ModelArena: A Competitive Environment for Multi-Agent Training

The Swarms Corporation
kye@swarms.world
https://swarms.ai

Abstract

We introduce ModelArena (A Competitive Environment for Multi-Agent Training), a novel training methodology that dynamically reallocates computational resources across multiple models during simultaneous training. Unlike conventional approaches that train models in isolation or with static resource allocation, ModelArena creates a competitive learning environment where models that demonstrate faster learning rates are dynamically rewarded with increased memory allocation. This introduces a selection mechanism inspired by evolutionary principles, where computational resources flow toward models exhibiting the most promising learning trajectories. We formulate the mathematical foundation for measuring relative learning rates, implement an adaptive memory reallocation strategy, and demonstrate its effectiveness across heterogeneous model architectures. Our experiments with transformer-based language models show that ModelArena can efficiently identify and prioritize high-potential models, leading to more effective resource utilization and accelerated training for the most promising architectures. Additionally, we discuss the implications of this approach for multi-agent systems and propose extensions for collaborative-competitive training regimes that could further enhance model development. The method introduces a new training paradigm that combines principles from meta-learning, neural architecture search, and evolutionary computation into a unified framework for model training optimization.

1 Introduction

The rapid evolution of deep learning models has led to a proliferation of architectures, each with distinct characteristics and performance profiles across various tasks. This diversification raises significant challenges for computational resource allocation during training, particularly when developing multiple candidate models simultaneously. Traditional approaches typically involve training models in isolation or with fixed resource allocations, potentially leading to inefficient utilization of computational resources and suboptimal training outcomes.

This paper introduces ModelArena (A Competitive Environment for Multi-Agent Training), a novel training methodology that dynamically reallocates computational resources across multiple models during simultaneous training. ModelArena creates a competitive learning environment where models demonstrating superior learning rates receive increased memory allocations, allowing them to train more efficiently with larger batch sizes or more gradient accumulation steps. This approach draws inspiration from evolutionary principles, introducing a selection mechanism where computational re-

sources flow toward models exhibiting the most promising learning trajectories.

The key contributions of this paper are:

- A mathematical framework for quantifying relative learning rates across heterogeneous model architectures, allowing for fair comparison between models with different scales and convergence characteristics.
- An adaptive resource allocation algorithm that dynamically adjusts memory distribution based on measured learning performance, creating a competitive training environment.
- A comprehensive implementation that handles practical considerations including memory management, batch size optimization, and gradient accumulation across diverse model architectures.
- Empirical evaluation demonstrating the effectiveness of ModelArena in identifying and prioritizing high-potential models, leading to more efficient resource utilization.
- Analysis of the emergent dynamics of competitive multi-model training, including observations on different phases of the competitive process and their implications for model development.

Our approach establishes a new paradigm for model training that combines principles from meta-learning, neural architecture search, and evolutionary computation. By introducing competition for computational resources, ModelArena creates a training environment that naturally selects and prioritizes promising architectures, potentially accelerating the development of more effective models.

2 Background and Related Work

2.1 Resource Allocation in Deep Learning

Resource allocation during neural network training has been extensively studied from various perspectives. Traditional approaches focus on static allocation strategies, where computational resources are predetermined before training begins (13; 4). Recent work has explored more dynamic approaches, including gradient-based methods for allocating computational resources (12), but these typically focus on single-model optimization rather than competitive multi-model scenarios.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) represents a related field that aims to automate the design of neural network architectures (15; 7). While NAS methods often train multiple candidate architectures, they typically employ separate training processes rather than creating direct competition for resources. Our approach can be viewed as complementary to NAS, providing an efficient mechanism for evaluating multiple architectures simultaneously.

2.3 Multi-Task and Continual Learning

Multi-task learning (2; 14) and continual learning (10) address the challenge of training models across multiple tasks or over time. These approaches often involve resource allocation decisions, but typically focus on sharing parameters or knowledge between tasks rather than creating a competitive environment for resource allocation.

2.4 Evolutionary and Population-based Training

Population-based training (PBT) (5) and evolutionary algorithms for neural networks (11) share conceptual similarities with our approach in using competition between models. However, these methods typically focus on hyperparameter optimization or architecture search rather than dynamic resource allocation during training. Our approach provides a more direct and continuous form of competition, where resources are reallocated throughout the training process based on learning performance.

2.5 Meta-Learning and Learning to Learn

Meta-learning approaches (3; 9) focus on developing models that can quickly adapt to new tasks. While these methods often involve training multiple models or model configurations, they typically focus on parameter initialization or adaptation strategies rather than resource allocation. Our approach can be viewed as a form of meta-optimization that focuses specifically on allocating computational resources based on learning potential.

3 Methodology

3.1 Problem Formulation

We consider the problem of simultaneously training N distinct model architectures $\mathcal{M} = \{M_1, M_2, \ldots, M_N\}$ on a shared dataset $\mathcal{D}$, with a constrained total memory budget $\mathcal{B}$. Each model M_i has associated parameters θ_i and an initial memory allocation m_i, where $\sum_{i=1}^{N} m_i = \mathcal{B}$.

Our objective is to dynamically adjust memory allocations $\{m_i\}_{i=1}^{N}$ during training to maximize overall training effectiveness, measured by the learning rates of the models. This creates a competitive environment where models that demonstrate superior learning performance are allocated additional resources, potentially at the expense of slower-learning models.

3.2 Learning Rate Quantification

A critical component of our approach is quantifying the learning rate of each model. We define the learning rate $\lambda_i(t)$ for model M_i at time t as the rate of improvement in evaluation performance over a window of recent evaluations:

$$\lambda_i(t) = \frac{1}{w} \sum_{j=1}^{w} \frac{S_i(t-j+1) - S_i(t-j)}{1 + |S_i(t)|} \quad (1)$$

where $S_i(t)$ is the evaluation score of model M_i at time t, and w is the window size for calculating the learning rate. The denominator term $1 + |S_i(t)|$ normalizes the learning rate relative to the model's current performance level, preventing bias toward poorly performing models that have more room for improvement.

For language models, we use the negative perplexity (or equivalently, negative loss) as the evaluation score, ensuring that higher values of $S_i(t)$ correspond to better performance across all model types.

3.3 Memory Allocation Strategy

Given the learning rates $\{\lambda_i(t)\}_{i=1}^{N}$, we redistribute memory allocations according to a weighted allocation strategy. The basic allocation weight $w_i(t)$ for model M_i at time t is proportional to its learning rate:

$$w_i(t) = \frac{\max(\lambda_i(t), \epsilon)}{\sum_{j=1}^{N} \max(\lambda_j(t), \epsilon)} \quad (2)$$

where ϵ is a small positive constant to ensure non-zero allocation for models with zero or negative learning rates.

To prevent extreme oscillations in allocation, we employ a smoothing factor α that considers both the current allocation and the new calculated weights:

$$\tilde{w}_i(t) = \alpha \cdot \frac{m_i(t-1)}{\mathcal{B}} + (1-\alpha) \cdot w_i(t) \quad (3)$$

Additionally, we enforce a minimum allocation threshold δ to ensure all models receive sufficient resources to continue training:

$$\hat{w}_i(t) = \begin{cases} \delta, & \text{if } \tilde{w}_i(t) < \delta \\ \tilde{w}_i(t), & \text{otherwise} \end{cases} \quad (4)$$

The final normalized weights are calculated as:

$$\bar{w}_i(t) = \frac{\hat{w}_i(t)}{\sum_{j=1}^{N} \hat{w}_j(t)} \quad (5)$$

These weights are then used to determine the new memory allocation for each model:

$$m_i(t) = \bar{w}_i(t) \cdot \mathcal{B} \quad (6)$$

3.4 Batch Size Optimization

For each model, the memory allocation $m_i(t)$ must be translated into concrete training parameters. We primarily adjust the batch size $b_i(t)$ based on the memory allocation, using a binary search approach to find the maximum batch size that fits within the allocated memory.

Given a model M_i with memory allocation $m_i(t)$, the optimal batch size $b_i(t)$ is determined by:

$$b_i(t) = \max\{b \in \mathbb{Z}^+ : \mu_i(b) \leq m_i(t)\} \quad (7)$$

where $\mu_i(b)$ is a function that estimates the memory required by model M_i with batch size b. This function accounts for parameters, gradients, optimizer states, and activations during the forward and backward passes.

For models that require larger batch sizes for effective training but are constrained by memory limitations, we employ gradient accumulation with steps g_i, effectively simulating a larger batch size while maintaining memory efficiency.

3.5 Training Process

The complete ModelArena training process alternates between standard training steps and periodic evaluation and reallocation phases. During training, each model M_i performs a number of steps proportional to its memory allocation, ensuring that models with larger allocations make more progress between evaluations.

Algorithm 1 provides the pseudocode for the ModelArena training process.

4 Experimental Setup

4.1 Models and Datasets

To evaluate the effectiveness of ModelArena, we conducted experiments with four transformer-based language models:

- **GPT-2 Small**: 117M parameters, autoregressive transformer model
- **GPT-2 Medium**: 345M parameters, larger version of GPT-2
- **BERT Base**: 110M parameters, bidirectional transformer model
- **RoBERTa Base**: 125M parameters, optimized BERT variant

All models were pre-trained and fine-tuned on the WikiText-2 dataset (8), a collection of verified articles from Wikipedia. We used a

Algorithm 1 ModelArena Training Process

procedure TrainWithModelArena($\mathcal{M}, \mathcal{D}, \mathcal{B}, T, E, R$)
  Initialize models $M_1, M_2, \ldots, M_N$ with parameters $\theta_1, \theta_2, \ldots, \theta_N$
  Initialize memory allocations $m_1, m_2, \ldots, m_N$ s.t. $\sum_{i=1}^{N} m_i = \mathcal{B}$
  Initialize optimizers and schedulers for each model
  Initialize performance trackers $S_1, S_2, \ldots, S_N$ for each model
  $t \leftarrow 0$ ▷ Global step counter
  Evaluate all models to establish baseline performance
  for $i = 1$ to N **do**
    $S_i(0) \leftarrow \text{Evaluate}(M_i, \mathcal{D}_{\text{val}})$
  end for
  while $t < T$ **do**
    for $i = 1$ to N **do**
      Compute batch size b_i based on allocation m_i
      $\text{steps}_i \leftarrow \max(1, \lfloor 10 \cdot m_i / \mathcal{B} \rfloor)$
      for $s = 1$ to steps_i **do**
        Sample batch B_i from $\mathcal{D}_{\text{train}}$ with size b_i
        $\mathcal{L}_i \leftarrow \text{Loss}(M_i(B_i; \theta_i))$
        Compute gradients and update parameters θ_i
      end for
      Record training loss
    end for
    $t \leftarrow t + 1$
    if $t \bmod E = 0$ **then** ▷ Evaluation interval
      for $i = 1$ to N **do**
        $S_i(t) \leftarrow \text{Evaluate}(M_i, \mathcal{D}_{\text{val}})$
        Compute $\lambda_i(t)$ based on recent performance
      end for
    end if
    if $t \bmod R = 0$ **then** ▷ Reallocation interval
      Compute weights $\bar{w}_1, \bar{w}_2, \ldots, \bar{w}_N$ based on λ_i
      for $i = 1$ to N **do**
        $m_i \leftarrow \bar{w}_i \cdot \mathcal{B}$ ▷ Update allocations
      end for
    end if
  end while
  return trained models and performance history
end procedure

sequence length of 128 tokens for all models to ensure consistent evaluation.

4.2 Training Configuration

We initialized the memory allocation equally across all models, with each receiving 25% of the available memory. Models were trained with different learning rate configurations:

- GPT-2 Small: Learning rate of 5e-5, gradient accumulation steps of 2
- GPT-2 Medium: Learning rate of 4e-5, gradient accumulation steps of 4
- BERT Base: Learning rate of 3e-5, gradient accumulation steps of 1
- RoBERTa Base: Learning rate of 3e-5, gradient accumulation steps of 1

We employed the Adam optimizer with weight decay for all models, with a warm-up period of 100 steps followed by a linear learning rate decay schedule.

4.3 Evaluation and Reallocation Parameters

Models were evaluated every 50 steps on the validation set, with performance measured by the negative log-likelihood (for GPT-2 models) or masked language modeling loss (for BERT and RoBERTa). Memory reallocation occurred every 100 steps based on the calculated learning rates.

Key parameters for the adaptive allocation strategy were:

- Learning rate window size (w): 5 evaluation points
- Smoothing factor (α): 0.7
- Minimum allocation threshold (δ): 0.05 (5% of total memory)

For comparison, we also conducted control experiments with static memory allocation, where each model maintained its initial 25% allocation throughout training.

4.4 Hardware Configuration

Experiments were conducted on NVIDIA A100 GPUs with 40GB memory. The implementation utilized PyTorch and the Hugging Face Transformers library for model architectures.

5 Results and Analysis

5.1 Memory Allocation Dynamics

Figure 1 illustrates the memory allocation dynamics across the four models during training. Initially, all models received equal allocations (25%), but these quickly diverged as learning rates were calculated and resources reallocated.

Figure 1: Memory allocation dynamics during training. The graph shows how the proportion of memory allocated to each model evolved over 1000 training steps.

Several key phases are evident in the allocation dynamics:

1. **Initial Exploration (steps 0-200)**: During this phase, all models were actively learning, with relatively small differentiation in allocations. GPT-2 Small and GPT-2 Medium gained slightly increased allocations, while BERT Base began to lose resources.
2. **Differentiation (steps 200-500)**: Clear separation emerged in the allocation patterns. GPT-2 Small established itself as the fastest learner, receiving up to 31% of memory resources. Concurrently, BERT Base's allocation declined significantly, dropping below 15%.
3. **Stabilization (steps 500-1000)**: Allocation patterns stabilized, with GPT-2 Small maintaining the highest share (approximately 31%), followed by RoBERTa Base (28%), GPT-2 Medium (27%), and BERT Base (13%).

Interestingly, the allocation did not converge to a winner-takes-all scenario, instead maintaining a balanced distribution that reflected the ongoing learning potential of each model. This demonstrates ModelArena's ability to maintain diversity in the model ecosystem while still prioritizing resources toward the most promising architectures.

5.2 Learning Rates and Model Performance

Figure 2 shows the calculated learning rates for each model throughout training, alongside the evaluation scores.

Figure 2: Learning rates of models during training. Higher values indicate faster improvement in performance.

Several important patterns emerge from the learning rate analysis:

1. **Early Learning Advantage**: GPT-2 Small demonstrated the highest learning rate throughout most of the training process, explaining its priority in resource allocation. This suggests that smaller models may exhibit faster initial learning, even if they ultimately converge to lower performance levels.
2. **Learning Rate Decline in BERT Models**: Both BERT Base and RoBERTa Base showed declining learning rates over time, suggesting these models were approaching their performance plateaus more quickly than the GPT-2 variants.

3. **Performance-Learning Rate Discrepancy**: Interestingly, BERT Base achieved the best absolute performance (evaluation score of -0.0009, representing very low loss), but exhibited the lowest learning rate (0.00009), resulting in reduced resource allocation. This highlights the distinction between absolute performance and learning potential in our allocation strategy.

Table 1 summarizes the final metrics for each model after 1000 training steps.

Table 1: Final metrics after 1000 training steps

Model	Eval Score	Learning Rate	Mem. Share	Batch Size
GPT-2 Small	-1.0861	0.001915	31.06%	1
GPT-2 Medium	-0.9450	0.001330	27.19%	1
BERT Base	-0.0009	0.000096	13.36%	1
RoBERTa Base	-0.0014	0.000200	28.39%	1

5.3 Training Efficiency Analysis

To evaluate the efficiency of the ModelArena approach, we compared the convergence speed of models under adaptive allocation versus static allocation. Figure 3 illustrates the evaluation loss for each model under both conditions.

Figure 3: Convergence comparison between ModelArena and static allocation. Solid lines represent models trained with ModelArena, while dashed lines represent models trained with static allocation.

The convergence analysis reveals several key insights:

1. **Accelerated Convergence for GPT Models**: Both GPT-2 variants showed significantly faster convergence under ModelArena compared to static allocation. GPT-2 Small reached a loss of 1.5 approximately 200 steps earlier with adaptive allocation.

2. **Minimal Impact on BERT Models**: BERT Base showed similar convergence patterns under both allocation strategies, despite receiving reduced resources under ModelArena. This suggests that the model was able to reach its performance plateau efficiently even with lower memory allocation.

3. **Improved Final Performance**: RoBERTa Base achieved better final performance under ModelArena (0.0014 vs. 0.0016), suggesting that the additional resources allocated to this model allowed it to reach a better optimization point.

5.4 Resource Utilization Efficiency

To quantify the efficiency gains of ModelArena, we calculated the Resource Utilization Efficiency (RUE) metric, defined as:

$$\text{RUE} = \frac{1}{N} \sum_{i=1}^{N} \frac{|\Delta S_i|}{m_i} \tag{8}$$

where ΔS_i is the improvement in score for model M_i and m_i is the memory allocation. This metric measures the average improvement gained per unit of allocated memory.

Table 2 compares the RUE for ModelArena versus static allocation across different training phases.

Table 2: Resource Utilization Efficiency comparison

Training Phase	ModelArena RUE	Static RU
Early (steps 0-300)	0.165	0.127
Middle (steps 300-700)	0.098	0.072
Late (steps 700-1000)	0.042	0.031
Overall	0.102	0.077

ModelArena demonstrated consistently higher resource utilization efficiency across all

training phases, with an overall improvement of approximately 32% compared to static allocation. The efficiency advantage was most pronounced in the early training phase, where ModelArena's ability to quickly identify and prioritize fast-learning models led to more effective resource utilization.

6 Discussion

6.1 Emergent Competitive Dynamics

Our experiments revealed several interesting emergent dynamics in the competitive training environment created by ModelArena:

6.1.1 Learning Rate vs. Performance Trade-off

A notable observation is the divergence between absolute performance and learning rate. BERT Base achieved the best absolute performance (lowest loss) but received the smallest memory allocation due to its lower learning rate. This highlights a fundamental trade-off in resource allocation strategies: optimizing for immediate performance versus optimizing for learning potential.

This trade-off becomes particularly relevant in practical applications where training resources are limited and must be allocated efficiently across multiple model candidates. ModelArena's approach of prioritizing learning potential can be especially valuable in early exploration phases of model development, where the goal is to identify the most promising architectures.

6.1.2 Architecture-Specific Learning Characteristics

The different learning patterns observed across model architectures suggest that certain architectural families may be inherently better suited to the adaptive allocation approach. In our experiments, autoregressive models (GPT-2 variants) exhibited more sustained learning rates compared to masked language models (BERT, RoBERTa), which reached performance plateaus more quickly.

This observation aligns with previous research suggesting that autoregressive language models may have higher capacity for continued improvement with additional training (1; 6). ModelArena naturally identified and exploited this characteristic, allocating more resources to the models with greater potential for continued improvement.

6.1.3 Resource Allocation Equilibrium

Rather than converging to a winner-takes-all scenario, ModelArena established a dynamic equilibrium in resource allocation that balanced immediate learning rates with long-term potential. The minimum allocation threshold (δ) played an important role in maintaining diversity in the model ecosystem, ensuring that even slower-learning models retained sufficient resources to continue making progress.

This balance between exploitation (allocating resources to the current best learners) and exploration (maintaining resources for potentially valuable alternatives) represents a key strength of the ModelArena approach, allowing it to efficiently navigate the complex landscape of model optimization without prematurely converging to suboptimal solutions.

6.2 Implications for Multi-agent Systems

While our experiments focused on training language models, the principles underlying ModelArena have broader implications for multi-agent systems:

6.2.1 Computational Resource Markets

ModelArena effectively creates a "market" for computational resources, where models "compete" based on their demonstrated learning performance. This market-based approach could be extended to more general multi-agent systems, where agents could bid for computational resources based on their expected utility or learning potential.

Such resource markets could potentially lead to more efficient allocation of computational resources in distributed systems, allowing agents to dynamically adjust their resource consumption based on their current tasks and learning needs.

6.2.2 Emergent Specialization

In more complex training scenarios with diverse tasks or objectives, ModelArena-like approaches could potentially lead to emergent specialization among models. Models that demonstrate particular aptitude for specific tasks could receive increased resources for those tasks, leading to a natural division of labor across the model population.

This specialization could be particularly valuable in multi-task learning scenarios, where different architectural characteristics may be better suited to different tasks. By allowing resources to flow toward the most effective architectures for each task, ModelArena could potentially lead to improved overall system performance.

6.3 Limitations

Several limitations of the current implementation of ModelArena should be noted:

- **Memory as Primary Resource**: Our implementation focuses primarily on memory allocation, with other resources (such as computation time) allocated proportionally. In some scenarios, a more nuanced approach to resource allocation might be beneficial.
- **Evaluation Overhead**: The periodic evaluation of all models introduces computational overhead that could potentially outweigh the benefits of adaptive allocation in some scenarios, particularly with a large number of models or very frequent evaluations.
- **Task Specificity**: The learning rate metrics used in our implementation are specific to the language modeling task. Adapting ModelArena to other domains would require appropriate task-specific performance metrics.
- **Cold Start Problem**: In the initial training phase, before sufficient evaluation data is available, allocation decisions must rely on heuristics or preset values, potentially leading to suboptimal initial allocations.

7 Future Work

7.1 Enhanced Allocation Strategies

Several directions for enhancing the allocation strategy could be explored:

7.1.1 Multi-objective Allocation

Extended ModelArena could incorporate multiple objectives in the allocation decision, balancing learning rate with other factors such as absolute performance, training stability, or resource efficiency. This could be formulated as a multi-objective optimization problem:

$$w_i(t) = \sum_{j=1}^{K} \beta_j \cdot \text{Objective}_j(M_i, t) \quad (9)$$

where $\{\beta_j\}_{j=1}^{K}$ are weights for K different objectives. These weights could themselves be learned or adapted based on the stage of training or specific application requirements.

7.1.2 Predictive Allocation

Rather than relying solely on observed learning rates, future implementations could incorporate predictive models that forecast the expected future performance of each model based on its learning trajectory. This could be particularly valuable for models with non-linear learning curves, where past performance may not directly predict future learning potential:

$$\hat{S}_i(t + \Delta t) = f_\phi(S_i(t - w : t), \theta_i, m_i) \quad (10)$$

where f_ϕ is a predictive model with parameters ϕ that forecasts the future score of model M_i based on its recent performance history, parameters, and memory allocation.

7.2 Collaborative-Competitive Training

A promising direction for future research is the integration of collaborative elements into the competitive framework of ModelArena:

7.2.1 Knowledge Distillation Between Models

Models could periodically share knowledge through distillation, with better-performing models acting as teachers for other models:

$$\mathcal{L}_{\text{distill}}(\theta_i) = \alpha \cdot \mathcal{L}_{\text{task}}(\theta_i) + (1 - \alpha) \cdot \mathcal{L}_{\text{KL}}(M_i(x; \theta_i), M_j(x; \theta_j)) \quad (11)$$

where $\mathcal{L}_{\text{KL}}$ is the Kullback-Leibler divergence between the outputs of model M_i and a teacher model M_j. This would allow knowledge to flow between models while still maintaining the competitive resource allocation mechanism.

7.2.2 Parameter Sharing and Mixing

Models could also share or mix parameters during training, creating a form of genetic algorithm where successful architectural components are combined:

$$\theta_i^{\text{new}} = \text{Crossover}(\theta_i, \theta_j, \theta_k, \ldots) \quad (12)$$

where Crossover is a function that combines parameters from multiple models. This could be particularly effective when models share similar architectural components but differ in specific configurations or initialization.

7.3 Dynamic Architecture Adaptation

Perhaps the most ambitious extension of ModelArena would be to allow for dynamic modification of model architectures during training:

7.3.1 Neural Architecture Search Integration

ModelArena could be integrated with Neural Architecture Search (NAS) methods, using learning rate as a signal to guide architectural modifications:

$$\mathcal{A}_i^{\text{new}} = \text{NAS}(\mathcal{A}_i, \lambda_i, \nabla_{\theta_i}\mathcal{L}) \quad (13)$$

where $\mathcal{A}_i$ represents the architecture of model M_i, and NAS is a neural architecture search algorithm that proposes modifications based on learning rate and gradient information.

7.3.2 Growth and Pruning Operations

Models could dynamically grow or shrink based on their performance and resource allocation, with high-performing models allocated additional parameters or layers, while underperforming models undergo pruning:

$$\begin{aligned} \theta_i^{\text{grow}} &= \text{Expand}(\theta_i) \quad \text{if } \lambda_i > \tau_{\text{grow}} \\ \theta_i^{\text{prune}} &= \text{Prune}(\theta_i) \quad \text{if } \lambda_i < \tau_{\text{prune}} \end{aligned} \tag{14}$$

where τ_{grow} and τ_{prune} are thresholds for growth and pruning operations, respectively. This would allow the system to dynamically adapt not only the resource allocation but also the model capacities during training.

7.3.3 Heterogeneous Hardware Targeting

In distributed training environments with heterogeneous hardware, ModelArena could be extended to match models to appropriate hardware based on their architectural characteristics and resource requirements:

$$(m_i, h_i) = \text{Allocate}(M_i, \lambda_i, \mathcal{H}) \tag{15}$$

where h_i is the hardware allocation for model M_i from the available hardware resources $\mathcal{H}$. This would allow for more efficient utilization of heterogeneous computing environments, with models automatically matched to the most appropriate hardware for their specific characteristics.

7.4 Theoretical Analysis

Future work should also focus on developing a more rigorous theoretical understanding of the dynamics of competitive multi-model training:

7.4.1 Convergence Analysis

A formal analysis of the convergence properties of ModelArena under different conditions could provide insights into optimal parameter settings and potential limitations:

$$\lim_{t \to \infty} m_i(t) = f(\lambda_i, S_i, \mathcal{A}_i, \mathcal{D}) \tag{16}$$

where f is a function that relates the asymptotic memory allocation to model characteristics and dataset properties.

7.4.2 Game-Theoretic Formulation

The competitive dynamics of ModelArena could be formalized using game theory, with models treated as agents competing for resources in a non-cooperative game:

$$\text{Utility}_i(m_i, m_{-i}) = \lambda_i(m_i) - \text{Cost}(m_i) \tag{17}$$

where Utility_i is the utility function for model M_i, m_{-i} represents the allocations to all other models, and $\text{Cost}(m_i)$ represents the opportunity cost of allocation m_i. This formulation could provide insights into the equilibrium properties of the resource allocation game and potential strategies for optimizing system performance.

8 Conclusion

This paper introduced ModelArena (A Competitive Environment for Multi-Agent Training), a novel training methodology that dynamically reallocates computational resources across multiple models based on their demonstrated learning rates. Our experiments with transformer-based language models demonstrated that ModelArena can effectively identify and prioritize models with higher learning potential, leading to more efficient resource utilization and accelerated training for promising architectures.

The key innovations of ModelArena include:

- A mathematically rigorous framework for quantifying and comparing learning rates across heterogeneous model architectures
- An adaptive allocation strategy that balances exploitation of high-performing models with exploration of alternative architectures

- A practical implementation that addresses the challenges of memory management and batch size optimization in competitive multi-model training

Our analysis revealed several interesting emergent dynamics in the competitive training environment, including the trade-off between absolute performance and learning potential, architecture-specific learning characteristics, and the establishment of resource allocation equilibria that maintain model diversity while prioritizing promising candidates.

The principles underlying ModelArena have broad implications for multi-agent systems, computational resource markets, and emergent specialization in distributed learning environments. Future directions for extending this work include enhanced allocation strategies, collaborative-competitive training frameworks, dynamic architecture adaptation, and theoretical analysis of the convergence and equilibrium properties of competitive multi-model training.

By introducing competition for computational resources as a fundamental element of the training process, ModelArena represents a step toward more adaptive and efficient training methodologies that can better navigate the increasingly complex landscape of neural network architectures and optimization.

References

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

[2] Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1), 41–75.

[3] Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*, 1126–1135.

[4] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., ... & He, K. (2017). Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

[5] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., ... & Kavukcuoglu, K. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

[6] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[7] Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

[8] Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

[9] Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

[10] Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113, 54–71.

[11] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01), 4780–4789.

[12] Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional

neural networks. *International Conference on Machine Learning*, 6105–6114.

[13] You, Y., Zhang, Z., Hsieh, C. J., Demmel, J., & Keutzer, K. (2018). Imagenet training in minutes. *Proceedings of the 47th International Conference on Parallel Processing*, 1–10.

[14] Zhang, Y., & Yang, Q. (2021). A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*.

[15] Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.