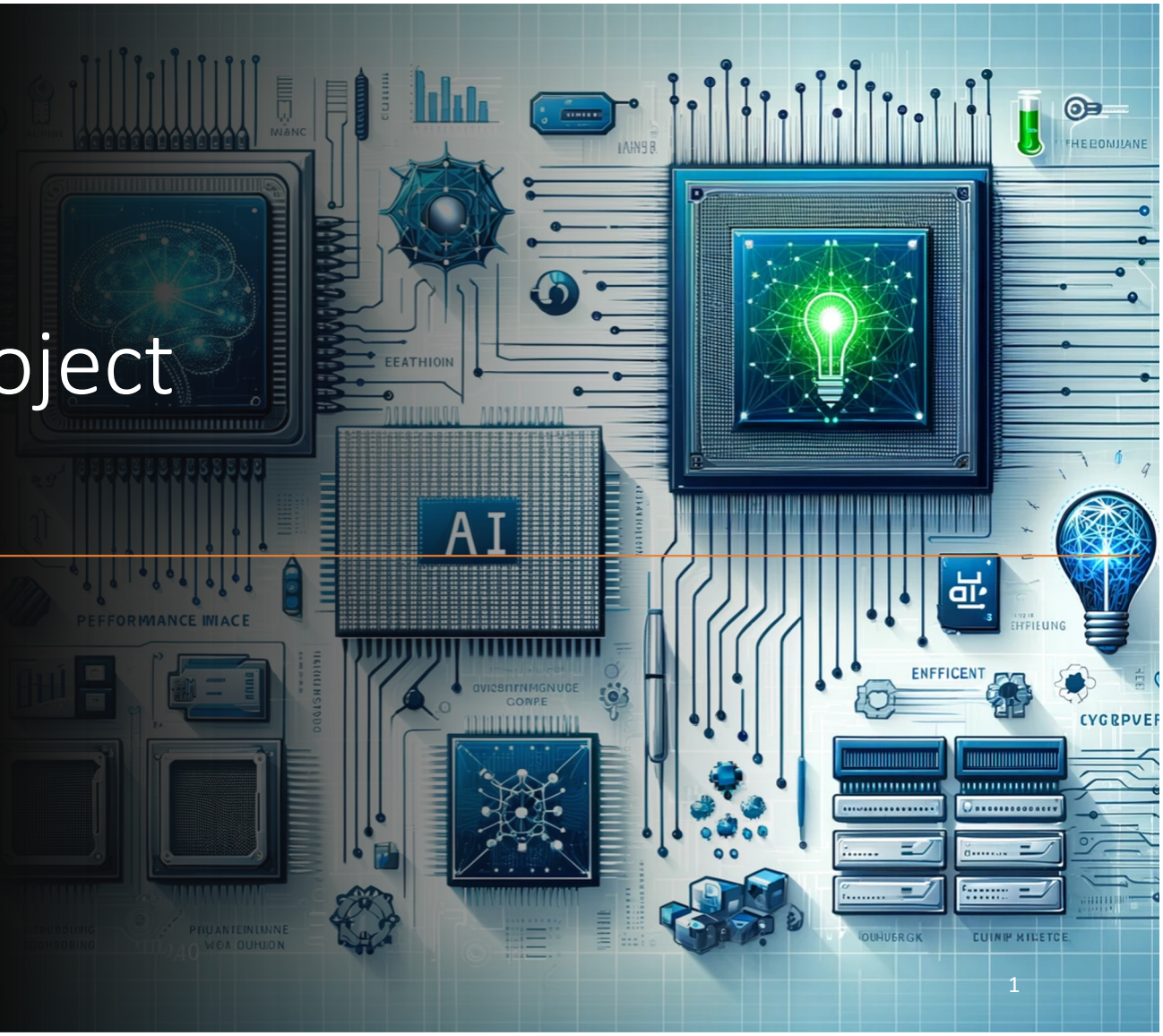
HPML Final Project Assignment

Dr. Kaoutar El Maghraoui

Fall 2025



Project Goals

01

Gain practical knowledge and apply high performance machine learning techniques using real AI benchmarks and datasets

02

Get familiar with profiling tools and performance optimization techniques that help speed up and improve the performance of

03

Understand the spectrum of AI runtime environments and multitude of tools and infrastructure to train and deploy AI

04

Perform a systematic performance analysis study

Teaming

- We encourage you to form teams of 4 for your project

- Guidelines for Forming Teams:
  - Define and document roles, responsibilities, communication methods, and conflict resolution strategies to create a shared understanding of what's expected from each team member. Ensure that these are clearly articulated in your project proposal.
  - When seeking team partners, consider individuals with skills that complement your own and those who share a genuine interest in the chosen AI model and project.
  - The formation of a team should stem from a mutual agreement where both members are wholeheartedly committed to the project's success.
  - Plan and adhere to a schedule of regular team meetings to discuss project implementation, track progress, and address any challenges that may arise.
  - Individual contributions and the quality of collaboration will be evaluated during the project assessment.

Final Project: **Performance Optimization of Machine Learning Models on Accelerator-Enabled Infrastructure**

- **Select a Machine Learning Model**:
  - Choose a machine learning model based on specific data types: Options include **large language models (LLMs)**, **image models**, **video models**, **text models**, **audio models**, **body gesture models**, **facial expression models**, **physiological signal models, etc.**
  - Focus on **LLMs** or Small LLMs to explore modern topics like **fine-tuning** and **prompt engineering**.
  - Identify and use an applicable **open dataset** for training or inference.
  - Leverage open-source implementations in **PyTorch** or other frameworks.
- **Explore Accelerator-Enabled Infrastructure**:
  - Use **Google Cloud** or other GPU-enabled compute environments (e.g., Columbia's Terremoto/Insomnia cluster).
  - Explore alternative AI accelerators such as **TPUs**, **Raspberry Pi**, or **NVIDIA Jetson Nano** for Edge AI applications.
  - Investigate serving frameworks like **vLLM** or **FasterTransformer** to enable efficient inference scaling.
- **Focus on Modern AI Topics**:
  - Include **LLM fine-tuning** techniques: Apply lightweight methods like **LoRA (Low-Rank Adaptation)** or **qLoRA** to fine-tune large models efficiently.
  - Explore **prompt engineering**: Design prompts to optimize LLM behavior for specific tasks.
  - Evaluate **LLM inference scaling**:
    - Use tools like **vLLM**, **TGI (Text Generation Inference)**, or **DeepSpeed** to optimize serving latency and throughput.
    - Analyze the impact of features like KV cache management, batch scheduling, or mixed precision inference.
- **Detailed Model Training and Inference Study**: Conduct a detailed analysis of time spent in the following phases:
  - Data Preprocessing
  - Data Loading
  - Model Training
  - Model Inference

Final Project: **Performance Optimization of Machine Learning Models on Accelerator-Enabled Infrastructure**

- **Apply Optimization Techniques**: Implement **1-3 optimization techniques** to improve performance:
  - Techniques discussed in class.
  - Techniques from the PyTorch literature (PyTorch Tuning Guide).
  - Novel approaches from peer-reviewed research papers.
- **Examples:**
  - **Fine-tuning optimization**: Apply quantization-aware training or knowledge distillation.
  - **Inference optimization**: Use model compression (e.g., sparsification, quantization), KV cache optimizations, or batch inference tuning.
  - Data pipeline optimizations: Optimize torch.utils.data.DataLoader or implement caching mechanisms.
- **Performance Profiling and Metrics**:
  - Measure model performance using tools like **PyTorch Profiler**, **NVIDIA Profiler**, or serving-specific profilers (e.g., **vLLM logs**).
  - Capture metrics such as:
    - GPU utilization, memory bandwidth, and I/O bottlenecks.
    - Latency and throughput (for serving frameworks).
    - Timing metrics for each phase of the ML pipeline.
- **Results and Analysis**:
  - Compare performance metrics before and after optimization.
  - Provide insights into:
    - Compute, memory, and I/O improvements.
    - Latency, throughput, and efficiency gains for inference at scale.
    - Trends and trade-offs in modern AI techniques (e.g., fine-tuning vs. prompt engineering).

Project Ideas

For your final project, you will design and conduct a performance optimization study on a machine learning model of your choice. **Ensure that your project idea is well-contained and feasible to complete within 6 weeks.** Below are several focus areas and ideas to guide your project.

1) Torch.compile and Model Optimization

- Choose a machine learning model (e.g., LLMs, image, video, text, audio, physiological signals) and experiment with **torch.compile** to optimize its performance.
- Explore how **torch.compile** can be applied to improve inference or training speed and identify specific model slices that benefit most from compilation.
- Study and report on components of the **torch.compile** pathway, such as the **Inductor backend**, to better understand how these contribute to optimization.

2) LLM Inference, Fine-tuning, and Optimization

- Work on large language models (LLMs) and focus on either optimizing their inference or fine-tuning for specific tasks.
- Pick the smaller LLMs for your studies
- Consider integrating serving platforms such as VLLM or TGI (Text Generation Inference) for scalable deployment of your models.
- Analyze the improvements these platforms offer in terms of inference speed, resource usage, and scalability.

3) GPU-enabled Infrastructure and AI Accelerators

- Run your experiments on GPU-enabled infrastructure like Google Cloud, Columbia's Terremoto cluster, or another compute cluster.
- Alternatively, explore AI accelerators such as TPUs or Edge AI devices (e.g., Raspberry Pi, NVIDIA Jetson Nano).
- Analyze model performance during both training and inference on these platforms and compare resource usage.

Project Ideas

For your final project, you will design and conduct a performance optimization study on a machine learning model of your choice. **Ensure that your project idea is well-contained and feasible to complete within 6 weeks.** Below are several focus areas and ideas to guide your project.

4) Performance Optimization Study

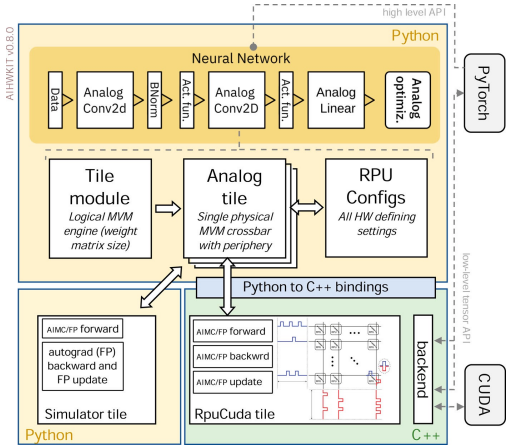
- Choose any machine learning model (LLM, image classification, video processing, etc.) and perform a comprehensive performance analysis.Identify bottlenecks in different stages of model training, such as data pre-processing, data loading, training, and inference.
- Apply at least 2-3 optimization techniques, such as:Data loading optimizations
  - Distributed training strategies
  - Mixed precision or quantization
  - Knowledge distillation, sparsification, or advanced techniques like:
    - Flash Attention (faster attention mechanisms in transformers)
    - Flex Attention (dynamic attention mechanism for efficiency)
    - Speculative Decoding (for faster text generation)
    - KV Cache Optimizations (key-value cache optimization for reducing redundant computations in transformers)
  - Incorporate techniques discussed in class or novel methods from recent research literature.

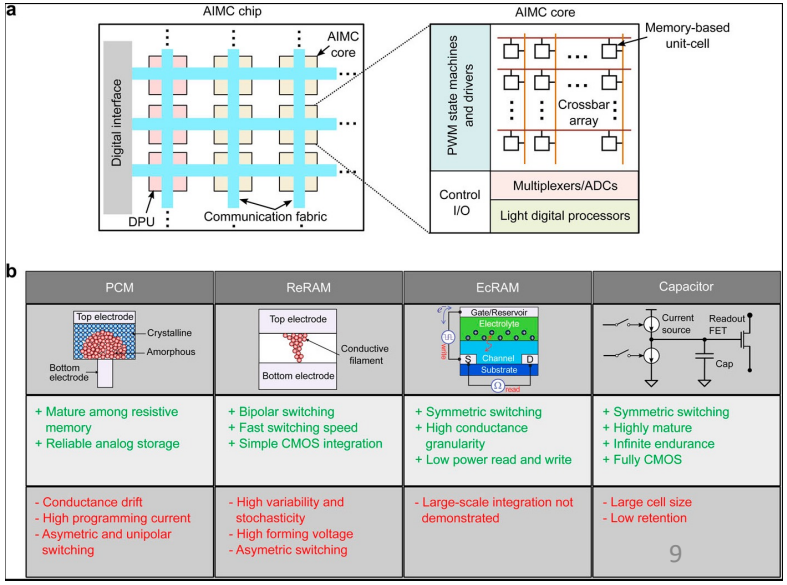
5) Research Paper Reproduction and Improvement

- Choose a research paper from top conferences like NeurIPS, ICLR, ICML, or CVPR that aligns with one of the themes (e.g., LLM optimization, torch.compile, inference optimization, AI hardware, etc.).
- Some suggested themes to explore in these conferences include performance optimizations for transformers, new attention mechanisms (e.g., Flash/Flex Attention), distributed training, and inference improvements.
- Attempt to reproduce the results presented in the paper using the tools and platforms available.
- Where possible, go beyond reproduction and aim to improve upon the results by applying additional optimizations, more efficient techniques, or new strategies not covered in the original paper.
- Provide a thorough analysis comparing your reproduced results to the original paper's findings and highlight any improvements you achieved.

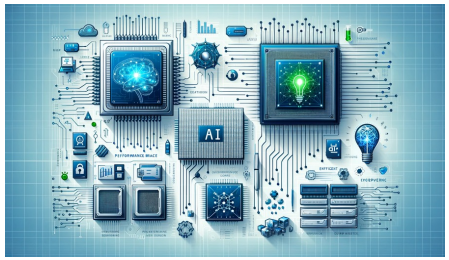
Other Project Ideas

- Explore emerging AI accelerators such as Analog AI accelerators for certain neural network architectures (E.g., Visual wake words, Bert model, LSTM model, etc).
- Use the IBM Analog AI hardware accelerator and explore the accuracy of the chosen models on various analog devices: PCM, RRAM, ECRAM.
- Perform hardware-aware training and assess the impact of hardware-aware training on drift. Contribute the results of your study as an example to the open-source AIHWKIT.
- You can access the toolkit at: https://github.com/IBM/aihwkit





Your novel ideas are welcome



- You can also propose your unique project ideas if they have enough complexity and are aligned with the goals of the HPML class

Detailed Reporting and Metrics

For **all projects**, you are expected to:

Measure performance metrics such as computation time, memory usage, and I/O utilization before and after applying optimizations.

Use profiling tools like the **PyTorch profiler** or **NVIDIA profilers** (or other hardware-specific profiling tools) to gather data on where time and resources are spent during training and inference.

Present a detailed report comparing pre- and post-optimization results, focusing on improvements in key metrics like time per iteration, memory footprint, and throughput.

Analyze the impact of your chosen optimization techniques or infrastructure on the model's overall performance and provide insights on which techniques or configurations yield the best results.

Project Deliverables

1. Code:

- All code must be written in **Python** or **Python Jupyter Notebooks**.
- Organize your code in a **GitHub repository** with a properly formatted **README file** that includes:
  - Instructions to run and recreate your results.
  - Dependencies and environment setup details.
  - Clear documentation for all code, scripts and notebooks.

2. GitHub Commit Tracking:

- All participants are required to contribute to the GitHub repository.
- The project will include an evaluation of individual participation based on GitHub commit history.

3. Experiment Tracking:

- Use the **Weights and Biases (WandB)** platform for tracking experiments.
- Set up project dashboards to log: Model configurations, Training and validation metrics, and Hyperparameter tuning experiments.
- Share your WandB dashboard link in the final submission.
- Weights and Biases platform: Use this for comprehensive experiment tracking and visualizations.

4. Final Project Report

- Write your report in **LaTeX** using **Overleaf**.

5. Bonus Opportunities: Up to **10% bonus** for

- **Medium or Blog Post Articles**: Write an engaging blog post or article summarizing your work, including a step-by-step guide or insights gained. Provide evidence with a published link.
- **Open-Source Contributions**: Submit a pull request (PR) to an existing open-source project with meaningful contributions. Provide evidence with PR links and descriptions.
- **Novel Results**: Achieve results that are publishable in conferences or journals.
- **Public Repositories**: Contribute back to the open-source community with useful code, datasets, or tools stemming from your project.

6. Final Project Presentation

- Create a presentation in PowerPoint (PPT) format.
- Use the provided class template.

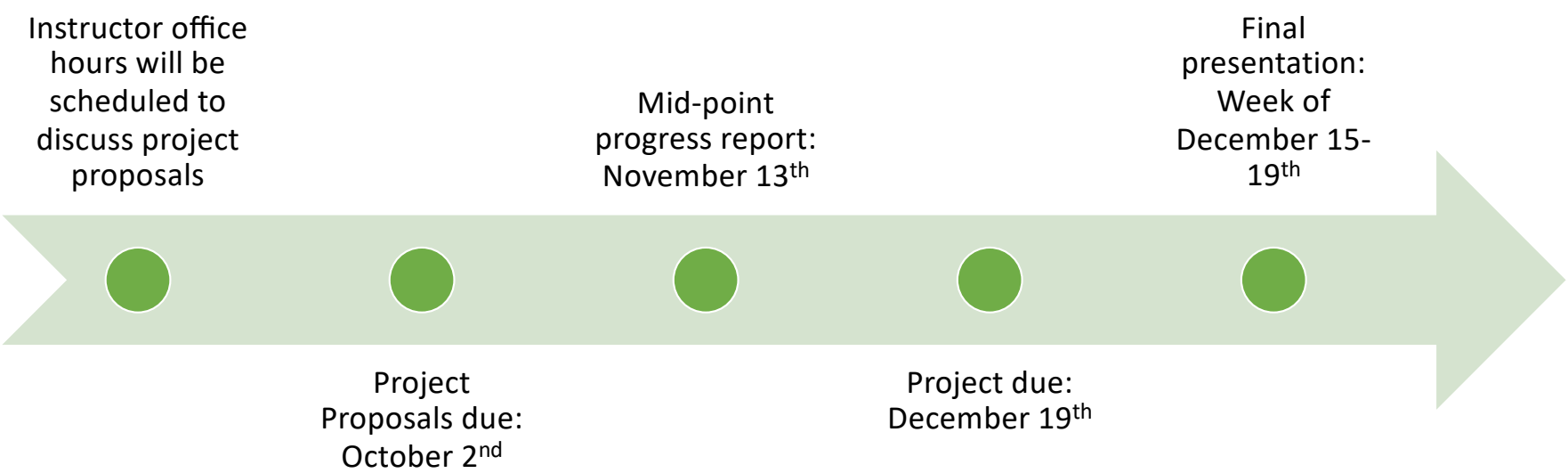
Final Project Report

- Write your report in **LaTeX** using **Overleaf**.
- Use the official **IEEE LaTeX format** for publication:
  - IEEE Template.
  - Overleaf Platform.
- Ensure the report is well-organized and professionally formatted.
  - **Abstract**: Summarize your project goals, methodology, and key results.
  - **Introduction**: Provide background and motivation for the project.
  - **Models and Data Description**: Describe the model(s) selected and the dataset(s) used.
  - **Training and Profiling Methodology**: Detail your training setup, profiling tools, and data processing techniques.
  - **Performance Tuning Methodology**: Explain the optimization methods applied (e.g., tuning hyperparameters, inference optimization).
  - **Experimental Results**: Present performance metrics (before and after optimizations), visualizations, and analysis.
  - **Conclusion**: Summarize findings, highlight limitations, and propose future work.

Project Grading

- **Grading scheme:**
  - 5% Project proposal
  - 25% on the quality of code, profiling, documentation, and reproducibility of the results
  - 20% on wandb experiments tracking and dashboard
  - 25% on the technical contributions, novelty, performance methodology, and analysis of the results
  - 10% on the presentation + Q/A performance
  - 15% Quality of the final report quality
  - 10% Bonus: For any novel or new contributions that can lead to workshop/conference publishable results, blog post such as Medium, or to existing open-source contributions

Project Timeline



Resources

- DL Optimizations:
  - https://towardsdatascience.com/deep-learning-model-optimizations-made-easy-or-at-least-easier-a0255182f427
  - https://towardsdatascience.com/optimize-pytorch-performance-for-speed-and-memory-efficiency-2022-84f453916ea6

- Model Zoo: https://pytorch.org/serve/model_zoo.html
- PyTorch datasets: https://pytorch.org/vision/stable/datasets.html
- Hugging face datasets: https://huggingface.co/datasets
- Hugging face model zoo: https://huggingface.co/models
- MLPerf benchmarks: https://mlcommons.org/
- PyTorch Profiler: https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html
- Weight and Biases: https://wandb.ai/site
- LLM optimizations:
  - https://attri.ai/blog/mastering-llm-optimization-with-these-5-essential-techniques
  - https://resources.nvidia.com/en-us-ai-large-language-models/mastering-llm-inference
  - https://platform.openai.com/docs/guides/optimizing-llm-accuracy/understanding-the-tools

Additional Resources

1. **LLM Fine-Tuning**:
    1. Hugging Face Transformers: Fine-tuning guides for popular models like BERT, GPT, and T5.
    2. LoRA (Low-Rank Adaptation): Lightweight fine-tuning method.
    3. qLoRA: Quantization for efficient fine-tuning.
2. **Prompt Engineering**:
    1. Anthropic's Guide to Prompt Engineering: Guidelines for designing optimal prompts.
    2. OpenAI API: Practical tips for using prompts with GPT models.
3. **Serving Frameworks**:
    1. vLLM: High-throughput serving framework for LLMs with optimized KV cache management.
    2. TGI (Text Generation Inference): Open-source inference serving for Hugging Face models.
    3. DeepSpeed-Inference: Optimized inference and training at scale.
4. **Optimization and Profiling Tools**:
    1. NVIDIA Nsight Systems: GPU profiling for inference pipelines.
    2. PyTorch Profiler: Profiling guide for training and inference pipelines.
    3. TorchBench: Benchmarking suite for PyTorch models.
5. **Research and Publications**:
    1. "Efficient Fine-Tuning of Large Pretrained Language Models" (LoRA Paper).
    2. "vLLM: A High-Throughput Optimized Serving System for Large Language Models."

Resources on Analog AI

- Analog AI Hardware Accelerator Toolkit (aihwkit): https://github.com/IBM/aihwkit
- Documentation: https://aihwkit.readthedocs.io/en/latest/
- Example Notebooks: https://github.com/IBM/aihwkit/tree/master/notebooks
- Publications: https://researcher.watson.ibm.com/researcher/view_group.php?id=7716
- Video on what is Analog AI: https://www.youtube.com/watch?v=BTnr8z-ePR4&t=7s
- Tutorial Journal paper: https://pubs.aip.org/aip/aml/article/1/4/041102/2923573/Using-the-IBM-analog-in-memory-hardware

Sample Projects for Prior Semesters

EfficientMedSAM: Accelerating Medical Image Segmentation via Neural Architecture Search and Knowledge Distillation

Arnold Caleb Asiimwe, *Columbia University*, New York, USA, aa4870@columbia.edu
William Das, *Columbia University*, New York, USA, whd2108@columbia.edu
Hadjer Benmeziane, *IBM Research Europe*, Rüschlikon, Switzerland, Hadjer.Benmeziane1@ibm.com
Kaoutar El Maghraoui, *IBM T. J. Watson Research Center*, Yorktown Heights, NY, USA, kelmaghr@us.ibm.com

***Abstract*—Medical image segmentation is crucial for precise diagnosis, treatment planning, and disease monitoring in clinical practice. Despite the advancements in segmentation models inspired by the Segment Anything Model (SAM) architecture, their real-time application is limited due to computational inefficiencies arising from transformer-based architectures. We introduce EfficientMedSAM, a suite of high-speed, memory-efficient foundation models for universal medical image segmentation. Our approach leverages differentiable neural architecture search (NAS) to explore a novel search space emphasizing efficient operations over traditional attention mechanisms. The generated candidate architectures are further refined through knowledge distillation from larger MedSAM models and evaluated on the Kvasir dataset of endoscopic images. EfficientMedSAM achieves competitive mean Average Precision (mAP) while substantially reducing Multiply-Accumulate operations (MACs) and model parameters, enhancing throughput. We integrate a knowledge distillation (KD) pipeline that transfers knowledge from logits and attention maps, using saliency maps as proxies for attention map-based distillation. Our findings establish a proof of concept for large-scale distributed training on the SA-Med2D-20M dataset, paving the way for real-time medical image segmentation advancements.**

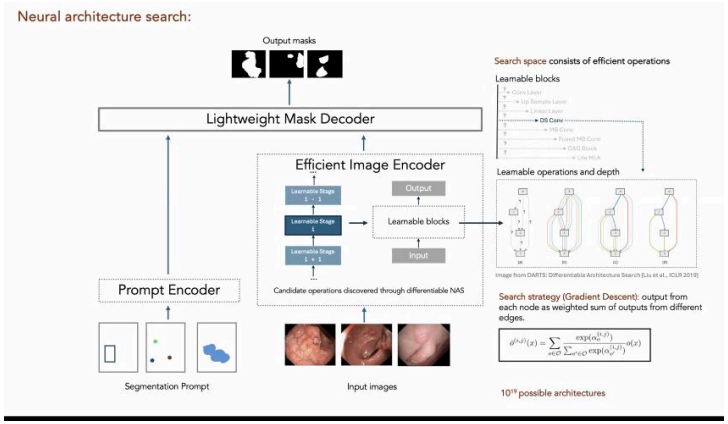
***Index Terms*—Medical Image Segmentation, Neural Architecture Search, Knowledge Distillation, Edge Computing**

memory-inefficient operations such as tensor reshaping and element-wise functions, is a major bottleneck in terms of inference speed [33, 9]. Moreover, the redundancy in attention maps across different heads leads to unnecessary computational overhead [19, 32]. These challenges hinder the deployment of transformer-based image segmentation models in resource-constrained environments and limit their practical utility.

To address these limitations, we propose **EfficientMedSAM**, a family of high-speed and memory-efficient foundation models for universal medical image segmentation. Our approach leverages differentiable neural architecture search [16, 39] to discover efficient alternatives to attention mechanisms in the transformer architecture. By exploring a search space that includes memory-efficient operations, we identify candidate architectures that strike a balance between computational efficiency and segmentation performance. We further enhance these architectures by distilling knowledge from larger MedSAM models [7, 24], allowing them to inherit the strong representational power of the teacher models while maintaining a lightweight structure.

Published at IEEE Edge Conference 2024

- EfficientMedSAM is a suite of high-speed, memory-efficient medical image segmentation models. Designed to overcome the computational inefficiencies of traditional transformer-based architectures.
- EfficientMedSAM utilizes differentiable neural architecture search (NAS) and knowledge distillation from larger models.
- This approach allows EfficientMedSAM to achieve competitive performance while significantly reducing computational load and memory usage.



Sample Projects for Prior Semesters

Exploring Inference Acceleration for Large Language Models

Wookje Han
Computer Science Department
Columbia University
New York, NY
wh2571@columbia.edu

Reed Evans
Computer Science Department
Columbia University
New York, NY
re2449@columbia.edu

Edison Chen
Computer Science Department
Columbia University
New York, NY
zc2738@columbia.edu

***Abstract*—Large Language Models (LLMs) have seen widespread adoption in academia and industry thanks to their capabilities in natural language processing and generation, but these models are often memory-intensive while failing to utilize much of the compute capabilities of modern parallel hardware. In this paper, we are the first to implement speculative streaming [1] to a widely used LLM, GPT-Neo [2] [3], and observe a significant compute speedup while keeping the memory footprint increase to a minimum. We compare it to FlashAttention-2 [4] and quantized models and find comparable performance boosts to both methods.**

***Index Terms*—LLM, Transformer, optimization, performance, parallelism**

I. Introduction

Inference with Large Language Models (LLMs) has be-

resources more effectively. This study is the first of its kind to integrate speculative streaming with GPT-Neo [2] [3].

We also explore another popular inference optimization technique, FlashAttention-2 [4] using the author's reference implementation. FlashAttention-2 solves the second bottleneck by optimizing for attention computations at the GPU kernel level.

Lastly, we examine a model compression technique: Quantization [6] to the bfloat16 [7] data type. This alleviates the third bottleneck of memory-boundedness by reducing the amount of memory the model uses at the cost of precision. We compare token-per-second results for all three optimization methods against the baseline.

II. Models and Data Description

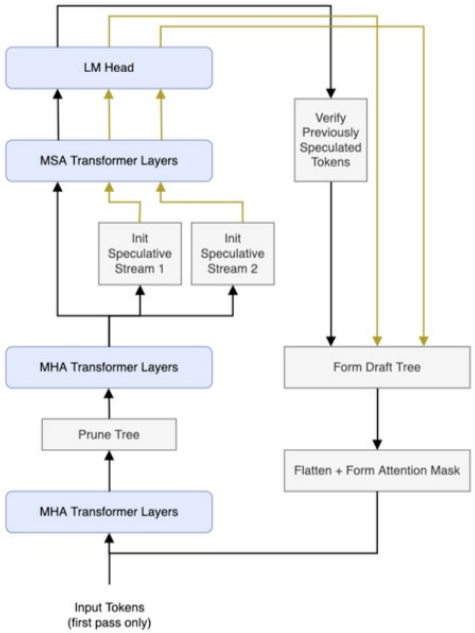
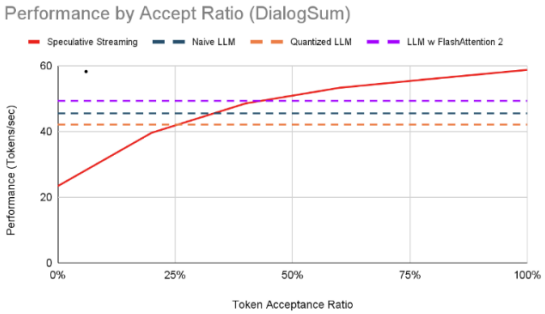


Fig. 1. Overall process of Speculative Streaming



Sample Projects for Prior Semesters

- The project proposes a novel **Neural Architecture Search (NAS)** method for **analog in-memory computing (IMC)**devices.
- A **Reinforcement Learning (RL) agent** is used, leveraging a **supervised Vector Quantized-Variational AutoEncoder (S-VQVAE)** to create a hardware-aware latent search space.
- The approach balances computational efficiency and high performance on **CIFAR-10** under analog constraints.
- This method reduces computational resources and search time by **over three times** compared to traditional evolutionary algorithms while maintaining comparable accuracy.

Enhancing IBM's Analog-NAS with a Reinforcement Learning NAS Agent

Sujeeth Bhavanam
Columbia University
New York, NY
sb4839@columbia.edu

Tawab Safi
Columbia University
New York, NY
as7092@columbia.edu

***Abstract*—This project introduces an innovative approach to neural architecture search (NAS) for analog in-memory computing (IMC) devices using a Reinforcement Learning (RL) based agent. By integrating a supervised Vector Quantized-Variational AutoEncoder (S-VQVAE), we develop a novel latent search space that is both hardware-aware and optimized for efficient exploration. The RL agent navigates this space, effectively selecting architectures that balance computational efficiency with high performance on the CIFAR-10 dataset under analog constraints. Our method significantly reduces the computational resources and search time by over three times compared to traditional evolutionary algorithms while achieving comparable accuracy, thus demonstrating a scalable and efficient approach to NAS in constrained hardware environments.**

Acknowledgement: Thank you Professor Kaoutar El Maghraoui for the opportunity and Dr Hadjer Benmeziane for your mentorship through the project.

1. Introduction

used in [1] while utilizing significantly less computational resources, reducing search time by a factor of more than 3.

2. Related Work

The pursuit of optimizing neural architecture search (NAS) using reinforcement learning has been extensively explored in the literature. Zoph and Le [2] initially demonstrated the use of reinforcement learning for designing neural network architectures, setting a foundational approach for subsequent research in this field. Further advancing the capabilities of NAS, Cai et al. [3] introduced an efficient method by leveraging network transformations, allowing for iterative improvements on existing architectures. Concurrently, the incorporation of graph-based methods as seen in the work by Wen et al. [4], where operation embeddings were utilized, signifies a shift towards more complex representations within NAS, enabling a nuanced understanding of architectural components.

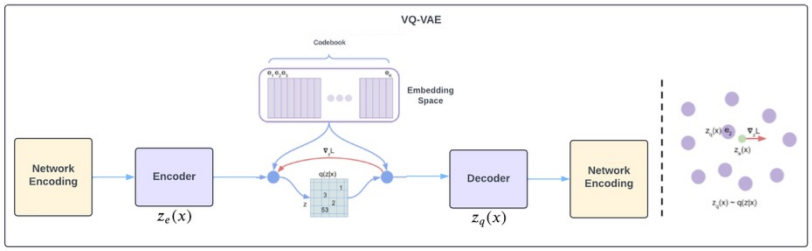


Figure 3: VQ-VAE Architecture.

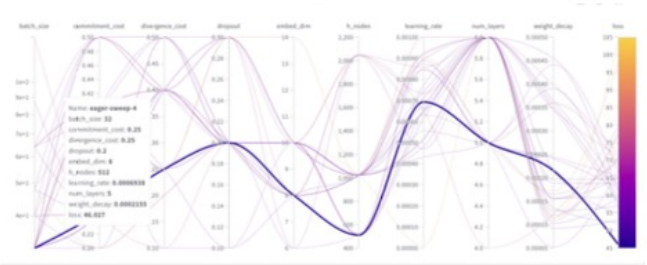


Figure 6: Hyperparameter Search.

Sample Projects for Prior Semesters

TVMerge-Stable-Diffusion: Optimized Diffusion Model Inference with TVM Compiler and Token Merging with Retraining

Yiming Li
yl5436@columbia.edu

Paul Huang
xh2373@columbia.edu

***Abstract*—Text-guided image synthesis has witnessed significant advancements with the advent of diffusion models. One recent example is Stable Diffusion. However, while offering remarkable capabilities in generating high-fidelity images from textual prompts, the computational demands of Stable Diffusion's inference process pose challenges for real-time and resource-constrained scenarios.**

In this paper, we implement an algorithmic and engineering approach to accelerate Stable Diffusion's inference without re-training. Leveraging techniques such as token merging (ToMe), the Tensor Virtual Machine (TVM), and the XFormers library from Meta, we accelerate the inference pipeline while preserving image quality. Our experiment results show up to 38%, 37%, and 17% of speedups using the TVM, XFormers, and token merging, respectively.

without necessitating re-training. By leveraging techniques such as token merging (ToMe) [2] to reduce the computational burden imposed by the transformer-based components and harnessing the capabilities of the Tensor Virtual Machine (TVM) [3] for optimized model deployment across diverse hardware platforms, we endeavor to streamline the inference pipeline while preserving the fidelity of generated images.

II. Background

A. Stable Diffusion

Stable Diffusion [1] is an open-source text-to-image model that allows for the creation of images in various styles and

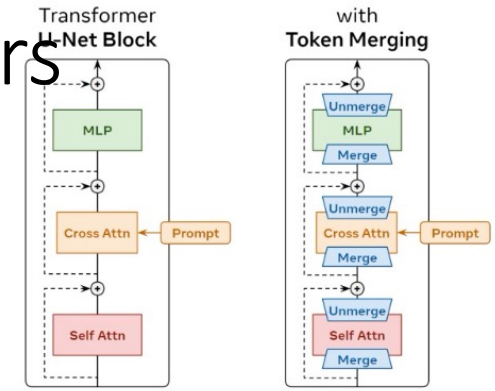


Fig. 1: Adopted from [8], an illustration of the token merging and unmerging concept applied to U-Net transformer blocks, namely the self-attention, cross-attention, and multi-layer perceptron (MLP) blocks.

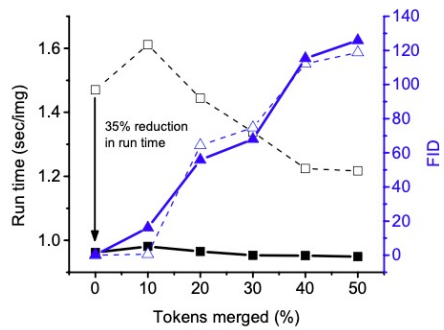
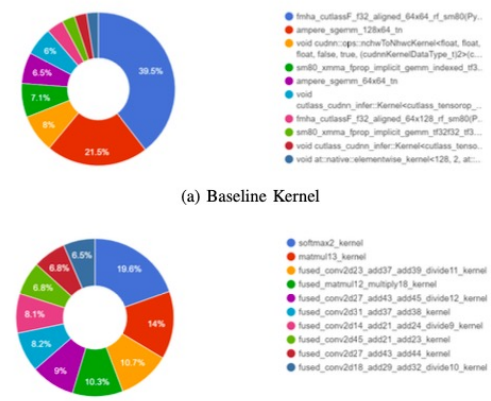


Fig. 4: The run time and FID score (solid symbols) of Stable Diffusion inference with the XFormers library. We also show the results from running without XFormers (hollow symbols) for comparison.



(a) Baseline Kernel

(b) TVM Optimized Kernel