CloserToMe: A Unified Framework for Accurate and Transferable Latency Prediction Across Heterogeneous Devices

Cheng Tang[1], Guochong Sui[1], Wenqi Lou[1,3*], Zihan Wang[1], Jiayi Tuo[1], Wenqian Xie[2], Yinkang Gao[1], Yixuan Zhu[1], Lei Gong[1], Chao Wang[1,3*], Xuehai Zhou[1]

[1]University of Science and Technology of China
[2]Duke University
[3]Suzhou Institute of Advanced Research, University of Science and Technology of China
sisyphustc@mail.ustc.edu.cn, {louwenqi,cswang}@ustc.edu.cn

Abstract

Hardware accelerators such as GPUs, NPUs, and FPGAs are essential to meeting AI's computational demands. With the proliferation of heterogeneous devices across cloud and edge, various model optimization techniques adapt to diverse hardware characteristics through operator transformations and structural modifications. Accurate, efficient latency prediction enables rapid selection of optimal strategies across hardware backends. Many existing methods treat hardware as a black-box executor, directly regressing latency without explicitly modeling the intricate interactions between neural network (NN) structures and device-specific execution behaviors. To address these challenges, we introduce a new modeling perspective that captures the interaction between neural architectures and hardware execution. To capture device-specific characteristics, we propose two complementary modeling strategies. The *Device Behavior Signature Selector* (DBSel) characterizes hardware execution behavior by selectively probing a small set of representative architectures, forming a compact, workload-driven profile. In parallel, we construct capability vectors that capture the hierarchical memory of each device and compute characteristics, providing a structured abstraction of its architectural capacity. To unify both behavioral and structural views, we introduce the *Hardware–Operation Dialogue Module* (HODM), which models fine-grained interactions between neural operators and hardware properties. Together, these components empower CloserToMe to deliver accurate and transferable latency predictions across unseen and diverse platforms.

Introduction

NNs are increasingly deployed on edge and cloud platforms, where inference latency has become a critical consideration (Cai et al. 2020; Lin et al. 2020; Lou et al. 2021; Chen et al. 2024; Geng et al. 2024; Misra, Saoda, and Abdelzaher 2025; Zhou et al. 2025). In such settings, accurate latency prediction plays a pivotal role in guiding architecture selection and workload scheduling strategies (Wang et al. 2024; Khare et al. 2025; Fu et al. 2025). However, the growing diversity of both neural architectures and hardware platforms makes it increasingly challenging to obtain accurate latency measurements at scale, especially for emerging or rarely profiled devices (Xing et al. 2022; Liu et al. 2024; Lv, Zhang, and Wang 2025). This motivates the need for fast, generalizable latency predictors that work across platforms (Lou et al. 2024), including edge and cloud GPUs (e.g., AGX Xavier, NVIDIA V100), FPGAs (e.g., ZC706), and CPUs (e.g., Snapdragon 855).

Early latency prediction relied on proxy metrics like FLOPs (He et al. 2018; Li et al. 2017; Tan et al. 2019; Hanxiao, Karen, and Yiming 2019) to estimate NN latency indirectly. Later works (Cai et al. 2019; Wu et al. 2019; Cai et al. 2020) used operator-level lookup tables, summing individual latencies. nn-Meter (Zhang et al. 2021) predicts kernel-wise latency via regression and aggregates them. However, these methods neglect end-to-end execution paths, failing to capture latency variations from graph-level optimizations and hardware scheduling. With deep learning advances, end-to-end predictors emerged. LAD (Xu et al. 2020) conducts large-scale profiling (e.g., 100K samples), encodes sub-architectures as binary vectors, and uses multilayer regressors. BRP (Dudziak et al. 2020) employs GNNs but relies heavily on device-specific training distributions. Still, such methods are platform-specific and require costly retraining for new or unseen hardware. FLAT (Akhauri and Abdelfattah 2024a) and FLAN (Akhauri and Abdelfattah 2024b) systematically improve predictor transfer via few-shot adaptation and encoding design across devices. However, these methods still do not explicitly model how neural architectures interact with device-specific execution patterns, which can limit their achievable accuracy, especially on unseen devices. In particular, devices are mostly represented as opaque learned identifiers rather than as structured abstractions that capture workload-dependent execution behavior and underlying physical characteristics. Moreover, while many existing latency-aware approaches primarily emphasize latency ranking, we instead aim for accurate absolute latency prediction across heterogeneous devices, which is a more sensitive objective under hardware-specific variability (Dudziak et al. 2020).

These limitations reflect broader shortcomings in existing methods: lack of device-specific execution behavior modeling, weak hardware representation, and the absence of intri-

*Corresponding authors.

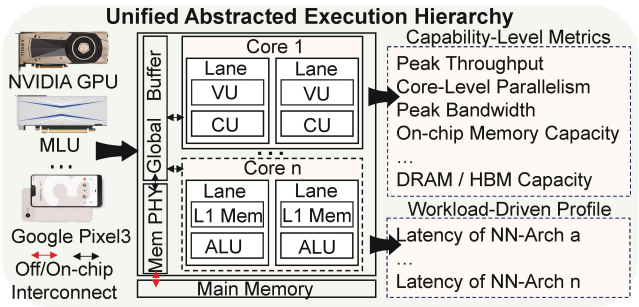

Figure 1: While GPU, MLU, and mobile SoC architectures differ considerably at the microarchitectural level, their key functional components can be mapped to a unified two-layer abstraction of compute and memory.

cate interactions between hardware and device-specific execution behavior hinder generalization to unseen platforms.

To enable accurate prediction across diverse hardware platforms, we jointly model structured device capabilities and interaction patterns between neural architectures and hardware behavior. We first model hardware behavior from a workload-driven perspective, and we propose the *Device Behavior Signature Selector* (DBSel), which constructs a *Device Behavior Signature (DBS)* by probing the latencies of a small set of representative architectures on the target device. This signature serves as a compact and informative descriptor of the device's execution behavior. Meanwhile, we build a structured *Hardware Capability Vector* (HCV), offering a unified, physically grounded representation of the platform's compute and memory capacity. Finally, the *Hardware–Operation Dialogue Module* (HODM) explicitly models the latent co-dependencies between network semantics and hardware traits. Together, these components enable *CloserToMe* to deliver accurate and transferable latency predictions across diverse device–architecture combinations.

The main contributions of this work are as follows:

- **A unified latency prediction framework.** We propose *CloserToMe*, an end-to-end predictor that jointly models neural architectures and hardware features for accurate cross-device latency prediction.
- **Device Behavior Signature Selector (DBSel).** We introduce *DBSel*, a sampling algorithm that constructs compact and representative workload-driven Device Behavior Signature to guide generalizable prediction.
- **Hardware-aware modeling via Hardware Capability Vector (HCV) and Hardware–Operation Dialogue Module (HODM).** We design an HCV and an HODM, enabling cross-modal attention between operators and device traits for robust inference latency modeling.
- **Extensive validation on real-world hardware.** Experiments on standard benchmarks and an extended dataset confirm our method's superior generalization and accuracy compared to state-of-the-art baselines.

Motivation

Due to the wide variability in hardware design, including fabrication technology, compute capacity, and microarchitectural organization, the same neural network can exhibit drastically different latency across devices (Li et al. 2025). This performance-level *platform dependency* makes cross-device latency prediction particularly challenging. Fortunately, recent research in the systems and architecture community suggests that the regularity of neural network computations enables performance to be approximated using a unified abstraction of hardware structure and dataflow. As illustrated in Figure 1, various hardware platforms share a similar hierarchical organization: CPUs typically contain a small number of cores with private L1 caches, while NPUs and GPUs often integrate a large number of parallel cores along with more powerful MAC units. Most platforms also adopt multi-level memory hierarchies, such as L1 and L2 caches. Although recent works (e.g., (Zhang et al. 2024)) have achieved promising simulation accuracy, their reliance on domain-specific hardware knowledge and time-consuming simulation procedures makes them impractical for real-time model feedback.

Leveraging this shared hierarchy, we distill device capabilities into a compact set of physically interpretable metrics. Execution is modeled as a data-processing pipeline over a multi-level compute–memory substrate, with each level summarized by its compute throughput, memory/storage capacity, and cross-level bandwidth. These structured descriptors form the *Hardware Capability Vector* (HCV), offering a unified, device-general interface for latency modeling. However, such structural capability alone cannot capture dynamic behaviors introduced by compiler- and runtime-level optimizations (e.g., tiling, fusion, and scheduling). To reflect how devices execute real workloads, we further go beyond purely structural capability descriptors by introducing a workload-driven profile that captures device–operation interactions beyond static hardware specifications.

Method

In this section, we present the overall architecture of *CloserToMe*, as illustrated in Figure 2. We begin by constructing a *Device Behavior Signature (DBS)* for the unseen (test) device using the *DBSel algorithm*. DBS is then concatenated with the encoded representation of the target neural network. The model representation, comprising the operation-level feature matrix and structural adjacency matrix, is first processed by a Graph Neural Network based on *Dense Graph Flow* and *Graph Attention*. The resulting embeddings are then passed through a second GNN layer to further refine the latency-relevant features from a workload perspective. The output feature tensor X, together with the hardware capability vector, is fed into the *Hardware–Operation Dialogue Module*. Finally, the fused representation from HODM and HCV is passed to an MLP to predict the final latency.

DBSel: Representative Signature Selection

Achieving accurate latency prediction across diverse hardware platforms remains a challenge, largely due to the

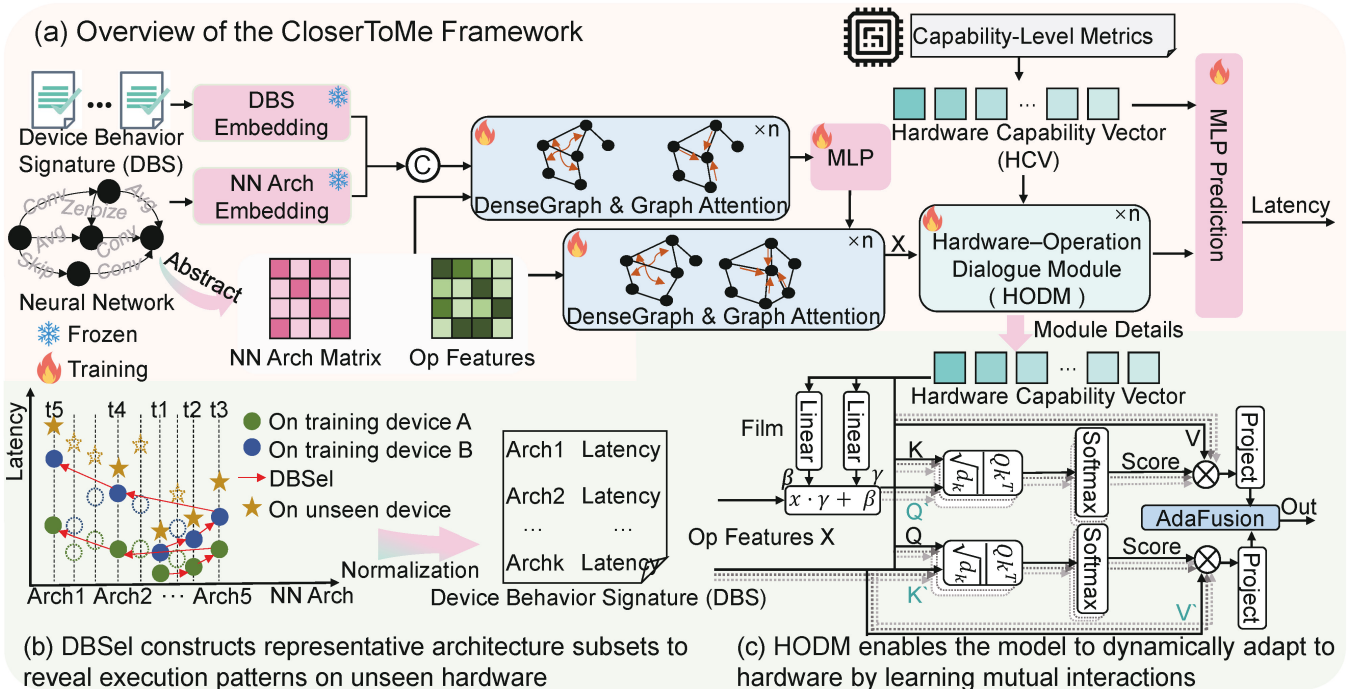

Figure 2: Overview of the proposed framework. (a) illustrates the overall architecture of CloserToMe; (b) depicts how representative architectures are selected from training devices via DBSel to form the Device Behavior Signature (DBS) on a previously unseen target device; (c) details the Hardware–Operation Dialogue Module (HODM).

complex and platform-specific execution behaviors that NN workloads exhibit. Notably, the same NN can demonstrate vastly different latency across devices, driven by variations in scheduling policies, operator fusion mechanisms, and low-level execution pipelines. These discrepancies necessitate representations effectively capturing nuanced hardware–NN workload interactions.

To achieve this, inspired by (Lee et al. 2021), we propose the *Device Behavior Signature Selector* (DBSel), a workload-driven mechanism that constructs hardware representations based on lightweight latency probing. Specifically, DBSel selects a small, representative subset of neural architectures from training sets, and evaluates them on the target platform to form a compact *Device Behavior Signature* (DBS), which captures how the device responds to diverse neural operations and enables more accurate, sample-efficient latency prediction. DBSel incorporates three key components: device-level similarity modeling, temperature-scaled weight initialization, and diversity–coverage-aware sampling. These modules collectively guarantee that the selected architectures capture salient execution patterns and remain representative across diverse hardware backends.

$$W_{ij} = \frac{\exp\left(\alpha_{\text{sim}} \cdot \frac{\mathbf{S}_{ij}-\mu_j}{\sigma_j+\epsilon}/\tau\right)}{\sum_{i'=1}^{D_{\text{train}}} \exp\left(\alpha_{\text{sim}} \cdot \frac{\mathbf{S}_{i'j}-\mu_j}{\sigma_j+\epsilon}/\tau\right)} \tag{1}$$

In Eq. (1), $\mathbf{S_{ij}}$ denotes a hybrid similarity score between training device i and test device j, computed by combining cosine similarity and Pearson correlation over their probe-based latency data. To normalize these scores, we apply z-score standardization across training devices using the per-test-device mean μ_j and standard deviation σ_j. The standardized similarities are then scaled by a factor α_{sim} and passed through a temperature-controlled softmax, resulting in an alignment weight W_{ij} for each device pair. We average W_{ij} across all test devices and interpolate with a uniform prior to obtain a final device weight w_i. These weights guide representative architecture selection by emphasizing architectures that show high variance on similar devices, while promoting broad latency-space coverage.

$$\text{DiversityCoverageScore}(i) = \gamma \cdot \sum_{d=1}^{D_{\text{train}}} w_d \left(L_{id} - \bar{L}_i\right)^2 + (1-\gamma) \cdot \frac{1}{1 + \left|\bar{L}_i - \text{median}(\bar{\mathbf{L}})\right|} \tag{2}$$

In Eq. (2), we define a hybrid scoring function to select the first representative architecture by balancing inter-device diversity and global representativeness. Lid is the latency of architecture i on training device d, and $\bar{L}i = \frac{1}{D_{\text{train}}}\sum_d L_{id}$ denotes its mean latency across devices. $\bar{\mathbf{L}}$ is the vector of $\bar{L}_i$ for all architectures. w_d is the similarity-derived importance weight of device d, and $\gamma \in [0,1]$ is a tunable coefficient controlling the trade-off between diversity and representativeness. The first term favors architectures with high latency variance across similar devices, while the second term encourages the selection of candidates whose average latency lies near the global median, thus avoiding extreme outliers

Algorithm 1: Device Behavior Signature Selector Algorithm

Input: Training latency matrix $\mathbf{L}_{\text{train}} \in \mathbb{R}^{N \times D_{\text{train}}}$;
Number of probe models p, selected models k;
Distance metric: hybrid; Probe mode: random;
Hyperparameters: τ, α_{sim}, β, α_{cov}
Output: Representative indices $\mathcal{S} = \{s_1, \ldots, s_k\}$ $(k \ll N)$
$\mathcal{P} = \text{RandomSample}(\{1, 2, \ldots, N\},\ p)$
// Device Similarity Computation
$\mathbf{P}_{\text{train}} = \mathbf{L}_{\text{train}}[\mathcal{P}, :]^\top \in \mathbb{R}^{D_{\text{train}} \times p}$
$\mathbf{P}_{\text{test}} = (\mathbf{L}_{\text{test}}^{\text{probe}})^\top \in \mathbb{R}^{D_{\text{test}} \times p}$
$\mathbf{S}_{ij} = \text{Joint-CosSimPearsonCorr}(\mathbf{P}_{\text{train}}[i, :], \mathbf{P}_{\text{test}}[j, :])$
// Sampling Device Weight Initialization
$\mu_j = \frac{1}{D_{\text{train}}} \sum_{i=1}^{D_{\text{train}}} \mathbf{S}_{ij}$
$\sigma_j^2 = \frac{1}{D_{\text{train}}} \sum_{i=1}^{D_{\text{train}}} (\mathbf{S}_{ij} - \mu_j)^2$
$W_{ij} = \frac{\exp\left(\alpha_{\text{sim}} \cdot \frac{\mathbf{S}_{ij} - \mu_j}{\sigma_j + \epsilon} / \tau\right)}{\sum_{i'=1}^{D_{\text{train}}} \exp\left(\alpha_{\text{sim}} \cdot \frac{\mathbf{S}_{i'j} - \mu_j}{\sigma_j + \epsilon} / \tau\right)}$
$W_{ij}^{\text{final}} = (1 - \beta) \cdot W_{ij} + \frac{\beta}{D_{\text{train}}}$
$w_i = \frac{1}{D_{\text{test}}} \sum_{j=1}^{D_{\text{test}}} W_{ij}^{\text{final}}$
// Weighted Point Sampling Begin
$s_1 = \arg\max_i \{\text{DiversityCoverageScore}(i)\}$ (Formula-2)
For $t = 2$ **to** k:
$d_i^{(t)} = \min_{s \in \mathcal{S}_{t-1}} \sum_{d=1}^{D_{\text{train}}} w_d \cdot |L_{id} - L_{sd}|$
$c_i^{(t)} = \sum_{d=1}^{D_{\text{train}}} w_d \cdot \frac{\max(0,\ \text{RangeExt}_{id})}{\text{CurrentRange}_d + \epsilon}$ if $t > k/2$
$s_t = \arg\max_{i \notin \mathcal{S}_{t-1}} \left[(1 - \alpha_{\text{cov}}) \cdot d_i^{(t)} + \alpha_{\text{cov}} \cdot c_i^{(t)}\right]$
Return: $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$

and improving coverage of the latency distribution.

Subsequent architectures are selected iteratively using a weighted farthest-point sampling strategy, where each candidate maximizes a combined objective of diversity and incremental coverage gain. The resulting subset constitutes the DBS for the target device and is used to guide downstream latency prediction.

Graph Interaction Modeling

To effectively model structural dependencies between neural operators and DBS, we adopt the *Dense Graph Flow (DGF)* framework (Ning et al. 2023), building on the strong performance of Graph Convolutional Networks (GCNs) for structured data (Ning et al. 2022). We further augment DGF with a graph-attention module inspired by (Akhauri and Abdelfattah 2024b). The attention branch inherits residual connection and identity mapping principles from GCNII (Chen et al. 2020), which mitigate the *over-smoothing* issue in deep GCNs and preserve high-order neighborhood dependencies.

Let O denote the concatenated operation embedding, X^l the node features at layer l, and A the sparse adjacency matrix. Trainable parameters include projection matrices W_o^l, W_f^l, W_p^l, an attention vector $\mathbf{a}$, and bias b_f^l. The DGF update is defined as:

$$X^{l+1} = \sigma\big(OW_o^l\big) \odot \big(A\left(X^l W_f^l\right)\big) + X^l W_f^l + b_f^l \quad (3)$$

where $\sigma(\cdot)$ is a non-linear activation (e.g., ReLU) and $\odot$ denotes element-wise multiplication. The first term modulates neighborhood aggregation via operation-aware gating, while the remaining terms form a residual path to retain identity information. To enhance node-level adaptivity, we incorporate a pairwise graph-attention mechanism. For each neighbor node j, its attention-weighted contribution is given by:

$$\text{Attn}_j(X^l) = \sigma\left(\text{LR}\left(A_j \cdot \mathbf{a}\left(W_p^l X^l \cdot W_p^l X_j^l\right)\right)\right) \cdot W_p^l X_j^l \quad (4)$$

where W_p^l projects node features into attention space, $\mathbf{a}(\cdot)$ is a trainable scoring function and $\text{LR}(\cdot)$ denotes LeakyReLU activation. Similarity between the center node and its neighbors is computed via the dot product. The final attention-based update aggregates all neighbor contributions and applies an operation-aware gating followed by normalization:

$$X^{l+1} = \text{LayerNorm}\left(\sigma\left(OW_o^l\right) \odot \sum_{j=1}^{n} \text{Attn}_j(X^l)\right) \quad (5)$$

Capability-level Metrics. Modern hardware platforms exhibit complex architectural heterogeneity across multiple abstraction levels in the compute and memory hierarchy (Dao et al. 2022; Srivastava, Arora, and Boneh 2024). Due to this diversity, the latency of an identical NN can vary significantly between devices, posing substantial challenges to cross-device latency prediction.

We observe that neural network computations exhibit structural regularity that permits their execution characteristics to be approximated by unified abstractions of hardware *capability*. As shown in Figure 1, we therefore construct a structured *Hardware Capability Vector (HCV)* that encodes a superset of capability-level metrics covering compute, memory, and parallelism resources, including `Peak Throughput`, `Core-Level Parallelism`, `Frequency`, `Peak Bandwidth`, `On-chip Memory Capacity`, and `DRAM/HBM Capacity`.

Guided by empirical insights, we instantiate this HCV with both linear-scale features (e.g., core frequency `clock_MHz`, FP32 peak throughput `fp32_tflops`) and higher-level architectural metrics. The latter capture device-family-specific resources, such as `core_count` and `fp32_au_per_sm` for GPUs, functionally analogous quantities (e.g., `dsp_count` on FPGAs), and general parameters such as inter-level memory bandwidth `BW_GBps`.

We define a unified HCV schema over this superset of metrics and populate each device as follows: if the architecture natively defines a metric, we directly record its value; if no exact counterpart exists but a semantically equivalent resource type is present (e.g., FP32 arithmetic units on GPUs vs. DSP slices on FPGAs), we use that functionally analogous quantity; otherwise, the metric is treated as non-applicable for that device family. To keep the representation tensorizable while preserving semantics, all HCV dimensions are numerically zero-padded and accompanied by an availability indicator, enabling the model to distinguish true zero-valued capabilities from metrics that are undefined or irrelevant for a given architecture.

This vector representation encapsulates the essential physical constraints that govern computation across the

compute and memory hierarchy, providing a compact, interpretable, and physically grounded abstraction of device execution capability. The resulting vectors are aligned with a predefined device list and stored in matrix form, yielding a unified encoding of the *compute–bandwidth–parallelism* profile. Crucially, by coupling structural insights with a workload-driven perspective, the HCV not only captures static hardware attributes but also serves as the foundational input to our Hardware–Operation Dialogue module, facilitating accurate and generalizable latency prediction across diverse hardware platforms.

Hardware–Operation Dialogue Module

In real-world deployments, NNs do not operate in isolation: their performance is co-determined by the target hardware's microarchitectural traits and runtime behavior. This forms a tightly coupled, bidirectional system where latency emerges from complex interactions between model structure and hardware execution dynamics. However, modeling such interactions is highly challenging. On one hand, hardware platforms differ vastly in their architectural hierarchies—e.g., compute throughput, memory capacity, and scheduling strategies, resulting in significant latency variation for the same network across devices. On the other hand, dynamic behaviors introduced by compiler-level optimizations (such as tiling, fusion, or memory coalescing) create intricate, dataflow-driven dependencies that cannot be captured by static architecture descriptors alone.

To bridge this gap, we propose the Hardware–Operation Dialogue module (HODM) that establishes a "dialogue" between the network and the hardware, dynamically capturing their runtime interactions and enabling accurate, hardware-aware latency prediction with strong generalization. As shown in Figure 2(c), the bottom-right panel highlights the HODM details. The GNN layers produce a feature tensor $X \in \mathbb{R}^{B\times L\times d}$, where B is the batch size, L the number of architectural operations, and d the hidden dimension. Each device yields a capability vector $H \in \mathbb{R}^{B\times k}$ (Hardware Capability Vector). We first apply *Feature-wise Linear Modulation* (FiLM) to adjust the channel distribution of each node in X based on hardware context:

$$\widetilde{X}_{b,i,:} = \gamma_{b,:} \odot X_{b,i,:} + \beta_{b,:}, \quad b = 1,\dots,B,\ i = 1,\dots,L \tag{6}$$

Here, $\gamma, \beta \in \mathbb{R}^{B\times d}$ are modulation vectors learned from H through two shared MLPs. This mechanism equips the predictor with *dynamic hardware awareness*: even on unseen platforms, a few target-device samples suffice to fine-tune γ and β for fast adaptation. Next, we compute cross-attention between the modulated features $\widetilde{X}$ and the hardware vector H. We project $\widetilde{X}$ to *queries* Q, and H to *keys* and *values* (K, V), then compute attention scores and aggregation:

$$A_{b,\ell,i} = \frac{\langle Q_{b,\ell,:}, K_{b,i,:}\rangle}{\sqrt{d_{head}}}, \quad \alpha_{b,\ell,i} = \underset{i}{\mathrm{softmax}}\,(A_{b,\ell,i}) \tag{7}$$

$$Z_{b,\ell,:} = \sum_{i=1}^{k} \alpha_{b,\ell,i} \cdot V_{b,i,:}, \qquad X' = \widetilde{X} + \alpha_{\text{res}} \cdot (ZW^O) \tag{8}$$

Here, $W^O \in \mathbb{R}^{d\times d}$ is a learnable projection, and α_{res} is a trainable scalar that controls the strength of residual fusion. This parameter allows the model to softly integrate hardware context, inspired by residual weight tuning in transformer backbones. Symmetrically, we reverse the attention direction: project H into queries and X into keys/values to compute hardware-aware response H'. Finally, the two branches are adaptively blended via a learnable gating coefficient $\lambda \in [0, 1]$:

$$\textsc{Output} = \lambda \cdot X' + (1 - \lambda) \cdot H' \tag{9}$$

The final representation thus encodes both how the *network queries the hardware* (X') and which network regions the *hardware attends to* (H'), substantially enhancing prediction robustness under heterogeneous devices.

Experiment

Experimental Setup

In this section, we rigorously evaluate the generalization capability of our latency predictor on hardware platforms and architectures that are not included in the training set. We report performance on the FBNet benchmark using Relative Approximation Error (RAE), and present the Cumulative Distribution Function (CDF) of absolute prediction errors across diverse heterogeneous devices. To further evaluate transfer reliability, we report the percentage of predictions falling within different RAE ranges. Finally, we extend our evaluation to several real-world platforms.

Dataset Construction: Following HW-NAS-Bench (Li et al. 2021), we evaluate our model on two representative NAS search spaces—NASBench-201 (Dong and Yang 2020) and FBNet (Wu et al. 2019), covering both cell-based and macro-level architectures. To further assess real-world generalization, we collect latency measurements across a diverse set of heterogeneous SoC and cloud platforms, spanning a broad range of compute capabilities and architectural characteristics. These devices form a challenging testbed for cross-device latency prediction. Rather than relying on arbitrary train-test splits, we adopt the correlation-aware device partitioning strategy introduced in FLAT (Akhauri and Abdelfattah 2024a). Specifically, we construct a device similarity graph based on pairwise negative Spearman correlations in latency behavior, and apply spectral bisection to produce dissimilar hardware groups. This yields four partition settings (N@1–N@4, F@1–F@4), designed to minimize latency correlation between training and test devices, thus enabling rigorous evaluation of cross-device generalization.

All models are trained under identical settings on A6000 GPUs, and all test devices are strictly held out during training to ensure fair and unbiased evaluation.

Evaluations

To evaluate prediction accuracy, we report the RAE, computed as $\text{RAE} = \frac{|\hat{y}-y|}{y}$, where $\hat{y}$ denotes the predicted latency and y is the ground-truth. Lower RAE indicates better prediction accuracy and is commonly used in regression tasks like latency prediction tasks (Xu et al. 2020).

	RAE↓ (%)								
	F@1					F@2			
	1080ti_1	1080ti_32	2080ti_32	titan_rtx_1	titanxp_1	eyeriss	fpga	raspi4	samsung_a50
#FLOPs	$12.65_{1.42}$	$14.72_{2.21}$	$13.91_{1.38}$	$20.53_{3.27}$	$18.37_{2.75}$	$21.83_{2.18}$	$35.97_{5.03}$	$22.46_{2.71}$	$40.15_{6.18}$
LAD	$7.85_{0.47}$	$5.42_{0.53}$	$9.04_{1.08}$	$15.38_{2.31}$	$13.43_{1.48}$	$18.54_{2.96}$	$17.67_{2.12}$	$13.39_{1.87}$	$30.76_{3.99}$
BRP	$5.85_{0.39}$	$6.20_{0.47}$	$11.18_{1.45}$	$10.45_{1.35}$	$14.66_{2.20}$	$7.64_{0.66}$	$17.14_{2.23}$	$13.29_{1.71}$	$20.39_{3.26}$
nn-Meter	$6.75_{0.54}$	$5.88_{0.58}$	$12.43_{1.86}$	$10.42_{1.36}$	$9.72_{1.65}$	$8.48_{1.10}$	$19.27_{2.74}$	$12.39_{1.63}$	$28.76_{4.02}$
FLAT	$5.45_{0.41}$	$5.31_{0.39}$	$6.12_{0.78}$	$10.25_{1.63}$	$8.66_{1.21}$	$7.56_{0.94}$	$20.29_{3.65}$	$11.33_{1.59}$	$39.67_{6.04}$
Ours	$\mathbf{3.58_{0.15}}$	$\mathbf{4.35_{0.51}}$	$\mathbf{5.14_{0.58}}$	$\mathbf{5.28_{0.72}}$	$\mathbf{3.08_{0.86}}$	$\mathbf{6.07_{0.41}}$	$\mathbf{13.54_{0.93}}$	$\mathbf{9.26_{1.22}}$	$\mathbf{19.41_{2.64}}$
	F@3					F@4			
	1080ti_1	1080ti_32	2080ti_1	titan_rtx_1	titan_rtx_32	1080ti_1	essential_ph_1	pixel2	
#FLOPs	$13.65_{1.38}$	$14.78_{1.84}$	$12.91_{1.15}$	$15.03_{2.36}$	$13.72_{1.78}$	$19.40_{2.33}$	$65.92_{9.01}$	$60.13_{11.02}$	
LAD	$5.16_{0.29}$	$5.81_{0.53}$	$4.90_{0.44}$	$7.45_{1.09}$	$6.76_{0.95}$	$14.70_{2.56}$	$53.88_{7.98}$	$52.17_{9.04}$	
BRP	$8.97_{0.83}$	$6.27_{0.58}$	$6.28_{0.51}$	$7.19_{1.03}$	$6.12_{0.62}$	$7.10_{0.77}$	$16.25_{2.34}$	$13.50_{2.28}$	
nn-Meter	$4.92_{0.24}$	$5.78_{0.49}$	$5.92_{0.56}$	$6.27_{0.84}$	$5.87_{0.51}$	$6.08_{0.73}$	$15.82_{2.76}$	$38.54_{6.18}$	
FLAT	$\mathbf{3.19_{0.19}}$	$5.65_{0.58}$	$5.78_{0.65}$	$9.32_{1.24}$	$6.06_{0.61}$	$5.56_{0.45}$	$14.83_{2.01}$	$24.03_{3.88}$	
Ours	$5.23_{0.26}$	$\mathbf{5.15_{0.21}}$	$\mathbf{3.76_{0.48}}$	$\mathbf{3.36_{0.58}}$	$\mathbf{3.15_{0.15}}$	$\mathbf{3.14_{0.11}}$	$\mathbf{13.03_{2.21}}$	$\mathbf{11.18_{2.23}}$	

Table 1: Relative Approximation Error (RAE ↓) across diverse hardware platforms under transfer learning. Devices are grouped by framework partitions (F@1–F@4). Bold indicates best performance per device. All baselines use open-source implementations and are evaluated on NVIDIA A6000 GPUs. Results are reported as mean_{std}.

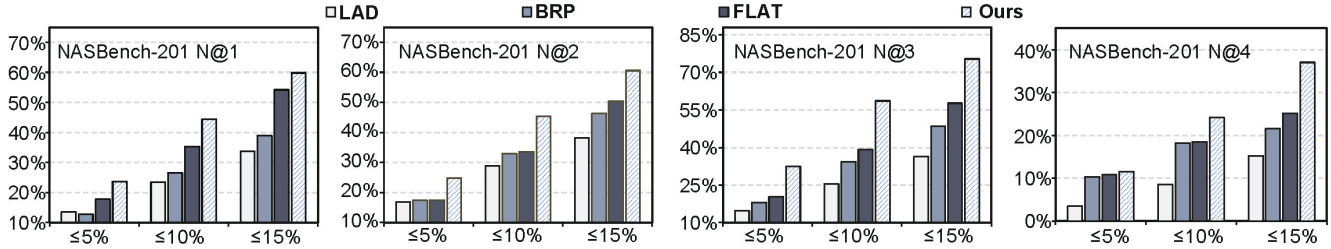

Figure 3: Percentage of predictions with error below 5%, 10%, and 15% across N@1–N@4. Each value indicates the proportion of correctly predicted samples, higher is better.

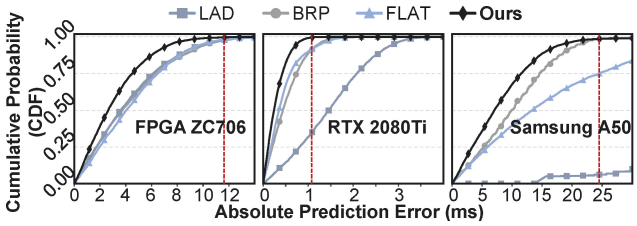

Figure 4: CDF of absolute prediction error on FBNet. A higher CDF indicates that a larger fraction of predictions fall within a given error threshold.

As shown in Table 1, our approach not only achieves the lowest RAE on an overwhelming majority of devices but does so with substantial margins that demonstrate its superior robustness. This superiority is particularly evident on challenging platforms; for instance, our model reduces prediction error on the high-end titan_rtx_1 (F@3) by 64% compared to FLAT, while simultaneously outperforming the FLAT competitor on the mobile pixel2 (F@4) by over 53%. Furthermore, our method demonstrates consistent performance across device categories, achieving the top rank on all nine devices in both the F@1 and F@2 partitions. In addition, we empirically observe that training on subsets of the training data whose hardware characteristics more closely match the target deployment scenario yields higher accuracy than using the entire training heterogeneous pool, suggesting that modest specialization to related domains can be beneficial.

CDF of Three Diverse Hardware Figure 4 compares the CDF of absolute latency-prediction errors for our method and the baseline across three diverse hardware platforms: an FPGA (Xilinx ZC706), a high-end GPU (RTX 2080 Ti), and a mobile phone (Samsung A50). A higher curve signifies a more accurate predictor, as it represents a larger fraction of predictions falling within any given error threshold. As shown, our CDF curve consistently lies above the baseline, demonstrating clear superiority across all platforms.

NASBench-201 Figure 3 summarizes the proportion of predictions with relative error within three predefined thresholds (5%, 10%, and 15%) across all NASBench-201 subsets. As shown, our method consistently outperforms all baselines across all test cases, demonstrating superior accuracy and generalization. The performance gap is especially

	Total RAE↓		Coverage (RAE ≤ x%) ↑					
			RAE ≤ 5.00%		RAE ≤ 10.00%		RAE ≤ 15.00%	
	FLAT	Ours	FLAT	Ours	FLAT	Ours	FLAT	Ours
AGX_15625	0.2109	**0.1820**	17.88%	**20.48%**	35.56%	**37.48%**	51.48%	**53.86%**
NX_8000	**0.1962**	0.2075	13.42%	**20.44%**	**36.70%**	30.22%	46.50%	**52.98%**
TX2	**0.8194**	1.0610	1.08%	**1.44%**	**7.50%**	7.32%	14.56%	**19.34%**
P4	0.2240	**0.2002**	17.82%	**19.72%**	35.72%	**36.26%**	48.54%	**51.90%**
P100	0.1785	**0.1575**	12.48%	**20.00%**	40.64%	**42.80%**	55.88%	**69.00%**
Titan XP	0.1604	**0.1200**	18.04%	**27.96%**	41.70%	**56.26%**	58.94%	**70.28%**
V100	0.1678	**0.1546**	15.54%	**21.08%**	43.68%	**52.52%**	57.62%	**61.44%**
Cambricon MLU270	0.1954	**0.1856**	**20.56%**	18.58%	**37.30%**	34.86%	51.76%	**53.62%**

Table 2: RAE and Prediction Coverage under Error Thresholds (5%, 10%, 15%) across Eight Heterogeneous Devices

Model	**1080ti_32**	**2080ti_32**	**titan_rtx_1**	**titanxp_1**
Baseline	5.16	10.48	9.36	12.23
Variant A	4.96 (↓**0.20**)	8.90 (↓**1.58**)	8.12 (↓**1.24**)	9.76 (↓**2.47**)
Variant B	4.73 (↓**0.43**)	7.91 (↓**2.57**)	7.11 (↓**2.25**)	6.85 (↓**5.38**)
Variant C	5.12 (↓**0.04**)	9.53 (↓**0.95**)	8.87 (↓**0.49**)	10.11 (↓**2.12**)
Ours	**4.35** (↓**0.81**)	**5.14** (↓**5.34**)	**5.28** (↓**4.08**)	**3.08** (↓**9.15**)

Table 3: Ablation study. RAE (%) comparison and delta (*vs GNN+One-hot*) across four hardware platforms.

notable on the N@3 dataset, where our model achieves 75.39% within the 15% error threshold—outperforming FLAT by 17.75%. Our approach also maintains a clear advantage in more challenging settings like N@4, underscoring its robustness across heterogeneous hardware.

Heterogeneous SoC and Cloud Devices Table 2 compares our method with FLAT across eight real-world platforms. Evaluation metrics include RAE and the proportion of predictions within relative error thresholds (5%, 10%, and 15%). Our method consistently achieves lower total RAE on most devices—showing notable gains. In terms of threshold-based accuracy, our predictor yields higher coverage under all error bounds on nearly all platforms. The prediction error on the Jetson TX2 is relatively higher than on other devices. This is mainly due to its significant distance from the training devices in terms of hardware hierarchy and compute capability.

Ablation Study

To assess the individual contributions of each core component in *CloserToMe*, we conduct a series of ablation experiments covering structured hardware modeling, the use of Device Behavior Signature (DBS), the DBSel-based sampling strategy, and Hardware–Operation Dialogue Module (HODM). We evaluate the following variants:

- **Baseline**: A widely used GNN-based predictor with identifier hardware encoding (like BRP); no hardware capability vector, no dialogue module, and no DBS.
- **Variant A**: Replaces identifier encoding with our Hardware Capability Vector and Dialogue Module; Device Behavior Signature is not used.
- **Variant B**: Builds on Variant A by incorporating DBS sampled randomly from the architecture space.
- **Variant C**: Similar to Variant B, but DBS are selected using an extreme policy (e.g., latency-homogeneous architectures) that lacks representativeness.
- **Ours**: Full model with DBSel-selected representative DBS, Hardware Capability Vector, and HODM.

As shown in Table 3, introducing structured hardware modeling via our HCV and HODM (Variant A) already yields consistent gains over the baseline. Adding DBS, even with random sampling (Variant B), further improves accuracy, while our DBSel-based variant (Ours) achieves the best overall performance—e.g., on `titanxp`, reducing RAE from 12.23% to 3.08%. In contrast, Variant C highlights the importance of profiling quality: unrepresentative DBS can mislead the predictor, whereas DBSel yields more informative and generalizable representations.

Conclusion

In this work, we present CloserToMe, a unified and hardware-aware framework for accurate and generalizable latency prediction across diverse deployment platforms. It integrates three key components: (1) the *Device Behavior Signature Selector* (DBSel), which efficiently samples representative model responses to characterize device behavior with minimal profiling cost; (2) a multi-level *Hardware Capability Vector* that encodes essential physical features such as frequency, bandwidth, and memory hierarchy; and (3) the *Hardware–Operation Dialogue Module*, a bidirectional attention mechanism that models fine-grained interactions between neural operators and hardware. Through explicit modeling of how architectures interact with platform-specific execution behavior, CloserToMe achieves robust prediction accuracy and strong cross-device generalization in deployment-oriented latency estimation.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grants 62502489 and 62172380; in part by the Jiangsu Provincial Natural Science Foundation under Grants BK20250479 and BK20241818; and in part by the Global Cooperation Expansion and Cultivation Fund of USTC YD5040008002.

References

Akhauri, Y.; and Abdelfattah, M. 2024a. On Latency Predictors for Neural Architecture Search. *Proceedings of Machine Learning and Systems (MLsys)*, 6: 512–523.

Akhauri, Y.; and Abdelfattah, M. S. 2024b. Encodings for Prediction-based Neural Architecture Search. In *International Conference on Machine Learning (ICML)*, 740–759. PMLR.

Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2020. Once-for-all: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations (ICLR)*.

Cai, H.; et al. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations (ICLR)*.

Chen, E.-C.; Chen, P.-Y.; Chung, I.; Lee, C.-R.; et al. 2024. Overload: Latency attacks on object detection for edge devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 24716–24725.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020. Simple and deep graph convolutional networks. In *International conference on machine learning (ICML)*, 1725–1735. PMLR.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems (NeurIPS)*, 35: 16344–16359.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 16487–16499.

Dudziak, L.; Chau, T.; Abdelfattah, M.; Lee, R.; Kim, H.; and Lane, N. 2020. Brp-nas: Prediction-based nas using gcns. *Advances in neural information processing systems (NeurIPS)*, 33: 10480–10490.

Fu, W.; Lou, W.; Tang, C.; Wen, H.; Qin, Y.; Gong, L.; Wang, C.; and Zhou, X. 2025. UniCoS: A Unified Neural and Accelerator Co-Search Framework for CNNs and ViTs. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.

Geng, X.; Wang, Z.; Chen, C.; Xu, Q.; Xu, K.; Jin, C.; Gupta, M.; Yang, X.; Chen, Z.; Aly, M. M. S.; et al. 2024. From algorithm to hardware: A survey on efficient and safe deployment of deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 36(4): 5837–5857.

Hanxiao, L.; Karen, S.; and Yiming, Y. 2019. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, 784–800.

Khare, A.; et al. 2025. SuperServe: Fine-Grained Inference Serving for Unpredictable Workloads. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 739–758.

Lee, H.; Lee, S.; Chong, S.; and Hwang, S. J. 2021. Hardware-adaptive efficient latency prediction for nas via meta-learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 27016–27028.

Li, C.; et al. 2021. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *In International Conference on Learning Representations (ICLR)*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations (ICLR)*.

Li, W.; Wang, D.; Ding, Z.; Sohrabizadeh, A.; Qin, Z.; Cong, J.; and Sun, Y. 2025. Hierarchical mixture of experts: Generalizable learning for high-level synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, 18476–18484.

Lin, J.; Chen, W.-M.; Lin, Y.; Gan, C.; Han, S.; et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in neural information processing systems (NeurIPS)*, 33: 11711–11722.

Liu, W.; Zhu, Z.; Li, B.; Xiong, Y.; Lian, Z.; Geng, J.; and Zhou, X. 2024. Arch2End: Two-Stage Unified System-Level Modeling for Heterogeneous Intelligent Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11): 4154–4165.

Lou, W.; Gong, L.; Wang, C.; Du, Z.; and Zhou, X. 2021. OctCNN: A high throughput FPGA accelerator for CNNs using octave convolution algorithm. *IEEE Transactions on Computers*, 71(8): 1847–1859.

Lou, W.; Gong, L.; Wang, C.; Qian, J.; Wang, X.; Li, C.; and Zhou, X. 2024. Unleashing network/accelerator co-exploration potential on fpgas: A deeper joint search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10): 3041–3054.

Lv, H.; Zhang, L.; and Wang, Y. 2025. In-situ NAS: A Plug-and-Search Neural Architecture Search framework across hardware platforms. *IEEE Transactions on Computers*.

Misra, A.; Saoda, N.; and Abdelzaher, T. 2025. Latency-constrained input-aware quantization of time series inference workflows at the edge. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications (INFOCOM)*, 1–10. IEEE.

Ning, X.; Zheng, Y.; Zhou, Z.; Zhao, T.; Yang, H.; and Wang, Y. 2023. A Generic Graph-Based Neural Architecture Encoding Scheme With Multifaceted Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(7): 7955–7969.

Ning, X.; Zhou, Z.; Zhao, J.; Zhao, T.; Deng, Y.; Tang, C.; Liang, S.; Yang, H.; and Wang, Y. 2022. TA-GATES: An encoding scheme for neural network architectures. *Advances*

in Neural Information Processing Systems (NeurIPS), 35: 32325–32339.

Srivastava, M.; Arora, S.; and Boneh, D. 2024. Optimistic verifiable training by controlling hardware nondeterminism. *Advances in Neural Information Processing Systems (NeurIPS)*, 37: 95639–95661.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (ICCV)*, 2820–2828.

Wang, Z.; Gong, L.; Lou, W.; Cheng, Q.; Chen, X.; Wang, C.; and Zhou, X. 2024. UniCoMo: A Unified Learning-Based Cost Model for Tensorized Program Tuning. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, 487–495. IEEE.

Wu, B.; et al. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 10734–10742.

Xing, J.; Wang, L.; Zhang, S.; Chen, J.; Chen, A.; and Zhu, Y. 2022. Bolt: Bridging the gap between auto-tuners and hardware-native performance. *Proceedings of Machine Learning and Systems (MLsys)*, 4: 204–216.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Shi, B.; Tian, Q.; and Xiong, H. 2020. Latency-aware differentiable neural architecture search. *arXiv preprint arXiv:2001.06392*.

Zhang, H.; Ning, A.; Prabhakar, R. B.; and Wentzlaff, D. 2024. Llmcompass: Enabling efficient hardware design for large language model inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 1080–1096. IEEE.

Zhang, L. L.; et al. 2021. Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 81–93.

Zhou, Q.; Wen, Y.; Chen, R.; Gao, K.; Xiong, W.; Li, L.; Guo, Q.; Wu, Y.; and Chen, Y. 2025. QiMeng-GEMM: Automatically Generating High-Performance Matrix Multiplication Code by Exploiting Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 39, 22982–22990.