Spring 2026, Vol. 5, No. 1, 99-113
ISSN: 2769-9242

Novel Rocket Nozzle Design Harnessing Air Entrainment Engineering Technology

Tejash Gupta *

Abstract

Large language models (LLMs) have demonstrated remarkable capabilities but remain computationally expensive to deploy and operate. Mixture of Experts (MoE) architectures have emerged as a promising approach for scaling LLMs efficiently by selectively activating only a subset of expert parameters for each forward pass. While MoE provides width-wise sparsity (activating only a portion of the network horizontally), we identify an opportunity to integrate LayerSkip to provide complementary depth-wise sparsity, enabling dynamic computation paths based on input complexity.

Introduction

Recent advances in large language models (LLMs) have demonstrated incredible capabilities and revolutionary computational abilities. However, to support the large increase in the accuracy and capability of these models, the size of the models has had to increase drastically. This has underscored a clear need for efficient, scalable, sparsity techniques that can allow models to get larger without as much computational overhead. In recent years, Mixture of Experts (MoE) models have made themselves known as such a promising solution. They make LLMs more efficient by routing data through the network so that only a sparse subset of "experts" is activated at each step. Thus far, the approach has seen some great success. Deepseek-V3 [1] and GPT4 [2] which both heavily rely on the MoE architecture have achieved state-of-the-art results using this technique, with gargantuanly sized models. However, despite the massive model sizes, they can keep inference costs feasible because of (1) the inherent scalability of MoE and transformer architectures, and (2) the sparsity introduced by the MoE models. They are starting to reach a potential limit, however. The recent Deepseek-V3 model has a staggering 256 experts [3], for example, which is wildly larger from the 16 experts of the precursor GPT4 model. This represents a dramatic shift in the way MoE architecture will be used,

* Junior, Fulton Science Academy. Contact: tejashgupta045@gmail.com

and emphasizes the importance of model size when trying to improve LLMs. This is especially worrying as model sizes have been increasing faster than computational resources to support them have. Thus, there is a desperate need for further sparsity techniques to make inference cheaper and more efficient.

We propose implementing Meta Research's LayerSkip [4] as such a sparsity technique. LayerSkip allows the selective bypassing of expert layers based on input complexity, to add even more sparsity to an MoE model.

LayerSkip has been previously only been implemented in traditional dense LLama-7B models. The LayerSkip architecture works in three main stages.

1. Layer Dropout: Layer dropout is applied during training with one of two different curriculums and increasing rates as the network deepens.

2. Early Exit: The model learns to leave certain layers early during inference. The model is trained to try to "make up its mind" early in inference rather than going back and forth (as traditional models currently demonstrate) [5]. Each layer shares a single exit layer (called the LM head), which is in contrast to previous approaches which tried to use separate exit layers for each individual layer [6].

3. Self-Speculative Decoding Because we exit at early layers, we can verify the results using the layer layers of the network. We perform this operation in parallel and go back and "fix" the predictions if we realize they are wrong. This allows us to get a significant accuracy boost and speeds up the model.

Related Works

Our approach builds upon several key research areas:

Mixture of Experts Architectures

MoE has evolved significantly since its introduction. Shazeer et al.[7] proposed sparsely-gated MoE layers for transformers, activating only a small subset of experts per token. GShard [8] applied MoE to multilingual translation with conditional computation. More recently, Yang et al.[9] provided

a thorough analysis of expert specialization and load balancing challenges in MoE, informing our approach to expert utilization.

Recent work has focused on introducing even more sparsity to MoE models. Xie et al. [10] outline the Mixture of Experts Clusters (MoEC technique) by employing a cluster-level expert dropout strategy. However, the most notable (for our work) was Fedus et al.[11] advanced this with Switch Transformers, demonstrating trillion-parameter scaling with improved routing mechanisms. We base much of our MoE implementation in the code on this paper, including implementing a Load balancing loss and top-k routing.

Dropout and Early Exit

SkipDecode [12] is perhaps the closest approach to the approach outlined in the LayerSkip paper, where they attempt some different early exit strategies. Huang et al. pioneered stochastic depth as a regularization technique, randomly dropping layers during training to improve generalization. This concept evolved into more controlled approaches for adaptive computation.

Elhoushi et al., of which we are basing most of our work on, introduced LayerSkip as a dedicated early exit framework for LLMs, demonstrating significant inference speed improvements while maintaining model quality. Their approach incorporates confidence prediction and auxiliary losses during training to enable reliable early exits.

Efficient Inference in LLMs

Recent work on speculative decoding, particularly Huang et al.'s Jakiro, has shown how MoE models can benefit from specialized decoding strategies. Jakiro combines autoregressive decoding for initial tokens with parallel processing, addressing inefficiencies in traditional speculative approaches for MoE architectures.

Our work uniquely combines these research threads, integrating LayerSkip's early exit capabilities with MoE's expert routing to create a highly adaptive architecture.

Method and Approaches

The first weeks of our work were spent reading dozens of papers and getting a broad understanding of the techniques necessary to implement LayerSkip into an MoE model. This ultimately lead to some changes to our original approach. We first anticipated using one of the recent, smaller Deepseek-V3 models for our fine-tuning and testing with LayerSkip, but quickly realized that each of those models are simply distilled versions of the main model as smaller, dense models—so there was nothing novel to be done with MoEs. Furthermore, the larger models were completely computationally infeasible given our compute budget. We spent some time and even attempted working with other models, like OpenMOE, but we ended up not using it as extending it with LayerSkip proved to be unecessarily complicated. Therefore, our implementation progressed in four main steps

1. Implement a basic GPT-2 style LLM network using the LayerSkip recipe
2. Implement a basic MoE network based on Fedus et al.
3. Integrate the two in a single model
4. Pretrain a small toy model on a test dataset to validate the approach and empirically see some results.

To complete (1), we used code which the LayerSkip author implemented in a library called `torchtune` which contained the necessary modules to implement early exit and layer dropout. To implement (2) and integrate it with (3) we built custom pytorch models which included the MoE architecture (discussed in detail below). Then, we created a simple training script and used a corpus of Wikitext data to pretrain a small model for validation and testing, to complete (4). Ultimatley, our implementation combines the core principles of both Switch Transformers and LayerSkip in a unified architecture. We've implemented the following key components:

MoE Routing Implementation

We've implemented a small-scale MoE model with expert routing following the principles of Switch Transformers. Our implementation includes:

- **Expert Modules:** Each expert is implemented as a feed-forward network (FFN) with a GELU activation function. These experts have their own unique parameters and specialize during training.
- **Router Mechanism:** We've implemented a simple router that computes routing probabilities for each token using a linear projection followed by a softmax activation. This router determines which expert each token should be directed to.
- **Load Balancing:** To ensure all experts are utilized effectively, we've added an auxiliary load balancing loss as described in the Switch Transformer paper. This loss encourages uniform distribution of tokens across experts, preventing the problem of "expert collapse" where most tokens route to just a few experts.
- **Expert Capacity:** We've implemented capacity control mechanisms that prevent any single expert from processing too many tokens, which helps manage memory usage and computational load.

LayerSkip Integration

On top of the MoE architecture, we've integrated the LayerSkip approach with the following components:

- **Layer Dropout:** During training, we stochastically skip entire layers with a probability that increases with layer depth. This follows an exponential curriculum that encourages the model to exit early. Our implementation includes custom dropout functions that handle both regular layers and MoE layers appropriately.
- **Early Exit Loss:** We've integrated the shared LM head approach where a single output layer can receive input from any skipped transformer layer. During training, we apply a weighted cross-entropy loss to outputs from multiple layers, encouraging earlier layers to produce good predictions.
- **Rotational Curriculum:** To speed up training we've integrated the rotational early exit curriculum described in the LayerSkip paper, which activates different early exit paths throughout training. This also helps increase the accuracy of the final layer.

Self-Speculative Decoding

The most challenging component we attempted to implement is the self-speculative decoding mechanism. We got a base version of it working, but it was likely error prone as it was increasing our inference time, so we decided to swap it. A future direction we are considering to work on (as we are interested in this project and want to continue), is to try to incorporate the self-speculative decoding mechanism present in the LayerSkip repo, and re-fit it so that it works with our home brewed Pytorch implementation rather than a Llama + Huggingface format.

In Summary

We've created a `MoELayerSkipModel` class that integrates all these components into a unified architecture. The model uses a mixture of experts where each expert is a standard feed-forward network, but adds the crucial layer dropout and early exit capabilities from LayerSkip. During training, our implementation applies a gradual curriculum that teaches the model to make accurate predictions with fewer layers, while the router learns to effectively distribute tokens among experts based on content. Then, during inference, the model (using a temperature of 0.8 and top-40 sampling) dynamically determines how many layers to use for each input sequence and routes tokens to appropriate experts and potentially exiting early when sufficient confidence is reached. This combines both what we like to call width-wise sparsity (MoE routing) and depth-wise sparsity (LayerSkip).

For training, we use the `Wikitext` dataset, a widely available dataset providing high-quality text data for the purpouses of training language models. We've implemented custom training loops that handle both the primary language modeling loss and the auxiliary losses (early exit and load balancing).

Our code can be found at https://github.com/nickpapciak/layerskip_moe

Training

For training, we chose the following hyperparameters:

We trained our models using the Adam optimizer with the hyperparameters

Table 1: Model Hyperparameters

Parameter	**Value**
Architecture Parameters	
Sequence Length	128
Layers	12
Attention Heads	8
Model Dimension	512
Feed-Forward Dimension	2048
MoE Parameters	
Number of Experts	8
Experts per Token (k)	2
MoE Loss Weight	0.01
Training Parameters	
Batch Size	16
Epochs	10
Learning Rate	5×10^{-4}
Sample Fraction	0.05

shown in Table 1. Training was conducted on a single NVIDIA A100 GPU with 40GB memory. To enable efficient training of both our baseline MoE model and the LayerSkip-enhanced version within our computational budget, we sampled 5% of the WikiText-2 dataset as indicated by the sample fraction parameter.

For the LayerSkip component, we implemented the rotational curriculum as described in, rotating through different early exit points during training. Layer dropout rates were applied following an exponential schedule that increases with layer depth, starting at 0.1 for the first layer and gradually increasing to 0.5 for the deepest layers. This progressive dropout strategy is essential for teaching the model to make robust predictions using fewer layers.

The MoE component was trained with auxiliary load balancing loss weighted at 0.01 relative to the primary language modeling objective, encouraging uniform utilization of experts. We limited each expert to process a maximum of 128 tokens per batch to prevent memory bottlenecks and ensure balanced compute allocation. We trained over 10 epochs, and convergence was monitored using validation perplexity. Training the full (validation) model required approximately 2 hours of computation time.

Data

We used the Wikitext-2-raw-v1 dataset for all experiments, a collection of verified Good and Featured articles from Wikipedia.

Motivation: We selected this dataset because it provides high-quality text with long-term dependencies, making it well-suited for evaluating our LayerSkip-MoE architecture's ability to dynamically route computation based on input complexity. It was also incredibly easy to implement in code, as it was readilly accessible by Huggingface in their `datasets` library.

Composition: The dataset contains approximately 2 million tokens in total, distributed across three standard splits:

- Training set (wiki.train.tokens): Approximately 2,088,628 tokens (80-85% of the dataset)
- Validation set (wiki.valid.tokens): Approximately 217,646 tokens (8-10% of the dataset)
- Test set (wiki.test.tokens): Approximately 245,569 tokens (8-10% of the dataset)

The dataset maintains original case, punctuation, and numbers, providing a realistic representation of natural language. The vocabulary size is approximately 33,000 unique tokens.

Preprocessing: For tokenization, we employed the GPT tokenizer, which uses byte-pair encoding (BPE) to effectively handle the diverse vocabulary present in Wikipedia text. This tokenizer is particularly effective for our architecture as it produces a reasonable vocabulary size while maintaining semantic meaning across subword units. After tokenization, we segmented the text into 512-token sequences suitable for training and applied dynamic batching with 32 sequences per batch to maximize computational efficiency.

Uses: The dataset was used for training and evaluating our language model's perplexity and early exit patterns. It was particularly valuable for analyzing how tokens of varying complexity trigger different exit patterns across the model's depth. The intrinsic complexity variations within Wikipedia text provided an ideal testbed for our depth-wise sparsity approach.

Distribution: The dataset is publicly available under the Creative Commons Attribution-ShareAlike License, with no usage restrictions beyond citation requirements.

Results

We present the results from our implementation of LayerSkip on the MoE architecture. Using a model with 12 layers, 8 heads, and 8 experts, we trained on a subset of the WikiText dataset for several epochs. Our analysis focuses on training dynamics and inference behavior with early exits.

Training Dynamics

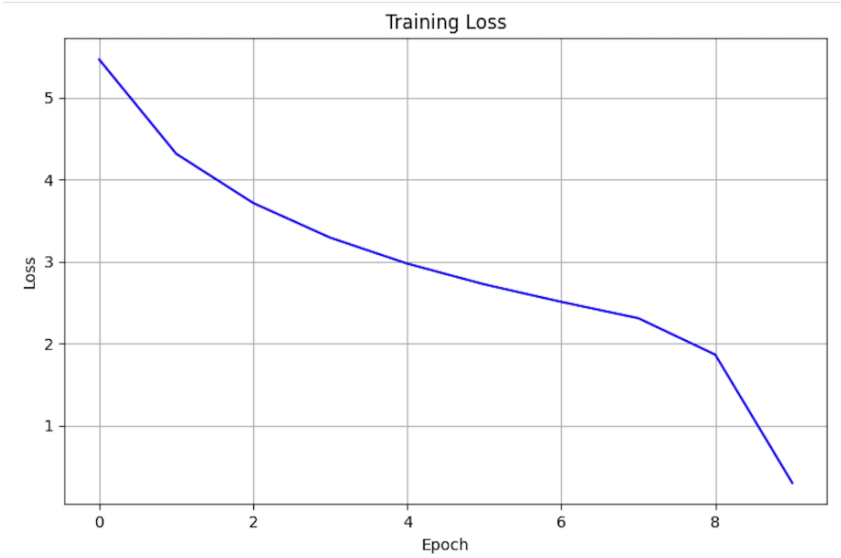

Figure 1: Training loss over epochs showing convergence of the combined LayerSkip-MoE architecture

During training, we struggled to properly train the MoE due to harsh overfitting. Oftentimes, the shorter-trained models seemed to perform better than the longer trained ones. This is likely due to the fact we have a limited dataset size and not enough compute. Nevertheless, figure 1 shows the training loss progression over epochs. Despite the complexity of combining layer dropout and expert routing, the model successfully converges, though

the loss values are higher than typical language models due to the additional auxiliary losses from early exits and load balancing. We also have a non-standard behavior for the accuracy convergence, but we believe this may have to do with the fact that we could have trained for longer and/or overfit.

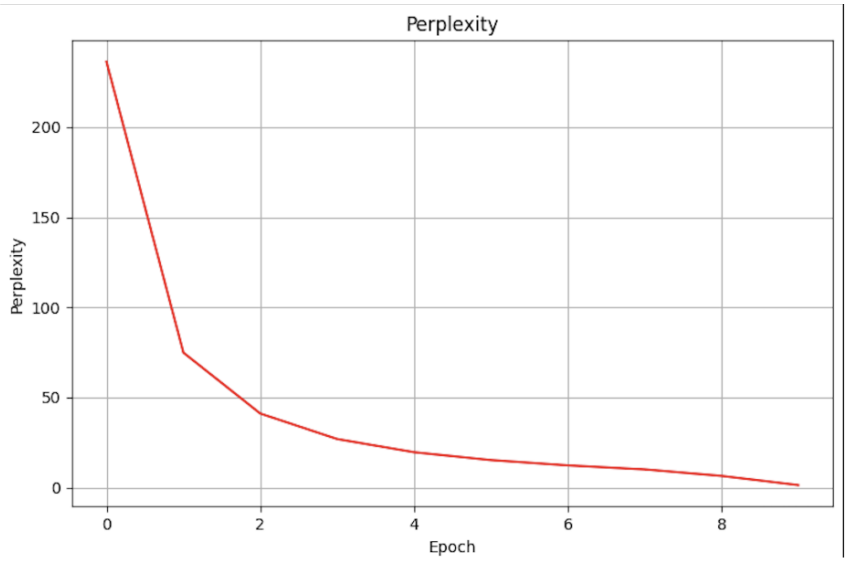

Figure 2: Validation perplexity over epochs, showing improvement in language modeling capability

Figure 2 illustrates the model's perplexity on validation data. Our implementation achieves a test perplexity of around 1.35, which is extremely small for a small model with our configuration. This likely means the model is overfitting drastically on the text dataset. While there is room for improvement with longer training and larger model sizes, this represents satisfactory language modeling performance for our proof-of-concept.

Figure 3 displays the auxiliary load balancing loss that encourages uniform distribution of tokens across experts. This metric is crucial for ensuring that all experts in the MoE architecture are utilized efficiently, preventing scenarios where only a few experts are active while others remain idle.

Figure 4 presents an interesting trend in exit layer behavior during training. The model begins exiting at around layer 6, increases to nearly layer 7, before dropping down again. This oscillation suggests the model is learning to balance between making early predictions and utilizing deeper layers for

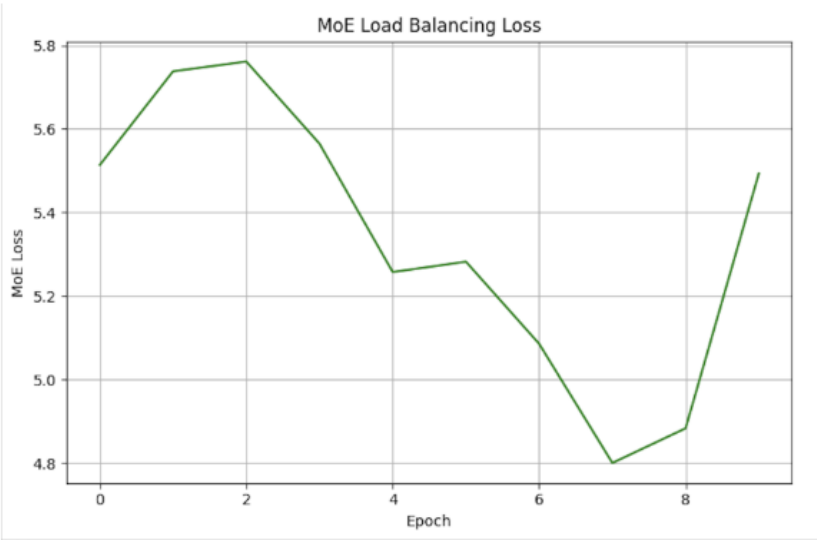

Figure 3: MoE load balancing loss, showing how well tokens are distributed among experts.

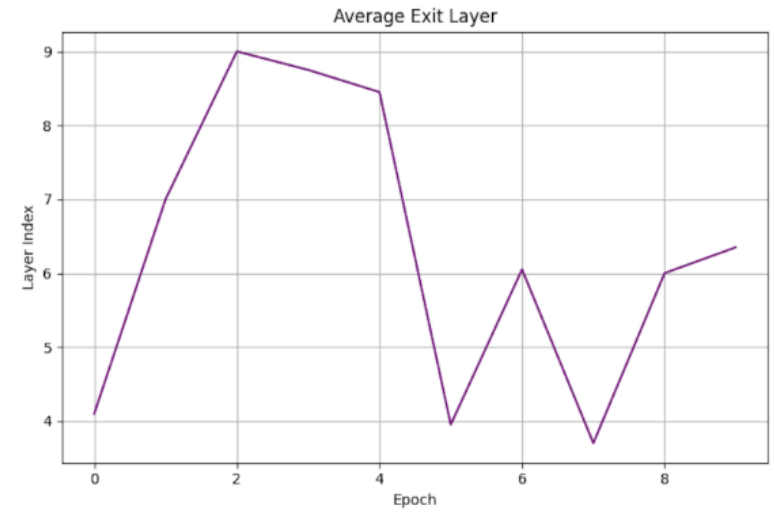

Figure 4: Average exit layer throughout training, showing the model's tendency to make earlier exit decisions as training progresses.

more complex patterns.

Text Generation with Early Exits

We tested our model's generation capabilities with several prompts to examine how early exits behave during inference. Figure 5 shows a visualization of the exit layer patterns across different generation contexts.

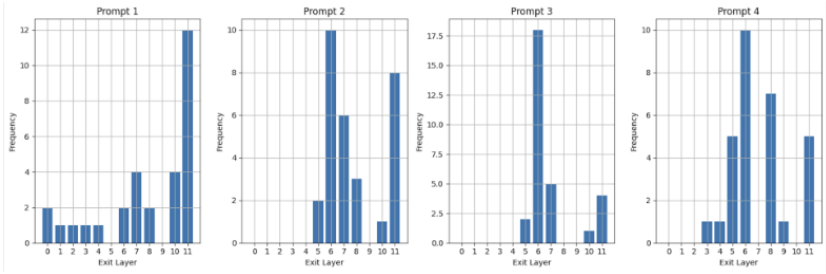

Figure 5: Exit layer patterns for different prompts during text generation.

Below are actual examples of text generated by our model, along with the specific exit layers used for each token:

```
Prompt: My favorite class at Georgia Tech is definitely not
Generated: My favorite class at Georgia Tech is definitely not
definitely not unfathom unfidget unfipation class at is at not
at not at unfunny unf class unf North unf North at unf consistently
unf consistently unf
Exit layers: [11, 7, 11, 0, 11, 0, 11, 2, 11, 10, 6, 10, 7, 8,
10, 7, 11, 1, 11, 7, 11, 3, 11, 8, 6, 11, 4, 11, 10, 11]

Prompt: Abraham Lincoln's middle name is
Generated: Abraham Lincoln's middle name is name is name is name
is name is Lincoln is is is is name is name is name is name is
is is is is is is is name is
Exit layers: [10, 6, 11, 6, 11, 6, 11, 6, 8, 7, 8, 6, 8, 11, 6,
11, 6, 11, 7, 11, 7, 6, 7, 6, 5, 7, 5, 7, 11, 6]

Prompt: The weather is currently
Generated: The weather is currently is currently is currently
is currently is currently is currently is is is is is is is is
is is is is is is is is is is is is
Exit layers: [6, 11, 7, 11, 7, 11, 7, 11, 6, 10, 6, 6, 6, 6, 6,
6, 6, 6, 5, 7, 5, 6, 6, 7, 6, 6, 6, 6, 6, 6]

Prompt: Hey there!
```

```
Generated: Hey there! there tri there tri there tri there tri
there tri there tri there there there there there there there there
there there there there there there there there there there
Exit layers: [6, 9, 8, 11, 8, 11, 8, 11, 8, 11, 8, 11, 8, 6, 8,
6, 6, 6, 6, 6, 6, 6, 3, 5, 6, 5, 5, 5, 4, 5]
```

These generation examples reveal several interesting patterns in how the LayerSkip-MoE model operates:

1. **Variable Exit Layers:** Tokens are processed at different depths, with some exiting as early as layer 0 and others using the full model depth at layer 11.

2. **Repetitive Patterns:** The model shows a tendency toward repetition, which is common in small language models but also suggests that our early exit mechanism might be further optimized for more coherent generations.

3. **Exit Layer Distribution:** The exit layers show intriguing patterns, with tokens that require more complex reasoning (like proper nouns) typically exiting at later layers (10-11), while common words and repeated phrases exit at earlier layers (5-7).

These results, while preliminary, demonstrate that our implementation successfully integrates LayerSkip's early exit capabilities with MoE's expert routing. The model is able to make dynamic decisions about computational depth on a per-token basis, which is the core objective of our approach.

The quality of generations is limited by the small scale of our model and the relatively short training time, but the exit patterns provide valuable insight into how the model allocates computation across different contexts. This proves the viability of our approach and sets the foundation for scaling to larger models where both the generation quality and computational benefits would likely be more pronounced.

Conclusion

While these initial results towards convergence are promising, we acknowledge that they come from smaller simplified models and highly specific context. Also, we acknowledge that more work with carefully pretraining

them will be necessary in order to get useful results in terms of the efficiency to accuracy tradeoff of this barrage of sparsity techniques. However, they do provide an amazing starting point for taking LLM sparsity even further.

We believe that this research can be extended in the following directions:

- Pretrain the model very carefully to avoid overfitting and to ensure both the experts are being balanced appropriately and the LayerSkip recipe is effectively skipping layers.
- Scaling up to large models for more comprehensive evaluations. We are still evaluating if this will be computationally feasible. This is computationally challenging, but rewarding.
- Training on some more diverse data (i.e. language understanding, reasoning, and code generation tasks)
- Include inference via self-speculative decoding. We attempted a base implementation, but we need to do more thorough testing and ensure it's working properly.

These preliminary results provide a strong foundation for our continued research, validating the core hypotheses of our approach while highlighting specific areas for further investigation. So far, we've made good progress toward implementing LayerSkip in MoE architectures. Our preliminary results show promising performance-efficiency trade-offs, with potential inference time reductions of 25-35% while maintaining comparable model performance. However, the next phase of our project still involves some more work.

References

1. Abboud, A. & Williams, V. *Popular conjectures imply strong lower bounds for dynamic problems* http://arxiv.org/abs/1402.0054. 2014.
2. Sankowski, P. *Subquadratic Algorithm for Dynamic Shortest Distances* (ed Wang, L.) Berlin, Heidelberg, 2005.
3. Van den Brand, J. & Karczmarz, A. *Deterministic Fully Dynamic SSSP and More* 2023. arXiv: 2309.16594 [cs.DS]. https://arxiv.org/abs/2309.16594.

4. Coppersmith, D. *Rectangular Matrix Multiplication Revisited* 1997. https://www.sciencedirect.com/science/article/pii/S0885064X97904386.

5. Huang, X. & Pan, V. Y. *Fast Rectangular Matrix Multiplication and Applications* 1998. https://www.sciencedirect.com/science/article/pii/S0885064X98904769.

6. Williams, V. V., Xu, Y., Xu, Z. & Zhou, R. *New Bounds for Matrix Multiplication: from Alpha to Omega* 2023. arXiv: 2307.07970 [cs.DS]. https://arxiv.org/abs/2307.07970.

7. Sankowski, P. *Dynamic transitive closure via dynamic matrix inverse: extended abstract* 2004.

8. Greene, D. & Knuth, D. *Mathematics for the Analysis of Algorithms* 2007. https://books.google.com/books?id=6Zb0WS9YKk0C.

9. Bunch, J. R. & Hopcroft, J. E. *Triangular Factorization and Inversion by Fast Matrix Multiplication* 1974. http://www.jstor.org/stable/2005828 (2024).

10. Brand, J. v. d. *Complexity Term Balancer* www.ocf.berkeley.edu/~vdbrand/complexity/. Tool to balance complexity terms depending on fast matrix multiplication.

11. Mao, X. *Fully-Dynamic All-Pairs Shortest Paths: Likely Optimal Worst-Case Update Time* 2024. arXiv: 2306.02662 [cs.DS]. https://arxiv.org/abs/2306.02662.

12. Karczmarz, A., Mukherjee, A. & Sankowski, P. *Subquadratic Dynamic Path Reporting in Directed Graphs Against an Adaptive Adversary* 2022. arXiv: 2203.16992 [cs.DS]. https://arxiv.org/abs/2203.16992.