CPM: A Cross-layer Power Management Facility to Enable QoS-Aware AIoT Systems

Xiaofeng Hou, Peng Tang, Tongqiao Xu, Cheng Xu, Chao Li, Minyi Guo
{xfhelen, tttppp, tqxu23, jerryxu}@sjtu.edu.cn; {lichao, guo-my}@cs.sjtu.edu.cn
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China

***Abstract*—With the rapid progress and widespread adoption of AI technology, integrating powerful DNN models into AIoT devices in close proximity to users has become increasingly appealing. However, a significant challenge is that it is not easy to achieve the stringent Quality of Service (QoS) standards, especially in terms of real-time latency, demanded by the computationally intensive DNN workloads in energy-limited AIoT environments. To address this challenge, prior research has focused on per-layer power management techniques, which aggressively exploit the unique energy and performance relationships exhibited by each layer of the DNN at an exceedingly fine-grained control granularity. In this study, we identify the limitations of the existing per-layer DVFS mechanisms. They severely overlook the significant DVFS overhead caused by the excessively fine-grained control which can introduce complexity to power management in practical scenarios, consequently deteriorating QoS. To mitigate these challenges, we propose CPM, a Cross-layer Power Mangement facility which automatically modularizes different DNN layers and performs the best DVFS policy, thereby enhancing QoS by ensuring lower latency in real-time AIoT systems. Additionally, we integrate CPM into mainstream commercial AIoT boards and systems to validate its effusiveness. The results show that CPM can reduce the execution latency by up to 45.76% while improving the energy efficiency by up to 31.58% of real AIoT systems compared to SOTA per-layer power management methods.**

I. Introduction

In recent years, deep learning neural networks (DNNs) have been increasingly applied for many real-time AIoT systems or edge microdatacenters [1]–[3] to achieve autonomy at the user end. For example, by using DNN models, intelligent voice assistants can provide smarter speech recognition services to users. In addition, many chip inventors have also released various AIoT devices such as NVIDIA Drive AGX to accelerate the deployment of DNN-driven autonomy at the edge in practice. However, AIoT systems face challenges in achieving a balance between high QoS, with a focus on reducing latency here, and energy efficiency, all within the constraints of limited resources. This balance is critical for maintaining user satisfaction and operational efficiency in AIoT applications, where any compromise on latency can significantly detract from the user experience, and energy inefficiency can hinder the system's sustainability and practical deployment.

One of the most effective approaches to this challenge that has emerged recently is to implement fine-grained power management based on dynamic voltage and frequency scaling (DVFS) techniques [1], [4], [5]. For example, ApNet [5], Predjoule [4] and NeuOS [1] are representative research efforts that leverage per-layer DVFS to guarantee inference latency while minimizing energy consumption for different AIoT systems. These approaches exploit a unique feature of DNN workloads in that they consist of multiple layers, each of which exhibits very different energy usage patterns and characteristics. Since the characterizations are independent for each layer, it is relatively straightforward to identify suitable frequency configurations for individual layers. Thus, these approaches always use handcrafted methods to assign frequency manually.

However, the benefits of per-layer DVFS mechanisms are grossly overestimated due to the disregard for the high DVFS overhead resulting from excessively fine-grained control. The reasons behind this are two-fold. *First, existing approaches aggressively overlook the control overheads of DVFS in real AIoT systems, which in turn precludes them from performing the most efficient designs.* Plenty of previous works have demonstrated that existing DVFS techniques involve a control latency ranging from tens to hundreds of milliseconds due to intricate and time-consuming runtime power resume processes [6], [7]. Although this control latency may seem negligible for a complete DNN inference task, it becomes significant for each individual layer that is typically completed within a few hundred microseconds to a few milliseconds. Our measurements show that the DVFS transition overhead can be up to 28X higher than the execution overhead of most layers (accounting for over 50% of the total). Thus, ignoring the control overhead inevitably leads to an inflated estimation of energy efficiency in AIoT systems.

Second, the presence of DVFS overhead disrupts the independence of power management between layers and significantly increases the complexity of the configuration space. Under ideal circumstances without DVFS overhead, it is possible to profile the characteristics of each layer and select an optimal frequency for each [1], [4]. However, when DVFS latency overhead is taken into account, this strategy becomes inadequate. For instance, assuming the optimization objective is to minimize execution latency, traditionally, each layer would select a DVFS configuration based on its minimum execution latency. However, when considering DVFS switching latency, this choice may lead to frequent DVFS switching in subsequent layers, resulting in high DVFS overhead. In such cases, global optimization and decision-making, considering DVFS overhead, become necessary, leading to an exponential growth of the configuration space [7], [8]. Moreover, real-world AIoT scenarios often involve multiple conflicting objectives and input dynamicity, further exacerbating the complexity and rendering traditional handcrafted solutions inefficient and impractical.

In this paper, we delve into the limitations of existing approaches and shed light on the significance and challenges of accounting for DVFS overhead. Based on these findings, instead of optimizing on a per-layer basis, we propose to exploit fine-grained DNN features at a cross-layer level to make a more reasonable tradeoff on the control overheads and benefits. Therefore, we propose CPM, a cross-layer power management facility to enable QoS-aware AIoT systems to balance latency reduction and energy consumption with automated global optimization while accounting for DVFS overheads. We design CPM with several features to overcome the challenges in the existing methods. It consists of two main components, the *CPM Engine* and the *CPM Controller*. The CPM Engine is responsible for navigating and determining cross-layer DVFS decisions. It incorporates various sub-modules that model the global latency and energy relationship under different cross-layer DVFS configurations, enabling effective identification of optimal configurations using automated approaches. On the other hand, the CPM Controller facilitates interaction between the system and the CPM Engine, ensuring that optimization decisions can be executed in real-world AIoT systems with multiple objectives and input resolutions. Additionally, we construct an equivalent agent of real AIoT systems to accelerate the training and implementation process of CPM.

Overall, we make the following contributions:

1) We identify the substantial DVFS overhead in the existing fine-grained power management approaches for per-layer DNN optimization. We verify this by real-world measurements and highlight the importance of considering this overhead in power management.
2) We design and implement CPM, a cross-layer power management policy tailored to highly-efficient AIoT systems. It enables the exploration of the inherent characteristics of DNNs and achieves a better tradeoff between the DVFS control overhead with the benefits of fine-grained control.
3) We develop a prototype of CPM by seamlessly integrating it into the widely-adopted Jetson AIoT systems. Through rigorous validation using multiple representative DNN workloads, we provide empirical evidence showcasing the superior performance of CPM over traditional methods in terms of energy efficiency and QoS.

II. Challenges and Motivations

In recent decades, DNN models have been pervasive in different AIoT applications. These models follow a layer-by-layer execution pattern, where input data sequentially passes through each layer, with the output of one layer becoming the input for the next until the desired output is generated. The layer-by-layer execution characteristic of DNN models offers many potential advantages [5], [9]. For instance, it allows for fine-grained optimization at each layer, a feature particularly valuable when deploying DNN models on AIoT systems with limited computational resources. This level of optimization ensures efficient use of available resources while preserving the model's overall accuracy and functionality. Consequently, the

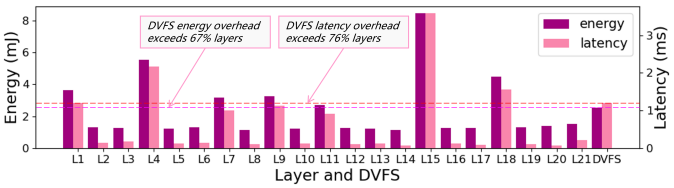


Figure 1: Overhead comparison between DVFS and execution.

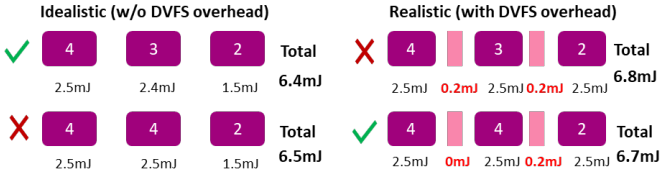


Figure 2: Existing methods overlook the DVFS overhead.

layered structure of DNNs not only enables sophisticated data processing but also facilitates their integration into resource-constrained environments, thereby enhancing their applicability across various real-world scenarios. However, achieving truly fine-grained optimization presents several challenges.

A. Challenge-I: Tedious Power Control Overhead

The intricately fine-grained optimization, specifically the per-layer power control currently employed, introduces a significant DVFS transition overhead that should not be underestimated. To provide empirical evidence, we conduct a comprehensive analysis to profile the per-layer execution latency and energy consumption of the `AlexNet` model. Additionally, we compare these metrics with those resulting from a single DVFS transition. We use the `INA3221 power monitor` [10] to collect power consumption and the `Python time module` to measure the execution latency. To ensure accuracy, we repeat these measurements 100 times and calculate the average results. The results are presented in Figure 1. Our findings indicate that the DVFS transition incurs several milliseconds of latency and results in the wastage of several millijoules of energy. Remarkably, this overhead exceeds the latency and energy of the 76% and 67% layers respectively.

Influence of Overhead on Optimality: The observed DVFS transition overhead significantly affects the optimality of power management strategies and deteriorates QoS in real-time scenarios for long latency overhead. In Figure 2, we illustrate how the existing methods overestimate their energy efficiency without considering DVFS overhead. We construct a simple example, that consists of a three-layer neural network. Without considering the overhead, the left scheme that switches frequencies frequently to obtain better execution performance (i.e., the three layers are executed at frequencies of 4, 3, and 2 respectively) appears to be less energy-intensive. However, when DVFS control overhead is taken into account, we find that the right strategy, with selective reduction of switching (i.e., the three layers executed at frequencies of 4, 4, and 2, respectively), consumes less energy. This indicates that disregarding DVFS overhead renders previous per-layer power

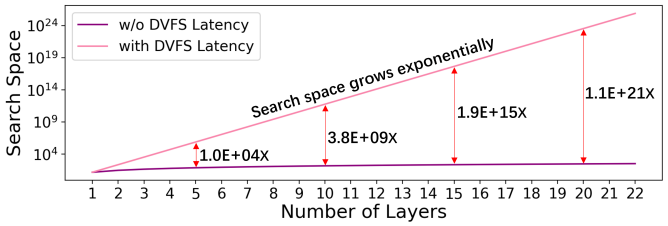


Figure 3: DVFS's impact on configuration space.

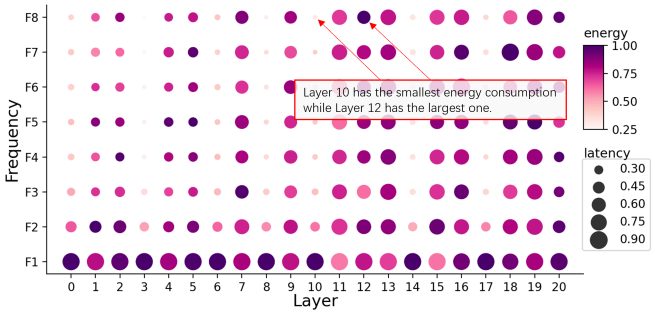


Figure 4: Characterization of per-layer energy usage pattern.

management approaches invalid, emphasizing the need to address this challenge effectively.

Control Dependence Arising from Overhead: Furthermore, the presence of DVFS overhead significantly increases the complexity and size of the configuration space. Under ideal circumstances without DVFS overhead, it is possible to profile the characteristics of each layer and select an optimal frequency for each. However, when DVFS latency overhead is taken into account, this strategy becomes inadequate, leading to exponential growth in the configuration space. Figure 3 visually demonstrates the substantial increase in the configuration space when DVFS latency is considered compared to when it is not. In the absence of DVFS latency considerations, the configuration space is represented by $C \times N$. However, with DVFS latency included, the configuration space exponentially grows to C^N, where C denotes the number of DVFS configurations (15 in this instance) and N represents the number of layers. For instance, in a simple 5-layer neural network, the configuration space becomes 10^4 times larger with DVFS latency. In the case of a 20-layer network, the configuration space expands to a staggering $1.1 * 10^{21}$ times larger than without DVFS latency. As the number of network layers keeps increasing, the enormity of the configuration space, exacerbated by DVFS latency, renders manual approaches impractical.

Takeaways: Prior per-layer power management techniques exploit DNN characteristics at a fine-grained control granularity but overlook the DVFS control overhead, severely limiting their benefits. To leverage the inherent characteristics of DNNs, a more adaptive granularity, such as cross-layer optimization, is crucial. Implementing fine-grained power management for AIoT systems is complex, rendering manual approaches impractical. Therefore, machine learning approaches are needed to discover efficient power management schemes.

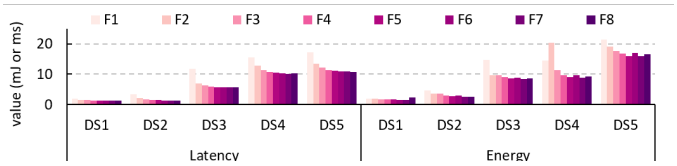


Figure 5: Execution latency and energy of the 2^{nd} layer in `Yolo` under different data size.

B. Challenge-II: Complex Power Configuration Space

The complex configuration space will be further exacerbated by multiple conflicting objectives and input dynamicity in real-world AIoT systems. In Figure 4, we measure the energy usage patterns of different layers in `AlexNet` on a `Jetson Xavier NX board`. When simply looking at the characteristics of each layer, it is clear that different layers exhibit different energy-performance correlations at different DVFS configurations. For example, for `Layer 0`, both its execution latency and energy are less sensitive to frequency changes, while the results of `Layer 8` are heavily affected by frequency changes. In this regard, achieving global optimization of power consumption across different layers becomes a complex task, and this complexity is further compounded by the simultaneous presence of multiple objectives and input dynamicity.

Multiple Conflicting Optimization Objectives: In the research field of power management, the target always comes with multiple conflicting goals, such as minimizing energy consumption and reducing latency. This presents an additional challenge for power management schemes. As depicted in Figure 4, the optimal configuration for one layer may result in suboptimal performance for another layer. For example, while `Layer 10` exhibits the lowest execution latency under the `F8` configuration, several other layers, such as `Layers 11, 12, & 18`, experience longer latency. Real-world applications require not only energy or latency minimization but also achieving a specific latency target while minimizing energy consumption. This inherent trade-off adds to the complexity of the problem. Furthermore, different applications have varying requirements. For instance, autonomous driving may necessitate a response time of 10 milliseconds, while an intelligent assistant may tolerate a latency of 1000 milliseconds. These divergent goals further contribute to the need for adaptive power management configurations.

Adaptability to Input Dynamicity: AIoT systems typically operate on domain-specific DNN workloads, where the behavior is primarily influenced by changes in input data rather than other factors like resource interference. Consequently, input data variability plays a crucial role. In Figure 5, we evaluate energy and performance across different DVFS configurations for a layer of the YOLO network under various input sizes, including 64*64 `(DS1)`, 224*224 `(DS2)`, 320*320 `(DS3)`, 448*448 `(DS4)` and 512*512 `(DS5)`. The results demonstrate that the energy-performance characteristics of a single-layer network vary under different input data scenarios. Moreover, real-world AI-enabled systems may encounter unknown hardware overheads that necessitate consideration in power management

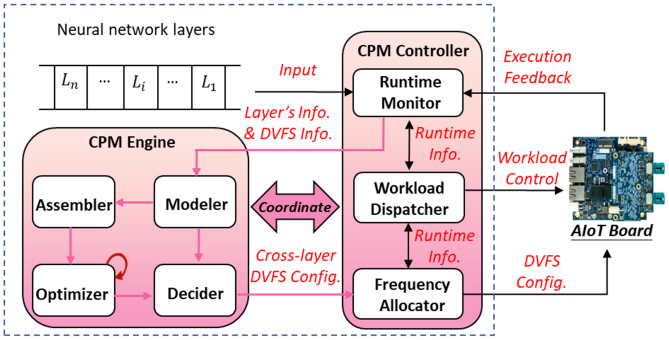


Figure 6: Overview of CPM.

decisions. Therefore, the adaptability to input dynamicity is an essential aspect that should also be taken into consideration.

Takeaways: Power management in AIoT scenarios poses conflicts among objectives, requiring an effective approach for multi-objective optimization. In addition, real-world AIoT systems are dynamic, and variations in input data affect power consumption characteristics. A robust power management solution should consider this dynamicity to adapt better to applications' characteristics.

C. Design Considerations

Drawing from the aforementioned challenges, the following design considerations are proposed to facilitate efficient power management in AIoT systems:

1) **Accounting for Fine-grained Power Control Overhead:** The power management system should effectively address the overhead associated with fine-grained power control, i.e., DVFS overhead. By incorporating this overhead into the optimization process, the system can identify global optimal solutions, thus enabling efficient power management while mitigating the impact of DVFS overhead in AIoT systems.
2) **Being Adaptable to Handle Complexity:** Power management policies should possess the capability to handle the inherent complexity arising from DVFS dependencies. Additionally, they should exhibit adaptability to accommodate conflicting objectives and dynamic inputs. This adaptability ensures that the power management solution can optimize power consumption effectively while considering the dynamic nature of AIoT applications.

III. CPM: Cross-layer Power Management

A. Overview of CPM

In this paper, we introduce CPM, a cross-layer power management solution tailored for highly efficient AIoT systems. Its core functions encompass monitoring and collecting runtime information from workloads (DNN models) running on the AIoT board. It dynamically optimizes and determines the DVFS configuration to achieve an optimal balance between power efficiency and execution latency while considering the power control overhead. Figure 6 provides an illustration of the CPM architecture. CPM consists of two main modules:

1) **CPM Engine:** The CPM Engine handles cross-layer DVFS decisions, taking into account the DVFS transition overhead in the power management process. It comprises four components: Modeler, Assembler, Optimizer, and Decider. The Modeler constructs an analytical model using runtime information acquired from the Runtime Monitor to optimize the DVFS control during DNN model execution. The Assembler merges DNN model layers into blocks based on latency characteristics and DVFS overhead. The Optimizer employs an autoencoder to map the configuration space to a latent space, reducing the configuration search space and accelerating the search for optimal configurations. The Decider ultimately finds the optimal DVFS configurations utilizing the Bayesian optimization algorithm in an automatic way.
2) **CPM Controller:** The CPM Controller facilitates system interaction with the CPM Engine, ensuring the proper execution of optimization decisions. It comprises three components: Runtime Monitor, Workload Dispatcher, and Frequency Allocator. The Runtime Monitor communicates runtime information and collects execution feedback from the system through system tools, forwarding the data to the CPM Engine for analysis and optimization. The Workload Dispatcher communicates runtime information with both the Runtime Monitor and the Frequency Allocator, controlling workload execution based on received information. The Frequency Allocator receives the DVFS configuration from the Decider within the CPM Engine and adjusts the GPU frequency of the AIoT board using the existing DVFS control interface.

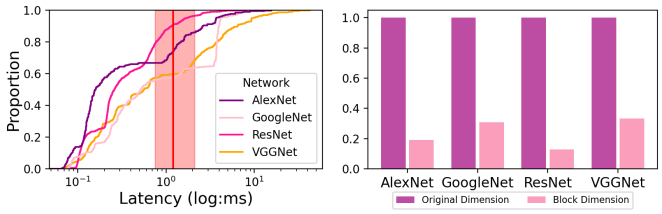
CPM is a novel power management facility specifically designed for highly efficient AIoT systems, while maintaining good QoS. Notably, CPM is compatible with various mainstream AIoT platforms, including the `Nvidia Jetson board`, making it easily adaptable and deployable across diverse AIoT applications.

B. The Design of CPM Engine

The CPM Engine is responsible for determining cross-layer DVFS decisions while accounting for the DVFS transition overhead in the power management process. Its main functionalities are described as follows.

Modeler: To evaluate the execution latency and energy consumption of DNN models across a range of cross-layer designs and frequency configurations, we develop the Modeler sub-module. Within the Modeler, we set several parameters to represent different cross-layer designs and DVFS frequencies that enable us to compute the execution latency and energy consumption of DNNs under different scenarios. In essence, the Modeler incorporates the consideration of DVFS overhead and introduces a comprehensive model that accurately reflects the execution latency and energy consumption under diverse cross-layer designs and power management mechanisms.

To be specific, we consider a DNN model $\mathcal{D}$ consisting of n layers, denoted as $\mathcal{D} = l_1, l_2, ..., l_n$. In the case of any fusion operation [11], we treat the fused layers as a single layer.



(a) Latency distribution (b) Effect of blocking

Figure 7: Priori knowledge-based blocking optimization.

We also consider k available frequency options, represented as $\mathcal{F} = f_i, \cdots, f_k$. Each layer $l_i \in \mathcal{D}$ is assigned a corresponding frequency $f_j \in \mathcal{F}$. To accurately execute these DNN layers and account for the overhead of DVFS, we introduce parameters t_i^j and e_i^j to represent the latency and energy of layer l_i under the selected frequency f_j. The overall objective is to find a mapping $S(l_i) = f_j$ that assigns a frequency to each layer l_i. Consequently, the execution delay of the DNN model under a specific cross-layer configuration and frequency sequence is calculated as:

$$T = \sum_{i=1}^{n} (t_i^{S(l_i)} + BT_i * t_{trans}) \tag{1}$$

Similarly, the total energy consumption is given by:

$$E = \sum_{i=1}^{n} (e_i^{S(l_i)} + BT_i * e_{trans}); \tag{2}$$

Here, BT_i is set to 1 if there is a change in the frequency for layer l_i. Conversely, if there is no change in frequency, BT_i is set to 0, indicating that l_i is a cross-layer design. Additionally, t_{trans} and e_{trans} represent the latency and energy associated with DVFS transition. By utilizing Modeler's capabilities, we can gain valuable insights into the performance characteristics of DNN models, thereby helping us make better decisions of enhancing AIoT systems' QoS and lowering energy use.

Assembler: As discussed in II-A, the overhead of DVFS transition introduces interdependencies among different DNN layers, resulting in a significantly expanded configuration space as the number of layers increases. While the Modeler is capable of encompassing various cross-layer designs, the resulting modeled configuration space becomes highly complex, posing challenges in finding an optimal solution. To address this issue, we develop the Assembler sub-module, which leverages a priori knowledge to reduce the configuration space by merging the layers of a DNN model into blocks.

To begin, we analyze the distribution of execution latencies in several representative DNN models (the details of the setup can be found in Section IV). Figure 7 (a) illustrates that the latency distribution across these neural networks exhibits similarities, with a significant portion of layers executing quickly (less than 1 ms). We also plot the measured DVFS transition latency as the vertical red line and the shadow area represent the range of transition latency. It's quite obvious the DVFS transition delay has exceeded the execution delay of many layers. To reduce the search space, we propose to combine layers with running times comparable to or even lower than the DVFS latency. To be specific, we assign a sequence of frequency block $\mathcal{B} = \{b_1, b_2, ..., b_m\}$ for the execution of DNN (while $m = n$ this reduce to the layer-by-layer DVFS paradigm). For each layer in a block, b_i is defined as,

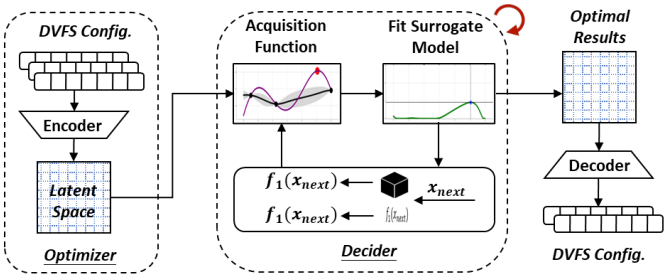


Figure 8: Design principles of the Optimizer and Decider.

$$b_i = \begin{cases} [l_j] & l_j > l_{trans} \\ [l_k, \cdots, l_{k+t}] & \forall_{i<t} l_{k+i} \leq l_{trans} \& l_{k+t+1} > l_{trans} \end{cases} \tag{3}$$

Through this novel representation, we can significantly reduce the search space, facilitating the discovery of optimal configurations in a more efficient and effective manner. We analyze the impact of this representation method on the same set of neural networks. Figure 7 (b) demonstrates that the proposed representation method can reduce the search dimensionality by up to 80%. Totally, By adopting the Assembler sub-module and its improved representation method, we can mitigate the challenges of the complex configuration space.

Optimizer: Despite building upon the pre-block-based configuration space representation, efficiently selecting the optimal frequency for each block remains challenging for existing optimization algorithms. For instance, the Bayesian optimization algorithm is known to perform well in low-dimensional spaces but struggles as the dimensionality increases [12]. To tackle the issue of high dimensionality, we have developed the Optimizer sub-module. This sub-module employs an Autoencoder to learn an embedding that effectively compresses the configuration space into a lower-dimensional representation. By leveraging this embedding, we are able to generate high-level features derived from the analytical model.

As depicted in Figure 8, we employ a 1D convolutional neural network (CNN) to effectively capture the positional properties of adjacent frequencies in the representation. The original frequency list is reconstructed using a decoder network. Additionally, to achieve a meaningful latent space, we incorporate high-level estimations of latency and energy as objectives for optimizing the latent space. Assuming the analytic model generates energy estimation e and latency estimation l, we introduce a multi-layer fully connected neural network to predict these values as $\hat{e}$ and $\hat{l}$, respectively. The overall loss function is defined as follows:

$$\mathcal{L} = \mathcal{L}_{recon} + \lambda ||\hat{e} - e||_2 + \gamma ||\hat{l} - l||_2 \tag{4}$$

Here, $\mathcal{L}_{recon}$ represents the reconstruction loss, and λ and γ are weights used to balance the prediction of energy and delay, respectively. With the Assember, we can effectively zoom out the configuration space into a smaller one, thus optimizing the process of finding the optimal configurations.

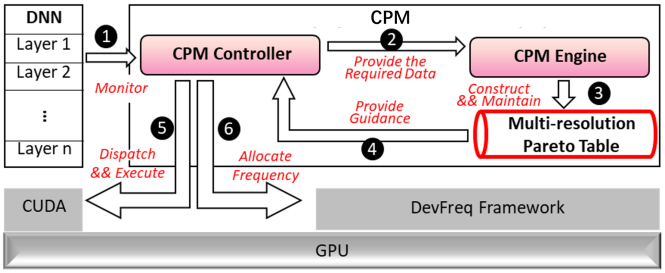


Figure 9: The workflow of CPM.

Decider: After constructing the analytic model and optimizing the search space with the above modules, we introduce the Decider sub-module to determine the optimal cross-layer and DVFS configuration for the DNN model. In the Decider, we leverage the Bayesian Optimization (BO) algorithm to find the optimal configuration automatically. The reason we choose BO is that it has proven to be an efficient approach for automating the search for optimal configurations and exhibits superior sample efficiency compared to alternative methods such as evolutionary algorithms.

In our task setting, we encounter two conflicting objectives: energy and latency. The Decider aims to generate a Pareto optimal set of solutions that outperform all other explored options. A solution x^* is considered Pareto optimal if it satisfies the following conditions:

$$f_k(x^*) \leq f_k(x) \forall k, x \text{ and } \exists j : f_j(x^*) < f_j(x) \forall x \neq x^*$$

where f_k is the k-th objective function.

Since the computational expense associated with directly evaluating these objective functions, we approximate them using a surrogate Gaussian Process (GP) model (`Fit Surrogate Model` in Figure 8). For each GP model, it estimates a probability distribution based on previously queried data $x_i \in X_{i=1}^n$. For each of the surrogate functions, it is assumed to be jointly Gaussian with mean m_k and co-variance K_k, *i.e.*, $f_k(x_n) \sim N(m_k, K_k)$.

Given these m_k and K_k, an acquisition function (`Acquisition Function` in Figure 8) α_{n+1} is constructed to determine the next query point x_{n+1} that can effectively trade-off the exploration and exploitation. The calculation of α_{n+1} is always much cheaper to evaluate than the actual functions, and hence x_{n+1} is determined as:

$$x_{n+1} = \arg\max_{x \in X} \ \alpha_{n+1} \tag{5}$$

For multi-objective Bayesian optimization, the Expected Hypervolume Improvement (EHVI) is widely used. Given a Pareto set $\mathcal{P}$ and reference point r, it can be defined as,

$$\alpha_{EHVI} = \mathbb{E}[HV(f(\mathcal{X}_{cand} \cup \mathcal{P}, r) - HV(f(\mathcal{P}, r))] \tag{6}$$

where $HV()$ is the hypervolume indicator [13].

In total, the Decider effectively finds optimal DVFS configurations to harmonize QoS with energy consumption. While alternative search algorithms can be considered, in Section V, we demonstrate that our algorithm outperforms most state-of-the-art methods.

Algorithm 1: The CPM Algorithm

Input: Config. Space X, Objective Functions O, Number of init iteration N_{init}, Number of optimize iteration N_{iter}, Multi resolutions List I_s, DNN model M
Output: Pareto Optimal Set: X^*

Construct the analytic model with Eq. (1 and 2)
Build the block-based config. space X_1 from X with Eq. (3)
Train the VAE model with Eq. (4)
$D = \phi, X^* = \phi$
for $i = 1$ ***to*** N_{init} **do**
  x_i = Random(X_1) r_i = Generate(M, I_s)
  Y_i = RUN(x_i, O, r_i, M) z_i =Encoder(x_i)
  $D = D \cup (z_i, Y_i)$
Z^* = Pareto_init(D, I_s) **for** $n = 1$ ***to*** N_{iter} **do**
  for k ***in*** $range(F)$ **do**
    f_k =GP$_k$(D)
  α_n =Build_acq($f_1, .., f_K$) Select z_n by maximizing EHVI from Eq. (6)
  x_n =Decode(z_n) r_n = Generate(M, I_s)
  Y_n =RUN(x_n, F, r_n, M) $D = D \cup (z_n, Y_n)$ Z^* = Pareto_update(z_n, Y_n, Z^*, $I_s(r_n)$)
return Decode(Z^*)

C. The Workflow of CPM Controller

The CPM Controller is in charge of executing the decisions made by the CPM Engine accurately on the target AIoT system. The overall workflow of the CPM is depicted in Figure 9. When executing a DNN model, the input data undergoes layer-by-layer processing. In this regard, the CPM Controller operates as follows: For each layer, the CPM Controller utilizes its Runtime Monitor to ❶ collect various execution characteristics, including the execution latency and energy consumption. Then, the CPM Controller ❷ sends the current state (including latency, layer number, method, and other model-specific data) to the CPM Engine. This state serves as input for the CPM Engine to make decisions regarding the next layer's frequency. After that, the CPM Engine ❸ obtains the next layer frequency by using the methodology described in Section III-B. To handle inputs of different resolutions, we introduce a resolution-aware Pareto frontier. This involves constructing a multi-resolution Pareto table and updating it based on the current resolution. Afterward, the CPM Engine ❹ relays the optimal configurations back to the CPM Controller. Then, the CPM Controller ❺ instructs the DNN to execute the next layer using its Workload Dispatcher sub-module. At the same time, it ❻ configures the GPU frequency for this layer using the Frequency Allocator sub-module that calls the system framework `Devfreq`. Of course, if the frequency remains unchanged from the previous layer, the CPM Controller intelligently avoids unnecessary DVFS adjustments. The aforementioned steps are repeated for all remaining layers until the execution of the entire DNN model is completed.

With the above, the CPM can find better solutions for DNN models that process data with different input sizes. The overall algorithm of CPM is shown in Algorithm 1.

- We first construct the mathematical model (i.e., the

Table I: The evaluated DNN models

Model	Layer number	Parameters	Input size	GFLOPs
AlexNet	21	58.69M	(3,227,227)	1.13
GoogleNet	26	6.63M	(3,224,224)	1.51
VGGNet	39	138.36M	(3,224,224)	15.47
Yolo	73	50.60M	(3,64,64) (3,224,224) (3,320,320) (3,448,448) (3,521,512)	0.35 4.27 8.72 17.09 22.32
ResNet	155	25.55M	(3,224,224)	4.13

Modeler in CPM Engine), build the block-based search space (i.e., the Assembler in CPM Engine) and train the VAE model with simulated data (i.e., the Optimizer in CPM Engine) as depicted in `Lines 1-3`.

- Then, we randomly select the configuration and get the latent representation and the metrics to initialize the optimization process (`Lines 4-7`).
- Finally, we continuously update the Pareto set to get a good configuration by using the BO algorithm (`Lines 8-12`). The searching process is mainly performed by the Decider and the model execution mainly depends on the CPM Controller.

Overall, the CPM Controller effectively manages the execution process, while the CPM Engine handles the decision-making and optimization aspects of the system. Their effective collaboration guarantees high QoS, particularly in terms of execution latency, and energy efficiency on AIoT systems.

IV. Experimental Methodologies

A. Hardware Platform

To validate CPM, we developed a prototype on NVIDIA's `Jetson Xavier NX`, featuring a 384-core Volta-based GPU with 15 DVFS settings. Consequently, the potential configuration space expands exponentially up to 15^n for a DNN with n layers. We adjusted DVFS by modifying the `set_freq` file and we use a single image for real-time AIoT testing. Meanwhile, we use the `INA3221 power monitor` at I2C address 0x40 [10] and python `time module` to obtain the system power and task execution latency respectively. To ensure the accuracy of our energy efficiency analysis, we measure the energy of whole system when running the application and then deducting the static energy consumed when the system is idle.

B. DNN models

To evaluate CPM's efficacy, we conducted experiments with four classification DNN tasks and one object detection DNN task, with layers ranging from tens to hundreds. This includes 21-layer `AlexNet` for image recognition, 26-layer `GoogLeNet`, `VGGNet` with 39 layers (VGG16 here), `YOLO` for object detection with 73 layers, and 155-layer `ResNet` (ResNet50 here), demonstrating CPM 's versatility across various input sizes and architectures. Details are in Table I, implemented in `Pytorch`.

Table II: The compared state-of-the-art baselines

Categories	Scheme	Description
Handcrafted Per-layer PM Policies	***Profile-L***	Select each layer's frequency that makes it have the minimum execution latency.
	Profile-E	Select each layer's frequency that makes it have the minimum energy consumption.
	NeuOS	Slect next layer's frequency according to the LAG [16] algorithm and Deadline.
Automated Per-layer PM Policies	***Bayesian***	Search each layer's freqeuncy using the BoTroch sampler and Bayesian optimization.
	NSGAII	Search each layer's frequency using the Non-dominated Sorting Genetic Algorithm II.
	Sampling	Search each layer's frequency using the random sampling algorithm.
	TPE	Search each layer's frequency using the Treestructured Parzen Estimator algorithm.
Cross-layer PM Policies	***CPM***	Our proposed method with cross-layer optimization considers the DVFS overheads.

C. Baseline methods

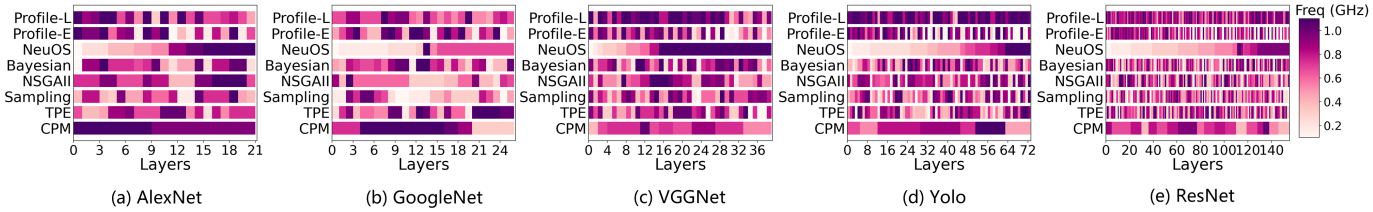
We benchmarked CPM against leading DNN power management (PM) techniques, including both handcrafted methods based on profiling (`Profile-L` for latency minimization, `Profile-E` for energy reduction) and SOTA `NeuOS` [1] using DVFS per-layer control with the LAG metric. Additionally, we explored automated PM strategies using statistical or machine learning to find optimal frequencies, such as Bayesian optimization (`Bayesian`), Nondominated Sorting Genetic Algorithm II (`NSGAII`), random sampling (`Sampling`), and Tree-structured Parzen Estimator (`TPE`). Unlike these methods, which don't fully address DVFS overheads, CPM offers a more efficient cross-layer PM approach, balancing DVFS's costs and benefits. Details are in Table II, and all automated methods were implemented via `Optuna` [14].

V. Evaluation Results

A. Execution Visualization

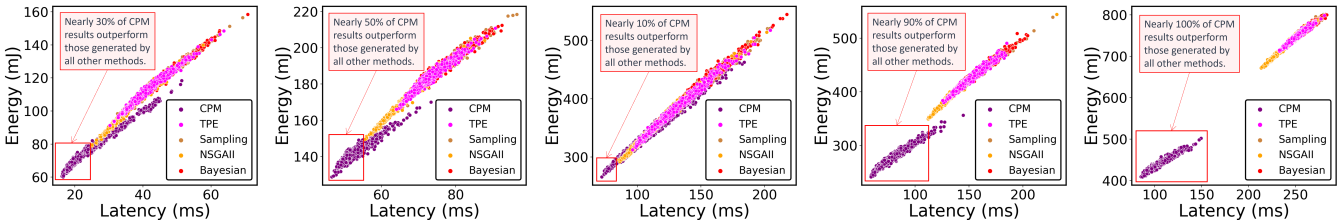
To conduct a comprehensive analysis and comparison of the configuration process of DVFS under different baseline methods, we first visualize the configurations chosen by each strategy for five different networks, as illustrated in Figure 10. These results provide valuable insights into the behavior exhibited by the different strategies across the five networks, highlighting the distinct characteristics of each approach.

As shown in the figure, the handcrafted strategies, namely `Profile-L` and `Profile-E`, demonstrate frequent frequency switching and generate optimal configurations for higher frequencies during execution. Conversely, `NeuOS` adopts a different approach, frequently selecting lower frequencies in most layers but opting for higher frequencies in the final few layers. This behavior is attributed to `NeuOS`'s focus on increasing processing frequency to meet tight deadlines while minimizing DVFS overhead by reducing frequency switching. However, this strategy results in longer execution latency and higher energy consumption, as illustrated in Figure 12. In contrast, the proposed `CPM` method exhibits a more balanced and efficient approach. It assembles layers into larger blocks with careful consideration of DVFS overhead and autonomously selects appropriate frequencies. By doing so, `CPM` significantly



(a) AlexNet (b) GoogleNet (c) VGGNet (d) Yolo (e) ResNet

Figure 10: Visualization of DVFS configurations of layers for different PM methods.



(a) AlexNet (b) GoogleNet (c) VGGNet (d) Yolo (e) ResNet

Figure 11: Comparison of the effectiveness of the search process between different automated PM methods and CPM.

Table III: Comparison of the adaptability to dynamic inputs between different PM methods and CPM using `Yolo`.

	64*64		224*224		320*320		448*448		512*512	
	E(mJ)	L(ms)	E(mJ)	L(ms)	E(mJ)	L(ms)	E(mJ)	L(ms)	E(mJ)	L(ms)
Profile-L	327.3	107.8	325.7	101.4	434.7	124.9	526.7	140.2	565.9	144.5
Profile-E	332.0	125.2	356.6	127.1	425.8	140.5	499.4	155.3	547.1	165.7
NeuOS	240.2	73.2	273.8	79.7	373.8	99.0	483.9	124.6	529.3	131.6
Bayesian	349.3	120.3	389.2	129.6	474.3	147.1	575.7	167.6	612.6	176.8
NSGAII	319.1	108.1	348.8	112.2	433.5	127.1	515.3	141.0	559.6	146.2
Sampling	358.2	127.1	386.5	131.2	474.2	146.7	572.6	166.7	603.5	169.7
TPE	349.7	119.6	379.9	126.6	462.2	141.9	563.8	165.2	602.1	165.0
CPM	***207.6***	***49.0***	***240.9***	***59.1***	***342.1***	***78.9***	***441.2***	***95.6***	***478.9***	***105.1***

reduces the frequency switching frequency. For instance, in the case of `AlexNet`, `CPM` only performs frequency switch once due to the network's low computational cost and small layer number. These observations collectively demonstrate the superior performance and efficiency of our proposed method.

B. Effectiveness Validation

In this section, we validate the effectiveness of CPM in adapting to the complex configuration space caused by fine-grained DVFS configurations and input dynamicity. We first assess the superiority of the search algorithm employed by CPM through a comparative analysis with four other automated methods mentioned in Section IV. In the interest of fairness, we limited all methods to 1000 trials when searching with `Optuna`. The total results of this comparison are visually represented in Figure 11. Evidently, `CPM` consistently excelled in searching for the optimal DVFS configuration across all cases. In stark contrast, the other four algorithms under examination struggled to discover sufficiently favorable configurations within the same computational budget. Notably, nearly 10%, 30%, 50% and 100% of the results obtained using `CPM` outperform those generated by all other methods under the `VGGNet`, `AlexNet`, `GoogleNet` and `ResNet` respectively. These results indicate that `CPM` yields superior outcomes as the complexity of the neural network, measured by the number of layers, increases. Among the baseline algorithms, `NSGAII` algorithm demonstrates the strongest performance, benefiting from its use of elitist mechanisms, diversity-preserving strategies, and emphasis on non-dominated solutions. However, it still fell short when compared to the effectiveness of our algorithm. In summary, our findings underscore the effectiveness of employing pre-blocking and autoencoders in enhancing the performance of Bayesian optimization algorithms when dealing with high-dimensional data.

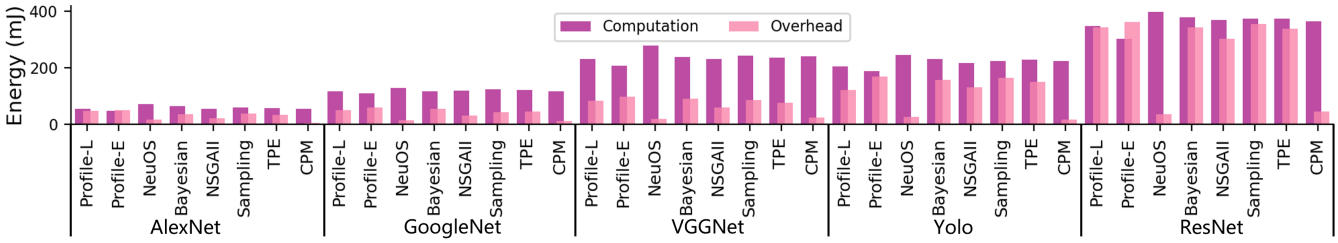
We also validate that CPM excels at handling input dynamicity by comparing it with different baseline methods under `YOLO` with different input resolutions. As illustrated in Table I, `YOLO` processes images of various resolutions, including 64*64, 224*224, 320*320, 448*448 and 512*512. We compare The results are shown in Table III. As we can see, `CPM` is also able to find the configuration with the lowest energy consumption and latency for different resolution scenarios. Among all methods, `NeuOS` also achieves the best performance besides `CPM`. By comparing with `NeuOS`, `CPM` can achieve great inference latency reduction from 20.07% to 32.99% and energy consumption reduction from 8.47% to 13.57% across five different resolution scenarios. These also show that our strategy can adapt to different resolution scenarios.

C. Efficiency Evaluation

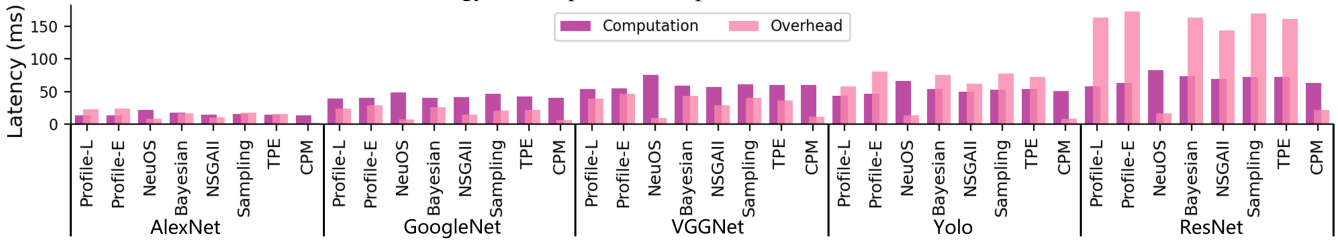
To evaluate the efficiency, we first compare CPM with the current SOTA power management methods mentioned in Section IV on five DNN models described in Table I. The experimental results are shown in Table IV, including measurements of energy consumption and latency achieved under optimal DVFS configurations using different methods. We can figure out that the frequency configurations found by `CPM` consistently achieve minimum energy consumption and latency across these five networks. The `Profile-L` should have achieved minimum latency for setting the frequencies that minimize the latency for each layer when ignoring DVFS latency overhead. However, when considering the DVFS latency overhead, this method fails to obtain optimal frequency configurations to minimize the total latency. Same as `Profile-L`, `Profile-E` strategy also fails to obtain optimal frequency configurations to minimize energy when

Table IV: Energy (mJ) and latency (ms) efficiency of the optimal DVFS configuration under different PM methods.

	AlexNet		GoogleNet		ResNet		VGGNet		Yolo	
	Energy	**Latency**	**Energy**	**Latency**	**Energy**	**Latency**	**Energy**	**Latency**	**Energy**	**Latency**
Profile-L	102.940	36.159	168.178	63.976	689.452	221.315	315.138	93.479	325.689	101.442
Profile-E	99.513	37.669	169.184	69.632	665.108	236.442	305.971	102.110	356.609	127.128
NeuOS	88.394	29.898	144.881	55.522	433.817	100.050	297.691	85.316	273.839	79.684
Bayesian	98.821	34.775	173.128	67.447	721.975	237.735	327.711	102.194	389.202	129.600
NSGAII	78.902	25.054	149.468	55.605	671.476	213.492	292.010	86.252	348.777	112.185
Sampling	98.158	33.690	168.062	67.052	729.485	242.743	329.281	102.589	386.509	131.242
TPE	91.258	30.308	166.061	64.045	711.180	233.379	312.392	96.300	379.856	126.576
CPM	***60.480***	***16.215***	***128.752***	***46.622***	***409.068***	***85.380***	***265.839***	**72.091**	***240.944***	***59.130***



(a) Energy consumption of computation and DVFS control.



(b) Execution latency of computation and DVFS control.

Figure 12: Analysis of DVFS control overheads across various DNNs under different PM methods.

taking into account the additional overhead of DVFS. `NeuOS` also performs per-layer DVFS, but its oversimplification, which does not take into account the overhead caused by DVFS, prevents it from obtaining better performance in terms of both energy and latency. Four automated methods, including `Bayesian`, `NSGAII`, `Random`, `TPE`, also perform badly because they all have a big search space without merging layers and can not find good frequency configurations in limited time even if they consider the overhead of DVFS.

Instead, `CPM` is able to find the optimal frequency configuration by merging layers into blocks for a smaller search space, modeling the DVFS delay, and searching with an efficient strategy. Since it can find suitable configurations for two objectives, we choose the solution that better trades off the two, from which we can see that our method has the lowest energy consumption and the delay is also quite low compared to the previous best solution, therefore achieving good QoS. Overall, as shown in Table IV, `NeuOS` achieves the best performance besides `CPM`. By comparing with `NeuOS`, `CPM` effectively reduces the inference latency from 14.66% to 45.76% and energy consumption from 5.71% to 31.58% across five networks. These show that our strategy is effective with consideration of DVFS overhead.

D. Overhead Analysis

In this section, we conduct an analysis of the DVFS control overhead associated with different methods. We examine how different strategies balance the trade-off between choosing the appropriate computational frequency to maximize computational efficiency and reducing DVFS overhead. To provide a comprehensive perspective, we break down the total energy consumption into computation energy and DVFS overhead energy and the total execution latency into computation latency and DVFS overhead latency. Figure 12 shows the results.

It can be seen that different strategies have different proportions of DNN computation and DVFS overhead. The handcrafted method, for instance, brings higher computational efficiency in terms of both energy consumption and execution latency. However, the severe DVFS overhead introduced by these methods diminishes the gains achieved through computation across the five networks. Particularly for `AlexNet` and `ResNet`, the handcrafted method incurs significantly higher DVFS overhead. In terms of energy consumption, the DVFS overhead nearly equals the computational cost for `AlexNet` and `ResNet`. Regarding execution latency, the DVFS overhead exceeds the computational cost by almost double. This phenomenon can be primarily attributed to the relatively low computational cost of `AlexNet` and the large number of layers in `ResNet`. On the other hand, `NeuOS` exhibits low DVFS overhead similar to `CPM`, but it incurs high computational costs. In contrast, `CPM` effectively balances the benefits and costs of DVFS, resulting in the lowest energy consumption and execution latency.

VI. Related Work

Energy-Efficient AIoT Systems: To build energy-efficient AIoT systems, researchers have explored massive power optimization techniques [15]–[18], including application-level approximate computing approaches like quantization [16], pruning, and multi-exit inference [17], as well as system-level power throttling strategies [19]. While fine-grained power management techniques have gained significant research attention in recent years [1], [4], [5], [9], their effectiveness is limited by the oversight of the DVFS control overhead, which can introduce substantial control latency in the millisecond or even second range, directly impacting QoS of AIoT systems. *Our work emphasizes adopting comprehensive solutions to address both fine-grained power optimization and efficient DVFS control, incorporating cross-layer optimizations to achieve good QoS and optimal energy efficiency in AIoT systems.*

Dynamic Voltage Frequency Scaling: DVFS enables substantial power and energy savings in different computing systems [7], [20]. To maximize power and energy optimization potential, there is a growing trend towards fine-grained DVFS management [7], [20]–[22] at the per-user, per-application, or even per-task level. For instance, Rubik [20] is an application-specific optimization to mitigate the inactive-to-active delay caused by power-saving mode transitions. In the context of DNN execution, DVFS is also leveraged for fine-grained tuning in computing systems, as observed in methods like ApNet [5], Predjoule [4], and NeuOS [1]. However, most of these works overlook that DVFS techniques inherently introduce control latency ranging from milliseconds to seconds. *Therefore, it is crucial to acknowledge the presence of DVFS overheads and incorporate them into the design of power management policies to achieve optimal energy efficiency.*

Automated DNN Management Methods: Automated DNN design and management methods which can be roughly categorized into three types based on their targets including neural architecture search [23], [24], automated DNN mapping [25], [26], and hardware design automation [27], [28], have been extensively researched. They employ intelligent methods to identify optimal solutions from a vast pool of candidates. These approaches have been proven effective in alleviating the cumbersome effort of designing and manufacturing new algorithms, systems, chips, etc. *Building upon these advancements, we propose more comprehensive power management techniques for DNNs that leverage these automated methods within the context of a large and complex PM configuration space.*

VII. Conclusion

Power management in AIoT systems is a pressing and challenging task. In this paper, our research introduces a cross-layer power management strategy for AIoT systems, aiming for energy efficiency while maintaining quality of service (QoS). This method notably enhances energy savings and reduces inference latency in these demanding embedded systems. Moreover, we demonstrate that optimization techniques based on manual approaches often degrade QoS due to latency from control overheads. By addressing these challenges, our findings pave the way for improved QoS and energy efficiency in future edge computing scenarios, including micro datacenters.

VIII. Acknowledgements

We sincerely thank all the anonymous reviewers for their valuable comments and feedback. This work is supported in part by the National Natural Science Foundation of China (No. 62122053), and the Shanghai S&T Committee Rising-Star Program (No.21QA1404400). Xiaofeng Hou is also sponsored by Shanghai Pujiang Program (No.23PJ1405100). Chao Li is the corresponding author.

References

[1] S. Bateni and et al., "Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems," in *Usenix ATC*, 2020.
[2] B. Yu and et al, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," *MICRO*, 2020.
[3] X. Hou and et al., "Characterizing and understanding end-to-end multi-modal neural networks on gpus," *IEEE CAL*, 2022.
[4] S. Bateni and et al., "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *RTSS*, 2018.
[5] S. Bateni and et al, "Apnet: Approximation-aware real-time neural network," in *RTSS*, 2018.
[6] A. Sriraman and et al., "μtune: Auto-tuned threading for oldi microservices," in *OSDI*, 2018.
[7] X. Hou and et al., "Ant-man: Towards agile power management in the microservice era," *SC*, 2020.
[8] C. Chou and *et al.*, "μdpm: Dynamic power management for the microsecond era," in *HPCA*, 2019.
[9] W. Kang and et al., "Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks," *RTSS*, 2021.
[10] Nvidia, "Nvidia jetson linux developer guide," 2021.
[11] W. Niu *et al.*, "Dnnfusion: accelerating deep neural networks execution with advanced operator fusion," *PLDI*, 2021.
[12] M. Binois *et al.*, "A survey on high-dimensional gaussian process modeling with application to bayesian optimization," *TELO*, 2022.
[13] S. Daulton *et al.*, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *NeurIPS*, 2020.
[14] T. Akiba and et al., "Optuna: A next-generation hyperparameter optimization framework," *CoRR*, vol. abs/1907.10902, 2019.
[15] L. Zhang and et al., "Characterizing and orchestrating nfv-ready servers for efficient edge data processing," in *IWQoS*, 2019.
[16] K. Wang and *et al.*, "Haq: Hardware-aware automated quantization with mixed precision," in *CVPR*, 2019.
[17] X. Hou and et al., "Mmexit: Enabling fast and efficient multi-modal dnn inference with adaptive network exits," in *Euro-Par*, 2023.
[18] L. Zhang and et al., "First: Exploiting the multi-dimensional attributes of functions for power-aware serverless computing," in *IPDPS*, 2023.
[19] W. Godycki and *et al.*, "Enabling realistic fine-grain voltage scaling with reconfigurable power distribution networks," in *MICRO*, 2014.
[20] D. Lo and *et al.*, "Towards energy proportionality for large-scale latency-critical workloads," in *ISCA*, 2014.
[21] X. Hou and *et al.*, "Power grab in aggressively provisioned data centers: What is the risk and what can be done about it," in *ICCD*, 2018.
[22] X. Hou and et al., "When power oversubscription meets traffic flood attack: Re-thinking data center peak load management," *ICPP*, 2019.
[23] Z. Wang and *et al.*, "Hierarchical memory-constrained operator scheduling of neural architecture search networks," in *DAC*, 2022.
[24] X. Luo and *et al.*, "You only search once: on lightweight differentiable architecture search for resource-constrained embedded platforms," in *DAC*, 2022.
[25] Y. N. Wu and *et al.*, "An architecture-level energy and area estimator for processing-in-memory accelerator designs," in *ISPASS*, 2020.
[26] D. Hong and *et al.*, "Enabling hard constraints in differentiable neural network and accelerator co-exploration," in *DAC*, 2022.
[27] A. Sohrabizadeh and *et al.*, "Automated accelerator optimization aided by graph neural networks," in *DAC*, 2022.
[28] P. Xu and *et al.*, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *FPGA*, 2020.