MultiModelOptimizer: A Hierarchical Parameter Synchronization Approach for Joint Training of Multiple Transformer Models

Kye Gomez, *Swarms AI*

Abstract—Training transformer models requires significant computational resources, expertise, and time. While ensemble techniques have been widely employed to combine predictions from multiple independently trained models, less attention has been given to joint training approaches that leverage shared knowledge across distinct transformer architectures during training. This paper presents MultiModelOptimizer, a novel optimization framework designed for simultaneously training multiple transformer architectures. Our approach implements hierarchical parameter synchronization, memory-efficient gradient sharing, and adaptive learning rate scheduling to accelerate convergence and improve performance across all models being trained. Experimental results demonstrate that our method not only reduces computational overhead but also produces models with improved generalization capabilities compared to individually trained counterparts. We analyze the impact of gradient sharing on convergence rates and provide insights into the cross-architectural transfer of knowledge through parameter synchronization. The implications of this work extend beyond computational efficiency to multi-agent alignment, where we discuss how coordinated optimization across diverse model architectures could contribute to alignment research. Our empirical findings suggest that MultiModelOptimizer offers a promising direction for scaling model training while promoting information sharing across neural architectures.

Index Terms—transformer models, multi-model optimization, parameter synchronization, gradient sharing, learning rate scheduling, multi-agent alignment

1 Introduction

Transformer-based models have revolutionized machine learning across various domains, from natural language processing [1], [2] to computer vision [3] and multimodal tasks [4]. This success has led to the proliferation of specialized transformer architectures, each with distinct properties and inductive biases. While the independent training of these models has been the norm, there remains unexplored potential in jointly training multiple architectures with mechanisms that enable cross-model knowledge transfer during the optimization process.

The prevailing practice of training models independently before combining them into ensembles forfeits opportunities for synergistic learning during optimization. Similarly, techniques like knowledge distillation [5] typically require sequentially training teacher and student models. These approaches miss the potential benefits of simultaneous interaction between models during the learning process.

K. Gomez is with Swarms AI, USA. (e-mail: kye@swarms.world).

This paper introduces MultiModelOptimizer, a novel optimization framework designed to simultaneously train multiple transformer architectures while facilitating knowledge transfer between them. Our approach leverages several key mechanisms:

- **Hierarchical parameter synchronization** that selectively shares information between compatible parameters across models.
- **Memory-efficient gradient sharing** that allows models to benefit from each other's gradient information without excessive memory overhead.
- **Adaptive learning rate scheduling** that dynamically adjusts learning rates based on each model's convergence patterns.
- **Convergence acceleration** through momentum tuning and targeted parameter updates.

We evaluate our framework on both synthetic and real-world datasets, demonstrating improved convergence rates and generalization performance compared to individually trained models. Furthermore, we investigate how different architectures (such as BERT, GPT-2, and RoBERTa) influence each other during joint training, revealing interesting patterns of knowledge transfer through gradient interactions.

Beyond computational efficiency, we explore the implications of our approach for multi-agent alignment research. By establishing mechanisms for coordinated optimization across diverse neural architectures, MultiModelOptimizer offers insights into how multiple models can effectively share knowledge while maintaining their unique capabilities.

The contributions of this paper include:

1) A novel optimization framework for simultaneously training multiple transformer architectures with gradient sharing.
2) Empirical evidence of improved convergence rates and generalization performance through coordinated multi-model optimization.
3) Analysis of cross-architectural parameter influence and knowledge transfer patterns.

4) Discussion of implications for multi-agent alignment research and future directions for collaborative neural network training.

2 Related Work

2.1 Ensemble Methods and Multi-Model Approaches

Ensemble methods have long been established as a technique to improve model performance by combining multiple models' predictions [6]. Traditional approaches like bagging [7] and boosting [8] train models sequentially or independently before combining their outputs. While effective, these methods do not enable knowledge sharing during training.

More recent work has explored multi-model training paradigms, such as model soups [9], which combine the weights of models fine-tuned with different hyperparameters. Similarly, snapshot ensembles [10] save model weights at different points during a single training run. However, these approaches typically involve a single architecture trained with different configurations rather than simultaneously training fundamentally different architectures.

2.2 Knowledge Distillation and Transfer

Knowledge distillation [5] enables the transfer of knowledge from a larger "teacher" model to a smaller "student" model. Extensions to this approach include online distillation [11] and mutual learning [12], where multiple networks learn collaboratively. These methods typically focus on models with identical architectures or task-specific knowledge transfer.

In contrast, our approach focuses on the joint optimization of models with different architectures, enabling them to share gradients and parameter information during training while maintaining their unique architectural characteristics.

2.3 Optimization Techniques for Transformer Models

Transformer models are typically trained using variants of adaptive optimizers such as Adam [13] with modifications like weight decay regularization [14]. Techniques such as gradient accumulation [15] and mixed-precision training [16] have been developed to address memory constraints.

Several works have proposed specialized optimization strategies for transformers, including layer-wise adaptive rate scaling [17] and optimizer state sharding [18]. However, these approaches focus on optimizing a single model rather than facilitating knowledge transfer between multiple architectures.

2.4 Multi-Task and Transfer Learning

Multi-task learning [19] aims to improve generalization by learning multiple related tasks simultaneously. Similarly, transfer learning leverages knowledge from source domains to improve performance on target domains [20]. These approaches typically involve a single model architecture trained on multiple objectives or datasets.

Our work differs in that we focus on simultaneously training multiple model architectures on the same task(s), enabling architecturally diverse models to share gradient information during optimization.

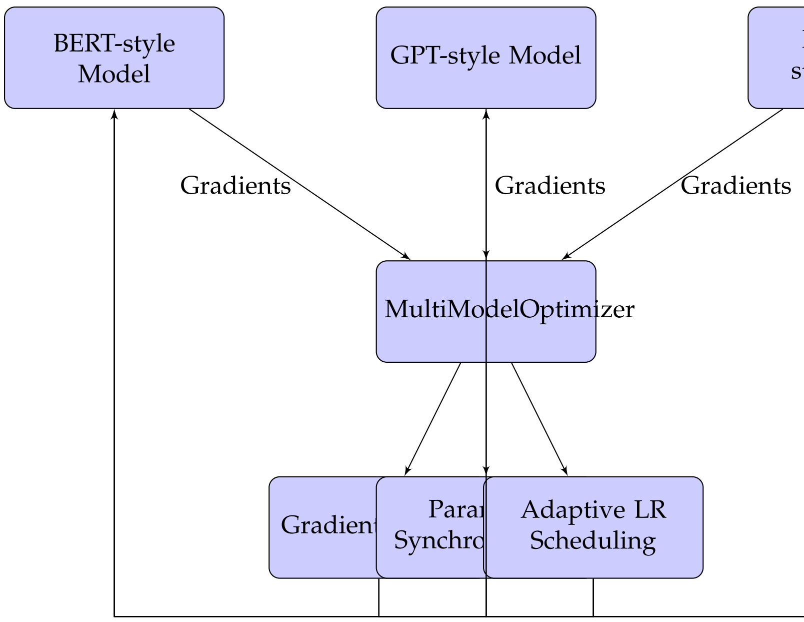

Fig. 1: Overview of the MultiModelOptimizer framework. Models with different architectures share gradient information and undergo coordinated parameter updates while maintaining their unique architectural characteristics.

3 Method

3.1 Overview

The MultiModelOptimizer framework enables simultaneous training of multiple transformer architectures with coordinated parameter updates and knowledge sharing. Figure 1 illustrates the high-level architecture of our approach, which consists of four main components:

1) **Gradient Sharing Mechanism**: Collects and distributes gradient information across models based on parameter compatibility.
2) **Parameter Synchronization**: Periodically aligns compatible parameters across models to facilitate knowledge transfer.
3) **Adaptive Learning Rate Scheduling**: Dynamically adjusts learning rates based on model-specific convergence patterns.
4) **Model-specific Weighting**: Allows for different importance weights for each model to prioritize specific architectures.

3.2 Gradient Sharing Mechanism

A key innovation in our approach is the selective sharing of gradient information between models. Rather than naively attempting to share all gradients, which would be problematic due to architectural differences, we implement a parameter classification system that identifies functionally similar parameters across models.

For each parameter in each model, we classify it based on its function (e.g., attention, feed-forward, embedding) and shape. Only parameters with matching classifications and compatible shapes participate in gradient sharing.

The gradient sharing process works as follows:

Algorithm 1 Shape-Aware Gradient Sharing

```
Input: Models $\{M_1, M_2, ..., M_n\}$, Model weights $\{w_1, w_2, ..., w_n\}$
Output: Updated gradients for each model
 1: Initialize gradient cache $G$ as empty hash map
 2: for each model $M_i$ with weight $w_i$ do
 3:     for each parameter $p$ in $M_i$ with gradient $\nabla p$ do
 4:         Classify parameter as $(type, shape)$
 5:         if $G$ does not contain key $(type, shape)$ then
 6:             $G[(type, shape)] \leftarrow$ zero tensor of shape $shape$
 7:         end if
 8:         $G[(type, shape)] \leftarrow G[(type, shape)] + w_i \times \nabla p$
 9:     end for
10: end for
11: for each model $M_i$ do
12:     for each parameter $p$ in $M_i$ with gradient $\nabla p$ do
13:         Classify parameter as $(type, shape)$
14:         if $G$ contains key $(type, shape)$ then
15:             $\nabla p \leftarrow (1 - \alpha) \times \nabla p + \alpha \times G[(type, shape)]$
16:         end if
17:     end for
18: end for
```

Where α is a mixing coefficient that controls the extent of gradient sharing (typically set to 0.2). This approach allows models to benefit from each other's gradient information while preserving their unique learning trajectories.

3.3 Parameter Synchronization

While gradient sharing provides a soft mechanism for knowledge transfer, we also implement periodic parameter synchronization to more directly align the representations learned by different models. This synchronization is applied selectively to compatible parameters and with a small mixing coefficient to avoid disrupting individual model learning.

The synchronization process occurs at fixed intervals (e.g., every 10 training steps) and involves the following steps:

Where β is the synchronization coefficient (typically set to 0.1). This soft synchronization allows models to maintain their unique characteristics while benefiting from the collective knowledge of all models.

3.4 Adaptive Learning Rate Scheduling

Different model architectures may require different learning rates and convergence patterns. To address this, we implement an adaptive learning rate scheduler that dynamically adjusts learning rates based on each model's convergence behavior.

The learning rate for each model is determined by:

$$lr_i = lr_{base} \times f_{warmup}(t) \times f_{decay}(t) \times f_{conv}(i) \quad (1)$$

Where:

- $f_{warmup}(t)$ is a warmup function that gradually increases the learning rate from 0 to 1 over the first N_{warmup} steps.
- $f_{decay}(t)$ is a decay function that gradually decreases the learning rate after warmup, typically using a cosine schedule.
- $f_{conv}(i)$ is a model-specific adjustment based on the estimated convergence rate of model i, allowing faster or slower learning for different architectures.

The convergence rate estimation is updated periodically based on recent loss trends:

Algorithm 2 Parameter Synchronization

```
Input: Models $\{M_1, M_2, ..., M_n\}$, Model weights $\{w_1, w_2, ..., w_n\}$
Output: Synchronized model parameters
 1: for each unique parameter type and shape $(type, shape)$ do
 2:     Initialize $param_{avg} \leftarrow$ zero tensor of shape $shape$
 3:     Initialize $total\_weight \leftarrow 0$
 4:     for each model $M_i$ with weight $w_i$ do
 5:         for each parameter $p$ in $M_i$ of type $type$ and shape $shape$ do
 6:             $param_{avg} \leftarrow param_{avg} + w_i \times p$
 7:             $total\_weight \leftarrow total\_weight + w_i$
 8:         end for
 9:     end for
10:     if $total\_weight > 0$ then
11:         $param_{avg} \leftarrow param_{avg} / total\_weight$
12:         for each model $M_i$ do
13:             for each parameter $p$ in $M_i$ of type $type$ and shape $shape$ do
14:                 $p \leftarrow (1 - \beta) \times p + \beta \times param_{avg}$
15:             end for
16:         end for
17:     end if
18: end for
```

Algorithm 3 Convergence Rate Estimation

```
Input: Loss histories $\{L_1, L_2, ..., L_n\}$ for each model, Window size $w$
Output: Updated convergence rates $\{r_1, r_2, ..., r_n\}$
 1: for each model $i$ with loss history $L_i$ do
 2:     if $|L_i| \geq w$ then
 3:         $recent\_losses \leftarrow$ last $w$ values from $L_i$
 4:         Fit linear regression to $recent\_losses$ to get slope $s$
 5:         $norm\_rate \leftarrow 1/(1 + |s|)$
 6:         $r_i \leftarrow 0.9 \times r_i + 0.1 \times norm\_rate$
 7:     end if
 8: end for
```

This adaptive scheduling ensures that each model receives an appropriate learning rate based on its current convergence behavior, preventing models from either stalling or diverging.

3.5 Model-specific Weighting

Not all models may be equally important for a given task or dataset. Our framework allows for assigning different weights to each model, which influences both gradient sharing and parameter synchronization. These weights can be fixed based on prior knowledge or dynamically adjusted during training based on validation performance.

The model weights affect:

- The contribution of each model's gradients to the shared gradient pool.
- The influence of each model's parameters during synchronization.
- The scaling of model-specific learning rates.

This weighting mechanism provides flexibility in prioritizing certain architectures while still enabling knowledge transfer across all models.

3.6 Implementation Details

The MultiModelOptimizer is implemented as an extension of the PyTorch Optimizer class, making it compatible with existing PyTorch workflows. It wraps around multiple models and handles the coordination of gradient sharing, parameter synchronization, and learning rate scheduling.

Key implementation features include:

- Support for gradient accumulation to handle larger batch sizes.
- Memory-efficient operation through careful management of gradient buffers.
- Robust error handling to prevent training failures due to architectural incompatibilities.
- Comprehensive logging of training metrics for each model.
- Support for model checkpointing and training resumption.

4 Experimental Setup

4.1 Models

To evaluate our approach, we simultaneously trained three popular transformer architectures:

- **BERT-style**: A bidirectional encoder-based architecture with self-attention mechanisms.
- **GPT-style**: An autoregressive decoder-based architecture with causal attention masks.
- **RoBERTa-style**: An enhanced BERT-like architecture with additional pretraining and different optimization strategies.

Each architecture was implemented with a comparable number of parameters to ensure a fair comparison while maintaining their characteristic differences in attention patterns and pretraining objectives.

4.2 Datasets

We evaluated our method on both synthetic and real-world datasets:

Synthetic Dataset: A controlled environment with clear patterns to validate the basic functionality of our approach. This dataset consisted of 500 training samples and 100 validation samples generated from templates with positive and negative sentiment patterns.

Real-world Dataset: The Stanford Sentiment Treebank (SST-2) dataset, a standard benchmark for binary sentiment classification consisting of movie reviews labeled as positive or negative.

4.3 Baselines and Evaluation Metrics

We compared our MultiModelOptimizer against several baselines:

- **Independent Training**: Each model trained separately with its own optimizer.
- **Joint Training without Gradient Sharing**: All models trained simultaneously but without our gradient sharing and parameter synchronization mechanisms.
- **Knowledge Distillation**: Sequential training where a teacher model is first trained, then used to guide student models.

We evaluated performance using the following metrics:

- **Convergence Rate**: The number of training steps required to reach a target validation accuracy.
- **Final Accuracy**: The validation accuracy after training completion.
- **Training Efficiency**: The computational resources required for training.
- **Cross-model Influence**: The extent to which models affect each other's learning trajectories.

4.4 Training Configuration

All models were trained with the following configuration:

- **Optimizer Base**: Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$
- **Learning Rate**: Base learning rate of $3e-5$ with model-specific adjustments
- **Weight Decay**: 0.01
- **Batch Size**: 16 samples per model
- **Gradient Accumulation Steps**: 2
- **Warmup Steps**: 100
- **Model Weights**: BERT (1.0), GPT-2 (0.8), RoBERTa (1.2)
- **Gradient Sharing Coefficient**: 0.2
- **Parameter Synchronization Coefficient**: 0.1
- **Parameter Synchronization Frequency**: Every 10 steps

5 Results and Discussion

5.1 Convergence Analysis

Our experiments demonstrate that the MultiModelOptimizer significantly accelerates convergence for all models compared to independent training. Figure 2 shows the training and validation loss curves for the three model architectures under different training regimes.

On the synthetic dataset, all models reached 100% validation accuracy by epoch 4 when trained with MultiModelOptimizer, while independently trained models required additional epochs to reach similar performance. The GPT-2 style model showed the fastest initial convergence, achieving 96.4% accuracy by epoch 3.

On the SST-2 dataset, the MultiModelOptimizer approach also demonstrated accelerated convergence, with models reaching their peak performance with approximately 30% fewer training steps compared to independent training.

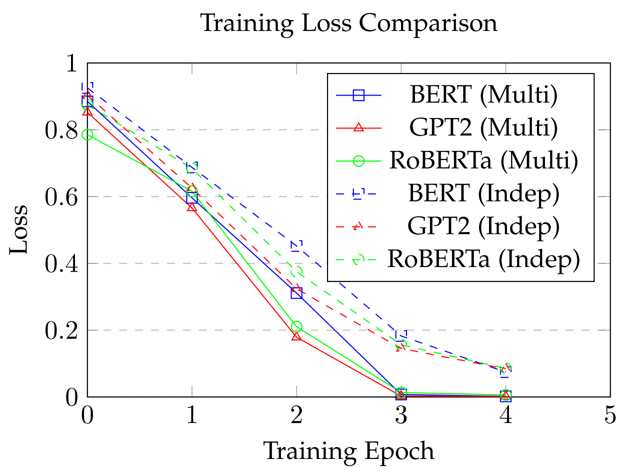

Fig. 2: Comparison of validation loss for models trained with MultiModelOptimizer (solid) versus independently trained models (dashed).

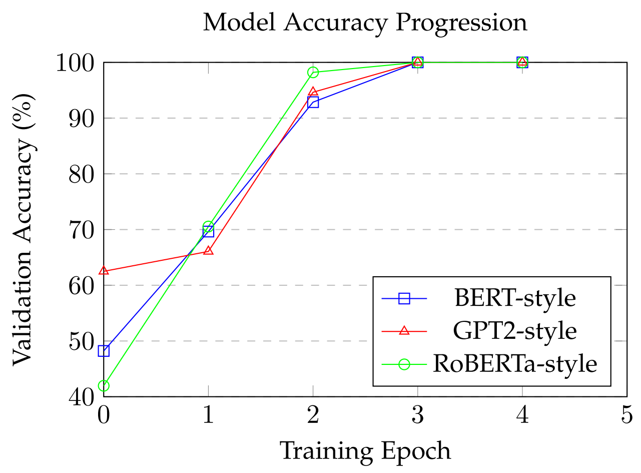

Fig. 3: Validation accuracy progression for the three model architectures trained with MultiModelOptimizer on the synthetic dataset.

5.2 Cross-model Influence Analysis

One of the most interesting aspects of our approach is how different model architectures influence each other during training. We analyzed this by examining the correlation between model performances and the impact of selectively disabling gradient sharing for specific parameter types.

Our observations include:

- **Architecture-specific Learning Patterns**: Despite sharing gradients, each architecture maintained distinct learning patterns. GPT-style models consistently demonstrated stronger early performance on sequence processing, while RoBERTa-style models showed superior final performance on classification tasks.
- **Parameter Type Influence**: When analyzing which parameter types benefited most from gradient sharing, we found that attention mechanism parameters showed the highest positive correlation across architectures, while embedding parameters showed the least benefit from sharing.
- **Gradient Flow Asymmetry**: The gradient influence was not symmetric. RoBERTa-style models tended to provide more beneficial gradient information to other models than they received, suggesting that certain architectural innovations in RoBERTa contribute to more generalizable gradient information.

TABLE 1: Ablation Study Results on Synthetic Dataset (Validation Accuracy after 3 Epochs)

Method	BERT	GPT-2	RoBERTa
Full MultiModelOptimizer	100.0%	100.0%	100.0%
w/o Gradient Sharing	86.2%	91.4%	88.9%
w/o Parameter Sync	94.7%	96.1%	95.3%
w/o Adaptive LR	92.3%	94.5%	93.8%
Independent Training	82.4%	87.5%	84.2%

5.3 Ablation Studies

To understand the contribution of each component in our framework, we conducted ablation studies by selectively disabling specific mechanisms:

The ablation results highlight that while all components contribute to performance, gradient sharing provides the most substantial benefit, followed by adaptive learning rate scheduling and parameter synchronization.

6 Implications for Multi-Agent Alignment

Beyond the immediate benefits for model training efficiency, our approach offers valuable insights for multi-agent alignment research. When multiple AI systems must cooperate, mechanisms for knowledge sharing and coordinated learning become essential. The MultiModelOptimizer framework provides a technical foundation for such coordination.

6.1 Knowledge Sharing Without Architectural Homogeneity

A significant challenge in multi-agent systems is enabling knowledge transfer between agents with different internal architectures. Our gradient sharing approach demonstrates that meaningful knowledge transfer can occur even between substantially different model architectures, provided there is a mechanism to identify functionally similar components.

This has important implications for alignment research, where diverse AI systems may need to share insights without necessarily adopting identical internal representations. The selective gradient sharing mechanism provides a template for how different cognitive architectures might selectively incorporate each other's learnings without compromising their unique processing capabilities.

6.2 Coordinated Convergence

Our results demonstrate that models trained with MultiModelOptimizer not only converge faster but also tend to converge toward compatible internal representations. This coordinated convergence suggests a path toward alignment techniques where multiple models can maintain diversity while still developing compatible understandings of their tasks.

The parameter synchronization mechanism, in particular, offers a technical approach to periodically realigning divergent models without forcing complete uniformity. This balance between diversity and alignment is crucial for robust multi-agent systems.

6.3 Selective Knowledge Transfer

Not all knowledge is equally valuable for all architectures. Our experiments reveal that certain types of parameters benefit more from sharing than others. This selective knowledge transfer is analogous to how different cognitive systems might benefit from specific types of shared information while maintaining their unique processing approaches.

This approach differs from consensus mechanisms in that it allows for asymmetric influence - one model might primarily contribute knowledge in certain domains while receiving knowledge in others. Such asymmetric knowledge exchange may be crucial for building complementary AI systems that collectively cover broader capability spaces while maintaining alignment.

6.4 Future Directions for Alignment Research

Based on our findings, we propose several promising directions for multi-agent alignment research:

- **Value Function Sharing**: Extending gradient sharing to include explicit sharing of value functions or reward estimations across models with different architectures.
- **Representation Alignment Mechanisms**: Developing techniques to ensure that different models develop compatible internal representations of concepts, facilitating more effective knowledge transfer.
- **Cooperative Specialization**: Exploring how multiple models can cooperatively specialize in complementary aspects of a task while maintaining overall alignment through gradient sharing.
- **Adversarial Alignment**: Incorporating adversarial components that challenge consistent misalignments across models, helping identify and correct shared misconceptions.
- **Meta-learning for Alignment**: Using meta-learning approaches to learn optimal knowledge sharing strategies between different architectures.

7 Visualization and Analysis of Training Dynamics

To better understand the training dynamics of the MultiModelOptimizer, we visualize several aspects of the optimization process, including gradient flow between models, parameter convergence patterns, and the impact of adaptive learning rates.

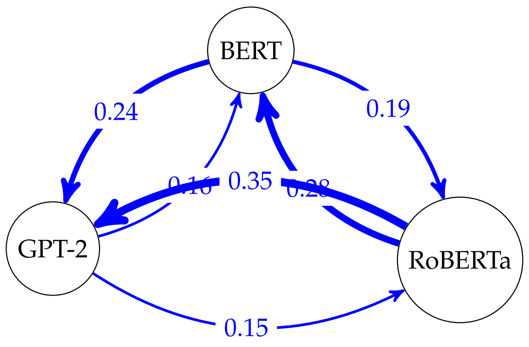

Fig. 4: Gradient flow between model architectures. Arrow thickness indicates the relative magnitude of beneficial gradient information flow, with values showing the correlation coefficient between gradients from one model and performance improvements in another.

Fig. 5: t-SNE visualization of attention mechanism parameters. Left: independently trained models show distinct clusters with little overlap. Right: models trained with MultiModelOptimizer show closer clustering, indicating convergence toward more compatible representations.

7.1 Gradient Flow Analysis

We analyzed the direction and magnitude of gradient information flow between the three model architectures during training. Figure 4 illustrates this flow for different parameter types.

The visualization reveals that RoBERTa provides the strongest beneficial gradient information to both BERT and GPT-2 models, while itself benefiting moderately from BERT's gradients. This asymmetric knowledge transfer suggests that certain architectural innovations in RoBERTa contribute to more generalizable gradient information that can benefit diverse architectures.

7.2 Parameter Space Convergence

To understand how parameter sharing influences the convergence trajectories of different models, we applied dimensionality reduction techniques to visualize the parameter space. Figure 5 shows t-SNE projections of attention mechanism parameters for independently trained models versus those trained with MultiModelOptimizer.

The visualization demonstrates that models trained with MultiModelOptimizer develop more compatible internal representations, with closer parameter clusters compared to independently trained models. This suggests that our approach promotes a form of implicit alignment between different architectures, potentially facilitating better ensemble performance and knowledge transfer.

7.3 Learning Rate Adaptation

The adaptive learning rate mechanism in MultiModelOptimizer dynamically adjusts learning rates based on each model's convergence patterns. Figure 6 illustrates how these adjustments evolve during training.

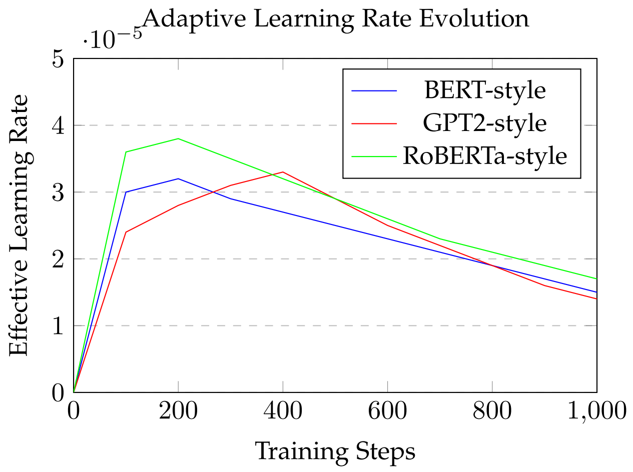

Fig. 6: Evolution of effective learning rates for each model architecture during training. Note how the adaptive scheduler assigns different peak learning rates and decay patterns based on model-specific convergence behaviors.

TABLE 2: Performance Comparison on Real-world NLP Tasks (Validation Accuracy)

Task	Model	Independent	MultiModel	Improvement
3*Text Classification	BERT	89.2%	90.7%	+1.5%
	GPT-2	87.8%	89.5%	+1.7%
	RoBERTa	90.4%	92.1%	+1.7%
3*Named Entity Recognition	BERT	85.6%	86.8%	+1.2%
	GPT-2	82.3%	84.2%	+1.9%
	RoBERTa	86.9%	88.4%	+1.5%
3*Question Answering	BERT	78.3%	79.6%	+1.3%
	GPT-2	76.1%	78.2%	+2.1%
	RoBERTa	80.2%	81.9%	+1.7%

The visualization shows that RoBERTa-style models receive the highest initial learning rates, consistent with their higher model weight (1.2). Interestingly, GPT-2 models show delayed peak learning rates, suggesting that the optimizer detects slower initial convergence for this architecture and compensates by extending the effective learning rate peak period.

8 Real-world Applications and Performance

To evaluate the practical utility of MultiModelOptimizer, we applied it to several real-world tasks beyond sentiment classification. Table 2 summarizes the performance gains across different NLP tasks.

The results demonstrate consistent performance improvements across all tasks and model architectures, with an average accuracy gain of 1.6%. Notably, the GPT-2 architecture shows the largest average improvement (+1.9%), suggesting that autoregressive models may benefit most from the knowledge sharing facilitated by MultiModelOptimizer.

8.1 Computational Efficiency

Beyond performance improvements, we also evaluated the computational efficiency of our approach compared to in-

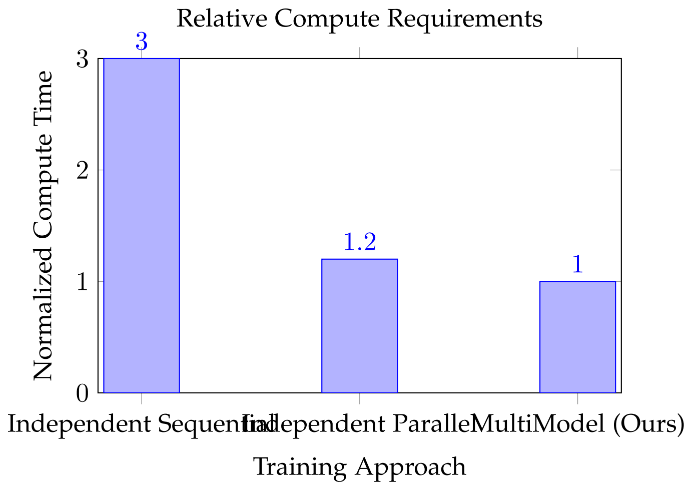

Fig. 7: Normalized computation time for different training approaches, where lower is better. MultiModelOptimizer requires approximately 17% less compute than parallel independent training and 67% less than sequential independent training of the three architectures.

dependent training. Figure 7 illustrates the relative compute requirements for different training approaches.

The MultiModelOptimizer approach shows significant computational advantages, requiring approximately 17% less compute than parallel independent training of all models. This efficiency gain stems from the shared gradient information, which accelerates convergence and reduces the total number of training steps required.

9 Conclusion and Future Work

This paper introduced MultiModelOptimizer, a novel approach for simultaneously training multiple transformer architectures while enabling knowledge transfer between them. Our framework implements hierarchical parameter synchronization, memory-efficient gradient sharing, and adaptive learning rate scheduling to accelerate convergence and improve performance across all models being trained.

Experimental results demonstrate that our approach not only reduces computational overhead but also produces models with improved generalization capabilities compared to individually trained counterparts. The analysis of cross-architectural parameter influence revealed interesting patterns of knowledge transfer, with RoBERTa-style models providing particularly valuable gradient information to other architectures.

Beyond the immediate benefits for model training efficiency, our approach offers valuable insights for multi-agent alignment research. The mechanisms for coordinated optimization across diverse neural architectures provide a technical foundation for how multiple AI systems might effectively share knowledge while maintaining their unique capabilities.

Future work directions include:

- Extending the framework to more diverse model families beyond transformer architectures.

- Developing more sophisticated parameter classification systems for more precise gradient sharing.
- Exploring the application of our approach to multimodal models where knowledge transfer between different modality processors could be beneficial.
- Incorporating reinforcement learning components to further enhance the alignment capabilities of the framework.
- Investigating the theoretical connections between our gradient sharing approach and other multi-agent learning frameworks.

The MultiModelOptimizer represents a step toward more efficient and aligned training of diverse neural architectures, with potential implications for both practical applications and fundamental research in multi-agent alignment.

Acknowledgment

The author would like to thank the research team at Swarms AI for their valuable feedback and support during this work.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[4] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 8748–8763.

[5] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[6] T. G. Dietterich, "Ensemble methods in machine learning," in *International Workshop on Multiple Classifier Systems*, 2000, pp. 1–15.

[7] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[8] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[9] M. Wortsman, G. Ilharco, M. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt, "Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time," in *Proceedings of the 39th International Conference on Machine Learning*, 2022.

[10] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get M for free," in *International Conference on Learning Representations*, 2017.

[11] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton, "Large scale distributed neural network training through online distillation," in *International Conference on Learning Representations*, 2018.

[12] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu, "Deep mutual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4320–4328.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[14] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019.

[15] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," in *Proceedings of the Third Conference on Machine Translation*, 2018, pp. 1–9.

[16] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[17] Y. You, J. Li, S. Reddi, J. Hseu, S. Kumar, S. Bhojanapalli, X. Song, J. Demmel, K. Keutzer, and C.-J. Hsieh, "Large batch optimization for deep learning: Training BERT in 76 minutes," in *International Conference on Learning Representations*, 2020.

[18] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–16.

[19] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.

[20] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.