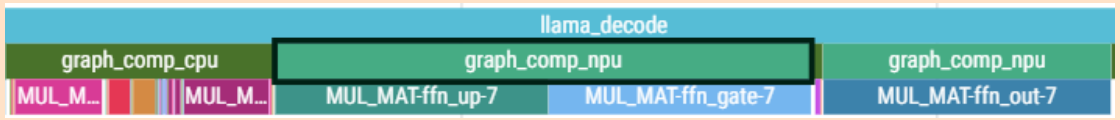
Evaluations: Profile-driven insights—ProfTime

- **Selective operator offloading** on different backends

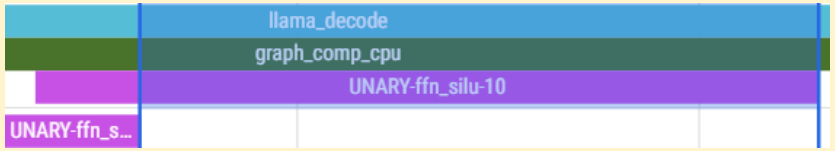


Evaluations: Profile-driven insights—ProfTime

- **Selective operator offloading** on different backends

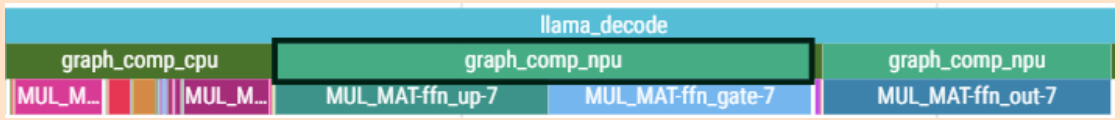


- **Unbalanced operators** across threads

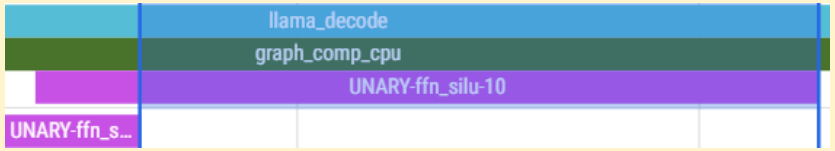


Evaluations: Profile-driven insights—ProfTime

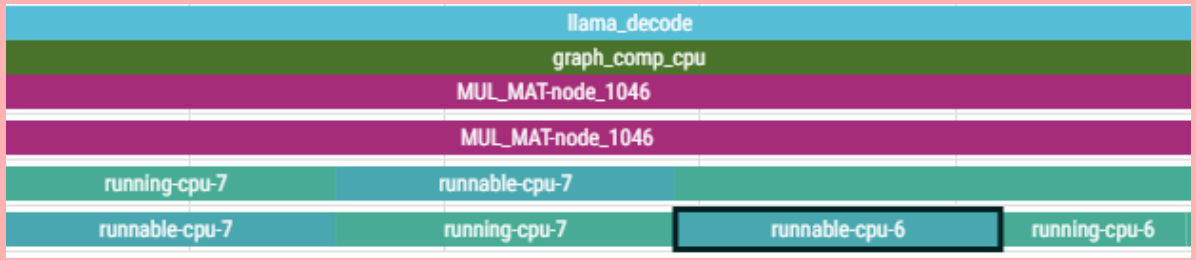
- **Selective operator offloading** on different backends



- **Unbalanced operators** across threads



- **Interfering task** detection



Evaluations: Profile-driven insights—ProfStat

- Across tokens:

KV cache growth impacts **dynamic operators**: **KQ** and **KQV**

- Per-operator:

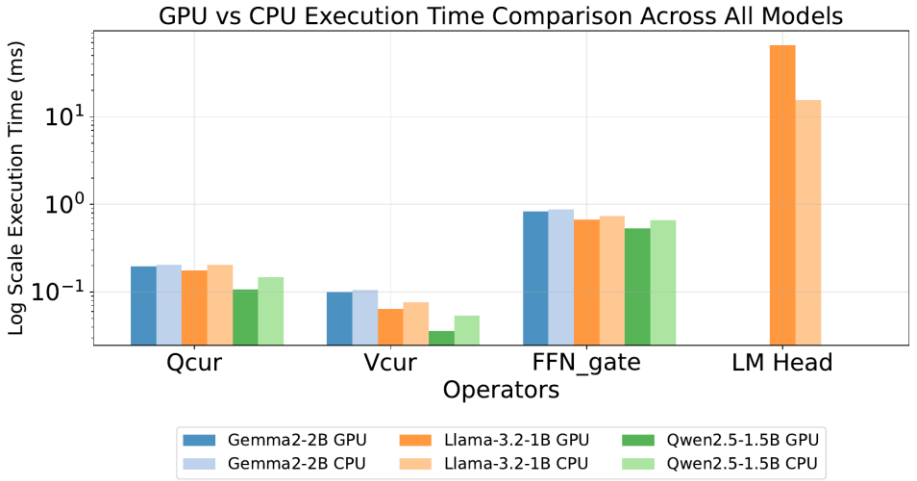
The execution time of **static** matrix multiplication are **proportional** to their **computational complexities**

- PMC-driven:

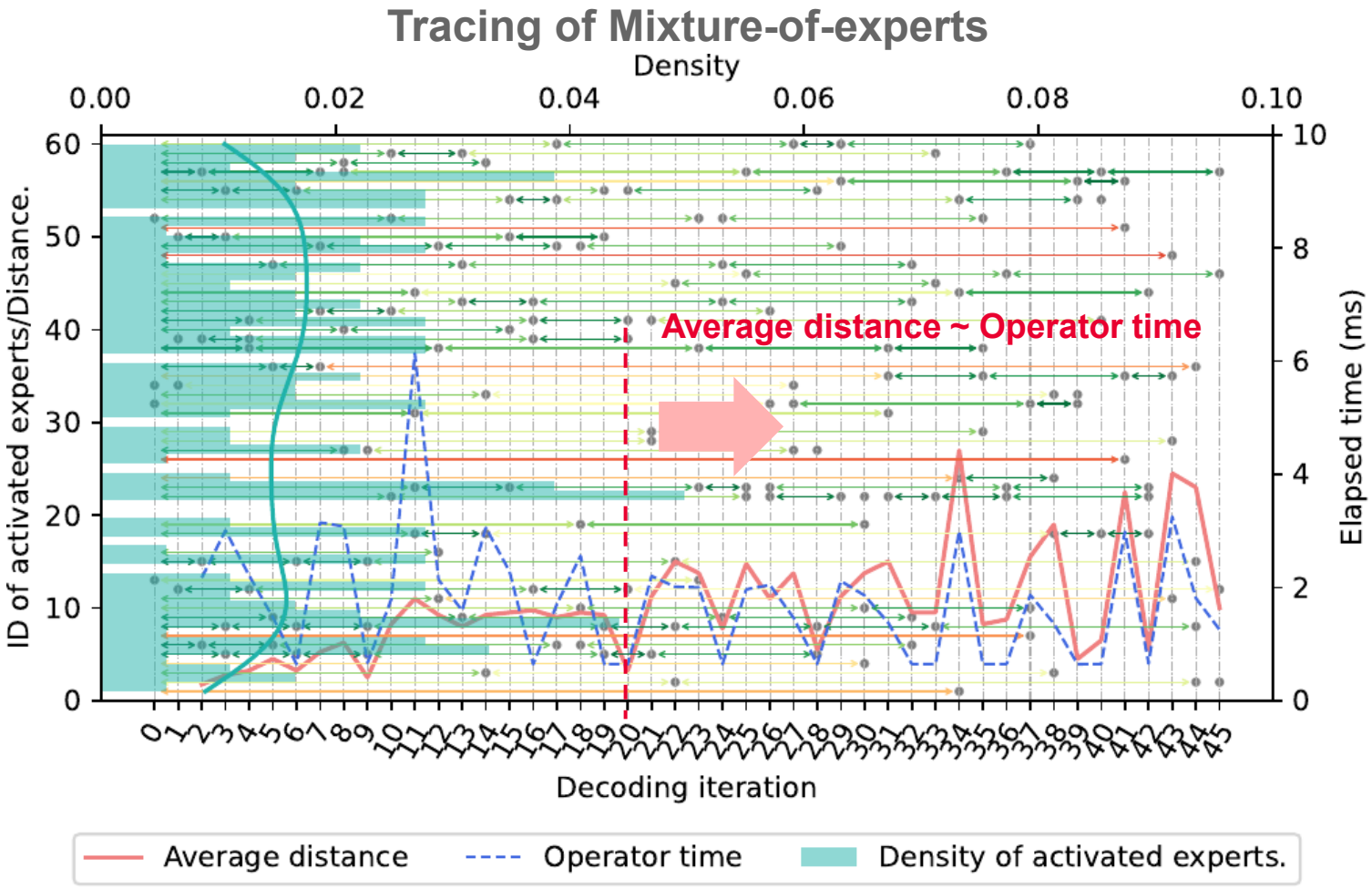
Matrix-vector multiplication performance is correlated to the **memory access**. More than 50% CPU cycles are **stalled**

- Across backends:

GPU/NPU only outperforms **CPU** at certain operator **dimensions**



Evaluations: Profile-driven insights—ProfStat



Distance: the token distance since the expert's last activation

Evaluations: Low-overhead online trace

Overhead analysis

	Structural information	PMC	Perf buffer (Ring buffer)	Relative decoding speed decrease (%)	CPU load of probe handlers (%)	Power Increase (W)	Temperature Increase (℃)
BCC	✓	✓	✓	3.67±1.21	0.70±0.01	0.06±0.29	1.22±0.62
	x	x	x	1.63±0.66	0.61±0.00	0.00±0.34	0.85±0.76
libbpf	✓	x	x	1.46±0.30	0.41±0.00	0.05±0.08	0.86±0.65
	x	x	x	**1.19±0.18**	**0.40±0.00**	**0.03±0.09**	**0.31±0.90**

ProfInfer vs. **ONNX Runtime profiler**

	Prefill Decrease (%)	Decode Decrease (%)
ProfInfer	**0.25±0.62**	**3.60±0.80**
ONNX Runtime profiler	0.46±0.70	8.91±1.94

Summary and discussions

- Summary: ***ProfInfer***—an eBPF-based fine-grained LLM inference profiler
  - > Fine-grained
  - > Non-intrusive
  - > Low-overhead

Summary and discussions

- Summary: ***ProfInfer***—an eBPF-based fine-grained LLM inference profiler
  - > Fine-grained
  - > Non-intrusive
  - > Low-overhead

- Potential use cases:
  - > Profile-guided optimization
  - > Lightweight runtime monitor

Summary and discussions

- Summary: ***ProfInfer***—an eBPF-based fine-grained LLM inference profiler
  - > Fine-grained
  - > Non-intrusive
  - > Low-overhead
- Potential use cases:
  - > Profile-guided optimization
  - > Lightweight runtime monitor
- Future work:
  - > Less overhead: libbpf and bpftime
  - > Performance adaptation

Thank you for your attention!

Questions?