Systems for Future Intelligence

Agents need abstractions too

Stephanie Wang (& Baris Kasikci)
Paul G. Allen School, University of Washington

Hello 👋

- 2024 - now: Asst professor at UW CSE, co-director of SyFI Lab with Prof Baris Kasikci
- Previously:
  - Co-creator of Ray, open source system for distributed ML and Python
  - Founding engineer at Anyscale
  - PhD at Berkeley, distributed ML and data-intensive systems

*ML systems are **easier to prototype**, but **harder to productionize**.*

ML systems are easier to build: **Workload consolidation, shared infrastructure, coding agents**

ML systems are harder to build: Vast **compatibility matrix, cross-layer interactions, large-scale behavior** is difficult to simulate and verify

TASK	2020	2026
Prototype new systems	*hard*	*easier*
Productionize it	*moderate*	*much harder*
Maintain compatibility	*moderate*	*very hard*
Achieve SOTA performance	*hard*	*extremely hard*

Systems for Future Intelligence

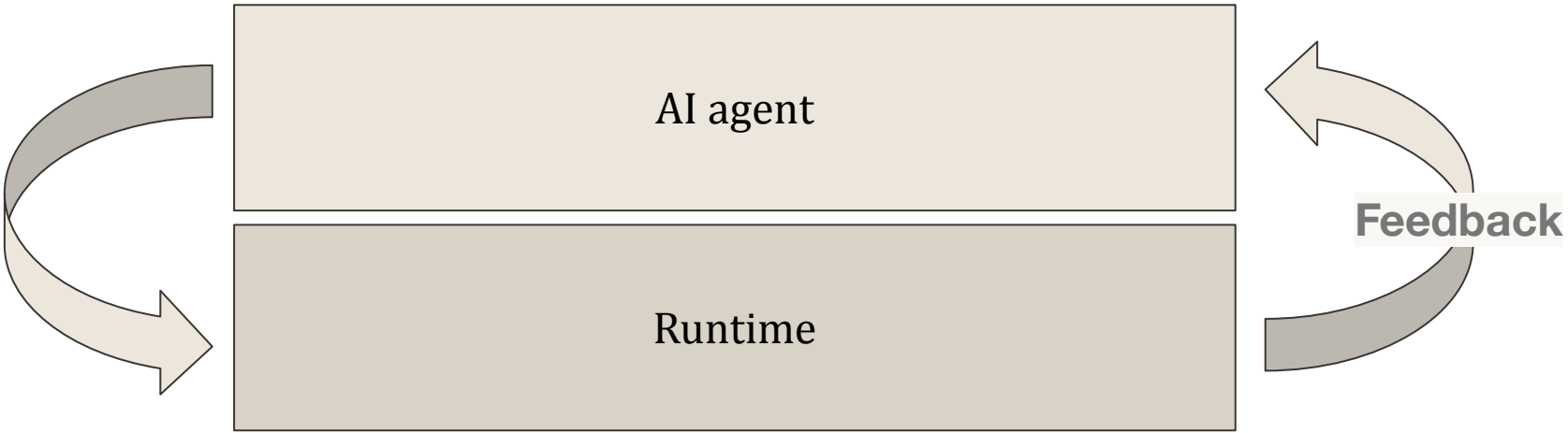
Our mission:

> *Enable the rapid evolution of efficient and resilient AI systems infrastructure* —
> through **abstractions** that
>
> > **adapt** by construction, **scale** from day one, and cut **across** the stack.

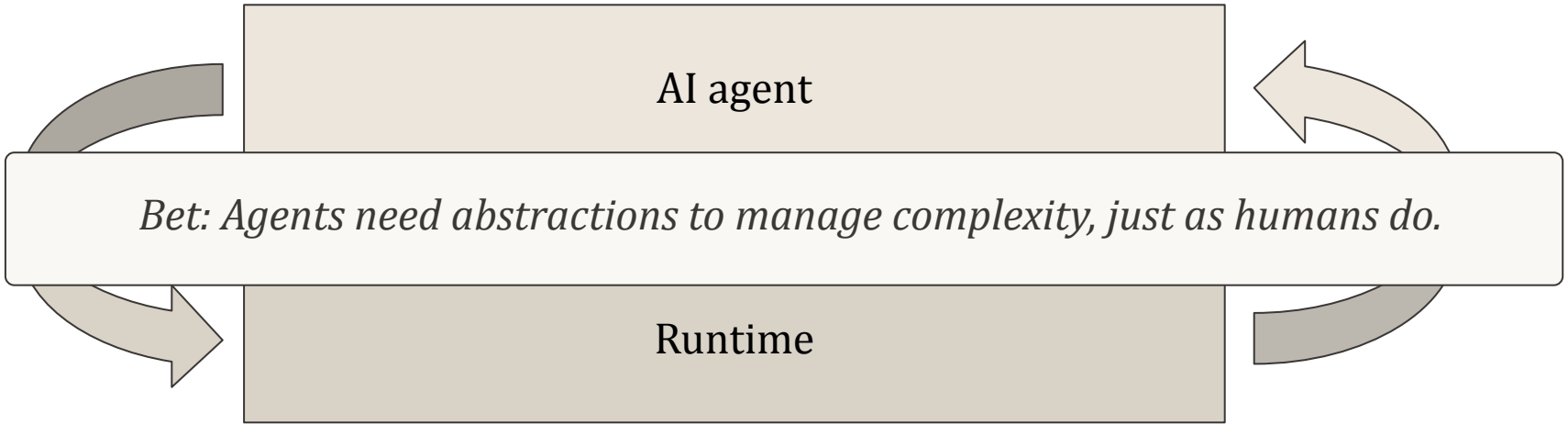
Research questions:

1. How should ML systems be (re)designed to make them amenable to AI-driven optimization?
2. What is needed for fully autonomous AI-driven optimization of systems?

The future systems stack?



The future systems stack?



Do we still need abstractions in AI-generated systems?

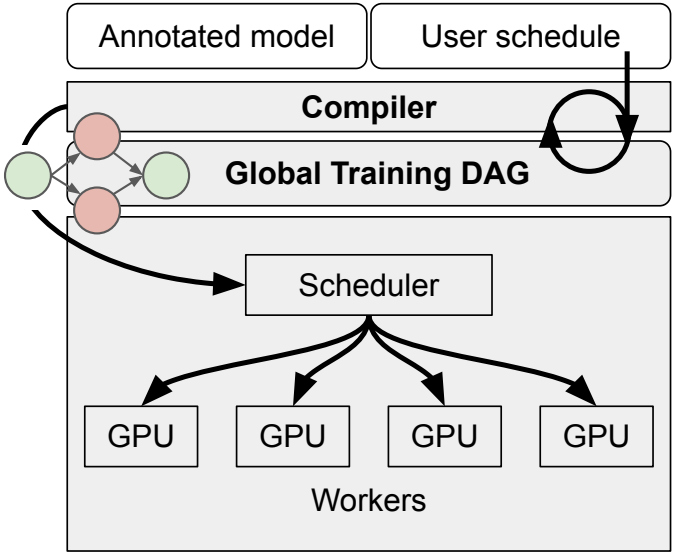
What agents need to build systems.

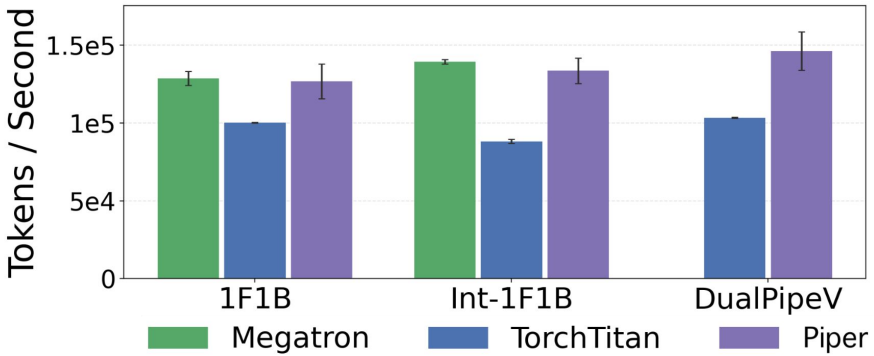
The same things human engineers need:

- **Observability:** Dense and timely feedback
- **Modular abstractions:** Reason about behavior without needing to run the whole system on each code change
- **The “right” abstraction:** Enough freedom that novel designs are possible

How should ML systems be (re)designed to make them amenable to AI-driven optimization?

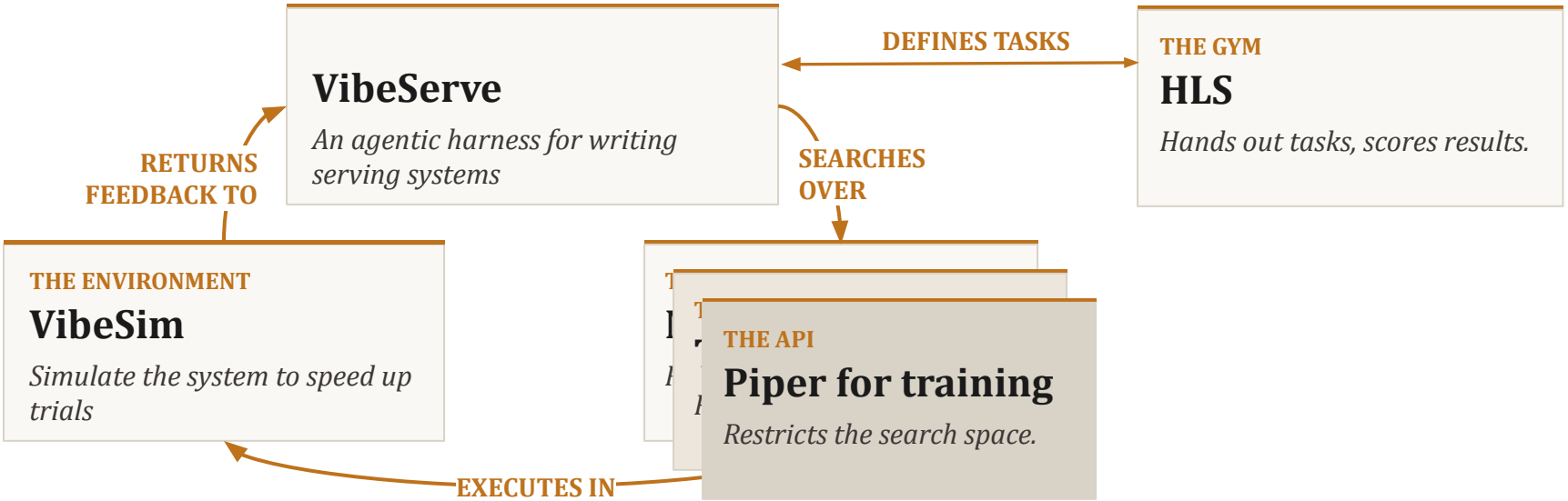
Piper: A programmable distributed training system



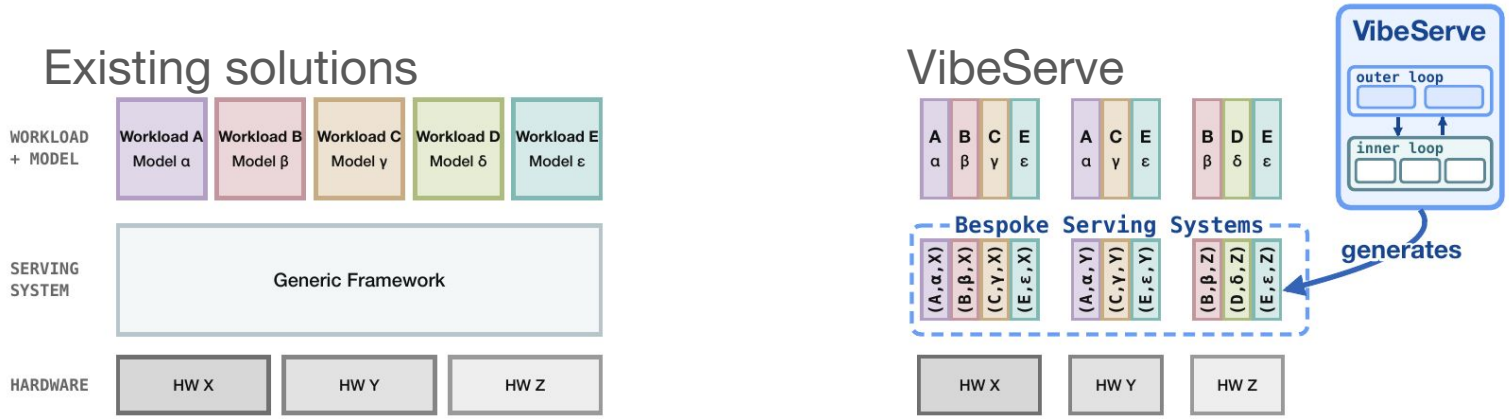


What is needed for fully autonomous AI-driven optimization of systems?

An agentic loop for generating serving systems



VibeServe

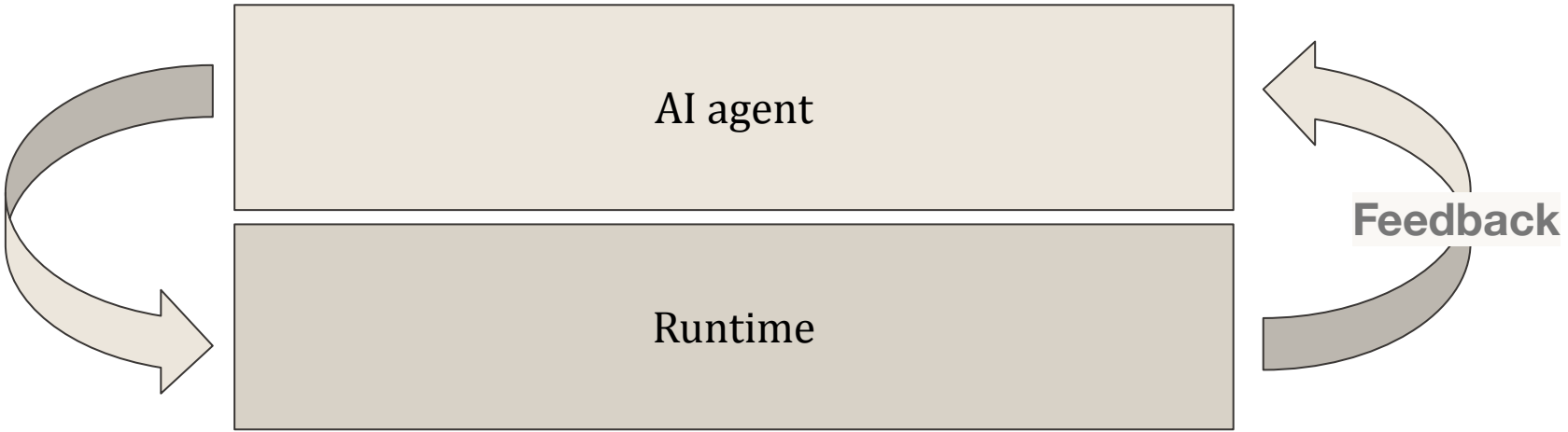


Key results:

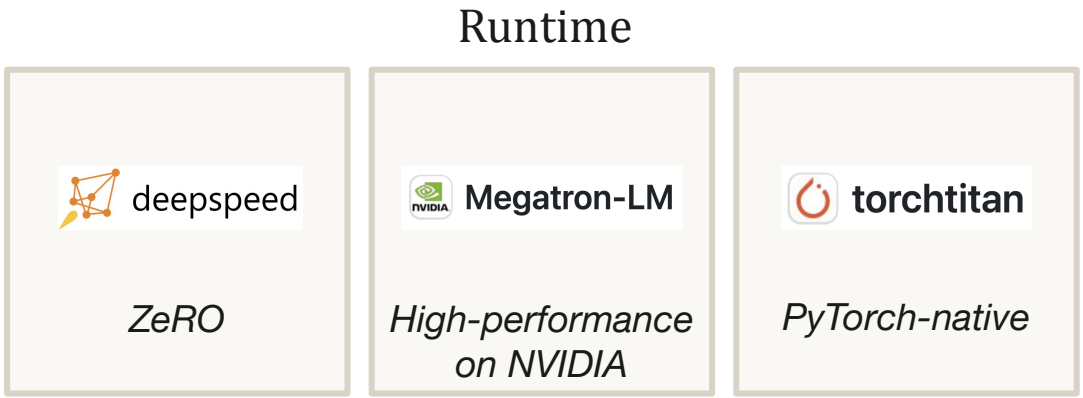
- **1-6x improvement** vs the next-best baseline, across a range of workload targets: Parity on traditional serving of a frontier open model, improvements across code editing, speech serving, prompt caching for hybrid attention, …
- **2.7x** improvement on pilot study on **AWS Trainium**

Piper
A Programmable Distributed Training System

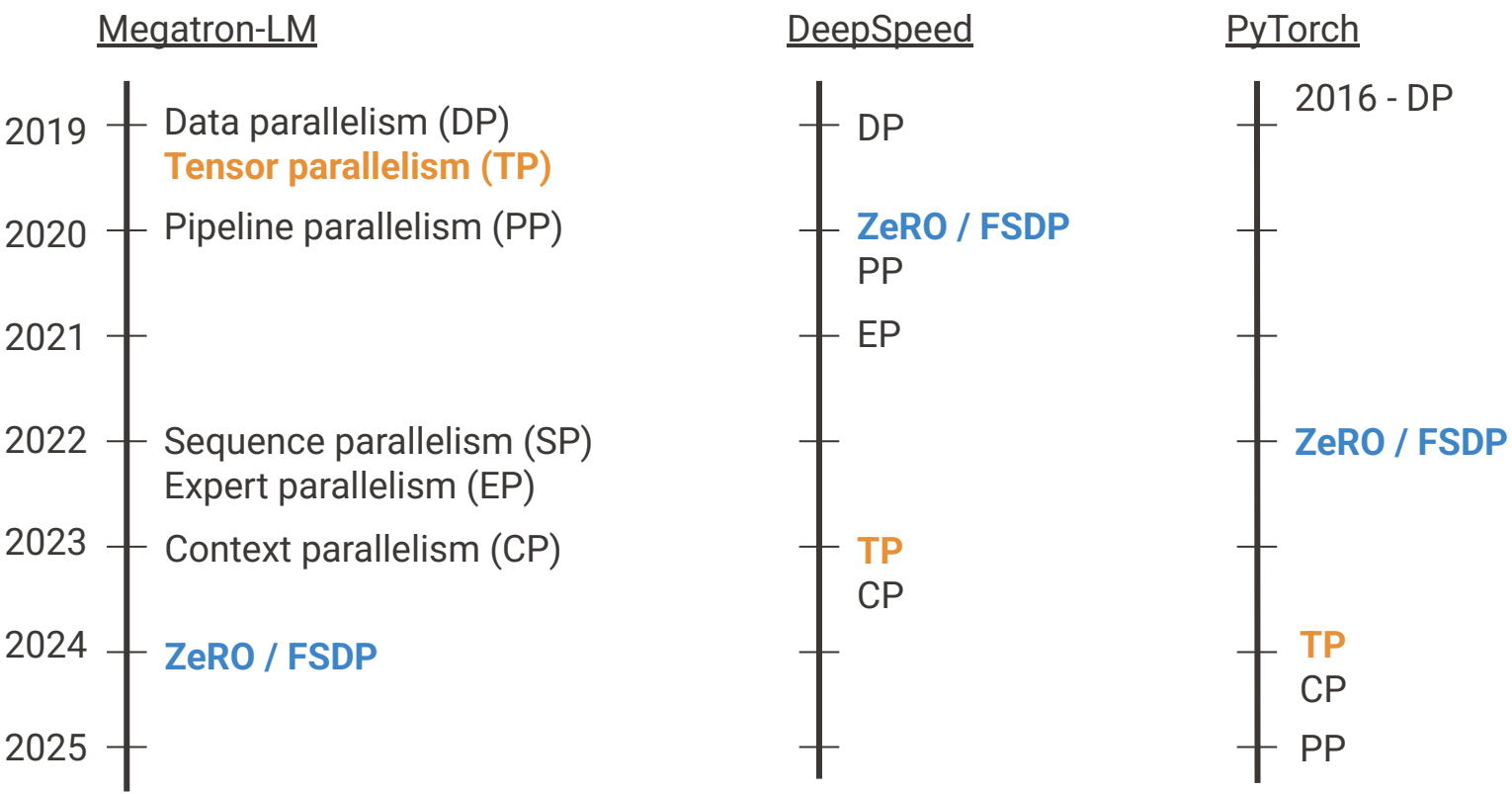
Distributed training today



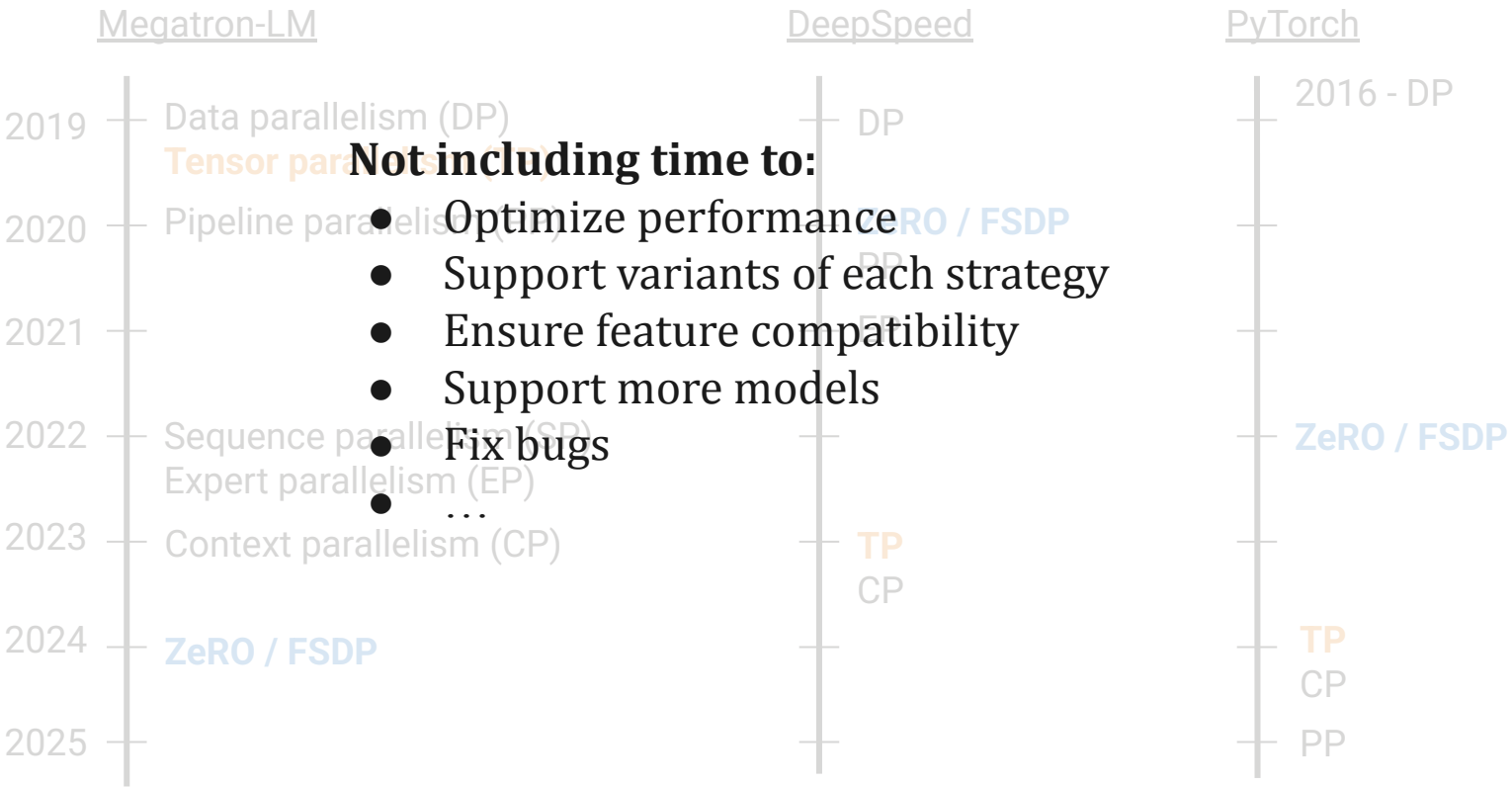
Distributed training today



Parallelism support in distributed training



Parallelism support in distributed training



Not including time to:

- Optimize performance
- Support variants of each strategy
- Ensure feature compatibility
- Support more models
- Fix bugs
- …

Existing systems offer limited extensibility

General-purpose frameworks achieve generality at the cost of flexibility


deepspeed

Megatron-LM


torchtitan

✔ Model-agnostic interface

✘ Dispatch operations for each parallelism dimension independently

→ Limited support for composing parallelism dimensions

→ Limited options for coordinating execution across dimensions

Existing systems offer limited extensibility

High-performance systems achieve their performance at the cost of flexibility

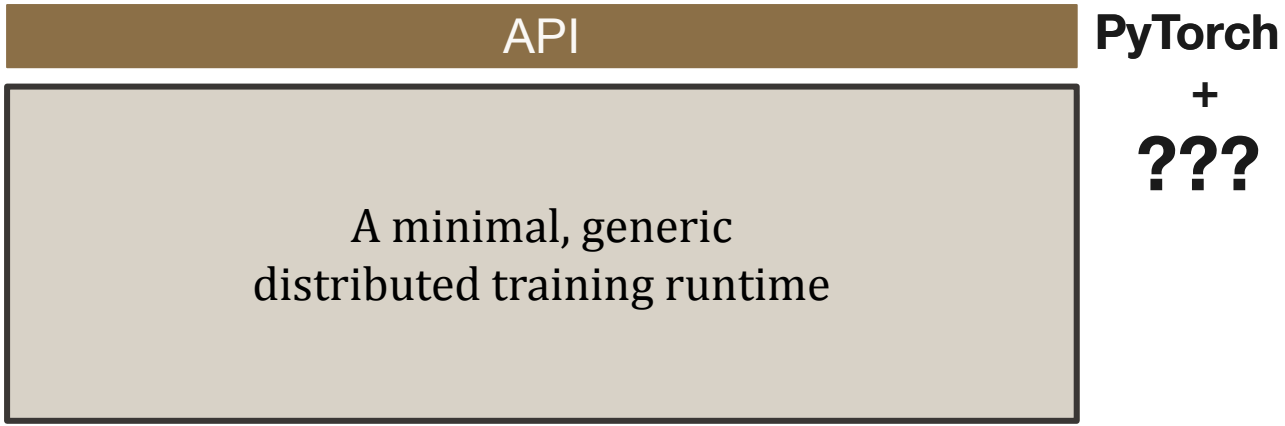
- ✔ High performance
- ✘ Human experts couple model architecture, parallelism strategy, system runtime, cluster topology

  → Hard to adapt to new strategies

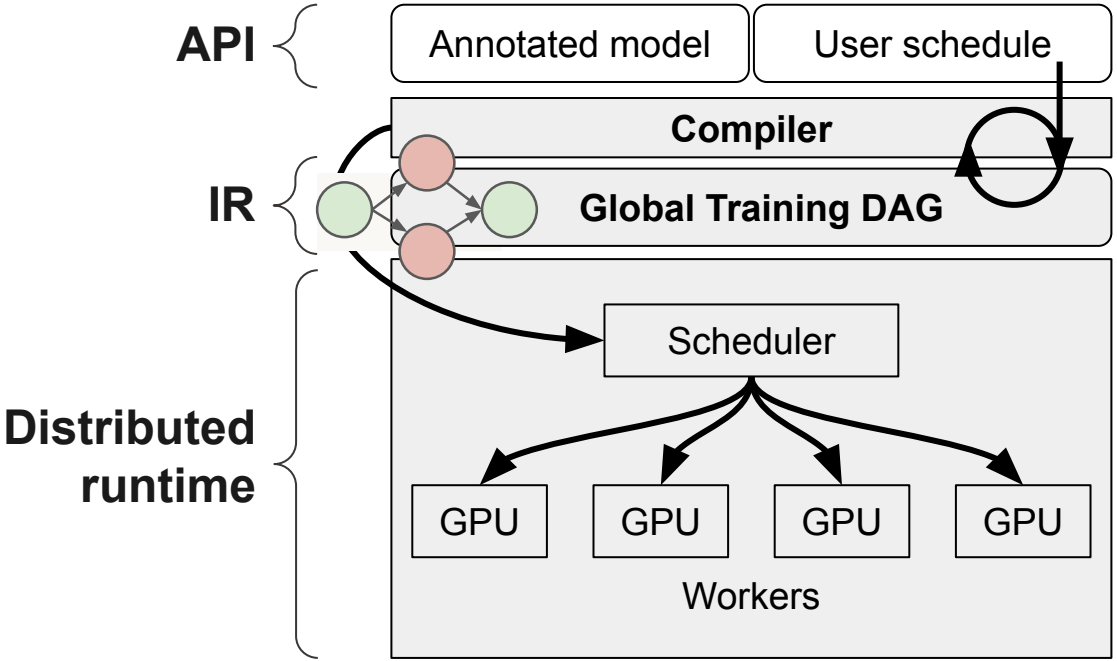
Existing systems offer limited extensibility

Problem: We need **extensibility** in distributed training systems.

Key idea: **Decouple** the execution strategy from the model implementation and the system runtime.



Piper overview



1. User schedule repeatedly transforms the IR:
   - Specify the high-level strategy
   - Partially specify the low-level strategy

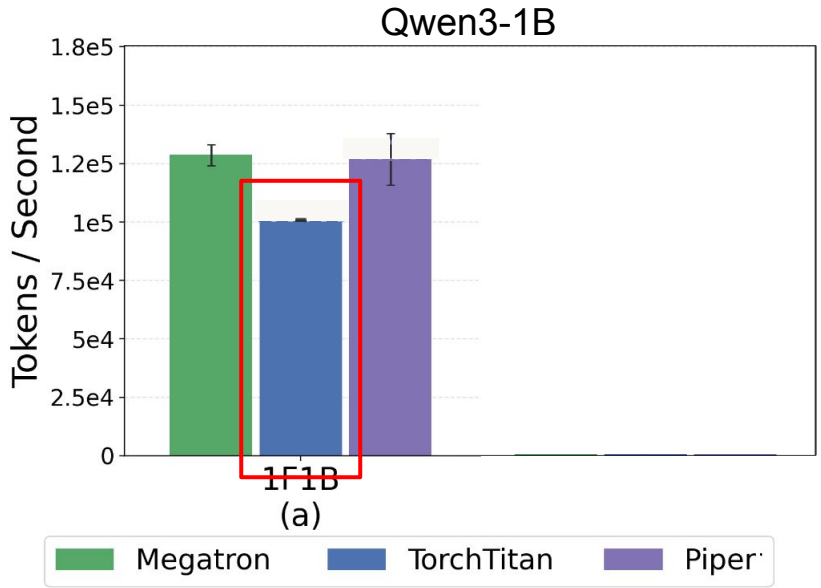
2. Compiler completes the low-level strategy.

3. Runtime executes the low-level strategy.
   - Asynchronous, single-controller

Evaluation: Composing pipeline (PP) and expert (EP) parallelism

1. PP: 1-forward-1-backward (1F1B) vs Interleaved 1F1B
    a. Interleaved reduces pipeline bubbles but adds communication
2. PP x EP: Interleaved 1F1B vs DualPipeV
    a. DualPipeV interleaves execution across some microbatches

Evaluation: Common PP strategies

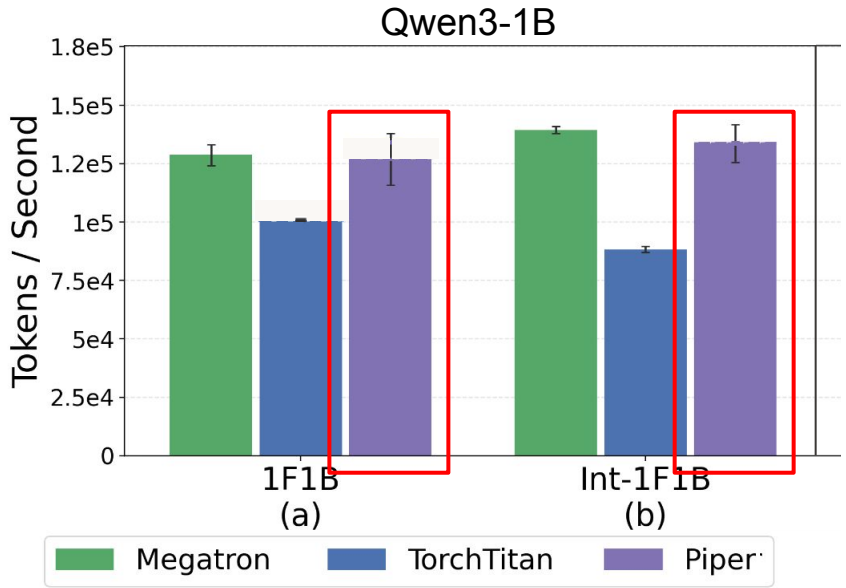


- PP 8 x DP/EP 4
- **TorchTitan: CUDA memory allocator delays hurt performance.**

Setup: 4 AWS EC2 NVIDIA 8xA100 nodes, NVLink + EFA

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

Evaluation: Common PP strategies

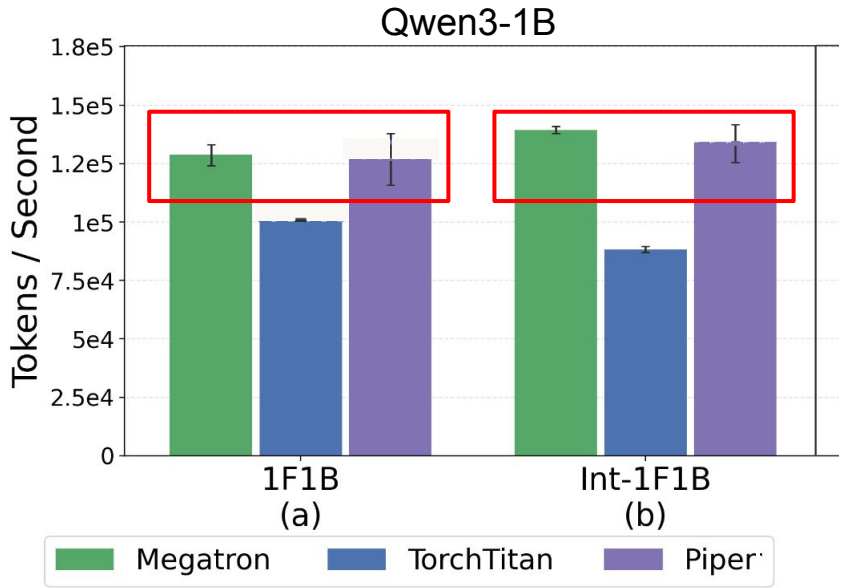


- TorchTitan: CUDA memory allocator delays hurt performance.
- **Piper: Int-1F1B is 5% better than 1F1B**

Setup: 4 AWS EC2 NVIDIA 8xA100 nodes, NVLink + EFA

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

Evaluation: Common PP strategies

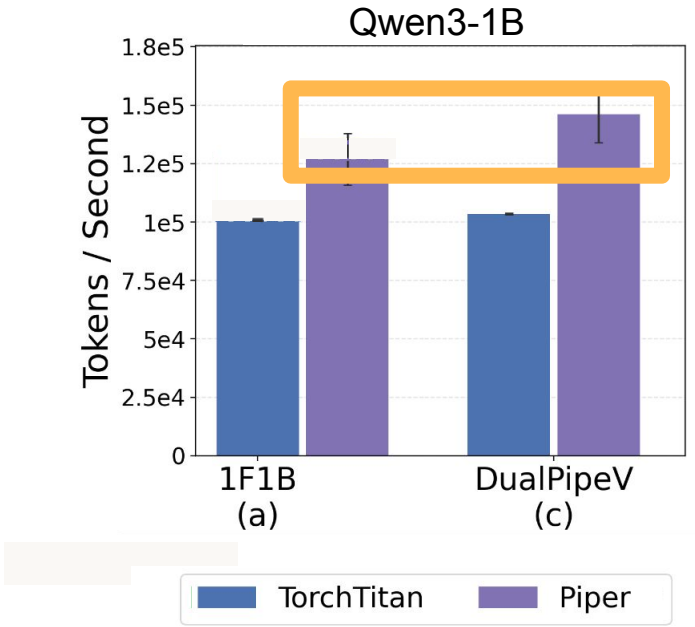


- TorchTitan: CUDA memory allocator delays hurt performance.
- Piper: Int-1F1B is 5% better than 1F1B
- **Piper's performance is comparable to Megatron's**
  - Close the gap with Megatron's custom kernels

Setup: 4 AWS EC2 NVIDIA 8xA100 nodes, NVLink + EFA

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

Evaluation: PP x EP

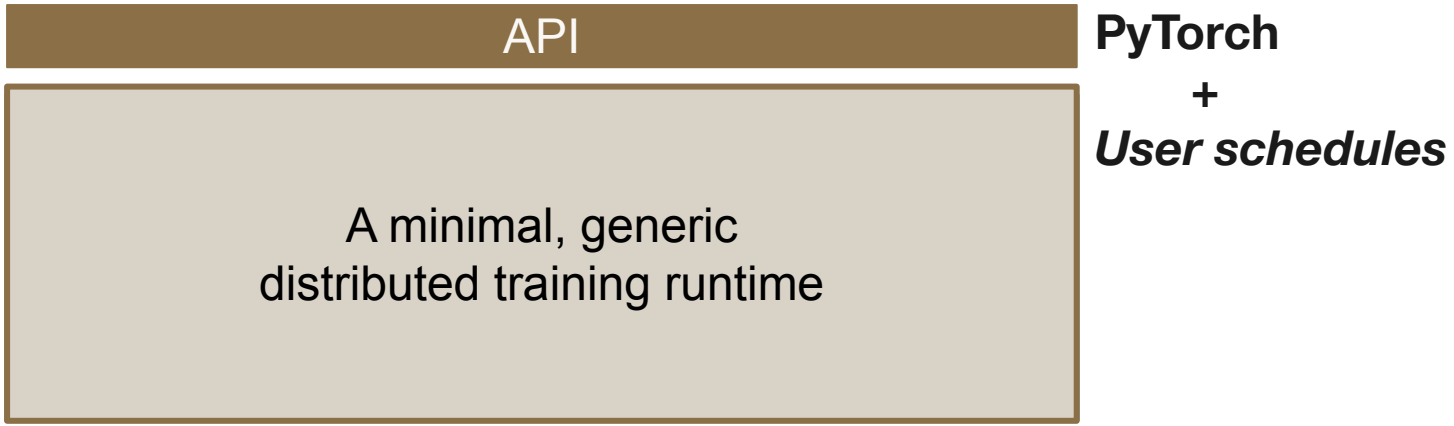


- TorchTitan: 3% improvement
  - Poor synchronization between compute and communication streams
- **Piper: 13% improvement**
  - Global scheduling across compute and communication ops
  - Further improvements ongoing!

Piper

Key idea: Decouple the execution strategy from the model implementation and the system runtime.

Theme: Achieve higher performance via greater flexibility.



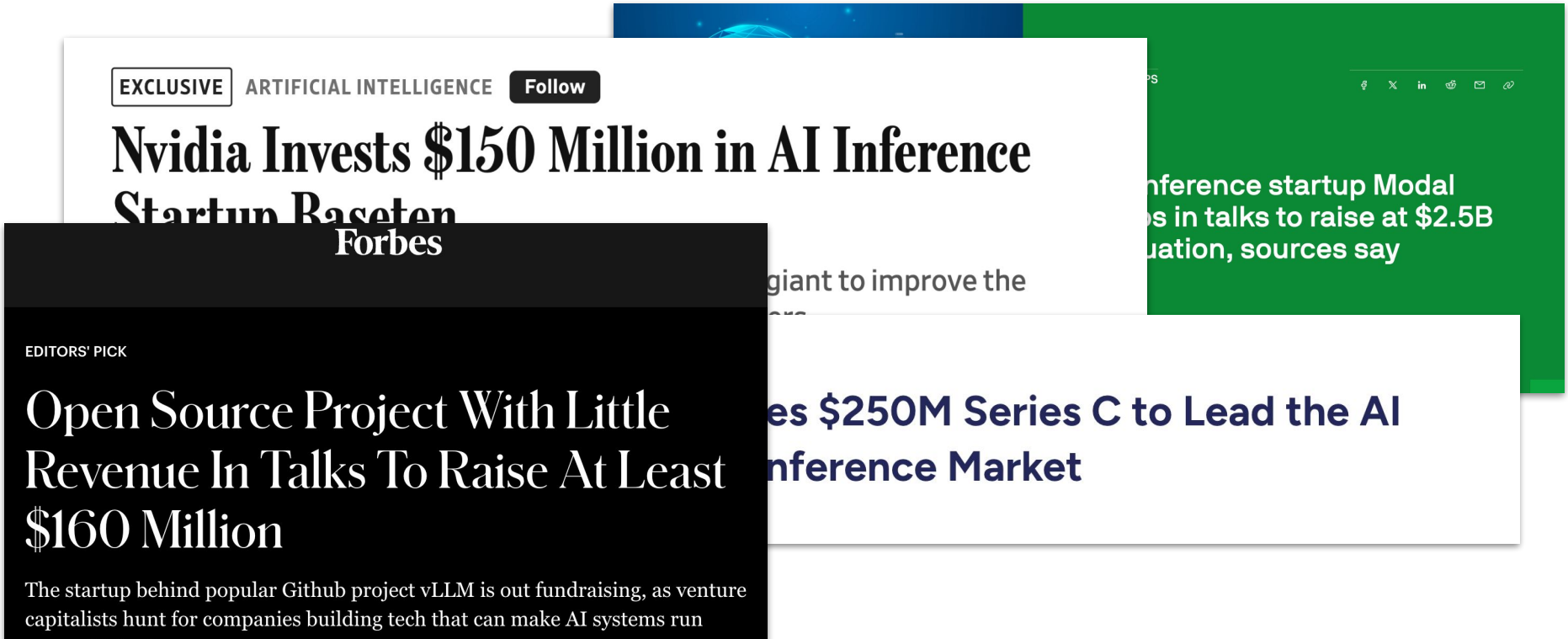
Piper: Next steps

- **Dynamic scheduling** to reduce overheads from
  - Performance interference
  - Memory pressure
  - …
- **Dynamic fault tolerance** strategies
- Combining VibeServe's agent with Piper's API and implementation: **Agentic search** over distributed training strategies

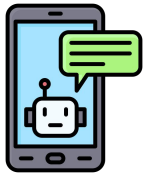
VibeServe

Can agents build bespoke LLM serving systems?

LLM Inference is a big economy now



The serving stack is complex



OPENCLAW

Qwen

Olmo 3

deepseek

LLM Serving Systems

vLLM

SGL

NVIDIA

AMD

arm

One size fits all approach has limitations

Combinatorial explosion of usage scenarios, each has different performance optimization opportunities

Model
- Multimodal models
- Hybrid attention/SSM models
- …

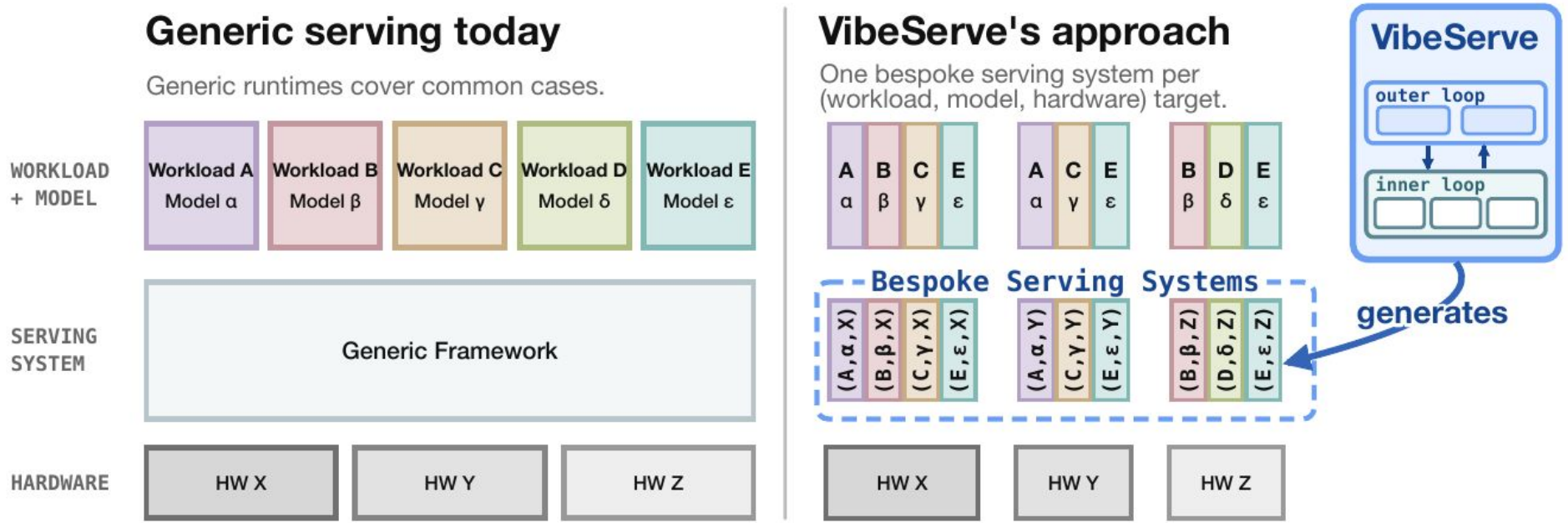
Hardware
- NVIDIA
- Macbook
- Trainium
- AMD
- TPU
- CPU
- …

Workload
- Chatbot
- RAG
- Agent
- Multimodal
- …

Rethinking how we build infrastructure

Coding agents have dramatically reduced the cost of implementing new software

→ Opportunity for *bespoke* systems to hyper-optimize for each use case



Challenges in long-horizon tasks

Goal drift

Feedback local maxima

Feedback signal variance

Reward hacking

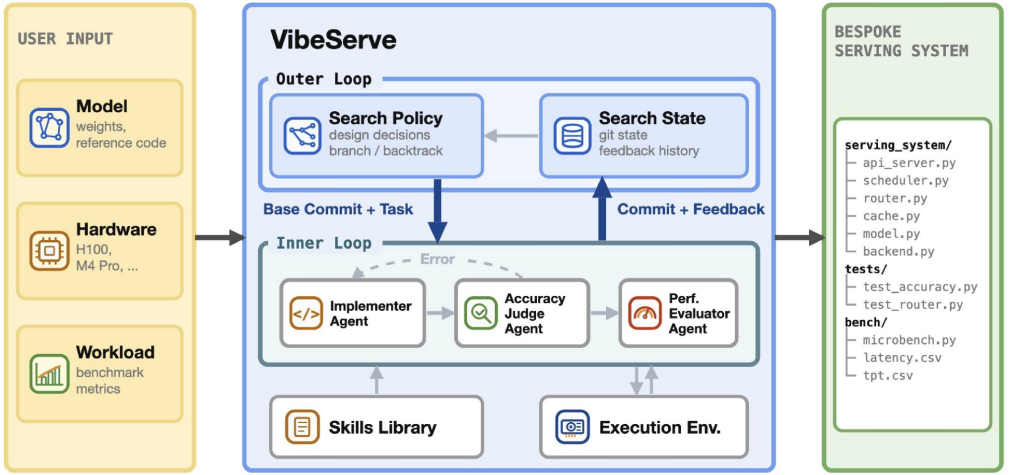
Hallucinations

…

Existing agents optimize components in an existing system.

VibeServe's goal is to generate and optimize systems end-to-end.

VibeServe: an agent that writes its own serving stack.



Frontier coding agents can patch a kernel. They cannot, on their own, design and build a serving system that holds together under load.

VibeServe splits the job across two loops at different scopes

LOOP A · HIGH-LEVEL

Plan the system. Pick what to optimize next.

Distinguishes "this idea was wrong" from "the implementation had a bug".

LOOP B · LOW-LEVEL

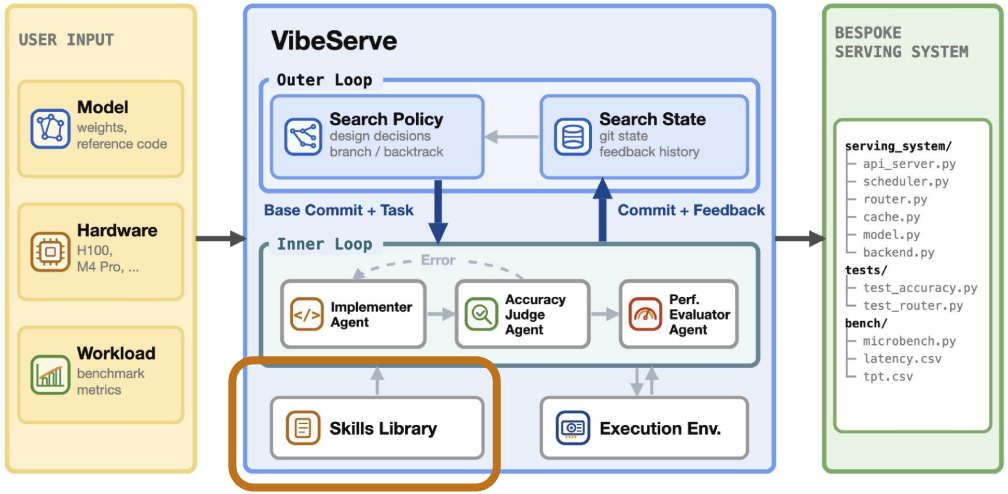
Iterate one feature until it is correct.

Has real hardware, profilers, and a curated library of serving techniques.

Skills

```
skills/serving-systems/
├── SKILL.md
├── models/
│   ├── text-dense/SKILL.md
│   ├── text-moe/SKILL.md
│   ├── ssm-hybrid/SKILL.md
│   ├── vision-language/SKILL.md
│   ├── speech-language/SKILL.md
│   ├── speech-generation/SKILL.md
│   ├── image-generation/SKILL.md
│   ├── video-generation/SKILL.md
│   └── omni-multimodal/SKILL.md
│
├── algorithms/
│   ├── attention-variants/SKILL.md
│   ├── async-scheduling/SKILL.md
│   ├── batched-sampling/SKILL.md
│   ├── chunked-prefill/SKILL.md
│   ├── continuous-batching/SKILL.md
│   ├── disaggregated-serving/SKILL.md
│   ├──
heterogeneous-kv-cache/SKILL.md
│   ├── moe-routing-dispatch/SKILL.md
│   ├── paged-attention/SKILL.md
│   ├── parallelism/SKILL.md
│   ├── quantization-schemes/SKILL.md
│   ├── radix-prefix-caching/SKILL.md
│   ├── speculative-decoding/SKILL.md
│   └── structured-output/SKILL.md
│
├── frameworks/
│   ├── pytorch/SKILL.md
│   ├── triton/SKILL.md
│   └── mlx/SKILL.md
│
├── backends/
│   ├── flashinfer/SKILL.md
│   │   └── references/*.md
│   ├── flashattention/SKILL.md
│   │   └── references/*.md
│   ├── triton-kernels/SKILL.md
│   └── cuda-graph/SKILL.md
│       └── references/*.md
│
├── hardware/
│   ├── nvidia/SKILL.md
│   ├── amd-mi300/SKILL.md
│   └── apple-silicon/SKILL.md
│
├── engines/
│   ├── vllm/SKILL.md
│   ├── sglang/SKILL.md
│   └── trtllm/SKILL.md
│
├── tooling/
│   ├── fastapi-serving/SKILL.md
│   │   └── references/*.md
│   ├── openai-api/SKILL.md
│   ├── accuracy-checker/SKILL.md
│   │   └── references/*.md
│   ├──
serving-benchmark/SKILL.md
│   ├── profiler/SKILL.md
│   ├── io-handling/SKILL.md
│   └── lora-serving/SKILL.md
│
└── repos/
    ├── vllm/
    ├── sglang/
    └── TensorRT-LLM/
```

Extensible knowledge base about serving system design distilled from papers and human-built systems



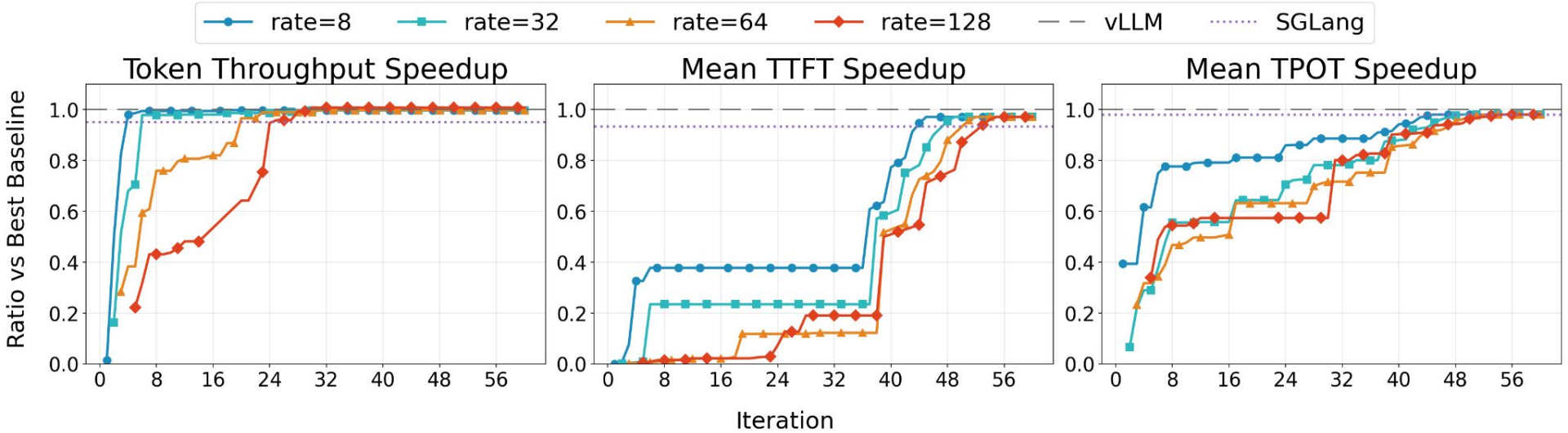
VIBESERVE · EVALUATION

Case study A: standard LLM serving

Llama-3.1-8B on H100 GPU

Standard scenario, existing systems are heavily optimized

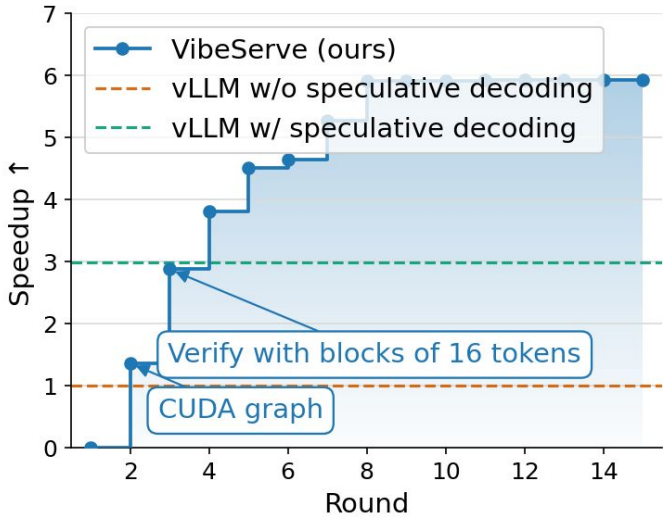
VibeServe matches vLLM/SGLang after ~50 iterations



VIBESERVE · EVALUATION

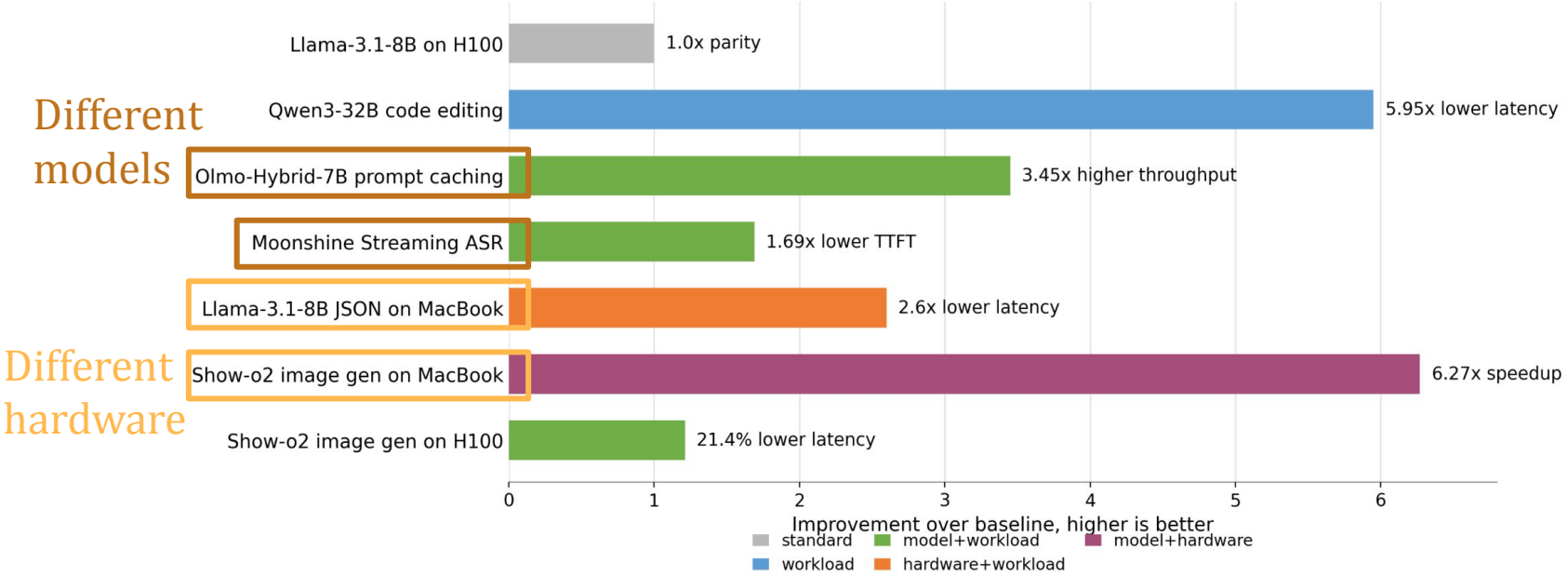
Case study B: Code editing with predicted outputs

- Qwen3-32B on H100
- Some applications like code editing benefit from speculative decoding based on predicted-outputs, not draft models
  - Cursor is using it; OpenAI/Fireworks API support it
- But mainstream systems only support draft model-based speculation
- VibeServe outperforms vLLM by ~6x



VIBESERVE · EVALUATION

VibeServe achieves 1-6x improvement vs the next-best baseline



How do we push VibeServe even further?

Workload scaling: Multi-node inference

→ *VibeSim* simulates inference systems to reduce time and cost to run a trial

Harness design: Is it better to modify an existing system or generate a system from scratch? Can agents discover genuinely new ideas, or only recombine existing ones?

→ Searching over APIs like *M** vs system generation from skills only

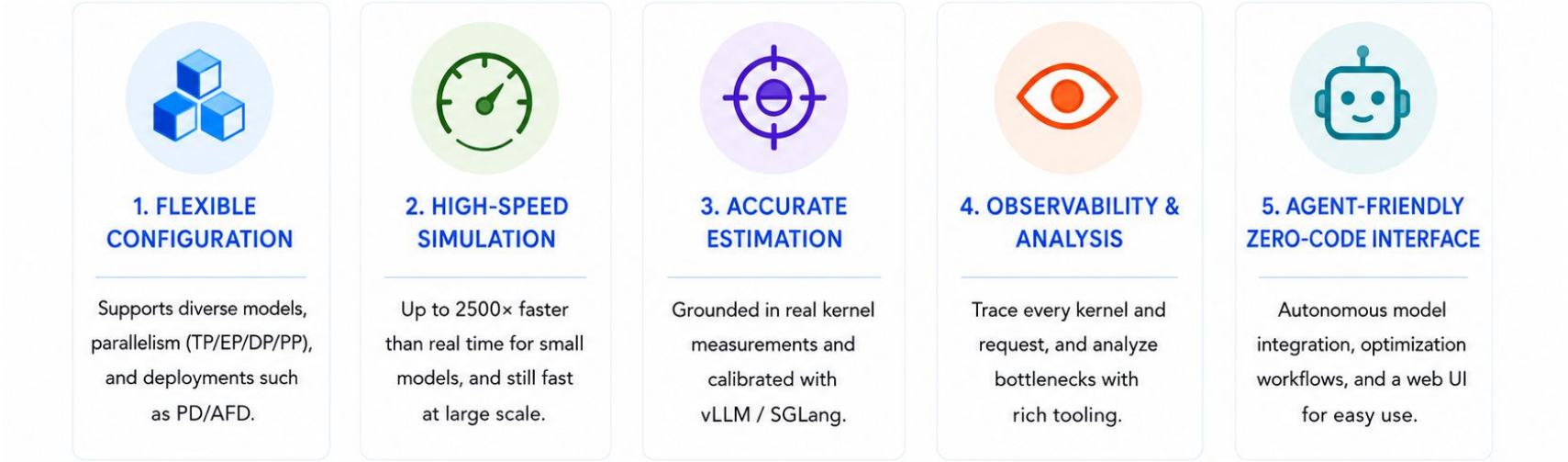
Task specification: How do we measure whether agents are becoming better at this task over time? How do we measure correctness?

→ *HLS:* A benchmark for agentic development of ML systems

VibeSim

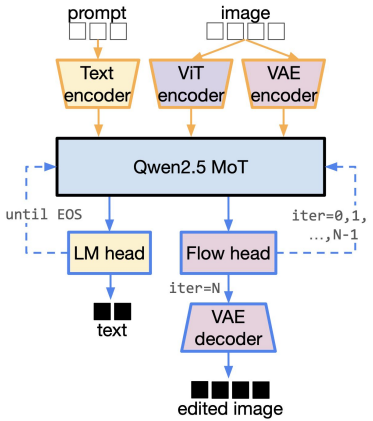
Premise: Future systems engineering will be bottlenecked on time to test new ideas, not time to develop them

VibeSim: A flexible simulator for LLM inference at scale.

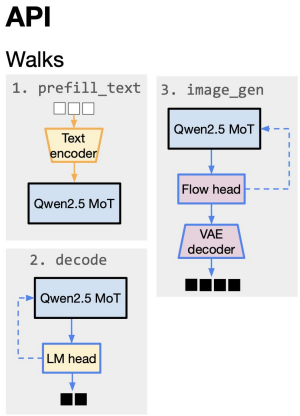


M: serving multimodal models as walks on a graph.*

- Models are composites now — encoders, backbones, flow heads, codecs.
- A model is a directed graph of those components.
- A request is a named *walk* over the graph.
- **The system chooses how to execute the *Walk Graph***



BAGEL –
A UNIFIED MULTMODAL MODEL



M: Evidence the API is well-chosen.*

End-to-end gains come not from new kernels but:

- Matching system execution to the model dataflow
- Flexible placement strategies
- First-class support for loops → apply optimizations like CUDA graphs uniformly to loop bodies

API is applicable to more than just multimodal models!

BAGEL · text-to-image, p50 latency

1.43× lower than vLLM-Omni (3-GPU CFG-parallel).

BAGEL · image understanding, B=8

11.5× lower TTFT; 12.1× higher throughput.

Qwen3-Omni · Seed-TTS, B=32

p95 RTF 0.45 versus vLLM-Omni at 1.57 — the baseline misses real-time.

V-JEPA 2 · AC rollout, H=30

12.5× faster than Meta's reference loop.

Comparable systems: vLLM-Omni (BAGEL, Qwen3-Omni), VoxServe (Orpheus), Meta vjepa2 (V-JEPA 2). H100 / H200 single-node configurations.

Humanity's Last System (HLS)

A fully-customizable evaluation testbed for long-horizon agentic capabilities in developing ML inference systems

Challenge: Ensuring correctness. End tasks are expensive and noisy but token-level verifiers can also produce false positives/negatives.

Task	Model Architecture	Modality	Hardware	Workload
Generic API serving	Qwen 3.6 27B (linear attn hybrid)	Text	1xH200	Generic API serving based on production trace (Alibaba Bailian)
Multimodal Serving	Qwen 3 Omni 30B (thinker-talker)	Omni	4xH200	Mixed input / output modality (text, vision, speech)
Long context agentic coding	Kimi K3 (novel hybrid KDA, MLA…)	Text	2x8xH200 with Infiniband	1M context agentic coding induced by FrontierSWE

VibeServe to VibeSys

Bet: most infrastructure code will be agent-generated

OK, maybe there will always be some human collaboration

Thank you! Questions?

Email: smwang@cs.washington.edu
Lab website and blog: syfi.cs.washington.edu

BACKUP

Piper allows the user to control the high- and low-level execution strategies

High-level execution strategy:

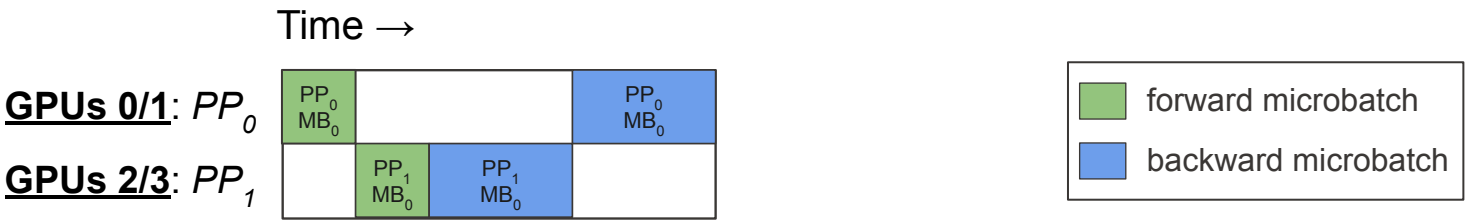
- How is each model component sharded/replicated?
- How should data flow through the pipeline?

Low-level execution strategy:

- How are tensor operators scheduled onto GPUs?

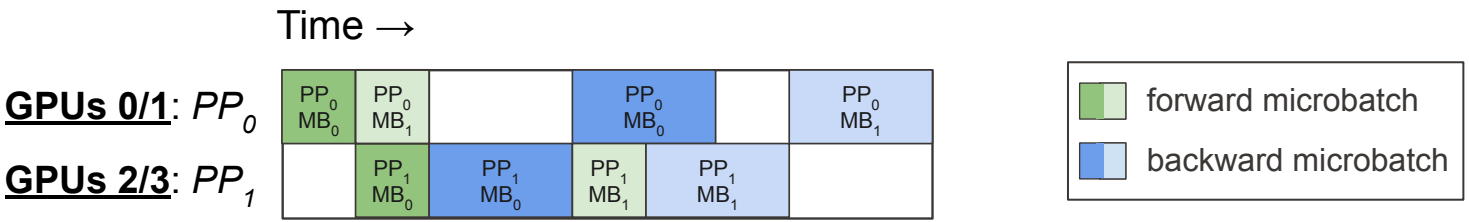
High-level parallelism strategy: Pipeline-parallel schedules

PP schedules describe how data flows through the pipeline.



High-level parallelism strategy: Pipeline-parallel schedules

PP schedules use microbatching to improve hardware utilization.

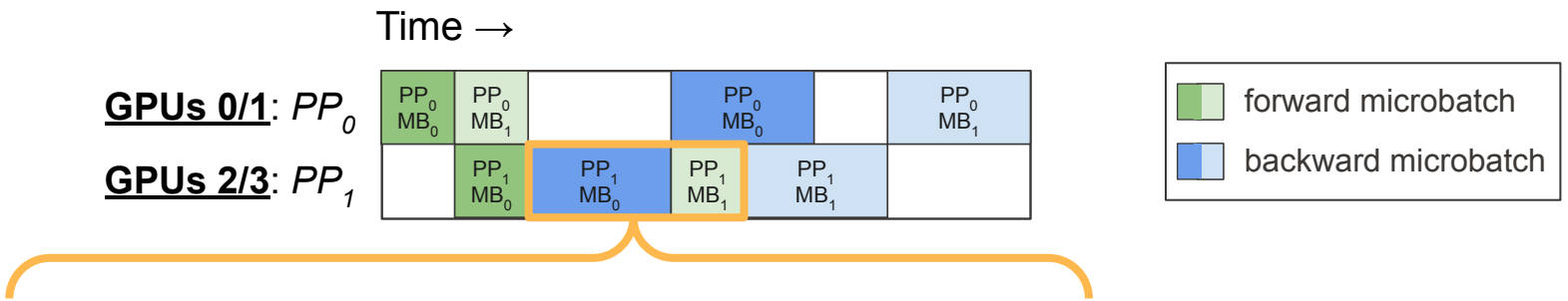


High-level parallelism strategy: Pipeline-parallel schedules

PP across layers and EP within layers.

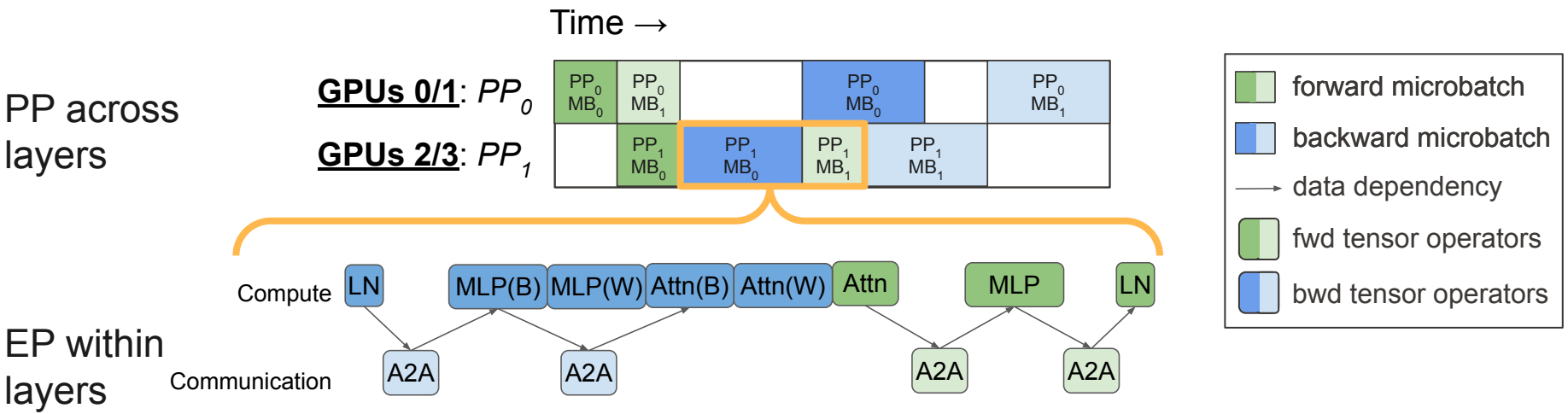
PP across layers

EP within layers



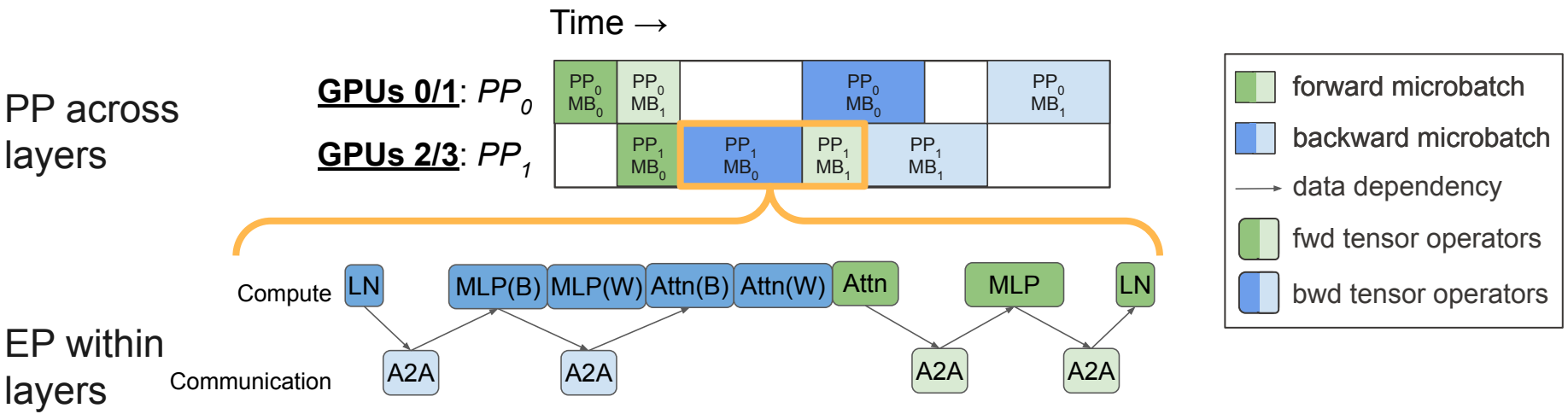
High-level parallelism strategy: Pipeline-parallel schedules

EP adds critical-path all-to-all communication.



High-level parallelism strategy: Pipeline-parallel schedules

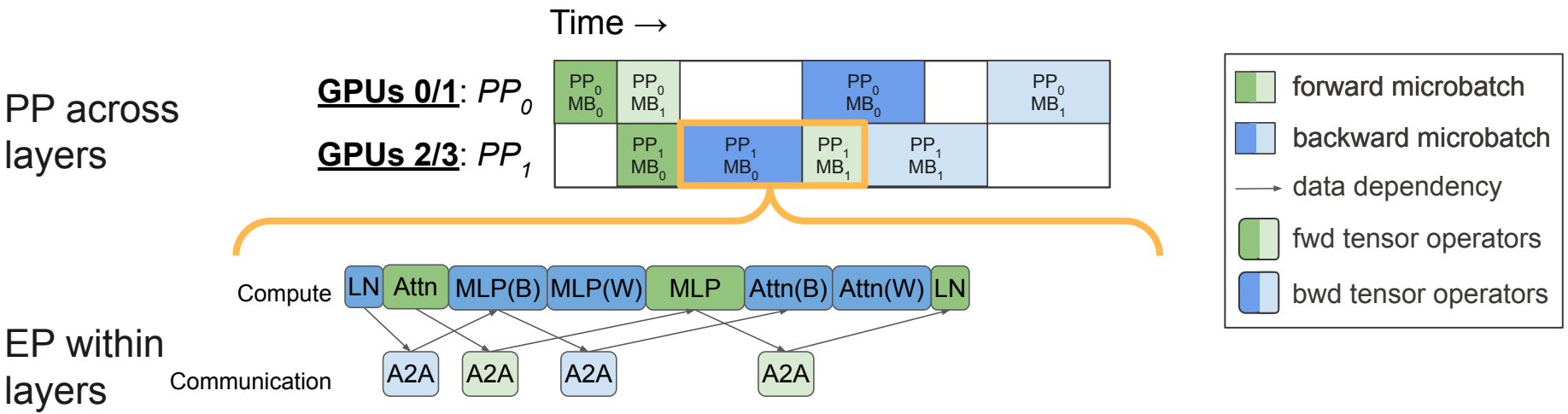
EP adds critical-path all-to-all communication.



General-purpose systems: sequential execution

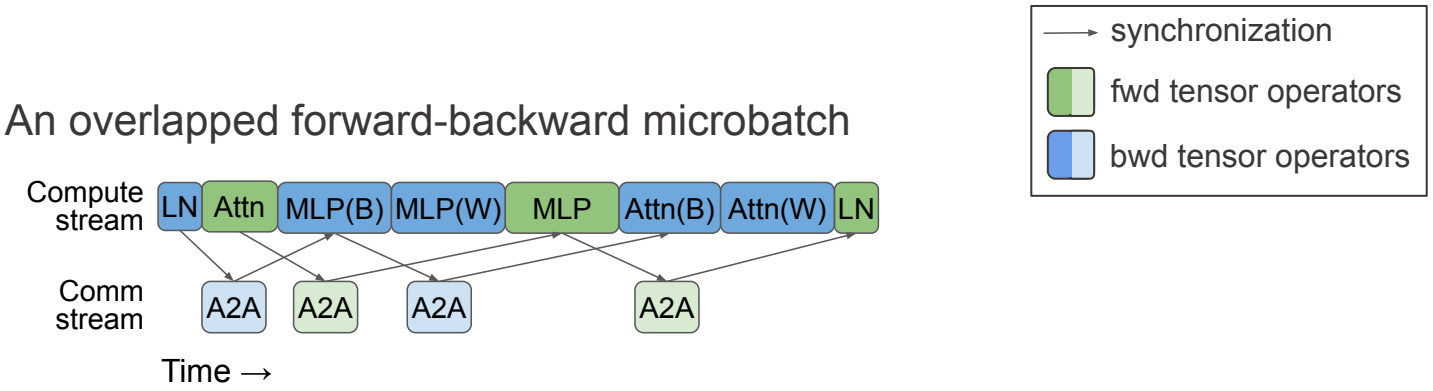
High-level parallelism strategy: Pipeline-parallel schedules

DualPipe-like schedule interleaves microbatches to hide A2A latency.



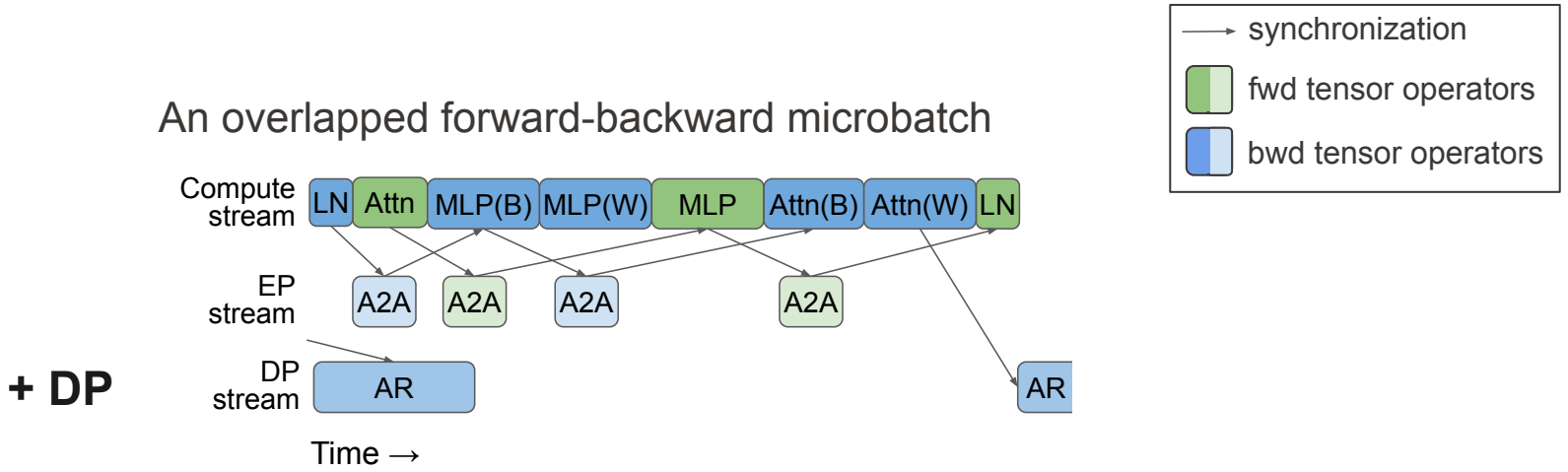
High-performance systems: interleaved execution

Low-level execution strategy: scheduling tensor operators



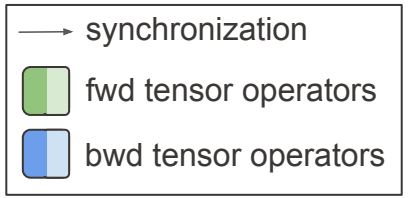
The schedule is simple when only EP is involved.

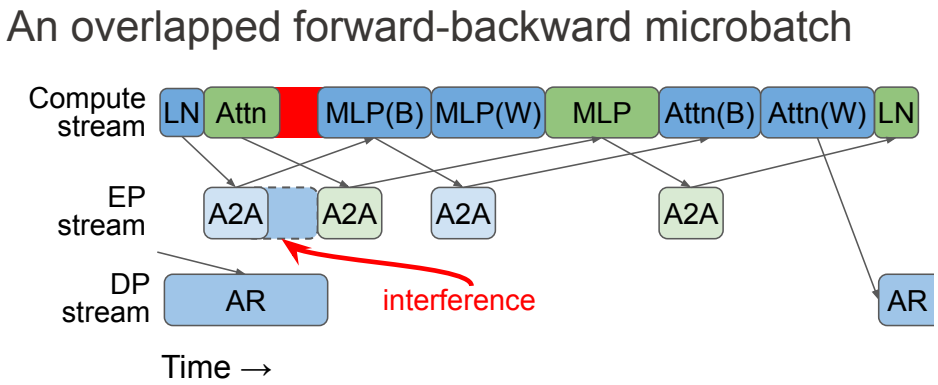
Control over low-level execution strategy



Current frameworks eagerly dispatch allreduce on a separate stream.

Control over low-level execution strategy

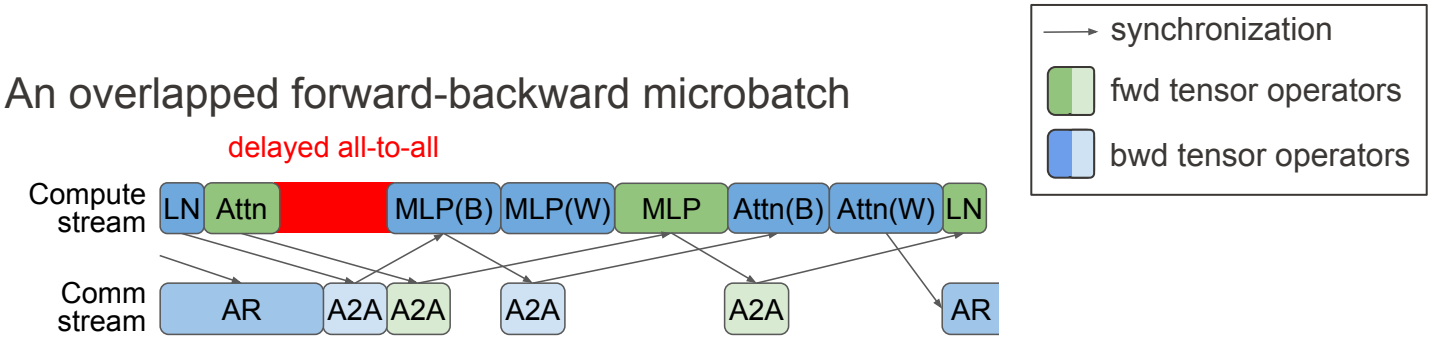




Contention on network bandwidth leads to interference.

Hard to predict interference when allreduce is eagerly dispatched.

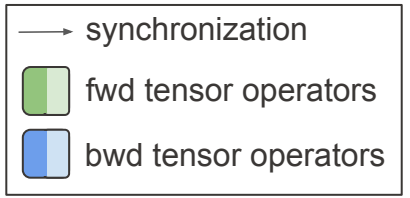
Control over low-level execution strategy

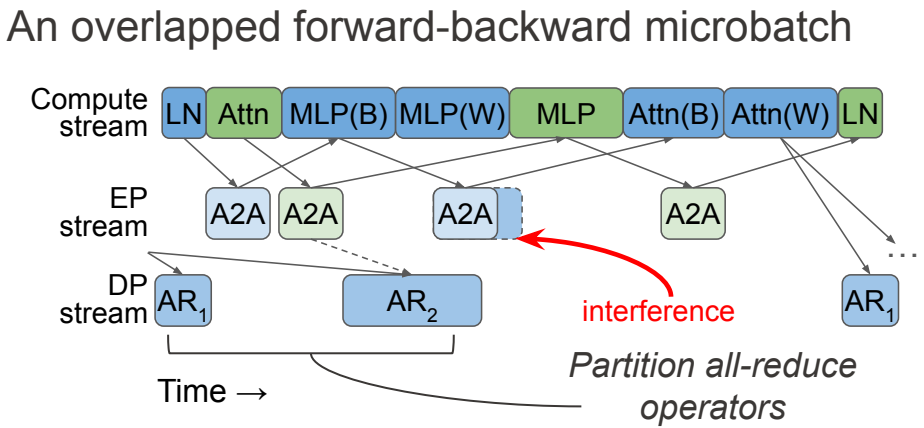


Executing on the same stream could help,
could also delay the critical path.

Related issues: pauses from out-of-memory, stragglers from hardware degradation or token imbalance, …

Control over low-level execution strategy





Partitioning and ordering allreduce can reduce interference.

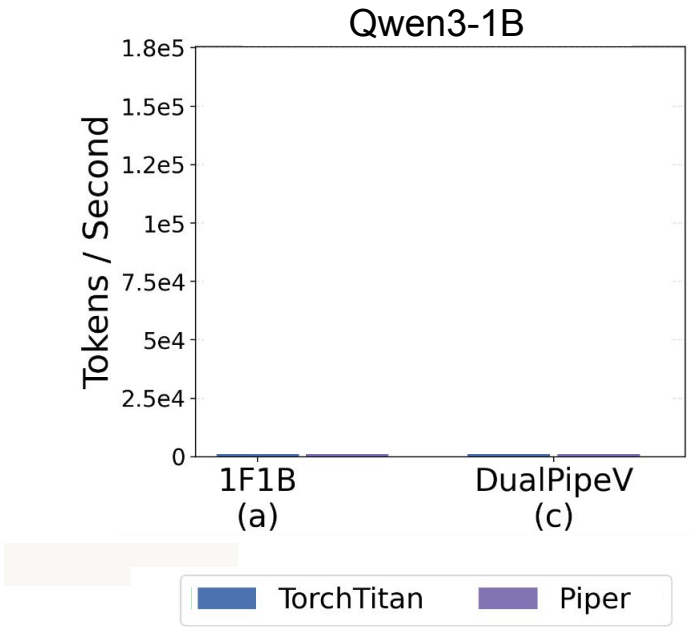
Hard to predict overall effect, e.g. partitioning can reduce efficiency.

Evaluation: Composing pipeline and expert parallelism

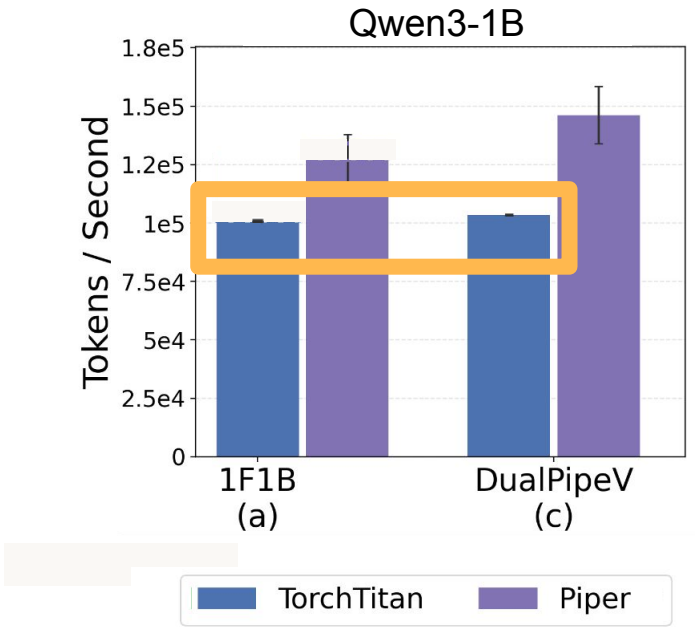
How well do current systems support DualPipe-style A2A overlapping?

- DualPipe-style schedule should perform better than 1F1B and Int-1F1B
- TorchTitan and Piper support DualPipe-like schedules
- Megatron and DeepSpeed do not support DualPipe-style overlapping

Evaluation: PP x EP

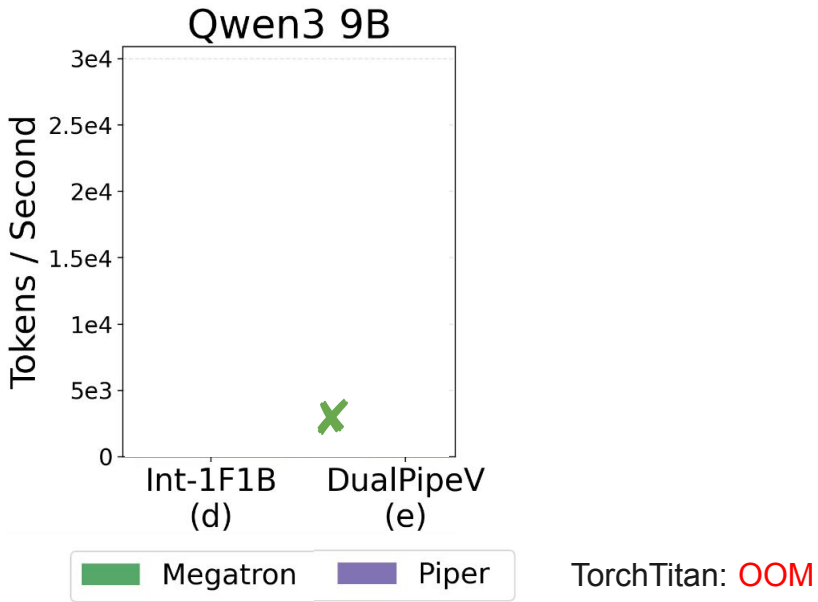


Evaluation: PP x EP



- **TorchTitan: 3% improvement**
  - Poor synchronization between compute and communication streams

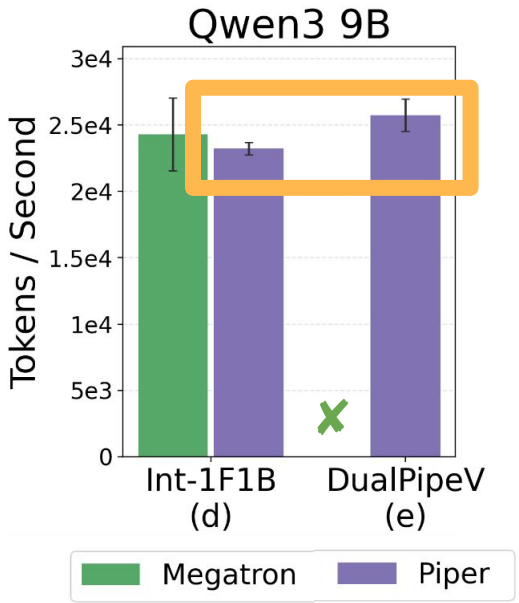
Evaluation: PP x EP



TorchTitan: OOM

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

Evaluation: PP x EP

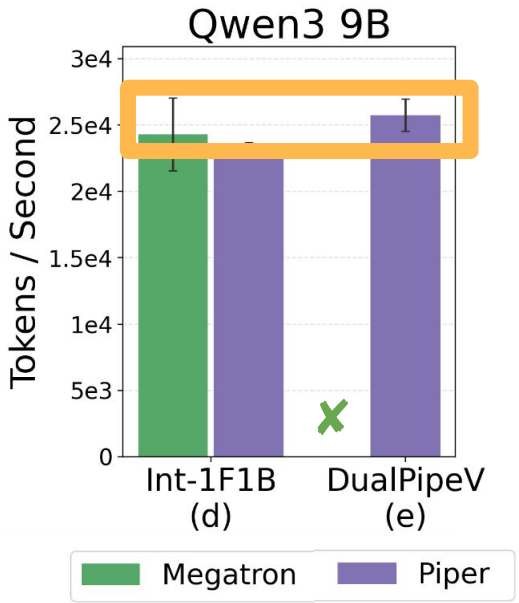


- Piper: DualPipeV is 10% better than Int-1F1B

TorchTitan: OOM

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

Evaluation: PP x EP



- Piper: DualPipeV is 10% better than Int-1F1B
- 6% better than Megatron-Int-1F1B
  - More improvements to come!

TorchTitan: OOM

**Megatron: requires PP to be the outer dimension – route intra-node traffic over PCIe to simulate lower DP bandwidth*

VIBESERVE · INPUTS

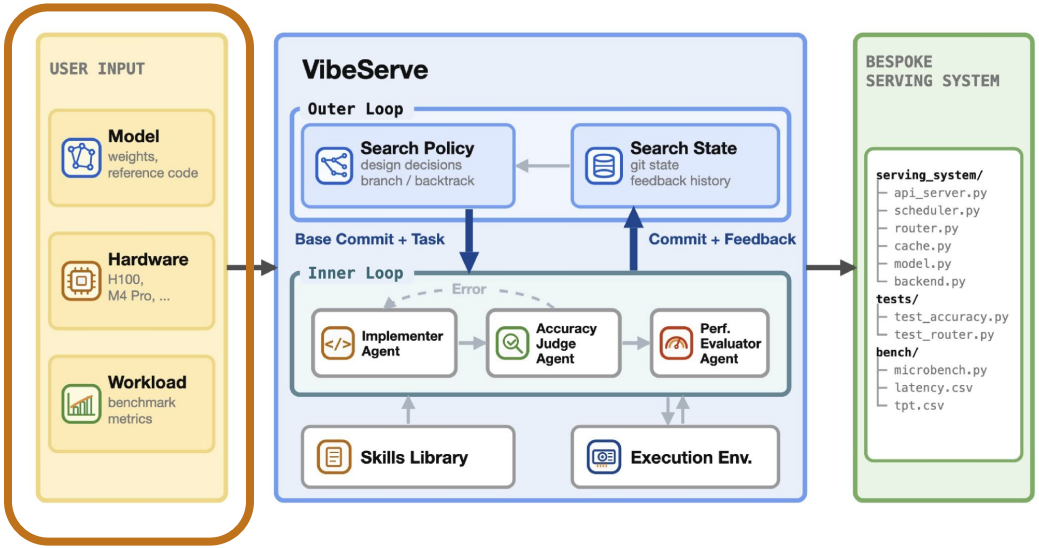
User inputs

Model weights + reference implementation (e.g., HuggingFace 🤗)

Accuracy checker – how to verify the correctness?

Benchmarking script – what to optimize?

Hardware specifications



VIBESERVE · ARCHITECTURE

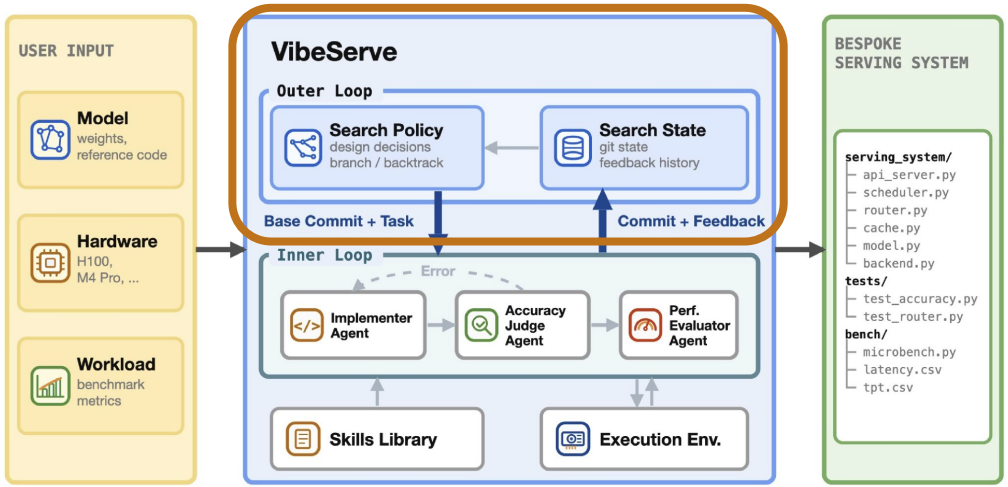
Outer loop

Searching optimizations over long-horizon is an active research area

- Our architecture supports many (e.g., Ralph loop, OpenEvolve, issue list)

Our instantiated approach: “GitHub issues”

- Inner loop files issues to indicate what to do next
- Outer loop agent decides what to dispatch



VIBESERVE · ARCHITECTURE

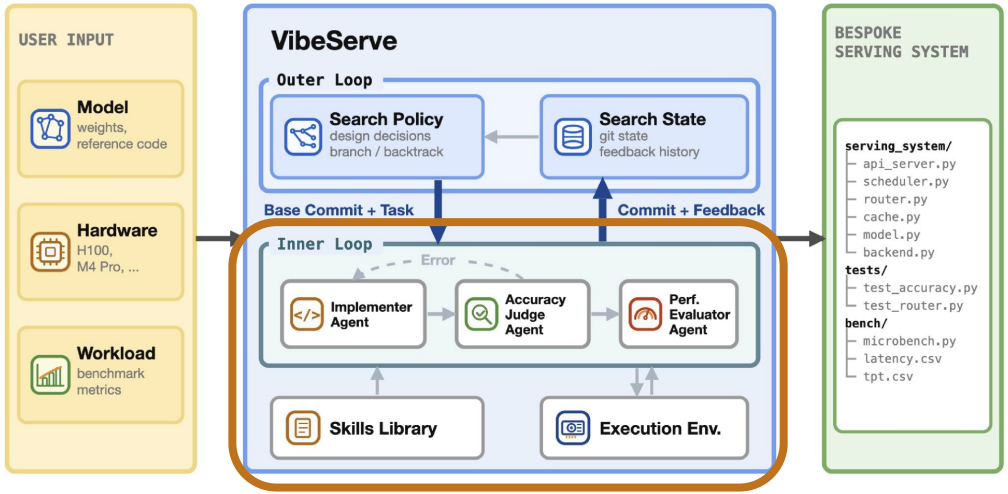
Inner loop

Separation of roles reduces hallucination (but slows down the loop)

Don’t gate loop exit on performance improvement

- Big design changes typically come with perf. degradation temporarily

Unstable perf. in execution environment is a challenge



Split context windows

Usually with humans, the person reviewing the code is not the person who authored the code. The person writing the code wants to merge the code, which can bias their actions to ship before it's ready.

Claude is the same way. The Claude that wrote the code wants the code to get accepted. The Claude that reviews wants to find issues in the code.

1 implementer, 2 or more adversarial reviewers per implementer. The reviewer's only job: find bugs & reasons why the code does not work. The implementer doesn't review. The reviewer doesn't implement.

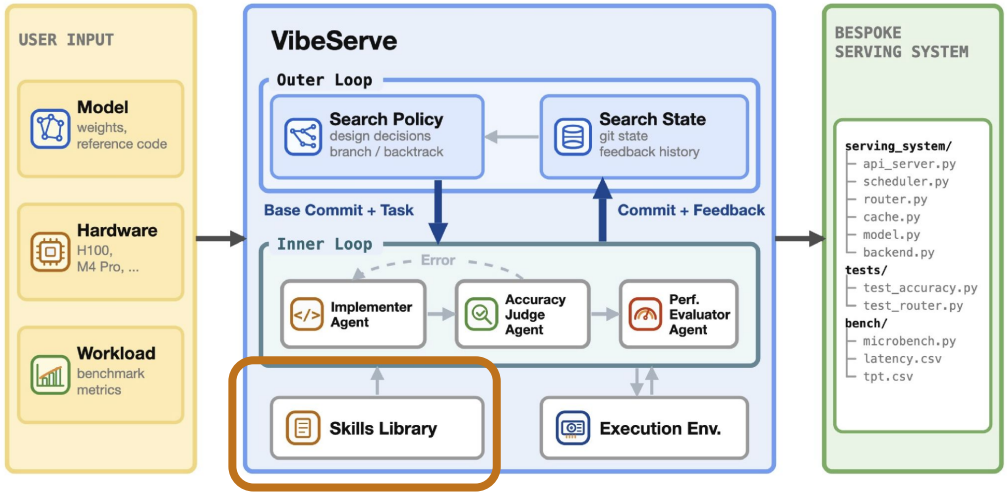
VIBESERVE · ARCHITECTURE

```
skills/serving-systems/
├── SKILL.md
├── models/
│   ├── text-dense/SKILL.md
│   ├── text-moe/SKILL.md
│   ├── ssm-hybrid/SKILL.md
│   ├── vision-language/SKILL.md
│   ├── speech-language/SKILL.md
│   ├── speech-generation/SKILL.md
│   ├── image-generation/SKILL.md
│   ├── video-generation/SKILL.md
│   └── omni-multimodal/SKILL.md
│
├── algorithms/
│   ├── attention-variants/SKILL.md
│   ├── async-scheduling/SKILL.md
│   ├── batched-sampling/SKILL.md
│   ├── chunked-prefill/SKILL.md
│   ├── continuous-batching/SKILL.md
│   ├── disaggregated-serving/SKILL.md
│   ├──
heterogeneous-kv-cache/SKILL.md
│   ├── moe-routing-dispatch/SKILL.md
│   ├── paged-attention/SKILL.md
│   ├── parallelism/SKILL.md
│   ├── quantization-schemes/SKILL.md
│   ├── radix-prefix-caching/SKILL.md
│   ├── speculative-decoding/SKILL.md
│   └── structured-output/SKILL.md
│
├── frameworks/
│   ├── pytorch/SKILL.md
│   ├── triton/SKILL.md
│   └── mlx/SKILL.md
│
├── backends/
│   ├── flashinfer/SKILL.md
│   │   └── references/*.md
│   ├── flashattention/SKILL.md
│   │   └── references/*.md
│   ├── triton-kernels/SKILL.md
│   └── cuda-graph/SKILL.md
│       └── references/*.md
│
├── hardware/
│   ├── nvidia/SKILL.md
│   ├── amd-mi300/SKILL.md
│   └── apple-silicon/SKILL.md
│
├── engines/
│   ├── vllm/SKILL.md
│   ├── sglang/SKILL.md
│   └── trtllm/SKILL.md
│
├── tooling/
│   ├── fastapi-serving/SKILL.md
│   │   └── references/*.md
│   ├── openai-api/SKILL.md
│   ├── accuracy-checker/SKILL.md
│   │   └── references/*.md
│   ├──
serving-benchmark/SKILL.md
│   ├── profiler/SKILL.md
│   ├── io-handling/SKILL.md
│   └── lora-serving/SKILL.md
│
└── repos/
    ├── vllm/
    ├── sglang/
    └── TensorRT-LLM/
```

Skills

Knowledge about serving system design distilled from papers and human-built systems

Extensible



VIBESERVE · EVALUATION

Evaluation questions

Research questions:

- How good is the system generated by VibeServe?
- Can VibeServe generate systems tailored to your particular needs?
  - Non-standard model/workload/hardware

Six scenarios:

- Case Study A: Standard LLM serving (Llama-3.1-8B on H100)
- Case Study B: Code editing with predicted outputs (Qwen3-32B on H100)
- Case Study C: Hybrid-architecture prompt caching (Olmo-Hybrid-7B on L4)
- Case Study D: Streaming ASR (Moonshine on L4)
- Case Study E: Local constrained JSON decoding (Llama-3.1-8B on a MacBook)
- Case Study F: Local image generation with Show-o2 (MacBook and H100)

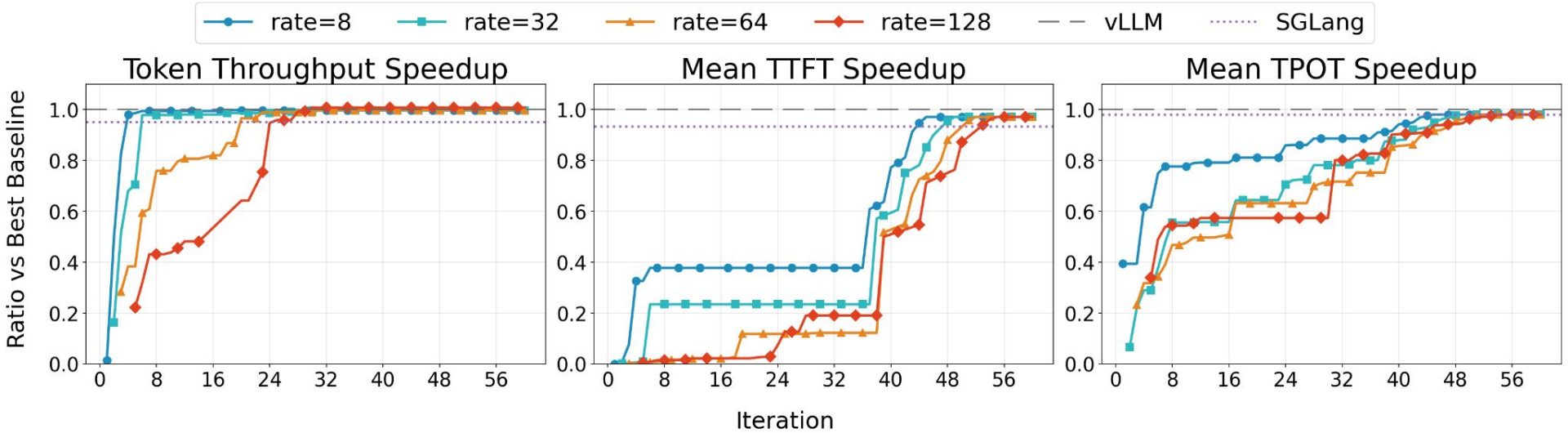
VIBESERVE · EVALUATION

Case study A: standard LLM serving

Llama-3.1-8B on H100 GPU

Standard scenario, existing systems are heavily optimized

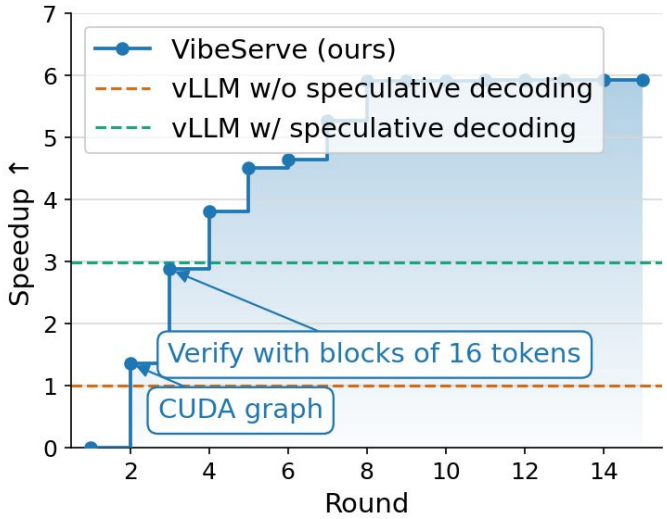
VibeServe matches vLLM/SGLang after ~50 iterations



VIBESERVE · EVALUATION

Case study B: Code editing with predicted outputs

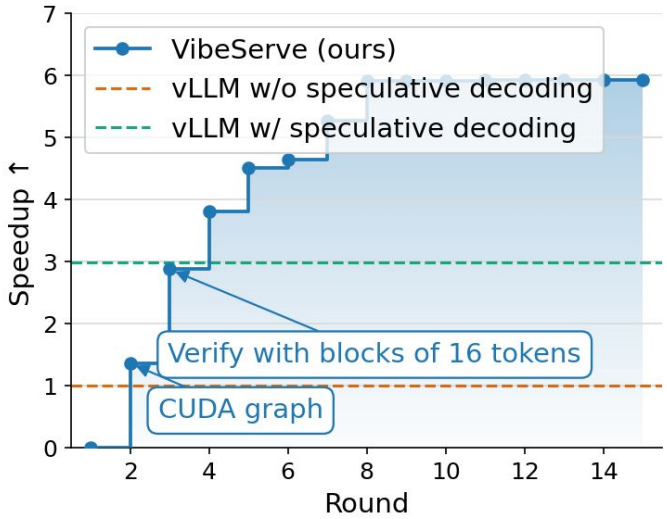
- Qwen3-32B on H100
- Some applications like code editing benefit from speculative decoding based on predicted-outputs, not draft models
  - Cursor is using it; OpenAI/Fireworks API support it
- But mainstream systems only support draft model-based speculation
- VibeServe outperforms vLLM by ~6x



VIBESERVE · EVALUATION

Case study C: Hybrid-architecture prompt caching

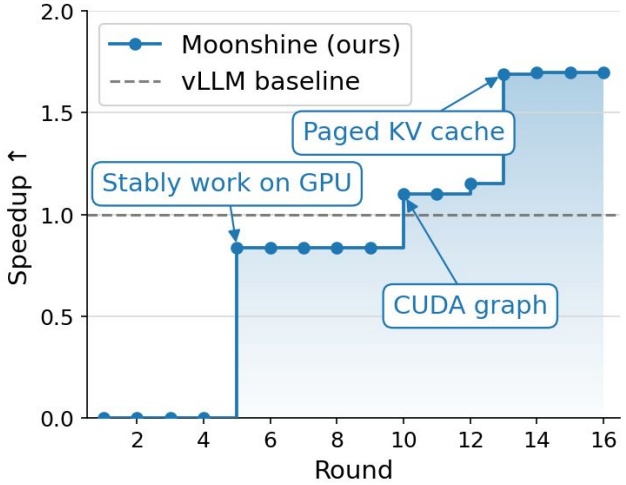
- Olmo-Hybrid-7B on L4
- Hybrid attention/SSM models reduce KV cache consumption, but prompt caching becomes difficult
- Knowing about workload pattern (e.g., RAG with predefined document) can allow you to optimize the system for that particular application
- VibeServe outperforms vLLM by ~3.5x



VIBESERVE · EVALUATION

Case study D: Streaming ASR

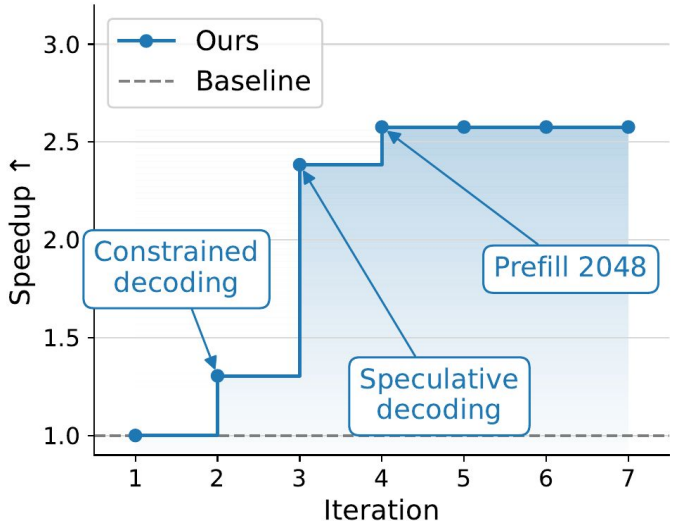
- Moonshine-Streaming on L4
- Variant of Whisper which uses sliding window attention in encoder to make streaming ASR efficient, but needs system-level support for caching
- vLLM can run the model with plugin, but encoder streaming requires heavy engineering
- VibeServe outperforms vLLM by ~1.7x



VIBESERVE · EVALUATION

Case study E: Local constrained JSON decoding

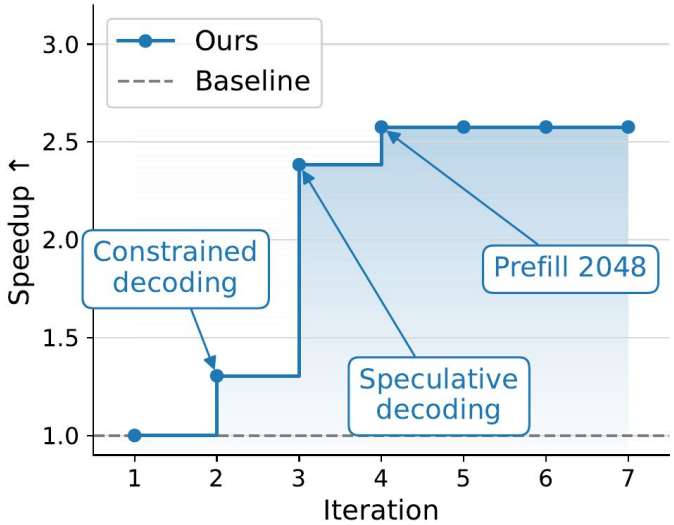
- Llama-3.1-8B on a MacBook
- JSON decoding allows you to jump-forward some tokens if those are determined by grammar specification
- MacBook is different from NVIDIA GPUs: unified memory, MLX framework
- VibeServe outperforms baseline by ~2.6x



VIBESERVE · EVALUATION

Case study F: Image generation with an exotic model

- Show-o2 on a MacBook and H100
- Show-o2 is a vision foundation model with vastly different architecture from standard LLMs, and none of serving systems support it
- VibeServe outperforms baseline by ~6.3x on a Macbook, ~20% lower latency on H100



VIBESERVE · TAKEAWAY

Takeaways

VibeServe can match the performance of human-engineered systems in standard settings

- No compromise in performance

VibeServe can significantly outperform existing systems in novel (yet realistic) scenarios

- It allows optimizations that break the abstractions of generic-systems

VIBESERVE · TAKEAWAY

Conclusion

VibeServe: rethinking how we build systems

- One-sizes-fits-all to design-time specialization

Future directions:

- How to truly verify the correctness of implementation?
- Any other systems
- Benchmarking

To learn more:

- Blog: https://syfi.cs.washington.edu/blog/2026-05-12-introducing-vibeserve/
- Paper: https://arxiv.org/abs/2605.06068
- Code: https://github.com/uw-syfi/vibe-serve

VIBESYS · BEYOND SERVING

Vibe-building systems — not just serving systems.

DEEPER INTO SERVING

VibeServe on Trainium — LLM serving on AWS silicon, from scratch. (2.7x improvement over Public Tranium baseline)

Vibe-MegaKernel — bespoke megakernels for exotic models.

Vibe-SpeechAgent — cascaded STT–LLM–TTS engines, tuned end to end.

VibeServe for C-to-Rust — bespoke serving for token-hungry translation agents.

BEYOND SERVING

Microservices — deployment topology, communication stacks, even the app itself.

Key-value stores — bespoke storage primitives.

Distributed queues — bespoke coordination primitives.

Vibe-Database — engines for non-monotonic streaming workloads.

SHARPENING THE LOOP

Vibe-M* — the agent searches M*'s own structured action space — cheaper to verify, safe by construction.

Simulators — large-scale search without burning real hardware.

HLS — a benchmark to grade every one of them.

WHAT WE DO NOT YET KNOW

Two questions still waiting on better answers.

01. SEARCH AT SCALE

How do we make agent search efficient and reliable as the design space grows?

The search space has to be designed for agents.

02. DISCOVERY

Can agents discover genuinely new ideas, or only recombine existing ones?

Optimizing within a known design space is one thing; proposing techniques that are not yet in the corpus is another. The right abstraction may matter more for discovery.