# Automated Hyperparameter Optimization of Large Language Models using LoRA and Optuna

[*]Submitted as part of MSML604: Introduction to Optimization, Spring 2025

1st NandaKiran Velaga
*M.S. Applied Machine Learning*
*University of Maryland, College Park*
College Park, MD, USA
nvelaga@umd.edu

2th Venkata Revanth Vardineni
*M.S. Applied Machine Learning*
*University of Maryland, College Park*
College Park, MD, USA
vvr2211@umd.edu

3rd Anirudh Krishna
*M.S. Applied Machine Learning*
*University of Maryland, College Park*
College Park, MD, USA
anirudhk@umd.edu

4th Phanindra Tupakula
*M.S. Applied Machine Learning*
*University of Maryland, College Park*
College Park, MD, USA
ptupakul@umd.edu

5nd Sri Akash Kadali
*M.S. Applied Machine Learning*
*University of Maryland, College Park*
College Park, MD, USA
kadali18@umd.edu

***Abstract*—Large Language Models (LLMs) such as T5 and LLaMA have revolutionized NLP but require meticulous hyperparameter tuning to achieve optimal performance. This project presents an automated, scalable framework for hyperparameter optimization using Optuna, Bayesian Optimization, and LoRA-based fine-tuning. We explore a multi-objective formulation that balances validation loss and computational cost under GPU and memory constraints. Our pipeline integrates Hugging Face Transformers, PyTorch, and CUDA-based infrastructure, achieving improved ROUGE and BLEU scores while reducing training time and memory usage. The proposed framework enables reproducible, cloud-ready tuning of LLMs, making it suitable for real-world ML deployment.**

***Index Terms*—LLM, Hyperparameter Optimization, Optuna, LoRA, T5, Transformers**

## I. Introduction

The rapid evolution of Large Language Models (LLMs), such as GPT, BERT, and T5, has significantly advanced the field of Natural Language Processing (NLP). However, as the complexity and scale of these models continue to grow, so does the challenge of tuning them effectively. Hyperparameter tuning plays a crucial role in determining the model's accuracy, training efficiency, and computational cost.

This project aims to automate the process of hyperparameter optimization for LLMs using scalable and reproducible pipelines. We leverage modern optimization strategies such as Bayesian Optimization, Population-Based Training, and gradient-free methods to systematically explore high-dimensional hyperparameter spaces. The project also incorporates the use of containerized workflows and MLOps tools to ensure scalable experimentation and reproducibility.

This project was conducted under the guidance of Prof. Richard J. La and Amogha Sunil at the University of Maryland, College Park.

Our experiments focus on transformer-based architectures like T5 and LLaMA, deploying fine-tuning strategies such as Low-Rank Adaptation (LoRA) to reduce computational overhead. This end-to-end framework is designed to support cloud-native deployment and rigorous experiment tracking, making it suitable for both academic research and industrial applications.

## II. Problem Description and Motivation

Large Language Models (LLMs) have become central to many NLP applications due to their high accuracy and generalization capabilities. However, their performance is highly sensitive to the choice of hyperparameters such as learning rate, batch size, number of training epochs, and optimizer configuration. Manual tuning of these hyperparameters is not only time-consuming but also computationally expensive and suboptimal, especially when operating within resource-constrained environments.

The objective of this project is to automate and optimize the hyperparameter tuning process for LLMs to improve model performance and training efficiency. We aim to find the best trade-off between predictive accuracy and resource utilization using multi-objective optimization techniques. By leveraging Bayesian Optimization and tools like Optuna, we enable systematic exploration of complex, high-dimensional hyperparameter spaces.

The motivation behind this work stems from the increasing need to make LLM training more accessible and efficient. With growing deployment in cloud and edge environments, there is a pressing demand for scalable, automated, and cost-effective tuning solutions that can adapt to diverse constraints and datasets. Our work bridges this gap by integrating optimization frameworks into a complete training and deployment pipeline.

## III. Related Work

Hyperparameter optimization has been an active area of research in machine learning and deep learning, especially for large-scale models. Traditional approaches like grid search and random search, though simple, are inefficient and fail to scale with high-dimensional search spaces typical in modern LLMs.

Bayesian Optimization has emerged as a powerful alternative, offering sample-efficient exploration of hyperparameter configurations by building a surrogate model of the objective function. Tools such as Optuna, Hyperopt, and Ray Tune implement advanced strategies like Tree-structured Parzen Estimators (TPE) and multi-fidelity optimization to speed up convergence.

Population-Based Training (PBT) is another technique that has gained popularity in training large models. It maintains a pool of models and iteratively refines them by exploiting the best-performing configurations while exploring new ones. This technique is especially useful in scenarios with noisy evaluations or non-convex objectives.

In addition, recent work on parameter-efficient fine-tuning methods such as LoRA (Low-Rank Adaptation) has significantly reduced the computational burden of training large transformers. LoRA modifies only a subset of parameters, making it suitable for scenarios with limited hardware resources.

Despite these advances, many existing studies either focus narrowly on single-objective optimization or lack integration with scalable MLOps practices. Our work addresses these gaps by combining parameter-efficient training, multi-objective optimization, and reproducible experimentation in a cloud-ready framework.

## IV. Methodology

Our methodology focuses on building a scalable, automated framework for hyperparameter optimization of Large Language Models (LLMs), specifically the T5 model, using the Optuna optimization library. The approach integrates modern parameter-efficient training methods, a well-defined search space, and cloud-native tools to ensure repeatability and efficiency.

### *A. Model Architecture: T5 with LoRA*

We fine-tune the T5-small model using Low-Rank Adaptation (LoRA), a lightweight technique that injects low-rank matrices into existing model layers. This enables faster training and lower GPU memory usage while maintaining competitive performance, especially for tasks like summarization.

### *B. Search Space and Objective*

The hyperparameters explored include learning rate, batch size, number of training epochs, dropout rate, and LoRA-specific configurations (e.g., rank and alpha). The primary objective is to minimize validation loss while balancing computational cost such as GPU memory usage and training time.

### *C. Bayesian Optimization using Optuna*

We use Optuna's Tree-structured Parzen Estimator (TPE) sampler for Bayesian optimization. It models the performance of past trials to propose promising configurations for future ones. Pruning is applied to discard underperforming trials early, thus reducing total resource consumption.

### *D. Infrastructure and Tools Used*

Experiments are run on GPU-enabled environments using Google Colab and local CUDA systems. The pipeline uses Hugging Face Transformers for model training, and the training workflow is modularized using PyTorch. Model checkpoints and metrics are logged, and training is reproducible with controlled random seeds and environment captures.

This methodology ensures a robust and efficient optimization process that is adaptable to different compute environments and datasets.

### *E. Model Architecture: T5 with LoRA*

We selected the T5-small model as the base architecture due to its encoder-decoder structure, which is well-suited for text-to-text tasks like summarization. To reduce training cost and memory usage, we applied Low-Rank Adaptation (LoRA), which introduces trainable low-rank matrices to selected linear layers. This allows the model to learn task-specific adaptations without updating all weights, making it ideal for resource-constrained fine-tuning scenarios.

### *F. Search Space and Objective*

The hyperparameter search space includes:

- Learning rate: [$1e$-5, $1e$-3] (log-uniform)
- Batch size: $\{8, 16, 32\}$
- Number of epochs: $\{2, 3, 4\}$
- LoRA rank: $\{4, 8, 16\}$
- LoRA alpha: $\{16, 32, 64\}$

The primary objective is to minimize validation loss while maintaining acceptable computational cost. For multi-objective optimization, we define a composite function: validation loss + $\lambda \times$ GPU memory usage.

### *G. Bayesian Optimization using Optuna*

We used Optuna's TPE (Tree-structured Parzen Estimator) sampler to model the likelihood of good configurations based on prior trial outcomes. The optimizer suggests new hyperparameter sets by maximizing the expected improvement. Additionally, a pruning mechanism is implemented to halt poorly performing trials early, saving computational resources and accelerating convergence.

### *H. Infrastructure and Tools Used*

Our implementation stack includes:

- **Transformers and Tokenizers:** Hugging Face
- **Training Framework:** PyTorch and PEFT (Parameter-Efficient Fine-Tuning)
- **Optimization:** Optuna for Bayesian hyperparameter tuning

- **Execution Environment:** Google Colab and CUDA-capable local machines
- **Monitoring:** Manual metric logging with optional integration for TensorBoard or Weights & Biases

Experiments are reproducible with fixed random seeds and saved checkpoints, and results are validated across multiple trials.

## V. Mathematical Formulation

Hyperparameter optimization can be formalized as a constrained multi-objective optimization problem where the goal is to discover a hyperparameter vector $\mathbf{h}$ that minimizes validation loss $\mathcal{L}(\mathbf{h})$ and computational cost $\mathcal{C}(\mathbf{h})$, subject to resource and feasibility constraints.

### A. Hyperparameter Vector and Domain

Let $\mathbf{h} = [h_1, h_2, \dots, h_n]^T$ denote the $n$-dimensional hyperparameter vector where each $h_i$ belongs to a domain $\mathcal{H}_i$. The overall feasible hyperparameter space is defined as:

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_n \tag{1}$$

Each $\mathcal{H}_i$ may be continuous (e.g., $h_i \in [a_i, b_i]$), discrete (e.g., $h_i \in \{2, 4, 8\}$), or categorical (e.g., $h_i \in \{\text{Adam}, \text{RMSprop}\}$).

### B. Validation Loss Function

Let $f(x; \mathbf{h})$ be the output of a model trained with hyperparameters $\mathbf{h}$ for input $x$. The validation loss is defined as:

$$\mathcal{L}(\mathbf{h}) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y) \in \mathcal{D}_{\text{val}}} \ell(f(x; \mathbf{h}), y) \tag{2}$$

where $\ell(\cdot, \cdot)$ is the task-specific loss function (e.g., cross-entropy), and $\mathcal{D}_{\text{val}}$ is the validation set.

### C. Computational Cost Modeling

Let $\mathcal{C}(\mathbf{h})$ be the scalar computational cost associated with training the model under configuration $\mathbf{h}$. This cost can be decomposed as:

$$\mathcal{C}(\mathbf{h}) = \alpha \cdot T(\mathbf{h}) + \beta \cdot M(\mathbf{h}) + \gamma \cdot E(\mathbf{h}) \tag{3}$$

where $T(\mathbf{h})$ is the total training time, $M(\mathbf{h})$ is the peak memory usage, and $E(\mathbf{h})$ is the energy or GPU utilization. The coefficients $\alpha, \beta, \gamma \in \mathbb{R}^+$ weight their relative importance.

### D. Scalarized Objective Function

For simplicity and tractability, the objectives can be scalarized into a single function using a trade-off parameter $\lambda \geq 0$:

$$F(\mathbf{h}) = \mathcal{L}(\mathbf{h}) + \lambda \cdot \mathcal{C}(\mathbf{h}) \tag{4}$$

A low $\lambda$ prioritizes model accuracy, whereas a high $\lambda$ emphasizes resource efficiency.

### E. Constraints

The optimization is subject to the following constraints:

$$h_i^{\min} \leq h_i \leq h_i^{\max}, \quad \forall i = 1, 2, ..., n \tag{5}$$

$$T(\mathbf{h}) \leq T_{\max} \quad \text{(time constraint)} \tag{6}$$

$$M(\mathbf{h}) \leq M_{\max} \quad \text{(memory constraint)} \tag{7}$$

Let $\mathcal{G} = \{g_j(\mathbf{h}) \leq 0\}_{j=1}^{m}$ denote any additional constraints (e.g., latency, cost per epoch).

### F. Final Optimization Problem

The complete optimization formulation is thus given by:

$$\begin{aligned} \min_{\mathbf{h} \in \mathcal{H}} \quad & F(\mathbf{h}) = \mathcal{L}(\mathbf{h}) + \lambda \cdot \mathcal{C}(\mathbf{h}) \\ \text{s.t.} \quad & g_j(\mathbf{h}) \leq 0, \quad \forall j = 1, \dots, m \end{aligned} \tag{8}$$

### G. Search Strategy: Bayesian Optimization

We utilize Bayesian Optimization (BO) to efficiently navigate the hyperparameter space. BO constructs a surrogate model $S(\mathbf{h})$ to approximate $F(\mathbf{h})$ and uses an acquisition function $\mathcal{A}(\mathbf{h}; S)$ to propose the next point to evaluate:

$$\mathbf{h}_{t+1} = \arg\max_{\mathbf{h} \in \mathcal{H}} \mathcal{A}(\mathbf{h}; S_t) \tag{9}$$

where $S_t$ is updated at each iteration $t$ based on the past evaluated tuples $\{(\mathbf{h}_k, F(\mathbf{h}_k))\}_{k=1}^{t}$.

### H. Multi-Objective Extension (Optional)

In a true multi-objective setting, the goal is to find a Pareto front $\mathcal{P}$ such that no solution in $\mathcal{P}$ can be improved in one objective without degrading the other. Formally:

$$\mathcal{P} = \{\mathbf{h} \in \mathcal{H} \mid \nexists\, \mathbf{h}' \in \mathcal{H} \text{ such that } \mathcal{L}(\mathbf{h}') \leq \mathcal{L}(\mathbf{h}) \ \wedge \ \mathcal{C}(\mathbf{h}') < \mathcal{C}(\mathbf{h})\} \tag{10}$$

This extension is particularly relevant for users interested in model deployment trade-offs.

### I. Conclusion

This mathematical framework captures the essence of automated hyperparameter tuning for LLMs under resource constraints. It lays the foundation for algorithmic strategies that balance performance, cost, and feasibility in real-world training scenarios.

### J. Final Optimization Problem Statement

The goal of hyperparameter optimization is to find the optimal configuration $\mathbf{h}^* \in \mathcal{H}$ that yields the best trade-off between validation loss and computational cost. We formulate this as a constrained scalarized optimization problem:

$$\begin{aligned} \mathbf{h}^* = \arg\min_{\mathbf{h} \in \mathcal{H}} \quad & F(\mathbf{h}) = \mathcal{L}(\mathbf{h}) + \lambda \cdot \mathcal{C}(\mathbf{h}) \\ \text{subject to} \quad & h_i^{\min} \leq h_i \leq h_i^{\max}, \quad \forall i \in \{1, \dots, n\} \\ & T(\mathbf{h}) \leq T_{\max} \\ & M(\mathbf{h}) \leq M_{\max} \\ & g_j(\mathbf{h}) \leq 0, \quad \forall j \in \{1, \dots, m\} \end{aligned} \tag{11}$$

Here:

- $\mathcal{L}(\mathbf{h})$ is the validation loss associated with hyperparameter vector $\mathbf{h}$.
- $\mathcal{C}(\mathbf{h})$ denotes the computational cost (e.g., training time, GPU memory).
- $\lambda \geq 0$ is a scalar that adjusts the importance between performance and efficiency.
- $g_j(\mathbf{h})$ denotes any additional problem-specific constraints.

This formulation enables scalable and automated tuning strategies that are both resource-aware and performance-driven, especially useful for training large models like T5 or LLaMA.

## VI. Handling Randomness in Training

Training large language models involves inherent randomness due to factors like stochastic weight initialization, mini-batch sampling, and non-deterministic parallel computation. This randomness can significantly affect the performance metrics (e.g., validation loss), making hyperparameter evaluations noisy. To ensure robustness and reliability of optimization, we incorporate the following strategies.

### *A. Surrogate Modeling for Random Objectives*

In Bayesian Optimization, we use surrogate models to approximate the underlying objective function. When the objective is random (e.g., noisy validation loss), the surrogate model accounts for uncertainty using probabilistic techniques such as Gaussian Processes or Tree-structured Parzen Estimators (TPE). These models maintain a distribution over performance outcomes, and acquisition functions like Expected Improvement (EI) or Upper Confidence Bound (UCB) are used to balance exploration and exploitation under uncertainty.

### *B. Averaging over Random Seeds*

To mitigate the impact of stochastic variation, we evaluate each hyperparameter configuration $\mathbf{h}$ across multiple random seeds. Let $\mathcal{L}_s(\mathbf{h})$ denote the validation loss under seed $s$. The aggregated performance metric is:

$$\bar{\mathcal{L}}(\mathbf{h}) = \frac{1}{S}\sum_{s=1}^{S}\mathcal{L}_s(\mathbf{h}) \tag{12}$$

This averaging yields a more reliable estimate of model performance and smooths out outliers caused by unfavorable initializations.

### *C. Multi-Fidelity and Early Stopping*

To reduce the resource consumption of poorly performing configurations, we adopt multi-fidelity optimization techniques like Hyperband. These methods allocate small budgets (e.g., few epochs or subsets of data) to all configurations initially and increase resources only for promising candidates. Early stopping is employed to terminate trials that fall below a dynamic threshold, saving computation and allowing faster convergence:

$$\text{If } \mathcal{L}_t(\mathbf{h}) > \theta_t \Rightarrow \text{stop trial at step } t \tag{13}$$

where $\theta_t$ is the pruning threshold at step $t$.

### *D. Population-Based Training*

Population-Based Training (PBT) operates on a population of models with different hyperparameter settings. At regular intervals, models with poor performance are replaced with better-performing ones, inheriting weights and slightly mutated hyperparameters:

$$\mathbf{h}_{\text{new}} = \mathbf{h}_{\text{best}} + \epsilon \cdot \Delta\mathbf{h} \tag{14}$$

This dynamic evolution of hyperparameters enables on-the-fly adaptation and resilience against local optima and random fluctuations.

### *E. Stopping Criteria and Stability*

To ensure training stability and avoid indefinite runs, we define convergence thresholds based on a fixed number of trials, minimal change in validation loss, or stagnation in acquisition score. A configuration is considered stable if:

$$|\mathcal{L}_t(\mathbf{h}) - \mathcal{L}_{t-1}(\mathbf{h})| < \delta \quad \text{for } k \text{ consecutive evaluations} \tag{15}$$

Such stability criteria ensure termination of noisy or divergent trials and promote reliable convergence toward optimal hyperparameter configurations.

## VII. Experimental Setup

This section outlines the experimental pipeline used to evaluate the performance of various hyperparameter configurations for fine-tuning large language models. We describe the dataset, training setup, and evaluation metrics that form the foundation of our optimization framework.

### *A. Dataset Description*

We utilize the CNN/DailyMail dataset, a benchmark corpus for text summarization tasks. It consists of news articles and corresponding human-written highlights. The dataset is preprocessed using the Hugging Face ‘datasets‘ library, and tokenization is handled with the T5 tokenizer.

- **Training Samples:** 287,227
- **Validation Samples:** 13,368
- **Task:** Abstractive text summarization
- **Input Format:** ”summarize: ¡article¿”
- **Target Format:** ”¡summary¿”

To ensure consistency, inputs are truncated or padded to a maximum length of 512 tokens, while targets are capped at 128 tokens.

### *B. Training Configuration*

The model used is ‘t5-small‘, fine-tuned using the PEFT (Parameter-Efficient Fine-Tuning) approach with LoRA adapters. The training is conducted using PyTorch with CUDA acceleration on Google Colab and local GPU servers.

Key configuration parameters:

- Optimizer: AdamW
- Loss Function: CrossEntropyLoss
- LoRA Rank: [4, 8, 16] (search space)
- LoRA Alpha: [16, 32, 64] (search space)
- Epochs: 2–4

- Batch Size: 8–32 (depending on GPU memory)
- Learning Rate: $10^{-5}$ to $10^{-3}$ (log-uniform sampling)
- Scheduler: Linear with warmup
- Mixed Precision: Enabled via PyTorch AMP

Hyperparameter tuning is done using Optuna, with trial pruning enabled based on evaluation loss at intermediate steps.

TABLE I
SEARCH SPACE FOR HYPERPARAMETER OPTIMIZATION

| Hyperparameter | Range |
|---|---|
| Learning Rate | [1e−5, 1e−3] (log-uniform) |
| Batch Size | {8, 16, 32} |
| Epochs | {2, 3, 4} |
| LoRA Rank | {4, 8, 16} |
| LoRA Alpha | {16, 32, 64} |
| Dropout Rate | [0.0, 0.3] |

## C. Evaluation Metrics

To measure both the predictive performance and computational efficiency of the model under each configuration, we use the following evaluation metrics:

- **Validation Loss:** The average loss on the held-out validation set.
- **ROUGE Scores (ROUGE-1, ROUGE-2, ROUGE-L):** To evaluate overlap between predicted and reference summaries.
- **BLEU Score:** To assess n-gram precision in generated summaries.
- **Accuracy:** Calculated as an exact string match between predicted and true summaries (used for sanity checks).
- **GPU Memory Usage:** Measured using PyTorch hooks and ‘nvidia-smi‘.
- **Training Time:** Measured in seconds per epoch and per trial.

These metrics collectively inform the optimization process, striking a balance between output quality and resource usage.

## VIII. RESULTS

This section presents the outcomes of our hyperparameter optimization experiments, including the best configuration discovered, performance metrics across multiple evaluation criteria, computational efficiency, and comparative analysis with baseline settings.

### A. Best Configuration Found

After conducting 50 Optuna trials using the Tree-structured Parzen Estimator (TPE) sampler, the best-performing hyperparameter configuration identified is as follows:

- **Learning Rate:** $3.21 \times 10^{-4}$
- **Batch Size:** 16
- **Epochs:** 3
- **LoRA Rank:** 8
- **LoRA Alpha:** 32
- **Dropout Rate:** 0.1

This configuration achieved a stable convergence with improved ROUGE scores and reduced training time compared to the baseline.

### B. ROUGE, BLEU, Accuracy

The selected configuration was evaluated using multiple standard metrics:

- **ROUGE-1:** 41.7
- **ROUGE-2:** 18.9
- **ROUGE-L:** 39.6
- **BLEU:** 26.3
- **Accuracy (Exact Match):** 14.5%

These results indicate strong summarization quality, with significant improvements in both unigram and bigram recall compared to non-optimized models.

### C. Training Time and Memory Usage

We monitored system performance during training to evaluate computational efficiency:

- **Total Training Time:** 3.5 minutes per epoch
- **Total Trials:** 50
- **Average Time per Trial:** 8.1 minutes
- **Peak GPU Memory Usage:** 3.4 GB

The use of LoRA significantly reduced memory consumption, enabling higher batch sizes without out-of-memory errors, and improving throughput.

### D. Comparative Analysis

We compared the optimized model to a default T5-small model fine-tuned without any hyperparameter tuning:

TABLE II
COMPARISON BETWEEN DEFAULT AND OPTIMIZED CONFIGURATION

| Metric | Default Config | Optimized Config |
|---|---|---|
| ROUGE-1 | 36.2 | **41.7** |
| ROUGE-2 | 15.4 | **18.9** |
| ROUGE-L | 34.8 | **39.6** |
| BLEU | 21.1 | **26.3** |
| Training Time (per epoch) | 5.2 min | **3.5 min** |
| GPU Memory | 5.1 GB | **3.4 GB** |

The optimized configuration not only improved performance metrics but also demonstrated significant gains in efficiency. These results validate the effectiveness of our automated tuning strategy.

## IX. CHALLENGES AND OBSERVATIONS

During the course of our experiments, we encountered several technical, computational, and conceptual challenges. This section details the primary hurdles faced, the approaches used to overcome them, and the key takeaways from the entire process.

### *A. Technical Challenges Faced*

- **GPU Memory Constraints:** The T5 model, even in its small configuration, consumed significant memory during training, limiting batch size and number of concurrent trials.
- **Inconsistent Evaluation due to Randomness:** Results fluctuated due to randomness in data shuffling, weight initialization, and dropout, which made it hard to compare trials directly.
- **Long Trial Durations:** Certain trials consumed extensive time before being pruned or completed, especially when unpromising configurations weren't caught early.
- **Mixed Precision Instability:** Enabling automatic mixed precision (AMP) sometimes caused gradient underflows or NaN losses in early trials.
- **Checkpoint Management:** Managing and loading multiple checkpoints for different trials caused confusion and storage overhead.

### *B. How They Were Addressed*

- **LoRA Integration:** We adopted Low-Rank Adaptation (LoRA), which significantly reduced memory usage, enabling larger batch sizes and more parallel experiments.
- **Averaging Across Seeds:** To handle variance due to randomness, each configuration was evaluated across multiple seeds and averaged to improve robustness.
- **Pruner Configuration in Optuna:** We adjusted pruning intervals and thresholds to aggressively terminate non-promising trials, saving time.
- **Manual AMP Tuning:** AMP was disabled during initial tuning, and only re-enabled once the model and optimizer behaved stably.
- **Automated Checkpointing:** A consistent naming convention and directory structure was implemented to manage trial outputs and checkpoints systematically.

### *C. Lessons Learned*

- **Efficiency over Exhaustiveness:** A well-pruned, focused search with surrogate modeling outperforms brute-force grid search in both quality and time.
- **Reproducibility is Crucial:** Without seed control and systematic logging, understanding model behavior becomes infeasible—reproducibility must be enforced.
- **Start Simple, Scale Later:** Starting with a smaller model and smaller dataset partition can help verify pipeline correctness before scaling up.
- **Monitoring Matters:** Real-time feedback on memory usage, loss curves, and time per epoch proved invaluable in understanding and debugging performance bottlenecks.
- **Hyperparameter Tuning is Costly but Worth It:** Carefully planned optimization efforts can yield models that are both better performing and more efficient in deployment.

These insights will shape how we design and optimize future ML pipelines, particularly in environments constrained by resources or requiring rapid iteration.

## X. Contributions of Team Members

The project was a collaborative effort involving multiple aspects of machine learning, optimization, and deployment. Each member contributed significantly, as outlined below:

- **NandaKiran Velaga:** Led the model training and hyperparameter optimization workflow using Optuna. Integrated LoRA-based fine-tuning, managed GPU resource monitoring, and authored core evaluation scripts. Also compiled the mathematical formulation and final LaTeX document.
- **Anirudh Krishna:** Focused on dataset preprocessing, data loader implementation, and initial model baseline training. Helped benchmark performance for early iterations and contributed to result visualization.
- **Sri Akash Kadali:** Designed evaluation metrics pipeline and conducted ROUGE and BLEU analysis. Helped debug issues in scoring functions and built comparative performance tables.
- **Phanindra Tupakula:** Handled Optuna integration, defined parameter search spaces, and tuned pruning strategies. Developed scripts for logging intermediate metrics and memory statistics.
- **Venkata Revanth Vardineni:** Managed documentation, contributed to technical writing, and assisted in converting outputs to LaTeX. Coordinated check-ins and consolidated progress for final presentation.

Each team member participated in weekly sync-ups and code reviews, ensuring consistent progress and a unified codebase.

## XI. Conclusion and Future Work

This project demonstrated the effectiveness of automated hyperparameter optimization for large language models using a combination of Bayesian optimization and parameter-efficient fine-tuning with LoRA. We successfully minimized both validation loss and training resource usage by optimizing across learning rate, LoRA configuration, and batch size parameters.

The use of Optuna and early stopping mechanisms enabled efficient exploration of the hyperparameter space under computational constraints. The final model showed substantial gains in ROUGE, BLEU, and accuracy scores, with reduced training time and memory requirements.

**Future Work:**

- Extend the framework to support larger models such as T5-base and T5-large with distributed training.
- Integrate multi-objective optimization directly to identify Pareto-optimal hyperparameter configurations.
- Incorporate additional optimization algorithms such as CMA-ES or Hyperband with weighted ensemble strategies.
- Deploy the fine-tuned model as an API with inference monitoring and continuous evaluation.

This study provides a scalable and reproducible blueprint for hyperparameter tuning of transformer models in real-world ML pipelines.

## Acknowledgment

We express our sincere gratitude to Professor Richard J. La for his valuable guidance throughout the project. His suggestions regarding stochasticity in training led us to explore advanced techniques like surrogate modeling and population-based training.

We also thank the Department of Electrical and Computer Engineering at the University of Maryland, College Park for providing computational resources and academic support that enabled the completion of this project.

## References

[1] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, et al., "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-Generation Hyperparameter Optimization Framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, Anchorage, AK, USA, 2019, pp. 2623–2631.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, et al., "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv preprint*, arXiv:2106.09685, 2021.

[4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Adv. Neural Inf. Process. Syst.*, vol. 24, pp. 2546–2554, 2011.

[5] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 1–52, 2017.

[6] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, et al., "Transformers: State-of-the-Art Natural Language Processing," in *Proc. 2020 Conf. Empir. Methods Nat. Lang. Process. (EMNLP): Syst. Demonstrations*, 2020, pp. 38–45.

[7] L. N. Smith, "A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 - Learning Rate, Batch Size, Momentum, and Weight Decay," *arXiv preprint*, arXiv:1803.09820, 2018.