RECURSIVE EXPERIENCE IMPROVES NPU MICROARCHITECTURE DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

NPU microarchitecture design space exploration seeks low latency configurations that satisfy power and memory constraints. Recent LLM design agents can learn useful hardware choices through simulator feedback, but this experience usually remains within the current workload. When another LLM workload arrives, the agent may spend additional simulator calls rediscovering similar choices. Reusing past experience could reduce this effort, yet a choice that helps one model may not help a larger model or remain feasible under a tighter power budget. We propose **RecurNPU**, a method for recursively accumulating and reusing design experience across LLM workloads. RecurNPU summarizes what each search learned about hardware choices in an experience card, retaining the workload and constraints behind those observations. It then selects relevant cards and adapts their guidance to the next workload. This gives the Designer a starting point based on earlier results, while measurements on the current workload help it revise choices that no longer work. Each completed source search adds new experience for later searches, without retraining the language model. We evaluate RecurNPU on 96 settings spanning changes in LLM family, model scale, and power budget. Across the 84 shifted settings, RecurNPU reduces negative transfer from 44.0% for an agent carrying notes from preceding searches (Raw-Mem) to 20.2%. Compared with Raw-Mem, time to first token improves by 5.4% at the median and 23.4% in the geometric mean. After one target evaluation, RecurNPU is within 1% of or faster than Raw-Mem's final latency in 51 of these settings.

1 INTRODUCTION

Neural Processing Units (NPUs) (Jouppi et al., 2017) are widely used to accelerate Large Language Model (LLM) inference (Jouppi et al., 2017; Yin et al., 2026). Their performance depends strongly on hardware choices such as compute units, on-chip memory, and memory bandwidth. NPU microarchitecture design space exploration (DSE) searches these choices to find a configuration that provides low latency while satisfying hardware constraints. Recent LLM agents have shown promising results in automating hardware DSE (Wang et al., 2025; Li et al., 2025; Zhang et al., 2026c; Xiong et al., 2026). These agents propose hardware configurations or optimization choices, evaluate them with their task-specific simulator, and improve later proposals from the feedback. During this process, they learn which hardware choices help performance and which choices lead to constraint violations. However, this learning usually remains within the current workload. When a new LLM workload arrives, the agent has limited access to what it learned from earlier searches and may spend additional simulator calls rediscovering similar design lessons. Human architects work differently: they gradually accumulate experience across projects and use it to guide later decisions. *This difference motivates us to study how an NPU DSE agent can recursively accumulate and reuse experience across LLM workloads without post-training the LLM agent.*

Cross-workload transfer itself has been studied in DSE outside the NPU domain. Prior work (Wang et al., 2023; Yu et al., 2023; Xue et al., 2025) reuses information from earlier workloads through transfer learning, workload-aware surrogate models, and meta-learning. Recent LLM-based hardware DSE (Wang et al., 2025; Zhang et al., 2026c; Xiong et al., 2026; Sharma et al., 2026; Zhang et al., 2026a) creates another opportunity: an agent can carry search observations and design experience from previous simulator interactions into later workloads. Recent memory methods (ReME (Cao et al., 2026), SimpleMem (Liu et al., 2026), ReasoningBank (Ouyang et al., 2026),

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

and ACE (Zhang et al., 2026b)) organize this experience for later tasks, and benchmarks evaluate its reuse over task sequences (Wei et al., 2025; He et al., 2026). Whether such accumulated experience continues to help NPU design across changing LLM workloads and hardware constraints, however, remains less clear. A practical obstacle is the lack of a common NPU DSE benchmark for sequential source and target workloads. To study this question, our benchmark covers three shift factors, model family, model scale, and hardware constraint, as well as their combinations. It spans dense Llama and MoE DeepSeek workloads over an NPU design space with nine hardware parameters and 90.3 million configurations, producing 96 transfer settings across different source-history lengths. In each setting, the same agent searches the sources in order, uses earlier experience and adds new observations after each search, and then uses the accumulated experience to search the target.

A natural question is whether this problem is already handled well by existing methods. First, can traditional transfer-DSE methods (Feurer et al., 2018; Wistuba et al., 2016) reuse information from previous workloads? Second, given the recent strength of LLM-based DSE, is it enough for an LLM design agent to directly carry its accumulated experience into later searches? We study both questions in Section 2, comparing traditional methods with an LLM agent that searches without source memory (No-Mem) or carries notes accumulated from its preceding source searches (Raw-Mem). Our results show that experience accumulated across LLM workloads can provide substantial value, and that an LLM agent can reuse such experience beyond what traditional transfer-DSE methods achieve in our study. **However, this experience is not universally helpful: Raw-Mem exceeds the No-Mem latency reference on 44.0% of the shifted settings.** Experience learned under one workload and hardware condition can become misleading when those conditions change, causing negative transfer even when the same agent and search process are used. This reveals a challenge in cross-workload NPU DSE: the goal is not simply to give an LLM agent more memory, but to help it reuse experience without blindly carrying source-specific assumptions into a new workload.



How can an NPU design agent retain useful experience from previous workloads while avoiding experience that becomes misleading on a new workload?

To address this challenge, we propose **RecurNPU**, a method that helps an NPU design agent accumulate experience across LLM workloads and decide how to use it in later searches. A hardware choice that helps one workload may need to change when the next model requires more memory or the power budget becomes tighter. We therefore keep earlier search results together with the workload and hardware constraints under which they were obtained. After each source search, RecurNPU records this experience in a card, including comparisons that show how hardware changes affected latency. When a new workload arrives, it selects cards according to workload similarity and the relative latency improvements achieved during source search. A short guidance note explains which earlier choices may need adjustment for the new model and resource requirements. The Designer uses the cards and guidance to choose designs to try, then revises its choices using simulator results on the current workload. Earlier experience can therefore give the search a useful starting point, while new measurements help it reconsider advice that no longer applies. Each source search uses the existing experience and adds its own findings, so the process is recursive: what one search learns becomes available to later searches without retraining the language model. Our contributions are:

- **We build a benchmark to study how NPU design experience transfers across LLM workloads.** Its 96 settings cover changes in model family, model scale, and power budget, over 90.3 million NPU design space. Each setting includes one to six source workloads: the agent uses and adds experience as it searches them in order, then uses this experience for the target.
- **We show that earlier experience can help NPU design but can also lead to negative transfer when workloads change.** An agent carrying notes from previous searches (Raw-Mem) reaches the lowest latency group more often than memoryless search and the traditional transfer methods we test. Yet it exceeds the No-Mem latency reference on 44.0% of shifted settings, showing that experience useful in one search can become misleading in the next.
- **We introduce RecurNPU to accumulate and reuse NPU design experience across changing LLM workloads.** It records hardware observations together with their workloads and constraints, selects and adapts relevant experience for the next workload, and uses current simulator results to guide the search. Across the 84 shifted settings, RecurNPU reduces negative transfer from Raw-Mem's 44.0% to 20.2% and improves TTFT by 5.4% at the median and 23.4% in the geometric mean, without retraining the language model.

Table 1: Number of wins over 96 source–target settings; N gives the number of settings in each row. A method receives one win if it achieves the lowest target latency, with results within 1% treated as ties. No-Mem denotes LLM-based NPU DSE without memory, while Raw-Mem accumulates notes from preceding source searches.

		Classical target search					Classical transfer				LLM-based search	
Shift type	N	Random	Grid	Greedy	CMA-ES	BO	T-BO	RGPE	TST-R	NaiveApply	No-Mem	**Raw-Mem**
Matched control												
Matched	12	0	0	0	0	0	4	1	0	2	0	**8**
Single-factor shifts												
Model family	12	0	0	0	0	2	1	1	0	0	0	**8**
Model scale	12	0	0	0	0	0	6	0	1	4	3	**7**
HW constraint	12	0	0	0	0	0	4	2	3	3	0	**5**
Two-factor shifts												
Family + Scale	12	0	0	0	0	0	2	0	0	1	4	**6**
Family + Constr.	12	0	0	0	0	0	5	1	0	0	1	**6**
Scale + Constr.	12	0	0	0	0	0	3	2	0	3	**8**	4
Three-factor shift												
All three	12	0	0	0	0	0	1	3	2	0	**8**	3
Total	**96**	0	0	0	0	2	26	10	6	13	24	**47**

2 Problem Setup and Empirical Motivation

Problem Definition of Recursive NPU DSE across LLM Workloads. For a target LLM workload w_t, we use an experience bank $\mathcal{B}_n$ from n preceding source searches to find a low latency NPU configuration satisfying hardware constraints. The workload w_t specifies the model, inference phase, batch size, and deployment. A configuration $\theta \in \Theta$ specifies compute units, clock frequency, on-chip memory, and bandwidth; Θ is the allowed discrete NPU design space. The search returns

$$\theta_t = \mathrm{Search}(w_t, \mathcal{B}_n; \mathcal{C}),$$

where Search uses the workload and experience bank to find a target configuration θ_t satisfying the feasibility condition $\mathcal{C}$. The search aims to solve

$$\min_{\theta \in \Theta} \ \mathcal{L}(\theta, w_t) \quad \text{s.t.} \quad \mathcal{C}(\theta). \tag{1}$$

Here, $\mathcal{L}(\theta, w_t)$ is latency measured by the simulator, and $\mathcal{C}(\theta)$ requires analytic peak power to stay within the target's power budget and HBM capacity per chip to meet its memory requirement. The bank guides which configurations to evaluate; the simulator measures their latency on w_t. With a fixed evaluation budget, θ_t is the lowest latency feasible design evaluated. The search reports failure if none is feasible. Each setting constructs $\mathcal{B}_n$ from an empty bank $\mathcal{B}_0 = \emptyset$ by searching source workloads $(w_1, \ldots, w_n)$ in order. For $k = 1, \ldots, n$, search on w_k uses $\mathcal{B}_{k-1}$ and adds experience from the completed search to obtain $\mathcal{B}_k$. The target uses $\mathcal{B}_n$ without updating it. Appendix A summarizes the main notation used throughout the paper.

Benchmark Design for Recursive Transfer. We build the benchmark around a practical question: when does experience from earlier workloads help design an NPU for the next one? We consider changes in model family, model scale, and power requirements, individually and in combination. For each target configuration, we vary the source family and the amount of accumulated experience, using one to six preceding workloads. The resulting 96 settings cover seven shift classes and matched conditions, with twelve settings in each group. Comparing these conditions on the same target helps us understand how the relevance and amount of prior experience affect the resulting design. The workloads span dense Llama and MoE DeepSeek models, smaller and larger scales, and power budgets of 400 W and 300 W. Each setting searches a space of nine hardware parameters with $90,316,800$ configurations; Appendix C gives the workload specifications and allowed values. We use at most 30 simulator evaluations per source and 12 per target, stopping early only when proposal retries are exhausted. Source searches receive no target evaluations, and No-Mem provides the reference for each target's experience conditions. We retain infeasible outcomes because experience can also prevent a search from finding a feasible design.

Empirical Observations. We use the benchmark to answer two questions about recursive experience transfer. First, does experience produced by preceding NPU searches provide value beyond conventional transfer of source evaluations or designs? Second, is simply carrying that recursively accumulated experience forward sufficient when the workload or hardware conditions change? ***1) Recursively accumulated LLM experience provides a larger gain than classical transfer.*** Table 1 shows that Transfer-BO reaches the lowest-latency group in 26 of the 96 settings. The memoryless agent reaches that group in 24 settings without source experience, while Raw-Mem reaches it in 47 by carrying the experience accumulated during its previous source searches. This suggests that the main opportunity is not only to transfer source designs, but to reuse the design experience produced along the preceding workload-search chain. ***2) Raw recursive experience is powerful but can propagate negative transfer.*** Table 2 compares Raw-Mem with the frozen No-Mem reference under the same target-search budget. Since No-Mem uses no source, we use the median of its twelve repetitions for each target as the reference. Raw-Mem incurs negative transfer on 37 of the 84 shifted settings (44.0%), with harm appearing across all shift types and becoming most frequent for scale-related and multi-factor shifts. Because later source searches are themselves conditioned on earlier notes, a source-specific assumption can remain in the experience chain after the workload or hardware condition that supported it has changed. To use this experience on a new workload, the Designer needs to know where an observation came from and use current measurements to decide whether it still helps. These findings motivate retaining the conditions behind each observation and adapting its use for the next search.

Table 2: Negative transfer of Raw-Mem across source–target shift classes.

Source–target relation	*Hurt*
Matched *(control)*	25.0
Model family	**33.3**
Model scale	**33.3**
HW constraint	25.0
Family+Scale	**50.0**
Family+Constr.	25.0
Scale+Constr.	**66.7**
All three	**75.0**

3 METHOD

RecurNPU uses experience from completed NPU searches to guide design for the next LLM workload (Figure 1). Since a hardware choice that helps one workload may be less useful for another, we keep both the measured outcome and the conditions behind it. After each source search, we **accumulate experience in cards that record hardware comparisons and their workload conditions**. Before the next search, we **select relevant cards for reuse on the new workload**. A short guidance note relates their observations to the current workload and constraints. The Designer then **uses the selected experience to propose NPU configurations**, with invalid or repeated proposals rejected before simulation. Simulator feedback guides later proposals within that search, and measurements on the current workload take priority when they conflict with earlier guidance. Each setting starts with an empty bank and searches its sources in order, so every source can use earlier cards and then contribute its own. The final target uses the accumulated bank without adding to it. The language model remains fixed throughout; code constructs and selects cards and checks proposals. In the following sections, k indexes completed source searches and t denotes the workload currently being searched, whether a source or the final target.

3.1 ACCUMULATING DESIGN EXPERIENCE

After source search k, we construct an experience card u_k from the completed source trajectory τ_k, which records the configurations evaluated on w_k and their simulator results. **The card keeps comparisons that change one hardware parameter while holding the others fixed, together with the workload and constraints.** A lower latency after increasing both compute and bandwidth does not tell a later search which change helped. Comparing feasible configurations that differ in only one parameter gives a more specific observation to reuse. We collect these pairs from the existing history, without scheduling extra evaluations.

The parameter comparison summary $\mathcal{E}_k$ records, for each parameter, how often the higher or lower value performs better and the median ratio of latency at the higher value to latency at the lower value. If the search contains no such pair, the card makes no claim about that parameter's effect. When different comparisons favor different directions, their counts remain in the card alongside the median

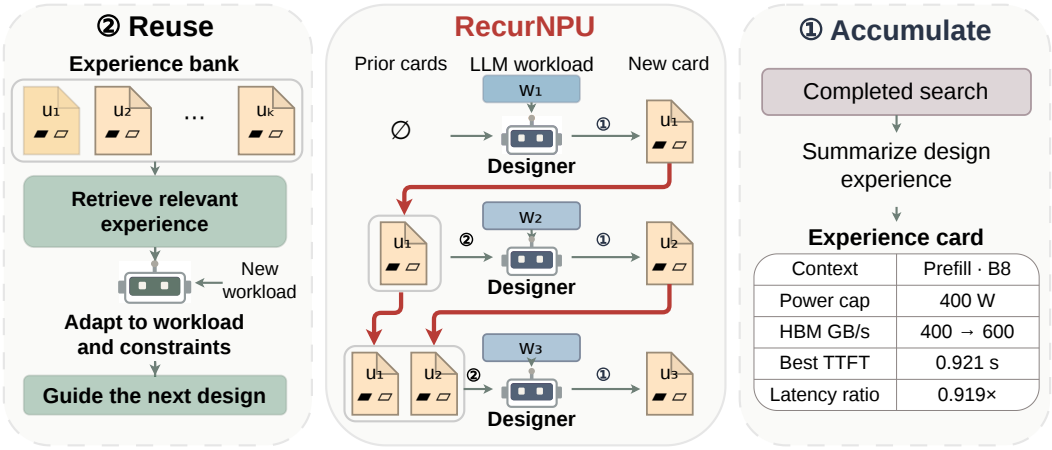


Figure 1: RecurNPU accumulates experience cards from completed source searches, selects and adapts them for the next LLM workload, and uses the resulting guidance to search for an NPU design. Each source contributes a new card after its search, while the final target uses the accumulated bank without updating it.

ratio. A later search can therefore see whether a trend was consistent across the measured configurations or varied with the other hardware choices. These comparisons describe the tested parameter values with the other hardware choices held fixed. They do not establish a general rule for all NPU configurations or workloads, where interactions among compute, memory, and communication may change the outcome. Each card u_k contains three types of information: source conditions, search outcomes, and parameter comparisons:

$$u_k = \Big(\underbrace{c_k, g_k}_{\text{workload and constraints}} , \underbrace{\mathcal{Q}_k, \theta_k^*, \mathcal{L}_k^*, \rho_k}_{\text{search results}}, \underbrace{\mathcal{E}_k}_{\text{parameter comparisons}} \Big). \tag{2}$$

Here, the source context c_k describes the model family, scale, inference phase, batch size, deployment, and measured resource bottleneck; the constraint vector g_k gives the power budget and required memory per chip. The tested parameter value sets $\mathcal{Q}_k$ record what was explored, alongside the best feasible configuration θ_k^* and best feasible latency $\mathcal{L}_k^*$. We define the relative latency ratio $\rho_k = \mathcal{L}_k^* / \mathcal{L}_k^{\text{first feasible}}$, where $\mathcal{L}_k^{\text{first feasible}}$ is the first feasible latency in that source search. Smaller ratios indicate a larger latency reduction; a ratio of one means no reduction. The best feasible design, its latency, and ρ_k are defined only when the search finds a feasible design; otherwise, the card is excluded from retrieval. All of these quantities come from existing results, so building a card needs no additional LLM or simulator call. We add u_k to the bank as $\mathcal{B}_k = \mathcal{B}_{k-1} \cup \{u_k\}$. The next source can draw on this bank, run its own search, and contribute a new card in turn. Appendix D gives the card construction details.

3.2 Selecting and Adapting Experience

For a new LLM workload, we need to choose which earlier searches to consult. A similar source can offer a useful starting point, while a source that made greater progress may suggest other hardware choices worth trying. We therefore **retrieve the card from the most similar workload and the card with the largest relative latency improvement**. These cards provide the measured observations for reuse; the Adapter adds a short note to help the Designer relate them to the new workload.

We consider only cards from source searches that found a feasible design. To find the most similar workload, we compare model family first, then inference phase, batch size, model scale, and deployment, using later attributes to break earlier ties. For the second card, we choose the smallest relative latency ratio ρ_k from Section 3.1, considering only searches that reduced latency by more than 1%. Because this ratio compares the best and first feasible latencies within the same search, a smaller model does not get preference simply because it runs faster. The retrieved card set R_t contains at

most two cards. A card selected by both rules is kept once, and a choice with no eligible card is omitted. Both choices use recorded workload metadata and source results, without another model call or simulator evaluation. Keeping at most two cards limits what the Designer reads at each step, while the bank remains available for later workloads. We apply these rules before each source or target search using the bank available at that point; Appendix D gives the remaining tie rules.

The Adapter helps interpret the selected cards without replacing their measurements. For example, a configuration that worked well for a smaller model may not have enough memory for the next model. It reads R_t, workload w_t, and the constraint vector $g_t = (\bar{P}_t, \bar{M}_t)$, where $\bar{P}_t$ is the power budget and $\bar{M}_t$ the required memory per chip. These requirements specify $\mathcal{C}$. The Adapter then writes a short guidance hint h_t:

$$h_t = \mathrm{Adapter}(R_t, w_t, g_t). \tag{3}$$

The note points out relevant workload differences and known constraint conflicts, with a reason for each suggested adjustment. A memory shortfall justifies a warning; a different model family alone does not establish that a measured bandwidth trend has reversed. The Designer receives h_t alongside the original cards and can refer directly to their measurements. The note supplements the cards: the ablations in Section 4.2 find only small latency changes when it is removed while the cards remain available. If no eligible card is available, the note is empty and no Adapter call is needed.

3.3 Designing for a New LLM Workload

For a new LLM workload, earlier experience can suggest which NPU designs to try. Each proposal still needs to satisfy the current power and memory requirements, and we need to measure its latency on that workload. We therefore *reject invalid or repeated proposals before simulation* and **use the current workload's measurements to override conflicting source guidance**.

The Designer reads w_t, the full model and execution configuration including inference phase, batch size, and deployment, together with the workload profile ϕ_t prepared by the Profiler. The profile summarizes facts such as dense or MoE structure, model scale, and memory demand; it adds no simulator measurements and leaves properties that cannot be determined from w_t unknown. A proposal can be rejected before simulation, so the evaluation index j counts simulator calls and the proposal attempt index r counts attempts for evaluation j. The evaluation history $\tau_t^{<j}$ contains the first $j-1$ measured configurations and their results. With this history, retrieved card set R_t, guidance hint h_t, and design space Θ, the Designer proposes a candidate configuration $\theta_t^{j,r}$:

$$\theta_t^{j,r} \sim \pi_{\mathrm{Designer}}\Big(\cdot \mid w_t, \phi_t, R_t, h_t, \Theta, g_t, \tau_t^{<j}, v_t^{j,r-1}\Big). \tag{4}$$

Here, π_{Designer} is the model's proposal distribution under the Designer prompt, and the validation feedback $v_t^{j,r-1}$ records the preceding rejection for the same evaluation. The first evaluation has an empty history, $\tau_t^{<1} = []$; each evaluation starts at $r = 1$ with $v_t^{j,0} = \emptyset$. Each proposal also states which workload fact or observation motivates the design and what the next measurement is intended to test. The source cards guide which designs to try; they do not narrow Θ to values seen before. A source design uses the current workload's evaluation budget whenever the Designer proposes it and it passes validation. No source latency is counted as a target measurement.

Before calling the simulator, the Harness checks allowed parameter values, rejects configurations already measured on this workload, and verifies $P_{\mathrm{peak}}(\theta) \le \bar{P}_t$ and $M_{\mathrm{HBM}}(\theta) \ge \bar{M}_t$. Here, P_{peak} is analytic peak power and M_{HBM} is HBM capacity per chip. These checks use the workload requirements and hardware parameters to screen feasibility before spending a simulator evaluation. A design that passes still needs a latency measurement, since meeting power and capacity requirements does not establish good performance. A rejection returns its reason and any power excess or memory shortfall as $v_t^{j,r}$, together with available power reductions when relevant. The Designer tries again at $r + 1$, while j and $\tau_t^{<j}$ stay unchanged. Once a proposal passes, it becomes the accepted configuration θ_t^j and is evaluated:

$$\tau_t^{<j+1} = \tau_t^{<j} \parallel \big[(\theta_t^j, \mathrm{Sim}(w_t, \theta_t^j))\big]. \tag{5}$$

Here, $\parallel$ denotes sequence concatenation, and the simulator evaluation operator Sim returns latency and resource feedback. The search then advances to evaluation $j + 1$ and resets the attempt counter

to $r = 1$. The next proposal can use this new measurement to revise a hardware choice suggested by earlier experience. Search returns the best feasible design found when the evaluation budget is exhausted or all ten attempts for the next evaluation fail. For a completed source search k, this history is the trajectory τ_k used to build its experience card in Section 3.1. Appendix D gives the validation details, and Appendix E provides the role prompts.

4 EXPERIMENTS

Experimental Settings. We compare RecurNPU, Raw-Mem, and NPU adaptations of ReME (Cao et al., 2026) and SimpleMem (Liu et al., 2026) on all 96 settings. All four use `gpt-5.4-mini` model family, start each setting with empty experience, and search its sources in order, allowing up to 30 simulator evaluations per source and 12 per target. Raw-Mem carries source notes forward, ReME extracts and revises lessons, SimpleMem compresses and retrieves observations, and RecurNPU records hardware comparisons with their workload conditions and selects cards for reuse. ReME, SimpleMem, and RecurNPU use the same Designer and feedback; Raw-Mem uses its original prompt, with separate comparisons using the same Designer. We report the best feasible target time to first token (TTFT). Negative transfer means finding no feasible design or exceeding the median of twelve No-Mem runs for that target by more than 1%. The main analysis covers 84 shifted settings alongside twelve matched controls, and we also check the results after excluding four development settings. Each condition has one execution in the full comparison, supplemented by repeated searches below. Appendices C, E, F, and G give the protocol, prompts, complete latencies, and further analyses.

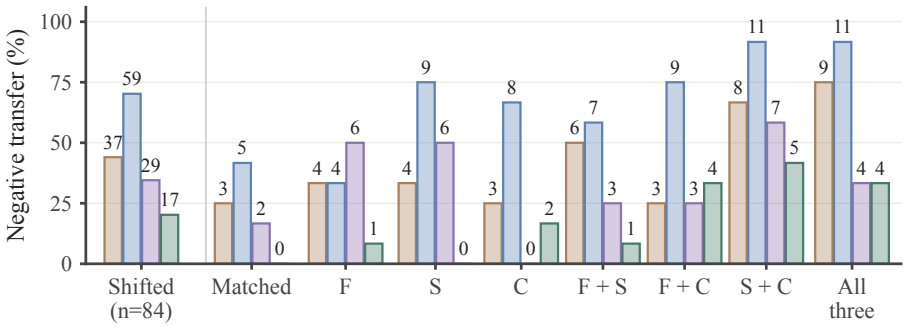


Figure 2: RecurNPU incurs negative transfer in 20.2% of shifted settings, compared with 44.0% for Raw-Mem, 70.2% for ReME, and 34.5% for SimpleMem. F, S, and C denote shifts in family, scale, and power budget.

4.1 OVERALL RESULTS

RecurNPU finds lower latency designs with fewer cases of negative transfer. Across the 84 shifted settings in Figure 2, negative transfer occurs in 17 cases for RecurNPU (20.2%), compared with 37 for Raw-Mem (44.0%), 59 for ReME (70.2%), and 29 for SimpleMem (34.5%). All four methods find feasible designs in all 96 settings, so these cases reflect latency differences. RecurNPU improves TTFT over Raw-Mem by 5.4% at the median and 23.4% in the geometric mean of setting-wise comparisons; the corresponding reductions against SimpleMem are 1.2% and 15.5%. The geometric mean reflects larger gains on some workloads alongside smaller changes elsewhere. ReME, SimpleMem, and RecurNPU use the same Designer and feedback, allowing us to compare how they store and reuse experience.

The reduction in negative transfer extends across most shift classes. RecurNPU has fewer cases than Raw-Mem in seven of the eight benchmark groups and fewer than SimpleMem in five, with one tie. Excluding the four development settings leaves 17 cases for RecurNPU and 34 for Raw-Mem among 80 shifted settings. With the same Designer, the earlier comparison on sixteen settings yields one case for RecurNPU, three for Raw-Mem, and eleven for ReME against the original No-Mem reference (Appendix G.1). Combined shifts account for 14 of RecurNPU's 17 cases: under Family + Constraint, it has four cases versus three each for Raw-Mem and SimpleMem; under Constraint alone, SimpleMem has none and RecurNPU has two.

4.2 Detailed Analysis

A. Target Latency and Search Efficiency. Lower negative transfer tells us how often experience avoids hurting a design. We also ask how much latency improves and how soon search finds a useful configuration, following all 84 shifted settings in Figure 3. Final geometric mean TTFT, normalized by each target's No-Mem median, is $0.890\times$ for RecurNPU, $1.161\times$ for Raw-Mem, and $1.644\times$ for ReME, corresponding to reductions of 23.4% and 45.9% against the two baselines. The benefit appears early: RecurNPU reaches $1.107\times$ after one target evaluation and $0.989\times$ after five. Its first design is already within 1% of or faster than Raw-Mem's final design in 51 of 84 settings. Earlier experience therefore often gives search a useful starting point, which later measurements refine for the current workload. This advantage concerns the target budget: the first proposal already benefits from the preceding source searches, whose cost is included in our later analysis. The curves carry the best latency forward when a search stops before the limit of twelve evaluations.

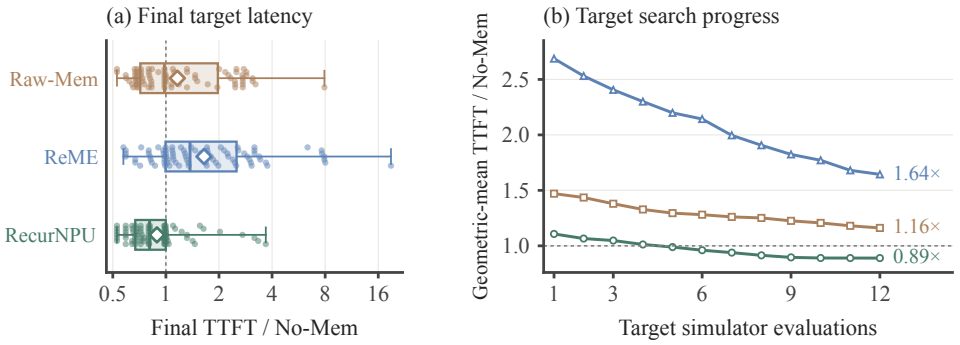


Figure 3: RecurNPU has lower geometric mean TTFT throughout target search, and its first evaluation is already below Raw-Mem's final value. Dots denote settings; boxes show medians and interquartile ranges, whiskers the full range, and diamonds geometric means.

B. Scaling with Source Experience. More source searches provide more observations, but those observations may come from workloads with different hardware needs. To see how this affects reuse, we independently rebuild histories of one to six sources for sixteen combinations of target and source family (Figure 4). RecurNPU has lower median TTFT with three through six sources and lower maximum TTFT with two through six. Raw-Mem has lower medians with one or two sources and a lower maximum with one. With six sources, the medians are similar at 31.1 seconds for RecurNPU and 31.4 for Raw-Mem, while the maxima are 131.7 and 623.0 seconds. At this history length, the clearest difference lies in the slowest designs found. A longer history does not guarantee a better result; the observations need to remain useful as workload conditions change.

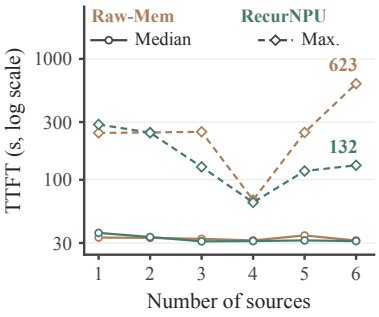


Figure 4: At six sources, RecurNPU has a lower maximum TTFT while the two methods have similar medians.

C. Component Ablations. We examine what experience should contain and how it should be reused on eight settings with four sources, covering all shift classes. The variants remove the Adapter's note, remove comparisons that isolate one hardware parameter, or select cards at random; each rebuilds its source history with the same Designer, feedback, and budgets. All find feasible designs, and their median TTFT ratios relative to full RecurNPU round to $1.000\times$ (Table 3). Yet removing comparisons or changing selection can more than double latency in individual settings. The pattern is clearest when scale and power change together: removing the parameter comparisons raises target TTFT to $2.82\times$, even though the median across all eight settings stays near one. Removing the Adapter changes TTFT by at most 4.9%, while the Designer still receives the cards and current workload. The larger changes thus concern what is recorded and selected, including how these choices affect experience acquired during source search.

Table 3: Removing single parameter comparisons or changing card selection produces the largest latency increases on this subset. Values are target TTFT / full RecurNPU; shaded entries exceed $2\times$.

	Matched	F	S	C	F + S	F + C	S + C	All three
Without Adapter	1.00	0.99	1.00	1.03	1.00	1.00	1.01	1.05
Without comparisons	1.00	0.98	1.00	0.99	1.00	1.04	**2.82**	**2.03**
Random retrieval	1.00	1.00	1.00	**2.62**	1.00	1.00	1.00	**2.67**

D. Repeated Searches. To check whether the gains persist across searches, we compare five executions per method on sixteen settings with the same Designer, feedback, and evaluation budgets. Each execution starts with empty experience and searches four sources in order before the target; five new No-Mem runs per target provide the reference. Across setting medians, RecurNPU lowers geometric mean TTFT by 15.3% relative to Raw-Mem, with negative transfer in 2/80 executions versus 17/80 (Table 4). The one infeasible RecurNPU search is included in this count. RecurNPU has a lower median TTFT in eleven settings, ties in three, and is slower in two. Restricting the comparison to the four newly executed sequences per setting gives a geometric mean ratio of $0.794\times$, so the advantage extends beyond the original execution. Within this subset, a bootstrap over target configurations and executions gives a 95% interval of $[2.5, 35.0]$ percentage points for the reduction in negative transfer. Appendix G.2 gives individual outcomes, tolerance checks, and bootstrap details.

Table 4: RecurNPU retains lower latency and fewer negative transfer outcomes across repeated searches. TTFT ratios are geometric means of setting medians relative to the median of five new No-Mem runs per target.

Method	TTFT / No-Mem ↓	Feasible runs ↑	Negative transfer (1%) ↓
Raw-Mem	$0.463\times$	**80/80**	17/80
RecurNPU	$\mathbf{0.393}\times$	79/80	**2/80**

E. Additional Analyses. In the earlier comparison on sixteen settings, the full source and target sequence costs a mean of \$0.581–\$0.693 for RecurNPU, \$0.553 for Raw-Mem, and \$0.582 for ReME per setting, including the acquisition of source experience (Appendix G.3). With LLMCompass and ONNXim, RecurNPU is faster than or tied with Raw-Mem in 11 of 16 and 15 of 16 settings, respectively, although the gains vary across the two simulators (Appendix G.4). On Qwen2.5-72B, a workload outside the primary benchmark, RecurNPU finds its best design on the first target evaluation, with 5.9% lower TTFT than Raw-Mem and a $2.07\times$ speedup over No-Mem (Appendix G.5).

These additional evaluations also clarify what transfers across the different tests. LLMCompass measures latency across decoder layers, while ONNXim evaluates a single layer using synthetic dense and gated feedforward workloads. We therefore compare methods within each simulator rather than combining their latency values with the main TTFT results. The many ties on ONNXim and the gains on a subset of LLMCompass settings indicate that useful experience need not improve every workload. The Qwen result complements these checks with a new model, using unchanged prompts and evaluation budgets.

5 Conclusion and Limitations

RecurNPU shows how experience from completed searches can help NPU design across changing LLM workloads, reducing negative transfer and finding lower latency designs than carrying source notes forward. The results suggest that useful experience includes both what a hardware choice achieved and the conditions under which it helped. Keeping these together gives later searches a basis for choosing what to try, while measurements on the new workload determine whether the choice remains useful. The ablations point more strongly to what is recorded and selected than to the Adapter's guidance note. The full benchmark uses one execution per condition, with four development settings; repeated searches cover sixteen settings from four target models. Ablations assess each component across both source and target searches. A good source design can still leave few comparisons for later workloads. This suggests a direction for future NPU search: evaluate promising configurations while also collecting hardware comparisons that may help the next workload.

AI USE STATEMENT

We used generative AI tools to assist with manuscript revision, literature search, and code development. The authors take responsibility for the accuracy and integrity of the final manuscript.

ETHICS STATEMENT

This study evaluates NPU designs using LLM model configurations and simulator measurements; it does not involve human participants or personal data. More efficient hardware could reduce the cost of LLM inference, although lower simulated latency alone does not establish lower energy use or environmental impact. Designs found through this search would still need validation on physical hardware before deployment.

REPRODUCIBILITY STATEMENT

Appendix C specifies the workload configurations, NPU design space, ordered source searches, evaluation budgets, baseline adaptations, and LLM settings. Appendices D and E describe how experience is recorded and reused, how proposals are checked, and the prompts used by each role. Appendix F reports the target latency for every setting. To assess variation in LLM search, Appendix G.2 reports repeated runs that rebuild the sequence of source and target searches with the same Designer, model version, and evaluation budgets.

REFERENCES

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026.

Kris Shengjun Dong, Sahil Modi, Dima Nikiforov, Sana Damani, Edward Lin, Siva Kumar Sastry Hari, and Christos Kozyrakis. KernelBlaster: Continual cross-task CUDA optimization via memory-augmented in-context reinforcement learning. *arXiv preprint arXiv:2602.14293*, 2026.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical transfer learning for Bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

Abraham Gonzalez, Raghav Gupta, Akanksha Jain, Hanna Alam, Alexander Novikov, Po-Sen Huang, Matej Balog, Marvin Eisenberger, Sergey Shirobokov, Ngân Vũ, et al. Archagent v2: A case study with the data prefetching championship. *arXiv preprint arXiv:2608.09874*, 2026.

Sina Hajimiri, Masih Aminbeidokhti, Jose Dolz, Ismail Ben Ayed, Issam H Laradji, Spandana Gella, and Nicolas Gontier. Are online skill and memory modules always worth their tokens? a budget-constrained study of web agents. *arXiv preprint arXiv:2606.15017*, 2026.

Hyungkyu Ham, Wonhyuk Yang, Yunseon Shin, Okkyun Woo, Guseul Heo, Sangyeop Lee, Jongse Park, and Gwangsun Kim. ONNXim: A fast, cycle-level multi-core NPU simulator. *IEEE Computer Architecture Letters*, 23(2):219–222, 2024. doi: 10.1109/LCA.2024.3484648.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in llm agents. *arXiv preprint arXiv:2604.27003*, 2026.

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*, pp. 1–12, 2017.

Ming-Yen Lee and Shimeng Yu. ChatNeuroSim: An LLM agent framework for automated compute-in-memory accelerator deployment and optimization. *arXiv preprint arXiv:2603.08745*, 2026.

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

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. SimpleMem: Efficient lifelong memory for LLM agents. *arXiv preprint arXiv:2601.02553*, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Vinamra Sharma, Xingjian Fu, Jude Haris, and José Cano. LLM-driven design space exploration of FPGA-based accelerators. *arXiv preprint arXiv:2605.05920*, 2026.

Chenyu Wang, Jiahe Caroline Shi, David Kong, Duane S. Boning, Zishen Wan, Yilun Du, and Vijay Janapa Reddi. AgentDSE: Reasoning-augmented architectural design space exploration. *arXiv preprint arXiv:2606.21836*, 2026.

Duo Wang, Mingyu Yan, Yihan Teng, Dengke Han, Haoran Dang, Xiaochun Ye, and Dongrui Fan. A transfer learning framework for high-accurate cross-workload design space exploration of cpu. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

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.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Two-stage transfer surrogate model for automatic hyperparameter optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pp. 199–214. Springer, 2016. doi: 10.1007/978-3-319-46128-1_13.

Jia Xiong, Runkai Li, Chenxu Niu, Guangyuan Gao, Changwen Xing, Yifan Zhang, Xinlai Wan, Jieran Cui, Chen Bai, Yusheng Hua, Ying Wang, Ming Ling, Xi Wang, and Tao Xie. MicroEvo: Knowledge-guided llm sampling for efficient microarchitecture design space exploration. In *2026 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2026.

Runzhen Xue, Hao Wu, Mingyu Yan, Ziheng Xiao, Xiaochun Ye, and Dongrui Fan. MetaDSE: A few-shot meta-learning framework for cross-workload cpu design space exploration. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025. doi: 10.1109/DAC63849.2025.11132668.

Wangsong Yin, Daliang Xu, Mengwei Xu, Gang Huang, and Xuanzhe Liu. Shadownpu: System and algorithm co-design for npu-centric on-device llm inference. In *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services*, pp. 294–308, 2026.

Ziyang Yu, Chen Bai, Shoubo Hu, Ran Chen, Taohai He, Mingxuan Yuan, Bei Yu, and Martin D. F. Wong. IT-DSE: Invariance risk minimized transfer microarchitecture design space exploration. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023. doi: 10.1109/ICCAD57390.2023.10323961.

Boqin Yuan, Yue Su, and Kun Yao. Diagnosing retrieval vs. utilization bottlenecks in LLM agent memory. In *MemAgents Workshop at the International Conference on Learning Representations*, 2026.

Genghan Zhang, Shaowei Zhu, Anjiang Wei, Zhenyu Song, Allen Nie, Zhen Jia, Nandita Vijaykumar, Yida Wang, and Kunle Olukotun. AccelOpt: A self-improving LLM agentic system for AI accelerator kernel optimization. In *Proceedings of Machine Learning and Systems (MLSys)*, 2026a.

Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. LLMCompass: Enabling efficient hardware design for large language model inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, pp. 1080–1096, 2024. doi: 10.1109/ISCA59077.2024.00082.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b.

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

Appendix of RecurNPU

CONTENTS

A Notation List **14**

B Related Work **14**

B.1 Accumulating and reusing experience. 14

B.2 LLM-guided hardware design. 14

B.3 Cross-workload transfer. 15

C Experimental Settings **15**

C.1 Workloads and NPU Design Space 15

C.2 Experience Accumulation and Baselines 16

C.3 Search Budgets and Evaluation 17

D RecurNPU Method Details **17**

D.1 Accumulating Design Experience 17

D.2 Selecting and Adapting Experience 17

D.3 Designing for a New LLM Workload 18

E LLM Prompts **18**

E.1 Profiler 18

E.2 RecurNPU Designer and Adapter 19

E.3 Raw-Mem Designer 21

E.4 Raw-Mem Note 21

F Complete Target Latency Results **22**

G Additional Analyses **26**

G.1 Comparison with the Same Designer 26

G.2 Repeated Searches with the Same Designer 26

G.3 Cost of Accumulating and Reusing Experience 27

G.4 Transfer Across NPU Simulators 28

G.5 Transfer to a New LLM Workload 29

A NOTATION LIST

We summarize the main notation used throughout the paper in the following table.

Notation	Description
w_k, w_t	k-th source workload and the current workload being searched.
Θ, θ	NPU design space and an NPU configuration.
$L(\theta, w_t)$	Simulator-measured latency of configuration θ on workload w_t.
$C(\theta)$	Feasibility condition under the current power and memory constraints.
B_k	Experience bank after source search k.
u_k	Experience card constructed from source search k.
τ_k	Completed trajectory of source search k.
c_k, g_k	Source workload context and constraint vector.
Q_k, E_k	Tested parameter values and parameter-comparison summary.
$\theta_k^\star, L_k^\star$	Best feasible configuration and latency from source search k.
ρ_k	Relative latency ratio within source search k.
R_t, h_t	Retrieved experience cards and Adapter guidance for workload w_t.
ϕ_t	Profile of the current workload.
$\tau_t^{<j}$	Measured search history before simulator evaluation j.
$\theta_t^{j,r}, v_t^{j,r}$	Proposal at attempt r for evaluation j, and its validation feedback.

B RELATED WORK

B.1 ACCUMULATING AND REUSING EXPERIENCE.

Experience memory aims to make earlier interactions useful for later decisions without retraining the language model. Existing methods differ in what they retain and how they prepare it for a new task. Memp combines trajectories with abstract scripts (Fang et al., 2026), ReME extracts and rewrites lessons from execution feedback (Cao et al., 2026), and SimpleMem compresses interactions into structured statements with semantic, lexical, and metadata retrieval (Liu et al., 2026). ReasoningBank distills strategies from self-judged successful and failed attempts (Ouyang et al., 2026), while ACE updates a growing playbook through generation, reflection, and curation (Zhang et al., 2026b). These methods show how experience can be accumulated and revised, but the usefulness of a memory also depends on which evidence is retrieved: a recent study of conversational memory finds that retrieval choices can matter more than write-time summarization (Yuan et al., 2026). In cross-workload NPU DSE, an additional concern is whether a design observation still applies under the next workload's memory requirement and power budget. RecurNPU therefore retains measured hardware comparisons together with their source conditions, selects relevant evidence, and adapts its interpretation before search.

B.2 LLM-GUIDED HARDWARE DESIGN.

Hardware exploration gives language models a concrete feedback loop: proposed designs can be evaluated and revised using simulator measurements. LLM-DSE explores high-level synthesis directives (Wang et al., 2025), LUMINA uses bottleneck analysis to guide GPU architecture search (Zhang et al., 2026c), and AgentDSE studies accelerator mapping, hardware/software co-design, and CPU cache optimization (Wang et al., 2026). Recent systems also automate parts of the evaluation process. ChatNeuroSim checks parameter dependencies and prepares simulations for compute-in-memory accelerators (Lee & Yu, 2026), while ArchAgent v2 combines prefetcher search with hardware-size feedback (Gonzalez et al., 2026). Experience reuse has also been explored in kernel optimization: AccelOpt stores insights from slow–fast kernel pairs (Zhang et al., 2026a), and KernelBlaster maintains knowledge across CUDA tasks (Dong et al., 2026). These results motivate using simulator feedback as a source of design experience. Our focus is how that experience transfers between LLM workloads when the search variables are NPU hardware configurations. Each completed source search contributes observations that may guide later searches, while measurements on the current workload determine whether the inherited guidance helps.

B.3 CROSS-WORKLOAD TRANSFER.

Reusing earlier evaluations is also a goal of classical DSE. RGPE combines source and target Gaussian-process models (Feurer et al., 2018); IT-DSE learns representations that transfer across microarchitecture design tasks (Yu et al., 2023); and MetaDSE adapts CPU performance prediction to new workloads through few-shot meta-learning (Xue et al., 2025). These approaches address workload differences through predictive models. RecurNPU instead carries measured observations through the context of a fixed language model, so later searches depend on experience produced by earlier experience-conditioned searches. Recent memory benchmarks examine related sequential settings: Evo-Memory evaluates memory updates over task streams (Wei et al., 2025), and MemoryArena tests whether experience from earlier actions helps solve interdependent tasks (He et al., 2026). Studies of interactive agents also find that accumulated experience can cause negative transfer, with its representation and retrieval affecting later performance (Hu et al., 2026). Our benchmark brings this question to NPU design by varying LLM family, scale, and power constraints across independently constructed workload chains. It measures both the latency benefit of reuse and cases where inherited experience worsens a target design under a common simulator-evaluation budget.

C EXPERIMENTAL SETTINGS

We give the workload configurations, hardware choices, and search budgets used in Sections 2 and 4. Appendix F reports the target latency for every setting.

C.1 WORKLOADS AND NPU DESIGN SPACE

Table 5 lists the nine hardware parameters and their allowed values in NeuSim, giving 90,316,800 configurations. The objective is to minimize latency subject to the power budget and HBM capacity requirement. NeuSim provides phase latency, analytic peak power, and memory footprint per chip; memory capacity and footprint use binary units, with one GB denoting 2^{30} bytes.

Table 5: The NPU design space for the primary NeuSim benchmark.

Parameter	Levels	Values
num_sa (systolic arrays)	10	2, 3, 4, 5, 6, 8, 10, 12, 14, 16
num_vu (vector units)	8	2, 3, 4, 5, 6, 8, 10, 12
sa_dim (array dimension)	7	48, 64, 96, 128, 192, 256, 384
freq_GHz	8	0.5, 0.625, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0
vmem_size_MB	10	16, 32, 48, 64, 96, 128, 192, 256, 384, 512
hbm_size_GB	8	8, 16, 24, 32, 40, 48, 64, 80
hbm_bw_GBps	7	100, 150, 200, 300, 400, 500, 600
ici_bw_GBps	6	10, 25, 50, 75, 100, 125
dcn_bw_GBps	6	1.25, 2.5, 5.0, 7.5, 10.0, 12.5

Tables 6 and 7 define the ordered sources and the eight targets. All workloads use context length 4096 and data parallelism one; targets use prefill with batch 16, while sources vary inference phase and batch size. Sources have a 400 W budget, and each target model is evaluated at 400 W and 300 W. Llama sources and base targets use 8 chips with TP8; large Llama targets use 128 chips with TP8/PP16. DeepSeek sources and base targets use 64 chips with TP8/PP4/EP2, and large DeepSeek targets use 128 chips with TP8/PP8/EP2. TP, PP, and EP denote tensor, pipeline, and expert parallelism.

We vary the source family and history length for each target to examine how the experience available before search affects the resulting design. A setting combines one target, one source family, and the first $n \in \{1, \ldots, 6\}$ workloads from that family's ordered source sequence, giving $8 \times 2 \times 6 = 96$ settings. Family shifts transfer between dense Llama and MoE DeepSeek; scale shifts change both the model size and its prescribed deployment; constraint shifts lower the target power budget from 400 to 300 W. Their combinations form seven shift classes, plus a matched control, with twelve settings in each class. Matched means the same family, base scale, and 400 W budget; individual sources can still differ from the target in phase, batch size, or model size. Exact workload matches occur at L_S6 and D_S3 when those sources are included.

Table 6: Ordered source workloads, each with a power budget of 400 W and context length 4096. Each LLM setting independently searches the first n workloads from its selected family.

Id	Model	Phase	Batch
L_S1	Llama3-8B	prefill	8
L_S2	Llama2-13B	prefill	8
L_S3	Llama3-8B	decode	8
L_S4	Llama2-13B	decode	8
L_S5	Llama3-8B	prefill	16
L_S6	Llama2-13B	prefill	16
D_S1	DeepSeek-v2-236B	prefill	8
D_S2	DeepSeek-v2-236B	decode	8
D_S3	DeepSeek-v2-236B	prefill	16
D_S4	DeepSeek-v2-236B	decode	16
D_S5	DeepSeek-v2-236B	prefill	32
D_S6	DeepSeek-v2-236B	decode	32

Table 7: Target workloads. M denotes the source power budget of 400 W, and T denotes the tighter target budget of 300 W.

Id	Model	Scale	Constraint	Budget (W)
T_L_base_M	Llama2-13B	base	matched	400
T_L_base_T	Llama2-13B	base	tight	300
T_L_large_M	Llama3.1-405B	large	matched	400
T_L_large_T	Llama3.1-405B	large	tight	300
T_D_base_M	DeepSeek-v2-236B	base	matched	400
T_D_base_T	DeepSeek-v2-236B	base	tight	300
T_D_large_M	DeepSeek-v3-671B	large	matched	400
T_D_large_T	DeepSeek-v3-671B	large	tight	300

C.2 Experience Accumulation and Baselines

Each LLM memory method builds its own source history in every setting, starting with empty experience. Source w_k can use only experience from the completed sources $w_1, \ldots, w_{k-1}$ in that setting, then contributes observations from its own search. After all n sources, the target uses the accumulated experience without updating it. Covering all settings therefore requires 336 source searches and 96 target searches per memory method. RecurNPU keeps the cards from completed searches and selects at most two before each new workload; the first source has no cards and skips adaptation. Raw-Mem appends a note of at most 50 words after each source and passes all preceding notes to the next search. Both methods follow the same sequence and evaluation budgets, while gathering their own observations along the way.

ReME and SimpleMem use the same Designer and validation feedback as RecurNPU, so these comparisons examine how each method stores and retrieves experience. Our ReME adaptation extracts up to three lessons supported by each source search without additional simulator evaluations. It retrieves up to ten candidates with `text-embedding-3-small`, then reranks and rewrites at most two for the Designer. SimpleMem compresses each completed source search into structured statements, using windows of 40 turns with an overlap of two turns. It combines semantic, lexical, and metadata retrieval, with at most two rounds of refinement. This adaptation uses `gpt-5.4-mini` for memory operations and the same embedding model as ReME; retrieval uses exact cosine search, BM25, and metadata filtering. Raw-Mem keeps its original Designer in the main comparison. Appendix G.1 compares Raw-Mem, ReME, and RecurNPU with the same Designer, and Appendix E gives the role prompts.

Table 1 includes five classical methods that search only the target and four that transfer source results. The transfer methods use evaluations from sequential Bayesian optimization searches, initializing each source search with the best designs found on preceding sources. These methods reuse one source history per family across target comparisons; each LLM memory method instead rebuilds its history within every setting. All methods have a limit of twelve target evaluations, and the

classical runs use random seed 0. Methods that search only the target have the same results across source histories because those histories are not part of their input.

C.3 Search Budgets and Evaluation

Each LLM source search allows up to 30 simulator evaluations and each target up to 12. The Designer has ten attempts to produce a valid proposal for the next evaluation; search stops if all ten fail. Rejected proposals do not consume simulator evaluations. We compute memory requirements before search and exclude any latency obtained during setup from the search results. With `gpt-5.4-mini-2026-03-17`, the Profiler and Adapter use temperature 0 and the Designer uses 0.8. Raw-Mem requested the same model family, but its earlier execution does not establish that it used the same dated model version. The common workload sequence and evaluation budgets therefore do not imply identical model responses.

We report the best feasible target TTFT and compare it with the median of twelve No-Mem runs for that target. A method incurs negative transfer if it finds no feasible design or exceeds this reference by more than 1%. The full 96-setting comparison uses one execution per method and setting; Appendix G.2 repeats sixteen settings with the same Designer. Four settings were used during prompt development: Constraint and Scale + Constraint for both target families, each with sources from the target's family and $n = 3$. We then fix the method configuration for all 96 settings. The primary analysis covers 84 shifted settings, and Section 4.1 also reports the 80 remaining settings after removing the four used during development.

D RecurNPU Method Details

D.1 Accumulating Design Experience

The card in Eq. 2 is computed from the completed source search. It keeps the source conditions, values tested for each hardware parameter, best feasible configuration, and comparisons that change one parameter at a time. The main resource bottleneck is determined from the simulator's compute, memory, and inter-chip communication components. This describes the source workload; it does not by itself establish how a hardware change affects latency. The required memory per chip is rounded upward to the next 0.01 GB. For each hardware parameter, we collect pairs of feasible configurations from source search k that differ only in that parameter. For each pair, we divide the latency at the higher parameter value by the latency at the lower value. A ratio below 0.99 favors the higher value, a ratio above 1.01 favors the lower value, and a ratio from 0.99 through 1.01 is treated as a tie. The parameter comparison summary $\mathcal{E}_k$ records the number of pairs, the counts of these three outcomes, the median ratio, and the pair whose ratio has the largest absolute logarithm. If no such pair exists, no trend is recorded for that parameter. These comparisons apply to the tested pair and the fixed values of its other hardware parameters. The full list of tested values may include configurations that do not meet the constraints, while the comparisons use only feasible configurations. The relative latency ratio $\rho_k = \mathcal{L}_k^* / \mathcal{L}_k^{\text{first feasible}}$ compares the best and first feasible latencies from the same source search; smaller values indicate a larger latency reduction.

D.2 Selecting and Adapting Experience

The similarity rule in Section 3.2 compares five attributes of source workload w_k and current workload w_t in order:

$$d(w_k, w_t) = (\mathbb{I}[f_k \neq f_t],\ \mathbb{I}[p_k \neq p_t],\ |\log_2(\beta_k/\beta_t)|,\ |\log_2(s_k/s_t)|,\ \mathbb{I}[\delta_k \neq \delta_t])\,. \tag{6}$$

Here f, p, β, s, and δ denote model family, inference phase, batch size, model scale in billions of parameters, and deployment, with subscripts k and t referring to the source and current workload. The indicator $\mathbb{I}[\cdot]$ is one when its condition holds and zero otherwise. We compare the entries from left to right, using a later entry only when the earlier entries tie. The closest card minimizes this ordered distance. A second card minimizes the relative latency ratio ρ_k among sources with $\rho_k < 0.99$. Both rules require a feasible source design and use source identifiers to resolve remaining ties. Missing or nonpositive batch-size or model-scale values receive infinite distance. Duplicate selections are included once. If no source has a feasible design, no card is selected.

The Adapter writes the guidance hint h_t from the selected cards R_t, exact model configuration, and computed facts for the current workload, including its power budget and required memory per chip. This note supplements the cards, which the Designer also receives in their original form. It retains measured source observations unless a current workload fact gives a reason to revise their use or flag a conflict. The note names the condition responsible for each revision or warning. A change in model family or scale alone is not evidence that a source observation has reversed. Claims about increasing or decreasing a hardware parameter must follow the comparisons in $\mathcal{E}_k$, including their tie outcomes and tested values. The Adapter also distinguishes a known capacity constraint from an untested performance claim: enough memory establishes that the capacity requirement is met, but does not establish low target latency. The selected cards remain fixed during adaptation. The Adapter does not predict current-workload latency or choose the next configuration. Values beyond those tested on the sources remain candidates for the Designer to consider within the allowed design space. Agreement among cards is not treated as independent replication: later cards may come from searches conditioned on earlier cards, making evidence along the recursive chain statistically dependent. The Profiler's workload summary is supplied separately to the Designer, together with the guidance hint h_t and selected cards R_t.

D.3 Designing for a New LLM Workload

The workload w_t provides the full model and execution configuration, while the workload profile ϕ_t summarizes its design-relevant facts without adding simulator measurements. The constraint vector $g_t = (\bar{P}_t, \bar{M}_t)$ specifies the power budget and required memory per chip. As in Section 3.3, $\tau_t^{<j}$ contains the first $j-1$ measured configuration–evaluation pairs, excluding proposals rejected before simulation. The index j counts simulator evaluations, and r counts proposal attempts for evaluation j, including the first attempt. Search begins with $\tau_t^{<1} = []$; each evaluation starts at $r = 1$ with empty validation feedback, $v_t^{j,0} = \emptyset$.

The Harness checks allowed parameter values, repeated configurations, HBM capacity, and analytic peak power. A rejected proposal updates the validation feedback $v_t^{j,r}$ and advances the attempt counter to $r+1$, without changing j or $\tau_t^{<j}$ or consuming a simulator evaluation. When a constraint is violated, the feedback gives the power excess or memory shortfall. For a power violation, it also computes the savings from lowering each parameter by one level or to its minimum, and the highest lower value that would satisfy the budget if only that parameter changed. Parameters with no further power saving from a decrease are identified, including those already at their minimum; this does not imply that increasing them has no power cost. These calculations describe feasibility without predicting latency, leaving the Designer to revise the candidate. Once a proposal passes validation, it is simulated and its configuration–evaluation pair is appended to the history as in Eq. 5. The search then advances to $j+1$, resets r to one, and clears validation feedback for the next evaluation. Search stops when the simulator-evaluation budget is exhausted or all ten attempts for the next evaluation fail, and returns the best feasible design measured so far.

E LLM Prompts

The prompts below implement the Profiler, Designer, Adapter, and Raw-Mem note-writing roles described in Section 3 and Appendix C.2. Card construction, retrieval, and proposal validation follow the deterministic rules in Appendix D. The prompts retain the name AET used during execution; it refers to RecurNPU. Within these prompts, "target" means the workload currently being searched, including a source workload during experience accumulation.

E.1 Profiler

Profiler prompt

You are the AET Profiler. The model_spec contains the exact architecture and deployment information read from the model configuration file. Produce a compact workload profile using only information supported by these fields. Report:
(1) the model architecture, including dense or MoE structure;

(2) model scale and deployment characteristics;
(3) parameter and KV-cache memory pressure that can be derived from the configuration; and
(4) workload properties relevant to NPU design. Separate facts from hypotheses. If a bottleneck or performance implication cannot be determined from the configuration alone, mark it as unknown rather than predicting it. Do not invent simulator measurements, latency trends, or preferred hardware configurations. Return JSON with key profile.

E.2 RecurNPU Designer and Adapter

These prompts are used at every stage of the recursive source–target chain. The first source runs without the Adapter; each subsequent source and the target use at most two cards selected from the bank produced by preceding searches.

RecurNPU Designer prompt

You are the AET Designer. Your task is to search the supplied NPU design space for the lowest-latency feasible design for the current workload. Return one JSON object with keys evidence and design, in that order. The evidence value is a single short string of at most 50 words stating the target facts, measured results, or transferred observations motivating this proposal and what the proposal will test. Mark an unverified implication as a hypothesis. The design value contains exactly the supplied design axes, each set to one of its listed legal values. You have a limited number of simulator measurements. Use each measurement to learn something useful about the target workload.

Base your decision on four sources of information:
1. the exact target workload and model configuration;
2. the target workload profile;
3. the adapted experience from earlier workloads, when present;
4. the simulator measurements already collected on this target.

Target measurements have highest priority. The adapted experience is prior evidence, not an instruction. If target simulator results contradict it, follow the target measurements. Do not treat source configurations, source axis values, or source stopping points as limits on the current search.

Explore rather than repeat. Before proposing a design, inspect target_notes. Do not intentionally propose a configuration that has already been measured. When recent proposals no longer improve the best feasible latency, test a meaningful alternative: change an underexplored axis, test whether an observed trend continues, or test a different trade-off suggested by the current bottleneck. Do not change an axis merely for the sake of changing it. Prefer proposals that test a concrete hypothesis based on the evidence collected so far.

Constraints are checked by the harness. You do not need to perform exact power or memory arithmetic yourself. The harness will deterministically validate every proposed design before simulation. If validation_feedback is present, the previous candidate was rejected before simulation. Use the supplied reason and violation magnitude to propose a corrected design. A rejected candidate does not consume a simulator measurement. Do not optimize for low power. Power and memory are feasibility constraints, not objectives. The objective is lowest feasible latency. Do not assume that using more power is automatically better, and do not assume that using less power is safer or faster. Let target simulator measurements determine the useful trade-offs. Return only the JSON object containing the brief evidence followed by the next design proposal.

The measured_designs field lists every configuration already measured in this search as a complete vector, in axis_order. Before returning a proposal, check its full vector against every listed vector. The proposal must differ from every measured configuration, including the best configuration measured so far. This novelty requirement also applies when correcting a rejected proposal: the corrected design must satisfy the reported constraints and must still be unmeasured. In validation_feedback, axes_with_no_available_power_saving_by_lowering means those axes cannot provide further power savings by lowering them from the rejected design. An axis already at its minimum is included. This does not mean increasing that axis has no power cost. Use the remaining diagnostic fields with this directional meaning.

RecurNPU Adapter prompt

You are the AET Experience Adapter. You receive the current workload and the one or two experience cards selected from earlier workloads. Each card carries structured fields filled by the

harness: the workload context, its constraints (power budget and per-chip memory need), the per-axis coverage of values that search actually measured, and the best feasible design found with its measured latency; and controlled one-axis effects extracted deterministically from feasible pairs. Compare coverage directly from these fields: a value outside a card's coverage was not measured there, and a trend holds only within the coverage that supports it. Your task is to decide how those observations should be presented to the Designer for the current workload.

Default to preserving measured observations. A difference between source and target conditions is not, by itself, evidence that an observation becomes false. Do not rewrite an observation merely because the model family, model scale, batch size, deployment, or another workload field has changed. For each source observation, choose one of three outcomes: 1. Retain. Preserve the observation when there is no concrete target fact that contradicts the conditions supporting it. If its applicability is uncertain, keep the observation but state that its transfer to the current workload is unverified. 2. Revise. Revise the implication only when a known target fact gives a concrete reason that the source implication changes. State exactly which condition changed and why that change matters. 3. Warn. If a source recommendation or design would conflict with an authoritative target constraint or is unsupported under the target conditions, present it as a warning rather than usable advice.

The target model_spec is exact and comes from the configuration file. The target_facts are authoritative deterministic facts supplied by the harness, such as the power budget and required per-chip memory. Do not infer an axis direction from model size, family, batch size, or deployment alone. Architectural differences help you judge relevance, but an axis trend is evidence only when the structured axis_effects field contains controlled feasible pairs that differ on that axis and no other axis. Correlated trajectory points are not causal evidence. Do not turn source stopping points into target limits. If a source trend was still improving at the edge of its explored range, retain it as unresolved and state that values beyond the source range remain worth testing.

The retrieved_anchor_source_ids were selected before this call by two complementary deterministic rules: closest target context, and largest improvement measured within the source's own search. They may name the same card, in which case only one is returned. Explain why each card's evidence is or is not applicable, but do not replace them based on guessed target performance. A source design is only source-side evidence: the Harness will not copy, project, repair, or evaluate it automatically on the target.

When cards agree, preserve their measured source observations and identify the conditions and coverage supporting each observation. Cards can come from a sequential source chain in which later searches used earlier experience; agreement is not independent replication and does not establish that the trend holds under changed target conditions. Transfer remains unverified until the target supplies evidence. Unexplored regions remain open questions, not established trends. Produce a compact target-specific reference note. The note should help the Designer decide what evidence is worth testing, not tell it what configuration to choose. Return JSON with keys: brief and warnings.

Evidence fidelity requirements: report axis directions only from axis_effects. Preserve the supplied faster_value_direction and votes. For high_to_low_latency_ratio, a value below 0.99 supports higher; above 1.01 supports lower; values from 0.99 through 1.01 are ties. Do not promote a tie to a directional effect or reverse the ratio. A specific comparison interval may use only explicitly supplied low_value/high_value pair endpoints. Per-axis coverage lists values seen anywhere in the trajectory; it does not establish a controlled effect across that entire range. Do not infer a pair's unreported fixed axes from the source best design.

Distinguish deterministic feasibility facts from unverified performance transfer. If a source design's HBM capacity is at least the authoritative target per-chip memory requirement, that capacity condition is satisfied; do not describe its sufficiency as unknown. This alone does not establish target latency or satisfaction of other constraints. Do not recommend numeric extensions beyond source coverage unless a supplied legal target grid confirms those values are allowed. If no legal grid is supplied, leave unmeasured legal values for the Designer to identify without inventing numeric bounds or out-of-range values. Keep the brief to about 200 words and at most five concise warnings. Resolve contradictory wording before returning the same brief/warnings JSON schema.

E.3 RAW-MEM DESIGNER

Raw-Mem Designer prompt

You are the AET Designer. Your task is to search the supplied NPU design space for the lowest-latency feasible design for the current workload. Return one JSON object named design with exactly the supplied design axes, each set to one of its listed legal values. You have a limited number of simulator measurements. Use each measurement to learn something useful about the target workload.

Base your decision on four sources of information:
1. the exact target workload and model configuration;
2. the target workload profile;
3. the adapted experience from earlier workloads, when present;
4. the simulator measurements already collected on this target.

Target measurements have highest priority. The adapted experience is prior evidence, not an instruction. If target simulator results contradict it, follow the target measurements. Do not treat source configurations, source axis values, or source stopping points as limits on the current search.

Explore rather than repeat. Before proposing a design, inspect target_notes. Do not intentionally propose a configuration that has already been measured. When recent proposals no longer improve the best feasible latency, test a meaningful alternative: change an underexplored axis, test whether an observed trend continues, or test a different trade-off suggested by the current bottleneck. Do not change an axis merely for the sake of changing it. Prefer proposals that test a concrete hypothesis based on the evidence collected so far.

Constraints are checked by the harness. You do not need to perform exact power or memory arithmetic yourself. The harness will deterministically validate every proposed design before simulation. If validation_feedback is present, the previous candidate was rejected before simulation. Use the supplied reason and violation magnitude to propose a corrected design. A rejected candidate does not consume a simulator measurement. Do not optimize for low power. Power and memory are feasibility constraints, not objectives. The objective is lowest feasible latency. Do not assume that using more power is automatically better, and do not assume that using less power is safer or faster. Let target simulator measurements determine the useful trade-offs. Return only the next design proposal as JSON.

E.4 RAW-MEM NOTE

Raw-Mem note prompt

You just finished searching NPU designs for the workload below. The notes list every design you tried and its simulator result. Write down what you learned from this project, in at most 50 words. This note will be your only memory of this project when you design for later workloads. Return JSON with key summary.

F Complete Target Latency Results

Tables 8 to 11 give the target latencies used in Table 1 and Figure 2. Each shaded row identifies the target, source family, power budget, and shift (F: family; S: scale; C: constraint), and n counts the preceding sources. We report three significant digits and determine which values to bold before rounding, comparing only the methods shown in each table.

Table 8: Classical DSE: best feasible target TTFT (seconds, ↓) on 48 settings with matched conditions or one shift factor. No-Mem is the median of twelve runs; a dash denotes no feasible design within twelve evaluations. Bold values are within 1% of the row minimum before rounding.

n	Random	Grid	Greedy	CMA-ES	BO	T-BO	RGPE	TST-R	NaiveApply	No-Mem
Matched	Target: DeepSeek-v2-236B		Source: DeepSeek		400 W					
1	51.6	90.0	231	112	180	47.0	46.2	180	50.0	**45.5**
2	51.6	90.0	231	112	180	**31.6**	150	180	36.6	45.5
3	51.6	90.0	231	112	180	**31.6**	**31.6**	**31.6**	**31.6**	45.5
4	51.6	90.0	231	112	180	**30.9**	31.6	31.6	31.6	45.5
5	51.6	90.0	231	112	180	31.4	**30.9**	31.6	31.4	45.5
6	51.6	90.0	231	112	180	**31.4**	**31.6**	**31.6**	**31.4**	45.5
Matched	Target: Llama2-13B		Source: Llama		400 W					
1	4.22	11.8	–	14.1	1.38	**1.00**	1.31	1.35	**1.00**	1.59
2	4.22	11.8	–	14.1	1.38	**1.00**	1.31	1.50	**1.00**	1.59
3	4.22	11.8	–	14.1	1.38	**1.00**	1.31	1.50	**1.00**	1.59
4	4.22	11.8	–	14.1	1.38	**1.00**	1.31	1.16	**1.00**	1.59
5	4.22	11.8	–	14.1	1.38	**1.00**	1.35	1.35	**1.00**	1.59
6	4.22	11.8	–	14.1	1.38	**0.966**	1.41	1.16	**0.966**	1.59
F	Target: DeepSeek-v2-236B		Source: Llama		400 W					
1	51.6	90.0	231	112	180	46.0	47.0	69.1	46.0	**45.5**
2	51.6	90.0	231	112	180	46.0	**37.8**	45.7	46.0	45.5
3	51.6	90.0	231	112	180	39.2	**37.8**	40.3	46.0	45.5
4	51.6	90.0	231	112	180	39.2	**37.8**	38.7	46.0	45.5
5	51.6	90.0	231	112	180	39.2	**36.6**	38.7	46.0	45.5
6	51.6	90.0	231	112	180	**32.6**	36.6	36.6	36.6	45.5
F	Target: Llama2-13B		Source: DeepSeek		400 W					
1	4.22	11.8	–	14.1	**1.38**	2.23	2.34	2.18	–	1.59
2	4.22	11.8	–	14.1	**1.38**	2.23	2.34	2.18	–	1.59
3	4.22	11.8	–	14.1	**1.38**	–	2.09	2.83	–	1.59
4	4.22	11.8	–	14.1	1.38	**1.35**	2.09	2.23	–	1.59
5	4.22	11.8	–	14.1	1.38	**1.06**	2.34	2.23	–	1.59
6	4.22	11.8	–	14.1	1.38	**1.06**	4.22	2.23	–	1.59
S	Target: DeepSeek-v3-671B		Source: DeepSeek		400 W					
1	511	185	193	128	322	105	80.7	131	105	**78.6**
2	511	185	193	128	322	85.1	109	162	105	**78.6**
3	511	185	193	128	322	**65.3**	95.9	97.3	**65.3**	78.6
4	511	185	193	128	322	**65.3**	77.6	97.3	**65.3**	78.6
5	511	185	193	128	322	**65.2**	76.9	97.3	**65.2**	78.6
6	511	185	193	128	322	**65.2**	76.9	94.8	**65.2**	78.6
S	Target: Llama3.1-405B		Source: Llama		400 W					
1	135	40.9	–	45.1	27.7	**22.4**	44.7	27.7	22.7	**22.3**
2	135	40.9	–	45.1	27.7	**22.4**	27.5	**22.4**	22.7	**22.3**
3	135	40.9	–	45.1	27.7	**22.4**	25.9	**22.4**	22.7	**22.3**
4	135	40.9	–	45.1	27.7	**22.4**	117	**22.4**	22.7	**22.3**
5	135	40.9	–	45.1	27.7	**22.4**	32.3	32.4	22.7	**22.3**
6	135	40.9	–	45.1	27.7	**22.2**	24.4	22.4	**22.2**	**22.3**
C	Target: DeepSeek-v2-236B		Source: DeepSeek		300 W					
1	875	129	–	112	**45.1**	46.6	62.7	**45.1**	50.0	48.5
2	875	129	–	112	45.1	**36.6**	70.7	67.3	**36.6**	48.5
3	875	129	–	112	45.1	**35.7**	46.1	58.6	36.6	48.5
4	875	129	–	112	45.1	**32.5**	46.1	58.6	**32.5**	48.5
5	875	129	–	112	45.1	**31.4**	46.1	71.9	**31.4**	48.5
6	875	129	–	112	45.1	**31.4**	37.1	71.9	**31.4**	48.5
C	Target: Llama2-13B		Source: Llama		300 W					
1	–	11.8	–	5.16	8.89	–	–	**1.93**	–	2.57
2	–	11.8	–	5.16	8.89	–	1.50	**1.46**	–	2.57
3	–	11.8	–	5.16	8.89	2.51	21.7	**1.46**	4.84	2.57
4	–	11.8	–	5.16	8.89	2.51	**1.35**	2.50	4.84	2.57
5	–	11.8	–	5.16	8.89	2.51	**1.35**	2.50	4.84	2.57
6	–	11.8	–	5.16	8.89	2.51	4.90	**1.35**	4.84	2.57

Table 9: Classical DSE: best feasible target TTFT (seconds, ↓) on 48 settings with combined shifts. No-Mem is the median of twelve runs; a dash denotes no feasible design within twelve evaluations. Bold values are within 1% of the row minimum before rounding.

n	Random	Grid	Greedy	CMA-ES	BO	T-BO	RGPE	TST-R	NaiveApply	No-Mem
F+S	Target: DeepSeek-v3-671B		Source: Llama	400 W						
1	511	185	193	128	322	82.9	482	322	94.5	**78.6**
2	511	185	193	128	322	82.9	85.1	322	94.5	**78.6**
3	511	185	193	128	322	**65.2**	85.1	88.0	94.5	78.6
4	511	185	193	128	322	**65.2**	92.9	96.9	94.5	78.6
5	511	185	193	128	322	**65.2**	92.9	96.9	94.5	78.6
6	511	185	193	128	322	**75.4**	80.0	96.9	**75.4**	78.6
F+S	Target: Llama3.1-405B		Source: DeepSeek	400 W						
1	135	40.9	–	45.1	27.7	27.7	58.9	42.9	–	**22.3**
2	135	40.9	–	45.1	27.7	34.7	58.9	42.9	–	**22.3**
3	135	40.9	–	45.1	27.7	45.4	58.9	75.1	–	**22.3**
4	135	40.9	–	45.1	27.7	45.4	58.9	75.1	–	**22.3**
5	135	40.9	–	45.1	27.7	32.4	58.9	42.9	–	**22.3**
6	135	40.9	–	45.1	27.7	32.4	52.0	42.2	–	**22.3**
F+C	Target: DeepSeek-v2-236B		Source: Llama	300 W						
1	875	129	–	112	45.1	45.1	**39.9**	81.3	–	48.5
2	875	129	–	112	**45.1**	**45.1**	46.5	81.3	–	48.5
3	875	129	–	112	45.1	**32.5**	46.5	59.7	63.8	48.5
4	875	129	–	112	45.1	**32.5**	39.2	47.9	63.8	48.5
5	875	129	–	112	45.1	**32.5**	39.2	67.6	63.8	48.5
6	875	129	–	112	45.1	**32.5**	39.2	47.9	63.8	48.5
F+C	Target: Llama2-13B		Source: DeepSeek	300 W						
1	–	11.8	–	5.16	8.89	5.73	17.6	–	–	**2.57**
2	–	11.8	–	5.16	8.89	8.69	8.92	–	–	**2.57**
3	–	11.8	–	5.16	8.89	5.67	–	–	–	**2.57**
4	–	11.8	–	5.16	8.89	–	–	–	–	**2.57**
5	–	11.8	–	5.16	8.89	**1.41**	–	–	–	2.57
6	–	11.8	–	5.16	8.89	**1.41**	–	1.80	–	2.57
S+C	Target: DeepSeek-v3-671B		Source: DeepSeek	300 W						
1	383	274	–	489	341	105	154	341	105	**90.2**
2	383	274	–	489	341	105	95.9	341	105	**90.2**
3	383	274	–	489	341	105	154	128	105	**90.2**
4	383	274	–	489	341	**68.1**	94.9	434	**68.1**	90.2
5	383	274	–	489	341	**65.2**	**65.2**	79.9	**65.2**	90.2
6	383	274	–	489	341	**65.2**	**65.2**	77.6	**65.2**	90.2
S+C	Target: Llama3.1-405B		Source: Llama	300 W						
1	120	340	–	251	46.7	46.7	50.7	35.3	–	**32.1**
2	120	340	–	251	46.7	46.7	120	35.3	–	**32.1**
3	120	340	–	251	46.7	120	89.5	35.3	131	**32.1**
4	120	340	–	251	46.7	120	63.8	37.0	131	**32.1**
5	120	340	–	251	46.7	120	63.8	37.0	131	**32.1**
6	120	340	–	251	46.7	120	42.5	43.5	131	**32.1**
F+S+C	Target: DeepSeek-v3-671B		Source: Llama	300 W						
1	383	274	–	489	341	1020	430	341	–	**90.2**
2	383	274	–	489	341	1020	517	341	–	**90.2**
3	383	274	–	489	341	103	**64.9**	**65.2**	136	90.2
4	383	274	–	489	341	103	**65.2**	132	136	90.2
5	383	274	–	489	341	103	111	**65.2**	136	90.2
6	383	274	–	489	341	**65.2**	**65.8**	91.0	136	90.2
F+S+C	Target: Llama3.1-405B		Source: DeepSeek	300 W						
1	120	340	–	251	46.7	46.7	43.7	46.7	–	**32.1**
2	120	340	–	251	46.7	120	43.7	46.7	–	**32.1**
3	120	340	–	251	46.7	120	43.7	80.7	–	**32.1**
4	120	340	–	251	46.7	120	43.7	66.8	–	**32.1**
5	120	340	–	251	46.7	32.8	33.3	63.0	–	**32.1**
6	120	340	–	251	46.7	32.8	50.7	63.0	–	**32.1**

Table 10: NPU DSE with LLMs: best feasible target TTFT (seconds, ↓) on 48 settings with matched conditions or one shift factor. No-Mem is the median of twelve runs; each memory method uses its own recursively accumulated source history. Bold values are within 1% of the row minimum before rounding.

n	No-Mem	Raw-Mem	ReME	SimpleMem	**RecurNPU**
Matched	Target: DeepSeek-v2-236B	Source: DeepSeek	400 W		
1	45.5	**30.5**	76.2	**30.5**	**30.5**
2	45.5	122	**30.5**	**30.5**	**30.5**
3	45.5	**30.5**	89.9	**30.5**	**30.7**
4	45.5	31.0	72.9	**30.5**	**30.5**
5	45.5	59.9	**30.5**	44.8	31.0
6	45.5	**30.5**	**30.7**	**30.5**	**30.5**
Matched	Target: Llama2-13B	Source: Llama	400 W		
1	1.59	1.78	5.99	0.966	**0.943**
2	1.59	**0.943**	1.39	1.39	**0.943**
3	1.59	1.30	5.87	1.88	**0.943**
4	1.59	**0.943**	**0.943**	0.972	**0.943**
5	1.59	**0.943**	**0.943**	2.54	**0.943**
6	1.59	**0.943**	1.00	**0.943**	**0.943**
F	Target: DeepSeek-v2-236B	Source: Llama	400 W		
1	45.5	**30.5**	45.4	95.4	**30.5**
2	**45.5**	103	93.1	115	46.0
3	45.5	**30.5**	32.5	61.0	**30.5**
4	45.5	**31.0**	37.8	**30.9**	**31.0**
5	45.5	**30.5**	112	44.8	31.0
6	45.5	**31.0**	32.9	103	**31.0**
F	Target: Llama2-13B	Source: DeepSeek	400 W		
1	1.59	1.78	**0.943**	**0.943**	**0.943**
2	1.59	3.19	1.75	1.00	**0.943**
3	1.59	1.11	1.44	**0.943**	**0.943**
4	1.59	**0.943**	**0.943**	2.84	**0.943**
5	1.59	1.88	**0.943**	**0.943**	**0.943**
6	1.59	**0.949**	5.99	1.77	**0.943**
S	Target: DeepSeek-v3-671B	Source: DeepSeek	400 W		
1	78.6	**62.9**	101	125	**62.9**
2	78.6	**62.9**	77.9	**62.9**	**62.9**
3	78.6	65.3	124	**62.9**	**62.9**
4	78.6	**62.9**	63.9	69.1	**62.9**
5	78.6	71.3	93.5	180	**62.9**
6	78.6	623	**62.9**	**62.9**	**62.9**
S	Target: Llama3.1-405B	Source: Llama	400 W		
1	22.3	55.9	49.7	49.8	**21.9**
2	22.3	24.4	24.2	26.2	**21.9**
3	22.3	**21.9**	27.1	24.1	**21.9**
4	22.3	**21.9**	30.0	**21.9**	**21.9**
5	22.3	**21.9**	171	80.4	**21.9**
6	22.3	55.9	24.2	**21.9**	22.3
C	Target: DeepSeek-v2-236B	Source: DeepSeek	300 W		
1	48.5	35.0	89.9	**31.2**	75.6
2	48.5	**33.3**	121	38.4	41.7
3	48.5	139	32.5	32.5	**31.2**
4	48.5	35.7	39.8	32.5	**31.6**
5	48.5	139	135	**31.2**	32.5
6	48.5	**31.6**	47.1	45.0	46.5
C	Target: Llama2-13B	Source: Llama	300 W		
1	2.57	1.77	2.23	**1.35**	2.57
2	2.57	2.12	5.79	1.61	**1.35**
3	2.57	3.86	4.32	**1.35**	1.71
4	2.57	1.59	8.69	**1.35**	8.57
5	2.57	**1.35**	4.23	1.77	**1.35**
6	2.57	1.89	2.83	1.41	**1.35**

Table 11: NPU DSE with LLMs: best feasible target TTFT (seconds, ↓) on 48 settings with combined shifts. No-Mem is the median of twelve runs; each memory method uses its own recursively accumulated source history. Bold values are within 1% of the row minimum before rounding.

n	No-Mem	Raw-Mem	ReME	SimpleMem	**RecurNPU**
F+S	Target: DeepSeek-v3-671B	Source: Llama	400 W		
1	78.6	244	**62.9**	193	77.9
2	78.6	183	241	**63.9**	**63.9**
3	78.6	154	156	**62.9**	68.5
4	78.6	**63.2**	135	**62.9**	**62.9**
5	78.6	63.9	78.3	**62.9**	63.9
6	78.6	244	234	**63.2**	**63.2**
F+S	Target: Llama3.1-405B	Source: DeepSeek	400 W		
1	22.3	**21.9**	24.4	47.1	**21.9**
2	22.3	**21.9**	**21.9**	**21.9**	45.1
3	22.3	26.4	**21.9**	47.8	**21.9**
4	22.3	**21.9**	82.2	**21.9**	**21.9**
5	22.3	**21.9**	**21.9**	**21.9**	**21.9**
6	22.3	26.4	170	**21.9**	**21.9**
F+C	Target: DeepSeek-v2-236B	Source: Llama	300 W		
1	**48.5**	120	140	140	59.6
2	48.5	32.8	93.1	32.5	**31.4**
3	48.5	119	32.5	**31.6**	36.6
4	48.5	33.3	63.7	32.5	**31.2**
5	48.5	32.2	61.3	**31.6**	32.5
6	48.5	**31.2**	124	36.6	**31.2**
F+C	Target: Llama2-13B	Source: DeepSeek	300 W		
1	2.57	**1.35**	4.32	1.65	2.90
2	2.57	1.77	8.81	2.60	**1.35**
3	2.57	1.77	2.48	**1.35**	**1.35**
4	2.57	8.28	16.3	**1.35**	**1.35**
5	2.57	**2.14**	4.37	2.48	4.20
6	2.57	1.77	**1.47**	8.92	2.63
S+C	Target: DeepSeek-v3-671B	Source: DeepSeek	300 W		
1	90.2	156	232	155	**64.9**
2	90.2	246	216	**77.5**	245
3	90.2	74.5	287	**64.9**	**64.9**
4	90.2	68.0	126	65.8	**64.9**
5	90.2	246	162	95.9	**64.9**
6	90.2	246	287	**64.9**	132
S+C	Target: Llama3.1-405B	Source: Llama	300 W		
1	**32.1**	34.4	47.0	254	42.2
2	**32.1**	**32.1**	257	32.4	**32.0**
3	**32.1**	47.0	36.4	36.9	**32.0**
4	**32.1**	**32.0**	68.6	66.4	**32.0**
5	**32.1**	37.0	**32.0**	119	118
6	**32.1**	42.2	254	43.7	33.3
F+S+C	Target: DeepSeek-v3-671B	Source: Llama	300 W		
1	90.2	223	273	**79.4**	287
2	90.2	245	287	80.6	**68.0**
3	90.2	249	154	**68.1**	128
4	90.2	**65.2**	1700	**65.2**	**64.9**
5	90.2	**65.2**	117	**64.9**	**64.9**
6	90.2	246	90.8	68.1	**65.8**
F+S+C	Target: Llama3.1-405B	Source: DeepSeek	300 W		
1	**32.1**	**32.1**	254	**32.0**	42.2
2	**32.1**	69.6	32.8	42.2	35.2
3	**32.1**	34.5	55.7	**32.0**	**32.0**
4	**32.1**	42.2	42.9	155	**32.0**
5	**32.1**	47.0	32.8	117	**32.0**
6	**32.1**	42.3	33.6	42.2	**32.0**

G Additional Analyses

G.1 Comparison with the Same Designer

The main comparison gives Raw-Mem its original Designer prompt, so we also ask whether the result changes when the methods use the same prompt. We compare Raw-Mem, ReME, and RecurNPU with the same Designer and validation feedback on sixteen settings with four sources each. Each method still builds its own source history. RecurNPU has the lowest median normalized TTFT throughout target search (Figure 5). After one evaluation, its median is $0.802\times$, compared with $1.011\times$ for Raw-Mem and $2.227\times$ for ReME; the final medians are $0.720\times$, $0.837\times$, and $1.370\times$, respectively. All three methods find a feasible design on the first evaluation in every setting. We normalize by the original No-Mem median for each target, so this comparison gives the three memory methods a common prompt but does not rerun No-Mem. The curves follow searches with a limit of twelve evaluations and retain the best latency after early stopping.

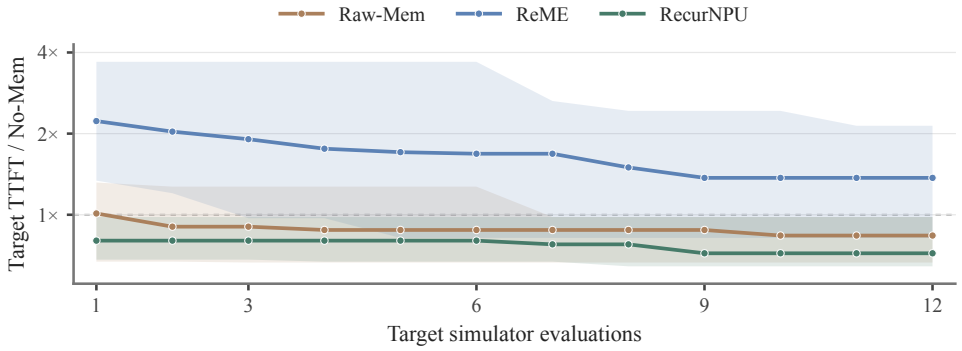


Figure 5: With the same Designer, RecurNPU has the lowest median target TTFT throughout the recorded search. Lines show medians and shading shows interquartile ranges across sixteen settings, normalized by the original No-Mem reference.

G.2 Repeated Searches with the Same Designer

An individual search can find an unusually good or poor design, so we repeat the full sequence of source and target searches. We use the sixteen settings with four sources, which cover all eight benchmark groups and eight target configurations. RecurNPU and Raw-Mem each have five independent executions per setting with the same Designer, feedback, model version (`gpt-5.4-mini-2026-03-17`), and evaluation budgets. Every execution begins with empty experience and searches its four sources in order before searching the target. One previously completed execution per method and setting is included after verifying its prompts, model version, and search protocol; the other four rebuild the entire sequence. We also run No-Mem five times on each of the eight targets with the same Designer. Its median provides a new reference for this experiment.

Across setting medians, RecurNPU lowers geometric mean TTFT by 15.3% relative to Raw-Mem ($0.847\times$). It is faster in 11 settings, tied in 3, and slower in 2, using the same 1% tolerance. Using only the four new executions gives a geometric mean ratio of $0.794\times$, so the observed advantage extends beyond the original execution. Figure 6 shows all five outcomes per method and setting. Negative transfer occurs in 2/80 RecurNPU searches and 17/80 Raw-Mem searches at the 1% tolerance; the difference remains at 5% and 10% (Table 12). Comparing setting medians with the new No-Mem reference gives 0/16 cases for RecurNPU and 3/16 for Raw-Mem. RecurNPU finds a feasible design in 79/80 executions, Raw-Mem in 80/80, and No-Mem in 40/40. The failed RecurNPU search is counted as negative transfer and ranked after feasible designs when computing its setting median. To quantify uncertainty while retaining this failed search, we bootstrap the difference in negative transfer rates. We resample the eight target configurations with both source families kept together, then resample executions within each method and setting and the five No-Mem runs for each target. Across 10,000 draws, the 95% interval for the reduction of 18.8 percentage points is $[2.5, 35.0]$. This interval remains conditional on the four model templates in the subset; the repetitions assess search variability and supplement the full benchmark.

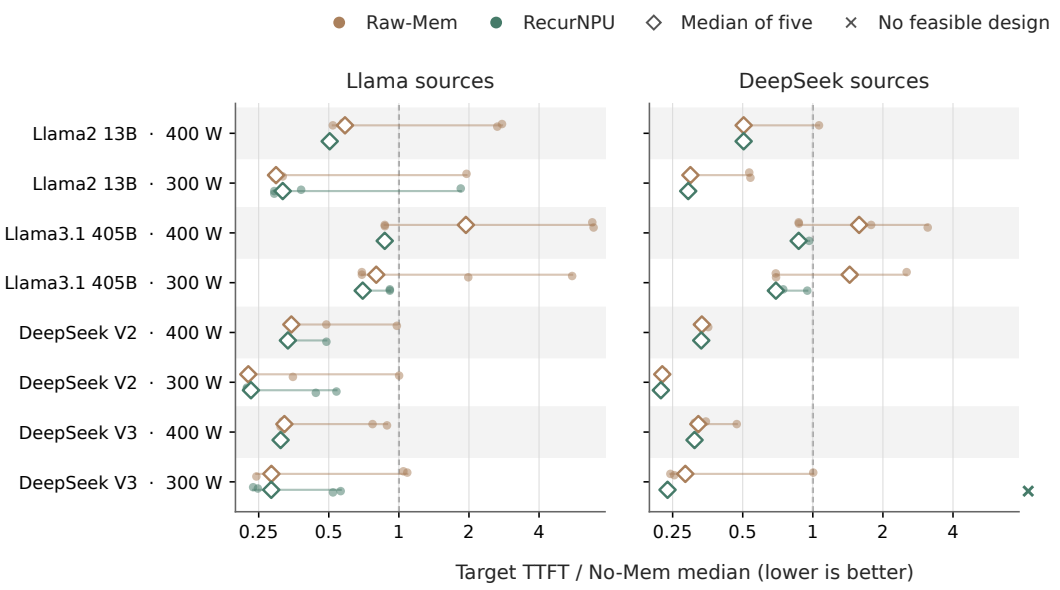


Figure 6: RecurNPU has lower median TTFT in eleven of sixteen settings after independently rebuilding source experience. Dots show individual executions and diamonds their medians; the cross marks a failed search outside the latency axis.

Table 12: RecurNPU has fewer negative transfer outcomes at each tolerance. TTFT ratios use the geometric mean of sixteen setting medians relative to the new No-Mem reference; feasibility and negative transfer count all eighty executions per method.

Method	Feasible runs	TTFT ratio ↓	Negative transfer (runs) ↓		
			1%	5%	10%
Raw-Mem	80/80	0.463×	17/80	16/80	14/80
RecurNPU	79/80	**0.393×**	**2/80**	**2/80**	**2/80**

G.3 Cost of Accumulating and Reusing Experience

Useful experience may reduce target latency, but the searches that produce it and the calls that prepare it for reuse also need to be counted. Recent evaluations of web agents show that including this cost can change comparisons under a fixed inference budget (Hajimiri et al., 2026). We therefore count all source and target calls in the comparison on sixteen settings, including profiling, memory construction, retrieval, adaptation, and proposal attempts. Figure 7 gives mean API costs per setting of $0.553 for Raw-Mem, $0.582 for ReME, and between $0.581 and $0.693 for RecurNPU. The interval for RecurNPU accounts for calls that did not return token usage: reported usage contributes $0.581, and the remaining calls can add at most $0.112 per setting. Each setting includes the full cost of its own four source searches. The estimates use recorded token prices and discounts for cached input; they exclude simulator compute and elapsed search time.

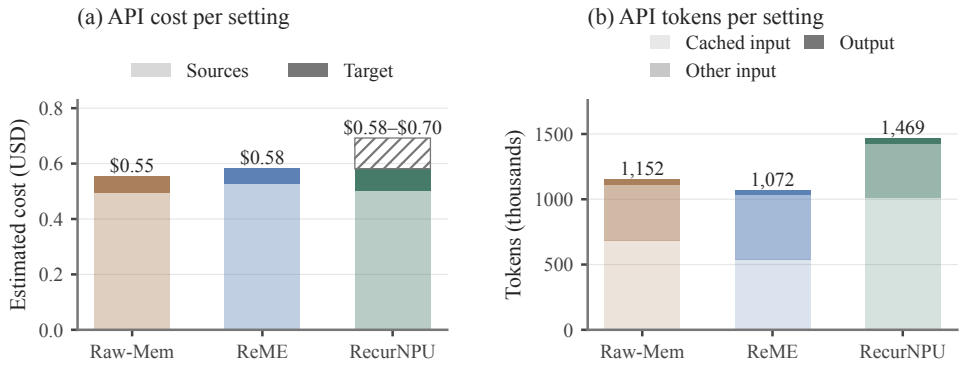


Figure 7: Source searches account for most of the API cost across the sixteen settings. Stacks separate source and target costs (left) and reported token use (right); hatching shows an upper bound on charges for calls without returned usage.

G.4 Transfer Across NPU Simulators

To see whether experience remains useful with a different simulator, we compare RecurNPU and Raw-Mem using LLMCompass (Zhang et al., 2024) and ONNXim (Ham et al., 2024). For each simulator, we use sixteen settings covering eight shift classes and two target families. Each method independently searches two sources before the target, using the same Designer and limits of 30 evaluations per source and 12 per target. LLMCompass evaluates Llama and DeepSeek decoder workloads, while ONNXim runs synthetic dense and gated feedforward templates with native cycle simulation. LLMCompass measures latency across all decoder layers, and ONNXim measures one decoder layer, so the results are compared within each simulator and do not represent complete TTFT. Both methods find feasible designs in every setting (Figure 8). On LLMCompass, the median ratio of RecurNPU latency to Raw-Mem latency is $0.996\times$, with RecurNPU faster in ten settings, tied in one, and slower in five. On ONNXim, the median ratio is $1.000\times$, with three faster results, twelve ties, and one slower result. Against one independent No-Mem search per target, RecurNPU incurs negative transfer in five LLMCompass settings and Raw-Mem in three; on ONNXim, the counts are zero and two. Experience reuse thus produces useful designs with both simulators, although the gains vary and RecurNPU reduces negative transfer relative to Raw-Mem only on ONNXim.

Target latency / Raw-Mem (×) < 1: lower latency > 1: higher latency

	Matched	Family	Scale	Constraint	F + S	F + C	S + C	All three
LLMCompass / Llama	0.988	0.997	1.022	0.434	0.995	1.026	1.101	0.977
LLMCompass / DeepSeek	0.992	0.992	0.926	0.999	1.007	0.948	1.160	1.000
ONNXim / Dense FFN	1.000	1.000	1.003	1.000	1.000	0.819	1.000	0.997
ONNXim / Gated FFN	0.946	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Figure 8: RecurNPU improves latency in some settings on both simulators, with many ties on ONNXim. Cells show RecurNPU / Raw-Mem latency for two target families in each shift class; values below one favor RecurNPU.

G.5 TRANSFER TO A NEW LLM WORKLOAD

We next ask whether the accumulated experience helps with a model outside the Llama and DeepSeek benchmark. We use Qwen2.5-72B (Qwen Team, 2024) in prefill with batch 16, context length 4096, a power budget of 300 W, and 32 chips with TP8 and PP4. Raw-Mem, ReME, and RecurNPU each independently search four Llama sources before the target; No-Mem searches only the target. We keep the Designer, validation feedback, and evaluation budgets unchanged, with no prompt tuning for this workload. RecurNPU reaches 6.47 seconds, compared with 6.88 for Raw-Mem and 13.37 for No-Mem (Table 13). This gives 5.9% lower TTFT than Raw-Mem and a $2.07\times$ speedup over No-Mem. RecurNPU finds its best design on the first target evaluation, while Raw-Mem reaches its best on the fifth. ReME finishes at 45.60 seconds, above the No-Mem reference. This workload provides an example of useful transfer to a new model, with one execution per method.

Table 13: RecurNPU obtains the lowest TTFT on the Qwen2.5-72B target; API costs include each method's complete source history and target search.

	Target latency		Simulator evaluations		API cost
Method	TTFT (s) ↓	/ No-Mem ↓	Source	Target	($)
No-Mem	13.37	$1.00\times$	0	12	0.060
Raw-Mem	6.88	$0.51\times$	120	12	0.515
ReME	45.60	$3.41\times$	112	12	0.744
RecurNPU	**6.47**	$\mathbf{0.48}\times$	85	12	0.614