AgentDSE: Reasoning-Augmented Architectural Design Space Exploration

Chenyu Wang[1,*], Jiahe Caroline Shi[2,*], David Kong[1], Duane S. Boning[2],
Zishen Wan[1,3,†], Yilun Du[1], Vijay Janapa Reddi[1]
[1]Harvard University [2]Massachusetts Institute of Technology [3]Columbia University

***Abstract*—Traditional architectural design space exploration (DSE) is highly inefficient, typically requiring tens of thousands of simulator evaluations across various optimization methods. This inefficiency arises because conventional methods treat the simulator as a black-box oracle. In contrast, human architects effectively guide exploration by reasoning through physical constraints, performance bottlenecks, data reuse, and workload structures. To bridge this gap, we introduce AgentDSE, a simulator-in-the-loop methodology driven by a general-purpose large language model (LLM) coding agent. AgentDSE automates this architectural-reasoning loop without requiring model fine-tuning, precomputed design databases, or domain-specific optimizer code. Across deep neural network (DNN) accelerator mapping, hardware/software co-design, and CPU cache-hierarchy optimization, AgentDSE achieves competitive or better design quality with up to two orders of magnitude fewer evaluations. AgentDSE also produces inspectable traces that surface architectural hypotheses, performance cliffs, implicit priors, and simulator artifacts, making every search decision traceable rather than buried in optimizer state.**

I. Introduction

Modern architectural design space exploration (DSE) operates over vast, discrete, and highly discontinuous spaces. A single DNN accelerator mapping may require choosing dataflows, tiling factors, parallelization dimensions, and processing-element allocations from a combinatorial space that can exceed 10^{20} configurations [9], [11], [13], [16]. CPU cache-hierarchy optimization and accelerator hardware/software co-design exhibit similar structure, where each design point is constrained by hardware resources, workload behavior, and simulator-specific validity conditions. Traditional DSE methods, including random search (RS), genetic algorithms (GA) [9], [20], Bayesian optimization (BO) [1], [17], [19], reinforcement learning (RL) [12], and differentiable DSE methods [8], introduce increasingly sophisticated samplers or surrogates over numeric design spaces. Yet these methods largely preserve the same black-box optimization paradigm: the simulator is queried for scalar feedback, while semantic information about bottlenecks, reuse, hierarchy, invalid regions, and workload structure remains outside the search loop. Moreover, black-box optimizers offer no insight into why a configuration was chosen, making it difficult to debug simulator artifacts or validate search decisions. Recent work has begun exploring large language models (LLMs) for hardware design and architecture exploration [2], [4], [7], [14], [18], [21]–[24]. These efforts highlight the promise of LLM-assisted architectural optimization, but many remain tied to target-specific workflows, precomputed databases, custom multi-agent systems, or specialized prompting pipelines. What is still missing is a reusable mechanism for turning this emerging reasoning capability into simulator-call selection in DSE. In human architectural practice, such reasoning does not take the form of black-box sampling. Architects use a small number of observations to infer constraints, bottlenecks, data reuse, parallelism, hierarchy, and workload structure, and then decide which configurations are worth evaluating next. Figure 1 illustrates this contrast.

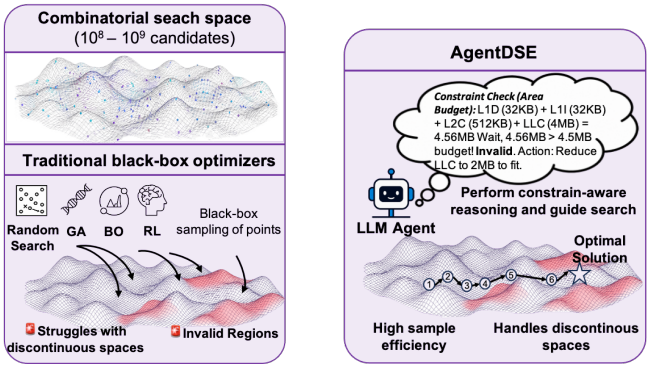

Fig. 1. Motivation for architectural-reasoning-guided DSE.

In this work, we bridge this gap by proposing AgentDSE, a simulator-in-the-loop agentic framework that puts architectural reasoning in the optimization loop. Instead of treating DSE as purely numerical sampling, we formulate it as an iterative, history-aware reasoning process driven by an LLM agent. Operating over a structured workspace of prior observations and hypotheses, the agent interprets task briefs through an autonomous hypothesis-test-refine loop: it proposes candidates, parses simulator feedback, and refines its hypotheses. This approach couples the architectural knowledge of foundation models [21] with a direct, grounded optimization loop.

We make the following contributions:

- **An Agentic DSE Paradigm:** We formalize AgentDSE as a domain-agnostic agent-simulator workflow that runs architectural reasoning as an iterative hypothesis-test-refine loop driven by simulator feedback. The same workspace abstraction is reused across MAESTRO, Timeloop/Accelergy, and ChampSim with no fine-tuning and no precomputed design database.

*Equal contribution.
†Corresponding author: zishenwan@seas.harvard.edu

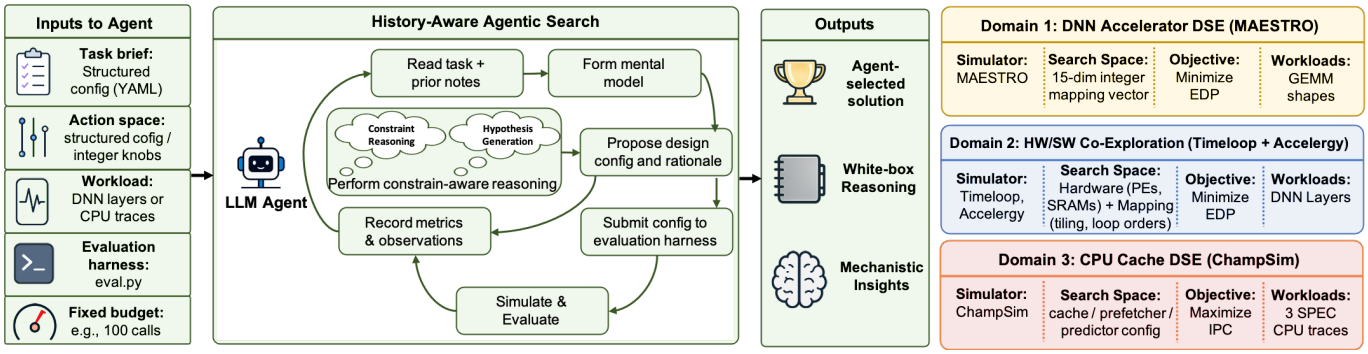

Fig. 2. Overview of AgentDSE for history-aware agentic design-space search.

- **Cross-domain Sample Efficiency Gains:** We evaluate AgentDSE across DNN accelerator mapping, hardware/software co-design, and CPU cache-hierarchy optimization. Across these settings, AgentDSE achieves competitive or better design quality with up to two orders of magnitude fewer simulator evaluations. We further decompose this gain through a controlled anonymization ablation on a representative ResNet50 layer, separating the LLM's strength as a black-box combinatorial optimizer from the additional acceleration provided by domain-specific architectural reasoning.
- **Auditable Optimization Traces:** AgentDSE produces inspectable hypothesis-test-refine traces that surface architectural insights, biased priors, non-monotonic performance cliffs, and simulator artifacts, making every search decision traceable and verifiable.

II. Agentic DSE Methodology

Figure 2 shows the AgentDSE workflow. AgentDSE casts design-space exploration as a simulator-in-the-loop coding task. Each run is packaged as a self-contained workspace that specifies the design objective, candidate schema, workload, hardware constraints, evaluator, and evaluation budget. A coding agent then iteratively edits candidate designs, invokes the evaluator, observes measured feedback, and records search state in workspace artifacts.

The methodological move is not simply attaching an LLM to a simulator, but exposing DSE as a persistent, editable, semantically grounded workspace in which the agent accumulates hypotheses, tests them against simulator feedback, and revises subsequent actions, shifting what the optimizer reads and writes from a scalar reward signal to a structured, inspectable record.

A. Workspace Interface

An AgentDSE workspace contains the following artifacts.

- **Task brief** (`README.md`): describes the objective, domain context, budget, and expected workflow.
- **Action-space specification** (`action_space.md`): defines candidate fields, valid ranges, and constraints.
- **Workload and hardware files** (`workload.yaml`, `hardware.yaml`): specify the target workload and either fixed hardware parameters or hardware search-space bounds.
- **Candidate file** (`candidate.yaml`): stores the next design to evaluate.
- **Evaluation harness** (`submit.sh`, `eval.py`): validates the candidate, invokes the simulator, updates logs, and returns measured metrics.
- **Search state** (`notes.md`, `history.jsonl`, `best.json`, `budget.json`): records free-form observations, all evaluations, the best design found so far, and remaining budget.

This interface deliberately exposes the DSE problem through ordinary files rather than through a custom optimizer API. Porting AgentDSE to a new domain requires only a domain-specific candidate schema and an evaluation harness that maps candidates to measured metrics.

B. Optimization Loop

Algorithm 1 summarizes the optimization procedure:

Algorithm 1 AgentDSE Optimization Loop

```
Read README.md, action_space.md, workload.yaml
Initialize search state from notes.md, history.jsonl, and best.json.
while budget remaining and agent has not terminated do
    Reason about the next action: form hypotheses about which design dimensions to explore and filter candidates against the constraints
    Write candidate to candidate.yaml
    Run ./submit.sh → receive metrics
    Append the result to history.jsonl and update best.json if improved
    Update notes.md with observations and next-step rationale.
end while
Return the best recorded design
```

The key distinction from prior LLM-for-DSE systems is the absence of a domain-specific control framework: no state machine [2], no multi-agent orchestration [22], and no retrieval over pre-computed design databases. The agent's "framework" is simply its native coding loop (read, edit, and execute) applied to simulator-driven optimization.

C. Auditable Optimization Traces

AgentDSE produces more than a final design point. Because the agent operates through editable files and explicit tool calls, the optimization process is externalized into persistent workspace artifacts. In particular, `notes.md`, `history.jsonl`, `best.json`, and post-processed Thinking-Action-Observation (TAO) traces together form an *auditable optimization trace*: they record the agent's intermediate hypotheses, the candidates it evaluated, the measured outcomes it observed, and the best-so-far progress over the course of search.

These traces are method artifacts rather than incidental logs. They make the search process inspectable at the level of concrete design decisions and their measured consequences, enabling post hoc analysis of why a design improved, why a trajectory failed, or whether an apparent gain was caused by a simulator artifact. As a result, AgentDSE exposes both an optimization *outcome* and a traceable optimization *process*.

D. Reusing the Workflow Across Domains

Figure 2 shows how the same workspace abstraction instantiates across three qualitatively different simulator stacks. Although the simulators target different design problems, they share the same optimization interface. In MAESTRO [13], an analytical cost model for DNN accelerator mappings, the candidate file encodes mapping decisions such as loop order, tiling, spatial dimensions, and PE allocation. In Timeloop/Accelergy [15], [25], a mapper and energy-modeling stack for tiled accelerators, the candidate specifies a joint hardware/software co-design point. In ChampSim [5], a trace-driven CPU microarchitecture simulator, the candidate encodes cache-hierarchy, prefetcher, and branch-predictor choices under structural constraints. Across all three settings, the agent edits the same candidate artifact, invokes the simulator, and records persistent search traces, allowing AgentDSE to reuse one coding-agent workflow without per-domain optimizer.

Taken together, this workflow exposes more than a scalar objective: it gives the agent semantic context, explicit constraint feedback, and persistent search history. These properties give the agent the inputs it needs to reason about each simulator call, and may enable more structured candidate selection than purely black-box or learned optimizers. We revisit this intuition in Section V through optimization traces, case studies, and ablations.

III. Experimental Setup

To demonstrate the effectiveness of AgentDSE, we conduct three studies. First, we perform a head-to-head comparison with DOSA [8] by fully replicating their experimental setup and substituting their DSE engine with AgentDSE under identical constraints. Second, we evaluate the generalization and scalability of our approach on MAESTRO [13] and ChampSim [5] simulators, comparing against traditional baselines including Random Search (RS), Genetic Algorithms (GA), and Reinforcement Learning (PPO). Finally, we conduct an ablation that separates AgentDSE's architectural reasoning from its combinatorial optimization capability to better understand the underlying drivers of its performance.

These three thrusts play complementary evaluative roles: the DOSA setting provides a direct matched-setup comparison against a representative differentiable DSE engine; MAESTRO tests whether the same agentic workflow transfers to a different DNN-mapping problem under a strict simulator-call budget against classical baselines; and ChampSim probes cross-domain transfer to a CPU memory-hierarchy task and the convergence efficiency of the agent's self-terminating search. Each setting emphasizes a different evaluation question, so the baselines and metrics are chosen to match established practice in that domain.

A. Primary Setting: Co-Design with DOSA/Timeloop

For our primary evaluation, we establish a head-to-head comparison against DOSA [8], a representative differentiable DSE methodology for accelerator co-design. The setting is a hardware/software co-design problem targeting a Gemmini-style systolic array [3] using the Timeloop/Accelergy evaluation stack [15], [25]. The optimizer must jointly select architectural parameters (e.g., spatial dimensions, buffer sizes) and dataflow mappings. While DOSA leverages a pre-computed dataset of 72,701 Timeloop samples (covering 65 distinct workload shapes) and performs 1,490 gradient-descent steps on the resulting analytical predictor across the 24-layer sweep ($\sim$62 steps/layer, with no real Timeloop calls in this inner loop), AgentDSE explores the discrete space from a cold start (no precomputed dataset and no surrogate predictor) under a strict budget of 80 ground-truth Timeloop calls per layer. We evaluate on the same 24 representative ResNet50 layers used in the DOSA study, targeting Energy-Delay Product (EDP). Detailed cost accounting and the inclusive per-layer comparison are deferred to Section IV-A.

B. Generalization: MAESTRO and ChampSim

To evaluate the generalization and scalability of our approach, we deploy the identical agentic methodology on two additional DSE domains.

DNN Accelerator Mapping (MAESTRO). We optimize spatial mapping and tiling on an edge-constrained accelerator using MAESTRO [13]. The action space spans 15 integer dimensions. We target EDP minimization across 10 distinct general matrix multiply (GEMM) shapes drawn from LLaMA-3.1-8B [6] and OpenVLA [10]. AgentDSE is restricted to 100 simulator calls per shape, compared against RS and GA at 20,000 calls, and Proximal Policy Optimization (PPO) at 50,000 steps.

CPU Cache-Hierarchy (ChampSim). We optimize a CPU memory hierarchy (L1/L2/last-level cache (LLC) sizes, associativities, prefetchers, and replacement policies) subject to a strict 4.5MiB static RAM (SRAM) area constraint using ChampSim [5]. We target Instructions Per Cycle (IPC) maximization on three distinct SPEC CPU2017 traces, with three seeds per workload (9 runs total). AgentDSE autonomously

terminates after $\sim$50 calls, while the baselines of RS, GA, and BO are assigned 1,000 evaluations.

IV. Results

A. Head-to-Head: Differentiable DSE vs. AgentDSE

We compare AgentDSE against DOSA under two strict apples-to-apples scopes that hold optimization scope fixed between the two methods: a per-layer comparison (Table I, with Figure 3 as visual companion) where each method optimizes each ResNet50 layer independently, and a network-level comparison (Table II) where each method optimizes one shared HW configuration across all 24 layers. Across both scopes, AgentDSE matches DOSA's EDP. For the per-layer comparison, we use a $\pm 5\%$ equivalence band to avoid over-interpreting small numerical EDP differences. AgentDSE achieves 9 wins, 13 ties, and 2 losses, with a geometric-mean EDP ratio of $0.99\times$. At the network level, AgentDSE achieves a $1.15\times$ improvement while using an average of 31 ground-truth simulator calls per layer, a $\sim 37\times$ reduction relative to DOSA's inclusive cost.

Cost accounting for DOSA. DOSA's reported cost has three components: (i) a 72,701-sample offline Timeloop dataset covering 65 distinct workload shapes, used to train an analytical predictor; (ii) for each layer, 7 random initializations of gradient descent on that predictor, each running 1,490 steps ($7\times 1{,}490 = 10{,}430$ predictor inferences per layer, with no real Timeloop calls in this inner loop); and (iii) 7 real Timeloop calls per layer at search time to validate the predicted optima. Amortizing the 72,701-sample offline dataset across the 65 shapes it covers yields $\sim$1,118 Timeloop calls per shape; each ResNet50 layer maps to one of these covered shapes (the dataset is constructed to include the layer dimensions used in DOSA's evaluation), so adding the 7 search-time calls per layer gives an inclusive cost of $\sim$1,125 real Timeloop calls per layer. Under this inclusive accounting, AgentDSE's cold-start search uses $\sim$**37**$\times$ fewer real Timeloop calls per layer than DOSA. We adopt inclusive accounting as the fairer common cost basis because AgentDSE has no comparable offline sampling phase to amortize: it issues real Timeloop calls from a cold start, with no precomputed dataset and no surrogate predictor. A search-only view, which charges DOSA only its 7 post-predictor verification calls per layer, omits the 72,701-sample modeling investment that any new accelerator or simulator stack would have to repeat.

a) Per-layer protocol and takeaways: The DOSA column in Table I comes from DOSA in per-layer mode: single-layer workloads, NUM_STARTS=7 random initializations per layer, with the gradient-descent loop patched with a steep quadratic-plus-cubic HW-bound penalty plus a hard projection step so DOSA's trajectory is constrained to AgentDSE's $[1, 2048]$ KB scratchpad/accumulator search bounds. AgentDSE wins on 9 layers, ties on 13, and loses on 2 across 24 layers, with a geometric-mean EDP ratio of $0.99\times$ (median $1.02\times$). The two losses (Conv40 at $0.43\times$ and Conv70 at $0.82\times$) are pointwise wins for DOSA on individual layers but do not propagate to the network level as shown in Table II, where AgentDSE wins outright. The headline takeaway at this protocol is that AgentDSE reaches DOSA-comparable per-layer EDP *without* DOSA's 72,701-sample offline predictor training set; the dominant remaining advantage is on sample efficiency rather than EDP itself.

TABLE I
STRICT PER-LAYER HEAD-TO-HEAD: AGENTDSE MATCHES DOSA'S PER-LAYER EDP (9 WINS, 13 TIES, 2 LOSSES; GEOMETRIC-MEAN RATIO $0.99\times$) AT $\sim 37\times$ LOWER SAMPLE COST. BOLD MARKS THE ROW WINNER; TIES (WITHIN 5%) ARE BOLDED BY LOWER EDP. PROTOCOL DETAILS AND COST ACCOUNTING ARE IN SECTION IV-A.

Layer	Dimensions (K, C, H, W, R, S)	AgentDSE (J·cyc)	DOSA (J·cyc)	EDP Ratio DOSA/AgentDSE
Conv83	256×256×14×14×3×3	**19.0**	20.4	1.07×
Conv76	1024×256×14×14×1×1	**7.01**	7.39	1.05×
Conv8	64×64×56×56×3×3	**10.1**	10.4	1.03×
Conv39	512×128×28×28×1×1	**8.11**	8.64	1.07×
Conv46	128×128×28×28×3×3	**7.70**	7.75	1.01×
Conv131	2048×512×7×7×1×1	**26.3**	27.7	1.05×
Conv2	64×3×112×112×7×7	**235**	235	1.00×
Conv5	64×64×56×56×1×1	2.79	**2.79**	1.00×
Conv11	256×64×56×56×1×1	**20.4**	20.5	1.00×
Conv15	64×256×56×56×1×1	20.5	20.5	1.00×
Conv33	128×256×56×56×1×1	**37.1**	37.2	1.00×
Conv36	128×128×28×28×3×3	**17.0**	17.3	1.01×
Conv40	512×256×28×28×1×1	88.0	**37.4**	0.43×
Conv43	128×512×28×28×1×1	**8.30**	8.34	1.00×
Conv70	256×512×28×28×1×1	22.1	**18.2**	0.82×
Conv73	256×256×14×14×3×3	**25.7**	27.5	1.07×
Conv77	1024×512×14×14×1×1	**33.8**	34.0	1.01×
Conv80	256×1024×14×14×1×1	**7.48**	8.22	1.10×
Conv125	512×1024×14×14×1×1	**21.8**	23.9	1.10×
Conv128	512×512×7×7×3×3	**133**	149	1.13×
Conv132	2048×1024×7×7×1×1	**112**	116	1.03×
Conv135	512×2048×7×7×1×1	**28.0**	29.0	1.04×
Conv138	512×512×7×7×3×3	**125**	139	1.11×
FC_out	1000×2048×1×1×1×1	**54.2**	54.8	1.01×

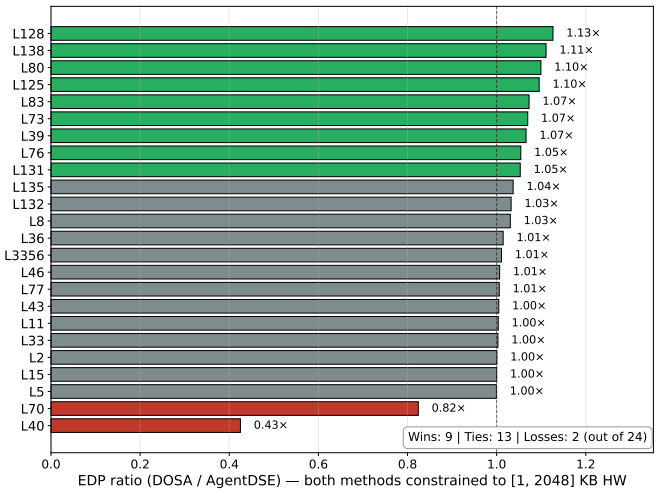

Fig. 3. Strict apples-to-apples per-layer comparison: DOSA in per-layer mode with HW search bounds matched to AgentDSE's $[1, 2048]$ KB via a patched gradient-descent penalty. 9 wins, 13 ties, 2 losses; geometric-mean EDP ratio $0.99\times$.

b) Network-level head-to-head: At the network level (one shared HW driving all 24 layers, the same setting as DOSA's published recipe), AgentDSE achieves a mean network EDP of 6.62×10^{10} uJ·cyc averaged across three random

seeds, compared to DOSA's reproduced best of 7.59×10^{10} ($\mathbf{1.15}\times$ AgentDSE improvement) and DOSA's paper-reported $\sim 1.0 \times 10^{11}$. All three AgentDSE seeds converged to the same shared HW configuration (pe_dim= 128, sp_size= 512 KB, acc_size= 512 KB), so the cross-seed variance enters only through the per-layer mapping search and the number of evaluations to convergence; seeds used $\sim$18.5 budget units on average (each unit invokes Timeloop on all 24 layers, so $\sim$444 underlying Timeloop calls per seed) at a mean API cost of $\sim$\$12. While DOSA's gradient descent must trade off HW size against per-layer fitness through a smooth predictor loss landscape, AgentDSE explicitly chose the maximum allowed PE dimension and balanced buffer sizes by reasoning over per-layer mapping requirements — a discrete design move that gradient descent does not naturally discover.

TABLE II
NETWORK-LEVEL HEAD-TO-HEAD: AGENTDSE ACHIEVES $1.15\times$ LOWER NETWORK EDP THAN DOSA'S REPRODUCTION WITH $\sim 165\times$ FEWER REAL TIMELOOP CALLS. AGENTDSE VALUES ARE MEAN ACROSS 3 RANDOM SEEDS; COST-ACCOUNTING BREAKDOWN FOR DOSA'S 72,869 INCLUSIVE CALLS IS GIVEN ABOVE.

Method	Network EDP (μJ·cyc)	Real Timeloop calls	API Cost (USD)
DOSA (estimated from paper)	$\sim 1.0 \times 10^{11}$	72,869	—
DOSA (our reproduction, best of 3)	7.59×10^{10}	72,869	—
AgentDSE (network, mean of 3 seeds)	$\mathbf{6.62 \times 10^{10}}$	$\sim$ **444**	$\sim$ **\$12**
AgentDSE vs. DOSA reproduction	$1.15\times$ **lower EDP**	$\sim 165\times$ **fewer Timeloop calls**	—

AgentDSE also self-terminates well below its 50-budget-unit ceiling, indicating its convergence detection is not an artifact of budget.

B. Anonymization Ablation: Reasoning vs. Optimization

To isolate the source of this efficacy, we conduct a controlled anonymization ablation on Layer 46. By stripping all semantic meaning from the workspace (i.e., hiding parameter names and physical metrics), we force the agent to optimize a purely numerical integer space.

TABLE III
ANONYMIZATION ABLATION ON RESNET50 LAYER 46: BOTH AGENT VARIANTS TIE DOSA'S STRICT PER-LAYER EDP; RESTORING PHYSICAL CONTEXT CUTS CALLS FROM 71 TO 29 (A $\sim 2.4\times$ SAMPLE-EFFICIENCY SPEEDUP). THE *Anonymized Agent* SEES ONLY OPAQUE INTEGER IDS; THE *Full-Context Agent* SEES PHYSICAL NAMES AND UNITS. DOSA BASELINE AS IN TABLE I; COST-ACCOUNTING IN SECTION IV-A.

Optimizer Setup	Best EDP (J·cyc)	EDP ratio DOSA / Agent	Timeloop calls	Timeloop calls (% of DOSA)
DOSA (per-layer mode)	7.75	1.00×	$\sim$1,125	100%
Anonymized Agent	7.79	0.99×	71	6.3%
Full-Context Agent	**7.70**	**1.01×**	**29**	**2.6%**

The results (Table III) point to two effects. First, the *Anonymized Agent* reaches DOSA-level per-layer EDP on this layer (within 1%) using only 71 Timeloop calls instead of $\sim$1,125, indicating that the LLM is itself a capable black-box combinatorial optimizer at vastly lower sample cost, even with no architectural semantics. Restoring the physical semantics (*Full-Context Agent*) further reduces the simulator calls required from 71 to 29 (a $\sim 2.4\times$ speedup) while reaching the same EDP quality. On this layer, then, the LLM-as-optimizer drives the headline sample-efficiency win, while architectural reasoning provides a multiplier on top of it; the two effects are complementary rather than substitutive. Figure 4 visualizes the corresponding convergence trajectories: both agent variants reach DOSA-comparable EDP within tens of simulator calls, with the full-context configuration converging earliest.

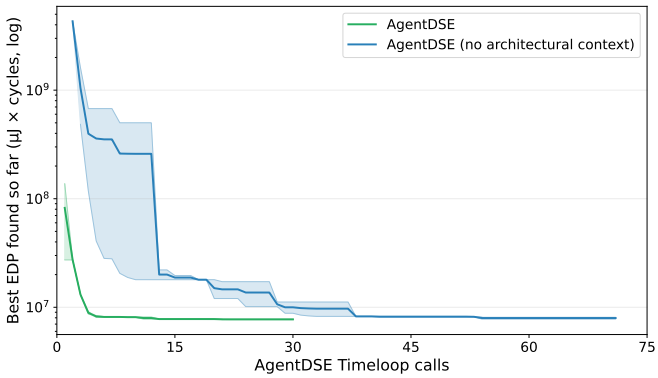

Fig. 4. Convergence on ResNet50 Layer 46: best-so-far EDP versus real Timeloop calls for the Full-Context and Anonymized AgentDSE variants (29 and 71 calls to converge, respectively). DOSA's $\sim$1,125-call inclusive per-layer budget does not fit visually on this axis; its EDP comparison is in Figure 3 and Tables I, II.

C. Cross-Domain Generalization

We summarize AgentDSE's performance on the remaining two hardware domains to test whether this sample efficiency generalizes.

MAESTRO DNN Mapping. As shown in Table IV, AgentDSE achieves the best model-level EDP on both LLaMA-3.1-8B and OpenVLA while using only 100 simulator calls per shape. In contrast, RS and GA use 20,000 evaluations per shape, and PPO uses 50,000 evaluations. Thus, AgentDSE attains the best EDP among the four methods evaluated while using a 200–500$\times$ smaller simulator budget.

TABLE IV
AGENTDSE ATTAINS THE BEST MODEL-LEVEL EDP ON BOTH LLAMA-3.1-8B AND OPENVLA AT 200–500$\times$ SMALLER SIMULATOR BUDGET. MAESTRO WEIGHTED EDP, SCALED BY 10^{-15}; LOWER IS BETTER, **BOLD** MARKS THE ROW WINNER. PER-SHAPE SIMULATOR BUDGETS IN PARENTHESES.

Model	AgentDSE (100/shape)	RS (20K)	GA (20K)	PPO (50K)
LLaMA-3.1-8B	**28.0**	85.7	30.9	29.4
OpenVLA	**7.17**	19.0	7.83	8.03

These aggregate results do not imply exhaustive coverage of every per-layer optimum; rather, they show that reasoning-guided search can identify strong model-level designs under a much smaller simulator budget.

ChampSim CPU Cache Hierarchy. We report ChampSim along three views, since they answer different questions:

(i) *final IPC* at each method's own budget (Table V); (ii) *equal-quality match-point*—the smallest AgentDSE budget that first reaches Random's 1,000-call IPC (Table VI); and (iii) *convergence-matched cost*—calls until each method plateaus (Table VII). Final IPC reaches parity (within $\sim 1.5\%$, bounded by simulator variance limits) with traditional optimizers given 1,000 evaluations, while AgentDSE self-terminates after ∼50 calls; the equal-quality and convergence-matched views then make the sample-efficiency advantage explicit.

TABLE V
AGENTDSE MATCHES RS, GA, AND BO ON FINAL IPC WHILE USING ∼20× FEWER SIMULATOR CALLS. BEST IPC PER CHAMPSIM WORKLOAD (HIGHER IS BETTER, **BOLD** = ROW WINNER). AGENTDSE: ∼50 SELF-TERMINATED CALLS; RS/GA/BO: 1,000 CALLS EACH. ON LBM AND OMNETPP ALL FOUR METHODS CLUSTER WITHIN ∼1.5% OF THE ROW WINNER, WELL WITHIN THE 5 M-INSTRUCTION SIMULATOR'S RUN-TO-RUN VARIANCE, SO WE TREAT THESE ROWS AS TIES.

Workload	AgentDSE (∼50 calls)	Random (1000)	GA (1000)	BO (1000)
mcf	**0.6141**	0.5941	0.6120	0.6112
lbm	0.5771	0.5752	0.5854	**0.5858**
omnetpp	0.3166	0.3176	**0.3193**	0.3181

TABLE VI
AT EQUAL IPC, AGENTDSE REACHES RANDOM'S 1,000-CALL QUALITY IN 20–50 CHAMPSIM CALLS (20–50× SPEEDUP). "AGENTDSE CHAMPSIM CALLS" IS THE EARLIEST CHECKPOINT MATCHING RANDOM@1000; SPEEDUP IS 1000 / CALLS. ON OMNETPP ALL FOUR METHODS CLUSTER WITHIN 0.002 IPC AT EVERY BUDGET, SO A MATCH POINT CANNOT BE DEFINED.

Workload	Random@1000 IPC	AgentDSE ChampSim calls	Speedup
mcf	0.594	**20**	50×
lbm	0.575	**50**	20×
omnetpp	0.318	no match point	—

To provide a tighter comparison, we evaluate convergence-matched sample efficiency. Table VII compares the number of calls required for each method to converge. Convergence-matched, AgentDSE's sample-efficiency advantage is **7–15**× across all workloads and baselines.

TABLE VII
CONVERGENCE-MATCHED, AGENTDSE IS 7–15× MORE SAMPLE-EFFICIENT THAN RS, GA, AND BO ON EVERY WORKLOAD. THE AGENTDSE COLUMN SHOWS CALLS TO PLATEAU (NO IPC IMPROVEMENT OVER THE NEXT 20 CALLS); THE OTHERS REPORT AGENTDSE'S SPEEDUP AT EACH BASELINE'S OWN CONVERGENCE POINT.

Workload	AgentDSE ChampSim calls	vs RS	vs GA	vs BO
mcf	55	8.5×	9.9×	11.4×
lbm	54	9.2×	6.9×	11.2×
omnetpp	41	15.4×	9.8×	14.3×

V. ANALYSIS AND INTERPRETABILITY

The sample-efficiency results in Section IV say *how well* AgentDSE performs; this section examines *how* it gets there. We present two illustrative case studies drawn from the trace logs (Section V-A) and one structural regularity observed across runs (Section V-B); these illustrate the kinds of design reasoning the trace exposes rather than serve as a statistical base.

A. Illustrative Trace Case Studies

AgentDSE's `notes.md` records the hypothesis behind each candidate and its measured outcome. We highlight two example patterns that an architect can read directly from the trace as design rules.

a) Workload-specific design rules absent from the prompt: On the ChampSim mcf trace (latency-bound, pointer-chasing), the agent forms an unusual conjecture about prefetcher utility:

> *"mcf is dominated by 200+-cycle DRAM latency. Aggressive prefetchers will inflate MPKI but useful prefetches still hide latency, MPKI is a proxy for miss rate, not for end-to-end performance."*

The agent tests this in two calls (toggling SPP at L2) and confirms the gain. We verified this MPKI-vs-performance distinction does not appear in any prompt file (`action_space.md`, `README.md`); the rule emerges from the agent's reasoning, not from retrieval.

b) Hypothesis-driven ablation: On mcf seed 3, after observing IPC=0.405 from a configuration with both SPP at L2 and SHIP at LLC, the agent voluntarily ablates one component:

> *"To attribute the gain, removing SPP from L2 in call 5: IPC drops to 0.329. SPP at L2 is responsible. Re-enabling SPP at both L2 and LLC in call 6: IPC=0.414, new best."*

The agent spends one of its budget calls on causal attribution rather than on refinement. Black-box optimizers allocate every evaluation to fitness improvement; this trace shows the agent reasoning explicitly about *why* a design works.

B. A Structural Regularity Across Runs

Beyond individual decisions, we observe a consistent structural pattern across runs. Across our 9 ChampSim runs (3 workloads × 3 seeds), 8 of 9 best-IPC designs satisfy the L1D ≤ L2C ≤ LLC hierarchy, and the constraint was *not* stated in `action_space.md` at the time these runs were collected. Uniform random samples from the unconstrained action space already satisfy this hierarchy roughly 85% of the time, so the gap on this small sample is too narrow to support a strong claim that the agent has internalized the hierarchy as a learned prior. We report the observation as suggestive: AgentDSE's outputs remain consistent with a standard memory-hierarchy convention without being told to.

VI. LIMITATIONS AND FUTURE WORK

LLM Dependence and Efficiency. A primary limitation is the reliance on high-capacity large language models, which introduces significant computational overhead and API costs during the iterative reasoning process. While this cost is offset by the reduction in simulator evaluations, future work should explore the use of smaller, domain-specific models

or distilled student models. Investigating whether specialized "architectural reasoning" models can maintain optimization quality while drastically reducing inference latency will be key to scaling the approach.

From Optimization to Discovery. Our current evaluation focuses on design spaces where human designers already possess strong prior heuristics. This allowed us to validate the agent's ability to mirror and accelerate expert-level reasoning. However, the true potential of coding agents lies in exploring complex, non-intuitive design spaces where human intuition is limited, such as novel heterogeneous interconnects or emerging non-von Neumann architectures. Future research will focus on "zero-shot" discovery, testing whether AgentDSE can identify novel architectural motifs and generalize new design principles that transcend existing human knowledge.

Scope of the Reasoning Ablation. Our anonymization ablation isolates the contribution of architectural semantics on a single ResNet50 layer; broader coverage across layers, workloads, and domains is a natural next step.

VII. Conclusion

We present an early but strong empirical case that general-purpose coding agents can serve as practical DSE engines across multiple architecture simulators. By substituting black-box numerical sampling with iterative architectural reasoning, AgentDSE matches or exceeds traditional baselines using up to two orders of magnitude fewer simulator evaluations across DNN accelerator mapping, hardware/software co-design, and CPU cache-hierarchy optimization.

References

[1] G. Dinh, I. K. J. Valsala, H. Luo, C. Hong, Y. Cho, J. Demmel, S. Li, and Y. Liu, "Sample-efficient mapspace optimization for dnn accelerators with bayesian learning," in *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*, 2023.

[2] Z. Fu, A. Manley, and M. Alian, "gem5 co-pilot: AI assistant agent for architectural design space exploration," *arXiv preprint arXiv:2510.19577*, 2025.

[3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. J. Ou, C. Schmidt, S. Steffl, J. C. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Design Automation Conference (DAC)*, 2021, pp. 769–774.

[4] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, "ORFS-agent: Tool-using agents for chip design optimization," in *Proceedings of the 7th ACM/IEEE Symposium on Machine Learning for CAD (MLCAD)*, 2025, pp. 1–13.

[5] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jiménez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.

[6] A. Grattafiori *et al.*, "The Llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[7] R. Gupta, A. Jain, A. Gonzalez, A. Novikov, P.-S. Huang, M. Balog, M. Eisenberger, S. Shirobokov, N. Vũ, M. Dixon, B. Nikolić, P. Ranganathan, and S. Karandikar, "ArchAgent: Agentic AI-driven computer architecture discovery," *arXiv preprint arXiv:2602.22425*, 2026.

[8] C. Hong, Q. Huang, G. Dinh, M. Subedar, and Y. S. Shao, "DOSA: Differentiable model-based one-loop search for DNN accelerators," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 209–224.

[9] S.-C. Kao and T. Krishna, "GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[10] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn, "OpenVLA: An open-source vision-language-action model," in *Proceedings of the 8th Conference on Robot Learning (CoRL)*, 2024, pp. 2679–2713.

[11] S. Krishnan, Z. Wan, K. Bhardwaj, P. Whatmough, A. Faust, S. Neuman, G.-Y. Wei, D. Brooks, and V. J. Reddi, "Automatic domain-specific soc design for autonomous unmanned aerial vehicles," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 300–317.

[12] S. Krishnan, A. Yazdanbakhsh, S. Prakash, J. Jabbour, I. Uchendu, S. Ghosh, B. Boroujerdian, D. Richins, D. Tripathy, A. Faust, and V. Janapa Reddi, "ArchGym: An open-source gymnasium for machine learning assisted architecture design," in *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[13] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarber, and T. Krishna, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[14] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu, B. Bhaskaran, B. Catanzaro, A. Chaudhuri, S. Clay, B. Dally, L. Dang, P. Deshpande, S. Dhodhi, S. Halepete, E. Hill, J. Hu, S. Jain, A. Jindal, B. Khailany, G. Kokai, K. Kunal, X. Li, C. Lind, H. Liu, S. Oberman, S. Omar, G. Pasandi, S. Pratty, J. Raiman, A. Sarkar, Z. Shao, H. Sun, P. P. Suthar, V. Tej, W. Turner, K. Xu, and H. Ren, "ChipNeMo: Domain-adapted LLMs for chip design," *arXiv preprint arXiv:2311.00176*, 2024.

[15] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. S. Emer, "Timeloop: A systematic approach to DNN accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[16] R. Raj, S. Banerjee, N. Chandra, Z. Wan, J. Tong, A. Samajdhar, and T. Krishna, "SCALE-Sim v3: A modular cycle-accurate systolic accelerator simulator for end-to-end system analysis," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2025, pp. 186–200.

[17] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. A. Gelbart, P. N. Whatmough, G.-Y. Wei, and D. M. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2017, pp. 1–6.

[18] K. Sankaralingam, "Computer architecture's AlphaZero moment: Automated discovery in an encircled world," *arXiv preprint arXiv:2604.03312*, 2026.

[19] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012.

[20] H. Sun, C. Wang, Z. Zhu, X. Ning, G. Dai, H. Yang, and Y. Wang, "Gibbon: Efficient co-exploration of NN model and processing-in-memory architecture," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 867–872.

[21] A. Tschand, C. Wang, Z. Wan, A. Cheng, I. Cristescu, K. He, H. Huang, A. Ingare, A. Kangaslahti, S. Kangaslahti, T. Lebryk, H. Lin, J. J. Ma, A. Meterez, C. Mohri, D. Morwani, S. Qin, R. Rinberg, P. Rodriguez-Diaz, A. M. Taliotis, P. Undrum Fathi, R. Zhao, T. Zhou, and V. Janapa Reddi, "GenAI for systems: Recurring challenges and design principles from software to silicon," *arXiv preprint arXiv:2602.15241*, 2026.

[22] H. Wang, X. Wu, Z. Ding, S. Zheng, C. Wang, N. Prakriya, T. Nowatzki, Y. Sun, and J. Cong, "LLM-DSE: Searching accelerator parameters with LLM agents," *arXiv preprint arXiv:2505.12188*, 2025.

[23] H. Wang, J. Gong, H. Zhang, J. Xu, and Z. Wang, "AI agentic programming: A survey of techniques, challenges, and opportunities," *arXiv preprint arXiv:2508.11126*, 2025.

[24] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "ChatEDA: A large language model powered autonomous agent for EDA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 10, pp. 3184–3197, 2024.

[25] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.