GraphPerf-RT: Graph-Driven Performance Modeling with Calibrated Uncertainty for OpenMP Scheduling on Heterogeneous Embedded SoCs

Mohammad Pivezhandi[1] , **Mahdi Banisharif**[2] , **Saeed Bakhshan**[1] , **Abusayeed Saifullah**[3]
Ali Jannesari[2]
[1]Wayne State University, USA , [2]Iowa State University, USA , [3]The University of Texas at Dallas, USA
{pivezhandi, sbakhshan}@wayne.edu, {msharif, jannesar}@iastate.edu,
abusayeed.saifullah@utdallas.edu

Abstract

Autonomous AI agents on embedded platforms require real-time, risk-aware scheduling under resource and thermal constraints. Classical heuristics struggle with workload irregularity, tabular regressors discard structural information, and model-free reinforcement learning (RL) risks overheating. We introduce GraphPerf-RT, a graph neural network surrogate achieving deep learning accuracy at heuristic speeds (2–7ms).[1] GraphPerf-RT is, to our knowledge, the first to unify task DAG topology, CFG-derived code semantics, and runtime context (per-core DVFS, thermal state, utilization) in a heterogeneous graph with typed edges encoding precedence, placement, and contention. Evidential regression with Normal-Inverse-Gamma priors provides calibrated uncertainty; we validate on makespan prediction for risk-aware scheduling. Experiments on three ARM platforms (Jetson TX2, Orin NX, RUBIK Pi) achieve $R^2 = 0.81$ on log-transformed makespan with Spearman $\rho = 0.95$ and conservative uncertainty calibration (PICP $= 99.9\%$ at 95% confidence). Integration with four RL methods demonstrates that multi-agent model-based RL with GraphPerf-RT as the world model achieves 66% makespan reduction and 82% energy reduction versus model-free baselines, with zero thermal violations.

1 Introduction

Autonomous AI agents deployed on heterogeneous embedded Systems-on-Chip (SoCs) must make real-time scheduling decisions under stringent resource, energy, and thermal constraints. These agents, which power autonomous vehicles, robotic systems, and edge AI applications, require accurate performance prediction with calibrated uncertainty to ensure safe operation. Heterogeneous embedded SoCs combine high-performance and energy-efficient cores with Dynamic Voltage and Frequency Scaling (DVFS). This architecture has created unprecedented opportunities for deploying parallel applications in resource-constrained environments. OpenMP, the dominant shared-memory parallel programming model, enables developers to express task-level parallelism through pragma-based annotations that the runtime system maps to available hardware resources. However, achieving optimal performance on these heterogeneous platforms requires careful scheduling decisions that balance execution time, energy efficiency, and thermal constraints. This presents a challenging optimization problem that current approaches fail to address comprehensively. Performance prediction for parallel workloads has traditionally relied on analytical models, simulation-based methods, and machine learning surrogates. Analytical models grounded in queueing theory and scheduling bounds such as Brent's theorem provide theoretical insights but require simplifying assumptions that break down under irregular control flow and cache contention [Graham, 1969; Brent, 1974]. Simulation-based approaches offer high fidelity but incur prohibitive runtime overhead for online scheduling decisions. Machine learning surrogates, including Graph Neural Networks (GNNs), can learn complex input-output mappings from data [Veličković *et al.*, 2018]. However, existing approaches such as ProGraML [Cummins *et al.*, 2021] and Ithemal [Mendis *et al.*, 2019] operate on static intermediate representations without incorporating runtime context, thermal state, or DVFS configurations that significantly affect embedded system performance. Complementary approaches address related challenges through hierarchical multi-agent DVFS scheduling [Pivezhandi *et al.*, 2026b], zero-shot LLM-guided allocation [Pivezhandi *et al.*, 2026a], statistical feature-aware task allocation [Pivezhandi *et al.*, 2025], and flow-augmented few-shot RL [Pivezhandi and Saifullah, 2024].

This paper addresses the problem of predicting performance metrics for OpenMP task-parallel applications on heterogeneous embedded SoCs. The framework supports multiple targets including execution time, energy consumption, and hardware counters; we focus on makespan prediction under varying DVFS configurations, core allocations, and thermal conditions. The prediction model must provide accurate point estimates and *calibrated uncertainty quantification* to enable risk-aware scheduling decisions. This is essential for respecting thermal constraints and avoiding failures in safety-critical deployments. Furthermore, the model should generalize across benchmarks, input sizes, and hardware platforms

[1]Supplementary material and released artifacts are available at https://github.com/pivezhan/GraphPerf-RT_IJCAI2026.

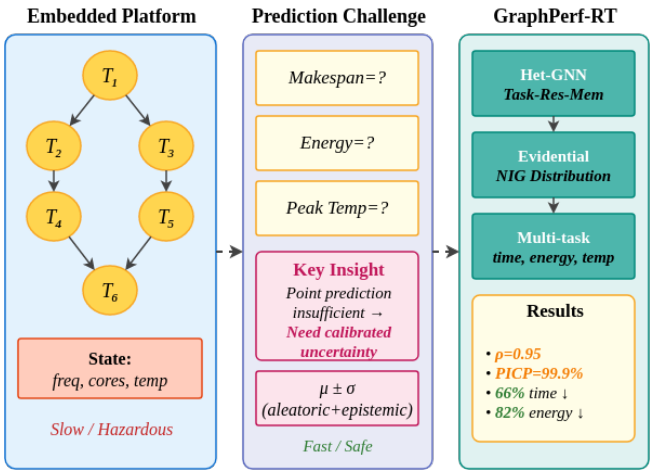

Figure 1: GraphPerf-RT as a world model for safe, uncertainty-aware scheduling.

without extensive retraining.

Accurate performance prediction on heterogeneous embedded SoCs presents four challenges. **First**, performance emerges from cross-layer interactions between application structure (task DAGs, CFGs), hardware state (frequencies, thermal headroom), and scheduling decisions; classical heuristics treat these independently, missing critical coupling effects that cause 3–5× execution time variation. **Second**, tabular models flatten task graphs into aggregate statistics, discarding dependency structure, while GNN approaches focus on static representations without runtime context. **Third**, standard regression lacks confidence estimates needed for risk-aware scheduling on thermally constrained systems. **Fourth**, model-free RL requires extensive on-device exploration risking overheating, while model-based RL needs an accurate environment model.

This paper introduces GraphPerf-RT, an AI technology that provides foundational infrastructure for safe, uncertainty-aware scheduling on embedded platforms. GraphPerf-RT addresses these challenges through a unified heterogeneous graph representation (see Fig. 1). Task nodes encode CFG-derived code semantics; resource nodes capture per-core DVFS state, utilization, and thermal headroom; typed edges encode task-task precedence, task-resource placement, and resource-resource topology. A heterogeneous Graph Attention Network (GAT) with type-specific encoders processes the graph through 3–6 attention layers. Evidential learning with Normal-Inverse-Gamma (NIG) distributions provides calibrated uncertainty in a single forward pass. The surrogate scores candidate configurations, filters low-confidence proposals, and supports Dyna-style model-based planning. Comprehensive experiments on three embedded ARM platforms (NVIDIA Jetson TX2, Jetson Orin NX, and RUBIK Pi) span 42 benchmarks from BOTS and PolyBench. GraphPerf-RT achieves $R^2 = 0.81$ on log-transformed makespan with Spearman $\rho = 0.95$ for ranking and conservative uncertainty calibration (PICP $= 99.9\%$ at 95% confidence), significantly outperforming tabular baselines and homogeneous GNN architectures. Integration with four RL methods (SAMFRL, SAMBRL, MAMFRL-D3QN, and MAMBRL-D3QN) demonstrates that MAMBRL-D3QN with GraphPerf-RT achieves 66% execution time reduction (0.97 ± 0.35s vs. 2.85 ± 1.66s) and 82% energy reduction (0.006 ± 0.005J vs. 0.033 ± 0.026J) compared to model-free baselines across 5 random seeds with 200 episodes each.

The main contributions of this paper, which establishes GraphPerf-RT as AI infrastructure for autonomous embedded agents, are:

1. A *unified heterogeneous graph representation* for AI agents that jointly encodes OpenMP task DAG topology, CFG-derived code semantics, and runtime context (per-core DVFS, thermal state, utilization) through typed nodes and edges, enabling explicit modeling of cross-layer performance interactions critical for autonomous decision-making.
2. An *evidential prediction framework* supporting multi-task learning with calibrated uncertainty quantification through Normal-Inverse-Gamma heads, producing both aleatoric and epistemic uncertainty in a single forward pass. We validate on makespan prediction; the architecture extends to energy and hardware counter targets. This provides essential AI safety infrastructure for risk-aware scheduling on embedded systems where overconfident predictions can cause thermal violations or hardware damage.
3. A *practical AI agent integration* demonstrating seamless combination with model-based RL, achieving 66% makespan and 82% energy improvements over model-free baselines while reducing hazardous on-device exploration through Dyna-style synthetic rollouts enabled by GraphPerf-RT as a world model.
4. A *reproducible AI4Tech evaluation framework* including a complete data pipeline (OMPi + ALF-llvm + SWEET + telemetry), extensive experiments across three ARM platforms and 42 benchmarks, and statistical significance tests with 5-seed confidence intervals.

2 Related Work

Performance Modeling. Performance modeling spans analytical queueing/Petri-net models [Eager and Sevcik, 1983; Marsan *et al.*, 1984], statistical surrogates [Lee and Brooks, 2006; Ipek *et al.*, 2006; Wang and O'Boyle, 2018; Chen and Guestrin, 2016], and neural predictors. Black-box predictors using counters or instruction mixes improve accuracy but lose structural signals. Ithemal [Mendis *et al.*, 2019] predicts x86 basic-block throughput via hierarchical LSTMs but operates on static assembly without runtime parallelism or thermal state. Runtime autotuners like Apollo [Ramadan *et al.*, 2024] adapt to input-dependent variations yet remain kernel-focused. We model OpenMP task DAGs on embedded SoCs, encoding dependencies, per-core DVFS, and thermal context for real-time decisions.

Graph Neural Networks. GNNs enable learning on structured program graphs [Wu *et al.*, 2021; Veličković *et al.*, 2018]. ProGraML [Cummins *et al.*, 2021] builds unified graphs from LLVM IR for compiler tasks but operates on

static IR without runtime or thermal context. HGT [Hu *et al.*, 2020] extends graph attention to multiple node/edge types but lacks: (1) edge decomposition separating structure from execution features, (2) thermal/DVFS integration, and (3) evidential heads for calibrated uncertainty. GNNs have predicted hardware-dependent performance [Huang *et al.*, 2010] and optimization effects [Brauckmann *et al.*, 2020]. We construct heterogeneous task-resource graphs with CFG-informed encoders and runtime context fusion.

Compiler Cost Models. Ansor, MetaSchedule, and ROLLER learn kernel-level evaluators with cross-device transfer [Zheng *et al.*, 2020; Shao *et al.*, 2022; Zhu *et al.*, 2022; Zhai *et al.*, 2023; Zhai *et al.*, 2024], but operate below task graphs and rarely encode DVFS/thermal state. Silhouette [Papon and Wasay, 2022] learns CPU embeddings for cross-platform prediction; Glimpse [Ahn *et al.*, 2022] develops mathematical hardware encodings for neural compilation. These focus on static kernel prediction without dynamic DVFS, thermal headroom, or task DAG structure. GraphPerf-RT integrates typed edges, execution features, and runtime device/thermal context into a unified graph representation.

Hardware-Aware ML and DVFS. Hardware-aware ML includes NAS with device constraints [Cai *et al.*, 2019] and cross-platform predictors [Walker *et al.*, 2017]. Energy/thermal DVFS spans heuristics, optimization, and RL [Xie *et al.*, 2021]. Utilization-driven governors miscale frequency by conflating compute and stalls [Hebbar and Milenković, 2021; Lin *et al.*, 2023]. Model-free RL adapts but is sample- and heat-intensive [Kim *et al.*, 2021]; model-based control reduces sampling via learned dynamics [Moerland *et al.*, 2023]. Recent work includes hierarchical multi-agent DVFS [Pivezhandi *et al.*, 2026b], feature-aware allocation [Pivezhandi *et al.*, 2025], zero-shot LLM guidance [Pivezhandi *et al.*, 2026a], and flow-augmented RL [Pivezhandi and Saifullah, 2024]. GraphPerf-RT is complementary: a calibrated graph-grounded evaluator supplying fast rollouts to MARL under thermal constraints.

Uncertainty Quantification. Uncertainty is critical for safe scheduling. Bayesian and ensemble methods add inference cost [Lakshminarayanan *et al.*, 2017; Kendall and Gal, 2017]. Evidential learning yields calibrated intervals without sampling [Sensoy *et al.*, 2018; Amini *et al.*, 2020]. Early formulations suffered from evidence contraction [Wu *et al.*, 2024], which we address through non-saturating regularization. We adopt evidential heads to expose confidence on makespan/energy and gate synthetic rollouts for real-time planning.

To our knowledge, GraphPerf-RT is the *first* unified graph representation combining OpenMP task DAGs with runtime context (per-core DVFS, thermal headroom, counters) and evidential regression for calibrated uncertainty in embedded scheduling. This enables risk-aware decisions under thermal constraints while maintaining computational efficiency for on-board deployment.

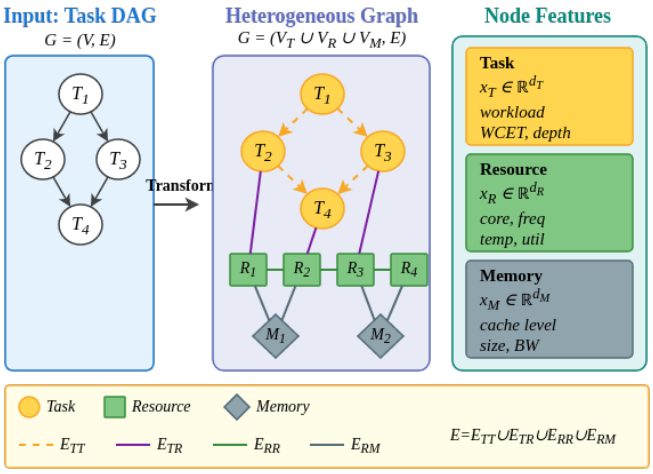

Figure 2: Heterogeneous graph with task (V_T), resource (V_R), and memory (V_M) nodes; typed edges encode cross-layer interactions.

3 Preliminaries

3.1 Problem Formulation

Extended formal definitions are provided in the supplementary material.

We predict multiple performance metrics for OpenMP task-parallel applications on heterogeneous embedded SoCs under varying DVFS configurations. Each application is a task DAG $G = (V, E)$ with tasks $v \in V$ characterized by CFG-derived features and precedence edges $e \in E$ (spawn, join, or data dependencies). The platform has C cores with discrete frequency levels $f_i \in \mathcal{F}_i$. The device sheet D captures immutable hardware constants (core types, cache specs, DVFS tables), while runtime state $\mathcal{S}$ includes per-core frequencies, utilization, and thermal readings.

A scheduling action $a = (m, f, p)$ specifies core mask $m \in \{0,1\}^C$, DVFS vector f, and optional priority p. Given $(G, D, \mathcal{S}, a)$, we learn $\mathcal{M} : (G, D, \mathcal{S}, a) \mapsto (y, \mathcal{U})$, where y can include makespan, energy, or hardware counters depending on the target configuration. We focus on makespan prediction with calibrated uncertainty $\mathcal{U}$ via NIG parameters $(\gamma, \nu, \alpha, \beta)$.

3.2 Heterogeneous Graph Abstraction

Extended graph formalization is provided in the supplementary material.

Performance emerges from interactions between application structure, hardware state, and scheduling decisions. As illustrated in Fig. 2, our heterogeneous graph has three node types: **Task nodes** V_T (orange in figure) encode CFG features, DAG topology (depth, distance-to-sink, centrality), and runtime snapshots. **Resource nodes** V_R (teal) encode per-core DVFS state, utilization, and thermal headroom. **Memory nodes** V_M (purple) encode cache hierarchy characteristics.

Four typed edges capture performance-critical cross-layer interactions: E_{TT} (task-task precedence with critical-path flags), E_{TR} (task-resource placement with affinity), E_{RR}

(resource-resource sharing/contention), and E_{RM} (resource-memory bandwidth constraints). Device constants from D are broadcast during encoding.

3.3 Data Collection Pipeline

Complete pipeline specification is provided in the supplementary material.

OpenMP sources are compiled through OMPi, with ALF-llvm emitting LLVM IR and ALF files. We use ALF not as a legacy artifact, but as a specialized intermediate representation that performs *semantic lifting*—explicitly exposing control-flow structure (loop bounds, recursion depths, branch patterns) that provides semantically dense features standard LLVM IR processing would miss. SWEET generates CFG/call graphs from this lifted representation, and post-processing maps entities to OpenMP tasks with topological encodings. This approach bridges static analysis and runtime performance modeling by capturing control-flow invariants critical for accurate prediction. Runtime logging captures one CSV row per execution with telemetry: timestamps, DVFS indices, measured frequencies, performance counters (cycles, instructions, cache/branch misses), energy, and thermal readings.

The dataset comprises 73,920 samples across three platforms: RUBIK Pi (8 Cortex-A72 cores), Jetson Orin NX (8 Cortex-A78AE cores), and Jetson TX2 (6 cores, Denver+A57). We evaluate 42 benchmarks from BOTS and PolyBench with 60/20/20 train/val/test splits stratified by (benchmark, input, core mask, DVFS index).

3.4 Learning Formulation

Extended architecture details are provided in the supplementary material.

Type-specific MLPs encode nodes into a common embedding space, with separate parameter sets for each node type to handle their distinct feature schemas. A heterogeneous GAT (3–6 layers) performs message passing over all edge types E_{TT}, E_{TR}, E_{RR}, and E_{RM}. Hierarchical pooling aggregates node embeddings per type and concatenates them to form graph representation $\mathbf{h}_G$. For each metric k, evidential heads output NIG parameters $(\gamma_k, \nu_k, \alpha_k, \beta_k)$ with the predictive mean $\hat{y}_k = \gamma_k$ and uncertainty decomposition: Aleatoric$_k$ $= \beta_k/(\alpha_k - 1)$ and Epistemic$_k$ $= \beta_k/(\nu_k(\alpha_k - 1))$. Training minimizes the negative log marginal likelihood with evidence regularization to prevent overconfidence. At runtime, batched configurations are scored, filtered by uncertainty gates, and executed with outcomes logged for continual learning.

4 Design Methodology

This section presents the GraphPerf-RT architecture. As shown in Fig. 3, the pipeline takes heterogeneous graph input with typed nodes (V_T for tasks, V_R for resources, V_M for memory), processes it through a GNN encoder with typed message passing, and produces multi-task predictions with calibrated uncertainty via evidential regression heads. The model takes task DAG $G = (V, E)$, device sheet D, runtime state $\mathcal{S}$, and action $a = (m, f, p)$, outputting predictions y with calibrated uncertainty $\mathcal{U}$.

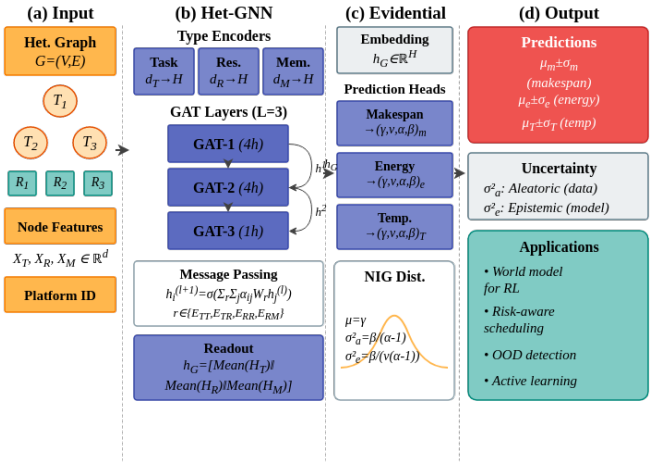

Figure 3: GraphPerf-RT architecture: heterogeneous graph encoding to evidential multi-task prediction.

4.1 Heterogeneous Graph Representation

The full intuition behind our task-resource-memory decomposition is provided in the supplementary material.

Node Types (complete specifications in the supplementary material). *Task nodes* V_T encode CFG-derived features (loop count, max depth, cyclomatic complexity, branch density), DAG topology metrics (depth, distance-to-sink, betweenness centrality), static code statistics (estimated instructions, bytes moved), and optional runtime snapshots (recent performance counters, run-mode flags). *Resource nodes* V_R encode per-core state: DVFS step (index and one-hot), core mask bit, cluster ID for heterogeneous architectures, utilization (exponential moving average), thermal headroom ($T_{\max} - T_{\text{current}}$), temperature trend ($\Delta T$ over recent samples), and bandwidth proxy. *Memory nodes* V_M encode cache hierarchy: level identifier, capacity/associativity/line size, and latency/bandwidth proxies.

Device constants from D (DVFS tables, cache sizes, governor flags) are broadcast as shared features during node encoding.

Edge Types (detailed rationale in the supplementary material). Four edge types capture performance-critical interactions. E_{TT} (task-task) edges encode precedence constraints with attributes: dependency type (spawn/join/data), critical-edge flag (on longest path), hop distance, and contention proxies. The critical path determines minimum makespan; fork/join patterns reveal synchronization overhead. E_{TR} (task-resource) edges connect tasks to cores under the scheduling assignment, with affinity strength and migration overhead attributes. Task-to-core affinity affects cache locality; load imbalance increases makespan. E_{RR} (resource-resource) edges connect cores sharing hardware components (L2 caches, memory controllers) to model contention, with sharing degree and interconnect latency attributes. E_{RM} (resource-memory) edges connect cores to cache levels, encoding bandwidth allocation. Memory-bound workloads are bottlenecked by bandwidth; cache miss rates determine effective memory latency.

4.2 GNN Architecture

Full architecture specifications are provided in the supplementary material.

Type-Specific Encoders. Since different node types have distinct feature spaces, we employ type-specific MLPs to encode raw features into a common embedding space of dimension $d = 128$: $\mathbf{h}_v^{(0)} = \text{MLP}_{\text{type}(v)}(\mathbf{x}_v; \theta_{\text{type}(v)})$ where $\text{type}(v) \in \{T, R, M\}$. Each MLP has 2–3 layers with ReLU activations and dropout $p = 0.1$. Topological encodings (positional embeddings from DAG depth, distance-to-sink, core cluster membership) are concatenated before encoding.

Heterogeneous GAT Layers. The core architecture consists of 3–6 heterogeneous graph attention layers that aggregate information from neighbors while accounting for edge types. For each layer l: $\mathbf{h}_v^{(l+1)} = \sigma\left(\mathbf{h}_v^{(l)} + \sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_r(v)} \alpha_{uv,r}^{(l)} \mathbf{W}_r^{(l)} \mathbf{h}_u^{(l)}\right)$ where $\mathcal{R} = \{TT, TR, RR, RM\}$, $\mathcal{N}_r(v)$ is the neighborhood under edge type r, $\alpha_{uv,r}^{(l)}$ is the attention coefficient, $\mathbf{W}_r^{(l)}$ is a type-specific transform, and σ is ELU. Attention coefficients incorporate edge features $\mathbf{e}_{uv,r}$ (e.g., contention proxies). Multi-head attention (4–8 heads) captures diverse interaction patterns.

Graph-Level Pooling. After the final layer L, hierarchical pooling aggregates per-type embeddings: $\mathbf{h}_G = \text{CONCAT}(\text{POOL}_T, \text{POOL}_R, \text{POOL}_M)$ where pooling can be mean, max, or attention-based. This yields a fixed-size graph representation (256–512 dimensions) regardless of graph size, enabling batched inference.

4.3 Evidential Learning for Uncertainty Quantification

Instead of directly predicting performance values, GraphPerf-RT learns parameters of a Normal-Inverse-Gamma (NIG) distribution for each metric, enabling uncertainty-aware predictions in a single forward pass (full formulation in the supplementary material).

Prediction Heads. For each metric $k \in \{\text{makespan}, \text{energy}, \text{cache misses}, \text{branch misses}, \text{utilization}\}$, an evidential head outputs NIG parameters: $(\gamma_k, \nu_k, \alpha_k, \beta_k) = \text{MLP}_k(\mathbf{h}_G; \theta_k)$ with $\nu_k = \text{softplus}(\tilde{\nu}_k) + 1$, $\alpha_k = \text{softplus}(\tilde{\alpha}_k) + 1$, $\beta_k = \text{softplus}(\tilde{\beta}_k)$ ensuring valid NIG parameters ($\nu_k, \alpha_k > 1, \beta_k > 0$). The mean prediction is $\hat{y}_k = \gamma_k$, with uncertainty decomposition: $\text{Aleatoric}_k = \beta_k/(\alpha_k - 1)$, $\text{Epistemic}_k = \beta_k/(\nu_k(\alpha_k - 1))$. Aleatoric uncertainty captures irreducible noise from OS scheduling and co-runner interference; epistemic uncertainty reflects model uncertainty that decreases with more training data.

Loss Function. Training minimizes the NIG negative log marginal likelihood with non-saturating uncertainty regularization [Wu *et al.*, 2024] to address evidence contraction on high-error samples, plus a ranking loss aligned to makespan-first scheduling objectives. The non-saturating term prevents the model from being overconfident on out-of-distribution inputs by penalizing high evidence when predictions are inaccurate, ensuring robust uncertainty estimates critical for thermal safety. The ranking loss operates on Lower Confidence Bounds (LCB = $\gamma - k\sigma$, where $\sigma = \sqrt{\beta(\nu+1)/(\nu(\alpha-1))}$ is the total predictive standard deviation) rather than point estimates, promoting risk-aware ordering that prioritizes configurations with both low predicted makespan and high confidence.

4.4 Training Procedure

GraphPerf-RT is trained using a multi-stage procedure designed for multi-task learning with uncertainty quantification (extended details in the supplementary material).

Stage 1: Feature Learning. The model is first trained using a standard multi-task regression loss (MSE per target, weighted by inverse variance), focusing on prediction accuracy without uncertainty quantification. Optimizer: AdamW with learning rate 10^{-3}, batch size 32–128 depending on graph size, epochs 50–100 with early stopping on validation MAE.

Stage 2: Evidential Training. The model is fine-tuned using the evidential loss function, enabling uncertainty quantification while maintaining prediction accuracy. Learning rate reduced to 10^{-4} with gradient clipping (norm 1.0) to handle NIG sensitivity, epochs 20–50.

Stage 3: Calibration. Post-hoc calibration on held-out data scales the predictive standard deviation $\sigma' = \tau \cdot \sigma$ where τ is fit to achieve target PICP>95% at 95% confidence. The scalar τ is optimized globally (across all platforms) to minimize $|\text{PICP} - 0.95|$ on the validation set. The model intentionally produces wider intervals than strictly necessary, ensuring scheduling decisions never rely on overconfident predictions—essential for safety-critical embedded systems.

Regularization. Graph edge dropout ($p = 0.1$–0.2), small feature noise ($\sigma = 0.05$), and mild device augmentation ($\pm 5\%$ DVFS tables) improve generalization. Scheduling-aware ranking losses align predictions with downstream policy selection.

4.5 Ablation Summary

Table 1: Ablation: component contributions to prediction accuracy (log-transformed makespan).

Configuration	R^2	ΔR^2	Spearman
Full GraphPerf-RT	0.81	—	0.95
w/o Hetero edges	0.77	−0.04	0.87
w/o Runtime telemetry	0.78	−0.03	0.89
w/o CFG features	0.79	−0.02	0.91
w/o Memory nodes (V_M)	0.79	−0.02	0.92
w/o Evidential (MSE)	0.80	−0.01	0.94
MLP (no graph)	0.75	−0.06	0.83
Homogeneous GCN	0.77	−0.04	0.87

Heterogeneous edges contribute the largest ranking improvement (Spearman +8%), followed by runtime telemetry (+6%), CFG features (+4%), and memory nodes (+3%). Memory nodes capture cache hierarchy interactions critical for memory-bound workloads. Evidential heads minimally affect point accuracy but are essential for calibrated uncertainty (PICP=99.9%), enabling safe scheduling decisions.

5 Experiments

5.1 Experimental Setup

Hardware Platforms. We evaluate GraphPerf-RT on three embedded ARM SoCs representing diverse architectures (full specifications in the supplementary material). **NVIDIA Jetson TX2** features a heterogeneous hexa-core configuration combining two Denver 2 cores with four Cortex-A57 cores, 12 discrete DVFS levels (345.6 MHz–2.0 GHz), and multiple thermal zones with a 50°C cap. **RUBIK Pi** is an 8-core Cortex-A72-class SBC with per-core userspace DVFS and energy measurement via qcom-battmgr or hwmon rails. **NVIDIA Jetson Orin NX** provides 8 Cortex-A78AE cores with userspace DVFS, representing the latest generation of embedded AI platforms. Across all platforms, we configure the Linux cpufreq subsystem for userspace governor control, discover DVFS indices dynamically, and sample thermal zones before/after execution.

Benchmark Suites. We evaluate 42 programs from two complementary suites (complete descriptions in the supplementary material). **BOTS** provides 12 task-parallel applications: *alignment*, *fft*, *fib*, *floorplan*, *health*, *concom*, *knapsack*, *nqueens*, *sort*, *sparselu*, *strassen*, and *uts* (unbalanced tree search with irregular parallelism). **PolyBench** contributes 30 kernels spanning linear algebra (gemm, cholesky, lu), stencils (jacobi-2d, seidel-2d), and data mining (correlation, covariance). Each benchmark executes with multiple inputs, core counts (1–8), and DVFS settings. Total dataset: 73,920 samples with 60/20/20 splits stratified by (benchmark, input, core mask, DVFS index).

Data Collection. We implement a client-server profiling framework (details in the supplementary material). The device client configures cpufreq, applies DVFS indices, launches benchmarks via a real-time wrapper, and collects performance counters (cycles, instructions, cache/branch misses), power samples, and thermal readings. Each run produces one CSV row with timestamps, frequencies, execution time, energy, counters, and temperatures.

Graph Extraction. OpenMP sources are compiled through OMPi; ALF-llvm emits LLVM IR and ALF files; SWEET generates CFG/call graphs (pipeline details in the supplementary material). Post-processing maps entities to OpenMP tasks with topological encodings (depth, distance-to-sink, centrality).

Baselines. We compare against seven approaches (detailed descriptions in the supplementary material): Linear Regression (Ridge regularization on tabular features), Random Forest (100 trees), MLP (3-layer feedforward), GCN (homogeneous message passing), Heterogeneous Graph Transformer (HGT) with type-aware attention, plus two uncertainty quantification variants (HGT+Ensemble with 5 models, HGT+MC Dropout). All use identical splits and tuning protocols.

5.2 Results

Overall Performance. Table 2 reports aggregate results on log-transformed makespan prediction. GraphPerf-RT achieves the lowest prediction error (RMSE=0.45, MAE=0.24 on log scale) and highest ranking quality (Spearman ρ=0.95). Compared to HGT (strongest baseline), GraphPerf-RT reduces RMSE by 31% (0.65→0.45) and improves R^2 from 0.78 to 0.81. The Spearman correlation of 0.95 indicates strong ranking capability, enabling effective scheduling decisions.

Table 2: Overall performance on log-transformed makespan. Lower RMSE/MAE better; higher R^2/Spearman better.

Model	RMSE	MAE	R^2	Spearman
Linear Reg.	1.21	0.89	0.67	0.71
Random Forest	0.98	0.72	0.74	0.78
MLP	0.83	0.59	0.75	0.83
GCN	0.71	0.50	0.77	0.87
HGT	0.65	0.45	0.78	0.89
HGT + Ensemble (5×)	0.62	0.43	0.79	0.90
HGT + MC Dropout	0.64	0.44	0.78	0.89
GraphPerf-RT	**0.45**	**0.24**	**0.81**	**0.95**

The progression from tabular baselines (Linear, RF, MLP) to graph-based methods (GCN, HGT, GraphPerf-RT) reveals the importance of structural information. MLP achieves Spearman=0.83 without graph structure; GCN improves to 0.87 with homogeneous edges; HGT reaches 0.89 with type-aware attention; GraphPerf-RT achieves 0.95 by combining heterogeneous message passing with evidential heads and runtime context.

Uncertainty Calibration. Table 3 shows uncertainty calibration metrics for makespan prediction. The model achieves exceptional Prediction Interval Coverage Probability (PICP) of 99.9% at 95% confidence, meaning the true values fall within the predicted intervals more often than the nominal rate. This *conservative* calibration is intentional and desirable for safety-critical embedded systems: the model produces slightly wider intervals than strictly necessary, ensuring scheduling decisions never rely on overconfident predictions. The uncertainty decomposition shows 94% aleatoric and 6% epistemic, indicating the model correctly identifies data-driven noise (OS scheduling jitter, thermal variation) as dominant, with model uncertainty contributing a smaller component that decreases with additional training data.

Table 3: Uncertainty calibration for makespan. MPIW = Mean Prediction Interval Width. PICP>nominal indicates conservative calibration.

Confidence	PICP	MPIW	Coverage Gap
95%	99.9%	58.1	+4.9%
90%	99.5%	47.2	+9.5%
80%	98.1%	35.8	+18.1%

Computational Efficiency. On-device inference completes in 2–7ms for typical task graphs (8 nodes, 56 edges), enabling real-time scheduling. Model size is 12.4MB, fitting embedded memory constraints (latency breakdown in the supplementary material).

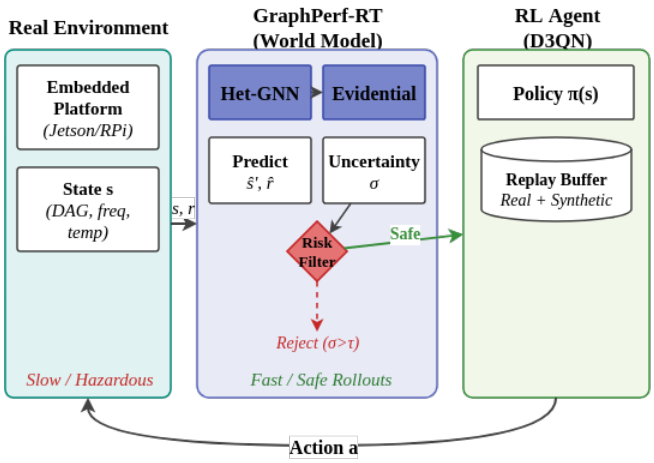

Figure 4: RL integration: GraphPerf-RT enables Dyna-style planning with uncertainty-aware filtering.

5.3 RL Baseline Evaluation

To validate end-to-end utility as a world model for scheduling, we integrate GraphPerf-RT with reinforcement learning on Jetson TX2 (see Fig. 4). The real embedded environment (left) provides state observations and rewards but is slow and hazardous for exploration; GraphPerf-RT as a world model (center) enables fast, safe synthetic rollouts with uncertainty-aware risk filtering; the D3QN agent (right) learns from both real and synthetic experiences stored in a unified replay buffer.

We use GraphPerf-RT as a deterministic transition model with epistemic uncertainty quantification, providing fast inference (2–7ms) with calibrated uncertainty bounds from the NIG posterior (design rationale in the supplementary material). We select Dueling Double DQN (D3QN) for its sample efficiency with Dyna-style planning, natural handling of discrete DVFS actions, and stability on embedded platforms (comparison with PPO/SAC in the supplementary material).

We compare four RL methods spanning single-agent vs. multi-agent and model-free vs. model-based paradigms (extended analysis in the supplementary material): (1) **SAMFRL**: Single-Agent Model-Free RL using standard Q-learning; (2) **SAMBRL**: Single-Agent Model-Based RL using GraphPerf-RT for synthetic rollouts; (3) **MAMFRL-D3QN**: Multi-Agent Model-Free RL with Dueling Double DQN; (4) **MAMBRL-D3QN**: Multi-Agent Model-Based RL with Dueling Double DQN, using GraphPerf-RT as shared world model. Each method trains for 200 episodes across 5 random seeds (42, 123, 456, 789, 1024). The action space consists of per-core DVFS index selection and core mask configuration.

Table 4: RL Performance with Thermal Safety (Jetson TX2, 200 episodes, 5 seeds, $T_{\max}$=50°C).

Method	Type	Makespan (s)	Energy (J)	$\bar{T}$ (°C)	T_{peak} (°C)	Viol.
SAMFRL	MF-SA	2.85 ± 1.66	0.033 ± 0.026	43.9	44.4	0%
SAMBRL	MB-SA	3.56 ± 3.07	0.038 ± 0.032	43.9	44.2	0%
MAMFRL-D3QN	MF-MA	4.97 ± 3.26	0.071 ± 0.056	42.9	44.0	0%
MAMBRL-D3QN	MB-MA	$\mathbf{0.97 \pm 0.35}$	$\mathbf{0.006 \pm 0.005}$	**41.9**	**43.2**	0%

Results. MAMBRL-D3QN achieves the best makespan (0.97 ± 0.35s) and energy (0.006 ± 0.005J), representing 66% makespan reduction (2.85s→0.97s) and 82% energy reduction (0.033J→0.006J) over single-agent model-free baselines. In the single-agent setting, SAMFRL slightly outperforms SAMBRL, suggesting model accuracy may limit single-agent planning. In the multi-agent setting, MAMFRL-D3QN (MF-MA) performs worse than single-agent methods without the world model, while MAMBRL-D3QN (MB-MA) significantly benefits from coordinated planning with GraphPerf-RT as the shared world model.

Analysis. The superior performance of MAMBRL-D3QN demonstrates that GraphPerf-RT provides sufficiently accurate predictions to enable effective model-based planning. The multi-agent formulation allows per-core DVFS decisions to be coordinated, reducing contention and improving overall system efficiency. Model-based methods exhibit faster initial convergence due to synthetic rollouts, though single-agent model-based (SAMBRL) plateaus at higher makespan due to limited coordination.

Table 4 shows MAMBRL-D3QN achieves the lowest average temperature (41.9°C vs. 43.9°C for model-free baselines), providing 8°C thermal headroom below the 50°C constraint compared to 6°C for baselines. While all methods maintain safe operation under our conservative test conditions, this 2°C improvement in thermal margin is critical for real deployments: it provides buffer against ambient temperature variations, enables sustained high-frequency operation without throttling, and extends hardware longevity. The uncertainty-aware filtering prevents the scheduler from proposing configurations that *would* violate thermal limits, ensuring safe exploration even under more aggressive optimization objectives.

Runtime Integration. Enumerate feasible actions under thermal caps, score with GraphPerf-RT, gate by uncertainty: $\text{gate}(a) = (\text{Epi}(a) \leq \eta) \wedge (\text{PI}(a) \leq T_{\max})$. Execute top-ranked safe action.

6 Conclusion

We presented GraphPerf-RT, an uncertainty-aware graph neural network surrogate for OpenMP scheduling on heterogeneous embedded systems. The unified heterogeneous graph representation integrates task DAG topology, CFG-derived code semantics, and runtime context (DVFS, thermal state). Evidential learning with NIG priors provides calibrated uncertainty decomposition into aleatoric and epistemic components. Evaluation across three ARM platforms and 42 benchmarks demonstrates $R^2 = 0.81$ on log-transformed makespan, Spearman $\rho = 0.95$, and PICP$= 99.9\%$ at 95% confidence. Integration with MAMBRL-D3QN achieves 66% makespan and 82% energy reduction over model-free baselines with zero thermal violations. See the supplementary material for limitations, broader impact, and future directions.

Reproducibility. Code, models, data, and figure scripts are publicly available at https://github.com/pivezhan/GraphPerf-RT_IJCAI2026.

Acknowledgement

The work was supported by the US Office of Naval Research through grant N00014-23-1-2151 and the US National Science Foundation through grants CNS-2601685 and CNS-2602744.

References

[Ahn *et al.*, 2022] Byung Hoon Ahn, Sean Kinzer, and Hadi Esmaeilzadeh. Glimpse: Mathematical embedding of hardware specification for neural compilation. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pages 1165–1170. ACM, 2022.

[Amini *et al.*, 2020] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 14927–14937, 2020.

[Brauckmann *et al.*, 2020] Alexander Brauckmann, Andrés Goens, Sebastian Ertel, and Jeronimo Castrillon. Compiler-based graph representations for deep learning models of code. In *Proceedings of the 29th International Conference on Compiler Construction (CC 2020)*, pages 201–211, 2020.

[Brent, 1974] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974.

[Cai *et al.*, 2019] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR 2019)*, 2019.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, pages 785–794, 2016.

[Cummins *et al.*, 2021] Chris Cummins, Zacharias V. Fisches, Tal Ben-Nun, Torsten Hoefler, Michael F. P. O'Boyle, and Hugh Leather. Programl: A graph-based program representation for data flow analysis and compiler optimizations. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, volume 139 of *Proceedings of Machine Learning Research*, pages 2244–2253, 2021.

[Eager and Sevcik, 1983] Derek L. Eager and Kenneth C. Sevcik. Performance bound hierarchies for queueing networks. *ACM Transactions on Computer Systems*, 1(2):99–115, 1983.

[Graham, 1969] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[Hebbar and Milenković, 2021] Ranjan Hebbar and Aleksandar Milenković. An experimental evaluation of workload driven DVFS. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE '21)*, pages 95–102, 2021.

[Hu *et al.*, 2020] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020 (WWW '20)*, pages 2704–2710. ACM, 2020.

[Huang *et al.*, 2010] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 883–891, 2010.

[Ipek *et al.*, 2006] Engin Ipek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*, pages 195–206, 2006.

[Kendall and Gal, 2017] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5574–5584, 2017.

[Kim *et al.*, 2021] Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. zTT: Learning-based DVFS with zero thermal throttling for mobile devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2021)*, pages 41–53, 2021.

[Lakshminarayanan *et al.*, 2017] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 6402–6413, 2017.

[Lee and Brooks, 2006] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII)*, pages 185–194, 2006.

[Lin *et al.*, 2023] Chengdong Lin, Kun Wang, Zhenjiang Li, and Yu Pu. A workload-aware DVFS robust to concurrent tasks for mobile devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom 2023)*, pages 1–16, 2023.

[Marsan *et al.*, 1984] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[Mendis *et al.*, 2019] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 4505–4515, 2019.

[Moerland *et al.*, 2023] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-

based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118, 2023.

[Papon and Wasay, 2022] Tarikul Islam Papon and Abdul Wasay. Silhouette: Toward performance-conscious and transferable CPU embeddings, 2022. Workshop on ML for Systems at NeurIPS 2023.

[Pivezhandi and Saifullah, 2024] Mohammad Pivezhandi and Abusayeed Saifullah. Flowrl: Flow-augmented few-shot reinforcement learning for semi-structured sensor data. *arXiv preprint arXiv:2409.14178*, 2024.

[Pivezhandi *et al.*, 2025] Mohammad Pivezhandi, Venkata Modekurthy, and Abusayeed Saifullah. Feature-aware task-to-core allocation in embedded multi-core platforms via statistical learning. IEEE RTCSA'25 (The 31st IEEE International Conference on Embedded and Real ..., 2025.

[Pivezhandi *et al.*, 2026a] Mohammad Pivezhandi, Mahdi Banisharif, Abusayeed Saifullah, and Ali Jannesari. Zerodvfs: Zero-shot llm-guided core and frequency allocation for embedded platforms. *arXiv preprint arXiv:2601.08166*, 2026.

[Pivezhandi *et al.*, 2026b] Mohammad Pivezhandi, Abusayeed Saifullah, and Ali Jannesari. Hidvfs: A hierarchical multi-agent dvfs scheduler for openmp dag workloads. *arXiv preprint arXiv:2601.06425*, 2026.

[Ramadan *et al.*, 2024] Tarek Ramadan, Ankur Lahiry, and Tanzima Z. Islam. Novel representation learning technique using graphs for performance analytics, 2024.

[Sensoy *et al.*, 2018] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 3183–3193, 2018.

[Shao *et al.*, 2022] Junru Shao, Xiyou Zhou, Siyuan Feng, Bohan Hou, Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro Masuda, Cody Hao Yu, and Tianqi Chen. Tensor program optimization with probabilistic programs. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pages 35783–35796, 2022.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR 2018)*, 2018.

[Walker *et al.*, 2017] Matthew J. Walker, Stephan Diestelhorst, Andreas Hansson, Anup Das, Sheng Yang, Bashir M. Al-Hashimi, and Geoff V. Merrett. Accurate and stable run-time power modeling for mobile and embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1):106–119, 2017.

[Wang and O'Boyle, 2018] Zheng Wang and Michael F. P. O'Boyle. Machine learning in compiler optimization. *Proceedings of the IEEE*, 106(11):1879–1901, 2018.

[Wu *et al.*, 2021] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[Wu *et al.*, 2024] Yuefei Wu, Bin Shi, Bo Dong, Qinghua Zheng, and Hua Wei. The evidence contraction issue in deep evidential regression: Discussion and solution. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*, volume 38, pages 21726–21734, 2024.

[Xie *et al.*, 2021] Guoqi Xie, Xiongren Xiao, Hao Peng, Renfa Li, and Keqin Li. A survey of low-energy parallel scheduling algorithms. *IEEE Transactions on Sustainable Computing*, 7(1):27–46, 2021.

[Zhai *et al.*, 2023] Yi Zhai, Yu Zhang, Shuo Liu, Xiaomeng Chu, Jie Peng, Jianmin Ji, and Yanyong Zhang. TLP: A deep learning-based cost model for tensor program tuning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023)*, pages 833–845, 2023.

[Zhai *et al.*, 2024] Yi Zhai, Sijia Yang, Keyu Pan, Renwei Zhang, Shuo Liu, Chao Liu, Zichun Ye, Jianmin Ji, Jie Zhao, Yu Zhang, and Yanyong Zhang. Enabling tensor language model to assist in generating high-performance tensor programs for deep learning. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 289–305, 2024.

[Zheng *et al.*, 2020] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 863–879, 2020.

[Zhu *et al.*, 2022] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, Fan Yang, Mao Yang, Lidong Zhou, Asaf Cidon, and Gennady Pekhimenko. ROLLER: Fast and efficient tensor compilation for deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 233–248, 2022.