The New Golden Edge for *the Computer Architect*

AI, Abstraction, and the Future of System Design

Sophia Shao
ysshao@berkeley.edu
Electrical Engineering and Computer Sciences
https://people.eecs.berkeley.edu/~ysshao/

Berkeley
UNIVERSITY OF CALIFORNIA

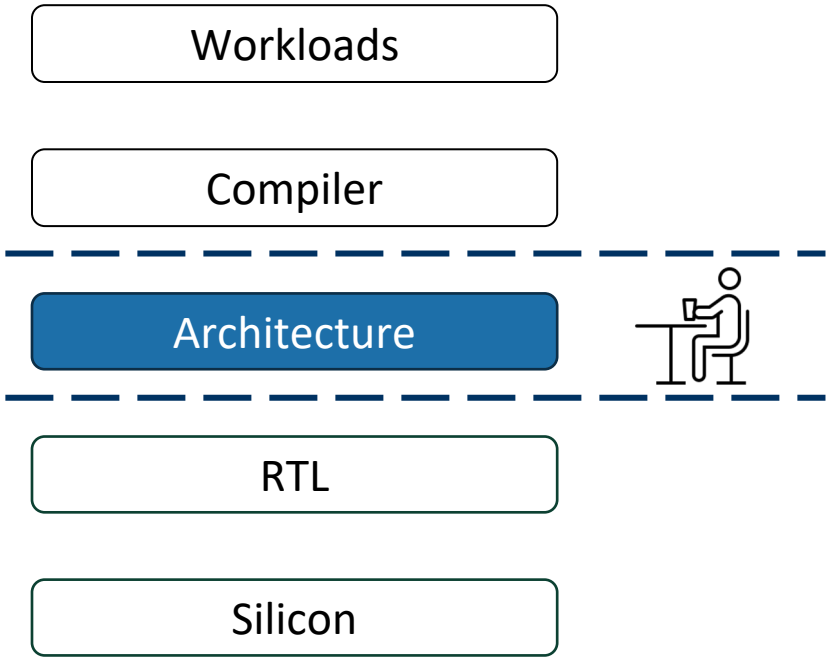
We're told this is a golden age for computer architecture.

But it's something more dangerous, and more exciting!

This is the golden age for the computer architect!

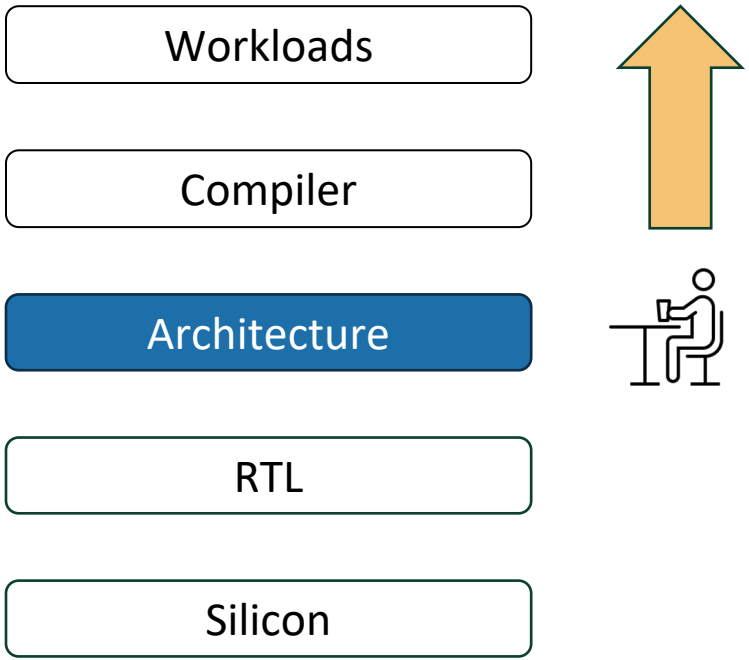
Shift from the field -> the role!

The Old Role



- Architects are used to be stuck in the middle.
  - with limited control over both ends.
- Months even years from
  - Arch idea -> SW
  - Arch idea -> Silicon

What AI enables



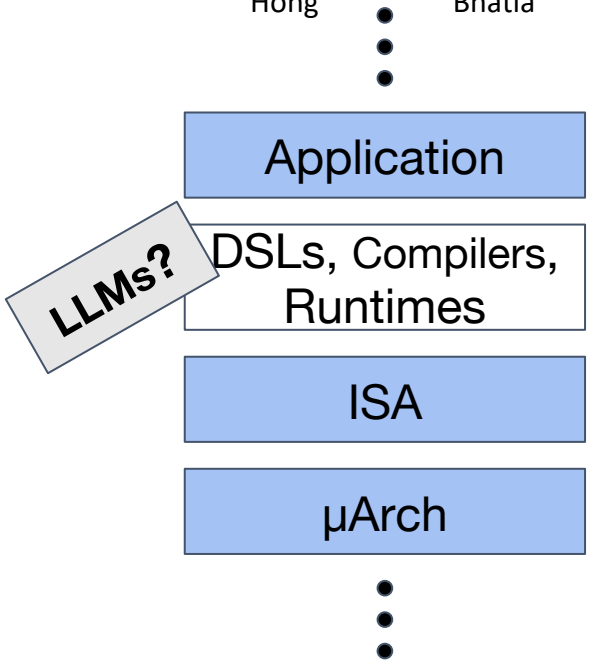
- AI removes the distance to realization.
  - Arch -> SW

The Autocomp Approach

- Early-stage accelerator compilers should be **agile**
  - Both the application and hardware levels can change during accelerator development
  - LLMs can code
  - Easy to build and update LLM-based implementations: just change the prompt
- But accelerator software is **low-resource**!
  - Not like generating Python…


Charles Hong

Sahil Bhatia



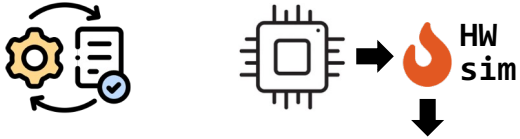
https://github.com/ucb-bar/autocomp

*[MLArchSys'2025, **Best Paper Award**]*

The Autocomp Approach

Hardware Feedback:

Correctness + Performance



- `Cycle Count`
- `Memory util`

1. Plan

Teach the accelerator's ISA.
Look at original code.
Suggest **a list of possible optimizations** to apply.
Select **one** and describe how to apply it (what variables/instructions/loops to apply to).

Sample!

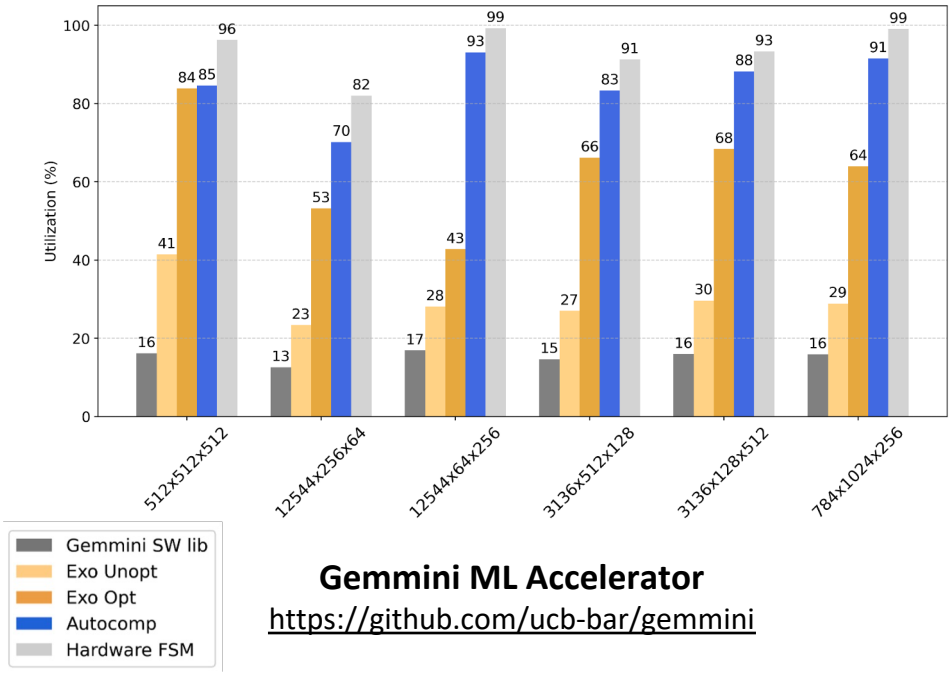
2. Implement

Look at original code.
Take the selected optimization, and implement it.

Sample!

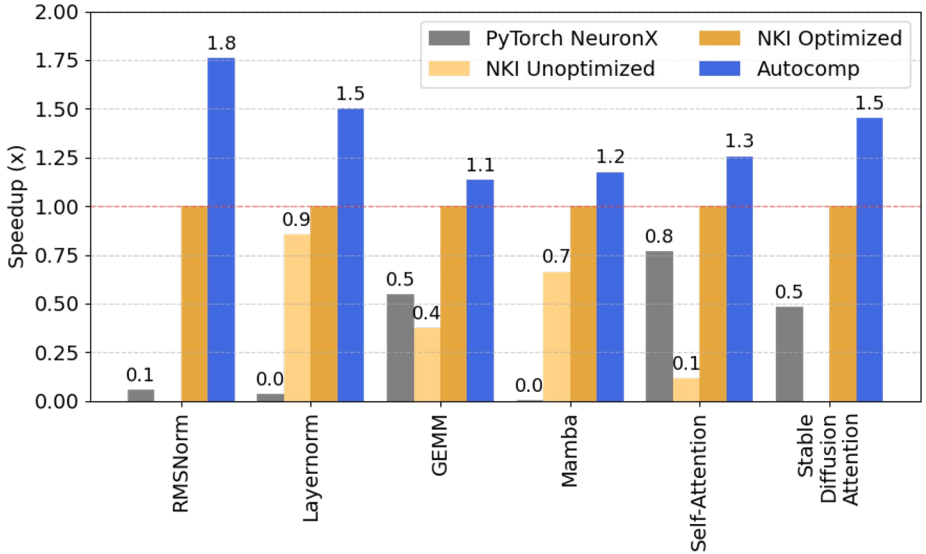
It beats everything!

- From our own accelerator to commercial ones.



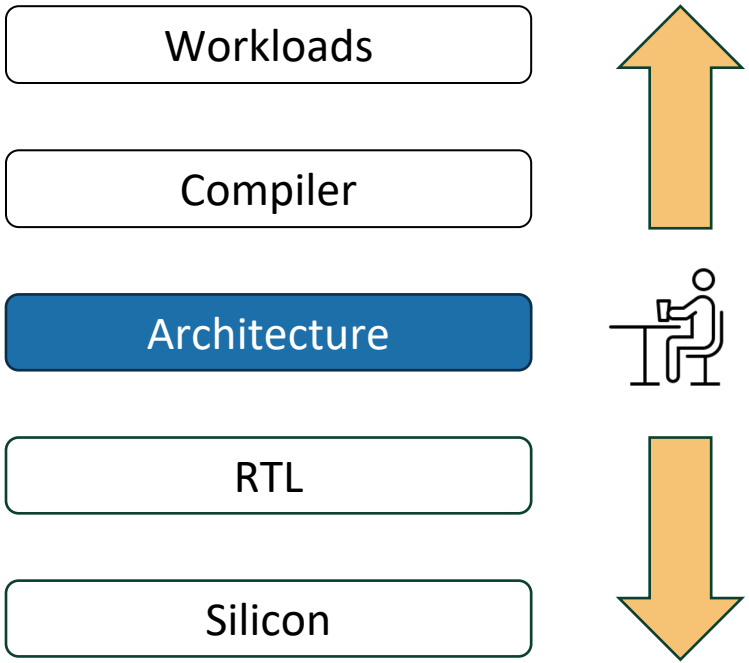
Gemmini ML Accelerator

https://github.com/ucb-bar/gemmini



AWS Trainium

What AI enables



- AI removes the distance to realization.
  - Arch -> SW
  - Arch -> Silicon

The Radiance Tapeout

- **AI-native GPU** with decoupled tensor cores, taping out in 2 weeks!

Muon SIMT Core

Stable, performant GPU core through occupancy and OoO support

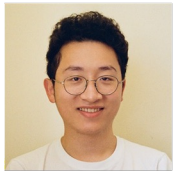
MxGemmini

Area-efficient fused PE for FP4/FP6/FP8

Decoupled Tensor Core

Efficient pipelining and fusion without register pressure

Amanda Shi*



Richard Yan*

Hansung Kim*

Shashank Anand

Joshua You

Nicolas Rakela

Coleman Hooper

Chloe Wong

Abhi Pomalapally

Reality Check

- **Concept-to-tapeout**
  - 6 months (!)
  - < 10 student researchers
  - First tapeout for most of the team
- **Where AI is helping**
  - Scripting
  - Unit testing
  - CI/CD integration
  - Tedious parts of the RTL generation
- **Still need human scaffolding**
  - Architecture reasoning was done in the old-fashioned way.

Muon SIMT Core

Stable, performant GPU core through occupancy and OoO support

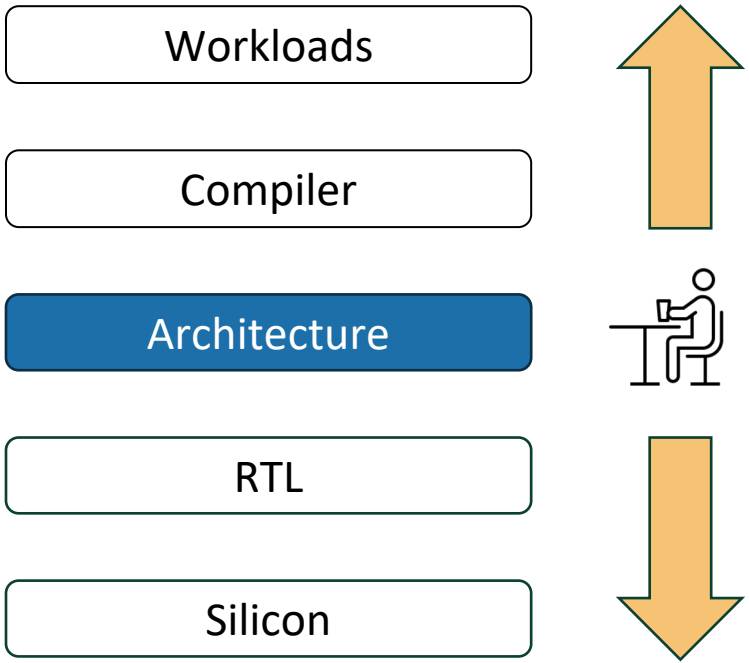
MxGemmini

Area-efficient fused PE for FP4/FP6/FP8

Decoupled Tensor Core

Efficient pipelining and fusion without register pressure

What AI enables

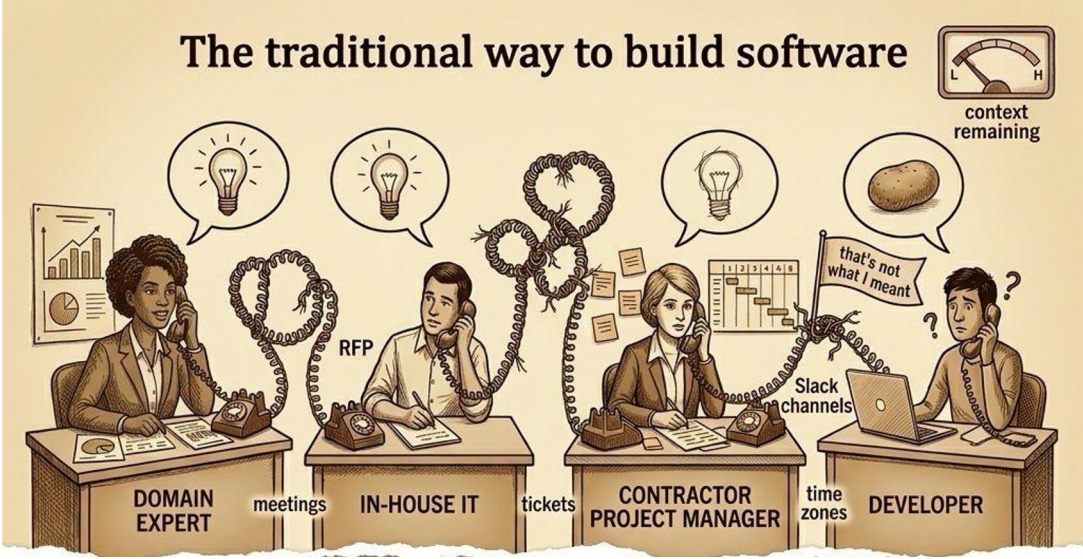


- **AI removes the distance to realization.**
  - Arch -> SW
  - Arch -> Silicon
- **Architects can potentially reach the entire stack.**
- **But not there yet!**

So why can’t we Auto-Architect?

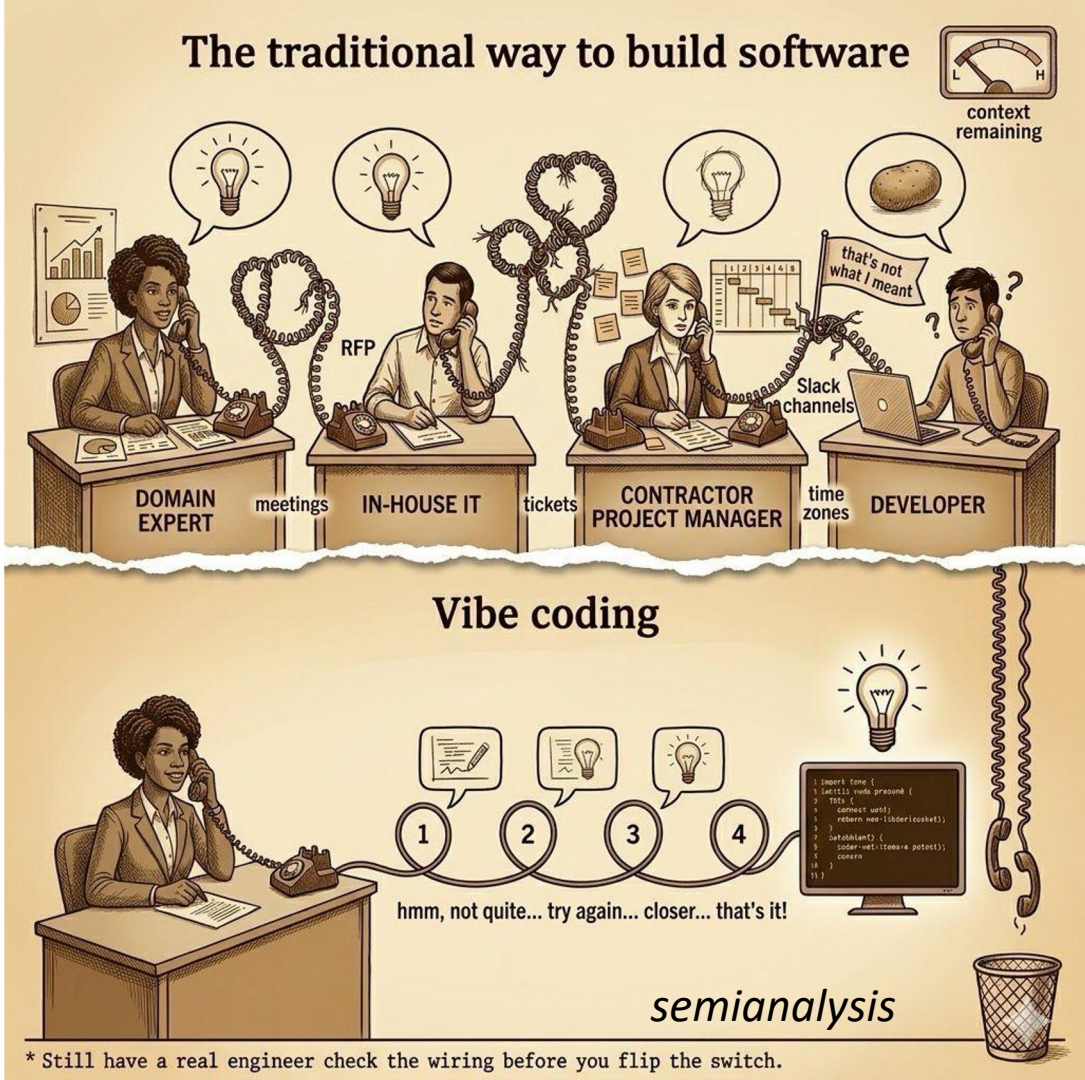
- AI can optimize code.
- AI can generate implementations.
- But it still can’t design full-stack architecture (yet).

Lessons from Vibe Coding



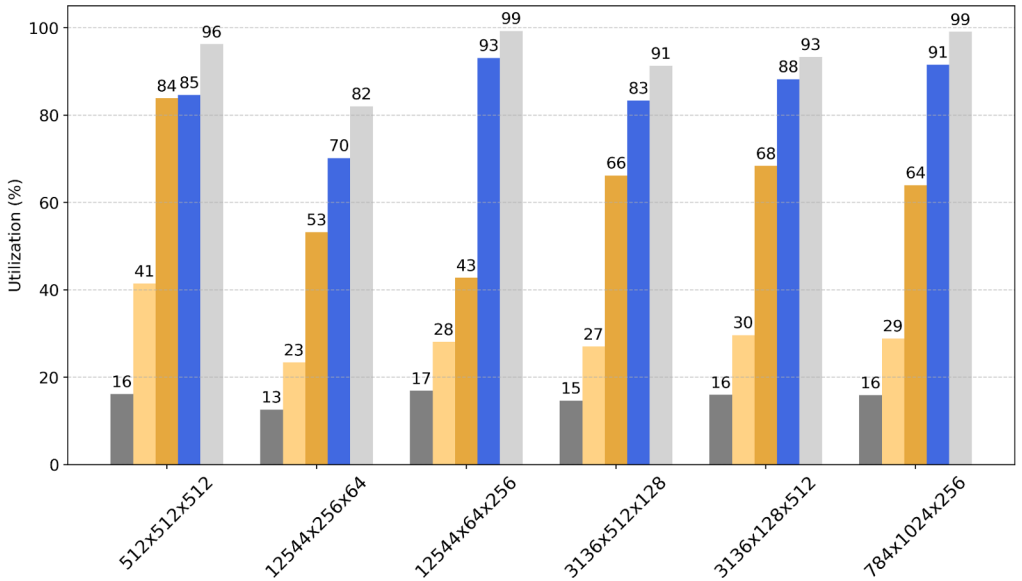
Why can’t we Auto-Architect?

- AI does not fully understand our workflow…
- Or we can not articulate our workflow.
  - How we think about tradeoffs
  - How we structure design space
  - How we define constraints
- Our domain expertise exists, but not in a form AI can use.



Why Does Autocomp Work So Well?

What actually makes Autocomp auto?



Domain expertise is encoded as menu options.

- Optimization menu explicitly encode the designers' heuristics.

Experiment	12544x64x256 GEMM Speedup	4x3x14x256x256 Conv Speedup
Baseline (Exo Unopt)	1.67×	0.91×
No Accelerator ISA	3.11×	2.51×
No Optimization Menu	2.34×	0.97×
No Optimization Menu Dropout	4.72×	2.30×
No LLM Ensemble (o3-mini only)	4.67×	2.08×
No Hardware Perf Feedback	4.91×	2.61×
LLM Selection (DeepSeek-R1)	4.89×	2.25×
LLM Selection (Gemini 2.5 Flash)	3.63×	2.42×
LLM Selection (Llama 4 Maverick)	4.14×	1.01×
Autocomp	**5.53**×	**2.72**×
Variance Checking	5.2±0.33×	2.61±0.11×

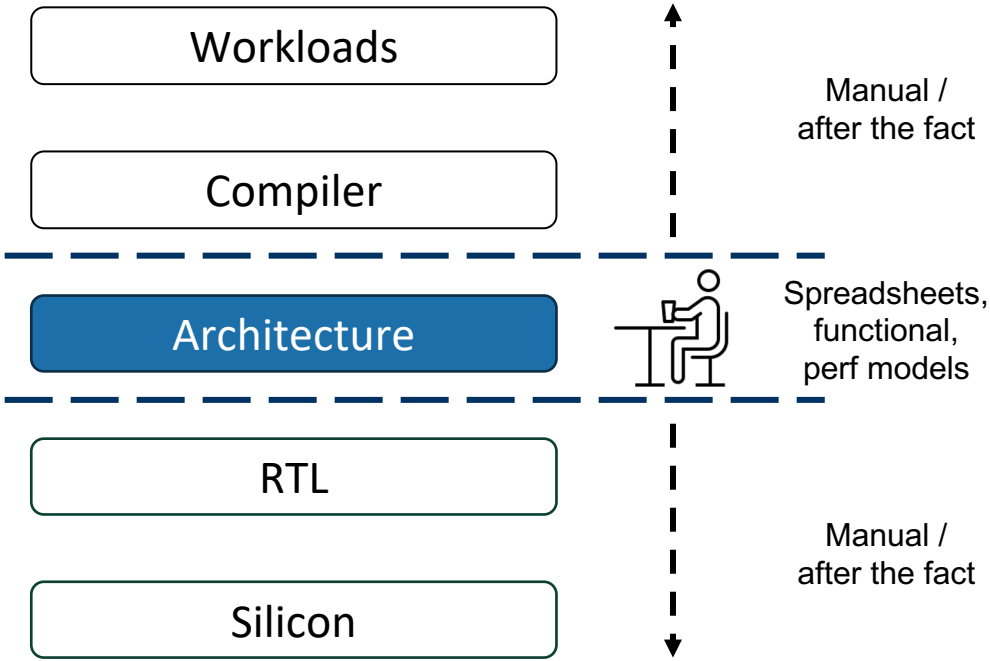
What did we actually learn from Autocomp?

- AI works best when the workflow is explicit and connected.
  - Design space is defined.
  - Options are structured.
  - Heuristics are encoded.
  - Feedback loops exist.

- Why can’t we repeat this in architecture?

What's missing in architecture?

- Architecture is not encoded.
  - No explicit structural design space.
  - Heuristics live in our heads.
- No connection to the stack.
  - No direct link to SW.
  - No direct link to RTL generation.
- We reason in isolation…then **manually** translate.



So what needs to change?

- We design architecture using intuition.
- Our workflow itself is fragmented.
- Architecture is reasoned by hand or in models…but realized somewhere else.

A Case Study: The Tapeout Class

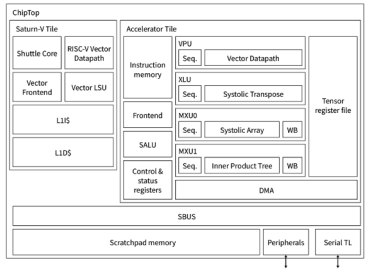
Workloads

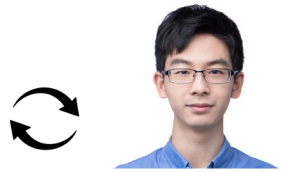
Compiler

Architecture

RTL

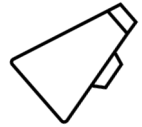
Silicon



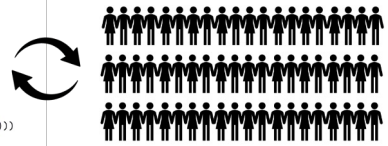


Yufeng Chi

Spec



```
// Activations: pass through IEEE E4M3
val req = io.compute.bits

// Weight double-buffer (stores IEEE)
val ieeeWidth = p.inputFmt.ieeeWidth

val wEn = RegInit(false.B)
val bufReadSel = ~wEn

val zeroRow  = VecInit(Seq.fill(p.vecLen)(0.U(ieeeWidth.W)))
val zeroTile = VecInit(Seq.fill(p.numLanes)(zeroRow))
```



Tapeout Class

https://ee194.org/

Challenge 1: Abstraction is broken.

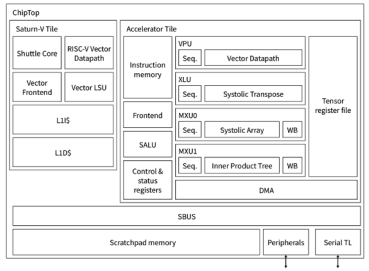
Workloads

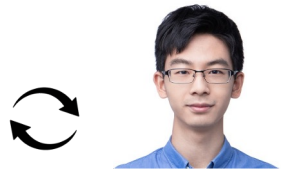
Compiler

Architecture

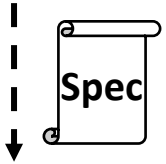
RTL

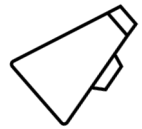
Silicon





Yufeng Chi

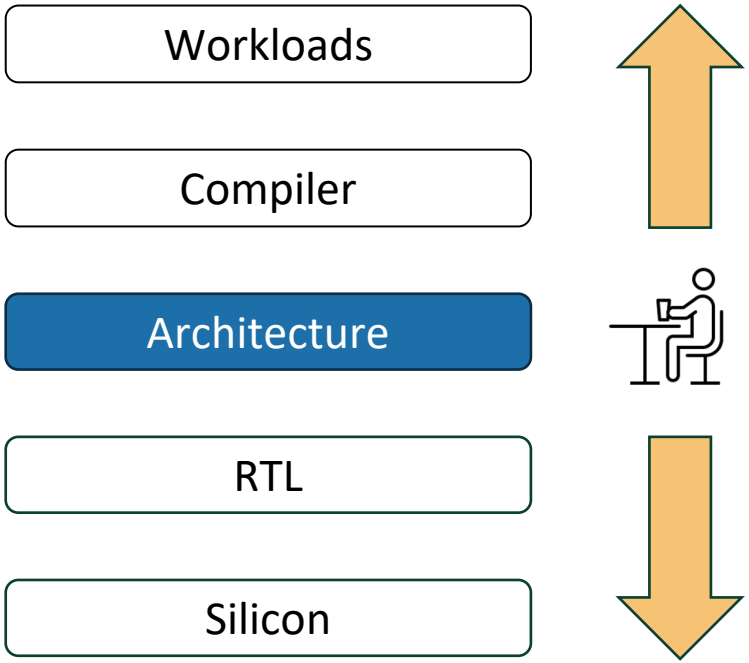




```
// Activations: pass through IEEE E4M3
val req = io.compute.bits

// Weight double-buffer (stores IEEE)
val ieeeWidth = p.inputFmt.ieeeWidth

val wEn = RegInit(false.B)
val bufReadSel = ~wEn

val zeroRow  = VecInit(Seq.fill(p.vecLen)(0.U(ieeeWidth.W)))
val zeroTile = VecInit(Seq.fill(p.numLanes)(zeroRow))
```

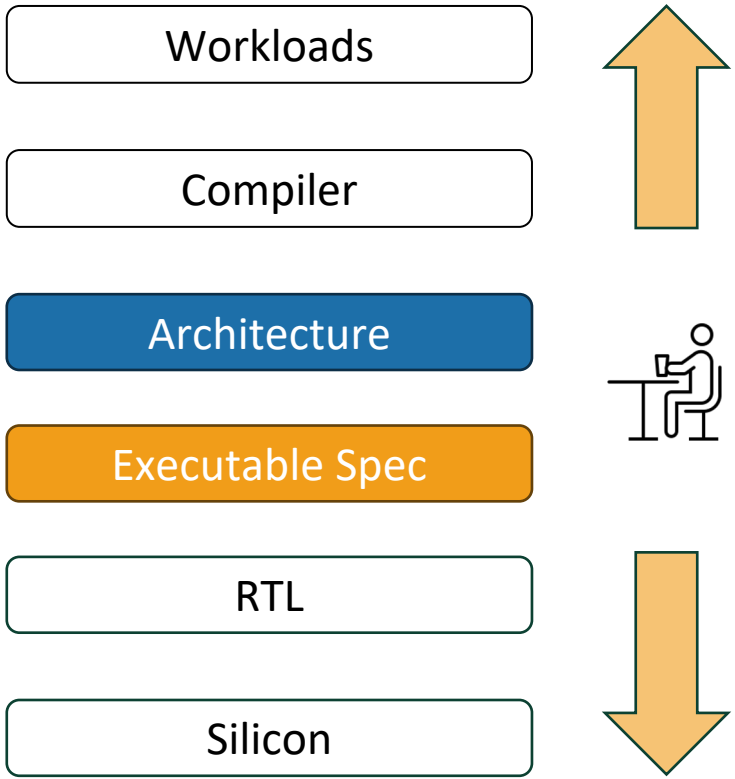
Between architecture and RTL…
there is no structured abstraction.

Challenge 1: Abstraction is broken.



- **Our abstraction stack is incomplete.**
  - Gap between Architecture -> Silicon
- **The only structured abstraction is RTL.**
  - Everything above: heuristics, experience, tribal knowledge
- **Architects reason in unstructured models.**
  - Not easily learnable by AI.
  - Not easily transferrable to RTL.

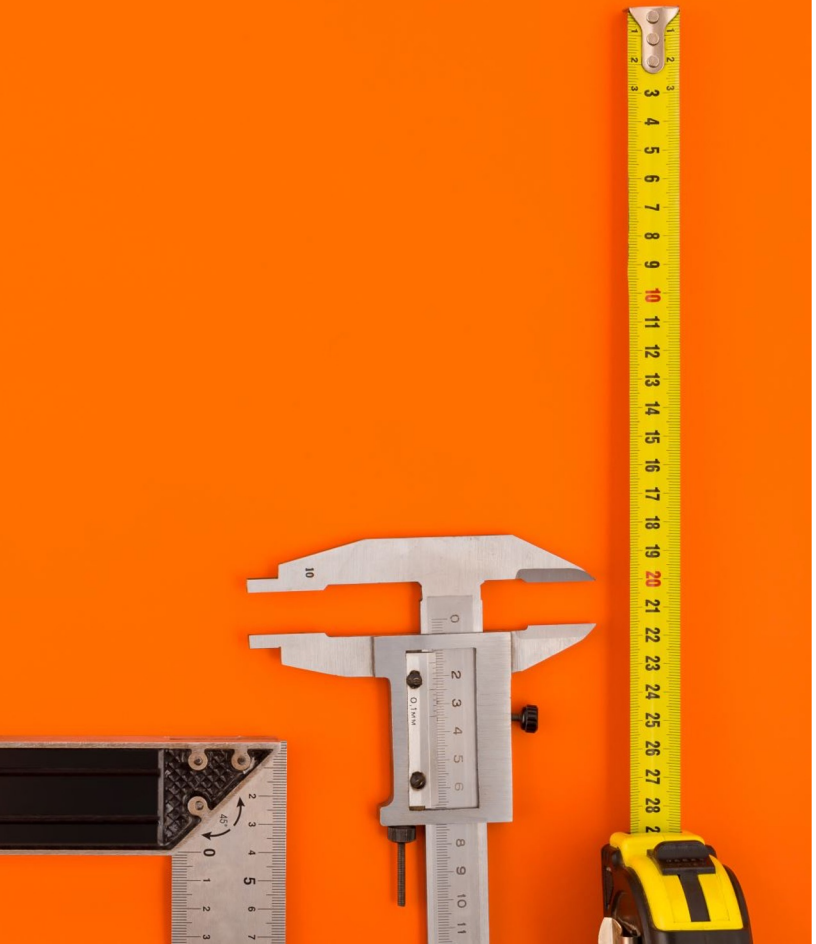
Need New Abstraction



- **We need a new structured abstraction of architectural intent.**
  - Make it learnable by machines.
- **The same abstraction can be directly connected to the RTL generation.**

Challenge 2: Tools are the Bottleneck

- The irony
  - AI is fast.
  - Our tools are not.
    - Arch -> RTL -> Feedback loop is long.
- Today's tools:
  - Designed for human
  - Sequential workflows
  - Manual iteration
  - Slow feedback loops
- Tool innovation is the bottleneck.



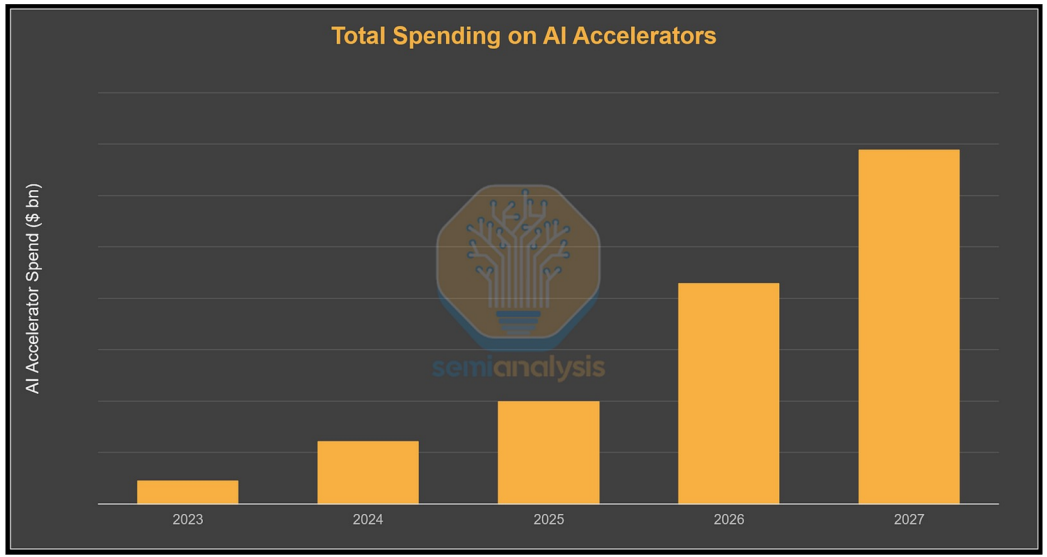
Need AI-Native Tooling

- We need not just faster tools but also different tools.
- We need tools that
  - Evaluate quickly
  - Verify automatically
  - Orchestrate exploration
- The architecture loop must become continuous, not staged.

Now let's assume we solve all of this

- We build the right abstractions.
- We have AI-native tools.
- Then the real question is…what should we build?

Explosion of Use Cases of AI



semianalysis

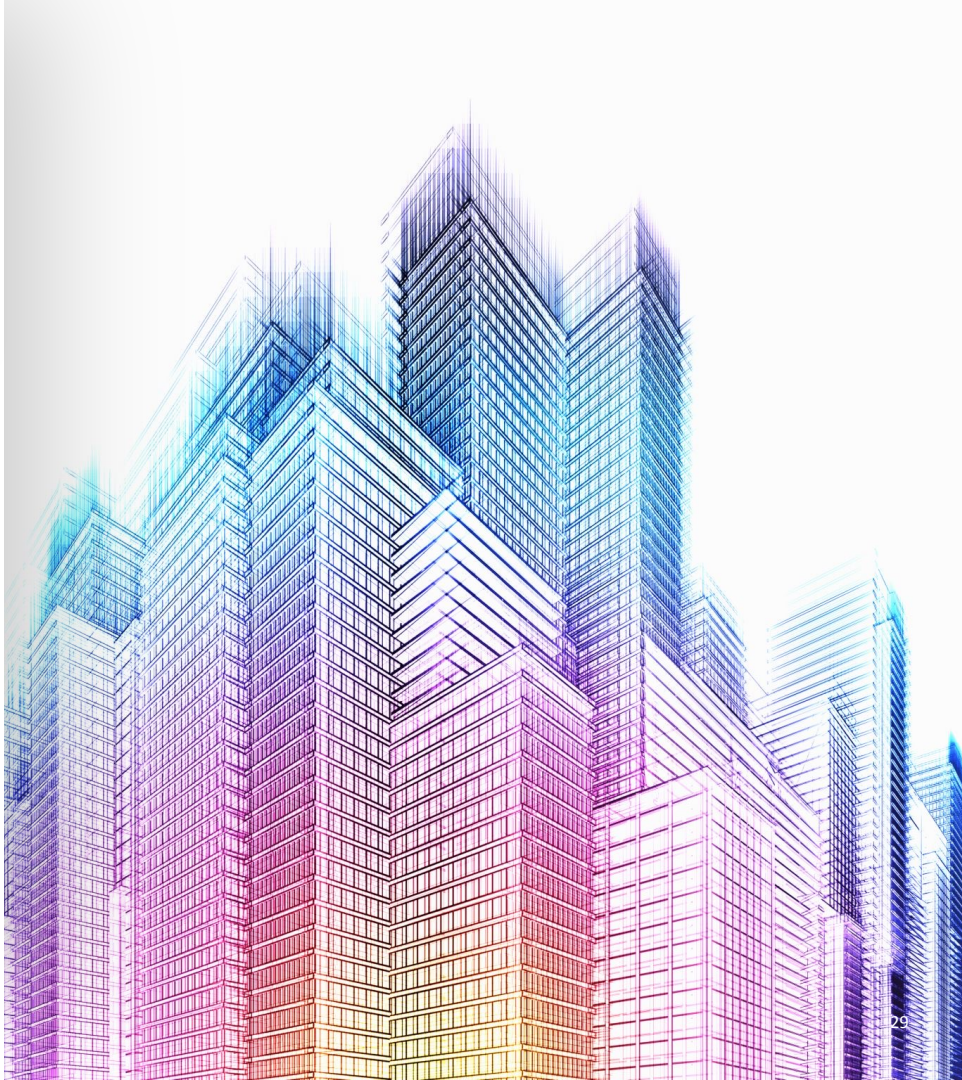
- AI Demand is Exploding
  - Training
  - Inference and reasoning
  - Edge
  - Datacenter
  - Real-time systems
  - Physical AI
  - Agent system
- This is no longer one workload.

End-to-End Architecture

- **Architectures must be designed for target use cases.**
  - Different AI deployments have fundamentally different requirements

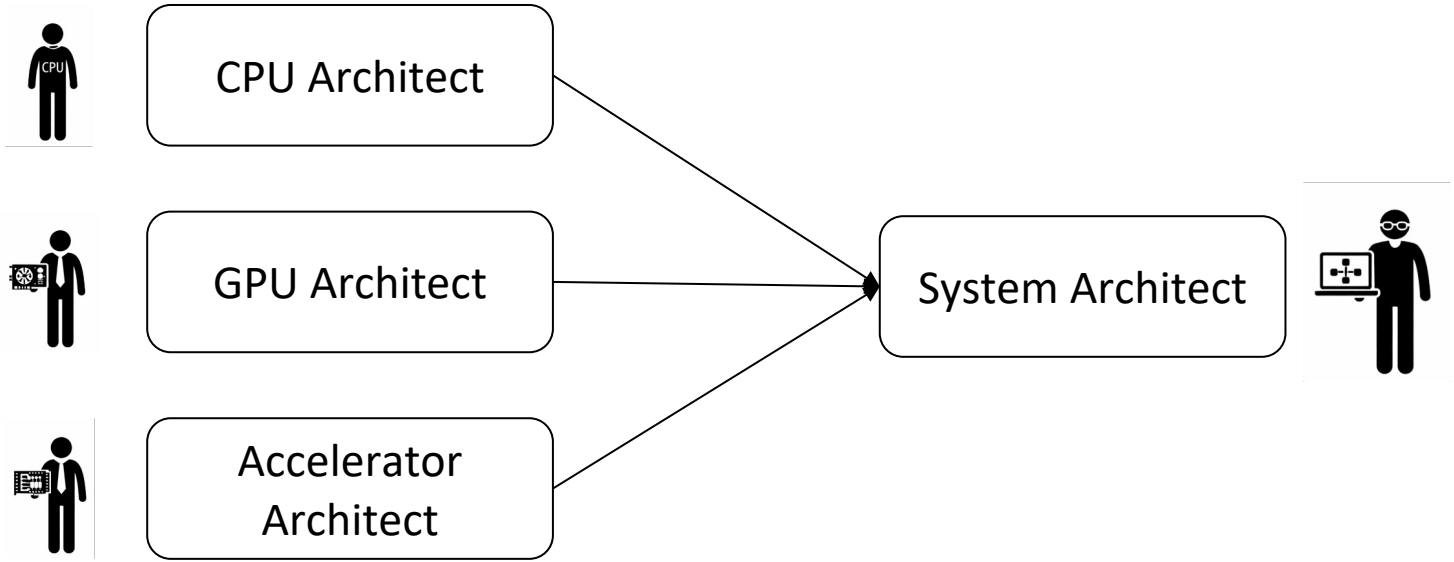
- **AI architectures in**
  - Training system -> throughput and scale
  - Inference system -> latency and efficiency
  - Edge systems -> energy and cost
  - Agent systems -> dynamic and stateful

- **At scale, specialization becomes economically viable.**



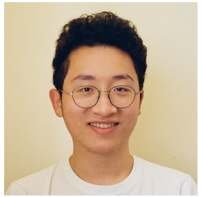
The Shift in Architecture

- Growing complexity in modern AI workflows requires heterogeneous components being integrated together.
- From IP Design -> System Design



The Need for System Architect

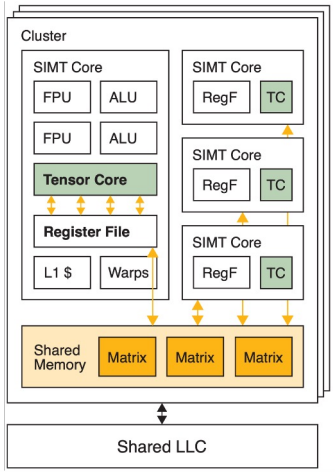
Hansung Kim*



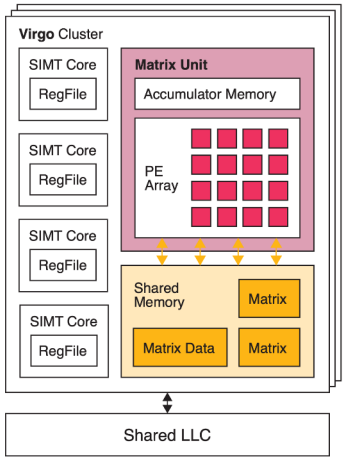
Richard Yan*

- Composition is now the architecture problem.

Core-coupled integration



Virgo: **Cluster-level integration**



Virgo **completely disaggregates** the matrix unit from the cores into a larger, unified design at the **cluster** level — Improving **scalability** and **energy efficiency**

https://github.com/ucb-bar/virgo

[ASPLOS'2025]

The Software Consequence

- Specialization breaks software…
  - Custom ISAs
  - Different execution models
  - Loss of compatibility

- AI may make it easier to build the SW flow for each HW, but not orchestration across HWs.

- While the hardware is diverging, software must unify it.

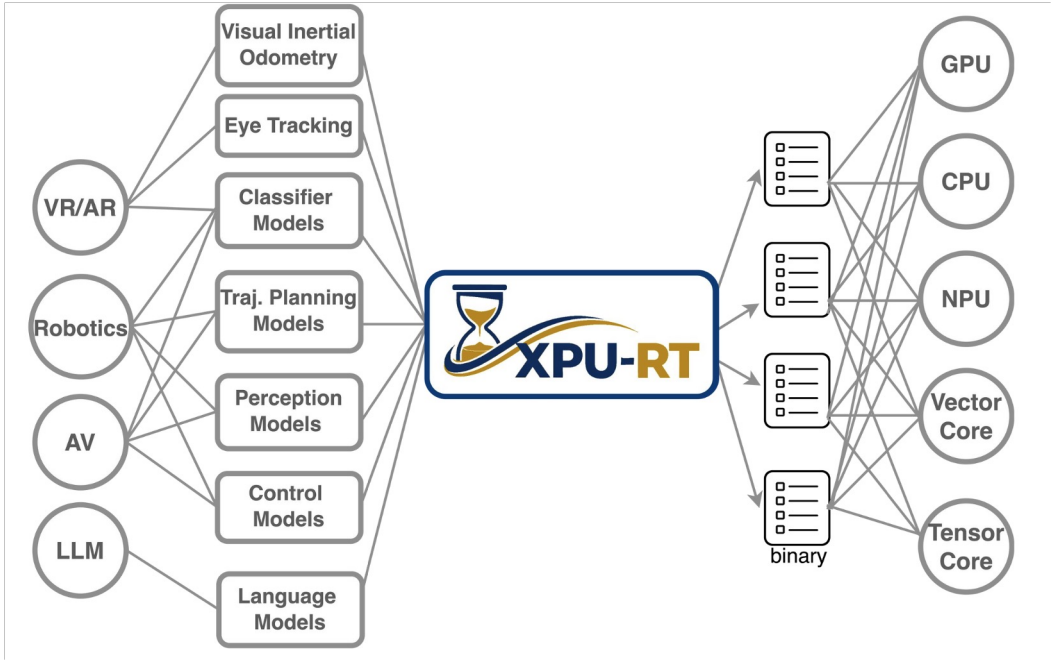
Portable System Abstraction


Kris Dong*


Agustin Hollmann*


Dima Nikiforov

- Dream:
  - Write once, map across systems, optimize per backend
- Need new abstraction for:
  - End-to-end system mapping
  - Hardware-aware scheduling
  - Cross-layer feedback
- Programming the entire system.



https://github.com/ucb-bar/XPU-RT

Back to the Beginning

- AI enables exploration
- Systems demand complexity
- Software requires abstraction
- Who connects all of this?

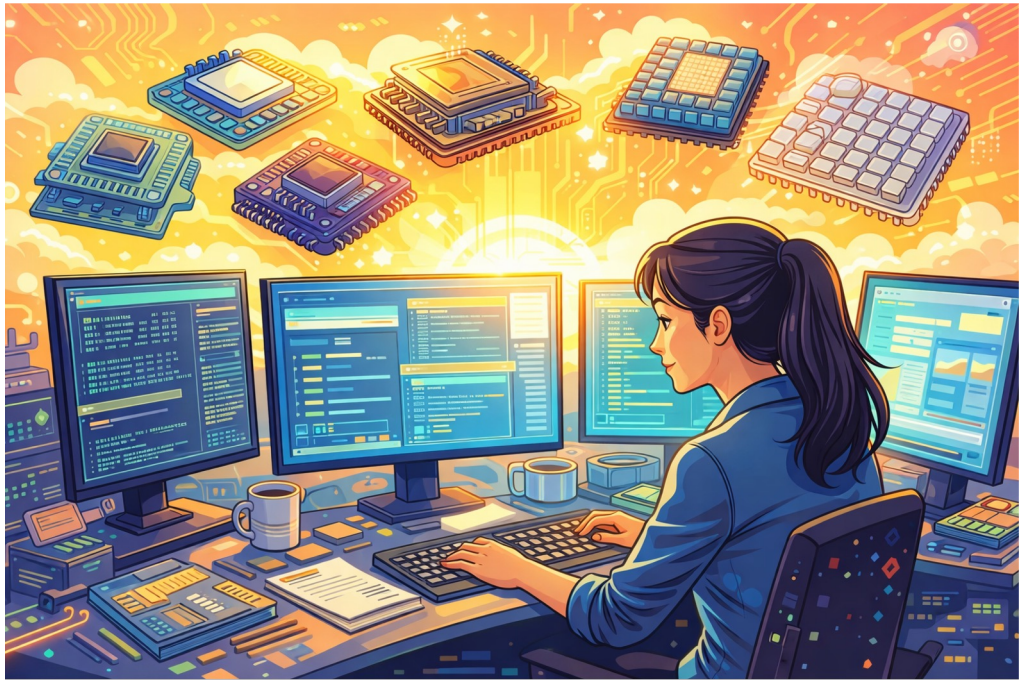
The New Golden Edge for the Computer Architect

The future belongs to architects

who can **think across layers**,

define abstractions, and **design**

end-to-end systems.



Acknowledgement

- Sponsored by
  - ME Commons,
  - DARPA,
  - NSF,
  - DOE,
  - Intel Rising Star Faculty Award,
  - Google Research Scholar Award,
  - AWS Build on Trainium,
  - BETR, SLICE and BWRC Industry sponsors!

PostDoc Researcher

Man (Amanda) Shi

Ph.D. Students

Shashank Anand

Yufeng Chi

Jun Sun Choi

Kris Dong

Prashanth Ganesh

Charles Hong

Coleman Hooper

Roger Hsiao

Hansung Kim

Dima Nikiforov

Richard Yan

Chengyi Lux Zhang