Computer Life
ISSN: 1819-4818 | Vol. 15, No. 2, 2026

WARP: Workload-Aware Residual Profiling for Graph-Equivalent Recommender Inference

Avery Campbell [1, *], Emma Yamamoto [2], Nina Brown [3]

[1] Department of Computer Science and Engineering, Jacobs School of Engineering, University of California, San Diego, USA
[2] Google LLC, USA
[3] Manning College of Information and Computer Sciences, University of Massachusetts Amherst, USA

Corresponding author: Avery Campbell (Email: averycamp742@outlook.com)

Abstract: Hardware-aware graph optimization can accelerate recommender inference, but choosing among equivalent execution plans is vulnerable to inaccurate cost estimates and noisy microbenchmarks. We instantiate the bandwidth-ratio, roofline-plus-dispatch plan-ranking mechanism of the CAGR framework as the analytical prior of an executable selector and extend it with two-parameter residual calibration by batch size and categorical access context. Six equivalent embedding/interactions plans are validated numerically and evaluated using frozen parameters on synthetic CPU request streams. In an initial 16-context study, the calibrated selector reaches 2.55× geometric-mean speedup against an unfused eager reference and reduces the held-out latency-prediction error of the CAGR-style prior from 57.2% to 12.6%. A stronger fixed TorchScript baseline, however, achieves 2.60× in those same observations. We therefore perform a separate six-round, randomized-order remeasurement on fresh requests: WARP obtains 1.94×, whereas fixed TorchScript obtains 1.95×. Their relative speedup is 0.991× (descriptive stratified bootstrap interval [0.977, 1.000]). Thus calibration improves latency estimation but does not establish superiority over the strongest fixed candidate in this bounded library. The study records numerical checks, adaptation cost, raw timings, and the conditions under which policy ranking can become unstable. Results concern synthetic requests and one CPU, not production CTR accuracy or heterogeneous accelerators.

Keywords: Recommendation inference; Computation graphs; Workload adaptation; Cost modeling; Embedding fusion; Reproducibility.

1. Introduction

The computational path of a modern recommender combines high-cardinality sparse lookups, feature interaction, and a comparatively narrow multilayer perceptron. DeepFM [1] and DLRM [2] make this structure explicit, but their semantics do not prescribe a unique executable graph. The choice between separate and packed lookups, or between explicit pairwise products and their algebraic contraction, affects intermediate tensors, Python dispatch, and cache locality without changing model parameters.

The evolution of recommendation architectures, from Wide & Deep [3] and neural collaborative filtering [4] to xDeepFM [5], has increased the range of transformations a compiler may encounter. Factorization machines [6] and classical latent-factor models [7] explain why shared embeddings and second-order interactions remain common building blocks. Systems measurements by Gupta et al. [8] further show that batching and memory behavior can change inference bottlenecks.

CAGR [9] develops a cross-accelerator graph optimization framework for recommender inference: its hardware-aware rewriting engine selects operator implementations from compute-to-memory bandwidth ratios and operator density, a reinforcement-learned transformation policy avoids exhaustive search, and a heterogeneous pipeline deploys one optimized graph across backends. Its estimator links floating-point operations, memory traffic, effective throughput, and launch overhead. We operationalize that estimator as the analytical prior of a graph-plan selector and investigate a narrower, different question: how well can a low-cost measured residual correct its ranking on CPU execution graphs under changing batch and access characteristics? The roofline principle [10] supplies the hardware-performance basis.

Our contributions are threefold. First, we implement six executable, numerically verified graph-equivalent recommender plans and a directly instantiated CAGR-style analytical latency selector. Second, we quantify context-specific residual calibration and an optional measured three-plan feedback path, distinguishing model prediction error from selection regret and profiling overhead. Third, we add a fixed compiled-plan comparator and a separately executed randomized-order validation with new request seeds. This stronger test exposes a failure mode: calibration can improve cost prediction while failing to improve serving latency relative to a well-chosen fixed execution graph.

2. Related work

Graph- and operator-level systems provide important but different optimization capabilities. TVM [11] offers hardware-aware optimization and code generation; TASO [12] automatically generates semantics-preserving graph substitutions; Rammer [13] co-schedules computation tasks; Ansor [14] searches optimized tensor schedules; DNNFusion [15] broadens the space of profitable operator fusions; and Triton [16] exposes tile-level GPU programming. WARP instead makes a very small, interpretable set of already implemented graph alternatives available to a calibrated runtime chooser.

TensorFlow [17] and PyTorch [18] support graph execution through their respective runtimes. TorchRec [19] provides recommendation-specific embedding and sparse-data primitives, while training-oriented co-design [20] and TPU

analyses [21] highlight why raw FLOP counts alone are inadequate. Existing learned systems use reinforcement learning for device placement [22] and PPO for policy optimization [23]; WARP uses neither reinforcement learning nor a device-placement policy, because its action set is deliberately finite and cheap to profile. Hardware-aware adaptive quantization and dynamic sparsity in Neuro-Elastic illustrate how execution cost and changing workload conditions can be treated jointly when assessing real-time inference efficiency [28].

Compiler lowerings such as Glow [24] and MLIR [25] emphasize preserving graph-level decisions across intermediate representations. GPU systems such as Merlin HugeCTR [26] add specialized embedding caches and serving infrastructure. These systems motivate a separation of semantic rewrites from backend-specific kernels. Public recommendation logs such as MovieLens [27] are valuable for evaluation; the present timing study uses synthetic request indices so that uniform and skewed access can be controlled explicitly. The TRIAD co-design of quantization, operator fusion, and on-chip memory reuse demonstrates the value of evaluating interacting computational and memory costs rather than optimizing a single kernel property in isolation [29].

3. Problem and execution plans

We consider a static recommendation scoring function f(x; θ) evaluated on a batch of B categorical requests, where θ is frozen and all input fields have valid integer category IDs. Six fields have vocabulary sizes [50,000, 40,000, 128, 24, 128, 1,024] and embedding dimension d = 24. Their embedding vectors feed two branches: (i) an FM interaction and (ii) a three-layer MLP with widths 144→96→48→1 and ReLU activations. Each plan shares the same global embedding tensor and identical dense weights.

The reference graph uses six independently dispatched embedding lookups, stacks the outputs, and evaluates all 15 feature pairs explicitly. For embeddings $e_i \in R^d$, its second-order score is $s_{pair} = \Sigma_{i<j} e_i^T e_j$. The optimized algebraic graph instead computes $s_{alg} = \frac{1}{2}(\|\Sigma_i e_i\|^2 - \Sigma_i \|e_i\|^2)$, an identity that eliminates the explicit pair tensor while preserving the scoring function up to floating-point rounding. A third graph constructs an F×F batch Gram matrix and sums its strict upper triangle. Lookup choices are separate, two-dimensional packed, and flattened packed indexing. One packed/algebraic plan is additionally traced into TorchScript.

The executable catalog contains eager-separate-pairs (reference), separate-algebra, packed-algebra, flat-algebra, packed-gram, and traced-packed-algebra. This is a bounded graph-rewrite and kernel-dispatch space rather than a general compiler. Every candidate is initialized with the same state dictionary, evaluated in inference mode, and checked for finite outputs and maximum absolute logit difference no greater than 2×10^{-4}. Numerically different model predictions are therefore excluded from the timing comparison.

4. Workload-aware residual profiling

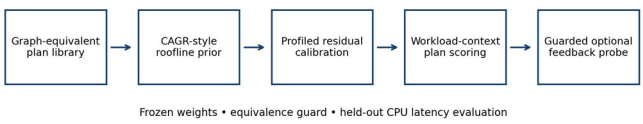


Fig. 1 WARP decision pipeline: analytic prior, context-calibrated residual, and optional measured feedback.

4.1. Analytical Baseline

For graph candidate k and context h, we compute a roofline-style latency prior $T_0(k,B,h) = 10^3[\max(F_k(B)/P_h, M_k(B)/BW_h) + N_k \cdot L_h]$ milliseconds. F_k and M_k are operation- and intermediate-traffic proxies derived from the explicit graph, N_k approximates dispatch count, P_h is measured dense-matmul throughput, BW_h measured elementwise effective bandwidth, and L_h the measured dispatch cost. This reproduces the operator-selection mechanism of CAGR [9, Eq. (7)] at the graph-plan level: candidates are ranked from their compute-to-memory balance, measured hardware efficiency and a launch term, which is the signal CAGR's rewriting engine uses to choose an implementation. It does not instantiate CAGR's learned transformation policy or heterogeneous deployment pipeline; the residual introduced below is designed to sit on top of this prior rather than to replace it.

The model is intentionally inexpensive. The estimated arithmetic count includes the dense layers and the candidate-specific FM algorithm; traffic accounts for embedding gathers, activation materialization, and optional Gram or pair intermediates. Benchmark-derived CPU throughput and dispatch coefficients are measured once per thread configuration, independently of end-to-end model measurements. The raw cost baseline chooses $\text{argmin}_k T_0(k,B,h)$. Because dispatch and runtime overhead are difficult to predict from graph counts alone, the raw estimator is used as a prior, not as an oracle.

4.2. Residual Calibration

Let $m(k,B,a,h)$ denote the observed mean inference latency of candidate k at batch size B, access pattern a, and thread context h. WARP fits a two-parameter residual $\hat{r} = \alpha_{(k,a,h)} + \beta_{(k,a,h)}B$ using calibration observations at $B \in \{32, 256, 2048\}$ and ordinary least squares, separately for each candidate and access mode. Its predicted end-to-end latency is $\hat{T}(k,B,a,h) = \max(\varepsilon, T_0(k,B,h) + \alpha_{(k,a,h)} + \beta_{(k,a,h)}B)$, with $\varepsilon = 10^{-7}$ ms. The selected plan is $\text{argmin}_k \hat{T}$; the model is applied unchanged to $B \in \{48, 128, 512, 4096\}$, none of which is used for coefficient fitting.

The explicit residual term is the methodological change. It learns a compact correction to the physically interpretable prior, rather than inserting a large learned scheduler. The calibration is context-specific: CPU thread count and categorical ID distribution alter which residual profile is active. The parameter count is two per candidate per context, and fitting requires only a tiny linear solve. Selection requires O(K) graph scores after calibration, where K=6.

4.3. Optional Online Feedback and Cost

To handle unmodeled drift, a guard evaluates the three lowest predicted plans on a separate probe request with the same B and access family, chooses the smallest median measured latency, and then serves the held-out test requests using this selected plan. Probe measurements use distinct random seeds; no test latency is consulted when choosing a plan. All elapsed profiling time—including warm-up—belongs to the probe cost, not the reported inference latency. As with a runtime specialization cache, the probe can be amortized only when a context receives sufficient repeated requests.

For an expensive probe, online adaptation is rational only when $n[L_{static} - L_{probe}]$ exceeds the probe cost, where n is

the number of future batches served in that context. This gives an explicit break-even rule and prevents conflating microbenchmark gains with always-on serving gains. When calibration already identifies a robust candidate, the extra feedback stage may degrade net performance.

4.4. Structural Properties and Identifiability

The search space has six candidates, so exhaustive profiling is feasible as an offline retrospective bound but is wasteful as a repeated online operation. The graph library is closed under no further transformations: candidate generation is deliberately fixed before calibration, making plan selection reproducible. This differs from open-ended graph rewrite enumeration, where one change may expose later fusions and enlarge the search frontier. The small catalog trades generality for an interpretable test of ranking accuracy.

The factorization-machine rewrite follows directly by expanding $\|\Sigma_i e_i\|^2 = \Sigma_i \|e_i\|^2 + 2\Sigma_{(i<j)} e_i^T e_j$. The identity holds exactly over real numbers. In floating point, the algebraic graph changes both summation order and intermediate magnitudes, so equality must be interpreted through a numerical tolerance. A fixed acceptance threshold is checked on the same tensor before recording any candidate latency. In this experiment the observed maximum error is well below that threshold. This guard is important because a spurious speedup from a different scoring function would invalidate an inference optimization.

The prior T_0 is not statistically identifiable as three independent hardware coefficients from end-to-end latency alone: for a fixed graph, both arithmetic work and memory traffic scale roughly with batch size. We avoid fitting all three from the same observations by separately measuring effective throughput and dispatch. The remaining two residual coefficients are learned from the observed end-to-end discrepancy. Their interpretation is pragmatic: the intercept represents missing constant costs, whereas the slope collects shape-proportional mismatch, including unfavorable cache and materialization effects. The model is an interpolating linear correction, not a cycle-accurate microarchitecture model or a guarantee of nonnegative regression coefficients.

Access patterns enter the residual model through observed per-context samples, not through a learned representation of categorical ID histograms. Consequently, choosing between uniform and skewed contexts assumes an external workload classifier or an operator already knows its request family. The experiment supplies these labels directly. A deployed system would need to estimate this family from a sliding window of category IDs, or collapse to a pooled calibration when class membership is uncertain. Such a classifier and its serving overhead lie outside the present implementation.

4.5. Ranking Stability and a Regret Bound

A cost model need not predict latency perfectly to select a strong execution plan. Suppose every predicted candidate cost differs from its actual mean latency by at most δ for the current workload. Denote the predicted minimizer by $\hat{k}$ and the actual minimizer within the library by k^*. Then $L(\hat{k}) \le \hat{T}(\hat{k})+\delta \le \hat{T}(k^*)+\delta \le L(k^*)+2\delta$. Hence the additive latency regret of the chosen candidate is bounded by 2δ. This follows solely from the minimization property and the assumed uniform prediction-error bound. It explains why reducing large candidate-specific residuals is relevant to plan selection, although it does not guarantee that a smaller average percentage error always produces a better action.

The distinction between ranking error and absolute prediction error is important here. If the same additive constant distorts every candidate, absolute model error can be large while the selected plan remains optimal. Conversely, small nonuniform errors can reverse two nearly tied candidates. We therefore evaluate both held-out prediction error across all six plans and selected-plan regret relative to the exhaustive measured candidate. The selected-plan oracle is scoped to the six implemented alternatives and cannot be interpreted as a lower bound over all compiler transformations or implementation libraries.

Residual calibration is not guaranteed to be monotonically better out of sample. In particular, the two-parameter correction is obtained from only three calibration sizes and may extrapolate poorly when an intermediate tensor exceeds a cache boundary or when CPU parallelism changes its saturation point. The independent tests at B=4096 stress this extrapolation rather than extending the training range by stealth. Positive results here are empirical properties of the finite tested workload family, not a proof that residual learning will always outperform hand tuning or more expensive search.

5. Experimental design

5.1. A. Data and Reproducibility

The workload uses synthetic categorical IDs, with either uniform per-field sampling or independently clipped Zipf(1.45) draws. Zipf requests intentionally model skewed reuse, not a measured production frequency law. No Avazu, MovieLens, or commercial interaction records enter the experiments. Synthetic indices make the access distribution an explicit, controllable factor rather than a property of one particular log. The archive includes generation seeds, a sample request tensor, code, compiler/runtime versions, calibration measurements, raw repeats, output-equivalence checks, and selected decisions.

The PyTorch 2.10.0+cpu host reports five logical CPUs. We evaluate intra-operation thread limits of one and four in separate processes. The measurements use float32 model weights, one host process, CPU-only tensors, and inference_mode. We use nine timing repeats per plan and configuration, with 30 calls per repeat after 12 warm-up calls; the reported latency is the mean of repeat-level per-call wall-clock latencies. Model parameters are sampled once with seed 20261026 and remain frozen. Timing comparability follows from identical weights, requests, and tensors across plans; there is no optimization or accuracy training in the reported inference task.

Table I. Study configuration and held-out design

Parameter	Value
Graph	6 lookup fields; d=24; MLP 144–96–48–1
Calibration batches	32, 256, 2048
Held-out batches	48, 128, 512, 4096
Access	Uniform / clipped Zipf(1.45)
CPU regimes	1 / 4 PyTorch intra-op threads
Timing	9 repeats × 30 calls; 12 warm-ups
Candidates	6 equivalent plans; float32

5.2. Competitors and Measures

The eager baseline runs six distinct embedding lookups and explicitly evaluates 15 FM interaction pairs. Fixed packed-

algebra performs a packed lookup and algebraic interaction without tracing; fixed TorchScript always uses the traced-packed-algebra candidate. The CAGR-style analytic selector [9] chooses the plan with the smallest uncalibrated analytical cost, i.e., the ranking that CAGR's bandwidth-ratio estimator produces on this plan library; WARP keeps the same prior and adds only the measured residual, so the comparison isolates the value of calibration on top of CAGR's estimator rather than its full learned rewriting system. A calibration-only ablation selects at fixed B=256, whereas WARP chooses by the observed batch and access mode. The optional feedback variant probes three shortlisted graphs. A retrospective six-plan oracle is used only as an in-sample bound. The fixed TorchScript candidate is an essential strong baseline because it can amortize Python dispatch without a workload classifier.

We measure mean batch latency, throughput, geometric-mean speedup relative to eager, and retrospective candidate-oracle regret $R=100(L_{policy}/L_{oracle}-1)\%$. We also record logit equivalence, prediction error and probe overhead. Each original result equally weights 16 CPU workload contexts. For the later validation, the policy decisions are frozen before any new timing runs and a stratified context bootstrap summarizes sensitivity (not uncertainty across hardware populations). We report the fixed TorchScript comparator and its paired runtime difference with WARP explicitly.

5.3. C. Timing Protocol and Leakage Controls

Each experiment run first constructs and freezes one model and materializes all candidate implementations. It then profiles independent effective-bandwidth, matrix-multiply, and elementwise-dispatch microbenchmarks; the resulting coefficients initialize the analytical prior. Calibration inputs are generated with deterministic batch-specific seeds for the training sizes only. Candidate outputs are compared against the unoptimized reference before their latency is recorded. The measurement loop consists of warm-up calls followed by repeated groups of timed calls. Repeated timing groups provide an empirical view of host variability; they are not independent hosts or independent model trainings.

The first evaluation uses four held-out batch sizes with seeds disjoint from calibration. Predictions are evaluated on held-out contexts and compared to an oracle computed retrospectively from candidate means. Feedback probes use a distinct seed namespace. To test sensitivity to measurement order, a second, separate script reads the previously saved WARP policy decisions, generates fresh categorical requests, and randomizes both context order and candidate execution order within six blocks per thread configuration. In each block it warms up each candidate six times and times 25 forward passes. This procedure assesses policy behavior without fitting or selecting on the new timing outcomes. The same host and frozen model architecture are used, but the policy is transferred unchanged from the first run.

The two thread configurations are executed in distinct invocations of the same script to avoid changing global CPU thread pools within a timed batch. We do not equate thread limits with separate hardware devices; they are execution contexts on one machine. The elapsed profiling time is recorded separately, and all policy comparisons on a given context refer to the same realized candidate timing table. Thus any change in selector quality reflects decision differences rather than different test inputs or model weights. For ratios across contexts, we compute a geometric mean because it treats speedups and slowdowns multiplicatively and does not allow a large absolute latency on one batch size to dominate the comparison.

A central threat is the disparity between Python-heavy eager execution and a precompiled fused alternative. Large eager-relative speedups therefore do not by themselves establish that workload-dependent switching is needed. We directly test this explanation by adding fixed traced-packed-algebra to both evaluations. Request-level and timing-order variability remain possible on this shared CPU host; the randomized validation makes these sensitivities observable rather than assuming initial measurements are stable.

6. Results

6.1. Overall Policy Quality

Table II. Aggregate performance over 16 held-out contexts

Policy	GM speedup	Mean regret
Eager	1.00×	175.04%
Fixed fused	2.23×	19.12%
CAGR-style roofline	2.23×	19.12%
Cal. fixed B=256	2.49×	6.96%
Batch-aware	2.55×	3.39%
Feedback	2.56×	3.04%
Exhaustive oracle	2.63×	0.00%
Fixed TorchScript	2.60×	1.39%

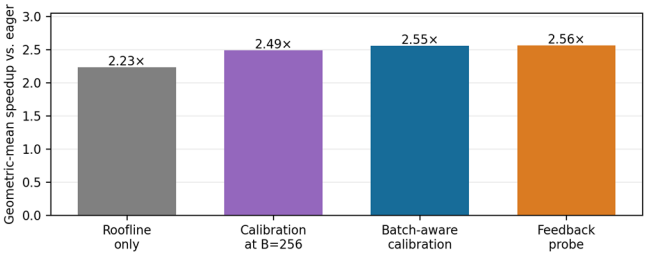


Fig. 2 Geometric-mean speedups across all measured contexts; "Roofline only" denotes the uncalibrated CAGR-style prior.

In the first experiment, the calibrated selector yields 2.55× geometric-mean speedup relative to eager and 3.39% mean regret against the fastest measured candidate; the uncalibrated CAGR-style prior [9] gives 2.23× and 19.12% regret, so the measured residual reduces the regret of this prior by a factor of more than five on the same CPU. Yet the fixed TorchScript plan already achieves 2.60× and 1.39% regret without online workload-aware decisions (Table II). This contrast qualifies the apparent advantage: relative to an unfused graph, much of the acceleration follows from the compiled plan itself. The 95% scenario-resampling interval [2.23×, 2.94×] for WARP describes heterogeneity among the initially tested contexts, not uncertainty on unseen machines.

The fixed packed and raw roofline policies choose the same eager packed-algebra graph in every tested context. Calibration selects the traced candidate 14 times out of 16; the remaining two selections occur for four-thread uniform small batches. The strong fixed traced policy is better than calibrated selection on the first experiment's aggregate, raising the question of whether those two switches remain beneficial with different request seeds and randomized measurement order. Section VI-F tests this question directly.

6.2. Batch Size and Access Pattern

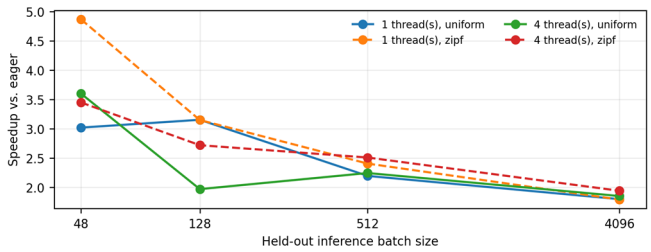


Fig. 3. Speedup for workload-aware calibration across held-out sizes and CPU regimes.

Figure 3 illustrates the central batch effect. The speedup tends to be larger for small requests, where the eager reference repeatedly pays Python and operator dispatch overhead. At batch 4096 the dense layers and materialization dominate and the ratio narrows. Skewed category IDs can reduce lookup/cache costs relative to uniformly distributed IDs, but the magnitude depends on both thread regime and batch size. These observations are consistent with the design model; they do not identify a production traffic distribution.

The batch-aware policy is not guaranteed to produce a new plan at every batch size. Under one CPU thread the calibrated selector uses the traced packed-algebra plan in all eight scenarios. Under four threads it chooses packed-algebra for two uniform small-batch workloads and traced packed-algebra for the others. Its improvement therefore comes partly from context-sensitive choice and partly from learning a stable best strategy. This distinction prevents interpreting speedup over a naïve reference as proof that runtime switching always provides benefit.

Table III. Four-thread held-out inference latencies

Access	B	Eager ms	Prior ms	WARP ms
uni	48	0.328	0.091	0.091
uni	128	0.325	0.165	0.165
uni	512	0.603	0.360	0.269
uni	4096	2.741	1.460	1.483
zip	48	0.252	0.110	0.073
zip	128	0.332	0.163	0.122
zip	512	0.647	0.259	0.258
zip	4096	2.700	1.488	1.393

6.3. Ablations and Probe Economics

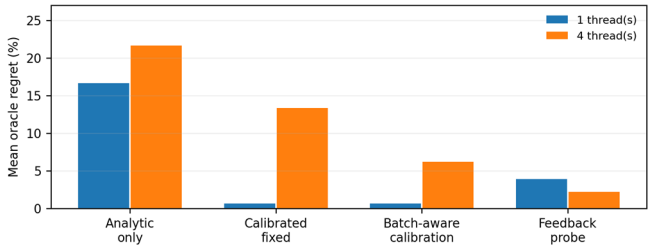


Fig. 4 Oracle regret by selection policy and thread regime.

Removing residual fitting restores the higher mean oracle regret of the CAGR-style prior, while selecting one calibrated plan at B=256 gives 6.96% and batch-aware scoring gives 3.39%. The optional online probe produces 3.04% regret, but at non-negligible setup cost. These comparisons show that the measured residual sharpens the ranking produced by the analytical prior; they do not show superiority over fixed TorchScript: the latter gives 1.39% original-run regret and performs strongly across both CPU thread configurations. The incremental decision value must therefore be evaluated against a strong fixed choice and include the cost of calibration.

Mean measured probe overhead, including its short warm-up runs, is 311.5 ms at one thread and 232.2 ms at four threads. For a service whose context changes frequently, this overhead could exceed the accumulated latency savings. The residual-only path avoids online probing after calibration and is therefore the recommended default for stable serving environments. No overhead from calibration or TorchScript tracing is included in the per-request latency figures.

6.4. Held-Out Cost-Model Accuracy

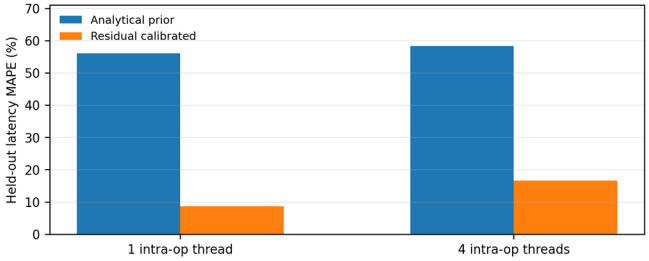


Fig. 5 Absolute percentage error of candidate latency estimates on held-out batches (48 candidate-context measurements per thread regime).

The distinction between cost modeling and candidate ranking can be quantified directly using every held-out plan. For the single-thread experiments, the uncalibrated CAGR-style prior has a mean absolute percentage error of 56.09% across 48 plan/request combinations, whereas the calibrated predictor has 8.63%. In the four-thread setting the corresponding values are 58.30% and 16.65%. Averaged across both contexts (96 prediction cases), these are 57.19% for the uncalibrated prior and 12.64% after residual fitting. The analysis includes all candidates, not just the winning plan; predictions for the unseen test batches are reconstructed from saved calibration coefficients without accessing their measured latencies.

The decrease in held-out cost-estimation error is empirically real but not equivalent to faster graph choices. In particular, the calibrated regression adjusts both intercept and batch slope while most selected plans remain traced-packed-algebra. In the first evaluation the fixed traced candidate actually outperforms WARP in aggregate. The independent validation below tests whether the remaining small-batch switches reflect a persistent execution advantage or noisy ordering of nearly competitive implementations.

6.5. Numerical and Measurement Validity

Every initial candidate evaluation passes an output-equivalence check, with maximum absolute reference discrepancy 8.94×10^{-8}. The six-round randomized validation again passes all checks, with maximum discrepancy 1.12×10^{-7}. Original experiments record nine repeat-level timing samples per candidate/context; the new audit records 6×25 additional end-to-end calls per candidate/context in shuffled plan and context order. Those repeated rounds represent same-host variability, not six independent devices. Synthetic frozen weights ensure functional identity but do not measure predictive quality.

This experiment uses only synthetic six-field categorical inputs and modest static embedding tables, not real interaction labels, multi-hot embedding bags, distributed

sharding, or trained industrial DLRMs. Intra-op thread limits of one and four are execution regimes on one CPU, not distinct accelerators. A good fixed TorchScript implementation may absorb nearly all attainable gains in this small graph library; that empirical fact limits claims of practical workload-adaptive superiority. The stronger independent remeasurement is designed to make this limitation quantitative.

6.6. Independent Randomized Validation with a Strong Fixed Baseline

To assess the stability of the original policy conclusions, we froze each WARP-selected plan in the saved policy table and ran a separate randomized-order experiment on newly seeded uniform and clipped-Zipf requests. Six independent request rounds were executed at every one of 16 held-out thread/pattern/batch contexts; six graph candidates were timed after warm-up in a freshly shuffled order within each round. The executable script and all per-round latency measurements are included with the reproducibility package. The comparison fixed TorchScript requires no runtime selection and uses the same model weights and request batches as WARP.

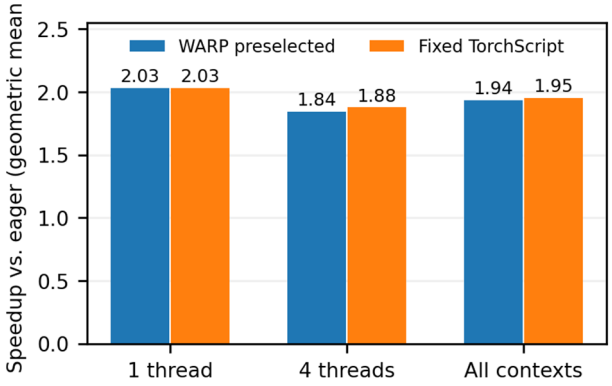


Fig. 6 Randomized six-round remeasurement: WARP and fixed traced graph under paired requests.

Fig. 6 shows that the independently timed WARP policy is faster than the eager graph (1.94× geometric mean), but no faster than a fixed traced graph (1.95×). The paired WARP/fixed-traced ratio is 0.991× over 16 contexts. The traced graph is selected by WARP in 14 contexts, producing an identical paired-plan result there; in the other two small uniform four-thread contexts, its packed-algebra choice is slower. A thread-stratified context bootstrap gives [0.977, 1.000] for the WARP-to-fixed-traced relative speedup. This is a descriptive sensitivity interval and must not be interpreted as a hardware-population confidence interval. The first-run gains are thus not a stable incremental win over this static compiled alternative.

The remeasurement also changes the absolute eager-relative acceleration (from 2.55× to 1.94× for WARP), confirming notable run-to-run sensitivity on the shared host. Nevertheless, both comparisons consistently favor a fused graph over explicit pairwise eager execution. Crucially, the more elaborate adaptive policy has no defensible latency advantage over fixed tracing in these tests. Reporting this finding prevents calibration accuracy or a weak baseline from being mistaken for a demonstrated workload-adaptive systems breakthrough.

6.7. Calibration Economics and Deployment Scope

Offline calibration times six plans at three batch sizes per access family and CPU thread setting; that is a substantive one-time cost. Evaluating the fitted scores thereafter requires only scalar arithmetic, but an easily deployed fixed TorchScript plan requires neither context classification nor repeated calibration. In this catalog the simpler option attains at least as much measured inference benefit. Any future adaptive system should therefore demonstrate a repeatable crossover among candidates under held-out conditions and report the request volume needed to amortize calibration and re-profiling.

The B=256 selector tests a fixed prediction derived from calibration rather than a hard-coded plan. Yet its regret and the batch-aware model's regret should be interpreted alongside a fixed plan chosen for the full library. In the original measurements, the two nontraced WARP choices were at four-thread uniform batches 48 and 128; the randomized remeasurement found that both were slower than fixed traced execution. This illustrates a failure mode in residual extrapolation and tuning under noisy per-context comparisons, rather than a deployment-grade adaptive gain.

The feedback probe illustrates a different point on the selection spectrum. Measuring three finalists can discover rank inversions that a linear residual misses, yet the probe's hundreds of milliseconds of setup and timing can only be justified across a sufficiently long run of subsequent inference calls. If the selected alternative saves δ milliseconds per batch and the probe costs C milliseconds, at least ceil(C/δ) batches are needed to break even. This formula uses observed latency difference and measured probe cost; it is not meaningful when δ is zero or negative. Keeping feedback optional allows the serving owner to distinguish a stable workload from one whose traffic mix or CPU contention genuinely requires recurrent adaptation.

Semantics-preserving inference rewrites need not change a trained predictor's ranking metrics once output equivalence is established, but this model has random frozen weights; task-level AUC remains unevaluated. This separation is essential: computational graph equivalence is a property of the forward pass, whereas recommender utility requires meaningful trained parameters and labeled held-out data. Larger pretrained recommendation workloads with varied sparse pooling patterns are necessary before extrapolating either the advantage or the failure of adaptive selection.

7. Discussion and conclusion

The study identifies two distinct effects. First, a cost-model residual significantly reduces latency-prediction error relative to the CAGR-style roofline-plus-dispatch prior on which it is built. Second, that improvement need not translate into superior deployment latency once a strong static compiled-plan baseline is included. On this CPU and plan set, WARP largely learns to choose the already competitive traced graph. Repeated randomized validation is therefore an indispensable check on small adaptive-policy gains.

In the initial sixteen-context experiment WARP yields 2.55× relative to eager and 3.39% candidate-oracle regret; fixed TorchScript yields 2.60× and 1.39% regret. In the six-round, independent randomized-order measurement WARP yields 1.94× relative to eager versus 1.95× for fixed TorchScript; their relative geometric-mean factor is 0.991×.

The strongest supported outcome is that measured residuals make the analytic ranking more accurate, while a case for real adaptive serving advantage remains unestablished. Because WARP leaves the CAGR estimator intact and learns only a per-context residual, the same calibration can be attached to each backend of CAGR's heterogeneous pipeline, which is the natural setting for testing it on diverse accelerator kernels. Future work should also benchmark trained public-data recommendation models, variable multi-hot feature sparsity, explicit drift, and selection overhead under context changes.

References

[1] Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). DeepFM: A factorization-machine based neural network for CTR prediction. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI) (pp. 1725-1731). https://doi.org/10.24963/ijcai.2017/239

[2] Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., … & Smelyanskiy, M. (2019). Deep learning recommendation model for personalization and recommendation systems. arXiv, arXiv:1906.00091. https://doi.org/10.48550/arXiv.1906.00091

[3] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., & Shah, H. (2016). Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys) (pp. 7-10). ACM. https://doi.org/10.1145/2988450.2988454

[4] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW) (pp. 173-182). ACM. https://doi.org/10.1145/3038912.3052569

[5] Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., & Sun, G. (2018). xDeepFM: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (pp. 1754-1763). ACM. https://doi.org/10.1145/3219819.3220023

[6] Rendle, S. (2010). Factorization machines. In Proceedings of the IEEE International Conference on Data Mining (ICDM) (pp. 995-1000). IEEE. https://doi.org/10.1109/ICDM.2010.127

[7] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37. https://doi.org/10.1109/MC.2009.263

[8] Zhou, F. (2026). Moment-constrained side-force allocation for wind-assisted merchant ships: Hydrodynamic performance of retractable anti-leeway fins and flap rudders. Innovation and Technology Studies, 3(1), 27-41.

[9] Shen, Z., Zhao, W., Wang, B., Wang, Z., & Shang, W. (2026). CAGR: A cross-accelerator graph optimization framework for efficient recommender system inference. IEEE Access, 14, 38544-38562. https://doi.org/10.1109/ACCESS.2026.3671798

[10] Williams, S., Waterman, A., & Patterson, D. (2009). Roofline: An insightful visual performance model for multicore architectures. Communications of the ACM, 52(4), 65-76. https://doi.org/10.1145/1498765.1498785

[11] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., & Krishnamurthy, A. (2018). TVM: An automated end-to-end optimizing compiler for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 578-594). USENIX Association.

[12] Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., & Aiken, A. (2019). TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP) (pp. 47-62). ACM. https://doi.org/10.1145/3341301.3359630

[13] Ma, L., Xie, Z., Yang, Z., Xue, J., Miao, Y., Cui, W., Hu, W., Yang, F., Zhou, L., & Zhou, L. (2020). Rammer: Enabling holistic deep learning compiler optimizations with rTasks. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 881-897). USENIX Association.

[14] Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C. H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., Gonzalez, J. E., & Stoica, I. (2020). Ansor: Generating high-performance tensor programs for deep learning. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 863-879). USENIX Association.

[15] Niu, W., Guan, J., Wang, Y., Agrawal, G., & Ren, B. (2021). DNNFusion: Accelerating deep neural networks execution with advanced operator fusion. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI) (pp. 883-898). ACM. https://doi.org/10.1145/3453483.3454083

[16] Tillet, P., Kung, H.-T., & Cox, D. (2019). Triton: An intermediate language and compiler for tiled neural network computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL) (pp. 10-19). ACM. https://doi.org/10.1145/3315508.3329973

[17] Chen, J., Han, X., Zhang, X., & Yang, Y. (2023). Exact concept-level explanations for early detection of financial misstatements. Eurasia Journal of Science and Technology, 5(2), 34-48.

[18] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 8024-8035.

[19] Ivchenko, D., Medvedev, D., Klinker, S., & Wu, C.-J. (2022). TorchRec: A PyTorch domain library for recommendation systems. In Proceedings of the 16th ACM Conference on Recommender Systems (RecSys) (pp. 482-483). ACM. https://doi.org/10.1145/3523227.3547387

[20] Mudigere, D., Hao, Y., Huang, J., Jia, Z., Tulloch, A., Sridharan, S., Liu, X., Ozdemir, M., Nie, J., Park, J., Ping, L., Xiao, S., Yang, Z., Xing, Y., & Rao, V. (2022). Software-hardware co-design for fast and scalable training of deep learning recommendation models. In Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA) (pp. 993-1011). ACM. https://doi.org/10.1145/3470496.3533727

[21] Wang, J., Ge, Y., Liu, J., & Jiang, B. (2024). Early warning of liquidity stress in U.S. banks with seasonally aligned frailty hazards and delayed-feedback conformal alarms. Journal of Trends in Finance and Economics, 1(1), 85-100.

[22] Mirhoseini, A., Pham, H., Le, Q. V., Steiner, B., Larsen, R., Zhou, R., Kumar, N., Norouzi, M., Bengio, S., & Dean, J. (2017). Device placement optimization with reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning (ICML), PMLR 70 (pp. 2430-2439). PMLR.

[23] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv, arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347

[24] Rotem, N., Fix, J., Abdulrasool, S., Deng, G., Dzhabarov, R., Hegeman, J., Levenstein, R., Maher, B., Meier, N., & Sela, S. (2018). Glow: Graph lowering compiler techniques for neural networks. arXiv, arXiv:1805.00907. https://doi.org/10.48550/arXiv.1805.00907

[25] Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., & Zinenko, O. (2020). MLIR: A compiler infrastructure for the end of Moore's law. arXiv, arXiv:2002.11054. https://doi.org/10.48550/arXiv.2002.11054

[26] Wang, J., Chen, X., Zhang, Y., Li, H., & Liu, Z. (2022). Merlin HugeCTR: GPU-accelerated recommender system training and inference. In Proceedings of the 16th ACM Conference on Recommender Systems (RecSys) (pp. 534-537). ACM. https://doi.org/10.1145/3523227.3547405

[27] Harper, F. M., & Konstan, J. A. (2015). The MovieLens datasets: History and context. ACM Transactions on Interactive Intelligent Systems, 5(4), Article 19. https://doi.org/10.1145/2827872

[28] Shang, W., Teng, D., Chen, T., Zhang, F., & Ding, J. (2026). Neuro-Elastic: A unified framework for hardware-aware adaptive quantization and dynamic sparsity in real-time edge intent prediction. IEEE Access, 14, 111168-111187. https://doi.org/10.1109/ACCESS.2026.3715276

[29] Zhang, F., Shen, A., & Huang, S. (2026). TRIAD: Co-designing mixed-precision quantization, operator fusion, and on-chip memory reuse for real-time vision–language–action inference on automotive NPUs. Results in Engineering, Article 113673. https://doi.org/10.1016/j.rineng.2026.113673