Self-defining systems

Ratul Mahajan

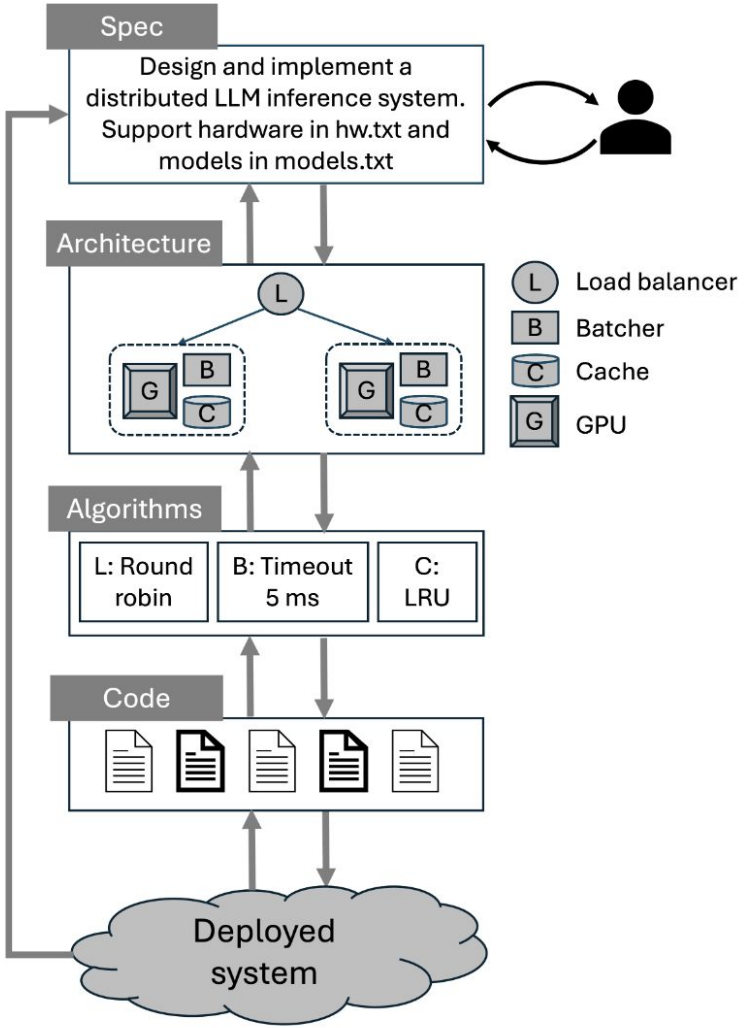
w/ Tom Anderson, Simon Peter, and Luke Zettlemoyer

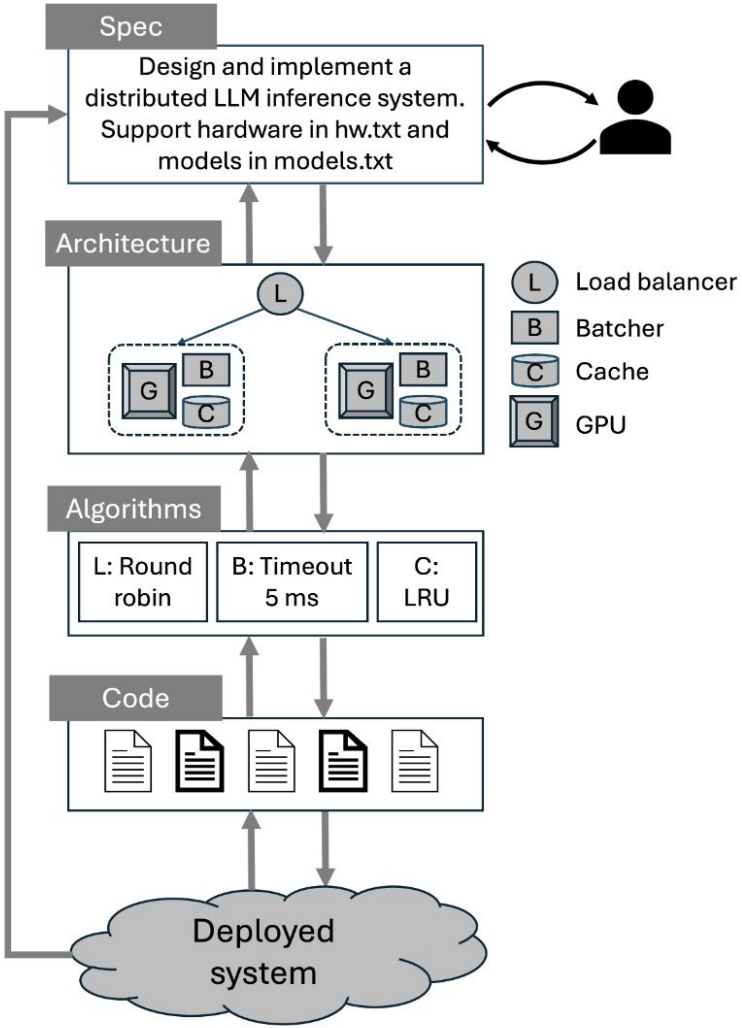
University of Washington

https://self-defining-systems.uw.edu/

Self-defining systems

Q: Can agents design, develop, and operate systems end-to-end?


Spec
Design and implement a distributed LLM inference system. Support hardware in hw.txt and models in models.txt
Architecture
L
G
B
C
G
B
C
Load balancer
Batcher
Cache
GPU
Algorithms
L: Round robin
B: Timeout 5 ms
C: LRU
Code
Deployed system



Research challenges

Specifying objectives

Searching over architectures

Efficiently rejecting hypothesis

Ensuring security and robustness

Knowing when to stop

Portion size of “bitter lesson”

Case study 1: A self-defining operator for distributed applications

Shihang Li, Tom Anderson, Ratul Mahajan,
Simon Peter, Luke Zettlemoyer

Operating distributed applications

Must maintain high availability despite the diversity of faults

SREs leverage a lot of operational context and history to lower MTTR

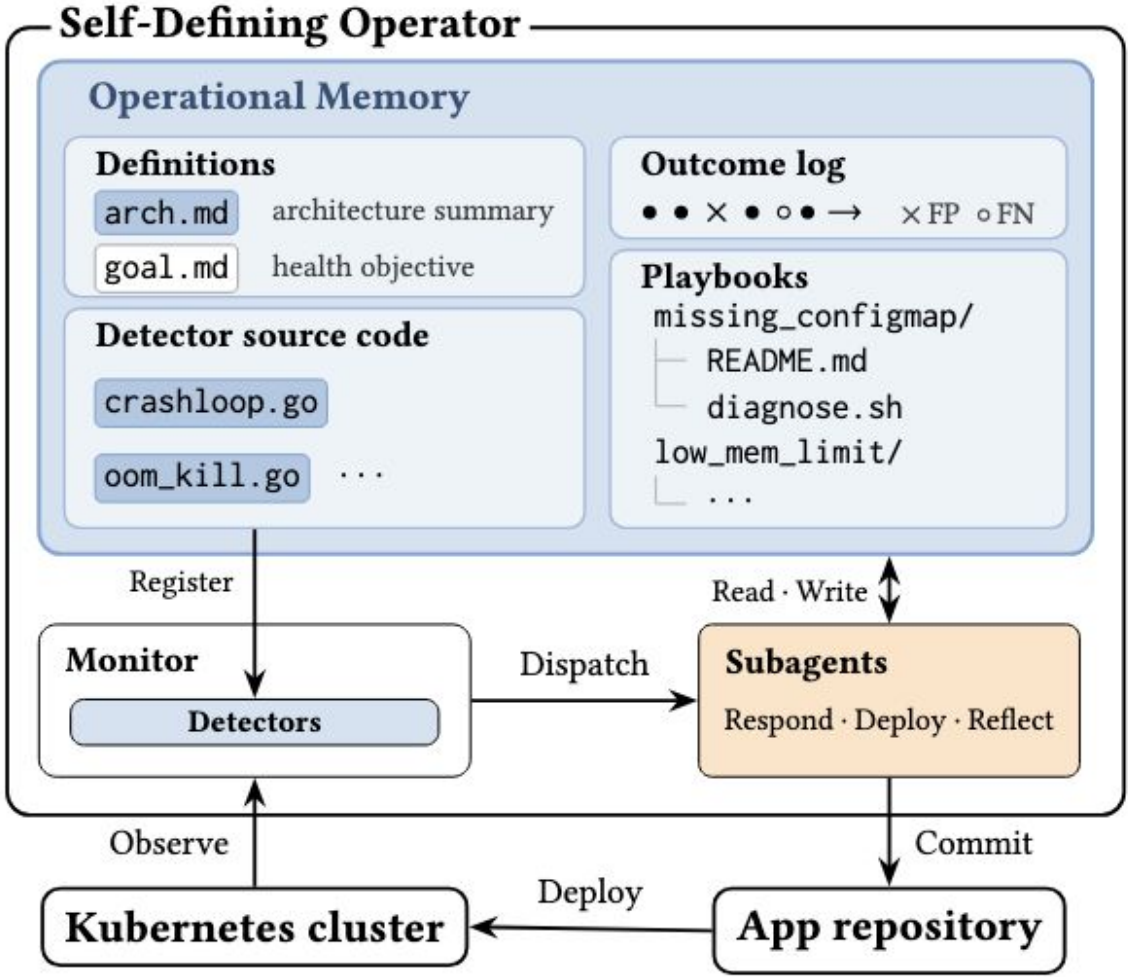
Self-defining operator

Autonomous and learning operator for distributed applications

Deploys applications from source and keeps them healthy

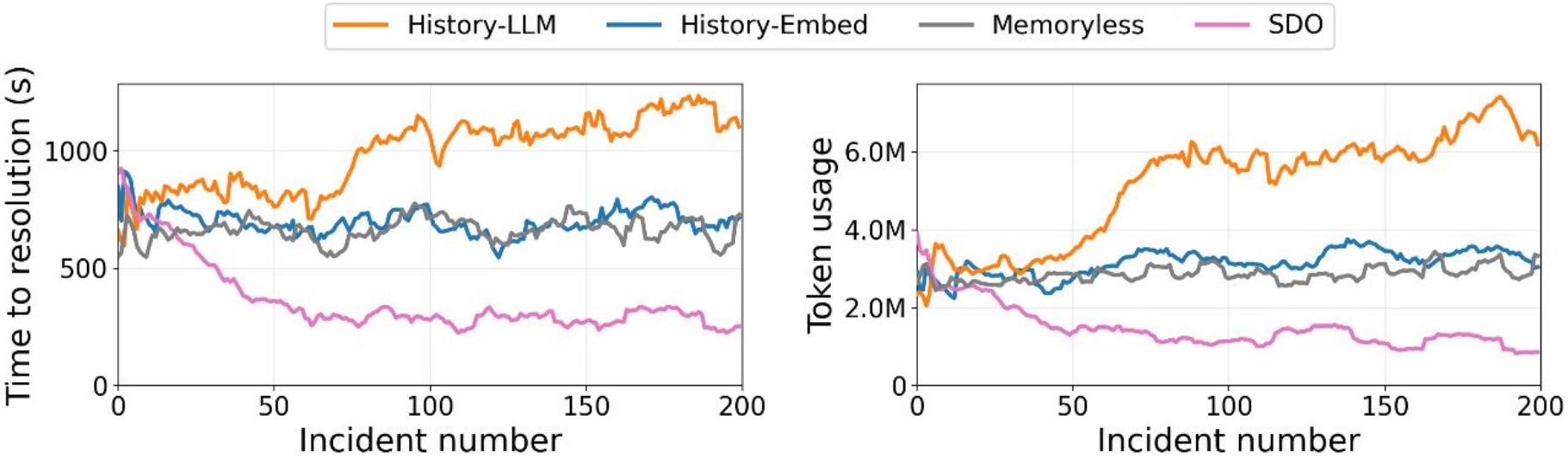
Creates and updates its own operational code and memory

Detectors
flag
unhealthy
states



Playbooks
encode
diagnostic
procedure

SDO is faster and cheaper



Case study 2: VibeServe: A bespoke LLM serving system

Keisuke Kamahori, Shihang Li,
Simon Peter, Baris Kasikci

https://arxiv.org/pdf/2605.06068

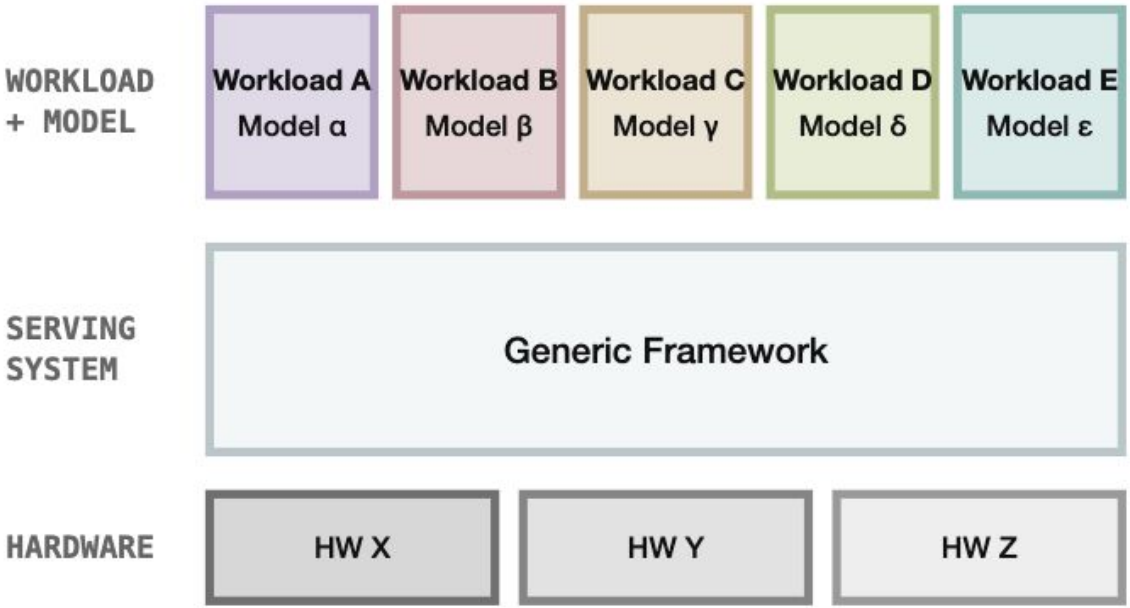
LLM serving systems

Goal: Efficiently map models to underlying hardware

Examples: vLLM, SGLang, TensorRT-LLM

Generic serving today

Generic runtimes cover common cases.

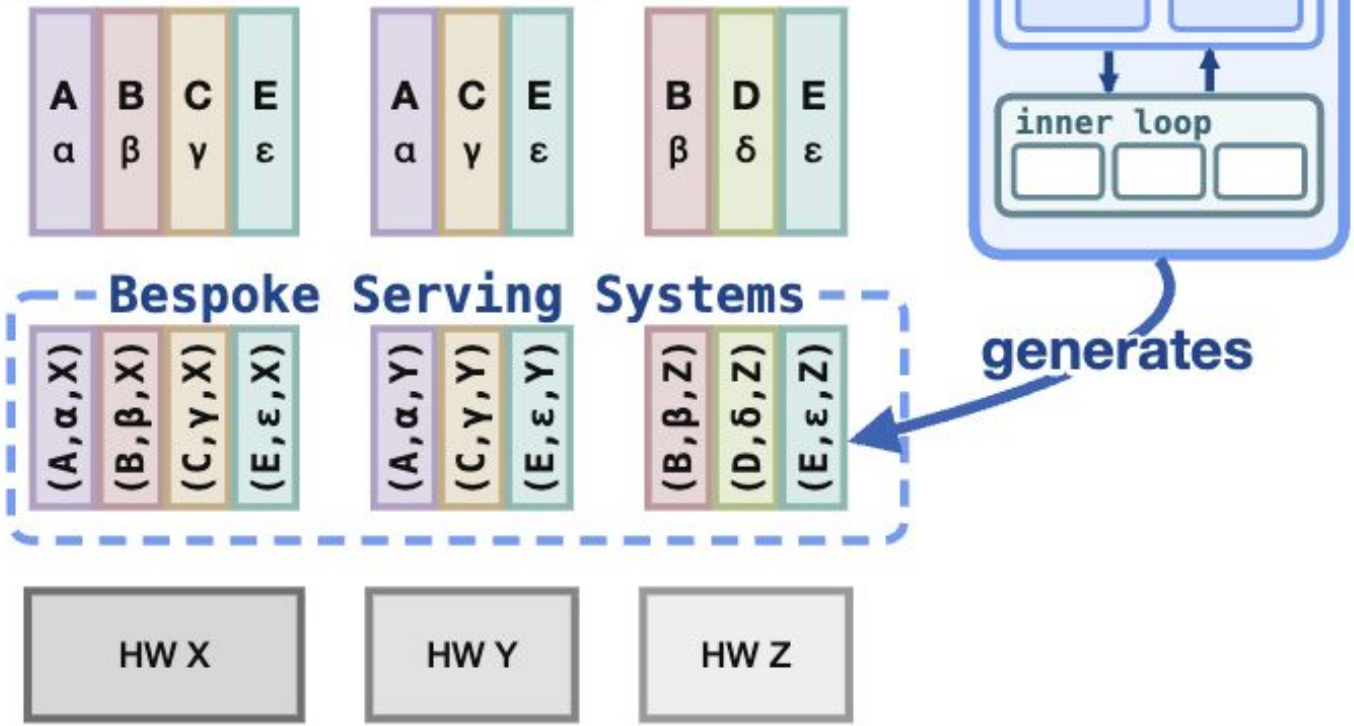


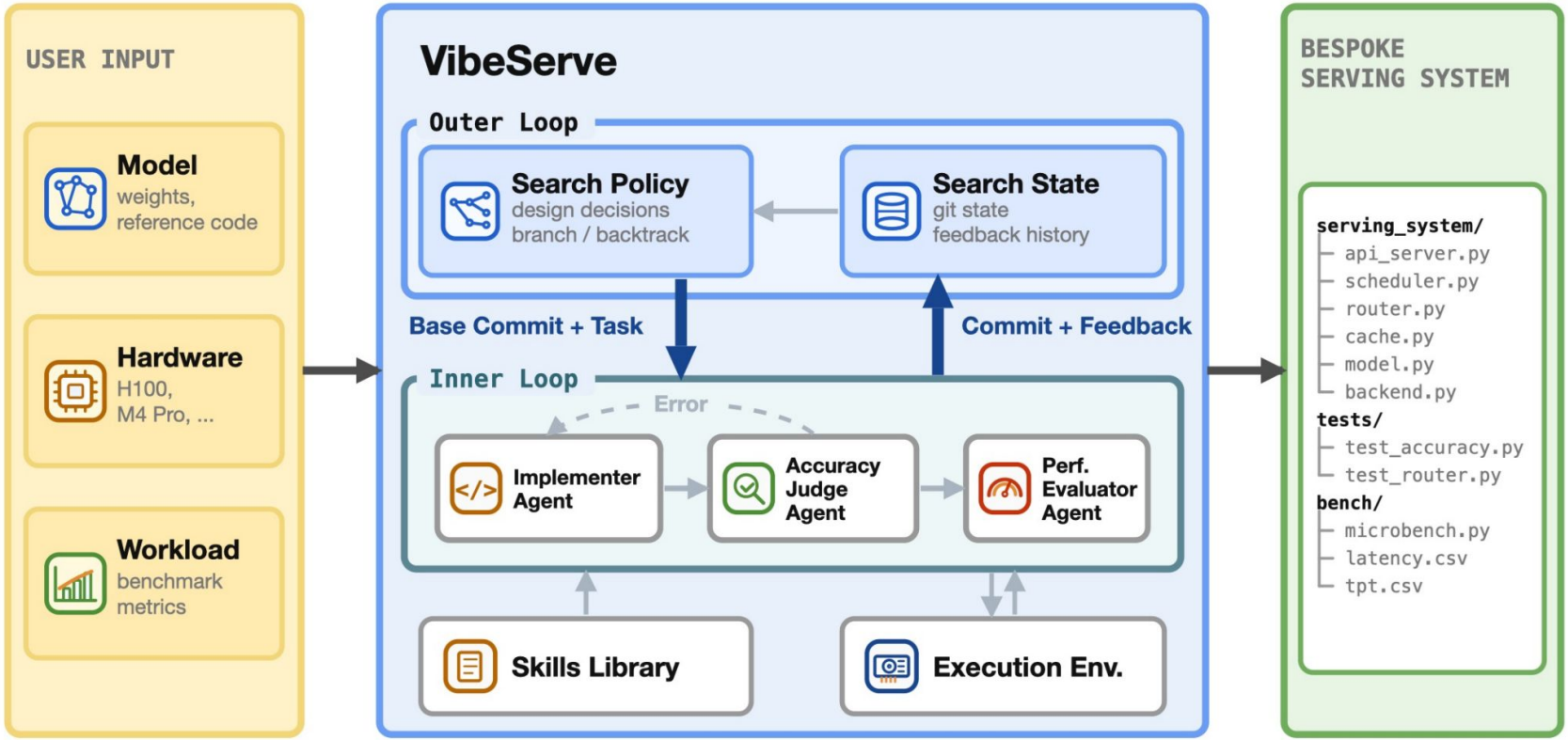
Need specialization for models, workloads, HW

Cannot handle the wide diversity of possibilities

VibeServe's approach

One bespoke serving system per (workload, model, hardware) target.




USER INPUT
Model
weights,
reference code
Hardware
H100,
M4 Pro, ...
Workload
benchmark
metrics
VibeServe
Outer Loop
Search Policy
design decisions
branch / backtrack
Search State
git state
feedback history
Base Commit + Task
Commit + Feedback
Inner Loop
Error
Implementer Agent
Accuracy Judge Agent
Perf. Evaluator Agent
Skills Library
Execution Env.
BESPOKE SERVING SYSTEM
serving_system/
api_server.py
scheduler.py
router.py
cache.py
model.py
backend.py
tests/
test_accuracy.py
test_router.py
bench/
microbench.py
latency.csv
tpt.csv

VibeServe performance on uncommon scenarios

Predicted-output code editing *Cursor-style long edits*	*5.95×*
Long shared context *RAG-style prompts*	*3.45×*
Streaming voice transcription *Real-time*	*1.69×*
On-device image generation *MacBook (Apple Silicon)*	*6.27×*

Matches vLLM and SGLang on common scenarios

Self-defining systems

Q: Can agents design, develop, and operate systems end-to-end?

A: Yes (likely)

What might *you* be doing in a few years?

Writing specs and benchmarks

Developing formal methods for correctness, performance