IPA: Inference Pipeline Adaptation to achieve high accuracy, cost-efficiency, and power-efficiency

Misagh Soltani, Sabah Anis, Xeerak Muhammad

Abstract:
Modern ML inference pipelines face the dual challenge of achieving high performance and minimizing energy consumption. Current systems, including IPA, lack the ability to monitor or optimize energy usage, making them unsuitable for real-world applications that demand both efficiency and sustainability. This project addresses this limitation by integrating energy profiling into IPA, enabling it to adapt model selection and resource allocation based on real-time energy usage.

Introduction:
The rapid adoption of machine learning (ML) across diverse domains such as autonomous vehicles, healthcare diagnostics, and recommendation systems has brought significant improvements to these fields. These applications often rely on complex ML inference pipelines that demand high accuracy, low latency, and scalability. However, the cost of deploying these models at scale, both in terms of operational expenditure and environmental impact, has become a critical concern.

Current inference systems primarily optimize for accuracy, latency, and cost, but energy consumption, which is a crucial factor for sustainability, is often overlooked. This gap in optimization hinders efforts toward achieving sustainable AI, especially as the carbon footprint of ML systems becomes increasingly evident. The development of energy-aware inference systems is thus essential for aligning the growth of AI with environmental goals.

While existing frameworks like InferLine [3], Loki [4], AutoInfer [5], and Swayam [6] have made strides in optimizing ML inference pipelines for latency and cost, they lack explicit integration of energy efficiency into their optimization objectives. Other studies, such as those by Harty et al. [7], have explored energy-efficient practices in ML training and inference. Systems like Clover [2] have introduced carbon-conscious runtimes but remain limited in their integration of accuracy, latency, cost, and energy trade-offs into a unified framework.

The Inference Pipeline Adapter (IPA) [1] system provides a foundation for optimizing ML inference pipelines. However, it does not yet incorporate energy consumption as a metric for decision-making. Addressing this limitation, our project, titled IPA: Inference Pipeline Adaptation to achieve high accuracy, cost-efficiency, and power-efficiency, extends the IPA to include energy considerations. By integrating energy metrics into IPA, we aim to operationalize energy-efficient AI practices while maintaining high performance and cost-effectiveness.

Therefore, this project makes the following contributions to the IPA system.

Energy-aware optimization: Extends IPA to use the perf tool to measure energy consumption and incorporate these measurements into IPA's optimization processes.

Benchmarking framework: Designed and executed experiments to benchmark IPA's energy performance under varying workloads, model sizes, and configurations.
Performance Analysis: Analyzed the results of energy and performance benchmarks and provided meaningful insights.

By addressing the critical challenge of energy efficiency, this project contributes to the broader goal of sustainable AI, bridging the gap between performance optimization and environmental responsibility.

Related Work:
[Will compile a discussion related to energy efficient inference pipelines]

Data:
We logged energy consumption in Joules (J) for the cpu package per second for four workloads. The workload includes a twitter trace dataset with text and images with varying throughput. Each line of the data log shows the energy consumption per second of the cpu package using the perf utility in linux as shown in Figure 1. The per second data was very noisy when first graphed so we visualized our data by using a per minute average as well as smoothing the graphs to analyze trends in our energy consumption with more granularity.

```
Starting CPU energy monitoring for video-mul-1...
     1.000096633              71.65 Joules power/energy-pkg/
     2.000322608              72.91 Joules power/energy-pkg/
     3.000516939              73.71 Joules power/energy-pkg/
     4.000755138              71.90 Joules power/energy-pkg/
     5.000993450              71.10 Joules power/energy-pkg/
     6.001219742              71.32 Joules power/energy-pkg/
     7.001451818              70.88 Joules power/energy-pkg/
     8.001678273              72.04 Joules power/energy-pkg/
     9.001848192              74.87 Joules power/energy-pkg/
    10.001999723             134.09 Joules power/energy-pkg/
    11.002119003              86.83 Joules power/energy-pkg/
    12.002278526              95.87 Joules power/energy-pkg/
    13.002396434              83.72 Joules power/energy-pkg/
    14.002558695              91.32 Joules power/energy-pkg/
    15.002682422              88.60 Joules power/energy-pkg/
    16.002849467              81.99 Joules power/energy-pkg/
    17.003013235              86.58 Joules power/energy-pkg/
    18.003181390              82.35 Joules power/energy-pkg/
    19.003315290              83.20 Joules power/energy-pkg/
    20.003504849              78.84 Joules power/energy-pkg/
    21.003688958              73.41 Joules power/energy-pkg/
    22.003932934              80.47 Joules power/energy-pkg/
    23.004118598              78.65 Joules power/energy-pkg/
    24.004303940              89.84 Joules power/energy-pkg/
    25.004484499              85.32 Joules power/energy-pkg/
    26.004725341              74.23 Joules power/energy-pkg/
```

Figure 1. Sample Energy Consumption Output

Methods:

Experiment Replication

Our first step included a replication of the IPA pipeline where we ran the pipeline on the twitter trace workload which included images and text. The inference pipeline included IPA, FA2-low, FA2-high, and RIM-high. The throughput of the workloads varied from bursty, steady low, steady high, and fluctuating. We ran the four pipelines across the twitter trace workload for four different workloads. This means that there were a total of 4 different workloads across 4 different inference pipelines for a total of 20 workloads.

Overview of Energy Profiling in ML Systems

Energy profiling in machine learning inference systems is crucial for understanding and optimizing the energy consumption of computational pipelines. Several tools are available for this purpose, ranging from hardware-integrated solutions like Intel's Running Average Power Limit (RAPL) to external power meters. The selection of an appropriate tool requires careful consideration of its compatibility, measurement granularity, integration ease, and operational

overhead. In this project, perf was selected as the primary tool for energy measurement due to its alignment with these requirements and its ability to integrate seamlessly into the existing Inference Pipeline Adapter (IPA) framework.

Criteria for Tool Selection

The choice of an energy measurement tool was guided by several critical factors. First, compatibility with the IPA system and the broader experimental environment was essential. The tool needed to support existing hardware and software configurations while providing sufficient flexibility to accommodate diverse workloads. Second, the tool's ability to deliver fine-grained energy consumption data was critical for capturing the nuances of energy usage across different model configurations, batch sizes, and workloads. Third, ease of use was prioritized to ensure that the tool could be efficiently integrated into automated workflows for benchmarking. Finally, the profiling tool needed to impose minimal computational overhead to avoid distorting the actual energy usage patterns of the inference pipelines.

Rationale for Choosing perf

Perf was chosen as the energy profiling tool for this project due to its strong alignment with the aforementioned criteria. As a widely adopted performance monitoring tool, perf integrates directly with Intel’s RAPL interface, enabling detailed energy measurements for CPUs. This compatibility makes it particularly well-suited for analyzing energy consumption in CPU-intensive inference tasks. Furthermore, perf provides granular energy consumption metrics, including per-core and package-level data, which are essential for fine-tuned energy optimization in ML inference systems.

The integration of perf into the IPA framework was straightforward due to its support for command-line operation and scripting. This feature facilitated its use in automated experiments, where energy consumption needed to be measured across a wide range of configurations and workloads. Additionally, perf introduces minimal overhead during profiling, ensuring that the energy measurements accurately reflect real-world performance. The tool’s extensibility, allowing for the measurement of additional performance metrics such as CPU cycles and cache usage, further supports its role as a comprehensive performance analysis solution.

Another key factor supporting the choice of perf was its robust community support and extensive documentation. As a standard tool in Linux environments, perf benefits from regular updates and well-maintained resources, which facilitated its adoption and use in this project. This reliability ensured that potential issues could be resolved quickly, allowing the focus to remain on the primary research objectives.

Addressing Limitations

While perf provides significant advantages, it has certain limitations. One notable constraint is its reliance on hardware support for RAPL, which restricts its usage to compatible processors. To address this, the experiments were conducted on hardware platforms in the Chameleon server with verified RAPL compatibility. Moreover, perf primarily measures CPU energy

consumption and does not directly account for GPU energy usage. Acknowledging this limitation, we only performed the experiments on CPUs.

Segment Anything Model Workload:
Segmentation is a fundamental pre-processing step in many video tracking and classification tasks. We aimed to benchmark one of the foundational one-shot segmentation models in the computer vision space known as Segment Anything Model (SAM) [8]. It generates high quality segmentation masks based on point and box prompts. It contains three different variants which include ViT-H, ViT-L, and ViT-B. ViT-B contains 91M parameters, ViT-L contains 308M parameters and ViT-L contains 636M parameters.

Integration of Q-learning Optimizer into the IPA Framework:
This part of the project aims to test Q-Learning as an approach to tackle optimization scenarios involving dynamic workloads, nonlinear trade-offs, and environments where traditional approaches like Gurobi might struggle. The key motivations included:
- Dynamic Workloads: Optimizing systems under changing input rates and system conditions.
- NonLinear Trade-offs: Addressing non-convex and non-linear relationships among accuracy, resource usage, latency, and throughput.
- RL experimentation: study the adaptability and decision-making by reinforcement learning and determine whether it's a viable alternative or complementary method to classical deterministic approaches.

Formulation of Q-Learning for Integrating into IPA
Q-Learning is formulated to solve the pipeline optimization problem by mapping the configuration options of the pipeline into a state-action-reward framework. The implementation is meant to interact directly with the pipeline's key parameters while adhering to its constraints.

Here is how the Q-Learning algorithm is adapted for this use case:
State Space
The state space was designed to the model through the current configuration of the pipeline. Each state is a combination of:

- **Model Variants:** The active variant of each stage in the pipeline.
- **Replica Counts:** The number of replicas allocated to each stage.
- **Batch Sizes:** The batch size used by each stage.

These dimensions define a multi-stage state space, where each state is a complete pipeline configuration.

Action Space
The action space consists of all possible adjustments that can be made to the pipeline configuration, including the following:

- Switching model variants for any stage.
- Increasing or decreasing the number of replicas for any stage within a prespecified range.

- Adjusting the batch size for any stage, with constraints from hardware and performance limitations.

Actions are designed such that the optimizer can easily explore neighbors of good configurations to balance exploration and exploitation.

Reward Function

The reward function is designed to reflect the objectives of optimization of the pipeline:

1. Maximizing Accuracy: The optimizer receives a reward for the settings improving the overall accuracy of the pipeline.

2. Minimizing Resource Usage: Higher utilization of CPU or GPU results in negative rewards.

3. Meeting SLAs: The settings that violate latency requirements or fail to sustain the arrival rate are heavily penalized.

4. Balancing Throughput and Latency: Rewards account for configurations that maintain throughput while staying within latency bounds.

This reward function directly incorporates the *objective* and *constraints* methods from the pre-defined *Optimizer* class.

Q-Table and Learning Process

The Q-Table keeps track of the expected return for every state-action combination. The learning process involves a step-by-step exploration of the state-action space where the optimizer executes different configurations, gets rewards with values as per the reward function, and updates the *Q-Table* accordingly. The optimizer learns to give more importance to actions yielding higher rewards over time, converging on optimal or near-optimal configurations.

Exploration-Exploitation Trade-off

The implementation uses an ε-greedy policy as described here. With probability ε, the optimizer explores new actions to find better configurations. On the other hand, it exploits the best-known actions with probability 1 - ε, based on the current *Q-Table*. However, the exploration rate ε decays with time to balance initial exploration w.r.t. final exploitation.

Migration and Integration Process

The Q-Learning optimizer was implemented in the current *Optimizer* class without changing the structure of the class, and without breaking any functionality present:

1. Conformity with the Current Framework: The newly added method is designed with the same design patterns followed by the class, reusing methods like *latency_parameters*, *throughput_parameters*, and *accuracy_parameters* for compatibility.

2. Preservation of Signatures: This approach has the same input parameters and return types as other optimizers so it can be called interchangeably.

3. Constraints Handling: SLA and throughput constraints are baked right into the reward function to ensure the optimizer narrows down to feasible configurations only.

Pipeline Interaction

The Q-Learning optimizer interacts with the pipeline in the following ways. It dynamically selects configuration through exploration of states and actions, and then applies the configurations on

the pipeline through methods like *model_switch*, *re_scale*, and *change_batch* and evaluates the pipeline performance using throughput, latency, accuracy, and resource utilization.

Experiments using Q-Learning:
The Q-learning component of the project is still an ongoing effort, and due to the technical issues and pivots that took place while formulating the problem, as a well-fitting problem for Q-learning, as well as the high configuration space of the whole project and the complexity of the different technologies used in the original project, we do not have appealing results for the current version of the document.
It is noteworthy that, as of now, we have done different experiments with different versions of the implementation for integration of Q-learning. Although all of these ended up in failed experiments, the latest formulation and version of the code was the most promising one based on the group decision, and we are putting all of our efforts into getting the results. We hope to see better results by the final version of this submission (and the video presentation).
However, the version of the code we have prepared so far is available in the project repository.
We believe that these failures have had an important role in our learning process, throughout the course and working on the project.

Experiments:
Replication Graphs:

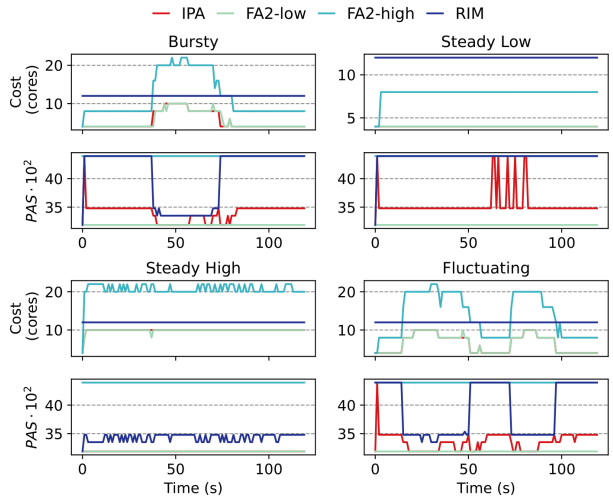

Figure 2. Original Paper Results

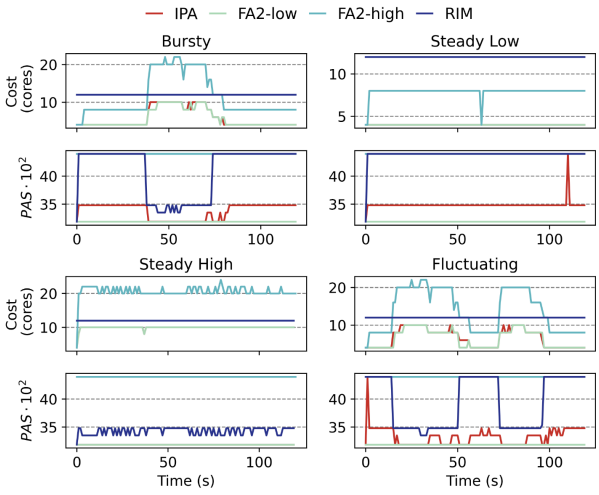

Figure 3. Replication Results

We see similar results to the inference adapter paper but with a few variations in pipeline accuracy score (PAS) and cost in the steady high graphs as shown in Figures 2 and 3. We encountered some errors with assigning minion nodes thus it may lead to some differences such as higher cost and PAS in some workloads such as steady high. We also see a variation in the pipeline accuracy score when looking at the bursty workload which could be due to all the load being provisioned on a single master node for kubernetes instead of 2 other minion nodes.

Energy Consumption of CPU per workload:

The following figures show the cpu energy consumption per minute across four workloads (bursty, steady low, steady high, and fluctuating) across four inference pipeline models (IPA, FA2-low, FA2-high, RIM-high). We noticed that IPA had the lowest energy consumption (J) throughout all four workloads as seen in Figures 4-6. RIM-high had the highest overall energy consumption with the exception of the steady low workload in Figure 5.

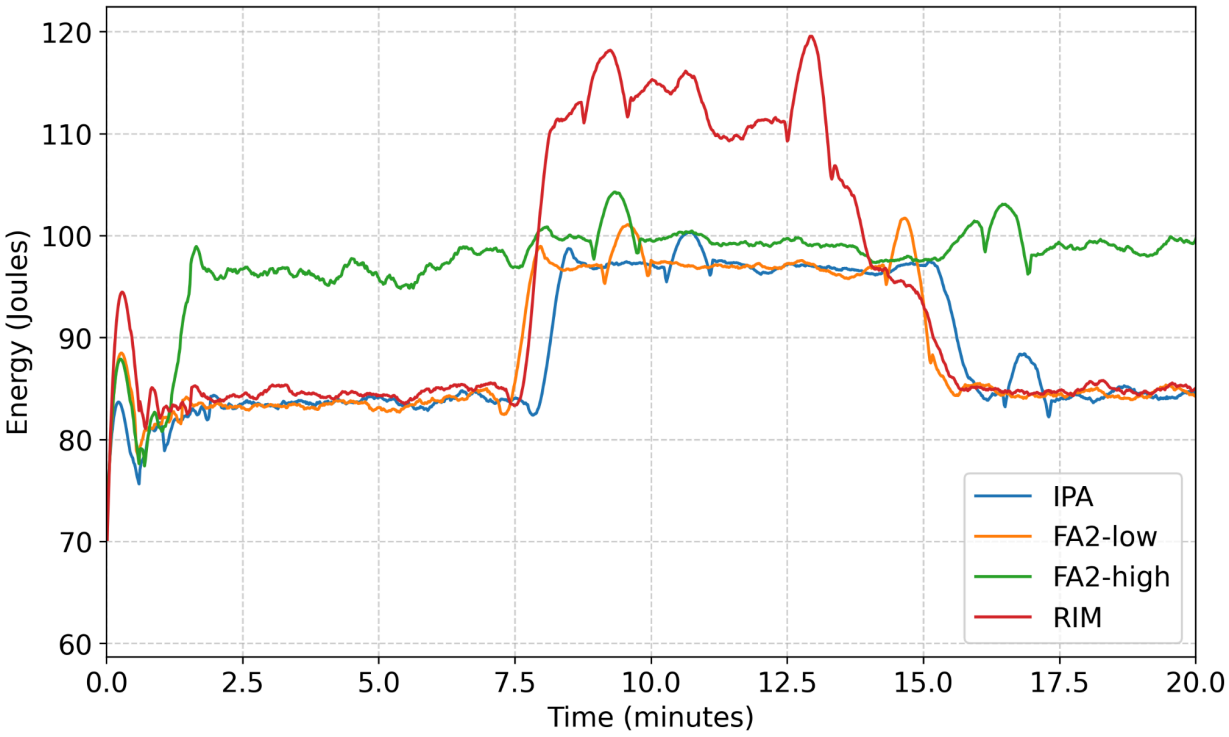

Figure 4: Energy measurement of bursty workload across four inference pipelines

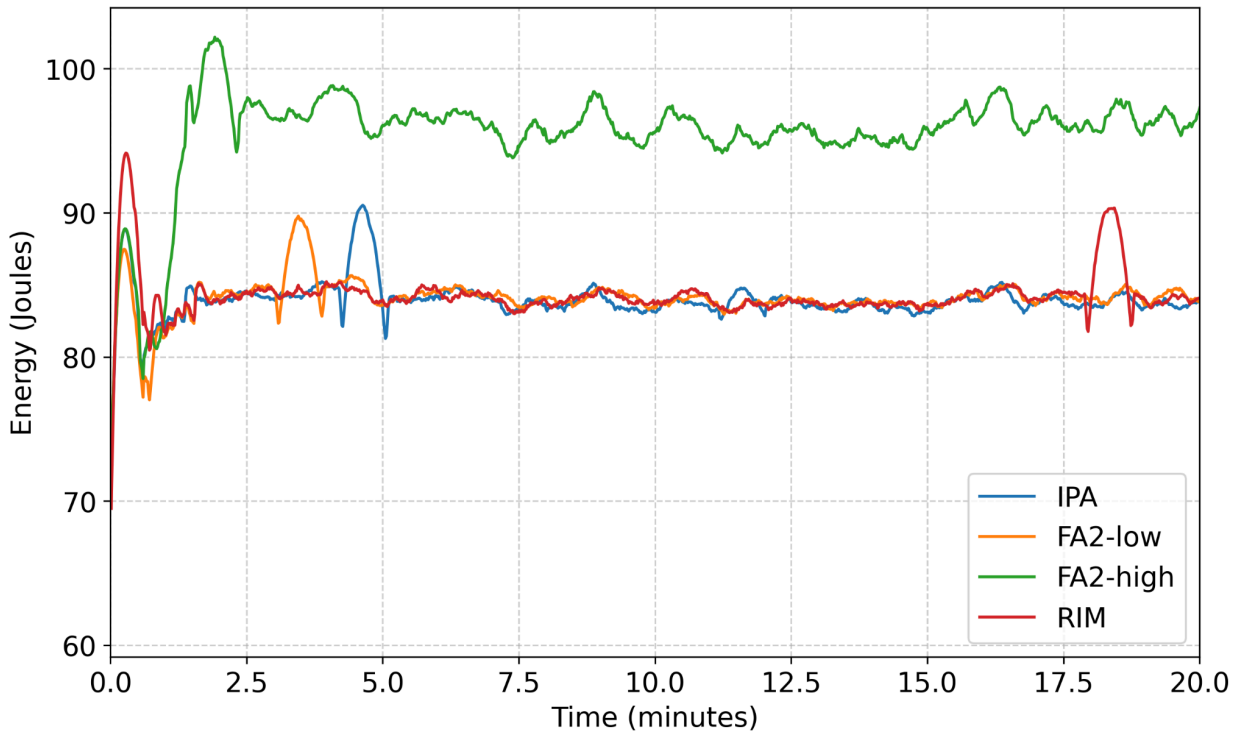

Figure 5: Energy measurement of steady low workload across four inference pipelines

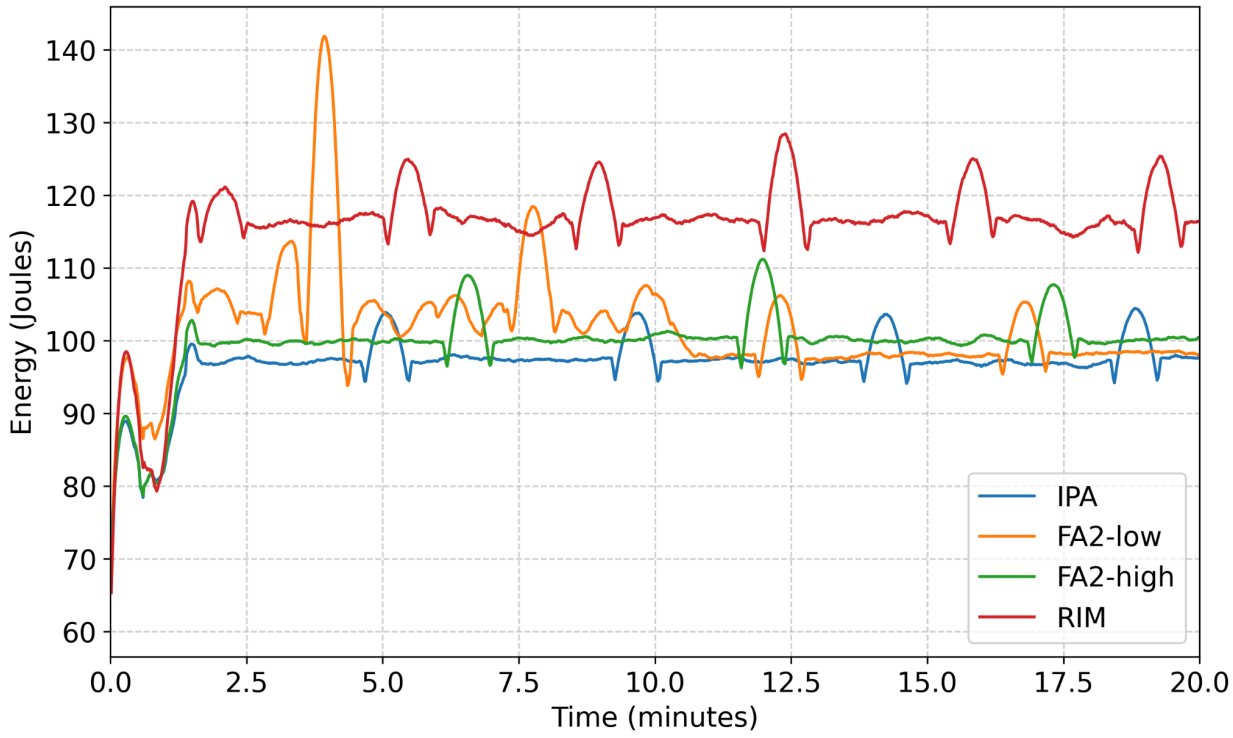

Figure 6: Energy measurement of steady high workload across four inference pipelines

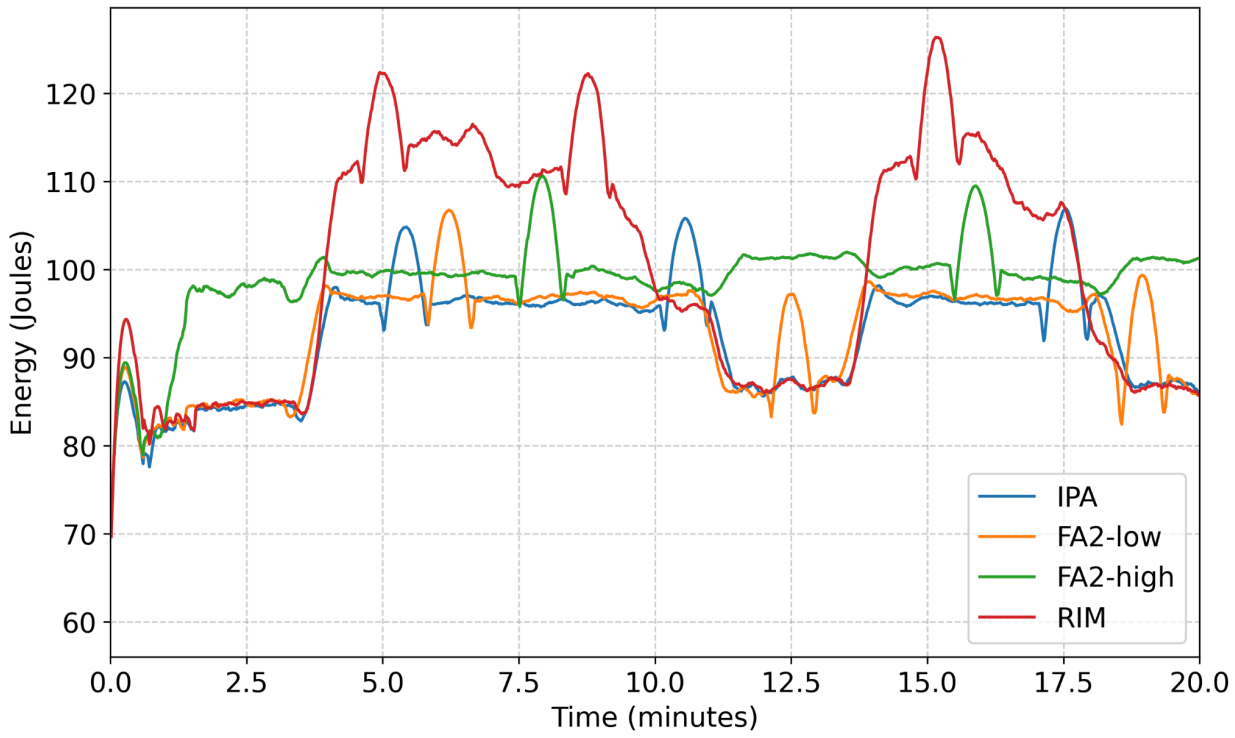

Figure 7: Energy measurement of fluctuating workload across four inference pipelines

Furthermore, we were able to benchmark the cpu energy consumption of the 3 different variants of the segment anything model as shown in Figure 8. We see that the models with a higher parameter count utilized more energy in most cases as well as more time. We can also see some cases where ViT-L requires more energy than ViT-H (~45 min) which shows that further optimization on energy and performance tradeoffs could be beneficial for the segmentation task. We have not been able to integrate the SAM model into IPA but we determined cpu energy consumption on a benchmark that contained 58 720x480 images in urban settings to understand some preliminary trends. This will be expanded to IPA and our energy optimization in the next step.

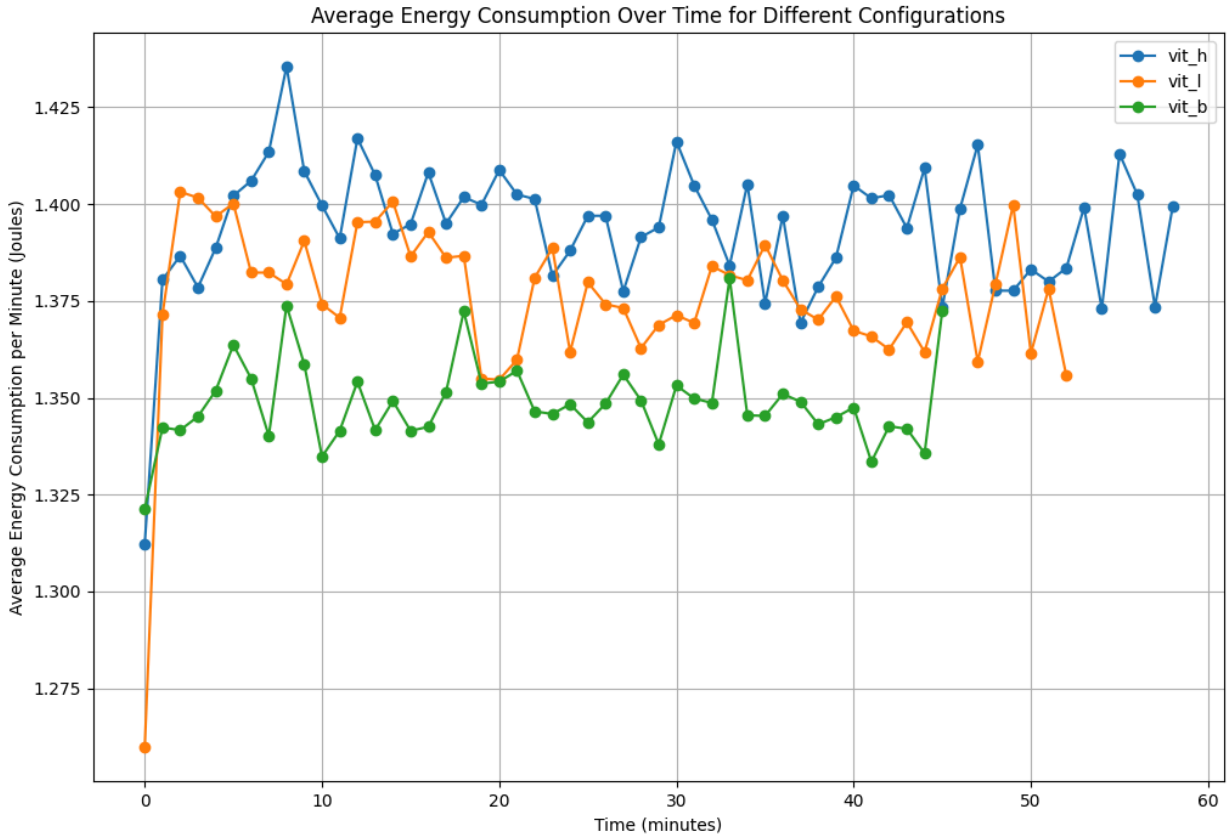

Figure 8. Energy measurement of the three different segment anything model variants

Conclusion:

In summary, we were able to monitor the energy consumption of four workloads across four inference models and understand some trends in energy consumption. IPA has the lowest power consumption when compared to other models. The next steps are to create a SAM workload in IPA and integrate power consumption into the optimization function to analyze the tradeoff between energy consumption, cost, and pipeline accuracy score.

Supplementary Materials:

References:

[1] Ghafouri, S., Razavi, K., Salmani, M., Sanaee, A., Lorido-Botran, T., Wang, L., ... & Jamshidi, P. (2023). IPA: Inference Pipeline Adaptation to Achieve High Accuracy and Cost-Efficiency. arXiv preprint arXiv:2308.12871.

[2] Li, B., Samsi, S., Gadepally, V., & Tiwari, D. (2023, November). Clover: Toward sustainable ai with carbon-aware machine learning inference service. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-15).

[3] Crankshaw, D., Wang, G., Zhou, X., Franklin, M. J., Gonzalez, J. E., & Stoica, I. (2020). InferLine: Latency-aware provisioning and scaling for prediction serving pipelines. Proceedings of the ACM Symposium on Cloud Computing (SoCC), 477–491.

[4] Du, Z., Guo, W., Wang, X., Gao, B., Wang, Z., & Zhang, Y. (2023). Loki: A system for serving ML inference pipelines with hardware and accuracy scaling. Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), 427–439.

[5] He, C., Sun, L., Li, J., & Li, K. (2021). AutoInfer: Self-driving management for resource-efficient, SLO-aware machine-learning inference in GPU clusters. IEEE Transactions on Parallel and Distributed Systems, 32(1), 92–105.

[6] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., & Ganger, G. R. (2017). Swayam: Distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. Proceedings of the 2017 ACM Symposium on Cloud Computing (SoCC), 185–197.

[7] Harty, A., Shah, N., Fleming, T., Asuni, N., & Jiang, N. (2023). Computing within limits: An empirical study of energy consumption in ML training and inference. arXiv preprint arXiv:2303.12101.

[8] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... & Girshick, R. (2023). Segment anything. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4015-4026).