Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

E3-Tune: Compiler-Directed Energy-Efficient Auto-Tuning on Edge GPUs

Qingzeng Song
Tiangong University
Mingze Sun
Tiangong University
Huayan Zhang
Tiangong University
Jingyu Wang
Tiangong University
Yongjiang Xue
xueyongjiang@tiangong.edu.cn

Tiangong University

Research Article

Keywords:

Posted Date: November 27th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9731480/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

E^3-Tune: Compiler-Directed Energy-Efficient Auto-Tuning on Edge GPUs

Qingzeng Song[1], Mingze Sun[1], Huayan Zhang[1], Jingyu Wang[1], Yongjiang Xue[1*]

[1*]School of Computer Science and Technology, Tiangong University, Tianjin, China.

*Corresponding author(s). E-mail(s): xueyongjiang@tiangong.edu.cn;
Contributing authors: qingzengsong@163.com;
2431101265@tiangong.edu.cn; zhanghy307@163.com;
2430101200@tiangong.edu.cn;

Abstract

Latency-oriented auto-tuning is insufficient for edge GPUs, where power, thermal, and dynamic voltage and frequency scaling (DVFS) constraints can make the fastest schedule energy-inefficient. This paper presents E^3-Tune, an energy-efficiency-aware auto-tuning framework for deep learning compilers on edge GPUs. E^3-Tune builds E^3-Set, a multimodal profiling dataset aligning intermediate representation (IR) schedule features, parallel thread execution (PTX) code features, hardware telemetry, and power measurements under iso-frequency conditions. It then uses a physics-aware power model in a cascaded search strategy, where latency prediction prunes the schedule space and PTX-aware power prediction re-ranks candidates by Energy-Delay Product (EDP). For DVFS, E^3-Tune dispatches frequency-specific kernels via Frequency-Guided Kernel Dispatch. Experiments show that E^3-Tune reduces energy by 20.1% and EDP by 19.9% over Ansor at high frequency, and reduces EDP by 14.4% under DVFS.

Keywords: Deep Learning Compilers, GPU Auto-Tuning, Edge GPUs, Energy-Efficient Inference, Power Modeling

1 Introduction

The growing deployment of deep learning in unmanned aerial vehicles (UAVs), mobile robots, and autonomous driving systems has made efficient and reliable edge inference increasingly important. In these scenarios, edge devices must sustain real-time execution under prolonged workloads and constrained operating conditions. Compared with data-center GPUs, edge GPUs such as the NVIDIA Jetson series [1] operate under tighter power budgets and more limited thermal headroom. Under sustained high load, these devices may trigger dynamic voltage and frequency scaling (DVFS) [2] and thermal throttling, leading to latency fluctuations, throughput degradation, and, in severe cases, protective downclocking or shutdown. Consequently, in edge deployment, raw performance alone is insufficient; energy efficiency and runtime stability are equally important.

Existing approaches to reducing energy consumption span multiple system layers. Hardware-level mechanisms, such as DVFS and power capping, can bound peak power, but their control granularity is often coarse and may sacrifice performance. Model-level approaches, including pruning [3] and quantization [4], reduce computation and memory traffic, but they may introduce accuracy degradation and usually require retraining. System-level strategies, such as batch-size adaptation and runtime scheduling [5], improve resource usage at a higher level of abstraction, yet they generally treat operators as black boxes and have limited visibility into instruction-level execution behavior. In contrast, kernel-level compilation and scheduling directly influence thread mapping, instruction scheduling, and memory hierarchy usage, making them a promising means of improving energy efficiency without changing model semantics.

Despite this potential, existing search-based auto-tuning frameworks, including TVM [6], Halide [7], Tensor Comprehensions [8], and FlexTensor [9], are still primarily optimized for latency. On edge GPUs, however, latency-oriented tuning may be misaligned with energy-efficient execution. Schedules that minimize runtime often rely on aggressive parallelism and high instantaneous resource utilization, which can increase power density and make throttling more likely. As a result, a schedule with the lowest latency does not necessarily provide the best end-to-end energy efficiency. Moreover, power and energy cannot be accurately inferred from latency alone. Prior studies on instruction-level power behavior on edge GPUs [10] have shown that power consumption is strongly affected by the PTX instruction stream, including instruction mix, memory-access behavior, and operand characteristics. Therefore, schedules with similar latency may still exhibit substantially different power and thermal characteristics after compilation, making latency an insufficient proxy for energy-aware tuning.

Introducing PTX information into an online energy-aware tuning process is nevertheless nontrivial. Power behavior depends not only on static instruction features but also on runtime microarchitectural activity and operating frequency. At the same time, edge devices provide only limited compute and memory resources for online decision making. Heavy semantic models, such as graph neural networks or large Transformers [11], are often too costly for deployment-time prediction, whereas efficient structured-feature models such as XGBoost [12] or LightGBM [13] may fail to capture instruction-sequence semantics adequately. This creates a practical tension between modeling accuracy and online overhead.

To address this challenge, this paper presents E^3-Tune, an energy-efficiency-aware auto-tuning framework for edge GPUs. The key idea is to decouple performance estimation from power estimation and to combine them in a cascaded optimization workflow. In the offline phase, E^3-Tune constructs a multimodal dataset, E^3-Set, which aligns IR-level schedule parameters, PTX features, and hardware telemetry under an iso-frequency profiling protocol. Based on this dataset, E^3-Tune trains a physics-aware hybrid power model that combines structured features, lightweight instruction-sequence semantics, and physically meaningful auxiliary supervision to replace expensive online power measurement with low-overhead prediction. In the online phase, E^3-Tune first uses an IR-based latency model to prune the schedule space to a high-performance subset, and then applies PTX-aware power prediction to rank candidates according to Energy-Delay Product (EDP). Final latency verification is performed only on a small number of candidates, thereby keeping the tuning process practical on resource-constrained edge devices. This design also enables the generation of frequency-specific kernels for representative operating modes, improving deployment adaptability under varying frequency conditions.

The main contributions of this work are threefold. First, we present E^3-Tune, an energy-efficiency-aware auto-tuning framework for edge GPUs that jointly considers performance and energy consumption while maintaining practical online tuning overhead. Second, we construct E^3-Set, a multimodal edge-GPU dataset that aligns IR-level schedule features, PTX information, power measurements, and hardware telemetry under controlled iso-frequency conditions, and develop a physics-aware hybrid power cost model for efficient and accurate power prediction. Third, we design an energy-oriented cascaded search strategy that combines latency-based coarse filtering with EDP-oriented fine ranking, enabling efficient discovery of low-EDP schedules and frequency-specific deployable kernels.

2 Related Work

2.1 Auto-Tuning in Tensor Compilers

Search-based auto-tuning has become a core technique in modern tensor compilers. Frameworks such as TVM, Halide, Tensor Comprehensions, and FlexTensor explore large scheduling spaces generated by transformations including data layout selection, loop tiling, thread binding, and vectorization, and use heuristic search methods, such as evolutionary algorithms [14], together with learned cost models to identify high-performance implementations. This workflow has proved effective for improving operator throughput on diverse hardware platforms.

Recent studies in *The Journal of Supercomputing* have further shown increasing interest in energy-aware optimization, edge GPU acceleration, and automatic performance tuning. Faqir-Rhazoui and García [15] evaluated SYCL for heterogeneous acceleration on edge platforms and reported both performance and energy behavior on Jetson-class devices. Martin-Salinas et al. [16] studied vision Transformer acceleration on GPU-based embedded edge AI systems, showing that GPU execution and compilation techniques can significantly reduce inference energy consumption. From the auto-tuning perspective, Cámara et al. [17] proposed an automatic optimization

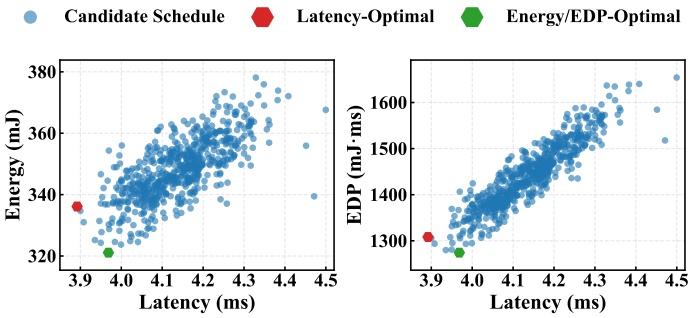

Fig. 1 Latency–Energy/EDP trade-off of candidate schedules on Jetson.

approach for selecting inter-GPU communication libraries on heterogeneous multi-GPU systems, while Dou et al. [18] presented a cross-layer multi-objective auto-tuning framework that jointly considers accuracy, execution time, and energy-related objectives for data analytic services. These recent studies confirm that energy efficiency, edge/heterogeneous acceleration, and auto-tuning are active topics in supercomputing systems.

However, most existing auto-tuning systems still focus primarily on performance metrics such as latency or throughput. Existing energy-aware and auto-tuning studies mainly focus on programming model evaluation, model-level acceleration, communication-library selection, or service-level hyperparameter tuning, while compiler-level schedule search for energy-efficient GPU kernels remains less explored. In contrast, E^3-Tune targets energy-efficiency-aware tensor compiler auto-tuning by integrating PTX-aware power prediction directly into the schedule search process and optimizing GPU kernels toward EDP under both fixed-frequency and DVFS edge-GPU settings.

2.2 Learning-Based Cost Modeling for Tensor Programs

To improve the sample efficiency of schedule search, a substantial body of work has investigated learning-based cost modeling for tensor programs. In particular, TenSet [19] provides a large-scale dataset that aligns runtime measurements with IR-level program features across diverse operators and schedules, enabling effective latency prediction for auto-tuning. Such efforts have significantly advanced performance-oriented compilation. Nevertheless, existing datasets and cost models mainly target latency prediction under relatively stable execution conditions. They generally do not include energy labels, DVFS-related effects, or microarchitectural telemetry, which limits their direct applicability to energy-aware optimization on edge GPUs.

3 Motivation

Edge GPUs (e.g., the NVIDIA Jetson series) operate under tight thermal design power (TDP) constraints and DVFS, which lead to energy-efficiency behaviors that differ

markedly from those of data-center GPUs. As illustrated in Figure 1, highly energy-efficient candidates tend to concentrate in the low-latency, high-performance region. This suggests that energy-optimal schedules are often contained within a subset of high-performance candidates.

However, latency optimality does not necessarily imply energy optimality within this subset. Schedules with similar latency can still exhibit substantial differences in energy consumption and EDP due to variations in instruction mix, resource utilization, and microarchitectural activity, such as register pressure, Tensor Core utilization, and memory contention. These observations suggest that an effective search strategy should first identify a promising high-performance subspace and then distinguish candidates within that subspace according to their energy behavior. This insight directly motivates the cascaded search design adopted in E^3-Tune.

4 Methodology

4.1 Problem Definition

Given an operator task on an edge GPU, let $\mathcal{S}$ denote the set of legal candidate schedules generated by the compiler, and let $f \in \mathcal{F}$ denote a target operating point (e.g., a locked-frequency mode or a representative frequency-capped mode). For each schedule $s \in \mathcal{S}$, compiling and executing s under f yields a kernel implementation with latency $T(s, f)$ and average power $P(s, f)$. Following prior energy-efficiency practice, we adopt EDP as the optimization objective:

$$\mathrm{EDP}(s, f) = P(s, f) \cdot T(s, f)^2. \tag{1}$$

Our goal is to identify, for each operator task and target operating point f, a schedule

$$s^*(f) = \arg\min_{s \in \mathcal{S}} \mathrm{EDP}(s, f), \tag{2}$$

while keeping the online tuning overhead practical on resource-constrained edge devices.

This problem is challenging for three reasons. First, the schedule space $\mathcal{S}$ is typically extremely large, making exhaustive hardware-in-the-loop evaluation infeasible. Second, on edge GPUs, power behavior depends not only on high-level schedule parameters but also on generated low-level code, microarchitectural activity, and frequency state. Third, direct online power measurement is expensive and noisy, especially on embedded platforms with limited sensor resolution and low sampling rates. As a result, latency-driven cost models alone are insufficient, whereas full online energy measurement is impractical.

To address these challenges, E^3-Tune adopts a performance–power decoupled design. Instead of directly searching the full schedule space with hardware power measurements, E^3-Tune first constructs a multimodal dataset for learning power behavior, then trains a low-overhead physics-aware predictor, and finally incorporates this predictor into a cascaded search process. This design approximates the EDP-optimal schedule under practical tuning constraints and enables frequency-specific deployment for different operating modes.

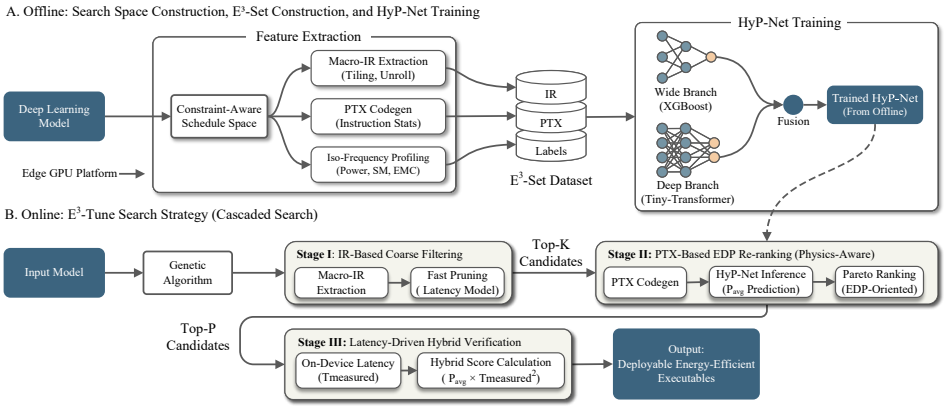

Fig. 2 Overview of the E^3-Tune framework: from multimodal data construction to energy-aware cascaded search.

4.2 Overview of E^3-Tune

Figure 2 illustrates the overall architecture of E^3-Tune. The framework consists of four tightly coupled components: constraint-aware search space construction, multimodal data collection and annotation, physics-aware power modeling, and energy-aware cascaded search. Together, these components enable practical schedule optimization toward EDP on edge GPUs.

Constraint-Aware Schedule Space Construction. E^3-Tune first constructs a legal and high-quality operator-level search space by incorporating hardware constraints during schedule generation. This process prunes invalid or inefficient configurations while preserving sufficient diversity in compute and memory behaviors for subsequent modeling and search. This component is detailed in Section 4.3.

E^3-Set: Data Collection and Annotation. Based on this executable search space, E^3-Tune builds E^3-Set, a multimodal dataset that aligns IR features, PTX features, power measurements, and hardware telemetry under an iso-frequency profiling protocol. The resulting dataset provides the foundation for learning power behavior beyond latency alone. The dataset construction and annotation process are described in Section 4.4.

HyP-Net: Physics-Aware Power Modeling. Using E^3-Set, E^3-Tune trains HyP-Net, a low-overhead hybrid model for power prediction. By combining structured scheduling features with PTX instruction semantics and auxiliary telemetry supervision, HyP-Net captures both explicit runtime factors and microarchitectural effects. The model design is presented in Section 4.5.

Energy-Aware Cascaded Search. Finally, E^3-Tune integrates latency prediction and power prediction in a cascaded search process. It first filters the full schedule space to a high-performance subset and then performs EDP-oriented re-ranking and

lightweight hardware verification to identify energy-efficient schedules under practical tuning cost. The search strategy is described in Section 4.6.

4.3 Constraint-Aware Search Space Construction

The effectiveness of E^3-Tune depends on a search space that is both expressive and executable. On the one hand, the space must be rich enough to cover diverse implementation strategies with different compute and memory behaviors. On the other hand, it must exclude schedules that are illegal, clearly inefficient, or unlikely to be deployable on resource-constrained edge GPUs. Therefore, rather than enumerating schedules in an unconstrained manner and filtering them only afterward, E^3-Tune incorporates hardware-aware constraints directly into schedule construction. This design reduces unnecessary compilation and profiling cost while ensuring that the retained candidates remain relevant for both offline modeling and online energy-aware search.

Starting from the computation graph, E^3-Tune decomposes the model into independently tunable operator tasks and instantiates schedule candidates from predefined sketches. Compared with data-center GPUs, edge GPUs are more sensitive to resource pressure, including shared-memory usage, register pressure, thread-block configuration, and memory-system contention. Accordingly, E^3-Tune performs schedule generation and pruning jointly through a hierarchical coarse-to-fine process.

4.3.1 Hierarchical Schedule Generation and Pruning

We construct the executable search space through four stages, each introducing a class of scheduling decisions together with corresponding legality and quality constraints. **(S1) Spatial and Reduction Tiling.** The first stage enumerates tiling configurations for spatial and reduction axes. These tiling decisions determine the basic granularity of computation, data reuse opportunities, and the balance between parallelism and locality. To avoid under-parallelized schedules and overly aggressive configurations that exceed on-chip resource limits, E^3-Tune prunes candidates with insufficient parallel work or excessive shared-memory demand. **(S2) Loop Transformation.** The second stage applies loop reordering and bounded unrolling to the tiled schedule skeletons. These transformations can improve data reuse and instruction-level parallelism, but they may also increase code size and register pressure. Since excessive unrolling is particularly problematic on edge GPUs, E^3-Tune excludes schedules that are likely to cause spilling or introduce disproportionate instruction overhead. **(S3) Parallelization and Thread Binding.** The third stage maps transformed loops onto CUDA execution hierarchies, including grid, block, and thread dimensions. This step determines available thread-level parallelism and directly affects occupancy and synchronization behavior. E^3-Tune enforces legality constraints such as the maximum number of threads per block and discards mappings that are clearly mismatched to the operator structure or likely to yield poor occupancy. **(S4) Vectorization and Memory-Hierarchy Optimization.** The final stage introduces vectorization, memory staging, layout decisions, and optional tensorization when applicable. These choices influence memory coalescing, shared-memory behavior, and the use of specialized compute units such as Tensor Cores. Since edge GPUs are sensitive to inefficient memory

behavior, this stage removes configurations that are likely to produce uncoalesced access, excessive staging overhead, or invalid tensorization patterns.

Through these four stages, the original combinatorial search space is contracted into a legal and high-quality executable subspace. This subspace is compact enough for practical tuning, while still preserving sufficient implementation diversity for subsequent modeling and search.

4.3.2 Search Space Diversity and Frequency Decoupling

Although E^3-Tune aggressively prunes invalid and low-quality schedules, it deliberately preserves microarchitectural diversity within the executable subspace. The retained candidates include both compute-oriented schedules with high arithmetic intensity and memory-oriented schedules dominated by data movement. As a result, the search space induces diverse runtime behaviors, including different levels of streaming multiprocessor (SM) activity, external memory controller (EMC) utilization, register pressure, and memory contention. This diversity is essential for E^3-Set, because it allows the power model to learn signals that are not captured by latency alone.

In addition, E^3-Tune adopts schedule–frequency decoupling as a core design principle. Rather than treating DVFS frequency as an explicit scheduling dimension, the framework models frequency as an external operating condition under which the same executable schedule can be evaluated. Consequently, the search space defined by Stages S1–S4 is frequency-agnostic, while the runtime performance and power behavior of each candidate are evaluated under specific operating points. This design avoids reconstructing the search space for every frequency mode and supports a write-once, evaluate-multi-frequency workflow for frequency-specific deployment on edge devices.

Overall, the proposed constraint-aware construction strategy transforms the original schedule space into a compact yet expressive executable subspace. This subspace serves two purposes in E^3-Tune: it provides diverse and physically meaningful samples for offline multimodal data construction, and it supplies the candidate pool on which the online energy-aware cascaded search operates.

4.4 E^3-Set: Data Collection and Annotation

Built on the constraint-aware executable search space described in Section 4.3, E^3-Tune constructs E^3-Set, a multimodal dataset designed for energy-aware cost modeling on edge GPUs. The purpose of E^3-Set is to provide aligned supervision across scheduling decisions, generated low-level code, and physical runtime behavior, so that power prediction can exploit information beyond latency alone. In contrast to existing tensor-program datasets that mainly focus on IR features and runtime performance, E^3-Set explicitly incorporates PTX-level representations and hardware telemetry signals under controlled profiling conditions.

To ensure that the dataset captures representative operator behaviors, we sample operator instances from diverse deep learning workloads, including both convolutional and Transformer-based models such as ResNet and BERT. Candidate schedules are

Table 1 Overview of the E^3-Set Dataset Structure.

Category	Feature / Component Details	Scale / Spec
1. Workload and Candidate Space		
Workloads	Representative CNNs and Transformers, including models such as ResNet, YOLO, and BERT.	150+ subgraphs
Candidates	Legal schedules sampled from the constraint-aware executable search space.	$>$ 50,000 samples
Coverage	Compute-intensive and memory-intensive behaviors preserved for diversity.	Diverse regimes
2. Program Representations		
IR features	Schedule-level features extracted from TIR, including tiling, loop structure, and unrolling information.	Macro-level
PTX statistics	Structured descriptors derived from PTX, such as instruction mix and resource-related summaries.	Tabular features
PTX semantics	Normalized PTX token sequences composed of opcodes, modifiers, and operand categories.	Sequence features
3. Physical Labels and Telemetry		
Power label	System-level total input power (P_{sys}) used as the primary regression target.	100 Hz sampling
Telemetry	Hardware activity signals including SM activity and EMC utilization.	Auxiliary targets
Windowing	Final labels computed from the mean over the steady-state profiling window.	Stable average
4. Profiling Protocol		
Frequency control	GPU core and memory frequencies locked during measurement.	Iso-frequency
Noise control	WiFi/Bluetooth disabled, fan speed fixed, and background interference minimized.	Controlled runtime
Measurement source	`tegrastats` monitoring of `VDD_IN` and related device telemetry.	Edge-device profiling

collected from the operator-level executable subspace generated by the constraint-aware construction process. Because this subspace already excludes illegal and clearly inefficient configurations while preserving microarchitectural diversity, the resulting samples span a broad range of execution regimes, including compute-intensive and memory-intensive behaviors. This diversity is important for learning power-relevant patterns that are not directly reflected by latency.

For each candidate schedule, E^3-Set records two complementary types of program representation. First, we extract IR-level scheduling parameters from the compiler intermediate representation, including features that describe transformations such as tiling, loop structure, and unrolling. These features provide a compact view of the schedule configuration and are efficient to obtain during tuning. Second, we invoke NVCC to generate PTX and derive PTX-level features from the compiled low-level code. To preserve both efficiency and expressiveness, we represent PTX in two forms: (1) structured statistical descriptors, such as instruction mix and resource-related summaries, and (2) normalized instruction-sequence semantics. For the latter, PTX instructions are converted into token sequences consisting of opcodes, modifiers, and

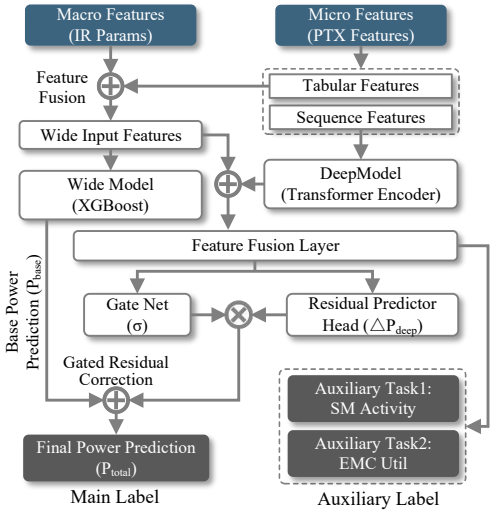

Fig. 3 Physics-Aware Wide & Deep Power Cost Model.

operand categories, while register indices and immediate values are masked to control vocabulary size without discarding structural information.

In addition to program features, E^3-Set provides physically grounded labels collected under an iso-frequency profiling protocol. We use system-level total input power as the primary regression target, since it more directly reflects deployment-time energy behavior than isolated component-level measurements. To reduce the confounding effect of DVFS and environmental noise, GPU core and memory frequencies are locked during measurement, and background interference is minimized through controlled runtime settings, including disabling WiFi/Bluetooth and fixing fan speed. We monitor `VDD_IN` via `tegrastats` [20] and sample power at 100 Hz. The final power label for each candidate is computed from the mean value over the steady-state profiling window.

To further improve the physical relevance of the dataset, E^3-Set also records hardware telemetry signals such as SM activity and EMC utilization. These signals are not treated merely as auxiliary statistics; instead, they provide intermediate observations of the underlying compute and memory behavior induced by each schedule. By aligning IR features, PTX representations, system-level power, and hardware telemetry within the same sample, E^3-Set forms a unified multimodal basis for training physics-aware power models.

Overall, E^3-Set serves as the data foundation of E^3-Tune. It bridges the gap between high-level scheduling decisions and low-level energy behavior on edge GPUs, enabling the subsequent HyP-Net model to perform low-overhead yet physically informed power prediction. Table 1 summarizes the main modalities and label definitions included in E^3-Set.

4.5 HyP-Net: Physics-Aware Power Modeling

The role of HyP-Net in E^3-Tune is to provide accurate yet low-overhead power prediction during online schedule optimization. This requirement is particularly important on edge GPUs, where power behavior depends not only on high-level schedule parameters but also on generated PTX code, runtime microarchitectural activity, and operating frequency. A practical predictor must therefore balance two competing objectives: it should be lightweight enough for deployment-time inference, while still remaining expressive enough to capture instruction-level effects that are not visible from IR features alone.

To meet this requirement, we design HyP-Net as an asymmetric physics-aware Wide & Deep model [21] trained on E^3-Set. As illustrated in Figure 3, the *Wide pathway* operates on structured features, including frequency settings, IR-level schedule parameters, and tabular PTX statistics, to capture the dominant and physically interpretable component of power consumption. In parallel, the *Deep pathway* processes normalized PTX instruction sequences with a lightweight semantic encoder to model fine-grained instruction-level effects. HyP-Net then combines the two pathways through a gated residual formulation and uses auxiliary physical supervision from hardware telemetry to improve prediction accuracy and physical consistency.

4.5.1 Design Rationale

The design of HyP-Net is motivated by the observation that power on edge GPUs exhibits both *structured regularity* and *instruction-level variability*. A substantial portion of power behavior can be explained by explicit factors such as operating frequency, schedule parameters, resource usage, and instruction statistics. These signals are naturally suited to the Wide pathway, where efficient tabular modeling can provide a stable baseline power estimate with low inference overhead. However, schedules with similar structured features may still compile into PTX programs with different dependency patterns, memory-access characteristics, or instruction ordering, resulting in noticeably different power consumption. These effects require the Deep pathway, which captures PTX instruction-sequence semantics beyond handcrafted statistics.

A purely deep semantic model, however, would be unnecessarily expensive and less stable for deployment-time inference on resource-constrained devices. HyP-Net therefore adopts a hybrid Wide & Deep design: the Wide pathway captures the dominant baseline trend from explicit physical and scheduling features, while the Deep pathway provides a compact residual correction for instruction-level effects. This decomposition improves both efficiency and robustness, since the semantic branch only needs to model the portion of power behavior that is not already explained by the structured physical features.

4.5.2 Wide Pathway for Structured Physical Features

The Wide pathway is responsible for modeling the dominant, physically interpretable component of power consumption. It operates on structured features extracted from E^3-Set, including locked frequency settings, IR-level scheduling parameters, and tabular PTX descriptors such as instruction statistics and resource-related summaries.

These features provide a compact representation of scheduling and code-generation behavior and can be evaluated efficiently during online tuning.

We use XGBoost as the predictor in the Wide pathway. This choice is motivated by its strong performance on heterogeneous tabular data, robustness under limited training cost, and low inference overhead compared with heavier neural alternatives. The output of this branch is interpreted as a baseline power estimate, denoted by P_{base}, which captures the dominant trend induced by explicit physical and schedule-level factors. In addition, a linear projection maps selected structured features into a dense physical context vector, denoted by $\mathbf{c}$, which is later used in fusion and gating. In this way, the Wide pathway provides both an efficient baseline regressor and a compact contextual summary for downstream correction.

4.5.3 Deep Pathway for PTX Instruction Semantics

The Deep pathway complements the structured branch by modeling PTX instruction-sequence semantics. Its input is the normalized PTX token sequence constructed during E^3-Set generation, in which opcodes, modifiers, and operand categories are preserved while raw register indices and immediate values are masked. This representation retains the structural regularities of the generated PTX code while avoiding an unnecessarily large vocabulary.

To encode these sequences, HyP-Net employs a lightweight Transformer-based module. Compared with recurrent models, the Transformer is better suited for capturing nonlocal dependencies in instruction streams, such as read-after-write interactions, instruction clustering, and memory-access patterns that may affect runtime power. At the same time, the model is deliberately pruned and kept shallow so that its inference cost remains compatible with edge-oriented tuning. The output of this branch is a semantic representation, denoted by $\mathbf{s}$, which summarizes instruction-level behaviors that are difficult to express through aggregated PTX statistics alone.

The semantic branch is not intended to replace the structured model. Instead, it acts as a complementary signal source that refines the baseline power estimate when instruction-level effects become significant. This role is particularly important for distinguishing schedules with similar latency or similar high-level structure but different microarchitectural execution behavior.

4.5.4 Multi-Task Prediction and Gated Fusion

After feature extraction, the physical context vector $\mathbf{c}$ and the semantic vector $\mathbf{s}$ are fused to support multi-task prediction. Rather than predicting power directly from the fused representation alone, HyP-Net decomposes prediction into a baseline term and a residual term. The final power estimate is computed as

$$P = P_{\text{base}} + g \cdot \Delta P, \quad g = \sigma(\text{MLP}([\mathbf{c}; \mathbf{s}])) \in (0, 1), \tag{3}$$

where P_{base} is the baseline prediction from the Wide pathway, ΔP is the residual correction inferred from the fused representation, and g is a learnable gate.

This formulation serves two purposes. First, it preserves the interpretability and stability of the baseline predictor by treating structured physical factors as the dominant component of power behavior. Second, it allows the semantic branch to contribute adaptively rather than uniformly. When the baseline term already explains most of the observed behavior, the gate suppresses unnecessary correction; when PTX-level effects become more pronounced, the residual contribution is amplified. This mechanism reduces the risk of over-correction and improves generalization across operators and operating points.

In addition to the power residual, the fused representation is used to predict two auxiliary hardware signals: SM activity and EMC utilization. These auxiliary outputs are not introduced merely to increase supervision volume. Instead, they provide physically meaningful intermediate targets that reflect the compute- and memory-side behavior induced by each schedule. By learning these related signals jointly, HyP-Net encourages the shared representation to encode distinctions between compute-intensive and memory-intensive execution patterns, thereby improving both physical consistency and robustness on unseen schedules.

4.5.5 Training Objective and Optimization

HyP-Net is trained with a physics-guided composite objective that combines absolute regression, relative ranking, and auxiliary-task supervision:

$$\mathcal{L} = \lambda_{mse}\mathcal{L}_{mse} + \lambda_{rank}\mathcal{L}_{rank} + \lambda_{aux}(\mathcal{L}_{SM} + \mathcal{L}_{EMC}). \tag{4}$$

Here, $\mathcal{L}_{mse}$ is the mean squared error for absolute power prediction, $\mathcal{L}_{rank}$ is a pairwise ranking loss [22] constructed from ground-truth EDP, and $\mathcal{L}_{SM}$ and $\mathcal{L}_{EMC}$ are auxiliary losses for hardware telemetry prediction.

Each term plays a distinct role. The regression loss improves numerical accuracy in predicting average power. The ranking loss encourages the model to preserve the relative ordering of schedules with respect to EDP, which is particularly important because the downstream search procedure relies on ranking quality rather than regression accuracy alone. The auxiliary losses inject additional physical structure into the learned representation, helping the model capture whether a schedule is dominated by compute activity or memory-system pressure.

We optimize HyP-Net using AdamW [23]. During training, the loss weights are adjusted dynamically so that the model focuses more on stable absolute regression in the early stage and progressively increases the influence of ranking and auxiliary supervision in later epochs. This curriculum-like strategy improves convergence stability while aligning the final predictor with the ranking-oriented objective used during online schedule optimization.

Overall, HyP-Net serves as the core power estimator in E^3-Tune. By combining efficient structured modeling, lightweight PTX semantic encoding, and physically meaningful auxiliary supervision, it replaces expensive online power measurement with a practical predictor that remains sensitive to microarchitectural effects. This capability enables the subsequent cascaded search procedure to optimize toward EDP under realistic tuning budgets on edge GPUs.

4.6 Energy-Aware Cascaded Search

With the executable search space defined in Section 4.3 and the low-overhead power estimator provided by HyP-Net, E^3-Tune performs schedule optimization through an energy-aware cascaded search process. The purpose of this design is to identify schedules with low EDP without incurring the prohibitive cost of exhaustive hardware-in-the-loop evaluation. This is particularly important on edge GPUs, where the candidate schedule space is extremely large, while online compilation and stable physical measurement remain expensive.

A key observation behind our design is that energy-efficient schedules are typically located within a high-performance subset of the full search space, but latency alone is insufficient to distinguish the best candidates within that subset. E^3-Tune therefore adopts a three-stage funnel that follows the cost hierarchy of evaluation:

$$\text{IR inference} \ll \text{PTX compilation and power prediction} \ll \text{on-device measurement}.$$

The search process first removes clearly low-quality candidates using inexpensive latency prediction, then applies PTX-aware power estimation to rank the retained candidates by predicted EDP, and finally verifies only a small number of top-ranked candidates on hardware. In this way, E^3-Tune concentrates the expensive evaluation budget on the most promising regions of the search space.

4.6.1 Stage I: IR-Based Coarse Filtering

The first stage performs large-scale coarse filtering over the full schedule space. For each candidate schedule, E^3-Tune extracts lightweight IR-level features without triggering low-level compilation and uses TVM's online-trained latency cost model to estimate its runtime, denoted by T_{pred}. Because IR feature extraction is inexpensive, this stage can evaluate a large number of candidates at negligible cost compared with PTX generation or device measurement.

The purpose of Stage I is not to identify the final schedule directly, but to discard candidates that are clearly unlikely to be competitive. After latency prediction, all candidates are ranked by T_{pred}, and only the Top-K candidates are retained for the next stage. This design is motivated by the observation that energy-optimal schedules are usually contained in a high-performance region of the search space. By restricting subsequent processing to this reduced subset, E^3-Tune avoids wasting compilation and prediction effort on candidates that are already unpromising from a performance perspective.

4.6.2 Stage II: PTX-Based EDP Re-ranking

The second stage refines the candidate set by incorporating power-related information that is not visible at the IR level. For the Top-K schedules retained from Stage I, E^3-Tune generates PTX in parallel and extracts microarchitectural descriptors such as instruction mix, resource-related statistics, and normalized instruction-sequence

semantics. These PTX-level signals capture low-level execution properties, including register pressure, memory-instruction density, and Tensor Core usage, which may substantially affect power consumption even when latency is similar.

HyP-Net is then used to predict the average power of each retained candidate at the target operating point, denoted by P_{pred}. Combining this prediction with the latency estimate from Stage I yields the predicted EDP:

$$\text{EDP}_{\text{pred}} = P_{\text{pred}} \cdot T_{\text{pred}}^2. \tag{5}$$

Candidates are re-ranked according to EDP_{pred}, and only the Top-P candidates, where $P \ll K$, are passed to the final stage.

This stage serves as the core energy-aware refinement step of E^3-Tune. Its role is to distinguish among candidates that are all relatively fast but may differ substantially in energy behavior due to PTX-level microarchitectural effects. By introducing power prediction only after the initial latency-based filtering, E^3-Tune preserves practical search cost while still moving beyond a purely performance-driven objective.

4.6.3 Stage III: Latency-Driven Hybrid Verification

The final stage performs lightweight hardware verification on the Top-P candidates. A straightforward strategy would be to measure both latency and power on device for these candidates. However, on edge platforms, direct online power measurement is often impractical for large-scale tuning. Embedded power sensors typically have limited sampling rates and are sensitive to noise, which means that stable average power estimation requires relatively long execution windows. Such measurements are significantly more expensive and less robust than short latency measurements based on hardware timers.

For this reason, E^3-Tune adopts a hybrid verification strategy. Instead of re-measuring power online, it measures only the true runtime of each of the Top-P candidates, denoted by T_{measured}, and combines this value with the predicted power from HyP-Net to compute the final score:

$$\text{Score}_{\text{final}} = P_{\text{pred}} \cdot T_{\text{measured}}^2. \tag{6}$$

The schedule with the best final score is selected as the optimized implementation for the target operating point.

This hybrid design has two advantages. First, it preserves the high precision of on-device latency measurement, which is inexpensive and stable. Second, it avoids the substantial overhead of repeated online power measurement while still allowing the search process to remain energy-aware. In addition, the measured latency values obtained in this stage can be fed back to the Stage I latency model as new supervision, improving its accuracy over the course of tuning.

4.6.4 Discussion of Search Efficiency

The cascaded structure of E^3-Tune is designed to match the cost profile of schedule evaluation on edge devices. Let $|\mathcal{S}|$ denote the size of the full candidate space.

Stage I performs low-cost latency inference over $|\mathcal{S}|$ candidates, Stage II performs more expensive PTX generation and power prediction only on K retained candidates, and Stage III applies hardware verification only to the final P candidates, where $P \ll K \ll |\mathcal{S}|$. As a result, the total search cost is dominated by a small subset of promising candidates rather than the entire combinatorial space.

More importantly, the staged design is aligned with the structure of the optimization objective itself. Stage I exploits the fact that energy-efficient schedules usually lie in a high-performance region, while Stage II and Stage III address the remaining ambiguity within that region by incorporating PTX-level power effects and measured runtime. In this sense, E^3-Tune does not treat performance and energy as separate optimization problems; instead, it combines them hierarchically according to both their predictive structure and their evaluation cost.

Overall, the proposed energy-aware cascaded search strategy enables E^3-Tune to approach EDP-optimal schedule selection under practical tuning budgets. By integrating inexpensive latency prediction, low-overhead physics-aware power estimation, and lightweight hardware verification, it provides a deployable alternative to both purely latency-driven search and prohibitively expensive full online energy measurement.

5 Evaluation

This section evaluates E^3-Tune from four complementary perspectives. First, we examine whether E^3-Tune improves end-to-end energy efficiency on real edge GPUs and whether the gains remain consistent across different operating frequencies. Second, we study whether the proposed cascaded search strategy preserves practical tuning cost under on-device deployment constraints. Third, we analyze whether the framework generalizes across GPU generations and frequency modes. Finally, we conduct an ablation study to quantify the contribution of the key design components in the proposed power model and search pipeline.

5.1 Experimental Settings

We implement E^3-Tune in a hybrid Python/C++ codebase. To evaluate both effectiveness and generalization, we conduct experiments on two representative edge GPU platforms: NVIDIA Jetson Orin NX (8 GB RAM) and the previous-generation NVIDIA Jetson TX2. On our primary platform, Orin NX, we run Ubuntu 22.04 with CUDA 12.6 and JetPack 6.2 (driver r36.4.3). On TX2, we use the JetPack 4.6 software stack with CUDA 10.2. Unless otherwise stated, all detailed end-to-end experiments and analyses are performed on Orin NX.

We compare E^3-Tune against two types of baselines. As industrial inference baselines, we use PyTorch v2.6.0 and TensorRT v10.3.0. As the compiler auto-tuning baseline, we use TVM/Ansor v0.19.0 with its default latency-driven tuning workflow. To ensure fairness, E^3-Tune and Ansor share the same task decomposition and the same underlying search-space definition, and both are given identical tuning budgets.

Our evaluation addresses the following questions:

- **RQ1:**Under fixed-frequency operating modes, can E^3-Tune consistently improve end-to-end energy efficiency over PyTorch, TensorRT, and latency-driven Ansor on edge GPUs?*Answered in Section 5.2.*
- **RQ2:** Under DVFS, can E^3-Tune improve energy efficiency over latency-driven Ansor by selecting frequency-specific kernels? *Answered in Section 5.3.*
- **RQ3:** Does the three-stage cascade keep tuning cost practical under edge-device constraints? *Answered in Section 5.4.*
- **RQ4:** How much do PTX semantics, auxiliary physical supervision, and gated fusion contribute to the final performance? *Answered in Section 5.5.*

5.1.1 Baselines and Measurement Protocol

For industrial baselines, we use a controlled configuration to minimize confounding factors. For TensorRT, we enforce FP32 execution by disabling FP16 and INT8, so that the comparison reflects compiler scheduling capability rather than reduced-precision acceleration. We also allow the maximum workspace size to enable aggressive kernel selection. For PyTorch, we use standard eager execution with `cudnn.benchmark` enabled. All frameworks perform 100 warm-up iterations before measurement.

For the auto-tuning baseline, we first apply TVM graph-level optimizations, including operator fusion and constant folding, before extracting tunable tasks. Ansor is then configured with a tuning budget of $1000 \times N_{\text{tasks}}$, where N_{tasks} is the number of extracted tuning tasks. E^3-Tune follows the same task partitioning and uses the corresponding hierarchical budget under its cascaded search scheme. In all experiments, we report inference latency, total energy, and EDP as the primary metrics.

5.1.2 Frequency Settings

To eliminate randomness introduced by DVFS during controlled comparison, we conduct the main experiments under locked-frequency conditions. We first fix the EMC (memory) frequency at 2.13 GHz. Although the platform supports up to 3.2 GHz, the peak setting is typically difficult to sustain under prolonged load and may induce thermal instability or divert power budget from compute units. We therefore use 2.13 GHz as a realistic steady-state memory setting. With EMC fixed, we vary only the GPU core frequency to define three representative operating points:

$$F_H = (2.13\ \text{GHz}, 918\ \text{MHz}), \quad F_M = (2.13\ \text{GHz}, 612\ \text{MHz}), \quad F_L = (2.13\ \text{GHz}, 306\ \text{MHz}).$$

We choose 918 MHz, rather than the peak 1.17 GHz, as the high-frequency point because it better reflects a sustainable operating mode under continuous workload, improving reproducibility. The medium- and low-frequency settings emulate frequency-capped deployment modes.

To avoid frequency-mismatch bias, tuning and evaluation are always performed at the same operating point. Specifically, for each $F_i \in \{F_H, F_M, F_L\}$, we independently tune and compile kernels for both E^3-Tune and Ansor, producing frequency-specific artifacts (E^3-Tune-H/M/L and A-H/M/L). This in-situ tuning protocol ensures that the reported results reflect the actual optimization target at each operating mode rather than cross-frequency reuse effects.

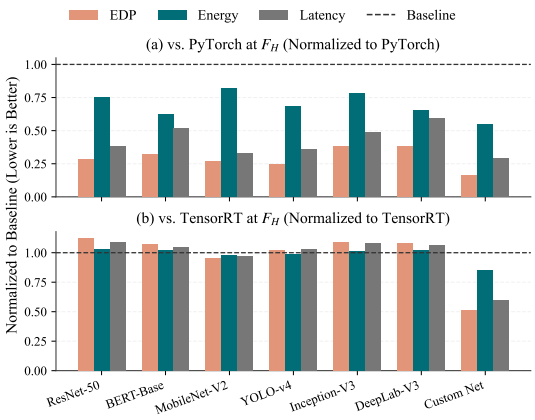

Fig. 4 E^3-Tune vs. industrial inference stacks (PyTorch and TensorRT) at F_H.

5.2 End-to-End Energy Efficiency Analysis

We first evaluate the end-to-end energy-efficiency benefits of E^3-Tune on real edge hardware. This analysis has two goals. The first is to place E^3-Tune in the context of practical deployment by comparing it with widely used industrial inference stacks. The second is to quantify the gains enabled by the proposed physics-aware search relative to the state-of-the-art latency-driven auto-scheduler Ansor.

We benchmark six representative models that cover both compute-bound and memory-bound behavior: ResNet-50 [24], BERT-Base [25], MobileNet-V2 [26], YOLO-v4 [27], Inception-V3 [28], and DeepLab-V3 [29]. To further stress-test adaptability, we also include a Custom Network containing long-tail operators and irregular topology patterns that are common in neural architecture search and domain-specific workloads.

5.2.1 Comparison with Industrial Baselines

We begin with the typical high-performance operating mode F_H. Figure 4 compares E^3-Tune against PyTorch and TensorRT under this setting. Relative to native PyTorch eager execution, E^3-Tune delivers a substantial efficiency improvement, reducing average EDP by more than 3.0×, with model-wise gains ranging from 2.6× to 6.3×. This improvement is primarily attributable to search-based code generation, which reduces framework overhead, improves operator fusion, and eliminates redundant intermediate materialization.

Against TensorRT, which represents a strong production-grade inference baseline, E^3-Tune remains highly competitive despite being fully automated. On standard models such as ResNet-50 and BERT-Base, the latency gap is limited to approximately 6%–10%. On memory-intensive workloads such as MobileNet-V2, E^3-Tune can even surpass TensorRT in EDP by roughly 7%, benefiting from more aggressive fusion and schedule specialization.

The custom network highlights a different advantage. Because it contains long-tail operators, fragmented DAG structures, and irregular tensor layouts, it lies closer to the coverage boundary of TensorRT's preset kernel library and rule-based fusion

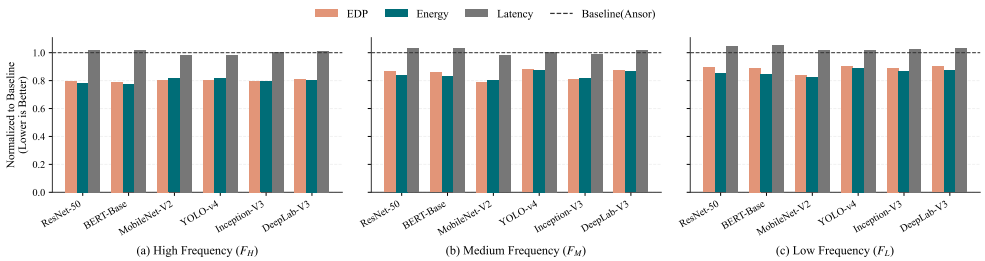

Fig. 5 End-to-end performance–energy trade-offs across frequency modes (normalized to Ansor).

mechanisms. In this case, TensorRT exhibits pronounced subgraph partitioning and fallback behavior, incurring extra intermediate materialization and possible layout-conversion overhead. In contrast, E^3-Tune can synthesize highly fused custom kernels directly from search, achieving an approximately 1.96× EDP advantage over TensorRT. These results indicate that software auto-tuning is not only competitive with industrial hand-optimized stacks on conventional workloads, but also more adaptable to rapidly evolving model structures.

5.2.2 Comparison with Ansor Across Frequency Modes

We next compare E^3-Tune with Ansor at each locked operating point under the in-situ tuning protocol. Figure 5 reports normalized EDP, energy, and latency for the six models, where each frequency tier is normalized to Ansor at the same frequency.

At the primary operating point F_H, E^3-Tune consistently improves end-to-end efficiency across all workloads. Relative to Ansor, EDP decreases by 18.7%–21.2%, with an average reduction of 19.9%, while system energy decreases by 20.1%. These gains are achieved with only small latency changes: ResNet-50 and BERT-Base incur increases of just 1.8%–2.1%, whereas MobileNet-V2 exhibits a dual gain, improving both latency and energy.

The benefits remain robust at lower frequencies. At F_M, EDP decreases by 12.1%–21.1% across models, with an average reduction of 15.3%. At F_L, the reduction remains 9.3%–16.2%, with an average of 11.3%. The corresponding energy reductions are 16.1% and 14.2%, respectively, while the average latency increases remain modest at 0.9% and 3.38%. Taken together, these results show that E^3-Tune improves energy efficiency primarily through better schedule selection and lower-power execution behavior, rather than by trading large slowdowns for reduced power.

5.2.3 Where the Gains Come From

To better understand the source of the improvements, we group workloads into compute-bound and memory-bound cases. For compute-intensive models such as BERT-Base and ResNet-50, Ansor often prefers aggressive parallelization and deeper unrolling in order to maximize throughput. While effective for latency, these schedules also increase instantaneous power density on Jetson devices. E^3-Tune instead tends to select milder tiling and unrolling strategies that avoid regimes in which additional

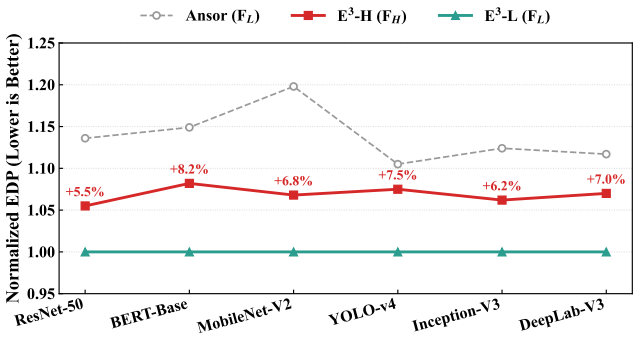

Fig. 6 Schedule divergence under cross-frequency validation.

parallelism yields only marginal performance gains but noticeably higher power. This explains why the framework can achieve significant EDP improvements with only slight latency increases.

For memory-bound workloads such as MobileNet-V2 and YOLO-v4, energy is dominated more strongly by data movement. Here, E^3-Tune tends to select schedules that reduce off-chip DRAM traffic and improve cache locality through better split, staging, and layout decisions. These schedules simultaneously reduce both memory stalls and high-energy accesses, which explains the observed cases of dual reduction in latency and energy. This distinction between compute-bound and memory-bound behavior also validates the need for PTX-aware modeling rather than latency-only tuning.

5.2.4 Cross-Platform Validation on TX2

To evaluate portability across edge GPU generations, we additionally validate E^3-Tune on NVIDIA Jetson TX2. Built on the Pascal architecture, TX2 differs substantially from Orin NX in memory hierarchy, execution behavior, and resource constraints. Following the same methodology as on Orin NX, we recollect the required data on TX2 and train a platform-specific HyP-Net. Due to TX2's tighter compute and memory budget, we evaluate a representative subset of workloads, including ResNet-50 and MobileNet-V2, and run the device in MAXN mode to reduce frequency variability.

Despite the pronounced architectural shift, E^3-Tune still achieves an average EDP reduction of 16.2% and an average energy reduction of 17.5% over Ansor. This result suggests that the proposed methodology—constraint-aware search-space construction, multimodal data collection, physics-aware modeling, and cascaded search—can be transferred across platforms, provided that the data collection and model training stages are adapted to the target device.

5.3 Analysis of Schedule Divergence Under DVFS

A natural question is whether a schedule tuned at one operating point can remain energy-efficient when the runtime frequency changes. If the same schedule were consistently optimal across frequency states, then maintaining multiple kernel versions would be unnecessary. To test this assumption, we first perform cross-frequency validation.

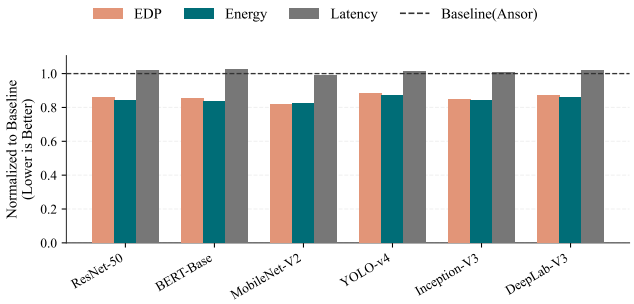

Fig. 7 DVFS: E^3-Tune-FGKD vs. Ansor.

Specifically, we execute the kernel tuned at the high-frequency operating point F_H (denoted E^3-Tune-H) under the low-frequency setting F_L, and compare it with both the latency-driven Ansor baseline evaluated at F_L and the kernel tuned directly for F_L (denoted E^3-Tune-L). As shown in Figure 6, E^3-Tune-H still achieves lower EDP than Ansor at F_L, indicating that the energy-aware schedules found by E^3-Tune remain more efficient than latency-driven schedules even after a cross-frequency shift. However, E^3-Tune-H is consistently worse than the frequency-matched E^3-Tune-L, with an average EDP gap of 5.8%–8.2%. This result reveals a clear ordering under the same low-frequency environment: E^3-Tune-L performs best, E^3-Tune-H remains competitive but suboptimal, and Ansor performs worst. Therefore, although E^3-Tune already provides an advantage over Ansor, the schedule preferred at F_H does not remain optimal when execution shifts to a lower-frequency regime. This observation shows that a single static kernel is insufficient for energy-optimal deployment across DVFS states, and thus motivates the use of multiple frequency-specific kernel versions.

This divergence reflects a shift in dominant bottlenecks across operating points. At high frequency, schedules often favor wider tiling, stronger parallel exposure, and more aggressive loop transformations to hide memory latency and expose instruction-level parallelism. Such choices are beneficial when the GPU core can sustain a high execution rate. However, when the frequency is reduced, the same schedules may suffer from excessive register pressure, heavier instruction streams, or ineffective resource utilization, making them less energy-efficient. Conversely, schedules tuned for lower-frequency regions tend to prefer lighter unrolling and more compact thread binding, which reduce contention and resource pressure at F_L but may fail to fully exploit the available compute capability at F_H. These observations explain why kernel optimality diverges across frequency states and directly motivate the multi-version deployment strategy of E^3-Tune.

Based on this observation, E^3-Tune adopts a DVFS-aware runtime strategy, named Frequency-Guided Kernel Dispatch (FGKD). Rather than relying on a single statically selected kernel, FGKD generates frequency-specific kernel versions offline for representative operating regions, e.g., K_H, K_M, and K_L, corresponding to high-, medium-, and low-frequency ranges. During deployment, the runtime monitors the current GPU frequency state and dispatches the matching pre-compiled kernel version.

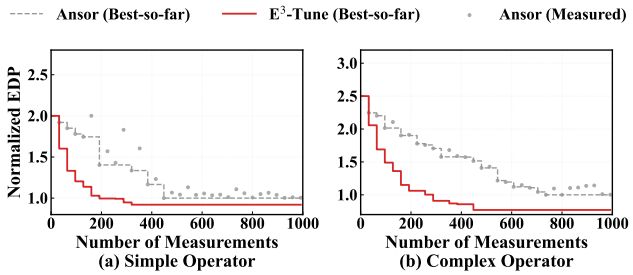

Fig. 8 Convergence of best-so-far EDP for E^3-Tune and Ansor on simple and complex operators.

Figure 7 evaluates this strategy under DVFS and compares it with latency-driven Ansor in the same runtime environment. The results are normalized to Ansor-DVFS, where lower values indicate better efficiency. Across the six workloads, FGKD reduces average EDP by 14.4% and average energy consumption by 15.5%, while introducing only a small average latency change of 1.3%. The EDP reduction ranges from 11.8% on YOLO-v4 to 18.2% on MobileNet-V2, showing that the benefit is consistent across both compute-intensive and memory-sensitive workloads. MobileNet-V2 achieves simultaneous reductions in latency, energy, and EDP, which indicates that frequency-aware kernel selection can also improve execution efficiency for memory-bound operators rather than simply trading latency for lower power.

These results show that FGKD preserves the energy-efficiency benefit of frequency-specific tuning when the device frequency changes dynamically. Compared with Ansor, whose schedules are selected according to latency-oriented objectives, FGKD selects kernels according to the current DVFS state and the EDP-oriented optimization objective of E^3-Tune. As a result, E^3-Tune maintains lower energy consumption and EDP under dynamic frequency variation, while keeping latency close to the Ansor baseline.

An important practical concern is whether FGKD introduces noticeable runtime overhead due to frequency monitoring and multi-version kernel switching. In E^3-Tune, this overhead is small for two reasons. First, all candidate kernel versions are generated and compiled offline, so runtime adaptation only requires selecting a different function entry from a dispatch table. It does not trigger recompilation, online tuning, or code regeneration. Second, frequency detection can be implemented with coarse-grained monitoring rather than fine-grained per-kernel or per-instruction sampling. Since DVFS transitions occur at a much slower timescale than instruction execution, the runtime only needs to periodically read the current frequency state or query the corresponding operating region before launching a batch of kernels. In addition, a simple hysteresis rule is used to avoid oscillatory switching near frequency boundaries. As a result, the overhead of FGKD is limited to lightweight state checking and dispatch-table lookup, which is negligible compared with end-to-end inference cost, while the energy-efficiency benefit remains significant.

5.4 Search Cost and Feasibility Analysis

We next evaluate whether the proposed energy-aware search remains practical on edge devices. Because schedule evaluation has a strongly imbalanced cost structure—IR

inference $\ll$ PTX compilation $\ll$ on-device measurement—E^3-Tune confines expensive operations to a small subset of candidates through the cascaded funnel. Unless otherwise stated, these experiments are conducted on Jetson Orin NX with default settings $K = 200$ and $P = 32$. Following Ansor, the maximum on-device measurement batch size is 32, and the measurement budget is 1000 trials per operator.

5.4.1 Sample Efficiency and Convergence

To assess sample efficiency, we plot the best-so-far EDP against the number of on-device measurements. Figure 8 compares E^3-Tune and Ansor on both simple and complex operators. Since Ansor is optimized for latency rather than energy, the EDP of its measured candidates exhibits substantial variance, shown by the scattered gray points. In contrast, the convergence curves of E^3-Tune are markedly steeper.

For simple operators, both methods eventually reach similar final EDP, but E^3-Tune enters a stable regime within approximately 200 measurements, whereas Ansor generally requires more than 400. The difference is more pronounced on complex operators: Ansor often needs roughly 750 measurements before approaching its stable region, while E^3-Tune converges within about 450 measurements and reaches a lower final EDP, with normalized values below 0.8. These results indicate that PTX-aware re-ranking effectively suppresses the exploration of high-performance but high-power schedules early in the search process, thereby steering the measurement budget toward the energy Pareto frontier.

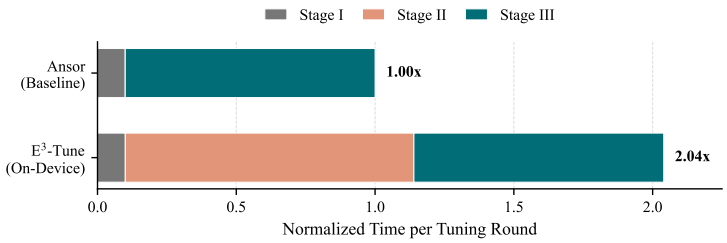

Fig. 9 Normalized time breakdown per tuning round for Ansor and E^3-Tune.

5.4.2 End-to-End Tuning Time Breakdown

We then analyze the end-to-end tuning overhead. Figure 9 reports the normalized per-round time breakdown under an on-device deployment setting, where Ansor's per-round tuning time is normalized to 1.0. Because all compilation is performed locally on the ARM CPU of Jetson Orin NX, Stage II, which includes PTX generation and feature extraction, becomes the dominant overhead in E^3-Tune. As a result, the per-round cost of E^3-Tune is higher than that of Ansor.

However, the more relevant metric is total time to convergence. Because E^3-Tune reaches high-quality solutions with fewer measurements and supports an early-stopping rule when best-so-far EDP plateaus, it avoids many late-stage low-yield

Table 2 Ablation study of E^3-Tune on Jetson Orin NX at F_H.

Variant	EDP Reduction ↑	Energy Saving ↑	MAE ↓
E^3-Tune (Full)	**19.9%**	**20.8%**	**6.3%**
w/o PTX Semantics	13.0%	14.3%	12.4%
w/o Auxiliary Tasks	15.7%	17.6%	9.8%
w/o Gated Fusion	14.9%	16.9%	9.2%

iterations that are common in latency-driven search. Across the evaluated networks, the total tuning time of E^3-Tune is only about 1.19×–1.32× that of Ansor. This indicates that microarchitecture-aware energy optimization can be achieved with modest overhead while yielding substantially better end-to-end efficiency.

5.4.3 Sensitivity to Funnel Width

Finally, we study the sensitivity of the search to the funnel parameters K and P, which control the trade-off between search breadth and feedback frequency. Increasing K broadens the number of candidates exposed to PTX-aware evaluation and can recover energy-efficient schedules that would otherwise be discarded by the latency filter, but it also increases PTX compilation cost approximately linearly. In contrast, P controls the amount of hardware feedback per round. A smaller P produces more frequent updates and may improve measurement efficiency, but it increases the number of rounds. A larger P improves batching efficiency but may introduce what we refer to as a *batch lag effect*, where many measurements are consumed before the latency model receives fresh supervision.

In practice, the values of K and P should be chosen according to the available time and hardware budget. Throughout the main experiments, we use $K = 200$ and $P = 32$, which provide a balanced trade-off between search overhead and final solution quality.

5.5 Ablation Study and Component Contribution

We finally evaluate the contribution of the main design components in E^3-Tune, namely PTX semantics, auxiliary physical supervision, and gated fusion. All ablation experiments are performed on Jetson Orin NX at the high-frequency operating point F_H. To reduce variance, all variants follow the same protocol as in Section 5.2, including the same search budget and the same cascade parameters. Table 2 reports the resulting EDP and energy gains together with the power-model mean absolute error (MAE).

Overall, the ablation results reveal a clear relationship between prediction fidelity and end-to-end energy-efficiency gain. The full system achieves both the largest EDP reduction (19.9%) and the lowest power-prediction error (6.3% MAE), indicating that accurate and physically consistent power estimation is critical to the quality of energy-aware search.

Effect of PTX semantics. Removing PTX semantics eliminates the instruction-sequence information used to capture microarchitectural behavior beyond aggregated

statistics. As a result, the MAE increases to 12.4%, and the end-to-end EDP reduction drops to 13.0%. This confirms that PTX semantics are essential for distinguishing schedules that appear similar at the IR level but differ in compiled low-level behavior.

Effect of auxiliary physical supervision. Removing auxiliary tasks weakens the model's ability to capture the distinction between compute-dominated and memory-dominated schedules. In this setting, the MAE rises to 9.8%, and the EDP reduction decreases to 15.7%. This result supports the role of SM activity and EMC utilization as useful physical supervision signals rather than merely auxiliary prediction targets.

Effect of gated fusion. Removing the gating mechanism reduces the model's ability to adaptively balance structured statistical cues against semantic features. This raises the MAE to 9.2% and reduces the EDP gain to 14.9%. The result indicates that residual correction should not be applied uniformly; instead, adaptive fusion is important for preserving robustness across different operators and execution regimes.

In summary, lower MAE consistently corresponds to larger end-to-end efficiency gains. This alignment supports the overall design of E^3-Tune as a physics-aware closed loop that combines instruction semantics, physical supervision, and adaptive fusion for practical EDP-oriented tuning.

6 Conclusion and Future Work

This paper presented E^3-Tune, an energy-efficiency-aware auto-tuning framework for edge GPUs. By combining offline physics-aware modeling with an online cascaded search strategy, E^3-Tune extends tensor compilation beyond latency-only objectives toward joint latency–energy optimization using EDP as the target criterion. The proposed design avoids expensive and noise-sensitive online power sampling by replacing it with low-overhead power prediction, while still retaining sensitivity to PTX-level microarchitectural behavior.

Experimental results on diverse CNN and Transformer workloads show that E^3-Tune consistently improves end-to-end energy efficiency over the latency-driven Ansor baseline, achieving substantial reductions in both energy consumption and EDP with only modest latency changes. Additional evaluations across multiple operating points further show that schedules optimized for one frequency mode do not remain uniformly optimal across the DVFS range, which highlights the importance of frequency-specific tuning and multi-version deployment on edge devices. Results on a second edge GPU platform also suggest that the overall methodology can generalize across hardware generations when the profiling and modeling stages are adapted to the target device.

Despite these promising results, the current framework still has several limitations. First, E^3-Tune currently relies on NVIDIA-specific low-level representations and telemetry signals, including PTX and platform-specific hardware monitoring interfaces, which may limit portability to other GPU ecosystems. Second, although the online search overhead is kept practical, the offline profiling and dataset construction process still introduces a non-trivial cold-start cost when adapting the framework to a new hardware platform. Third, the current design focuses primarily on intra-operator

scheduling, and therefore does not yet capture graph-level opportunities such as cross-operator fusion, global memory planning, or end-to-end pipeline co-optimization.

There are several promising directions for future work. One direction is to extend E^3-Tune from operator-level optimization to graph-level co-optimization, so that compiler scheduling and end-to-end execution planning can be considered jointly. Another direction is to investigate cloud–edge collaborative tuning, where heavy feature extraction and model updating are performed in the cloud while lightweight predictors are distilled and deployed on edge devices. In addition, it would be valuable to explore large-language-model-assisted warm-start strategies for schedule generation, seed selection, and pruning-rule synthesis, with the goal of reducing the number of on-device measurements required to approach the energy Pareto frontier. More broadly, we believe that integrating compiler optimization with physically informed energy modeling is a promising direction for enabling efficient and robust deep learning deployment on future edge accelerators.

Acknowledgements. The authors gratefully acknowledge the support from the Tianjin Major Special Project for Artificial Intelligence Science and Technology (25ZXRGGX00100).

Funding. This work was supported by the Tianjin Major Special Project for Artificial Intelligence Science and Technology (25ZXRGGX00100).

Author contributions. Conceptualization was done by M.S. and Q.S.; methodology was performed by M.S., H.Z., and Y.X.; software was done by M.S. and J.W.; validation was done by M.S., H.Z., Y.X., and J.W.; formal analysis was done by M.S., H.Z., and Y.X.; investigation was done by M.S. and H.Z.; resources were provided by Q.S.; data curation was done by M.S. and J.W.; writing–original draft preparation was done by M.S.; writing–review and editing was done by H.Z., Y.X., J.W., and Q.S.; visualization was done by M.S. and J.W.; supervision, project administration, and funding acquisition were done by Q.S. All authors read and approved the final manuscript.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

[1] NVIDIA Corporation: NVIDIA Jetson Systems for Autonomous Machines. Accessed 10 May 2026 (2026). https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[2] NVIDIA Corporation: Jetson Linux Developer Guide: Power Management. Accessed 10 May 2026 (2024). https://docs.nvidia.com/jetson/archives/r35.4.1/DeveloperGuide/text/SD/PlatformPowerAndPerformance/JetsonOrinNxSeriesAndJetsonOrinNanoSeries.html

[3] Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, vol. 28, pp. 1135–1143 (2015)

[4] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2704–2713 (2018). https://doi.org/10.1109/CVPR.2018.00286

[5] You, J., Chung, J.-W., Chowdhury, M.: Zeus: Understanding and optimizing GPU energy consumption of DNN training. In: Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation, pp. 119–139 (2023). https://www.usenix.org/conference/nsdi23/presentation/you

[6] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: TVM: An automated end-to-end optimizing compiler for deep learning. In: Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, pp. 578–594 (2018). https://www.usenix.org/conference/osdi18/presentation/chen

[7] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., Amarasinghe, S.: Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 519–530 (2013). https://doi.org/10.1145/2491956.2462176

[8] Vasilache, N., Zinenko, O., Theodoridis, T., Goyal, P., DeVito, Z., Moses, W.S., Verdoolaege, S., Adams, A., Cohen, A.: Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions (2018)

[9] Zheng, S., Liang, Y., Wang, S., Chen, R., Sheng, K.: Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous systems. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 859–873 (2020). https://doi.org/10.1145/3373376.3378508

[10] Rafi, M.A., Chau, K., Jeon, H.: Uncovering detailed power characterizations of GPUs on edge platforms. In: Proceedings of the 17th Workshop on General Purpose Processing Using GPU, pp. 48–54 (2025). https://doi.org/10.1145/3725798.3725806

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)

[12] Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings

of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016). https://doi.org/10.1145/2939672.2939785

[13] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y.: LightGBM: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems, vol. 30, pp. 3146–3154 (2017)

[14] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA (1989)

[15] Faqir-Rhazoui, Y., García, C.: SYCL in the edge: Performance and energy evaluation for heterogeneous acceleration. The Journal of Supercomputing **80**, 14203–14223 (2024) https://doi.org/10.1007/s11227-024-05957-6

[16] Martin-Salinas, I., Badia, J.M., Valls, O., Leon, G., Amor, R., Belloch, J.A., Amor-Martin, A., Naranjo, V.: Evaluating and accelerating vision transformers on GPU-based embedded edge AI systems. The Journal of Supercomputing **81**, 349 (2025) https://doi.org/10.1007/s11227-024-06807-1

[17] Cámara, J., Cuenca, J., Galindo, V., Vicente, A., Boratto, M.: An autotuning approach to select the inter-GPU communication library on heterogeneous systems. The Journal of Supercomputing **81**, 283 (2025) https://doi.org/10.1007/s11227-024-06794-3

[18] Dou, H., Zhu, S., Zhang, Y., Chen, P., Zheng, Z.: HyperTuner: A cross-layer multi-objective hyperparameter auto-tuning framework for data analytic services. The Journal of Supercomputing **80**, 17460–17491 (2024) https://doi.org/10.1007/s11227-024-06123-8

[19] Zheng, L., Liu, R., Shao, J., Chen, T., Gonzalez, J.E., Stoica, I., Haj-Ali, A.: TenSet: A large-scale program performance dataset for learned tensor compilers. In: Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (2021)

[20] NVIDIA Corporation: Tegrastats Utility: Jetson Linux Developer Guide. (2023). Accessed 10 May 2026. https://docs.nvidia.com/jetson/archives/r35.4.1/DeveloperGuide/text/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html

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

[22] Burges, C.J.C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of the

22nd International Conference on Machine Learning, pp. 89–96 (2005). https://doi.org/10.1145/1102351.1102363

[23] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: Proceedings of the 7th International Conference on Learning Representations (2019). https://openreview.net/forum?id=Bkg6RiCqY7

[24] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

[25] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 4171–4186 (2019). https://doi.org/10.18653/v1/N19-1423

[26] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4510–4520 (2018). https://doi.org/10.1109/CVPR.2018.00474

[27] Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y.M.: YOLOv4: Optimal Speed and Accuracy of Object Detection (2020)

[28] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016). https://doi.org/10.1109/CVPR.2016.308

[29] Chen, L.-C., Papandreou, G., Schroff, F., Adam, H.: Rethinking Atrous Convolution for Semantic Image Segmentation (2017)