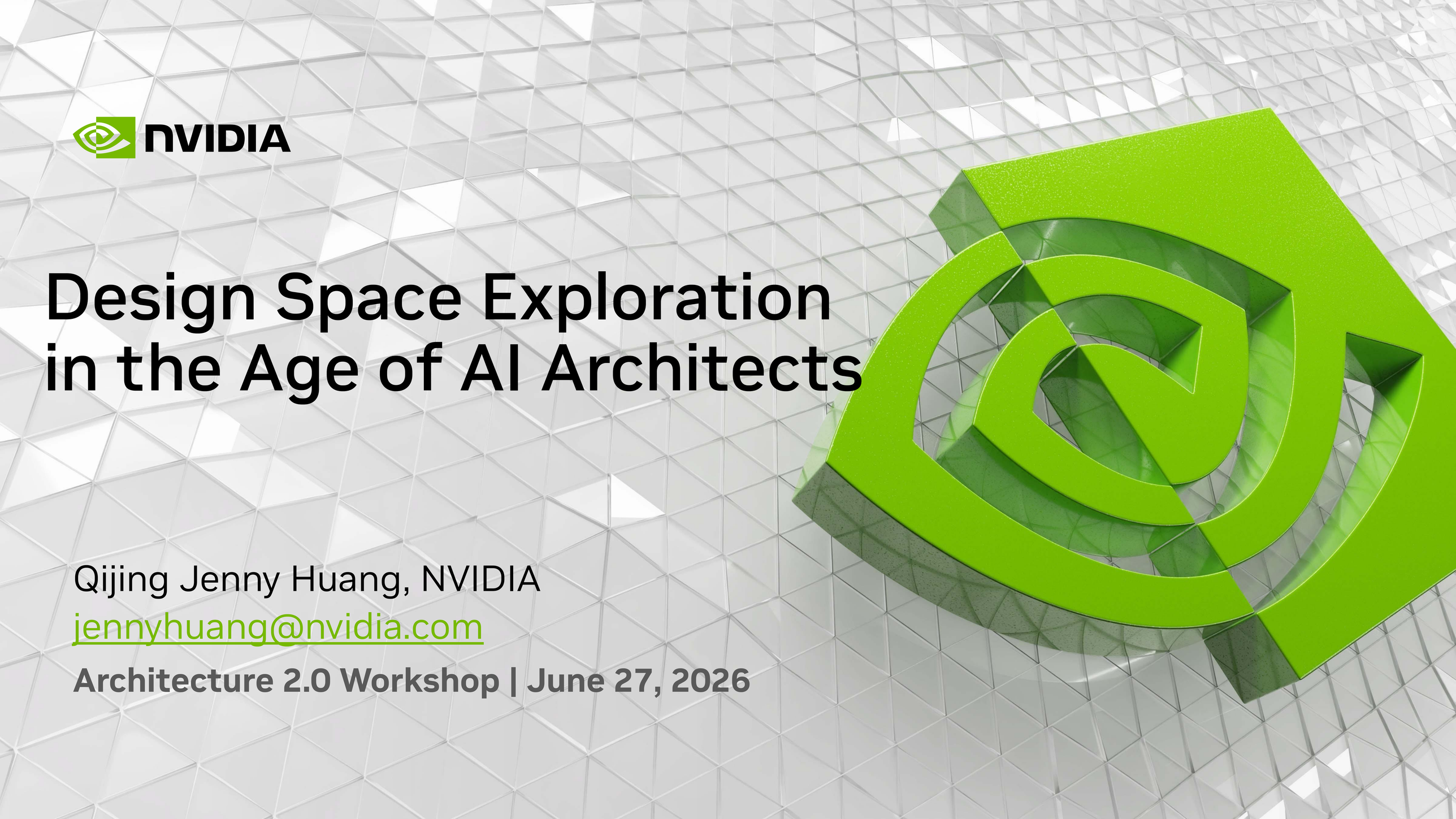


Design Space Exploration in the Age of AI Architects

Qijing Jenny Huang, NVIDIA
jennyhuang@nvidia.com

Architecture 2.0 Workshop | June 27, 2026

Acknowledgement

Jason Clemons
Sana Damani
Joel Emer
Siva Hari
Aditya Manish Kane
Christos Kozyrakis
Edward Lin
Sahil Modi
Angshuman Parashar
Karu Sankaralingam
Humphrey Shi
Athinagoras Skiadopoulos
Jingquan Wang
Zhifan Ye

and many others

Agenda

1. AlphaZero moment for AI architects
2. What does generative AI enable in DSE?
   - SOLAR
3. AI for arch vs “AI for X”
   - Lessons from AI for coding
4. Challenges towards AI architects

Core skills required for architects

Analytical Skills

- Find and distill knowledge from data

Optimization Skills

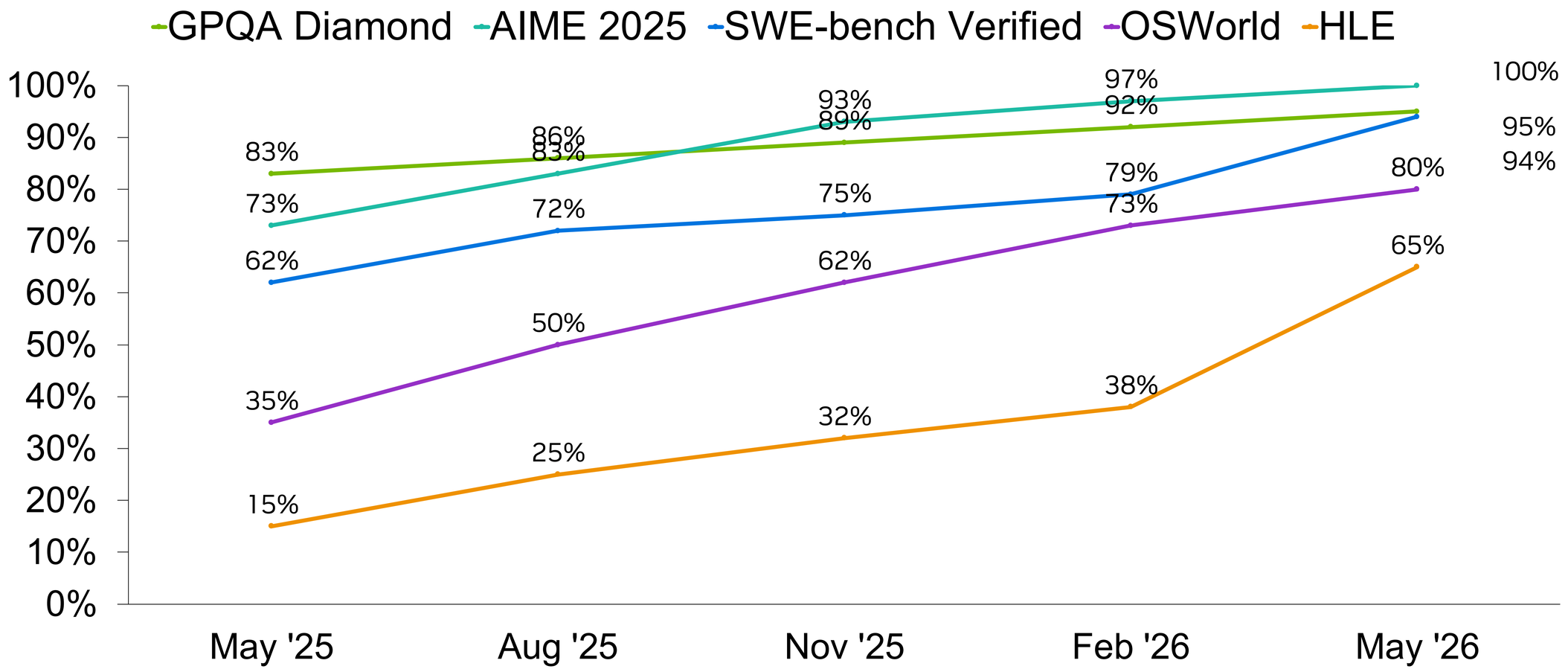
- Reason about useful attributes

Generative Skills

- Write code
- Suggest new ideas

AI is rapidly acquiring these skills

Best frontier scores, May 2025 – May 2026



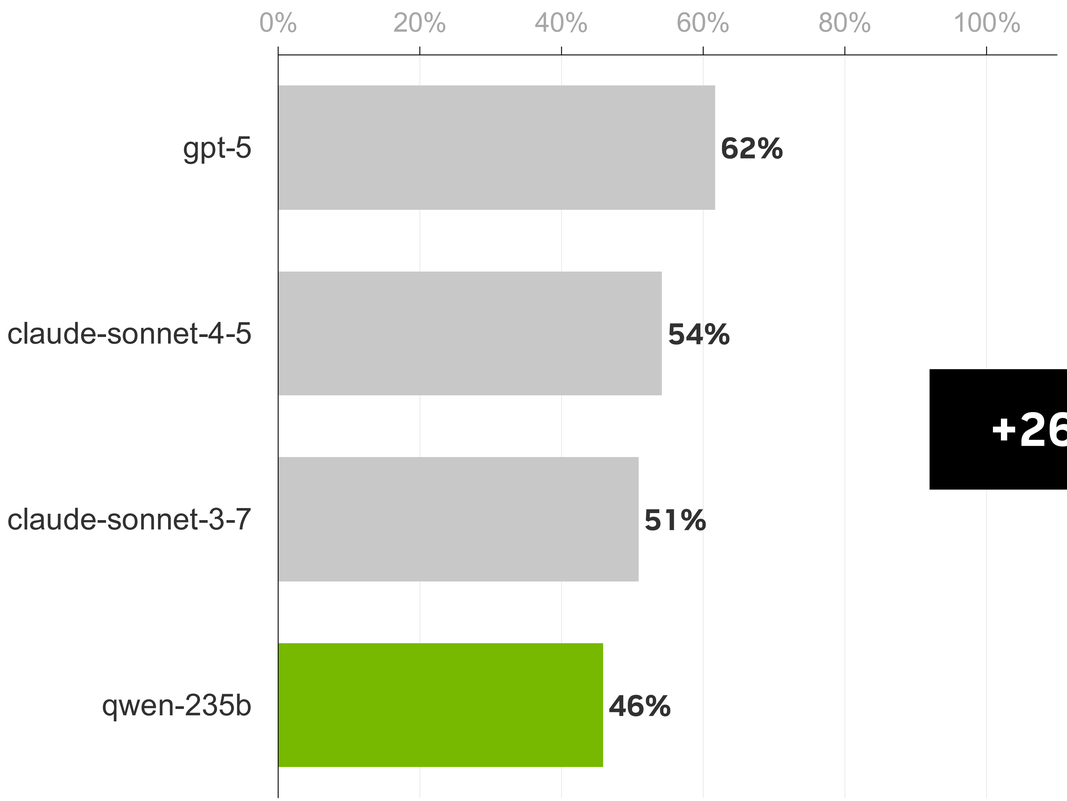
AI can write correct software and hardware

Pass Rate on Coding and Chip Design Benchmarks in May 2026

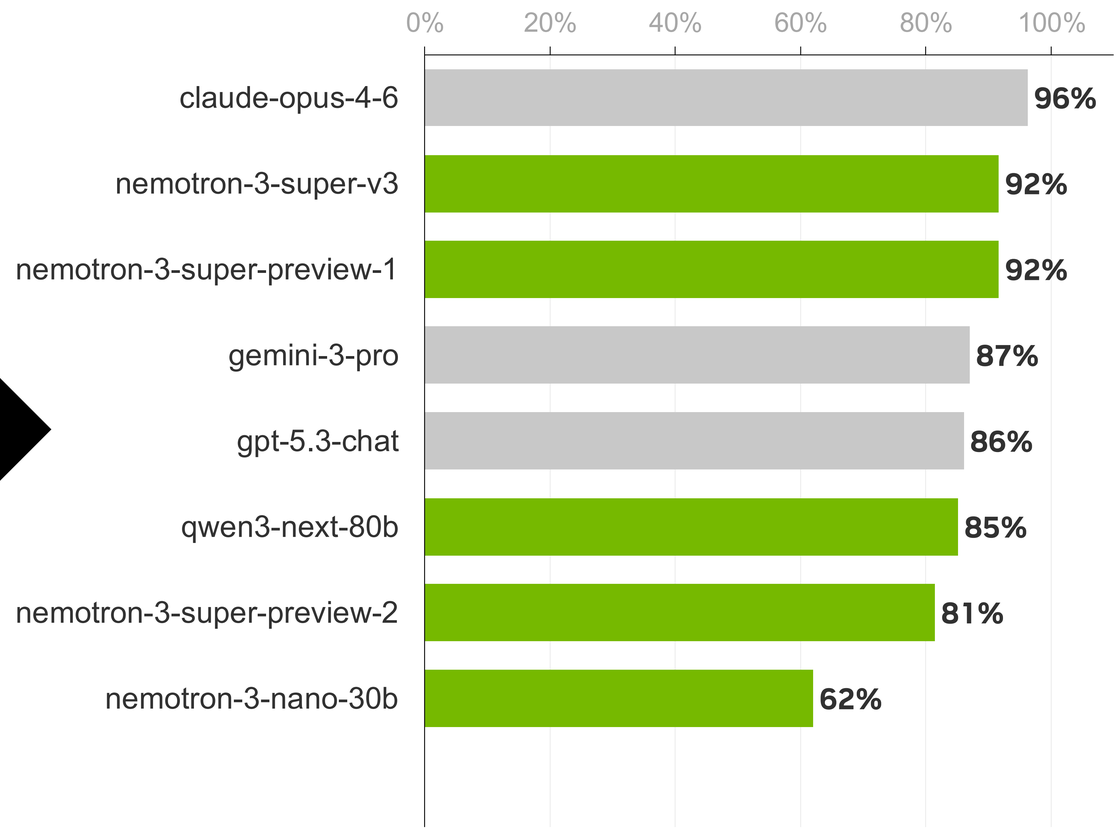
Benchmark	Scope	Pass Rate
KernelBench	250 GPU kernel tasks, 3 levels	**100%**
SOL-ExecBench	235 GPU kernels tasks, 3,957 workload specs	**100%**
VerilogEval v2	156 Verilog spec-to-RTL tasks	**97.4%**
CVDP	783 problems, 13 categories	**95.8%**
RTLLM v1.1	Spec-to-RTL	**72.9%**

AI can reason about performance

On our 108-question mapping-reasoning benchmark, top accuracy jumped from 62% to 96% in six months



+26%



Nov 2025

May 2026

** Evaluation by Jenny Huang and Dan Zhao.*

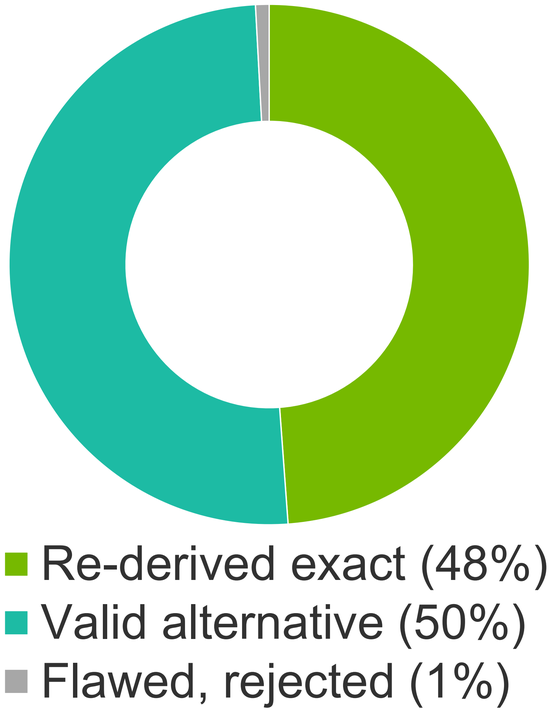
AI can generate high-quality ideas

Gauntlet shows:

- **High success rate:** 95%
- **Solution diversity:** 50% new ideas.
- **Speed:** 10–20 minutes per idea

95% viable

471 of 475 ideation runs · 95 papers × 5



Karu Sankaralingam, "Computer Architecture's AlphaZero Moment: Automated Discovery in an Encircled World," arXiv:2604.03312, 2026

AlphaZero moment for arch is approaching

Analytical Skills

- Find and distill knowledge from data

Optimization Skills

- Reason about useful attributes

Generative Skills

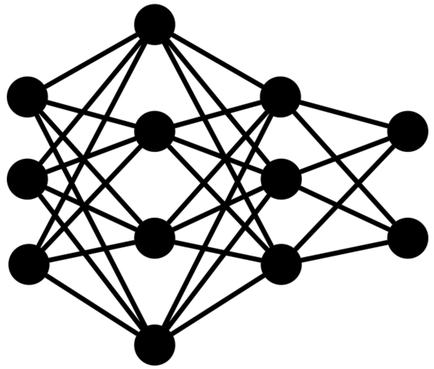
- Write SW&HW code
- Suggest new ideas

What does *Generative AI* enable beyond existing design space exploration (DSE) automation?

Traditional DSE confines to manually defined space

The classical parameter search is automated and powerful in this space

Algorithm



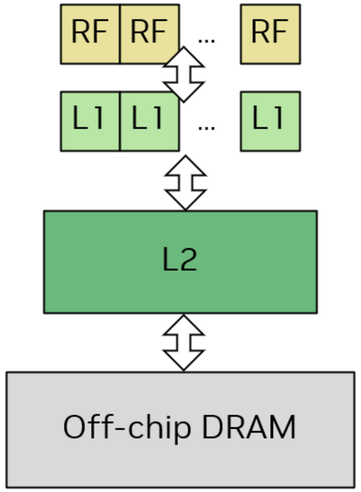
AI Models

10

Models
Transformers. Conv, LSTM, Mamba, …

×

Architecture Design Space

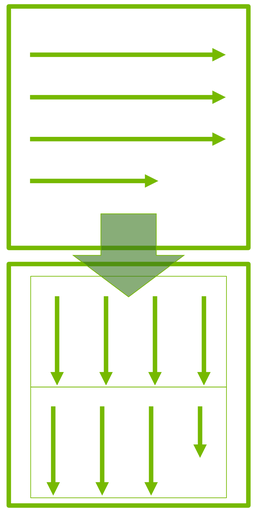


10^{17}

Parameter	# of Values
# of MACs	16384
Buffer sizes	131072
Network	8

×

Mapping Space



10^{5}

Parameter	# of Values
Temporal Factor	64
Spatial Factor	64
Loop ordering	8

×

Evaluation Time

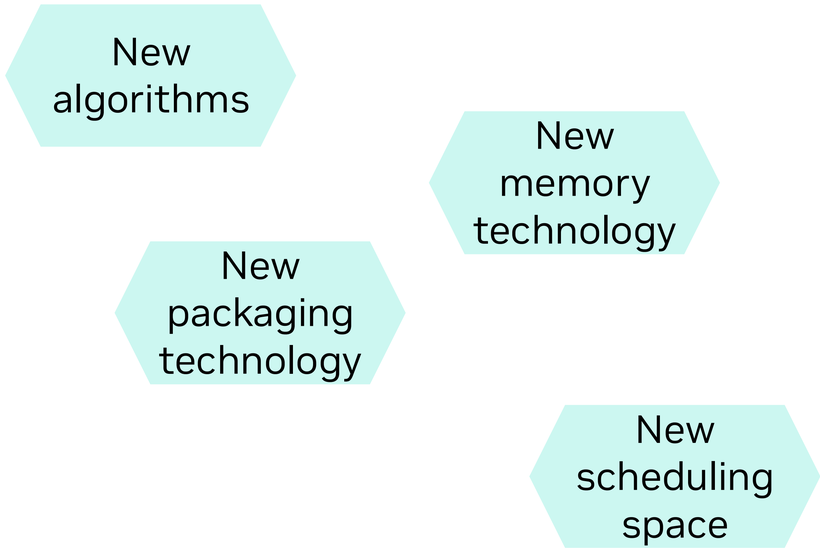
Tool	Eval Time
Timeloop	0.01s
FPGA	2 mins
VCS	10 mins
Power	6 hrs

0.01s

= 310T years

Generative AI expands the design space

- **Traditional DSE:** Human structured input and output space

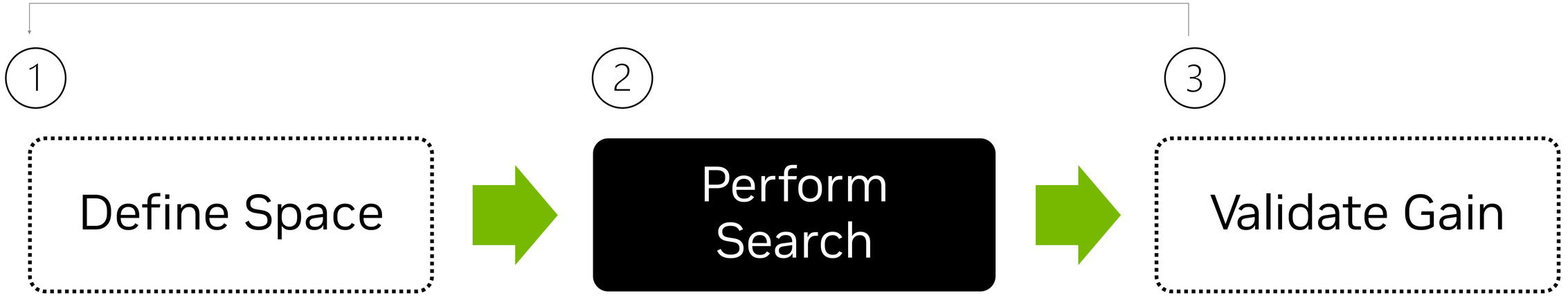


- Unformalized input and output space

Generative AI can automate more design tasks

1. perform search and opt

Generative AI can automate more design tasks

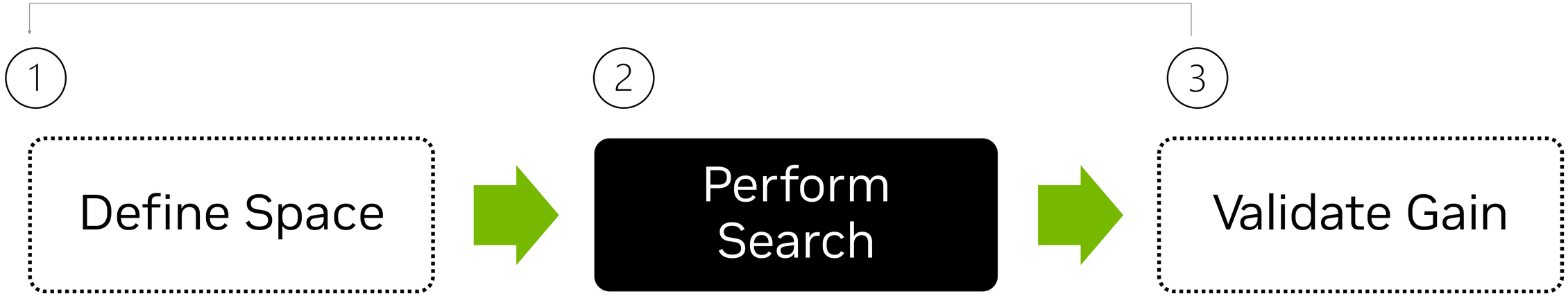


1.find killer apps
2.create benchmarks and metrics
3.research algo details
4.define hardware search space
5.define mapspace

1. perform search and opt

1.select evaluation approaches and fidelities
2.develop and validate performance models OR develop and validate real HW+SW systems
3.map design space and config to evaluator input

Generative AI can automate more design tasks

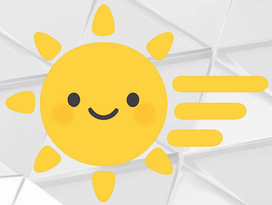


1.find killer apps
2.create benchmarks and metrics
3.research algo details
4.define hardware search space
5.define mapspace

1. perform search and opt

1.select evaluation approaches and fidelities
2.develop and validate performance models OR develop and validate real HW+SW systems
3.map design space and config to evaluator input

Example leveraging AI to bridge the automation gaps in DSE: *SOLAR*



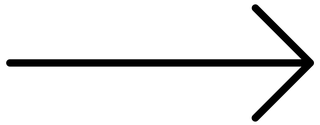
SOLAR: AI-Powered Speed-of-Light Performance Analysis

Jenny Huang, Sana Damani, Zhifan Ye, Athinagoras Skiadopoulos, Siva Hari, Jason Clemons, Sahil Modi, Jingquan Wang, Aditya Manish Kane, Edward Lin, Humphrey Shi, Christos Kozyrakis

Paper Preprint and *Github Repo*

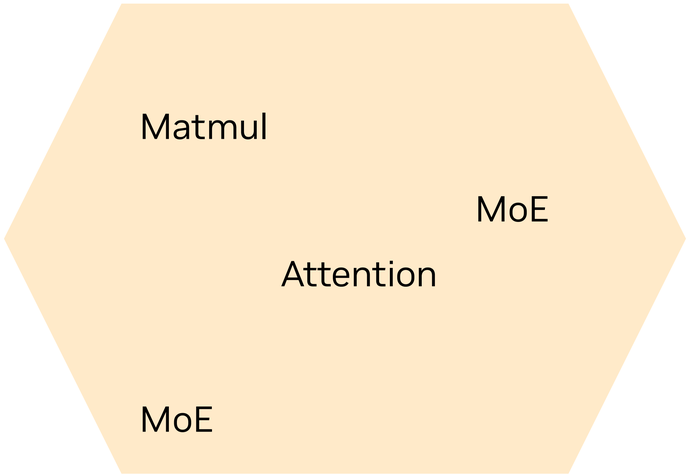
The automation gap in performance modeling

Algorithms implemented in different input languages



Performance model that models specific constructs

Intense and repetitive manual efforts, can be error prone



Target use case: speed-of-light (SOL) performance analysis

KernelBench problems for CUDA codegen

PyTorch Perf: 10s
Claimed CUDA Perf: 0.1s

SOL Perf: ??? s

Key Questions:

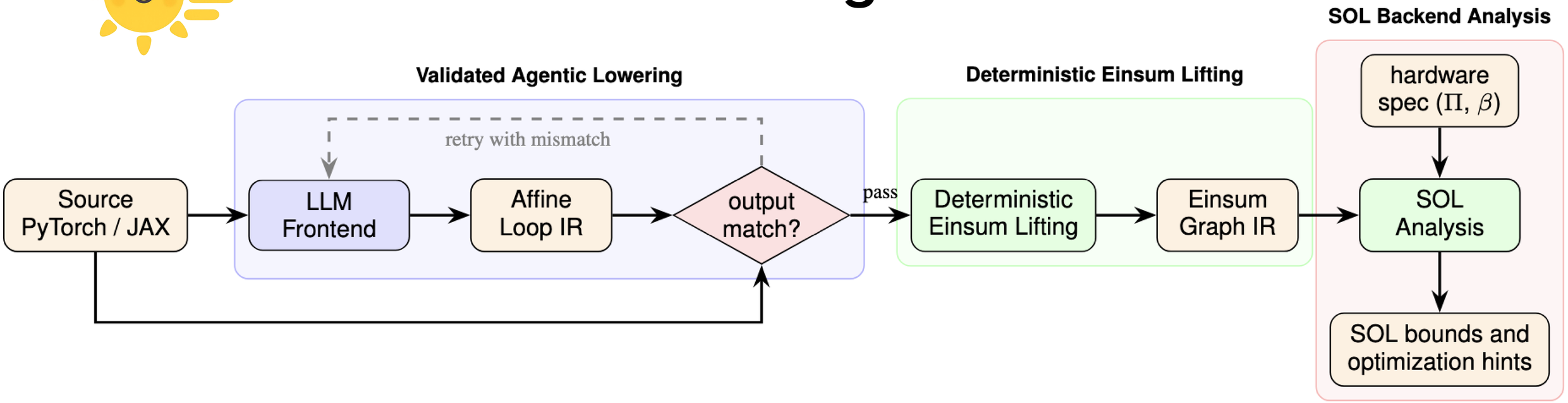
*1.Is a **100x** speedup achievable?*
2.What headroom remains vs. SOL?
3.How do we reach SOL?

SOL analysis helps us to:

1.Validates Results
2.Set Return of Investment (ROI)
3.Suggest Optimizations

Anne Ouyang et al., "KernelBench: Can LLMs Write Efficient GPU Kernels?," arXiv:2502.10517, 2025

Three-Phase Agentic Flow

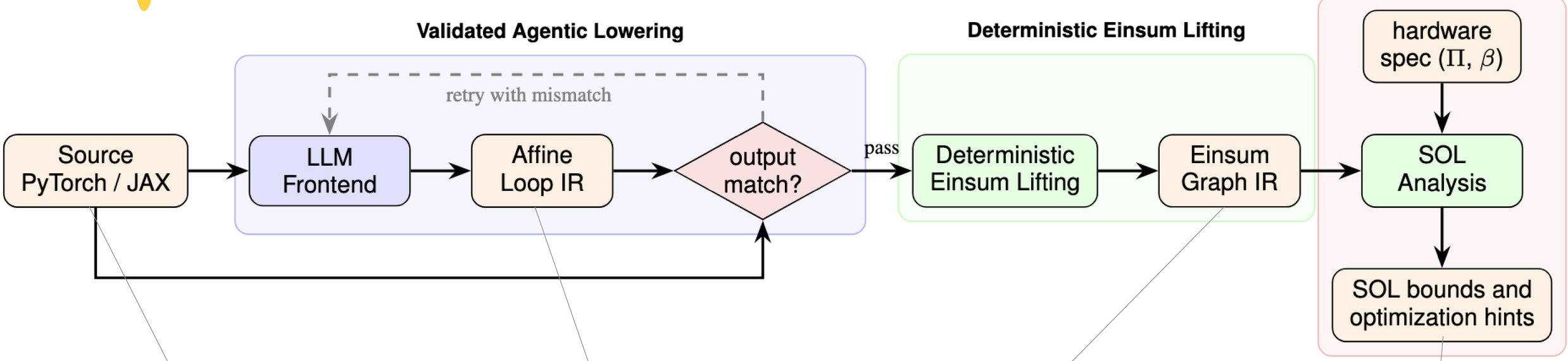


A language-agnostic agentic workflow that translates an input program into an einsum graph for SOL performance modeling:

1. **Agentic LLM frontend:**
   - *source → an executable Affine Loop IR*
   - *validate through output comparison*
2. **Deterministic lift:** Affine Loop IR → Einsum Graph
3. **Analytical backend:** Einsum Graph → Performance Models

Qijing Huang et al., "SOLAR: AI-Powered Speed-of-Light Performance Analysis," arXiv:2606.26383, 2026

Three-Phase Agentic Flow



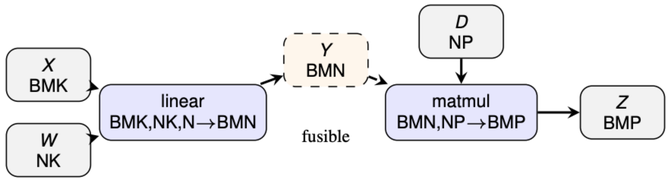

```
class Model(nn.Module):
    def __init__(self):
        self.fc = nn.Linear(256, 512, bias=True)
        self.D = nn.Parameter(torch.randn(512, 64))

    def forward(self, x):
        y = self.fc(x)
        return torch.matmul(y, self.D)

def get_inputs():
    """Return␣[batch=2,␣seq=128,␣hidden=256]."""
    torch.manual_seed(0)
    return [torch.randn(2, 128, 256)]
```

```
@kernel
def linear(X: Tensor("B,M,K"), W: Tensor("N,K"),
           Bias: Tensor("N")) -> Tensor("B,M,N"):
    Y = Tensor(B=X.shape[0], M=X.shape[1], N=W.shape[0])
    for b in Dim("B", parallel=True):
      for m in Dim("M"):
        for n in Dim("N"): Y[b, m, n] = Bias[n]
    for b in Dim("B", parallel=True):
      for m in Dim("M"):
        for n in Dim("N"):
          for k in Dim("K"): Y[b, m, n] += X[b, m, k] * W[n, k
              ]
    return Y

@kernel
def matmul(Y: Tensor("B,M,N"), D: Tensor("N,P")) -> Tensor(
      "B,M,P"):
    Z = Tensor(B=Y.shape[0], M=Y.shape[1], P=D.shape[1])
    for b in Dim("B", parallel=True):
      for m in Dim("M"):
        for p in Dim("P"):
          for n in Dim("N"): Z[b, m, p] += Y[b, m, n] * D[n, p
              ]
    return Z
```



Rank Sizes: B=2, M=128, K=256, N=512, P=64

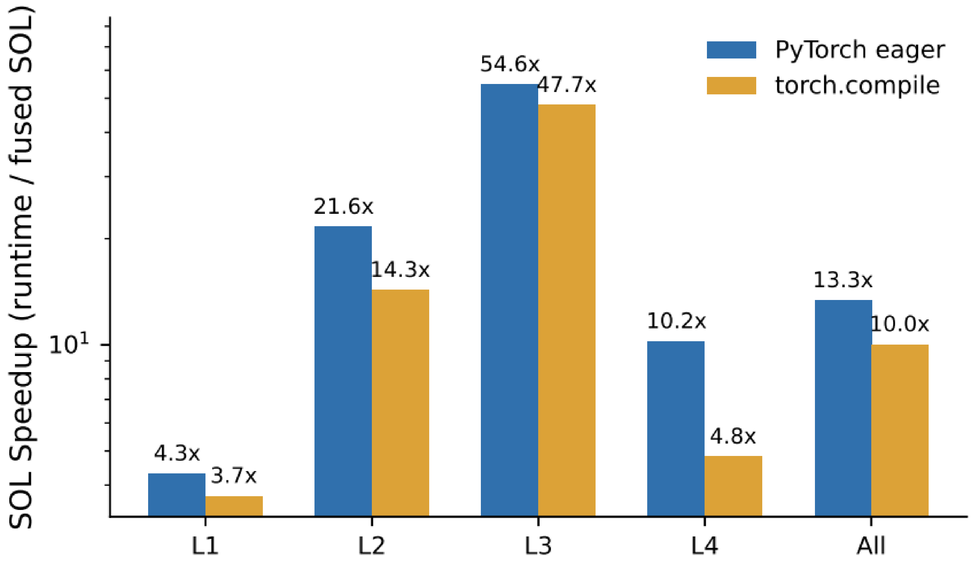
Node	Einsum	MACs	Memory
linear#0	BMK,NK,N→BMN	BMKN	BMK+NK+N+BMN
matmul#0	BMN,NP→BMP	BMNP	BMN+NP+BMP

Mode	Compute	Memory	SOL	Bottleneck
Unfused	83.9 MFLOP	2.03 MB	1.00 μs	Memory
Fused	83.9 MFLOP	0.99 MB	0.82 μs	Compute

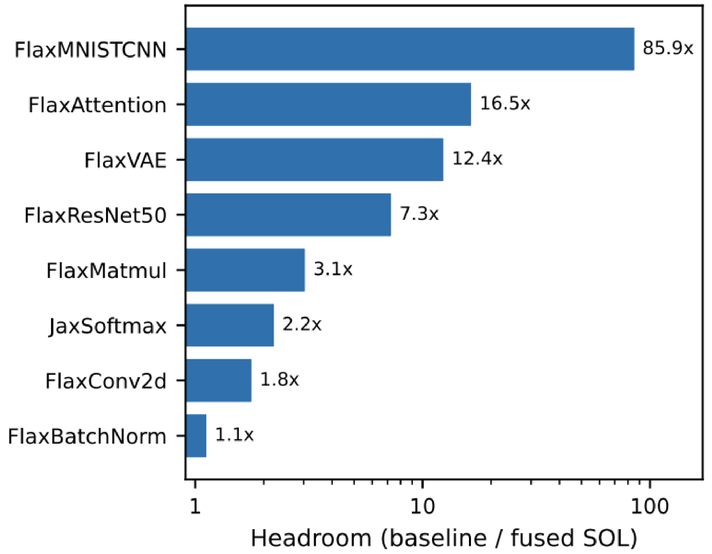
AI expands operator and language coverage

Enabling auto headroom analysis

SOLAR automates the SOL headroom for 100% KernelBench problems in PyTorch and example programs in JAX.



KernelBench Headroom (L1-L4)



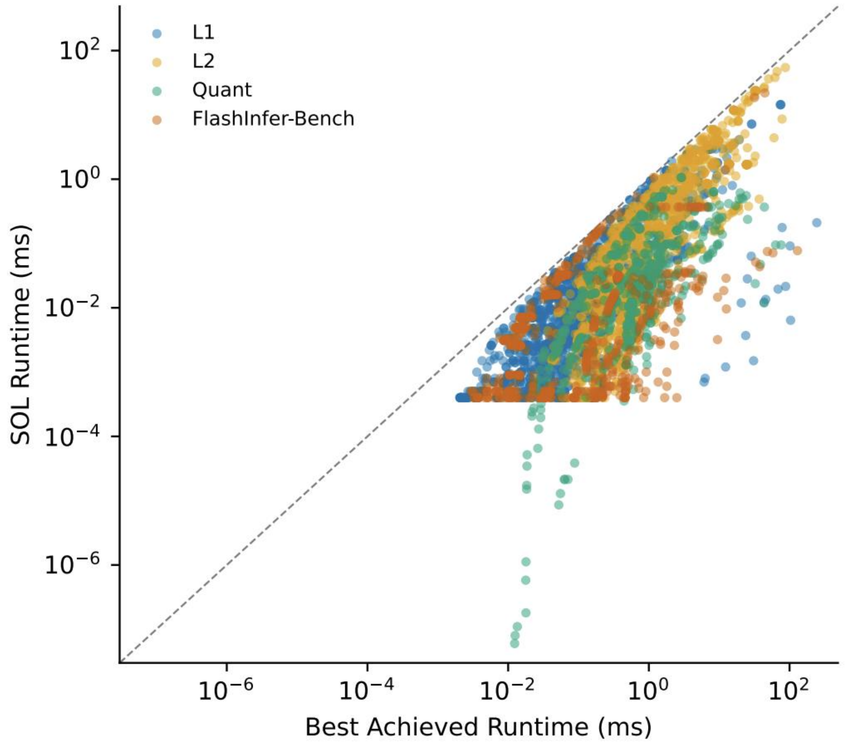
JAX Headroom

AI expands operator and language coverage

Enabling auto headroom analysis

SOLAR is used in SOL-ExecBench for automatic the SOL score derivation.

SOL vs. best achieved runtime on SOL-ExecBench

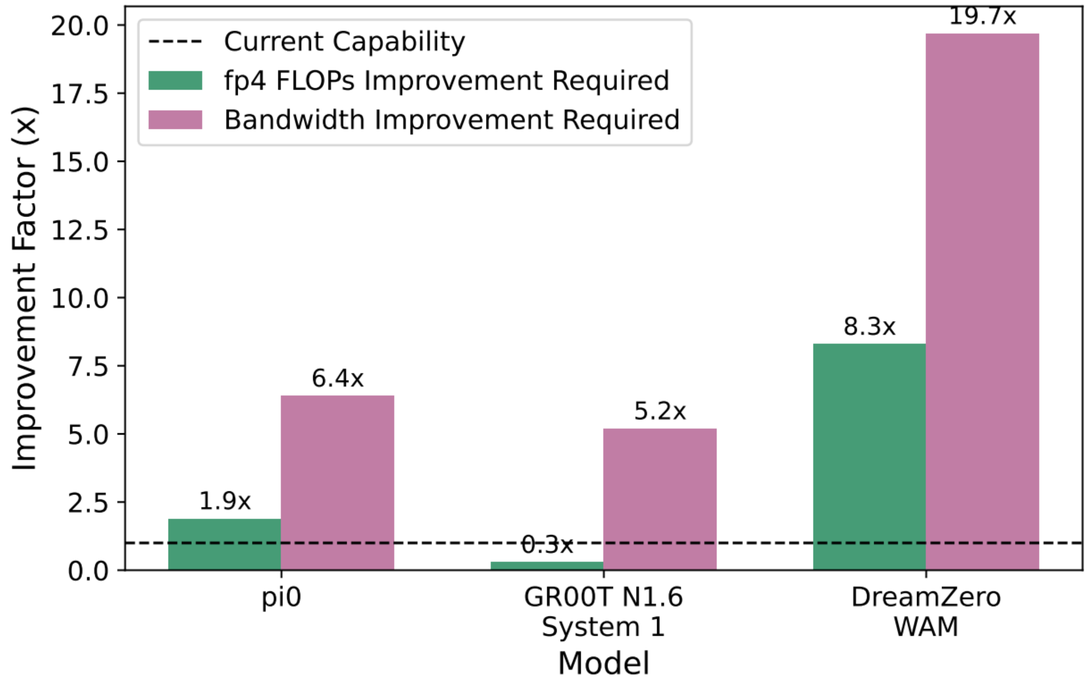


Edward Lin et al., "SOL-ExecBench: Speed-of-Light Benchmarking for Real-World GPU Kernels Against Hardware Limits," arXiv:2603.19173, 2026

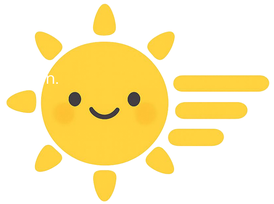
AI expands operator and language coverage

Enabling auto hardware requirement derivation

SOLAR is also applied to analyze robotics workloads and derive the hardware requirement for meeting real-time latency target.



HW improvement required for real-time inference



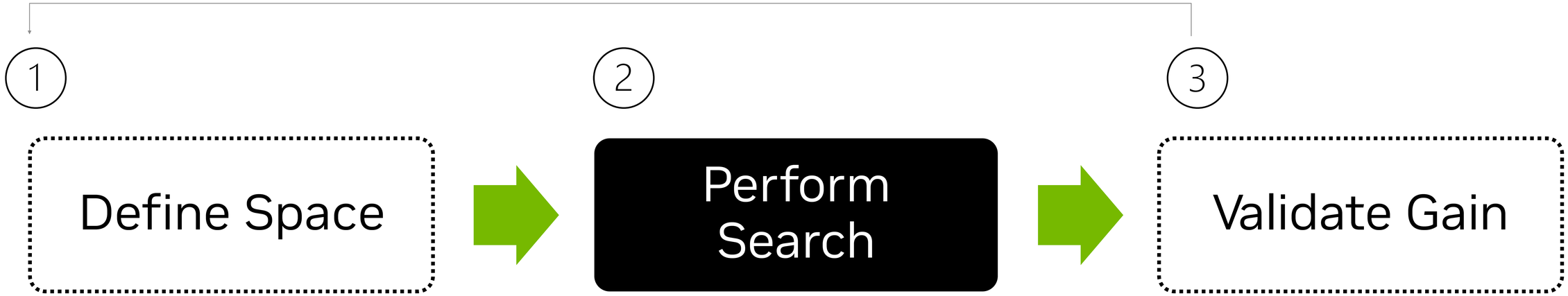
Lessons learned

1. **AI can handle broader and less structured inputs than traditional SW flow**
   - This is especially useful for real-world applications

2. **AI translation can be validated via results comparison**
   - Executable output specs and result comparison help to increase confidence.

3. **It is useful to choose carefully between generative and deterministic solutions in real software flow**
   - AI still can be costly, error-prone, and hard to reproduce.
   - We should decide where AI adds value and where deterministic flow is needed for reliability, repeatability, and scalability.

Generative AI can automate more design tasks



1.find killer apps
2.create benchmarks and metrics
3.research algo details
4.define hardware search space
5.define mapspace

1. perform search and opt

1.select evaluation approaches and fidelities
2.develop and validate performance models OR develop and validate real HW+SW systems
3.map design space and config to evaluator input

The remaining process is still largely human-driven

More automation opportunities

Create Metrics and Benchmarks

Identify performance goals, create small examples for fast evaluation

Define HW Space

Clarify technology assumptions, memory hierarchy, interconnect, and power and thermal constraints

Define Mapping Space

Specify tiling, fusion, layout, ordering, parallelism, work item granularity and runtime policy

Develop and Validate Performance Model

A performance model chooses

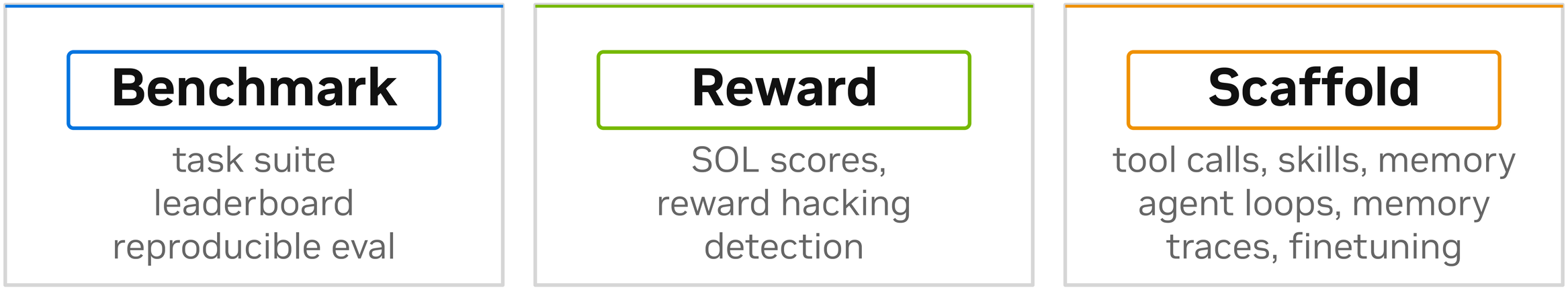
which effects matter,

which effects are ignored,

and where uncertainty enters.

How does AI for arch differ from AI for X? (e.g., AI for Coding)

AI-for-Coding As A Playbook



Benchmarks

make progress measurable

Rewards

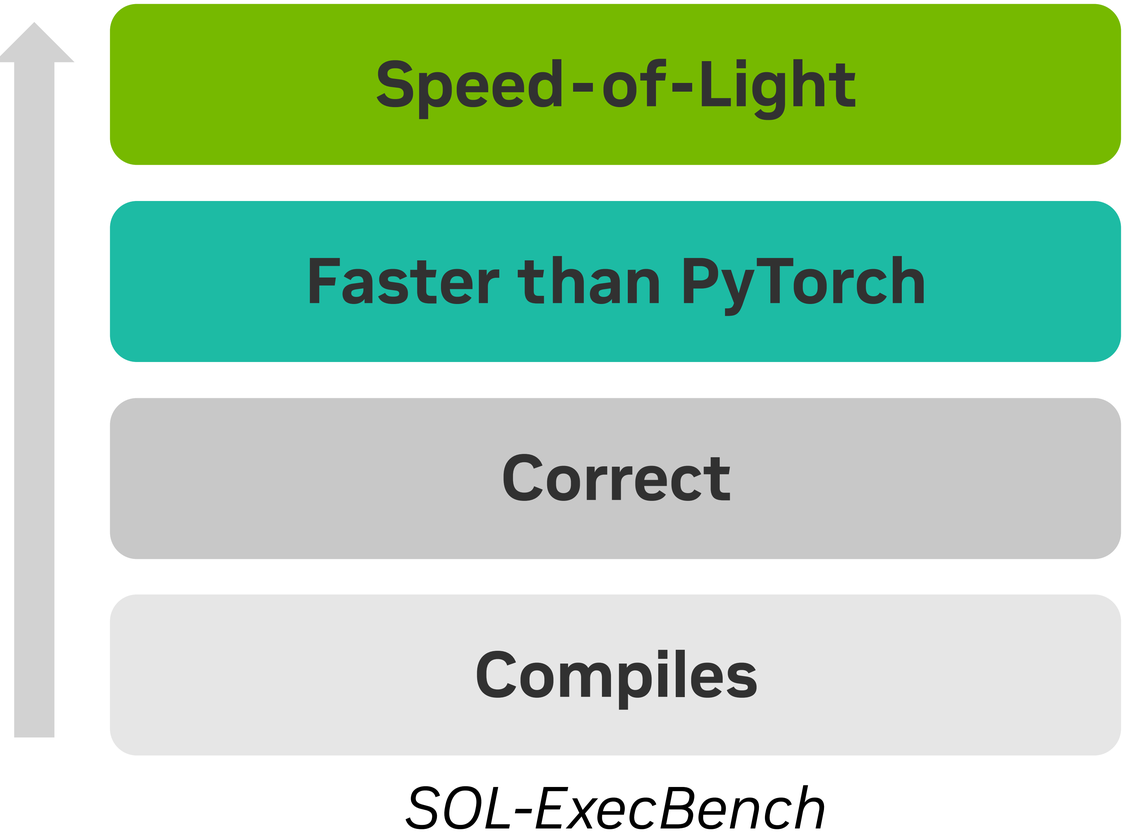
turn execution into signal

Scaffolds

tools, search, memory, repair

Benchmarks: Measure AI Progress

- **Good practice**
  - Samples real workloads in different precision
  - Provides the hardware Speed-of-Light (SOL) targets
  - Hosts and maintains a leaderboard



Edward Lin et al., "SOL-ExecBench: Speed-of-Light Benchmarking for Real-World GPU Kernels Against Hardware Limits," arXiv:2603.19173, 2026

Reward Metrics: Anchor to Speed-of-Light

Good practice

- Reports the percent of SOL reached (**SOL-ExecBench**)
- Uses the SOL signal to improve token efficiency. (**µCUTLASS**)

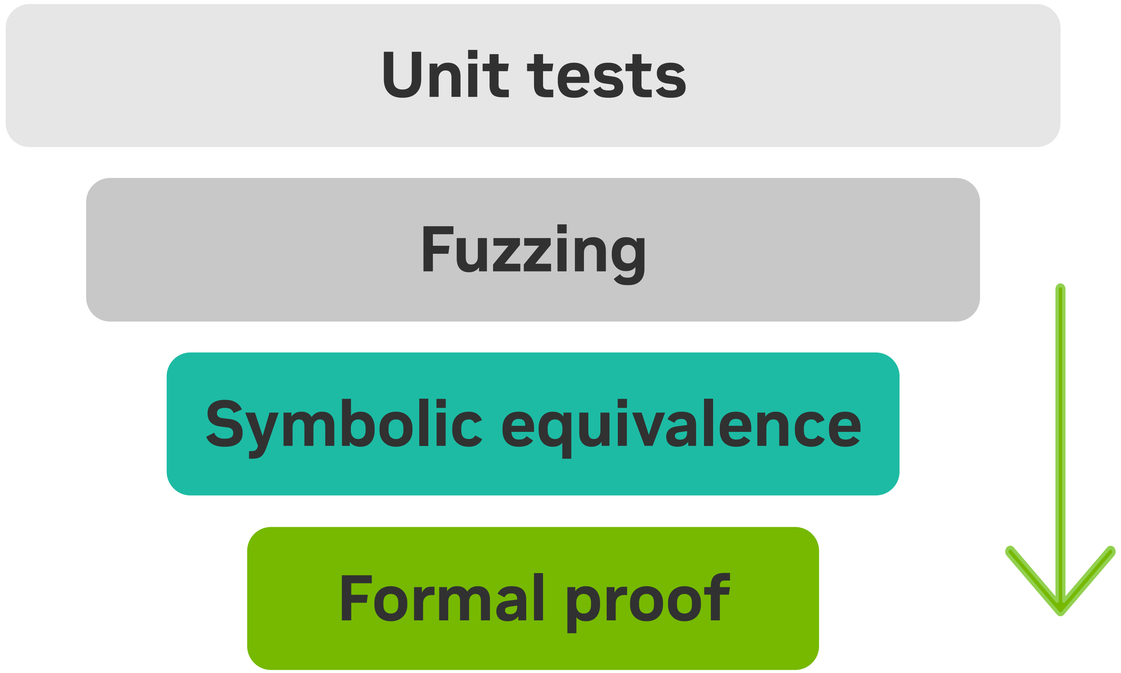
How close to the hardware limit?

SOL-ExecBench (arXiv:2603.19173) · µCUTLASS (arXiv:2603.29010) · SOLAR (arXiv:2606.26383)

Validation: Trust, but Verify

- **Good practice**
- A passing unit test is not a correct kernel on its own
- Fuzzing and symbolic equivalence stress wide inputs and prove math equivalence
- Formally prove is even more thorough (**ProofWright**)
- Leverages DSL features is another alternative (**µCUTLASS**)

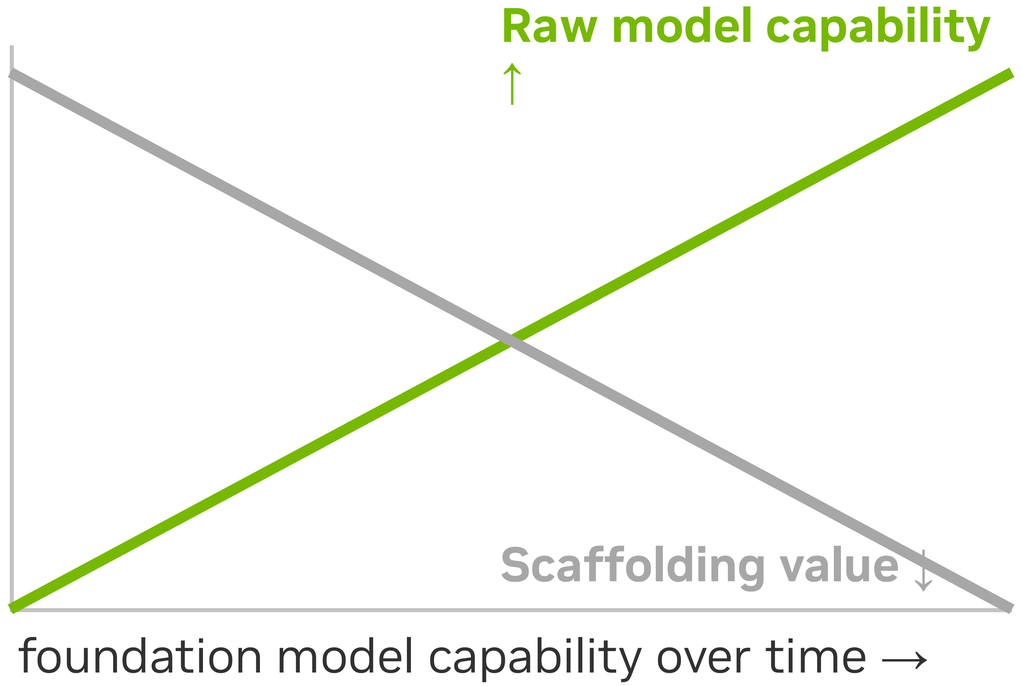
Catch what tests miss



ProofWright (arXiv:2511.12294) · µCUTLASS (arXiv:2603.29010)

Search & Agents: Scaffolding That May Fade

- **Tool calling** lets agents compile, run, and profile kernels in the loop instead of guessing
- **Skills** give the agent reusable optimization playbooks to call on demand
- **Evolutionary search** (**AlphaEvolve**, **EvoEngineer**) breeds better kernels over many candidates
- **Multi-agent loops and RL** add **STARK**-style critique and repair, rewarding correctness and speedup
- **As base models improve** though, much of this scaffolding fades



AlphaEvolve (arXiv:2506.13131) · *STARK (arXiv:2510.16996)* · *GEAK (AMD, 2025)*

Training & Finetuning: Collect Data, Own Your Model

- **Proprietary access is fragile** since an API can be rate-limited, deprecated, or priced out
- **A durable path** is to collect data plus in-house fine-tuning but it is costly (**DeepCoder-14B**)

Proprietary API

- Rate-limited
- Deprecated overnight
- Priced out

✗ **fragile dependency**

vs

Open + Fine-tuned

- Own your data
- Own your model
- Own your roadmap

✓ **durable advantage**

DeepCoder-14B (Together AI & Agentica, 2025) · arXiv recipe: DeepScaleR/DAPO GRPO+

Key Difference 1: abstractions are less settled

Coding has firmer contracts than early-stage architecture

Software	Hardware	Architecture DSE
defined semantics fast tests clear diffs	many levels many tools many constraints	moving objective proxy models expert priors
APIs, tests, compilers	ISA, RTL, GDS, timing	requirements, proxies, tradeoffs

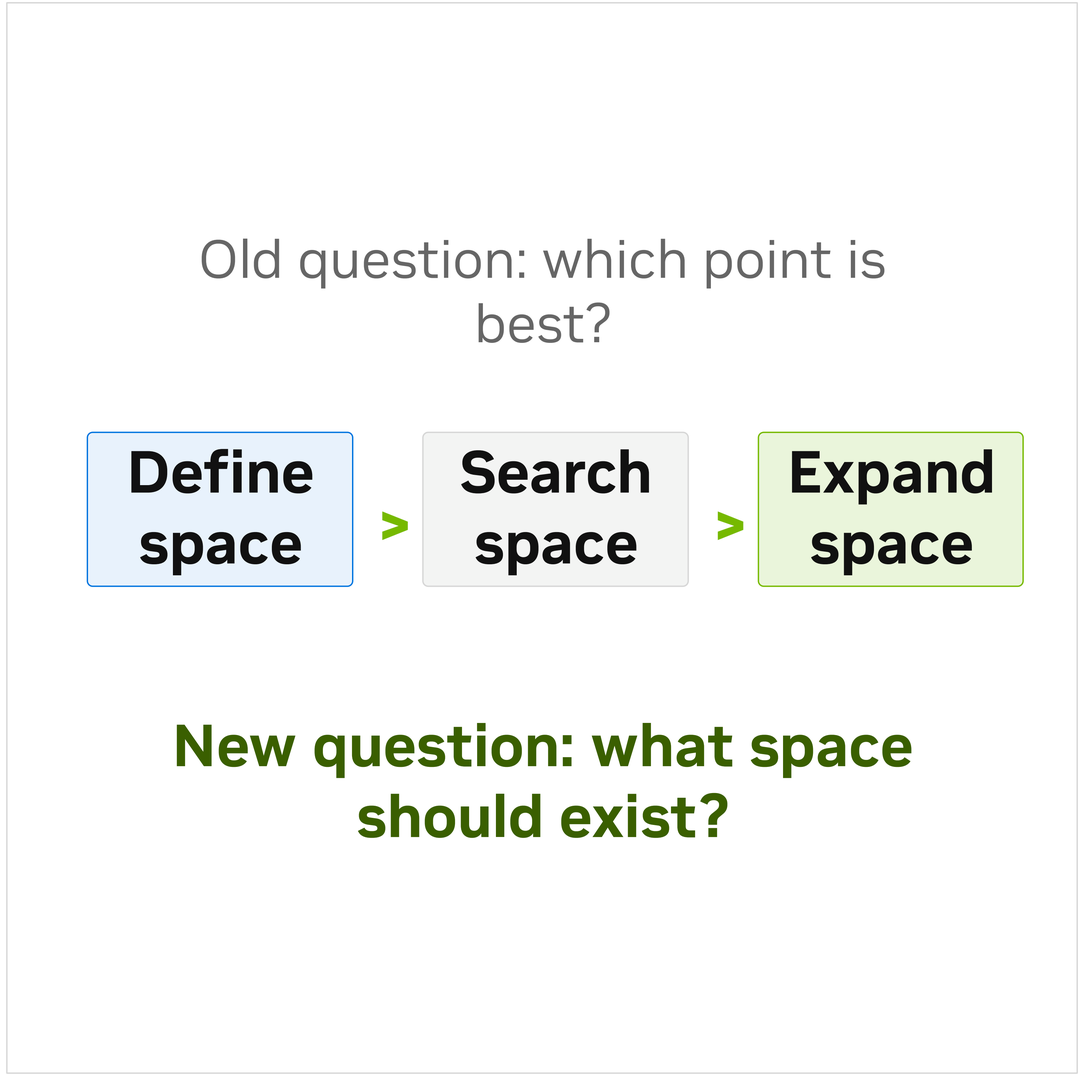
Key Difference 2: the search space moves

The agent may change the workload,

the mapping, the architecture,

and the technology assumptions

at the same time.



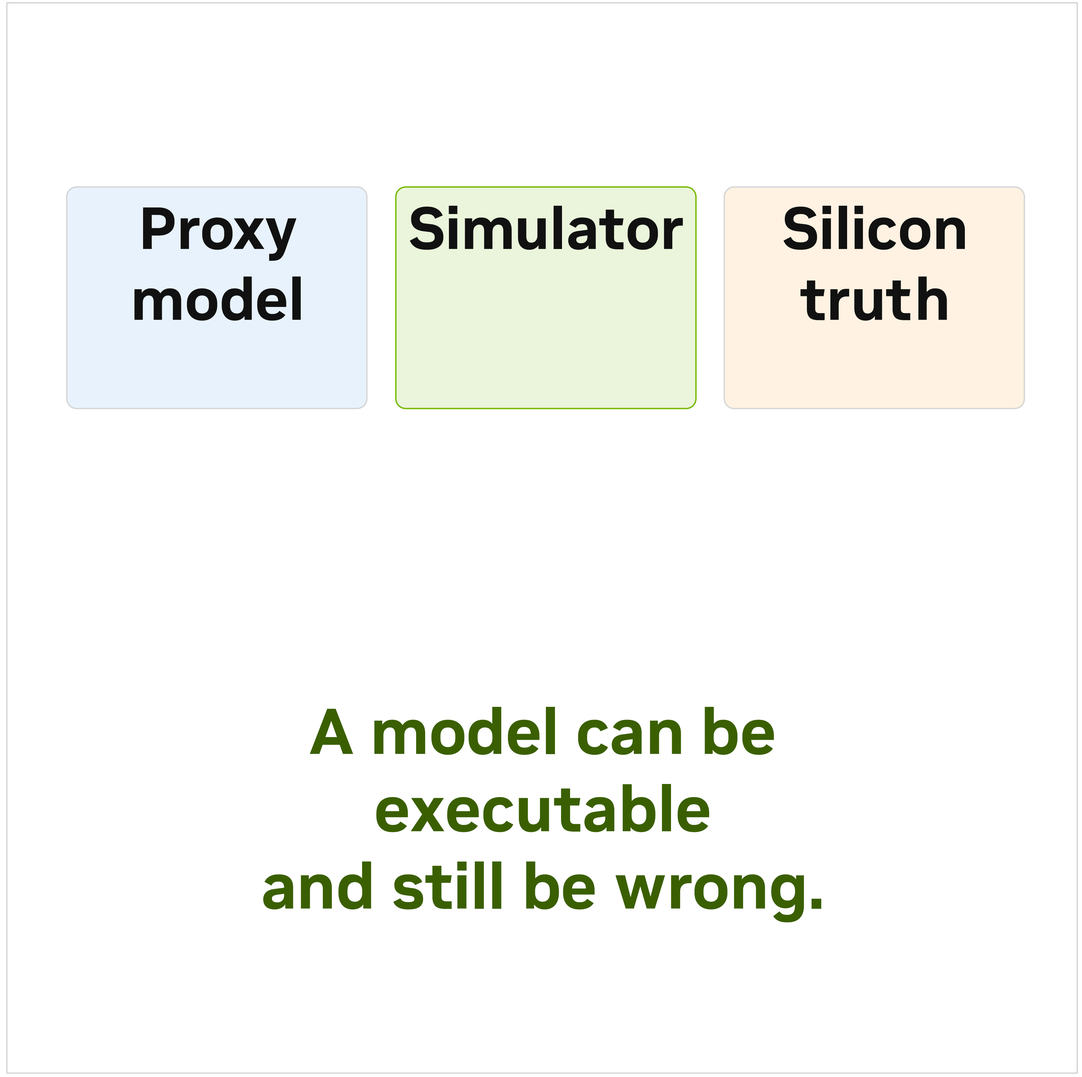
Moving design space

Key Difference 3: validation signal is hard

For new hardware and new mappings,

we may not know whether the proxy

is faithful until much later.



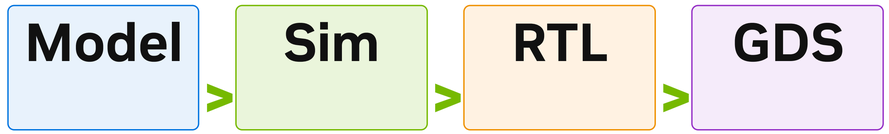
Proxy to truth gap

Difference 4: the feedback loop is long

Simulation, synthesis, place-and-route,

and silicon feedback are orders **slower**

than a unit test.



We need fast imperfect signals that eventually reconcile with slow truthful ones.

Feedback latency

The call: innovate in AI-native DSE infrastructure

01

benchmarks for formalization tasks

02

metrics and perf models for AI-generated designs

03

validators and reward-hacking detection

04

shared data, tool-skills, and agent loops

Let’s build creative, trustworthy AI architects for scalable DSE.

Thank you!

hqjenny@nvidia.com