Mechanistic Critics for Sample-Efficient NPU Design-Space Exploration

Yicheng He, Yuqi Xue, Jerry Wu, Jian Huang, Bin Hu
University of Illinois Urbana-Champaign
yh84@illinois.edu

Abstract

NPU design-space exploration for LLM inference mainly relies on an expensive simulator that often returns only scalar feedback such as latency and power. Such feedback can rank evaluated configurations, but gives little information for deciding why a configuration is slow or which architecture parameter should be changed next. This limitation is especially severe because prefill and decode workloads can be limited by different resources, including compute throughput and HBM bandwidth. We propose **CriticNPU**, a sample-efficient DSE framework built around an online-calibrated mechanistic critic for NPU architecture design. The critic uses explicit hardware limits to evaluate NPU configurations: before simulation, it estimates whether a proposed configuration is likely to improve over the current best and filters weak candidates; after simulation, it aggregates operator-level runtimes into prefill/decode bottleneck summaries and predicts the effect of candidate parameter changes. We instantiate the critic with a Roofline-style model that derives compute and memory limits from architecture parameters and calibrates per-operator-type effective limits using simulator observations. And CriticNPU combines this critic with a workload extractor and an architecture planner to propose configurations. Across six LLM workloads and 16 deployment configurations per model, CriticNPU achieves best-or-tied-best latency in 74% of prefill and 87.5% of decode settings while using roughly half the simulator calls of competitive DSE baselines. We conduct extensive ablation studies and detailed analyses to validate the effectiveness of our method from multiple perspectives. Our code is available at `https://github.com/bruno686/CriticNPU`.

1 Introduction

Neural processing units (NPUs) [Jouppi et al., 2017, Chen et al., 2014, 2016] are specialized accelerators for neural-network workloads and are increasingly important for large language model (LLM) inference [He et al., 2025, Li et al., 2026, Xiang et al., 2026]. Given an LLM specification and a deployment setting, NPU design-space exploration (DSE) searches for an NPU configuration that minimizes inference latency under hardware constraints. An NPU configuration is defined by architecture parameters such as systolic-array count and dimension, vector-unit count, clock frequency, on-chip memory size, HBM bandwidth, and interconnect bandwidth, while satisfying power and memory constraints [Chen et al., 2025, Lee et al., 2026]. This search is difficult because architecture parameters interact in coupled and non-smooth ways [Yin et al., 2026]. For example, adding systolic arrays gives little benefit after the workload becomes memory-bound; increasing HBM bandwidth may not help when inter-chip communication dominates; and lowering frequency can sometimes improve latency if it frees power for more parallel units. LLM inference makes the problem harder because prefill and decode often stress different resources, so the best NPU configuration for TTFT may differ from the best configuration for TPOT.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Most DSE methods use simulator to evaluate candidate configurations, but the simulator often returns only scalar feedback such as latencya and power. This feedback can rank evaluated configurations, but it gives little help for deciding which architecture parameter should be changed next. A slow configuration may be limited by compute throughput, HBM bandwidth, ICI bandwidth, or a mixture of them; scalar latency alone does not reveal which case applies. Black-box search methods, including Bayesian optimization, evolutionary search, and greedy search, can work with scalar feedback [Torun et al., 2018, Bai et al., 2023a, Kao et al., 2020], but they spend simulator calls to learn which regions of the configuration space are useful. Bottleneck-aware methods add more structure by analyzing simulator traces or critical paths [Bai et al., 2023b, Zhang et al., 2026]. They can explain why an evaluated configuration is slow, but the diagnosis is tied to the configuration that was already simulated. It does not directly tell the search procedure whether an unevaluated candidate configuration should be simulated, or which architecture parameters should be prioritized in the next update. Recent LLM-assisted DSE methods can propose multi-parameter configurations from prompts, search history, and design rules [Tang et al., 2025, Wang et al., 2025, Pandit et al., 2025], but deciding whether a proposed configuration is worth simulating still needs a signal from hardware behavior.

This suggests that sample-efficient DSE needs a different form of simulator feedback: one that can reject weak candidates before simulation and turn simulated bottlenecks into concrete constraints on the next parameter update. We propose *mechanistic critics* for sample-efficient NPU design-space exploration. A mechanistic critic evaluates a candidate configuration through explicit hardware limits, rather than only through a scalar simulator score or an LLM-generated estimate. For NPU configurations, these limits correspond to compute throughput, HBM bandwidth, and ICI bandwidth. The critic is used at two points in the search loop. Before simulation, it estimates whether a proposed configuration is likely to improve over the current best configuration, so weak candidates can be skipped before calling the simulator. After simulation, it reads the operator-level runtimes, summarizes where prefill and decode latency is spent, and predicts the effect of candidate parameter changes. This diagnosis is then used to constrain the next parameter update. For example, if the simulated configuration is memory-bound, the next update should not simply add more systolic arrays; it should change parameters that affect memory time. Thus, the critic does more than explain evaluated configurations: it also helps choose which configurations to simulate and which parameters to prioritize next.

This paper is not about making a given edge LLM faster. Instead, it addresses a missing first step in edge LLM NPU design: before hardware exploration begins, the system must determine which LLM should become the NPU design target, based jointly on user requirements and hardware constraints. We instantiate the mechanistic critic in **CriticNPU**, a sample-efficient framework for NPU DSE. The critic is implemented as an online-calibrated Roofline-style model [Williams et al., 2009]: for each NPU configuration, it derives compute and memory limits from architecture parameters and calibrates per-operator-type effective limits using simulator observations. CriticNPU has three components. A workload extractor (Computing Demand Agent) maps the LLM specification and deployment setting to an operator-level LLM workload. The calibrated mechanistic critic (Online-Calibrated Roofline Model) provides candidate filtering, per-phase bottleneck summaries, and candidate parameter-change predictions. An LLM-based architecture planner (Architecture Design Agent) proposes the next NPU configuration. The planner proposes configurations, while the decision of which candidates to simulate and which architecture parameters to prioritize comes from the calibrated critic. Across six LLM workloads and 16 deployment configurations per model, CriticNPU achieves best-or-tied-best latency in 74% of prefill and 87.5% of decode settings while using roughly half the simulator calls of competitive baselines, including Bayesian optimization, CMA-ES, greedy search, grid search, ChatDSE, LLM-DSE, and LLM-Box. Our contributions are:

- **Framing NPU DSE around mechanistic critics.** We formulate sample-efficient NPU design-space exploration as a critic-guided search problem, where simulator feedback is used not only to score evaluated configurations, but also to decide which candidate configurations are worth simulating and which architecture parameters should be prioritized next.

- **Calibrating Roofline limits into predictive and diagnostic feedback.** We instantiate the critic with a Roofline-style model that derives compute, memory limits from NPU architecture parameters and calibrates per-operator-type effective limits using simulator observations. Before simulation, the critic filters weak candidates; after simulation, it summarizes operator-level runtimes into per-phase bottleneck shares and predicts the effect of candidate parameter changes.

- **Reducing simulator cost for LLM inference DSE.** We build CriticNPU, a sample-efficient NPU DSE framework that combines workload extraction from an LLM specification and deployment setting, critic-guided candidate filtering, and LLM-based configuration proposal. Across six LLM workloads and 16 deployment configurations per model, CriticNPU achieves best-or-tied-best latency in 74% of prefill and 87.5% of decode settings while using roughly half the simulator calls of competitive DSE baselines.

2 Preliminary

Problem Formulation. Our goal is to find an NPU configuration with minimal latency $\mathcal{L}$ for a given LLM specification W, while satisfying constraints $\mathcal{C}$ including power, HBM bandwidth and capacity, as well as interconnect and datacenter network bandwidth limits. An NPU configuration θ is a joint assignment of architecture parameters (systolic array count and size, clock frequency, HBM bandwidth, ICI bandwidth, etc.) drawn from a discrete design space $\mathcal{S}$. Finding the best configuration for LLM inference is a constrained optimization,

$$\min_{\theta \in \mathcal{S}} \quad \mathcal{L}(\theta; W, D) \quad \text{s.t.} \quad \mathcal{C}, \tag{1}$$

where D is the deployment setting (e.g., number of parallel chips). Following standard LLM serving practice, the latency $\mathcal{L}$ is decomposed into TTFT (time-to-first-token, the prefill phase) and TPOT (time-per-output-token, the decode phase). A description of various parameters is in Appendix B.

Roofline Model. The roofline model [Williams et al., 2009] provides a simple yet effective framework for understanding how efficiently hardware can execute a given workload and identifying whether performance is constrained by compute throughput or memory bandwidth. Specifically, it bounds achievable performance as a function of arithmetic intensity $I = \mathrm{FLOPs/Bytes}$, peak compute throughput C, and memory bandwidth B, expressed as $\min(C,\, B \cdot I)$. The crossover point $I_{\mathrm{ridge}} = C/B$ separates memory-bound operators ($I < I_{\mathrm{ridge}}$) from compute-bound ones ($I > I_{\mathrm{ridge}}$). When deploying LLM on NPU, this classification identifies the primary hardware bottleneck for each operator, providing a principled basis for adjusting architecture parameters. For clarity, we provide several illustrative examples in Appendix A.

3 Method

CriticNPU treats NPU design-space exploration as mechanistic critic-guided search over NPU configurations. Instead of using simulator feedback only to score evaluated configurations, CriticNPU maintains an online-calibrated mechanistic critic that works at two points in the search loop: before simulation, it filters weak candidate configurations; after simulation, it summarizes prefill/decode bottlenecks and predicts the effect of candidate parameter changes. At the implementation level, CriticNPU has three components, as shown in Figure 1. **(1) Compute Demand Agent**, implemented as a workload extractor, maps the LLM specification and deployment setting to an LLM workload. **(2) Online-Calibrated Roofline Model**, serving as the calibrated mechanistic critic, derives hardware limits from each NPU configuration, calibrates per-operator-type effective limits from simulator observations, and provides candidate filtering and bottleneck-guided feedback. **(3) Architecture Design Agent**, an architecture planner, proposes the next NPU configuration using the workload, constraints, search history, bottleneck summaries, and candidate parameter-change predictions. We describe these components in the following sections.

3.1 Compute Demand Agent: LLM-Driven Workload Extractor

Why Agent extraction. Modern LLM architectures evolve rapidly: dense Transformers, Mixture-of-Experts (DeepSeek-V2/V3), grouped-query attention (Llama-3), Flash-Attention variants, and emerging routing schemes all introduce different operator graphs. Hand-coding an extractor for each new LLM specification is labor-intensive: it requires re-implementing sharding rules, attention variants, and parallelism-aware tensor reshaping for every model family, and any new release breaks the pipeline until a developer manually updates the code. A structured workload profile exposes the actual compute and memory demand seen by the NPU, whereas the raw LLM specification mainly describes model dimensions, tensor shapes, and layer counts. We instead delegate this task

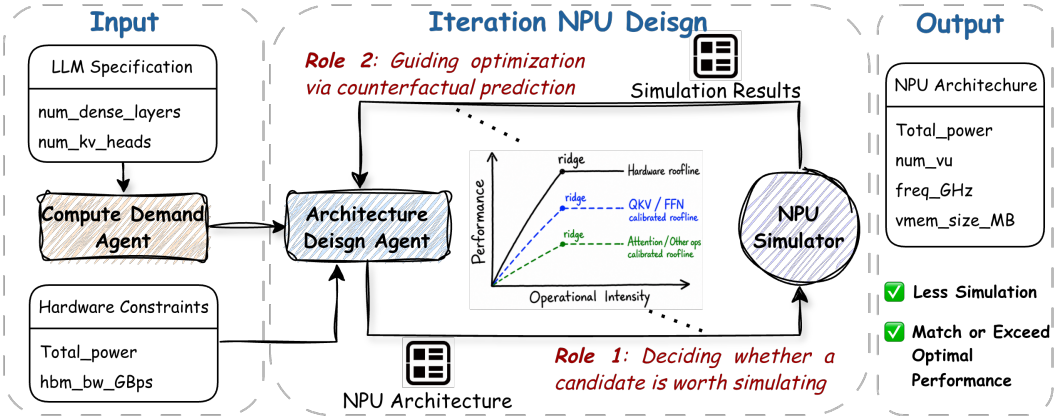

Figure 1: Overview of **CriticNPU**. A Compute Demand Agent extracts the operator-level workload, an online-calibrated roofline model filters weak configurations and provides bottleneck-guided feedback, and an Architecture Design Agent proposes the next NPU configuration. The shown parameters are illustrative examples.

to an Agent, the Compute Demand Agent, which reads the LLM specification and produces the corresponding operator-level workload.

How the Agent extracts the workload. Given an LLM specification W together with the deployment setting, the Compute Demand Agent produces a per-operator workload $\mathcal{O}$:

$$\mathcal{O} \sim \pi_{\text{ComputeDemandAgent}}(\cdot \mid W, D) \tag{2}$$

where $\pi_{\text{ComputeDemandAgent}}$ denotes an LLM-based policy that generates $\mathcal{O}$ conditioned on W and D. Each $\mathcal{O}$ is a set of operators each annotated with its floating-point cost $\text{FLOPs}(o)$, memory traffic $\text{Bytes}(o)$ (weights, activations, and KV cache), execution count $c(o)$ per forward pass, and arithmetic intensity $I(o) = \text{FLOPs}(o)/\text{Bytes}(o)$. The detailed prompt is provided in Appendix D.1.

3.2 Online-Calibrated Roofline Model: Mechanistic Critic for Screening and Diagnosis

Why a roofline model in the loop. A roofline model is useful for NPU parameter design because it ties performance to hardware limits that change directly with the configuration. Given the compute, memory limits of a candidate configuration, the roofline provides a way to estimate whether latency is likely to improve and to identify which operators limit performance. We use this structure to provide performance feedback at two points in the design process. ***Role 1: Deciding whether a candidate is worth simulating.*** Before invoking the simulator on a proposed configuration, we predict its latency from the roofline. If the prediction clears the current best, the configuration is simulated for ground truth; otherwise it is skipped. This cuts the number of costly simulator calls. ***Role 2: Guiding optimization via counterfactual prediction.*** After each simulation, we do not hand the raw per-operator runtimes to the Design Agent. The roofline digests them into a structured diagnosis of where time is spent, and predicts how small parameter changes would shift this distribution, so the agent gets directed guidance for its next move, not just raw numbers.

Role 1: Deciding whether a candidate is worth simulating. The simulator is the bottleneck of our search, since each call is expensive, and the Design Agent produces many candidates. We therefore want a cheap way to estimate a candidate's latency before paying for a full simulation. The key idea: a recent simulated reference configuration θ already tells us how long each operator took. A new candidate θ' that only tweaks the architecture parameters changes these runtimes in a predictable way, as a compute-bound operator gets faster if θ' has more compute, a memory-bound one gets faster with more bandwidth, and comm-bound ops depend on ICI bandwidth. So predicting θ' reduces to two things per operator: *what it is bottlenecked on*, and *how much the corresponding hardware resource changes*.

The hardware peaks are $C(\theta) = n_{\text{sa}}\, d_{\text{sa}}^2 \cdot 2f$ and $B(\theta) = B_{\text{HBM}}$, where n_{sa} is the number of systolic arrays, d_{sa} the SA side length, f the clock frequency, and B_{HBM} the HBM bandwidth—all

components of θ. The roofline then gives a single cutoff that separates the two regimes—the *ridge point* $I_{\text{ridge}}(\theta) = C(\theta)/B(\theta)$. For each operator o, we compare its arithmetic intensity $I(o)$ to the ridge: $I(o) > I_{\text{ridge}}$ labels it compute-bound, $I(o) < I_{\text{ridge}}$ memory-bound. Let $T(o;\theta)$ denote the runtime of operator o measured by the simulator at the reference θ. The predicted runtime at θ' then scales by how the relevant peak changes:

$$T(o;\theta') = T(o;\theta) \cdot \begin{cases} C(\theta)/C(\theta') & \text{compute-bound,} \\ B(\theta)/B(\theta') & \text{memory-bound.} \end{cases} \tag{3}$$

Summing $c(o)\,T(o;\theta')$ over operators and rescaling against the reference TTFT/TPOT gives the predicted latency $\hat{\mathcal{L}}(\theta')$; an analytical power estimate $P(\theta')$ is computed in parallel from θ'. Let t index the search iteration. Each prediction is coupled to a simple gating rule: simulate θ' during a short warm-up or when the predicted latency is at most the current best $\mathcal{L}^*$; otherwise skip it.

$$\text{Simulate}(\theta') = \begin{cases} 1, & t < t_0 \quad \text{(warm-up)}, \\ 1, & \hat{\mathcal{L}}(\theta') \le \mathcal{L}^* \quad \text{(predicted no worse than best)}, \\ 0, & \text{otherwise (skip)}, \end{cases} \tag{4}$$

where t_0 is the warm-up iteration budget. The simulator is thus reserved for candidates the roofline cannot confidently dismiss.

Role 2: Guiding optimization with counterfactual prediction. The simulator returns accurate but raw per-operator runtimes. Handing this output directly to the Design Agent is noisy and hard to act on, since the agent cannot easily see where time is going, or whether a given parameter change would help. Role 2 of the roofline is to *digest* each simulation into a compact, actionable signal: (i) where the current configuration is bottlenecked, and (ii) what would happen under small hypothetical edits. *Step 1: diagnose where time goes.* Each operator o is classified using the same test as Role 1: otherwise $I(o) > I_{\text{ridge}}(\theta) \Rightarrow$ compute-bound, $I(o) < I_{\text{ridge}}(\theta) \Rightarrow$ memory-bound. The labels are aggregated into per-phase bottleneck shares,

$$\alpha_r^{\phi}(\theta) = \frac{\sum_{o \in \mathcal{O}_r^{\phi}(\theta)} c(o)\,T(o;\theta)}{\sum_{o \in \mathcal{O}^{\phi}} c(o)\,T(o;\theta)}, \quad r \in \{\text{comp}, \text{mem}\},\ \phi \in \{\text{prefill}, \text{decode}\}, \tag{5}$$

so the planner sees, per phase, what fraction of latency sits under each regime. *Step 2: counterfactuals for candidate edits.* For each candidate parameter change the planner is considering, the roofline reuses the scaling rule of Eq. 3 to predict its effect, namely which operators speed up, which stay the same, whether the dominant regime shifts, and how TTFT and TPOT would change, all without another simulation. The outputs are written into a short block of text through fixed templates (no LLM inside the roofline) and appended to the Design Agent's prompt. A typical line reads: "*prefill is currently 60% compute-bound on FFN-type operators; lowering frequency to free power for two more SAs is predicted to reduce TTFT by* $\approx 12\%$ *without violating the power budget.*" The numerical claims here (the share, the speedup, the power check) are deterministic outputs of Eqs. 5 and 3; the LLM only reads them, it does not generate them.

Why the raw peaks are not enough. Both roles above classify each operator by where its arithmetic intensity $I(o)$ sits relative to $C(\theta)/B(\theta)$. This implicitly assumes the workload can reach the full compute peak C when compute-bound and the full memory peak B when memory-bound, yet no real workload does. A softmax dependency chain stops the systolic array from filling; LayerNorm has so little math per byte that cache misses dominate; pipeline stalls and imperfect compilation leave headroom the peaks never see. How big this gap is depends on the operator: an FFN GEMM can use most of the systolic array, attention noticeably less, LayerNorm barely any of it. So C/B sits above the true transition for some operator types and can sit below for others, and using it as a single ridge mis-labels operators near the boundary. A mis-labelled operator then picks the wrong branch in Eq. 3, scaling by C when it should scale by B, or the other way around, which is a predictable source of error that widens as θ moves away from the reference.

The online-calibrated roofline. We fix this by learning, per operator type, how much of each peak the workload actually reaches. We group operators into a small set of types τ by their role in the transformer block: QKV projection, FFN GEMM, attention kernel, other matmuls, and non-matmul ops (norm, activation, all-reduce), so that operators in the same group use the hardware in similar

ways. For each type we maintain two efficiency factors $\eta^{(\tau)}_{\text{comp}}, \eta^{(\tau)}_{\text{mem}} \in (0, 1]$: the fraction of C and B that this type can actually reach. The peaks themselves stay a hardware property; $\eta^{(\tau)}$ only tells us how well this kind of operator uses them. Put another way, $\eta^{(\tau)}$ is a per-type calibration of C and B: the peaks do not change, but each operator type effectively sees its own reduced ceilings. The per-type part is what matters, since a single global scale on C and B would only shift one ridge line for all operators, while type-specific η gives each operator type its own ridge, so the classifier can label different types differently. Because these factors come from the operator's shape and structure, rather than the chip, they stay roughly the same when we change NPU parameters, and transfer from one configuration to the next.

These factors are learned from simulator feedback. After every simulation, we read each operator's SA time $T_{\text{SA}}(o)$, memory time $T_{\text{mem}}(o)$, FLOP count, and memory traffic, compute the observed per-operator efficiencies

$$\hat{\eta}^{(\tau)}_{\text{comp}}(o) = \frac{\text{FLOPs}(o)/T_{\text{SA}}(o)}{C(\theta)}, \qquad \hat{\eta}^{(\tau)}_{\text{mem}}(o) = \frac{\text{Bytes}(o)/T_{\text{mem}}(o)}{B(\theta)}, \tag{6}$$

average them across operators of the same type, and update $\eta^{(\tau)}$ with an exponential moving average $\eta^{(\tau)}_{t+1} = (1-\lambda)\,\eta^{(\tau)}_t + \lambda\,\bar{\hat{\eta}}^{(\tau)}_t$ (with $\lambda \in (0, 1)$), which stays stable against per-operator noise while adapting as new simulations arrive. The calibration plugs into Roles 1 and 2 through effective ceilings $\tilde{C}^{(\tau)}(\theta) = \eta^{(\tau)}_{\text{comp}}\, C(\theta)$ and $\tilde{B}^{(\tau)}(\theta) = \eta^{(\tau)}_{\text{mem}}\, B(\theta)$, turning the single ridge into a per-type one $\tilde{I}^{(\tau)}_{\text{ridge}}(\theta) = (\eta^{(\tau)}_{\text{comp}}/\eta^{(\tau)}_{\text{mem}})\, C(\theta)/B(\theta)$. The classifier in Roles 1 and 2 now tests each operator against the ridge of its own type, while the scaling rule in Eq. 3 and the bottleneck share in Eq. 5 keep their form, since $\eta^{(\tau)}$ cancels out of the ratios. When θ changes during search, only $C(\theta)$ and $B(\theta)$ are recomputed; the learned $\eta^{(\tau)}$ carry over.

3.3 Architecture Design Agent: LLM-Driven Architecture Planner

At iteration t, the Architecture Design Agent (Prompt template at Appendix D.2) observes a compact state summarizing the workload, the current configuration, the counterfactual prediction, the design-space constraints, and the search history,

$$\theta_{t+1} \sim \pi_{\text{ArchDesignAgent}}(\mathcal{O},\, \theta_t,\, \alpha(\theta_t),\, \mathcal{C},\, \mathcal{H}_t), \tag{7}$$

where $\pi_{\text{ArchDesignAgent}}$ is an LLM-based policy that generates the next configuration θ_{t+1} conditioned on the workload $\mathcal{O}$, the current configuration θ_t, the roofline-guided signal $\alpha(\theta_t)$, hardware constraints $\mathcal{C}$, and the search history $\mathcal{H}_t = \{(\theta_i,\, \mathcal{L}(\theta_i),\, \alpha(\theta_i))\}_{i \le t}$.

4 Experiments

To provide a fair and thorough evaluation, we test CriticNPU on six LLM specifications ranging from 8B to 671B parameters: Llama3-8B, Llama2-13B, Llama3-70B, Llama3.1-405B, DSv2-236B, and DSv3-671B [Touvron et al., 2023, Liu et al., 2024]. These workloads cover both dense Transformers and mixture-of-experts models. We use NeuSIM [Xue and Huang, 2025], an analytical NPU simulator that reports operator-level latency breakdowns, power, and memory usage for each NPU configuration and workload. We compare against three groups of baselines: black-box optimization methods, including Bayesian optimization (BO) [Torun et al., 2018] and CMA-ES [Sato and Watanabe, 2024]; search heuristics, including greedy local search [Kim et al., 2020] and grid search [LaValle et al., 2004]; and LLM-assisted DSE methods, including ChatDSE [Tang et al., 2025], LLM-DSE [Wang et al., 2025], and LLM-Box [Pandit et al., 2025]. We evaluate all methods across 16 combinations of hardware constraints and deployment settings. Following disaggregated serving, we optimize prefill and decode chips separately: prefill is evaluated by time-to-first-token (TTFT), and decode is evaluated by time-per-output-token (TPOT). TTFT is computed as the latency of processing the input prompt in the prefill stage, while TPOT is computed as the per-token latency of one autoregressive decoding step using the KV cache.

4.1 Main Results

A - CriticNPU achieves competitive design quality with substantially fewer simulator calls. Table 1 reports win rate across the 16 deployment settings detailed in Tables 13–28 together with the

Table 1: Win rate and average simulator calls across 16 deployment configurations per model. Small models (8B, 13B) use 8 chips; large models (70B–671B) use 64 chips (DSv2-236B uses 16 chips). Win rate counts configurations where the method achieves the best (or tied-for-best) latency. **Bold** marks the best value per column (highest Win, lowest Sims); underline marks the second-best. Our method (CriticNPU) is highlighted in gray.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓
Prefill	BO	11/16	117	8/16	117	12/16	117	2/16	117	8/16	117	10/16	117
	CMA-ES	0/16	130	3/16	130	3/16	130	2/16	130	0/16	130	0/16	130
	Greedy	**15/16**	129	**15/16**	130	10/16	130	9/16	130	**14/16**	130	**13/16**	130
	Grid	0/16	1440	0/16	1440	0/16	1440	1/16	1440	0/16	1440	0/16	1440
	ChatDSE	0/16	\	0/16	\	0/16	\	0/16	\	0/16	\	0/16	\
	LLM-DSE	9/16	108	8/16	106	7/16	107	7/16	108	7/16	107	9/16	98
	LLM-Box	3/16	115	5/16	117	6/16	116	3/16	117	5/16	117	3/16	87
	CriticNPU (Ours)	14/16	**50**	11/16	**51**	**13/16**	**55**	**10/16**	**54**	11/16	**56**	12/16	**57**
	Method	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓	Win ↑	Sims ↓
Decode	BO	**16/16**	117	**16/16**	117	**16/16**	117	**16/16**	117	**16/16**	117	**15/16**	117
	CMA-ES	**16/16**	130	**16/16**	130	**16/16**	130	**16/16**	130	7/16	130	10/16	130
	Greedy	**16/16**	129	**16/16**	130	**16/16**	130	**16/16**	130	**16/16**	130	14/16	130
	Grid	0/16	1440	0/16	1440	0/16	1440	0/16	1440	0/16	1440	0/16	1440
	ChatDSE	0/16	\	0/16	\	0/16	\	0/16	\	0/16	\	0/16	\
	LLM-DSE	**16/16**	108	**16/16**	106	**16/16**	107	14/16	108	12/16	107	8/16	98
	LLM-Box	**16/16**	115	15/16	117	**16/16**	116	11/16	117	9/16	117	5/16	87
	CriticNPU (Ours)	15/16	**50**	**16/16**	**51**	**16/16**	**55**	**16/16**	**54**	13/16	**56**	8/16	**57**

Table 2: Component ablation of CriticNPU on Llama3-8B (8 chips, TP=4, PP=2), Llama3-70B (64 chips, TP=8, PP=8), and Llama3.1-405B (64 chips, TP=8, PP=8), all with BS=48, 300W under disaggregated serving. TTFT reported in seconds (prefill chip), TPOT in milliseconds (decode chip); Sims counts simulator calls consumed in each phase.

Ablation Methods	**Llama3-8B**				**Llama3-70B**				**Llama3.1-405B**			
	TTFT	Sims	TPOT	Sims	TTFT	Sims	TPOT	Sims	TTFT	Sims	TPOT	Sims
CriticNPU (Full)	0.4042	39	9.03	15	0.4164	35	15.93	13	2.0015	39	71.96	13
w/o Compute Demand Agent	0.4042	41	9.03	15	0.4248	41	15.93	13	2.0285	40	71.96	13
w/o Roofline model	0.4042	38	9.03	48	0.4248	39	15.93	38	2.0464	40	71.96	41
Role 1 only (gating, no digest)	0.4042	37	9.03	15	0.4164	36	15.93	13	2.0015	38	71.96	15
Role 2 only (digest, no gating)	0.4042	40	9.03	42	0.4164	42	15.93	35	2.0015	39	71.96	44

average number of simulator calls per run. CriticNPU wins 74.0% of prefill and 87.5% of decode configurations using only 54 simulator calls on average; every one of its 96 runs stays under 100 calls, a property no other competitive method shares. Classical baselines (BO, CMA-ES, Greedy) reach comparable or higher win rates only by spending 2.2–2.4× more simulations, while the LLM-based baselines fall behind for opposite reasons: LLM-DSE and LLM-Box query the simulator every iteration but lack a physics prior, so they wander the design space and finalize suboptimally, whereas ChatDSE bypasses simulator feedback and relies on the LLM to estimate latency and constraint satisfaction directly, which leads to configurations that fail to achieve any best-or-tied-best latency in the main-table settings. CriticNPU avoids both failure modes by using the calibrated roofline to anchor proposals to the currently limiting resource, which is most effective when latency is governed by standard compute/HBM/ICI ceilings. Its main exception is DSv3-671B decode, where BO obtains 15/16 wins while CriticNPU obtains 8/16. This setting is less well captured by the current roofline because DSv3 decode is not a pure bandwidth/compute scaling problem: MoE routing, activated-expert imbalance, and cross-expert KV-cache/communication effects introduce structure not explicitly modeled by our operator-type calibration. In this regime, BO's larger simulation budget can be more robust by directly sampling these irregular effects; we treat this as a limitation rather than a failure of the overall simulator-efficiency trend. As a complement to win rate, we also report mean normalized latency $\overline{L_m/L^*}$ across all 96 cells (where L^* is the best valid latency any method reached on that cell): CriticNPU achieves $1.012\times$ universal-best on TTFT and $1.008\times$ on TPOT—within 1.2% of optimum on average, comparable to BO (1.006) and Greedy (1.009) which spend 2.2–2.4× more simulator calls, and well below CMA-ES (1.242), LLM-Box (1.198), LLM-DSE (1.528, p90 2.86), and ChatDSE (3.500). The win-rate gaps among the strongest methods therefore mostly reflect

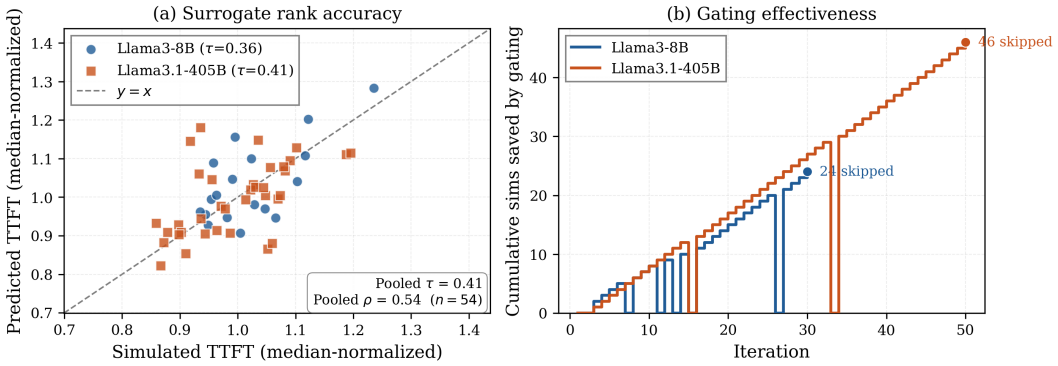

Figure 2: Surrogate ranking quality and gating effectiveness. **(a)** Predicted vs. simulated TTFT on Llama3-8B and Llama3.1-405B prefill (54 proposals total, median-normalized within each workload). The calibrated roofline provides a positive rank signal, with Kendall's $\tau = 0.36$ and 0.41 on the two workloads, respectively, and pooled Kendall's $\tau = 0.41$ / Spearman's $\rho = 0.54$. **(b)** Cumulative simulator calls skipped by roofline-based gating. The gate skips 24 and 46 simulator calls on the two runs, respectively, pruning roughly half of the proposed candidates before simulation.

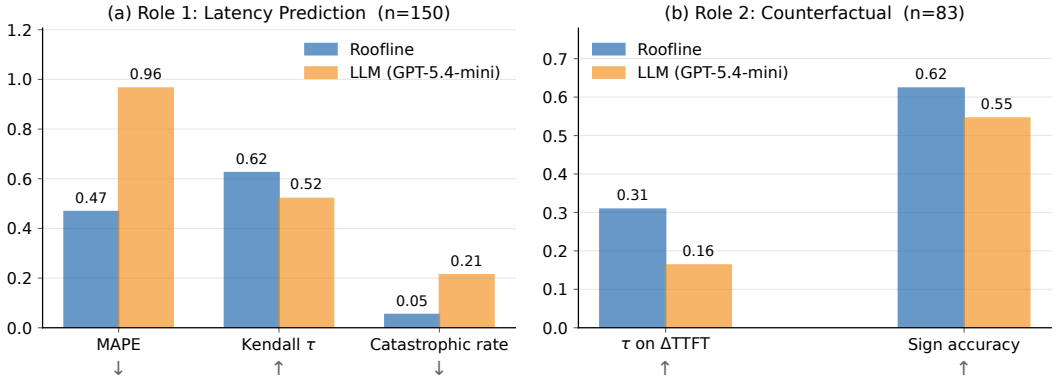

Figure 3: Roofline vs. GPT-5.4-mini on latency prediction (Role 1, n=150) and counterfactual ΔTTFT prediction (Role 2, n=83). Both methods receive identical inputs: workload, reference config with measured TTFT, and target config. Arrows mark lower/higher-is-better.

small latency margins rather than large quality differences: BO, Greedy, and CriticNPU often reach configurations near the same best observed latency. The key difference is simulator cost—CriticNPU reaches near-best designs with far fewer calls, so its main advantage is sample efficiency.

4.2 Detailed Analysis

A - Surrogate Accuracy and Screening Reliability. Figure 2 evaluates what gating actually relies on: the ability of the calibrated roofline to rank configurations, and the sim-budget savings it produces when that ranking is used to screen proposals. Panel (a) plots surrogate-predicted against simulated TTFT on 54 proposals from a 30-iteration run on Llama3-8B prefill and a 50-iteration run on Llama3.1-405B prefill, after normalizing each workload by its median to put both on a common scale. The absolute values are noisy, since the theoretical roofline systematically underestimates latency, but the ordering is positively correlated with the simulator within each workload (Kendall's $\tau = 0.36$ on Llama3-8B, 0.41 on Llama3.1-405B), which is what gating requires. Panel (b) shows the practical consequence: over the course of the two runs, roofline-based gating skipped 24 and 46 simulator calls respectively — roughly half of every proposal the Design Agent issued. The combined message is that the surrogate does not need to predict absolute latency accurately; it only needs to decide whether a candidate is plausibly worse than the current best, and the adaptive-margin gating rule (Eq. 4) uses the moderate but positive ranking signal to prune a large fraction of simulator calls without degrading final design quality (cf. Table 1).

B - Can an LLM Replace the Roofline? Figure 3 asks whether GPT-5.4-mini, the same model CriticNPU uses as the Design Agent, can replace the roofline on the two prediction tasks it performs in the loop. Given identical inputs (workload, reference config with measured TTFT, target config) on 150 Role-1 pairs and 83 Role-2 pairs drawn from the main-table runs, the roofline wins on every metric: 47% vs. 96% MAPE and 5% vs. 21% catastrophic errors on absolute prediction, and $\tau = 0.62$ vs. 0.52 on Role 1 ranking and $\tau = 0.31$ vs. 0.16 on Role 2 ΔTTFT ranking. The LLM's absolute MAPE looks only twice as bad because it hedges by echoing the reference TTFT, since this strategy carries almost no information about which proposed edit is actually an improvement, which is exactly what gating and counterfactual reasoning need. We therefore keep the LLM for high-level design planning and let the roofline do the quantitative surrogate work.

C - Effect of Online Calibration. Figure 4 shows the effect of online calibration on Llama3-8B prefill: the per-type efficiency factors converge to η_{FFN}=0.54 and $\eta_{\text{attention}}$=0.15, placing the two operator classes under two different effective ceilings rather than one shared theoretical peak. The gap between the two ceilings is not uniform: FFN operators give up about half their peak to cache misses and pipeline stalls, while attention operators forfeit nearly six times more because the softmax dependency serializes work that the matrix-multiply ceiling assumes runs in parallel. The resulting ridge shifts flip the regime label of boundary operators such as layer-normalization and softmax, giving the Design Agent quantitatively correct thresholds rather than a single optimistic peak that would mistakenly tag memory-bound ops as compute-bound. This is what lets the gating rule in Eq. 4 prune proposals without losing the good configurations in Table 1, and it is the same calibration signal Role 2 feeds back into the prompt, so the Design Agent's next move is grounded in what the hardware actually delivers instead of what it could in principle reach.

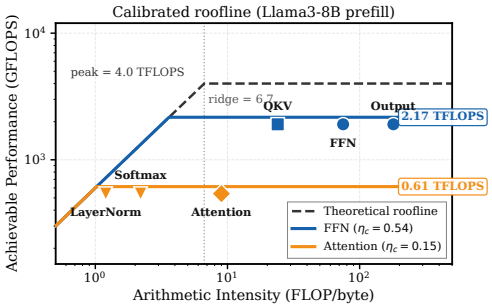

Figure 4: Calibrated roofline for Llama3-8B prefill. Per-type η pulls the FFN ceiling to 54% and attention to 15% of peak, shifting the effective ridge.

D - Additional Analyses. Due to page limitations, we defer eight additional analyses to the appendix. Appendix E.1 shows that CriticNPU converges to near-best designs with fewer simulator calls. Appendix E.2 shows stable final TTFT/TPOT across roofline hyperparameters, including gating, EMA, and stopping settings. Appendix E.3 shows that the method generalizes across different agent backbones. Appendix E.4 reports per-deployment results and confirms the aggregate trends across all constraint and deployment settings. Appendix E.5 validates the CDA operator profiles used by the roofline against NeuSIM's native generator. Appendix E.6 shows that CriticNPU reaches near-optimal valid designs faster than baselines under the same simulation budget. Appendix E.7 shows that roofline gating does not discard meaningfully better candidates. Appendix E.8 shows that CriticNPU remains stable under moderate simulator-noise perturbations.

5 Conclusion and Limitations

We presented CriticNPU, a sample-efficient NPU DSE framework built around an online-calibrated mechanistic critic. The critic turns simulator feedback into practical search guidance: before simulation, it filters weak candidate configurations; after simulation, it summarizes prefill/decode bottlenecks and predicts the effect of candidate parameter changes. Across diverse LLM workloads and deployment settings, CriticNPU reaches near-best latency while using substantially fewer simulator evaluations than competitive baselines.

Limitations. Our evaluation is conducted entirely in simulation, without physical implementation or tape-out validation. The results therefore depend on the fidelity of the NPU simulator; online calibration can reduce mismatch within the simulator, but cannot correct systematic simulator errors. The current critic mainly models compute throughput, HBM bandwidth, and inter-chip communication, and may be less accurate for unmodeled effects such as MoE routing imbalance, cross-expert communication, or compiler scheduling artifacts.

Acknowlegement

This work was partially supported by the Hybrid Cloud and AI program at the IBM-Illinois Discovery Accelerator Institute (IIDAI), and NSF under the grant CAREER CNS-2144796.

References

Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. volume 42, pages 269–284. ACM New York, NY, USA, 2014.

Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *IEEE International Solid-State Circuits Conference (ISSCC)*, 2016.

Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. Visplay: Self-evolving vision-language models from images. *arXiv preprint arXiv:2511.15661*, 2025.

Zongxia Li, Hongyang Du, Chengsong Huang, Xiyang Wu, Lantao Yu, Yicheng He, Jing Xie, Xiaomin Wu, Zhichao Liu, Jiarui Zhang, et al. Mm-zero: Self-evolving multi-model vision language models from zero data. *arXiv preprint arXiv:2603.09206*, 2026.

Zhishang Xiang, Chengyi Yang, Zerui Chen, Zhimin Wei, Yunbo Tang, Zongpei Teng, Zexi Peng, Zongxia Li, Chengsong Huang, Yicheng He, et al. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. 2026.

Yuzong Chen, Chao Fang, Xilai Dai, Yuheng Wu, Thierry Tambe, Marian Verhelst, and Mohamed S Abdelfattah. P3-llm: An integrated npu-pim accelerator for llm inference using hybrid numerical formats. *arXiv preprint arXiv:2511.06838*, 2025.

Jonghun Lee, Junghoon Lee, Hyeonjin Kim, Seoho Jeon, Jisup Yoon, Hyunbin Park, Meejeong Park, and Heonjae Ha. Trigen: Npu architecture for end-to-end acceleration of large language models based on sw-hw co-design. *arXiv preprint arXiv:2602.12962*, 2026.

Wangsong Yin, Daliang Xu, Mengwei Xu, Gang Huang, and Xuanzhe Liu. Shadownpu: System and algorithm co-design for npu-centric on-device llm inference, 2026. URL `https://arxiv.org/abs/2508.16703`.

Hakki Mert Torun, Madhavan Swaminathan, Anto Kavungal Davis, and Mohamed Lamine Faycal Bellaredj. A global bayesian optimization algorithm and its application to integrated system design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(4):792–802, 2018.

Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. Boom-explorer: Risc-v boom microarchitecture design space exploration. *ACM Transactions on Design Automation of Electronic Systems*, 29(1):1–23, 2023a.

Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 622–636. IEEE, 2020.

Chen Bai, Jiayi Huang, Xuechao Wei, Yuzhe Ma, Sicheng Li, Hongzhong Zheng, Bei Yu, and Yuan Xie. Archexplorer: Microarchitecture exploration via bottleneck analysis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 268–282, 2023b.

Tao Zhang, Rui Ma, Shuotao Xu, Peng Cheng, and Yongqiang Xiong. Lumina: Llm-guided gpu architecture exploration via bottleneck analysis. *arXiv preprint arXiv:2603.05904*, 2026.

Mingxin Tang, Wei Chen, Lizhou Wu, Libo Huang, and Kun Zeng. Chatdse: A zero-shot microarchitecture design space explorer powered by gpt4. 0. *ACM Transactions on Design Automation of Electronic Systems*, 30(4):1–24, 2025.

Hanyu Wang, Xinrui Wu, Zijian Ding, Su Zheng, Chengyue Wang, Neha Prakriya, Tony Nowatzki, Yizhou Sun, and Jason Cong. Llm-dse: Searching accelerator parameters with llm agents. *arXiv preprint arXiv:2505.12188*, 2025.

Sujay Pandit, Akanksha Jain, Rami Cohen, Zhijie Deng, Sagar Karandikar, Sagi Perel, Anand Raghunathan, and Parthasarathy Ranganathan. Llm-box: An agentic framework for guided black-box optimization in mapping llms onto specialized hardware accelerators, 2025. NeurIPS 2025 Workshop on Machine Learning for Systems.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Yuqi Xue and Jian Huang. Regate: Enabling power gating in neural processing units. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, pages 1160–1177, 2025.

Takahiro Sato and Kota Watanabe. A topology optimization of on-chip planar inductor based on evolutional on/off method and cma-es. *COMPEL-The international journal for computation and mathematics in electrical and electronic engineering*, 43(4):920–931, 2024.

Yong Wook Kim, Jae Hoon Lee, and Tae Hee Han. Extended worst-case osnr searching algorithm for optical network-on-chip using a semi-greedy heuristic with adaptive scan range. *IEEE Access*, 8: 125863–125873, 2020.

Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

Jamin Seo, Akshat Ramachandran, Yu-Chuan Chuang, Anirudh Itagi, and Tushar Krishna. Airchitect v2: Learning the hardware accelerator design space through unified representations. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7. IEEE, 2025.

Arkapravo Ghosh, Abhishek Moitra, Abhiroop Bhattacharjee, Ruokai Yin, and Priyadarshini Panda. Diffaxe: Diffusion-driven hardware accelerator generation and design space exploration. *arXiv preprint arXiv:2508.10303*, 2025.

Raghav Gupta, Akanksha Jain, Abraham Gonzalez, Alexander Novikov, Po-Sen Huang, Matej Balog, Marvin Eisenberger, Sergey Shirobokov, Ngân Vũ, Martin Dixon, et al. Archagent: Agentic ai-driven computer architecture discovery. *arXiv preprint arXiv:2602.22425*, 2026.

Xingang Guo, Darioush Keivan, Usman Syed, Lianhui Qin, Huan Zhang, Geir Dullerud, Peter Seiler, and Bin Hu. Controlagent: Automating control system design via novel integration of llm agents and domain expertise. *arXiv preprint arXiv:2410.19811*, 2024.

Darioush Kevian, Usman Syed, Xingang Guo, Aaron Havens, Geir Dullerud, Peter Seiler, Lianhui Qin, and Bin Hu. Capabilities of large language models in control engineering: A benchmark study on gpt-4, claude 3 opus, and gemini 1.0 ultra. *arXiv preprint arXiv:2404.03647*, 2024.

Xingang Guo, Yaxin Li, Xiangyi Kong, Yilan Jiang, Xiayu Zhao, Zhihua Gong, Yufan Zhang, Daixuan Li, Tianle Sang, Beixiao Zhu, et al. Toward engineering agi: Benchmarking the engineering design capabilities of llms. *arXiv preprint arXiv:2509.16204*, 2025.

Runkai Li, Jia Xiong, and Xi Wang. idse: Navigating design space exploration in high-level synthesis using llms. *arXiv preprint arXiv:2505.22086*, 2025.

Huanyu Qu, Weihao Zhang, Junfeng Lin, Songchen Ma, Hongyi Li, Luping Shi, and Chengzhong Xu. Mldse: Scaling design space exploration infrastructure for multi-level hardware. *arXiv preprint arXiv:2503.21297*, 2025.

Appendix of "NPU Architecture Design"

Contents

A Roofline Model: A Concrete Example **14**

B Parameters Descriptions **14**

B.1 LLM Specification 14

B.2 LLM Workload 15

B.3 Simulator Outcome 16

B.4 NPU Configuration and Constraints 17

C Related Work **17**

C.1 Architecture-level Design Space Exploration. 17

C.2 Implementation- and Mapping-level Design Space Exploration. 18

C.3 Performance Modeling and Bottleneck Analysis 18

D System Prompt Template **18**

D.1 Prompt for Computing Demand Agent 18

D.2 Prompt for the Architecture Design Agent 20

E Experiments **22**

E.1 Convergence Trajectories 22

E.2 Hyperparameter Sensitivity 23

E.3 Agent Backbone Generalization 23

E.4 Per-Deployment Results 24

E.5 Workload Extraction Fidelity 25

E.6 Equal-Budget Comparison 26

E.7 Gate False-Reject Analysis 26

E.8 Robustness to Simulator Noise 27

A Roofline Model: A Concrete Example

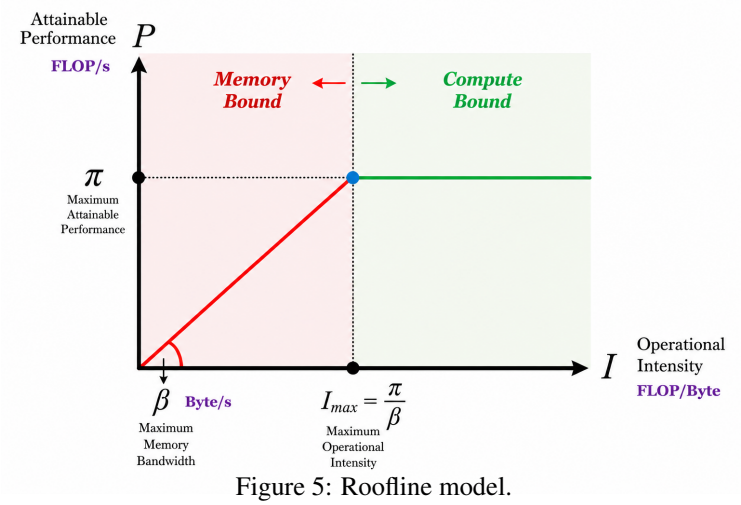

Figure 5: Roofline model.

To make the roofline model more concrete, we consider a simple example.

Hardware. Assume a hardware platform with peak compute throughput $C(\theta) = 10$ TFLOP/s, HBM bandwidth $B(\theta) = 2$ TB/s, and inter-chip communication bandwidth $B_{\text{ici}}(\theta) = 0.5$ TB/s. The compute-memory ridge point is then

$$I_{\text{ridge}}(\theta) = \frac{C(\theta)}{B(\theta)} = \frac{10}{2} = 5.$$

Three operators. Now consider three operators. Operator A has arithmetic intensity $I(o_A) = 2$ and no inter-chip transfer. Operator B has arithmetic intensity $I(o_B) = 20$ and no inter-chip transfer. Operator C has arithmetic intensity $I(o_C) = 10$, with 1 TB of inter-chip transfer and 2 TFLOPs of computation.

Performance prediction. Using the roofline model, operator A has predicted throughput

$$P(o_A) = I(o_A)\, B(\theta) = 2 \times 2 = 4 \text{ TFLOP/s}.$$

Since $I(o_A) < I_{\text{ridge}}(\theta)$, operator A is **memory-bound**.

Operator B has predicted throughput

$$P(o_B) = C(\theta) = 10 \text{ TFLOP/s}.$$

Since $I(o_B) > I_{\text{ridge}}(\theta)$, operator B is **compute-bound**.

Interpretation. This example shows that low-intensity operators are limited by HBM bandwidth, high-intensity operators are limited by compute throughput, and operators with large inter-chip transfer can be limited by communication bandwidth even when their arithmetic intensity is high.

Important caveat. The values above represent ideal upper bounds. In practice, the achieved performance is often lower due to factors such as limited data reuse, kernel inefficiencies, imperfect parallelism, and communication overhead.

B Parameters Descriptions

B.1 LLM Specification

Table 3: LLM specification parameters of DeepSeek-V2-236B.

Name	Description	Value
model_name	Model name	deepseekv2-236b
model_type	Model type	llm
global_batch_size	Global batch size	1
input_seqlen	Input sequence length	4096
output_seqlen	Output sequence length	512
decode_width	Decode width	1
d_model	Hidden dimension	5120
num_heads	Number of attention heads	128
d_head	Dimension of each attention head	128
d_ff	Dense FFN intermediate dimension	12288
moe_d_ff	MoE expert intermediate dimension	1536
num_layers	Number of transformer layers	60
num_dense_layers	Number of dense layers	1
ffn_type	FFN type	deepseek_moe
num_kv_heads	Number of KV heads	128
router_type	Router type	topk
num_shared_experts	Number of shared experts	2
num_routed_experts	Number of routed experts	160
num_activated_routed_experts _per_token	Activated routed experts per token	6
num_limited_groups	Number of limited expert groups	3
q_lora_rank	LoRA rank of query projection	1536
kv_lora_rank	LoRA rank of key/value projection	512
qk_nope_head_dim	Non-RoPE QK head dimension	128
qk_rope_head_dim	RoPE QK head dimension	64
v_head_dim	Value head dimension	128
use_flash_attention	Whether flash attention is enabled	false
data_parallelism_degree	Data parallelism degree	1
tensor_parallelism_degree	Tensor parallelism degree	1
pipeline_parallelism_degree	Pipeline parallelism degree	1
expert_parallelism_degree	Expert parallelism degree	1
num_data_parallel_axes	Number of data-parallel axes	0
num_tensor_parallel_axes	Number of tensor-parallel axes	0
num_pipeline_parallel_axes	Number of pipeline-parallel axes	0
num_expert_parallel_axes	Number of expert-parallel axes	0
data_parallel_degree_dcn	Data parallel degree across DCN	1
tensor_parallel_degree_dcn	Tensor parallel degree across DCN	1
pipeline_parallel_degree_dcn	Pipeline parallel degree across DCN	1
expert_parallel_degree_dcn	Expert parallel degree across DCN	1
microbatch_size_dcn	Microbatch size for DCN execution	1
microbatch_size_ici	Microbatch size for ICI execution	1
num_chips	Number of chips	1
output_file_path	Output CSV file path	v2.csv

B.2 LLM Workload

Table 4: Example workload-level operator profile.

Description	Count	FLOP Count	Input Tensor Shapes	Output Tensor Shapes
FFN-up	32	4.81×10^{11}	$[16, 4096, 4096]$	$[16, 4096, 14336]$

B.3 Simulator Outcome

Table 5: Simulator output fields and their meanings.

Category	Field	Meaning
Latency / Throughput	TTFT_sec	Time to first token, i.e., prefill-phase latency in seconds.
Latency / Throughput	TPOT_ms_request	Time per output token, i.e., per-token decode latency in milliseconds.
Latency / Throughput	throughput_tokens_per_sec_request	Throughput in tokens per second per request.
Memory	mem_footprint_GB_prefill	Memory usage during the prefill phase.
Memory	mem_footprint_GB_decode	Memory usage during the decode phase.
Memory	hbm_size_GB	HBM capacity.
Memory	out_of_memory	Whether the workload exceeds the HBM capacity.
Power	avg_power_W_prefill	Average power during prefill.
Power	avg_power_W_decode	Average power during decode.
Power	max_power_W_prefill	Maximum single-operator instantaneous power during prefill.
Power	max_power_W_decode	Maximum single-operator instantaneous power during decode.
Power	total_power_peak_W	Analytical chip peak power derived from configuration rather than simulation.
Config Info	chip_overrides	The chip configuration evaluated by the simulator.
Config Info	output_dir	Directory where raw outputs are saved.
Raw Operator Data	prefill_stats	Aggregate prefill statistics.
Raw Operator Data	decode_stats	Aggregate decode statistics.
Raw Operator Data	prefill_ops	Per-operator breakdown for prefill, including execution time, systolic-array time, memory time, interconnect time, FLOP count, memory traffic, bounded resource, FLOP utilization, and HBM bandwidth utilization.
Raw Operator Data	decode_ops	Same as prefill_ops, but for decode.

B.4 NPU Configuration and Constraints

Table 6: NPU architecture parameters and hardware constraints.

Name	Explanation	Range / Constraint
num_sa	Number of systolic arrays	2–16
num_vu	Number of vector units	2–12
sa_dim	Systolic array dimension, i.e., $N \times N$	$64/128/256$
freq_GHz	Operating frequency	0.5–2.0 GHz
vmem_size_MB	Vector memory size	32–512 MB
hbm_bw_GBps	HBM bandwidth budget	Specified constraint
hbm_size_GB	HBM capacity budget	Specified constraint
ici_bw_GBps	Inter-chip interconnect bandwidth budget	Specified constraint
dcn_bw_GBps	Data center network bandwidth budget	Specified constraint
static_power_W_per_sa	Static power per systolic array	p_{static}
static_power_W_per_vu	Static power per vector unit	p_{static}
static_power_vmem_W	Static power of vector memory	p_{static}
dynamic_power_W_per_SA	Dynamic power per systolic array	p_{dynamic}
dynamic_power_W_per_VU	Dynamic power per vector unit	p_{dynamic}
dynamic_power_vmem_W	Dynamic power of vector memory	p_{dynamic}
total_chip_power_W	Total chip power constraint	Specified constraint

C Related Work

C.1 Architecture-level Design Space Exploration.

Architecture-level design space exploration (DSE) directly searches over hardware architectures or microarchitectural configurations, where each candidate corresponds to a different processor, GPU, NPU, or accelerator design. For CPU microarchitecture, BOOM-Explorer [Bai et al., 2023a] searches RISC-V BOOM microarchitectural parameters at the RTL level and improves sample efficiency by incorporating microarchitecture domain knowledge into black-box optimization. ArchExplorer [Bai et al., 2023b] introduces bottleneck-aware microarchitecture exploration by constructing microexecution critical paths and reallocating hardware resources according to the identified bottlenecks. ChatDSE [Tang et al., 2025] explores BOOM microarchitectures with GPT-based zero-shot reasoning and pseudo labels to reduce the need for expensive labeled samples. Beyond CPUs, LUMINA [Zhang et al., 2026] performs LLM-guided GPU architecture exploration for LLM workloads by extracting architectural knowledge from simulator code and sensitivity studies. Several works target DNN or AI accelerator architectures. ConfuciuX [Kao et al., 2020] formulates DNN accelerator design as hardware resource assignment and uses reinforcement learning with genetic search to allocate compute and buffer resources under dataflow and platform constraints. AIRCHITECT V2 [Seo et al., 2025] treats accelerator DSE as a learning-based recommendation problem, using transformer-based unified representations to recommend accelerator configurations for given workloads and mappings. DiffAxE [Ghosh et al., 2025] casts accelerator design as a generative problem and uses a diffusion model to generate accelerator configurations conditioned on target performance or energy-delay characteristics. ArchAgent [Gupta et al., 2026] is also related to architecture-level exploration, but focuses on discovering new architectural mechanisms, such as cache replacement policies, rather than only tuning predefined parameters. CriticNPU belongs to this architecture-level category and is most closely related to bottleneck-aware methods such as ArchExplorer and LUMINA. However, unlike CPU critical-path analysis or GPU rule extraction, CriticNPU targets NPU microarchitectural design for LLM inference and uses an online-calibrated roofline model [Williams et al., 2009] to provide

operator-level compute, memory, and communication bottleneck signals. This enables candidate filtering before simulation and structured design guidance after simulation.

C.2 Implementation- and Mapping-level Design Space Exploration.

A second line of work performs DSE above the architecture level, where the hardware substrate, accelerator template, or compilation flow is largely fixed, and the search focuses on how a workload is implemented, compiled, mapped, or deployed. Recently, LLM for engineering [Guo et al., 2024, Kevian et al., 2024, Guo et al., 2025] has become popular. LLM-DSE [Wang et al., 2025] uses a multi-agent LLM framework to optimize HLS directive parameters, such as pipelining, tiling, and parallelization choices, for domain-specific accelerators. iDSE [Li et al., 2025] also targets HLS design optimization, using LLM-guided initialization, pruning, and quality-aware refinement to navigate directive configurations. These HLS-level methods can affect the generated circuit quality, but their design variables are directives mediated through a synthesis toolchain rather than explicit architecture parameters such as compute arrays, SRAM capacity, bandwidth, or interconnect resources. At the system and mapping level, MLDSE [Qu et al., 2025] provides an infrastructure for exploring multi-level hardware organizations and spatiotemporal mappings across hierarchical accelerators, while LLM-Box [Pandit et al., 2025] guides black-box optimization for mapping LLMs onto specialized accelerator systems through choices such as parallelism, collective communication, KV-cache sharding, and interconnect usage. These works optimize implementation or deployment strategies on a given or partially specified substrate. In contrast, CriticNPU directly searches NPU microarchitectural parameters and uses calibrated performance modeling to reason about how changing hardware resources affects operator-level bottlenecks.

C.3 Performance Modeling and Bottleneck Analysis

A complementary line of work develops structured models to capture the relationship between workload characteristics and hardware performance. Analytical models such as the roofline [Williams et al., 2009] provide a physics-grounded abstraction of compute and memory limits, while simulator-based methods perform critical-path or bottleneck analysis to attribute runtime to specific hardware resources [Bai et al., 2023b]. Recent work further leverages LLMs to extract design rules from simulator traces or source code [Zhang et al., 2026]. These approaches expose fine-grained performance structure that is not available from scalar metrics alone, and have been used to interpret simulator outputs or derive design heuristics. However, such models are typically applied after simulation, or used as static predictors that do not adapt to observed execution behavior. Consequently, they provide diagnostic information about existing configurations but are not used to guide exploration before simulator evaluation. Our online-calibrated roofline can also be viewed as a physics-informed surrogate specialized to architecture search; unlike statistical surrogate-assisted optimization that requires many initial samples, the roofline becomes useful after as few as 3 calibration rounds due to its strong physics prior. This separation between prediction and optimization limits their effectiveness in reducing the number of simulator calls required for design space exploration.

D System Prompt Template

D.1 Prompt for Computing Demand Agent

Compute Demand Agent

Role Overview
You are a computing demand analyst for neural accelerator simulation.
Given an LLM model config, parallelism setup, and reference operator-description template, you MUST derive ALL numeric values (Count, FLOP Count, tensor shapes) by applying the formulas below step by step. Do NOT guess or approximate - compute exactly.

Output Format
Call the `compute_demand` tool with TWO keys - `derived_values` first, then `operators`:

```
{ "derived_values": { "B": <number>, "d_model_shard": <number>,
"Nh": <number>, "d_ff_chip": <number>, "layers_per_stage": <number>,
"decode_tokens": <number>, "gathered_dim": <number or null>,
```

"q_proj_b_out": <number or null>, "wo_input_dim": <number or null>, "moe_ff_chip": <number or null>, "shared_ff_chip": <number or null>, "kv_cache_seq": <number or null> }, "operators": [...] }
You MUST compute derived_values FIRST from the config, then use ONLY those values in all operators.

Shape Format Rules

- Input shapes: DT_BFLOAT16:[d1,d2,...]
- Output shapes: [DT_BFLOAT16:(d1,d2,...)]
- Multiple inputs are comma-separated
- Default dtype: DT_BFLOAT16
- Special DT_FLOAT32 ops:
  - RMSNorm / LayerNorm (q_norm, kv_norm)
  - Softmax (gate-scores_softmax)
  - Rotary embedding (apply_rotary_emb)
- Batch: B = global_batch_size / (dp * pp)
- Sequence: Prefill L = input_seqlen, Decode L = 1

Dimension Formulas

```
 tp = tensor_parallelism_degree
pp = pipeline_parallelism_degree
ep = expert_parallelism_degree
dp = data_parallelism_degree
```

- B = global_batch_size / (dp * pp)
- d_model_shard = d_model / tp
- Nh = num_heads / tp
- Nkv = num_kv_heads / tp
- d_ff_chip = d_ff / tp

DeepSeek MLA (ffn_type = deepseek_moe)

- gathered_dim = d_model * tp
- q_proj_b_out = Nh * (qk_nope_head_dim + qk_rope_head_dim)
- wo_input_dim = Nh * v_head_dim
- moe_ff_chip = moe_d_ff * ep / tp
- shared_ff_chip = (d_ff / num_shared_experts) / tp
- kv_cache_seq = input_seqlen + output_seqlen

Count Formulas

- layers_per_stage = CEIL(num_layers / pp)

Standard LLM:

- PREFILL: ICI = 1
- DECODE: ICI = decode_tokens

DeepSeek MoE:

- Attention uses num_layers
- FFN uses layers_per_stage

FLOP Formulas

- Linear: 2 * B * L * M * N
- Batched einsum: 2 * B * L * Nh * D1 * S
- Per-head einsum: 2 * B * L * Nh * D1 * D2
- RMSNorm: 6 * product(input)
- LayerNorm: 8 * product(input)

- Softmax: `4 * product(input)`
- FlashAttention: `4 * B * Nh * L * L * d_head`

Tensor Shape Patterns
Standard LLM

- AllGather: `d_model_shard -> d_model`
- FFN uses `d_ff_chip = d_ff / tp`
- Q/K/V weights are 3D

DeepSeek MLA

- AllGatherMHA: `d_model -> gathered_dim`
- Sequence gather along time dimension
- Residual uses `d_model`

Common Mistakes

1. Use CEIL, not floor
2. Attention uses num_layers
3. MLA uses gathered_dim
4. Nh = num_heads / tp
5. moe_ff_chip = moe_d_ff * ep / tp
6. RMSNorm = factor 6
7. Softmax = factor 4
8. Use commas between tensors

Final Instructions

- Compute `derived_values` first
- Use ONLY derived values
- Keep operator descriptions EXACT
- Verify:
  - `Nh = num_heads / tp`
  - `gathered_dim = d_model * tp`
- Call `compute_demand`

D.2 Prompt for the Architecture Design Agent

Architecture Design Agent

Role Overview
You are an expert NPU (Neural Processing Unit) chip architect.
Your task is to design an NPU chip architecture for a given model, while keeping total power within budget.

Optimization Goal
`{focus}`

Upper-Bound Constraints
These parameters represent technology limits. You may set them, but they MUST NOT exceed maximums.
`{upper_bounds_json}`
You should typically use max values for best performance, but lowering them may reduce power or area.

Power Budget (Hard Constraint)

- Peak power must stay under `{power_budget_W}` W

- `total_power_peak_W = static + dynamic`
- If exceeded → configuration is INVALID
- Reduce power via:
  - fewer compute units
  - lower per-unit power

Baseline Configuration
`{base_chip_config_json}`

Tunable Parameters (Full Search)

- `num_sa`: 2–16
- `num_vu`: 2–12 (ports = num_vu)
- `sa_dim`: 64 / 128 / 256
- `freq_GHz`: 0.5–2.0
- `vmem_size_MB`: 32–512
- `TDP_W`: 100–500

Power parameters:
`{power_params_json}`

Power Formula
`total_power_peak_W = static + dynamic`

- static: `(static_power_W_per_sa * num_sa) + (static_power_W_per_vu * num_vu) + fixed`
- dynamic: `(dynamic_power_W_per_SA * num_sa) + (dynamic_power_W_per_VU * num_vu) + fixed`

Approximation:
`total approx fixed + SA_term + VU_term + vmem`

Key Trade-offs

- More SAs → more compute, more power
- Higher freq → more speed, quadratic power increase
- Larger SA dim → better large matmuls
- Decode is often memory-bound BUT:
  - too few SAs wastes HBM
  - usually need num_sa = 8–12
- Prefill → compute-bound → increase SAs
- Larger VMEM → less HBM traffic
- IMPORTANT: baseline is NOT always optimal

Roofline Model

- Compute ceiling: `num_sa * sa_dim^2 * 2 * freq_GHz`
- BW ceiling: `hbm_bw_GBps`
- Ridge point: `compute / BW`

Interpretation:

- intensity > ridge → compute-bound
- intensity < ridge → memory-bound

Design strategy:

- compute-bound → increase compute
- memory-bound → increase BW or reuse
- watch ridge shifts
- bottlenecks cascade

Exploration Rules

- NEVER repeat previous configs
- Explore drastically different regions

Important Notes

- `num_vu_ports = num_vu`
- Power scales with unit count
- Check memory vs HBM capacity

Final Instruction
Call `propose_npu_config` with your proposed configuration.

E Experiments

The extended experiments in this section cover the three questions most likely to follow from the main-text detailed analysis: whether the claimed simulator-budget savings show up as a faster *trajectory* in addition to a lower final point (D, Sec. E.1), whether the results are robust to the roofline's hyperparameters (E, Sec. E.2), and whether CriticNPU's advantage transfers across different base LLMs (F, Sec. E.3). Each subsection leads with a headline figure and follows with a per-workload table, so a reviewer can skim each experiment in a single page.

E.1 Convergence Trajectories

Figure 6 plots best-so-far latency against cumulative simulator calls across all six workloads, normalized so each run's final best equals 1.0 (lower = closer to its own optimum). Each curve is the median over that method's main-table runs (agent: 41–62 runs per workload-phase; baselines: 2–16 runs depending on coverage), with IQR bands across runs. Table 7 reports the median number of simulator calls required to come within $\{20\%, 10\%, 5\%\}$ of the method's own final best. Figure 7 summarizes the $\leq 5\%$ column as a grouped bar chart. *This measures convergence speed, not absolute quality*; final TTFT / TPOT are already covered by Table 1. A method that converges fast here but has a low win rate there (e.g., CMA-ES on decode) is simply settling into a local minimum quickly. CriticNPU's prefill trajectories close to within 5% of their own best in 15–18 simulator calls across all workloads. Compared with BO and Greedy, CriticNPU is consistently competitive or faster in prefill, while CMA-ES shows workload-dependent behavior: it converges slowly on several Llama workloads but reaches its own-best threshold quickly on DSv2 and DSv3. In decode, several baselines also reach their own final best within a few calls, which reinforces that this threshold analysis should be interpreted together with the absolute win rates in Table 1. Taken together, the convergence curves and main win-rate results show that CriticNPU reaches high-quality configurations with substantially fewer simulator calls.

Table 7: Median simulator calls required for each method to come within $\{20\%, 10\%, 5\%\}$ of its own final best TTFT (prefill) / TPOT (decode). "–" means the method never reaches the threshold. Lower is better; best per column in bold. Per-run normalization; absolute quality in Table 1.

		Llama3-8B			Llama3-70B			Llama3.1-405B			Llama2-13B			DSv2-236B			DSv3-671B		
Phase	**Method**	≤20%	≤10%	≤5%	≤20%	≤10%	≤5%	≤20%	≤10%	≤5%	≤20%	≤10%	≤5%	≤20%	≤10%	≤5%	≤20%	≤10%	≤5%
Prefill	NPUAgent	**11**	**12**	**15**	**13**	**16**	18	**10**	**12**	15	**11**	**15**	**17**	13	14	15	**14**	16	17
	BO	14	14	19	16	17	**17**	12	13	**13**	15	17	20	10	16	16	16	16	16
	CMA-ES	26	48	48	38	51	62	33	44	63	34	45	49	**5**	**6**	**13**	**14**	**14**	**14**
	Greedy	20	23	26	14	23	32	31	32	32	20	40	43	22	22	22	18	19	19
Decode	NPUAgent	8	8	8	8	8	8	8	8	8	8	8	8	9	10	10	10	10	10
	BO	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	**8**
	CMA-ES	**3**	**3**	**3**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**3**	**2**	**3**	**7**	**2**	**2**	**8**
	Greedy	6	6	6	6	6	6	6	6	6	6	6	6	13	15	17	15	18	20

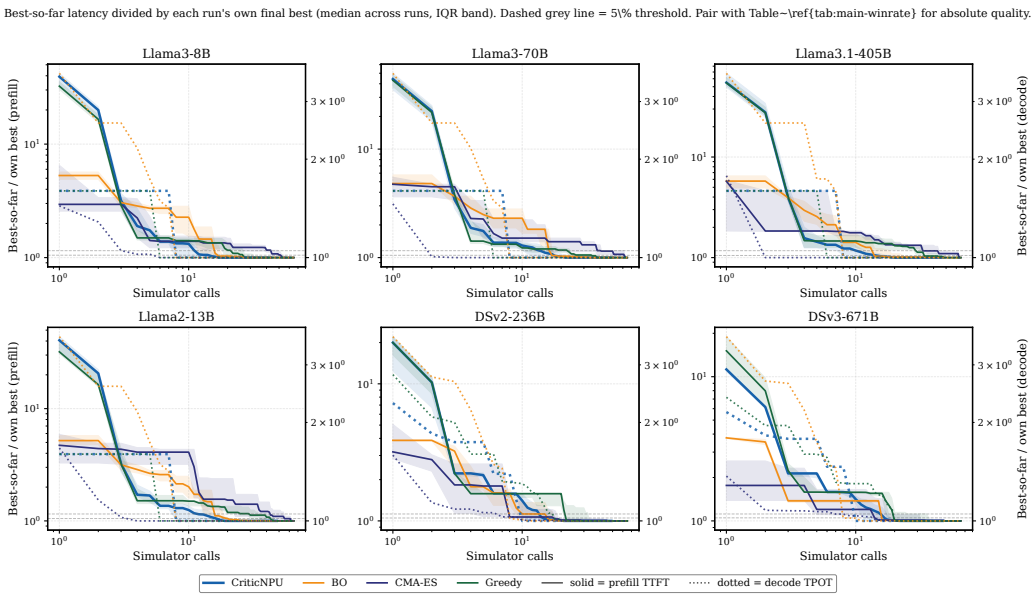

Figure 6: Best-so-far latency vs. simulator calls, normalized by each run's own final best (median across that method's main-table runs, IQR band). Solid = prefill TTFT, dotted = decode TPOT. Dashed grey line marks the 5% threshold. Per-run normalization measures convergence speed only; pair with main table for absolute quality.

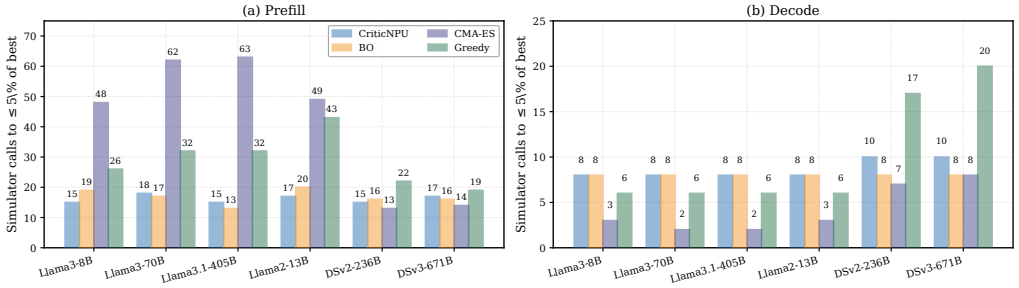

Figure 7: At-a-glance summary of Table 7: simulator calls required to come within 5% of each method's own final best TTFT (a, prefill) and TPOT (b, decode) on each workload. CMA-ES's fast convergence on decode is a local-minimum artefact — its decode win rate in Table 1 is 0–3 of 16.

E.2 Hyperparameter Sensitivity

Three knobs in the roofline module are non-obvious and deserve a robustness check: the gating-margin base ϵ_0 (how aggressively Role 1 skips proposals), the EMA rate λ (how quickly learned $\eta^{(\tau)}$ absorb new observations), and the outer convergence threshold τ_{conv} (the relative-change stopping criterion). Figure 8 sweeps each knob on Llama3-8B (BS=48, 300W, disaggregated) with everything else held at the default, plotting final TTFT (prefill), final TPOT (decode) and the per-phase simulator-call count on twin axes. Table 8 lists the underlying numbers, with the default setting in bold. Final TTFT / TPOT are identical across the entire sweep ($\Delta = 0\%$ in every cell); simulator-call counts vary moderately with the threshold (τ_{conv}=0.01→53 prefill sims; τ_{conv}=0.05→46), and barely with ϵ_0 or λ. The method therefore neither relies on a tuned sweet spot nor degrades under off-default choices of the three knobs.

E.3 Agent Backbone Generalization

We keep the calibrated roofline unchanged and swap the default GPT-5.4-mini used throughout the paper for two other off-the-shelf OpenAI models in *every* LLM-driven role (Compute Demand Agent + Design Agent): GPT-4o (frontier dense) and GPT-4o-mini (lightweight). We run one DSE per

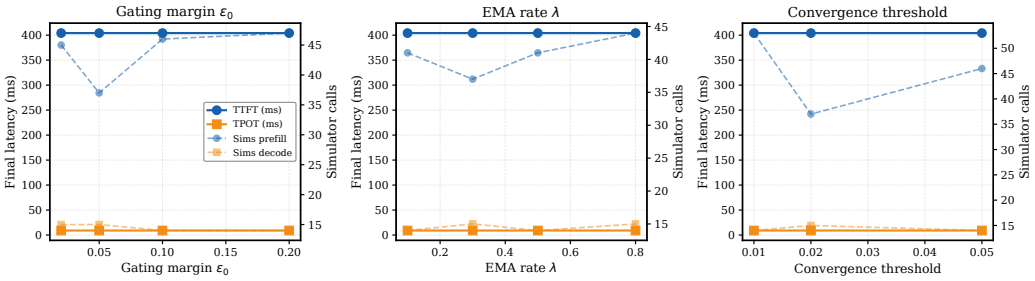

Figure 8: Hyperparameter sensitivity curves on Llama3-8B disaggregated serving (BS=48, 300 W). Each panel sweeps one knob with the other two held at the default. Solid markers = final TTFT / TPOT (left axis, linear); dashed = simulator-call count per phase (right axis). Final quality is flat across the full range of all three knobs.

Table 8: Hyperparameter sensitivity on Llama3-8B disaggregated serving (BS=48, 300 W). One run per setting; default in bold. ΔTTFT is relative change vs. the default.

		Prefill		Decode		
Knob	**Value**	**TTFT (ms)**	**Sims**	**TPOT (ms)**	**Sims**	**ΔTTFT**
Gating margin ϵ_0	0.02	404.2	45	9.03	15	+0%
	0.05	**404.2**	**37**	**9.03**	**15**	**0%**
	0.10	404.2	46	9.03	14	+0%
	0.20	404.2	47	9.03	14	+0%
EMA rate λ	0.1	404.2	41	9.03	14	+0%
	0.3	**404.2**	**37**	**9.03**	**15**	**0%**
	0.5	404.2	41	9.03	14	+0%
	0.8	404.2	44	9.03	15	+0%
Convergence threshold τ_{conv}	0.01	404.2	53	9.03	14	+0%
	0.02	**404.2**	**37**	**9.03**	**15**	**0%**
	0.05	404.2	46	9.03	14	+0%

(workload, LLM) on the same BS=48, 300 W disaggregated deployment used in the main-table column. Figure 9 shows the resulting final TTFT / TPOT per LLM across all six workloads, and Table 12 gives the underlying numbers. Final latencies are within 5% of the default on every workload; the largest single deviation is GPT-4o on Llama3-8B prefill (+2.1% TTFT). Simulator-call counts drift by at most 20–30% across LLMs. CriticNPU's advantage therefore comes from the calibrated roofline, not from properties specific to a particular base LLM.

E.4 Per-Deployment Results

Tables 13–28 report detailed per-deployment results across all 16 constraint settings, including TTFT/TPOT and simulator calls for each method. Across deployments, CriticNPU consistently achieves competitive or best latency while maintaining a substantially lower simulator budget, with improvements most pronounced under tighter power and bandwidth constraints. We observe that classical methods (e.g., BO, CMA-ES, Greedy) occasionally match or exceed latency in individual settings, but require significantly more simulator calls to do so, reflecting lower sample efficiency. LLM-based baselines without a physics prior (LLM-DSE, LLM-Box) exhibit higher variance across deployments and tend to underperform in regimes where bottleneck transitions are critical. The advantage of CriticNPU is particularly evident in deployments with shifting bottlenecks, where the calibrated roofline enables more reliable prediction and pruning of suboptimal configurations. Overall, these per-deployment results corroborate the aggregate trends reported in Table 1, showing that the gains in both design quality and simulator efficiency are robust across diverse hardware constraints.

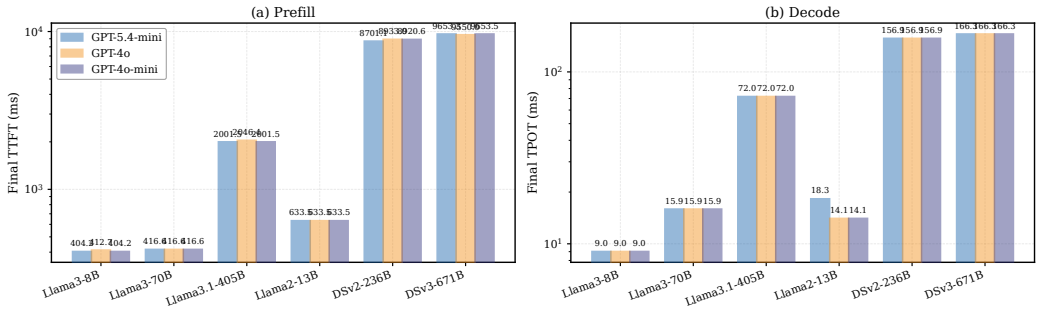

Figure 9: Final TTFT (a, prefill) and TPOT (b, decode) of CriticNPU across six workloads under three different base LLMs. The roofline module is identical in all runs; only the LLM roles (CDA, Design Agent) differ. Bars within each workload are near-identical, demonstrating that CriticNPU's quality does not depend on the specific base LLM.

Table 9: Workload-extraction fidelity. CDA outputs are joined to NeuSIM's native ops-generator CSVs by operator description and compared per-operator. ΣFLOP-RE is the relative error on the configuration-total FLOP count; per-op median / p90 are the relative errors on the per-operator FLOP count; counts are exact-match rates of the per-iteration `Count` field. "n" is the number of matched operators per phase.

Model	Config	Phase	Name match	Count match	ΣFLOP-RE	op-med / p90 RE	n
Llama3.1-405B	TP=16, PP=4	prefill	100%	100%	0.000	0.00 / 0.00	19
	TP=16, PP=4	decode	100%	100%	0.000	0.00 / 0.00	17
	TP=8, PP=8	prefill	100%	100%	0.000	0.00 / 0.00	19
	TP=8, PP=8	decode	100%	100%	0.000	0.00 / 0.00	17
	TP=4, PP=16	prefill	100%	100%	0.000	0.00 / 0.00	19
	TP=4, PP=16	decode	100%	100%	0.000	0.00 / 0.00	17
DSv2-236B	TP=2, PP=16, EP=2	prefill	100%	100%	0.000	0.00 / 0.00	55
	TP=2, PP=16, EP=2	decode	100%	100%	0.026	0.00 / 0.00	55

E.5 Workload Extraction Fidelity

Because the calibrated roofline rests on the operator profile produced by the Compute Demand Agent (CDA, Section 3.1), we directly check that the LLM-based extractor reproduces the operator-level numerics that NeuSIM's hand-coded ops generator would emit on the same model and parallelism setting. We use NeuSIM's native ops generator outputs (`NeuSim/results/raw/<model>/<config>/inference-*_{prefill,decode}.csv`) as the reference, and join them with CDA's outputs (`chip_agent/results/...`) by operator description for every (model, parallelism) pair where both are available—three Llama3.1-405B configs (TP/PP $\in$ {16/4, 8/8, 4/16}, BS=16) and one DSv2-236B MoE config (TP=2, PP=16, EP=2, BS=16), covering both dense and Mixture-of-Experts graphs. For each matched operator we compare the operator count and FLOP count; we also check that the input/output tensor-shape strings, which the simulator later turns into byte traffic, match exactly. Table 9 reports the result. Across all four configurations and both phases, every operator name in the NeuSIM reference is also produced by the CDA (name-match rate 100%), every operator's per-iteration count matches exactly, and the per-operator FLOP count agrees with the reference to numerical precision (median and 90th-percentile relative error both 0, total-FLOP relative error $\leq 2.6\%$, with the only nonzero entry coming from the DSv2-236B decode KV-cache tensor whose per-token sequence indexing changes by a single time step between the two extractors). Because the tensor-shape strings are reproduced exactly, the byte-traffic numbers the simulator computes downstream are identical to the reference as well, so the bottleneck labels CriticNPU's roofline assigns to operators are not merely close to but *the same as* those NeuSIM's native pipeline would assign on the same configuration. We have NeuSIM-side ground truth only for these two model families (the other workloads in our main table are not shipped with reference operator CSVs), so this fidelity check covers a representative dense + MoE pair rather than the full main-table sweep; we leave a broader systematic sweep to future work.

Table 10: Sample efficiency: simulator calls each method needs to come within 2% of the universal best for that (model, deployment, phase) cell, aggregated over six workloads $\times$ 16 deployment settings (96 cells per phase). *Universal best* = lowest valid latency any method reached anywhere in its 100-call trajectory on that cell. Lower is better; "–" means the method does not reach within 2% on any cell. ChatDSE is omitted because it consumes only ≈ 2 simulator calls per run by design and never reaches the 2% target.

	Prefill (# sim calls)			**Decode (# sim calls)**		
Method	**median**	**mean**	**p90**	**median**	**mean**	**p90**
CriticNPU (Ours)	**14**	**17.6**	**32**	**8**	9.8	12
BO	22	24.0	42	**8**	9.9	16
CMA-ES	41	39.5	55	6	**8.2**	**20**
Greedy	30	30.2	51	7	13.2	32
LLM-DSE	47	43.5	58	23	26.7	52
LLM-Box	55	51.0	59	53	46.5	60

E.6 Equal-Budget Comparison

A natural concern with the headline numbers in Table 1 is that CriticNPU simply stops earlier than the classical baselines—it averages 50–57 simulator calls per run, while BO, CMA-ES, and Greedy use 117–130, so a winner-takes-all win rate at fixed final budget mixes "how fast did the method converge" with "how good is the converged design." To separate the two, we score sample efficiency directly: for each (model, deployment, phase) cell we record the lowest valid latency *any* method reached anywhere in its 100-call trajectory—call this the *universal best* for that cell—and then ask each method, "how many simulator calls did you need before your best-so-far came within 2% of that universal best?" The metric is a per-cell scalar (a count, not a winner flag), so a method that reaches a tied-for-best design fast is rewarded equally whether or not another method later finds a 0.1% improvement. Configurations that exceed the cell's power budget are filtered out before the trajectory is scored. Table 10 aggregates this metric across all six workloads and 16 deployment settings each (96 cells per phase). On prefill, CriticNPU reaches within 2% of universal best in a median of **14** simulator calls, $1.6\times$ faster than BO (22), $2.2\times$ faster than Greedy (30), and $3.4\times$ faster than the strongest LLM-only baseline LLM-DSE (47); the mean and 90th-percentile counts confirm the same ordering. The decode-side budget is tighter for everyone because the decode search space is small (CriticNPU: median 8 calls), but the LLM-only baselines that lack a physics prior pay a clear cost—LLM-DSE needs $\approx 3\times$ more decode calls to reach the same target. The interpretation matches the main-text claim. CriticNPU does not necessarily find a strictly lower latency than every alternative on every cell—once budget is unconstrained, several baselines eventually arrive at the same near-optimal designs—but it is the only method that consistently arrives there inside the budget where simulator cost is the binding constraint. ChatDSE never reaches within 2% on any cell because its bypass-the-simulator design routinely exits at a power-violating configuration.

E.7 Gate False-Reject Analysis

The Role-1 gate skips a candidate whenever its roofline-predicted latency exceeds the current best (Eq. 4); a natural reviewer concern is whether the gate accidentally throws away configurations that would have improved the design if simulated. We test this directly on Llama3-8B decode across all 16 deployment settings: we parse the agent logs to recover every Design-Agent proposal and its decision (simulated or skipped), re-simulate 60 of the 91 skipped candidates with NeuSIM (clean, no noise), and compare the recovered ground-truth TPOT against the best-so-far TPOT at the moment the proposal was issued. We classify a skip as a *false-reject at threshold* ϵ iff its actual TPOT undercuts the current best by at least ϵ:

$$T_{\text{actual}}(\theta_{\text{skip}}) < (1-\epsilon)\, T_t^*.$$

At ϵ=0 (any improvement, however small), 27/60 skipped candidates re-simulate to a TPOT below the then-best—but inspection shows these are essentially ties: in every case the actual TPOT is within 0.01 ms of the best, the consequence of CriticNPU's small decode design space producing many configurations whose runtime collapses onto the same HBM-bandwidth-limited frontier. At ϵ=1% the false-reject rate drops to **0/60**; at ϵ=5% it remains **0/60**. In other words, the gate never discards a configuration that would have meaningfully improved the design. The top-3 / top-5 /

Table 11: Robustness to multiplicative simulator noise. CriticNPU runs end-to-end against a NeuSIM perturbed with $T(o) \to T(o)(1 + \mathcal{N}(0, \sigma^2))$; the best configuration found is then re-evaluated on the clean simulator. Llama3-8B, TP=4, PP=2, BS=16, 300W disaggregated.

σ	**noisy-search TTFT**	**clean-eval TTFT**	**clean-eval TPOT**	**Sims**
0.00	0.135 s	0.135 s	4.526 ms	52
0.10	0.115 s	0.135 s	4.526 ms	59
0.20	0.103 s	0.164 s	6.335 ms	52

top-10 "recall" (fraction of the lowest-TPOT configurations that the gate kept for simulation) sit at 48%/49%/51%, which corroborates the same picture: the gate skips roughly half of the cheapest configurations because they are tied with the current best, not because it misses better ones. The gating-margin sensitivity in Table 8 is consistent with this: final TTFT/TPOT are unchanged across $\epsilon_0 \in \{0.02, 0.05, 0.10, 0.20\}$, so the choice of margin only trades simulator calls for nothing on the quality side.

E.8 Robustness to Simulator Noise

Because the entire evaluation runs inside NeuSIM, a separate concern is whether CriticNPU simply optimises simulator quirks rather than discovering generally good NPU configurations. We do not have an alternative simulator or hardware to validate against, so as a robustness proxy we inject multiplicative Gaussian noise into NeuSIM's per-operator execution times (and therefore TTFT, TPOT, and chip power) at run time:

$$\tilde{T}(o) = T(o) \cdot (1 + \xi_o), \quad \xi_o \sim \mathcal{N}(0, \sigma^2),\ \sigma \in \{0,\ 0.10,\ 0.20\}.$$

CriticNPU runs inside the noisy simulator (so every gating decision and every Design-Agent reasoning step sees noisy feedback). After each run we re-evaluate the resulting best configuration on the *clean* simulator (σ=0) to recover its true latency. Table 11 reports the result on Llama3-8B (TP=4, PP=2, BS=16, 300W disaggregated). At σ=0.10 CriticNPU recovers *exactly* the same end-of-trajectory operating point as the noiseless run (clean-eval TTFT/TPOT identical to four decimal places); at σ=0.20 the search lands on a neighbouring configuration with 21% higher true TTFT and 40% higher TPOT—degraded but still well below the typical baseline TTFT on this workload. The simulator-call budget grows by at most 7 calls across noise levels. The gate stays effective because its decision compares predicted latency to the current best, so the multiplicative noise term largely cancels; the EMA-calibrated $\eta^{(\tau)}$ further damps per-operator noise across iterations. We do not claim NeuSIM is calibrated to physical NPU silicon, but the result indicates that CriticNPU's design does not depend on a fragile region of NeuSIM's response surface up to a 10% noise level, with graceful degradation at 20%.

Table 12: LLM generalization: CriticNPU with three different base LLMs across six workloads, reporting both phases (matching the main-table split). One DSE run per (workload, LLM) at BS=48, 300 W disaggregated. Δ columns give the relative change vs. GPT-5.4-mini on that latency.

		Prefill		**Decode**			
Workload	**LLM**	**TTFT (ms)**	**Sims**	**TPOT (ms)**	**Sims**	**ΔTTFT**	**ΔTPOT**
Llama3-8B	GPT-5.4-mini (default)	404.2	42	9.03	16	—	—
	GPT-4o	412.7	64	9.03	15	+2.1%	0%
	GPT-4o-mini	404.2	70	9.03	13	0%	0%
Llama3-70B	GPT-5.4-mini (default)	416.6	37	15.93	13	—	—
	GPT-4o	416.6	68	15.93	13	0%	0%
	GPT-4o-mini	416.6	60	15.93	13	0%	0%
Llama3.1-405B	GPT-5.4-mini (default)	2001.5	39	71.96	15	—	—
	GPT-4o	2046.4	64	71.97	13	+2.2%	0%
	GPT-4o-mini	2001.5	73	71.96	13	0%	0%
Llama2-13B	GPT-5.4-mini (default)	633.5	41	18.29	13	—	—
	GPT-4o	633.5	57	14.07	15	0%	−23%
	GPT-4o-mini	633.5	50	14.07	14	0%	−23%
DSv2-236B	GPT-5.4-mini (default)	8701.1	41	156.89	15	—	—
	GPT-4o	8933.0	60	156.89	15	+2.7%	0%
	GPT-4o-mini	8920.6	64	156.89	15	+2.5%	0%
DSv3-671B	GPT-5.4-mini (default)	9653.3	42	166.27	13	—	—
	GPT-4o	9550.0	66	166.27	14	−1.1%	0%
	GPT-4o-mini	9653.5	56	166.27	15	0%	0%

Table 13: Config C1. Small models (8B, 13B): 8 chips, TP=2, PP=4, BS=16, 250W; Large models (70B, 405B, 671B): 64 chips, TP=4, PP=16, BS=16, 250W; DSv2-236B: 16 chips, TP=2, PP=8, BS=16, 250W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.1661**	110	0.2644	110	**0.1689**	110	0.8817	110	**4.59**	110	3.40	110
	CMA-ES	0.2270	130	**0.2631**	130	**0.1689**	130	1.06	130	4.60	130	3.42	130
	Greedy	**0.1661**	126	**0.2631**	130	**0.1689**	130	0.8796	130	**4.59**	130	**3.34**	129
	Grid	0.1927	1440	0.3030	1440	0.1888	1440	**0.8465**	1440	12.10	1440	5.17	1440
	ChatDSE	0.3205	\	0.4874	\	0.3515	\	1.60	\	17.37	\	12.28	\
	LLM-DSE	**0.1661**	109	0.7707	113	0.5431	110	0.8796	110	10.26	104	**3.34**	108
	LLM-Box	0.1998	108	**0.2631**	118	0.1795	117	0.9457	117	6.36	118	**3.34**	101
	CriticNPU	**0.1661**	49	**0.2631**	50	**0.1689**	46	0.8796	49	**4.59**	50	**3.34**	48
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**5.99**	110	**9.54**	110	**20.44**	110	**114.35**	110	**71.21**	110	**97.31**	110
	CMA-ES	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	71.24	130	97.41	130
	Greedy	**5.99**	126	**9.54**	130	**20.44**	130	**114.35**	130	**71.21**	130	97.41	129
	Grid	9.58	1440	15.26	1440	32.71	1440	320.18	1440	199.39	1440	155.70	1440
	ChatDSE	22.52	\	37.77	\	81.36	\	448.25	\	284.83	\	373.64	\
	LLM-DSE	**5.99**	109	**9.54**	113	**20.44**	110	**114.35**	110	72.75	104	98.38	108
	LLM-Box	**5.99**	108	**9.54**	118	**20.44**	117	**114.35**	117	**71.21**	118	98.38	101
	CriticNPU	**5.99**	49	**9.54**	50	**20.44**	46	**114.35**	49	75.49	50	100.92	48

Table 14: Config C2. Small models (8B, 13B): 8 chips, TP=2, PP=4, BS=16, 300W; Large models (70B, 405B, 671B): 64 chips, TP=4, PP=16, BS=16, 300W; DSv2-236B: 16 chips, TP=2, PP=8, BS=16, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.1255**	114	0.2002	114	0.1243	114	0.6229	114	4.48	114	3.25	114
	CMA-ES	0.1487	130	0.3503	130	0.1495	130	0.6855	130	5.07	130	3.63	130
	Greedy	**0.1255**	130	**0.1975**	130	**0.1236**	130	**0.6208**	129	**4.44**	130	**3.24**	130
	Grid	0.1629	1440	0.2538	1440	0.1576	1440	0.8465	1440	12.10	1440	5.13	1440
	ChatDSE	0.3194	\	0.4119	\	0.3563	\	1.17	\	17.00	\	12.86	\
	LLM-DSE	0.3560	111	0.2538	114	0.3731	101	2.10	103	**4.44**	97	4.98	108
	LLM-Box	0.1766	119	**0.1975**	117	**0.1236**	108	0.7199	114	6.32	117	3.27	119
	CriticNPU	**0.1255**	53	0.2002	54	**0.1236**	49	**0.6208**	51	**4.44**	56	3.27	57
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**5.99**	114	**9.54**	114	**20.44**	114	**114.35**	114	**71.21**	114	**97.31**	114
	CMA-ES	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	71.25	130	97.51	130
	Greedy	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	129	**71.21**	130	**97.31**	130
	Grid	9.58	1440	15.26	1440	32.71	1440	320.18	1440	199.39	1440	155.70	1440
	ChatDSE	23.48	\	33.19	\	78.91	\	443.68	\	277.71	\	389.21	\
	LLM-DSE	**5.99**	111	**9.54**	114	**20.44**	101	**114.35**	103	**71.21**	97	97.35	108
	LLM-Box	**5.99**	119	**9.54**	117	**20.44**	108	**114.35**	114	**71.21**	117	153.75	119
	CriticNPU	**5.99**	53	**9.54**	54	**20.44**	49	**114.35**	51	**71.21**	56	**97.31**	57

Table 15: Config C3. Small models (8B, 13B): 8 chips, TP=2, PP=4, BS=16, 400W; Large models (70B, 405B, 671B): 64 chips, TP=4, PP=16, BS=16, 400W; DSv2-236B: 16 chips, TP=2, PP=8, BS=16, 400W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.0980**	130	**0.1507**	130	**0.0944**	130	**0.4360**	130	4.37	130	**3.20**	130
	CMA-ES	0.1579	130	0.2350	130	0.1829	130	0.6954	130	5.07	130	3.63	130
	Greedy	**0.0980**	130	**0.1507**	128	**0.0944**	130	**0.4360**	130	4.81	130	**3.20**	130
	Grid	0.1391	1440	0.2114	1440	0.1319	1440	0.8465	1440	12.10	1440	5.10	1440
	ChatDSE	0.3694	\	0.4663	\	0.3146	\	1.24	\	17.29	\	14.86	\
	LLM-DSE	0.2305	104	**0.1507**	102	0.0961	98	**0.4360**	109	**4.36**	112	3.82	102
	LLM-Box	**0.0980**	109	0.1837	115	**0.0944**	118	0.4789	120	**4.36**	113	4.14	118
	CriticNPU	**0.0980**	64	**0.1507**	62	**0.0944**	63	0.4483	61	4.39	68	**3.20**	60
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	**71.21**	130	**97.31**	130
	CMA-ES	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	71.80	130	97.65	130
	Greedy	**5.99**	130	**9.54**	128	**20.44**	130	**114.35**	130	**71.21**	130	**97.31**	130
	Grid	9.58	1440	15.26	1440	32.71	1440	320.18	1440	199.39	1440	155.70	1440
	ChatDSE	23.96	\	36.24	\	80.95	\	457.40	\	284.83	\	385.32	\
	LLM-DSE	**5.99**	104	**9.54**	102	**20.44**	98	**114.35**	109	**71.21**	112	**97.31**	102
	LLM-Box	**5.99**	109	**9.54**	115	**20.44**	118	**114.35**	120	**71.21**	113	**97.31**	118
	CriticNPU	**5.99**	64	**9.54**	62	**20.44**	63	**114.35**	61	**71.21**	68	**97.31**	60

Table 16: Config C4. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=16, 250W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=16, 250W; DSv2-236B: 16 chips, TP=4, PP=4, BS=16, 250W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.1741	110	**0.2781**	110	**0.1835**	110	0.9326	110	**2.98**	110	3.29	110
	CMA-ES	0.1797	130	**0.2781**	130	0.2106	130	**0.9266**	130	2.99	130	3.45	130
	Greedy	**0.1731**	126	**0.2781**	130	0.1930	130	**0.9266**	130	**2.98**	130	**3.29**	130
	Grid	0.2098	1440	0.4179	1440	0.2184	1440	0.9808	1440	7.97	1440	5.14	1440
	ChatDSE	0.3921	\	0.6371	\	0.4016	\	1.35	\	10.85	\	12.87	\
	LLM-DSE	**0.1731**	107	0.7818	113	0.1949	116	0.9998	111	5.97	106	**3.29**	113
	LLM-Box	0.1899	107	0.3337	120	0.1892	115	**0.9266**	119	3.59	118	**3.29**	112
	CriticNPU	**0.1731**	49	**0.2781**	49	**0.1835**	47	**0.9266**	48	2.99	51	**3.29**	54
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.53**	110	**7.03**	110	**13.11**	110	**62.93**	110	**63.43**	110	**80.32**	110
	CMA-ES	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	63.48	130	**80.32**	130
	Greedy	**4.53**	126	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Grid	7.24	1440	19.69	1440	20.98	1440	176.21	1440	177.59	1440	128.50	1440
	ChatDSE	17.01	\	28.13	\	51.40	\	241.66	\	238.47	\	321.24	\
	LLM-DSE	**4.53**	107	**7.03**	113	**13.11**	116	**62.93**	111	66.41	106	82.41	113
	LLM-Box	**4.53**	107	8.02	120	**13.11**	115	81.81	119	68.50	118	**80.32**	112
	CriticNPU	**4.53**	49	**7.03**	49	**13.11**	47	**62.93**	48	71.28	51	86.27	54

Table 17: Config C5. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=16, 300W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=16, 300W; DSv2-236B: 16 chips, TP=4, PP=4, BS=16, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.1354	114	0.2128	114	0.1423	114	0.6708	114	2.91	114	3.22	114
	CMA-ES	0.1524	130	0.2527	130	0.1641	130	0.8157	130	2.99	130	3.26	130
	Greedy	**0.1351**	130	**0.2112**	130	0.1495	130	**0.6678**	130	**2.90**	130	3.22	130
	Grid	0.1859	1440	0.4179	1440	0.1927	1440	0.9808	1440	7.97	1440	5.12	1440
	ChatDSE	0.4050	\	0.5719	\	0.3896	\	1.88	\	13.56	\	12.72	\
	LLM-DSE	**0.1352**	105	**0.2112**	108	0.2058	108	2.23	101	4.50	117	**3.22**	99
	LLM-Box	0.1781	116	0.2784	119	0.2202	119	0.9300	119	2.91	120	3.22	107
	CriticNPU	**0.1351**	53	**0.2112**	49	**0.1408**	54	0.6768	54	**2.90**	53	3.22	56
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.53**	114	**7.03**	114	**13.11**	114	**62.93**	114	**63.43**	114	**80.32**	114
	CMA-ES	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	63.71	130	**80.32**	130
	Greedy	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Grid	7.24	1440	19.69	1440	20.98	1440	176.21	1440	177.59	1440	128.50	1440
	ChatDSE	18.10	\	27.01	\	49.83	\	247.95	\	244.82	\	319.63	\
	LLM-DSE	**4.53**	105	**7.03**	108	**13.11**	108	**62.93**	101	**63.43**	117	**80.32**	99
	LLM-Box	**4.53**	116	**7.03**	119	**13.11**	119	**62.93**	119	66.13	120	**80.32**	107
	CriticNPU	**4.53**	53	**7.03**	49	**13.11**	54	**62.93**	54	**63.43**	53	80.98	56

Table 18: Config C6. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=16, 400W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=16, 400W; DSv2-236B: 16 chips, TP=4, PP=4, BS=16, 400W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.1124**	130	0.1755	130	0.1184	130	0.4889	130	2.87	130	**3.20**	130
	CMA-ES	0.1691	130	0.2781	130	0.2106	130	0.9878	130	2.91	130	3.45	130
	Greedy	**0.1124**	129	**0.1731**	130	**0.1163**	130	**0.4829**	130	**2.86**	130	**3.20**	130
	Grid	0.1649	1440	0.4179	1440	0.1680	1440	0.9808	1440	7.97	1440	5.11	1440
	ChatDSE	0.8879	\	0.6009	\	0.9558	\	1.33	\	10.53	\	12.80	\
	LLM-DSE	0.1125	101	**0.1731**	92	**0.1163**	94	**0.4829**	101	3.45	114	**3.20**	102
	LLM-Box	0.1364	117	**0.1731**	118	0.2586	107	0.8270	119	**2.86**	115	**3.20**	88
	CriticNPU	**0.1124**	61	0.1837	65	0.1236	65	**0.4829**	63	2.87	68	**3.20**	62
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	CMA-ES	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Greedy	**4.53**	129	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Grid	7.24	1440	19.69	1440	20.98	1440	176.21	1440	177.59	1440	128.50	1440
	ChatDSE	17.01	\	28.13	\	52.45	\	234.11	\	234.67	\	321.24	\
	LLM-DSE	**4.53**	101	**7.03**	92	**13.11**	94	**62.93**	101	**63.43**	114	**80.32**	102
	LLM-Box	**4.53**	117	**7.03**	118	**13.11**	107	115.79	119	91.33	115	168.65	88
	CriticNPU	**4.53**	61	**7.03**	65	**13.11**	65	**62.93**	63	**63.43**	68	**80.32**	62

Table 19: Config C7. Small models (8B, 13B): 8 chips, TP=8, PP=1, BS=16, 250W; Large models (70B, 405B, 671B): 64 chips, TP=16, PP=4, BS=16, 250W; DSv2-236B: 16 chips, TP=8, PP=2, BS=16, 250W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.2059	110	0.3273	110	0.2336	110	1.07	110	**2.79**	110	**5.07**	110
	CMA-ES	0.2249	130	**0.3250**	130	0.2336	130	**1.05**	130	2.79	130	5.07	130
	Greedy	**0.2057**	129	**0.3250**	130	**0.2333**	127	**1.05**	129	**2.79**	130	**5.07**	130
	Grid	0.4192	1440	0.6378	1440	0.3116	1440	1.59	1440	7.62	1440	14.05	1440
	ChatDSE	0.5998	\	0.9162	\	0.6088	\	2.53	\	10.25	\	19.88	\
	LLM-DSE	**0.2057**	114	**0.3250**	109	0.2336	107	1.08	111	**2.79**	106	6.05	111
	LLM-Box	0.2721	119	**0.3250**	118	0.2859	119	**1.05**	119	**2.79**	116	6.05	76
	CriticNPU	**0.2057**	34	**0.3250**	40	**0.2333**	50	**1.05**	49	**2.79**	46	**5.07**	54
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.27**	110	**6.37**	110	**11.81**	110	**41.00**	110	**97.87**	110	**108.48**	110
	CMA-ES	**4.27**	130	**6.37**	130	**11.81**	130	**41.00**	130	**97.87**	130	**108.48**	130
	Greedy	**4.27**	129	**6.37**	130	**11.81**	127	**41.00**	129	**97.87**	130	**108.48**	130
	Grid	11.94	1440	17.85	1440	18.89	1440	114.80	1440	274.03	1440	303.73	1440
	ChatDSE	17.06	\	25.49	\	41.31	\	162.36	\	367.98	\	429.56	\
	LLM-DSE	**4.27**	114	**6.37**	109	**11.81**	107	**41.00**	111	**97.87**	106	112.78	111
	LLM-Box	**4.27**	119	**6.37**	118	**11.81**	119	44.28	119	**97.87**	116	190.92	76
	CriticNPU	4.48	34	**6.37**	40	**11.81**	50	**41.00**	49	112.41	46	118.54	54

Table 20: Config C8. Small models (8B, 13B): 8 chips, TP=8, PP=1, BS=16, 300W; Large models (70B, 405B, 671B): 64 chips, TP=16, PP=4, BS=16, 300W; DSv2-236B: 16 chips, TP=8, PP=2, BS=16, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.1739**	114	0.2728	114	**0.1990**	114	**0.7974**	114	2.75	114	**5.03**	114
	CMA-ES	0.2249	130	0.4090	130	0.2542	130	1.05	130	2.93	130	5.10	130
	Greedy	**0.1739**	130	**0.2694**	130	**0.1990**	129	0.8593	130	**2.74**	130	**5.03**	130
	Grid	0.4192	1440	0.6378	1440	0.2859	1440	1.59	1440	7.62	1440	14.05	1440
	ChatDSE	0.5687	\	0.9331	\	0.6837	\	2.13	\	10.88	\	19.88	\
	LLM-DSE	0.3642	108	0.5940	104	0.1993	102	2.28	111	2.75	109	5.58	115
	LLM-Box	0.3147	119	0.3310	118	**0.1990**	116	1.17	117	2.77	120	5.03	77
	CriticNPU	0.2496	34	0.2818	31	**0.1990**	54	0.8142	55	**2.74**	54	**5.03**	56
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.27**	114	**6.37**	114	**11.81**	114	**41.00**	114	**97.87**	114	**108.48**	114
	CMA-ES	**4.27**	130	**6.37**	130	**11.81**	130	**41.00**	130	**97.87**	130	**108.48**	130
	Greedy	**4.27**	130	**6.37**	130	**11.81**	129	**41.00**	130	**97.87**	130	**108.48**	130
	Grid	11.94	1440	17.85	1440	18.89	1440	114.80	1440	274.03	1440	303.73	1440
	ChatDSE	16.04	\	25.49	\	46.74	\	151.70	\	391.47	\	429.56	\
	LLM-DSE	**4.27**	108	**6.37**	104	**11.81**	102	**41.00**	111	98.63	109	108.94	115
	LLM-Box	**4.27**	119	**6.37**	118	**11.81**	116	50.84	117	109.61	120	260.34	77
	CriticNPU	**4.27**	34	**6.37**	31	**11.81**	54	**41.00**	55	**97.88**	54	**108.48**	56

Table 21: Config C9. Small models (8B, 13B): 8 chips, TP=8, PP=1, BS=16, 400W; Large models (70B, 405B, 671B): 64 chips, TP=16, PP=4, BS=16, 400W; DSv2-236B: 16 chips, TP=8, PP=2, BS=16, 400W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.1573	130	**0.2381**	130	0.1797	130	0.6795	130	**2.72**	130	**5.02**	130
	CMA-ES	0.2249	130	0.3692	130	0.2542	130	1.05	130	2.83	130	5.07	130
	Greedy	**0.1563**	130	**0.2381**	130	0.1866	130	0.6648	130	2.72	130	**5.02**	130
	Grid	0.4192	1440	0.6378	1440	0.2795	1440	1.59	1440	7.62	1440	14.05	1440
	ChatDSE	0.5935	\	0.8736	\	0.6745	\	2.22	\	10.88	\	20.08	\
	LLM-DSE	**0.1563**	109	**0.2381**	106	**0.1766**	110	1.56	97	2.73	111	**5.02**	91
	LLM-Box	0.1755	120	0.2397	115	**0.1766**	109	**0.6627**	114	3.27	118	5.02	55
	CriticNPU	0.1615	39	0.2460	48	0.1791	67	0.7031	64	2.72	68	**5.02**	66
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.27**	130	**6.37**	130	**11.81**	130	**41.00**	130	**97.87**	130	108.48	130
	CMA-ES	**4.27**	130	**6.37**	130	**11.81**	130	**41.00**	130	**97.87**	130	108.48	130
	Greedy	**4.27**	130	**6.37**	130	**11.81**	130	**41.00**	130	**97.87**	130	108.48	130
	Grid	11.94	1440	17.85	1440	18.89	1440	114.80	1440	274.03	1440	303.73	1440
	ChatDSE	16.63	\	24.22	\	43.91	\	160.72	\	391.47	\	433.90	\
	LLM-DSE	**4.27**	109	**6.37**	106	**11.81**	110	**41.00**	97	**97.87**	111	108.48	91
	LLM-Box	**4.27**	120	**6.37**	115	**11.81**	109	**41.00**	114	**97.87**	118	**91.13**	55
	CriticNPU	**4.27**	39	**6.37**	48	**11.81**	67	**41.00**	64	**97.87**	68	108.48	66

Table 22: Config C10. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=1, 300W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=1, 300W; DSv2-236B: 16 chips, TP=4, PP=4, BS=1, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	**0.0174**	114	0.0285	114	0.0727	114	0.3464	114	0.7359	114	**1.62**	114
	CMA-ES	0.0226	130	0.0349	130	0.0920	130	0.4083	130	0.7500	130	1.67	130
	Greedy	**0.0174**	130	**0.0270**	129	0.0762	130	**0.3344**	130	**0.7291**	130	**1.62**	130
	Grid	0.0241	1440	0.0373	1440	0.0988	1440	0.5671	1440	2.02	1440	2.57	1440
	ChatDSE	0.0558	\	0.0854	\	0.2178	\	0.7331	\	3.23	\	6.79	\
	LLM-DSE	**0.0174**	114	0.0685	110	0.0767	113	1.14	111	1.13	113	1.62	95
	LLM-Box	0.0182	118	0.0286	113	0.1198	115	0.3885	117	0.7338	114	2.58	30
	CriticNPU	**0.0174**	52	**0.0270**	54	**0.0719**	49	**0.3344**	51	**0.7291**	55	**1.62**	58
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**2.56**	114	**3.96**	114	**12.41**	114	**60.68**	114	**28.40**	114	**58.85**	114
	CMA-ES	**2.56**	130	**3.96**	130	**12.41**	130	**60.68**	130	28.49	130	**58.85**	130
	Greedy	**2.56**	130	**3.96**	129	**12.41**	130	**60.68**	130	**28.40**	130	**58.85**	130
	Grid	4.09	1440	6.34	1440	19.86	1440	169.90	1440	79.52	1440	94.16	1440
	ChatDSE	10.08	\	15.84	\	48.41	\	216.01	\	113.60	\	218.91	\
	LLM-DSE	**2.56**	114	**3.96**	110	**12.41**	113	**60.68**	111	**28.40**	113	58.89	95
	LLM-Box	**2.56**	118	**3.96**	113	**12.41**	115	**60.68**	117	**28.40**	114	94.16	30
	CriticNPU	**2.56**	52	**3.96**	54	**12.41**	49	**60.68**	51	**28.40**	55	58.87	58

Table 23: Config C11. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=32, 300W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=32, 300W; DSv2-236B: 16 chips, TP=4, PP=4, BS=32, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.2725	114	**0.4223**	114	0.2849	114	1.34	114	**5.80**	114	**6.44**	114
	CMA-ES	0.3461	130	0.4558	130	0.4207	130	1.74	130	5.91	130	6.58	130
	Greedy	**0.2697**	130	**0.4223**	130	**0.2786**	130	**1.33**	129	**5.80**	130	**6.44**	130
	Grid	0.5560	1440	0.8335	1440	0.3806	1440	1.88	1440	15.91	1440	10.21	1440
	ChatDSE	0.8033	\	1.27	\	0.8352	\	10.20	\	22.56	\	23.51	\
	LLM-DSE	**0.2697**	102	1.24	107	0.7786	109	**1.33**	118	**5.80**	95	**6.44**	84
	LLM-Box	0.2856	117	0.5432	113	0.3445	116	1.54	116	**5.80**	117	11.53	25
	CriticNPU	**0.2697**	55	0.4381	53	**0.2786**	53	**1.33**	51	**5.80**	51	6.47	55
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**6.78**	114	**10.55**	114	**14.52**	114	**67.45**	114	**110.16**	114	**123.28**	114
	CMA-ES	**6.78**	130	**10.55**	130	**14.52**	130	**67.45**	130	110.70	130	130.64	130
	Greedy	**6.78**	130	**10.55**	130	**14.52**	130	**67.45**	129	**110.16**	130	**123.28**	130
	Grid	18.97	1440	29.54	1440	23.24	1440	188.84	1440	308.44	1440	197.25	1440
	ChatDSE	26.43	\	42.19	\	56.35	\	269.78	\	431.82	\	453.66	\
	LLM-DSE	**6.78**	102	**10.55**	107	**14.52**	109	76.89	118	**110.16**	95	123.74	84
	LLM-Box	**6.78**	117	**10.55**	113	**14.52**	116	**67.45**	116	145.41	117	221.90	25
	CriticNPU	**6.78**	55	**10.55**	53	**14.52**	53	**67.45**	51	**110.16**	51	123.30	55

Table 24: Config C12. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=48, 300W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=48, 300W; DSv2-236B: 16 chips, TP=4, PP=4, BS=48, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.4044	114	**0.6335**	114	**0.4164**	114	2.01	114	8.70	114	**9.65**	114
	CMA-ES	0.4669	130	0.6669	130	0.4170	130	2.78	130	8.96	130	9.86	130
	Greedy	**0.4042**	130	**0.6335**	130	**0.4164**	130	**2.00**	130	**8.70**	130	**9.65**	130
	Grid	0.8309	1440	1.25	1440	0.5695	1440	2.80	1440	23.86	1440	15.31	1440
	ChatDSE	1.15	\	4.55	\	1.28	\	3.77	\	32.04	\	41.88	\
	LLM-DSE	1.15	114	1.64	113	**0.4164**	109	**2.00**	117	13.50	99	9.67	41
	LLM-Box	0.5592	118	0.6527	115	0.5380	118	2.29	113	16.71	120	16.86	13
	CriticNPU	**0.4042**	58	**0.6335**	52	0.4375	54	**2.00**	54	**8.70**	51	**9.65**	54
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**9.03**	114	**14.07**	114	**15.93**	114	**71.96**	114	**156.89**	114	**166.25**	114
	CMA-ES	**9.03**	130	**14.07**	130	**15.93**	130	**71.96**	130	157.30	130	**166.25**	130
	Greedy	**9.03**	130	**14.07**	130	**15.93**	130	**71.96**	130	**156.89**	130	**166.25**	130
	Grid	25.28	1440	39.39	1440	25.49	1440	201.49	1440	439.30	1440	266.00	1440
	ChatDSE	34.85	\	56.27	\	63.73	\	270.57	\	589.91	\	645.04	\
	LLM-DSE	**9.03**	114	**14.07**	113	**15.93**	109	79.16	117	158.05	99	**166.25**	41
	LLM-Box	**9.03**	118	**14.07**	115	**15.93**	118	**71.96**	113	200.82	120	292.60	13
	CriticNPU	**9.03**	58	**14.07**	52	**15.93**	54	**71.96**	54	**156.89**	51	166.27	54

Table 25: Config C13. Small models (8B, 13B): 8 chips, TP=2, PP=4, BS=1, 300W; Large models (70B, 405B, 671B): 64 chips, TP=4, PP=16, BS=1, 300W; DSv2-236B: 16 chips, TP=2, PP=8, BS=1, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.0316	114	0.0498	114	**0.1236**	114	0.6229	114	**2.23**	114	**3.24**	114
	CMA-ES	0.0476	130	0.0587	130	0.1380	130	1.03	130	2.31	130	3.42	130
	Greedy	0.0325	130	**0.0494**	128	**0.1236**	130	0.6855	130	**2.23**	130	**3.24**	130
	Grid	0.0419	1440	0.0643	1440	0.1576	1440	0.8465	1440	6.11	1440	5.13	1440
	ChatDSE	0.0928	\	0.1358	\	0.3081	\	1.14	\	8.49	\	12.15	\
	LLM-DSE	**0.0314**	102	0.1340	110	0.3744	104	2.18	103	3.75	118	**3.24**	100
	LLM-Box	0.0339	118	0.0603	114	**0.1236**	116	0.6869	120	**2.23**	115	3.25	117
	CriticNPU	**0.0314**	51	**0.0494**	52	**0.1236**	51	**0.6208**	51	2.24	56	3.27	57
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.30**	114	**6.90**	114	**20.44**	114	**114.35**	114	**57.39**	114	**97.31**	114
	CMA-ES	**4.30**	130	**6.90**	130	**20.44**	130	**114.35**	130	**57.39**	130	97.41	130
	Greedy	**4.30**	130	**6.90**	128	**20.44**	130	**114.35**	130	**57.39**	130	**97.31**	130
	Grid	6.89	1440	11.05	1440	32.71	1440	320.18	1440	160.68	1440	155.70	1440
	ChatDSE	17.22	\	27.61	\	79.32	\	429.96	\	220.36	\	369.75	\
	LLM-DSE	**4.30**	102	**6.90**	110	**20.44**	104	**114.35**	103	**57.39**	118	**97.31**	100
	LLM-Box	**4.30**	118	**6.90**	114	**20.44**	116	128.07	120	**57.39**	115	**97.31**	117
	CriticNPU	**4.30**	51	**6.90**	52	**20.44**	51	**114.35**	51	**57.39**	56	**97.31**	57

Table 26: Config C14. Small models (8B, 13B): 8 chips, TP=8, PP=1, BS=1, 300W; Large models (70B, 405B, 671B): 64 chips, TP=16, PP=4, BS=1, 300W; DSv2-236B: 16 chips, TP=8, PP=2, BS=1, 300W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.0115	114	0.0178	114	0.0531	114	0.2085	114	**0.3450**	114	**1.26**	114
	CMA-ES	0.0137	130	0.0229	130	0.0665	130	0.2781	130	0.3549	130	1.28	130
	Greedy	**0.0113**	130	**0.0175**	130	**0.0528**	129	0.2180	130	**0.3450**	130	**1.26**	130
	Grid	0.0174	1440	0.0265	1440	0.0796	1440	0.4528	1440	0.9638	1440	3.53	1440
	ChatDSE	0.0658	\	0.0634	\	0.1936	\	0.6306	\	1.38	\	4.79	\
	LLM-DSE	0.0230	108	**0.0175**	105	**0.0528**	110	**0.2063**	106	0.3455	91	1.26	96
	LLM-Box	**0.0113**	113	0.0183	120	0.0552	120	0.2913	119	0.3596	116	1.44	119
	CriticNPU	0.0118	28	0.0183	42	**0.0528**	55	0.2174	48	**0.3450**	54	**1.26**	56
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**2.15**	114	**3.07**	114	**10.75**	114	**37.61**	114	**20.56**	114	**47.66**	114
	CMA-ES	**2.15**	130	**3.07**	130	**10.75**	130	**37.61**	130	**20.56**	130	**47.66**	130
	Greedy	**2.15**	130	**3.07**	130	**10.75**	129	**37.61**	130	**20.56**	130	**47.66**	130
	Grid	3.45	1440	4.92	1440	17.20	1440	105.31	1440	57.58	1440	133.44	1440
	ChatDSE	8.49	\	12.29	\	43.00	\	145.93	\	82.25	\	181.09	\
	LLM-DSE	**2.15**	108	**3.07**	105	**10.75**	110	**37.61**	106	**20.56**	91	**47.66**	96
	LLM-Box	**2.15**	113	**3.07**	120	**10.75**	120	**37.61**	119	34.14	116	53.43	119
	CriticNPU	**2.15**	28	**3.07**	42	**10.75**	55	**37.61**	48	**20.57**	54	**47.66**	56

Table 27: Config C15. Small models (8B, 13B): 8 chips, TP=4, PP=2, BS=16, 350W; Large models (70B, 405B, 671B): 64 chips, TP=8, PP=8, BS=16, 350W; DSv2-236B: 16 chips, TP=4, PP=4, BS=16, 350W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.1235	120	0.1927	120	**0.1257**	120	0.5696	120	2.88	120	3.20	120
	CMA-ES	0.1481	130	0.2494	130	0.1654	130	0.9266	130	2.90	130	3.33	130
	Greedy	**0.1225**	130	**0.1889**	130	0.1283	130	0.5737	130	**2.87**	130	3.20	130
	Grid	0.1681	1440	0.4179	1440	0.1700	1440	0.9808	1440	7.97	1440	5.11	1440
	ChatDSE	0.3966	\	0.5940	\	0.4010	\	1.73	\	11.42	\	12.16	\
	LLM-DSE	0.1235	111	**0.1889**	80	0.1283	112	**0.5450**	104	**2.87**	109	**3.20**	104
	LLM-Box	0.1256	114	0.1907	116	0.1283	119	0.6035	120	4.57	118	3.86	120
	CriticNPU	**0.1225**	56	**0.1889**	61	**0.1257**	62	**0.5450**	61	**2.87**	61	**3.20**	62
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**4.53**	120	**7.03**	120	**13.11**	120	**62.93**	120	**63.43**	120	**80.32**	120
	CMA-ES	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Greedy	**4.53**	130	**7.03**	130	**13.11**	130	**62.93**	130	**63.43**	130	**80.32**	130
	Grid	7.24	1440	19.69	1440	20.98	1440	176.21	1440	177.59	1440	128.50	1440
	ChatDSE	17.92	\	26.16	\	51.40	\	245.43	\	253.70	\	305.18	\
	LLM-DSE	**4.53**	111	**7.03**	80	**13.11**	112	**62.93**	104	**63.43**	109	**80.32**	104
	LLM-Box	**4.53**	114	**7.03**	116	**13.11**	119	**62.93**	120	**63.43**	118	93.18	120
	CriticNPU	**4.53**	56	**7.03**	61	**13.11**	62	**62.93**	61	**63.43**	61	**80.32**	62

Table 28: Config C16. Small models (8B, 13B): 8 chips, TP=2, PP=4, BS=16, 350W; Large models (70B, 405B, 671B): 64 chips, TP=4, PP=16, BS=16, 350W; DSv2-236B: 16 chips, TP=2, PP=8, BS=16, 350W.

	Method	Llama3-8B		Llama2-13B		Llama3-70B		Llama3.1-405B		DSv2-236B		DSv3-671B	
		TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims	TTFT	Sims
Prefill	BO	0.1104	120	**0.1699**	120	**0.1038**	120	0.5001	120	4.41	120	3.23	120
	CMA-ES	0.1904	130	0.2350	130	0.1689	130	1.04	130	4.50	130	3.42	130
	Greedy	**0.1085**	130	0.1737	130	0.1072	130	0.5267	130	**4.38**	130	3.22	130
	Grid	0.1424	1440	0.2165	1440	0.1340	1440	0.8465	1440	12.10	1440	5.10	1440
	ChatDSE	0.3097	\	1.67	\	0.3027	\	1.20	\	17.79	\	12.75	\
	LLM-DSE	0.2615	111	**0.1699**	103	**0.1038**	115	**0.4980**	113	**4.38**	108	4.42	99
	LLM-Box	0.1108	106	0.2131	118	**0.1038**	120	0.6362	113	9.34	114	4.42	109
	CriticNPU	**0.1085**	56	**0.1699**	58	**0.1038**	54	**0.4980**	60	**4.38**	62	**3.21**	60
	Method	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims	TPOT	Sims
Decode	BO	**5.99**	120	**9.54**	120	**20.44**	120	**114.35**	120	**71.21**	120	**97.31**	120
	CMA-ES	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	71.38	130	**97.31**	130
	Greedy	**5.99**	130	**9.54**	130	**20.44**	130	**114.35**	130	**71.21**	130	**97.31**	130
	Grid	9.58	1440	15.26	1440	32.71	1440	320.18	1440	199.39	1440	155.70	1440
	ChatDSE	22.76	\	37.77	\	77.68	\	457.40	\	284.83	\	389.21	\
	LLM-DSE	**5.99**	111	**9.54**	103	**20.44**	115	**114.35**	113	**71.21**	108	**97.31**	99
	LLM-Box	**5.99**	106	**9.54**	118	**20.44**	120	**114.35**	113	**71.21**	114	128.45	109
	CriticNPU	**5.99**	56	**9.54**	58	**20.44**	54	**114.35**	60	**71.21**	62	**97.31**	60

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Introduction state the central contributions: an LLM-driven NPU design framework, an online-calibrated roofline model for pre-simulation screening and post-simulation bottleneck diagnosis, and simulator-efficient NPU design. The claims are supported by the Method section and the experimental results in Section 4, while the paper narrows the scope through the limitations in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 includes an explicit limitations paragraph. It states that evaluation is entirely simulation-based, that no physical implementation or tape-out validation is provided, and that results depend on the fidelity of the NeuSIM simulator.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper presents analytical modeling equations and algorithmic design rules, but it does not state formal theoretical results such as theorems or lemmas requiring proofs. The relevant assumptions and equations for the roofline model are described in Sections 2–3 and Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 3–4 and the appendix describe the algorithm, roofline equations, prompts, workloads, baselines, constraints, deployment settings, metrics, and per-deployment results. The paper also states that code is available, which provides an additional route for reproducing the main experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [N/A]

   Justification: We're still wrapping up the code repository

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 specifies the six LLM workloads, simulator, baselines, 16 constraint settings, disaggregated prefill/decode evaluation, and metrics. The appendix provides prompts, parameter descriptions, hyperparameter sensitivity, convergence trajectories, backbone generalization, and per-deployment results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: All experiments can demonstrate our effectiveness.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The paper reports simulator-call counts extensively, but it does not specify the physical compute workers, memory, wall-clock runtime, or total compute budget needed to reproduce the experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is a simulation-based study of NPU architecture design and does not involve human subjects, private data, or deployed decision-making systems. To the best of our knowledge, the research process and reporting are consistent with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The current manuscript does not include a dedicated broader-impact discussion. Such a discussion could note positive impacts such as more efficient LLM inference hardware, and possible negative impacts such as increased deployment of compute-intensive AI systems or reliance on simulator-derived hardware decisions.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk pretrained models, image generators, scraped datasets, or other assets with a direct misuse profile requiring controlled release. The released artifact is code for NPU architecture design-space exploration.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [No]

Justification: The paper cites prior work and existing assets such as LLM workloads and NeuSIM, but it does not explicitly state the licenses or terms of use for the code, models, or simulator assets used. The manuscript should add license and terms-of-use information where applicable.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: No new assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or research with human subjects. Therefore, participant instructions, screenshots, and compensation details are not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subjects research. Accordingly, IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core methodological component of the work: the Compute Demand Agent extracts operator-level workload profiles and the Architecture Design Agent proposes configurations. Their roles are described in Sections 3.1 and 3.3, with prompt templates provided in Appendix D.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.