Knowledge Fusion By Evolving Weights of Language Models

Guodong Du[1] **Jing Li**[1*] **Hanting Liu**[2] **Runhua Jiang**[2]
Shuyang Yu[2] **Yifei Guo**[2] **Sim Kuan Goh**[2*] **Ho-Kin Tang**[1*]
[1]Harbin Institute of Technology, Shenzhen, China
[2]Xiamen University Malaysia
duguodong7@gmail.com jingli.phd@hotmail.com
simkuan.goh@xmu.edu.my denghaojian@hit.edu.cn

Abstract

Fine-tuning pre-trained language models, particularly large language models, demands extensive computing resources and can result in varying performance outcomes across different domains and datasets. This paper examines the approach of integrating multiple models from diverse training scenarios into a unified model. This unified model excels across various data domains and exhibits the ability to generalize well on out-of-domain data. We propose a knowledge fusion method named `Evolver`, inspired by evolutionary algorithms, which does not need further training or additional training data. Specifically, our method involves aggregating the weights of different language models into a population and subsequently generating offspring models through mutation and crossover operations. These offspring models are then evaluated against their parents, allowing for the preservation of those models that show enhanced performance on development datasets. Importantly, our model evolving strategy can be seamlessly integrated with existing model merging frameworks, offering a versatile tool for model enhancement. Experimental results on mainstream language models (i.e., encoder-only, decoder-only, encoder-decoder) reveal that `Evolver` outperforms previous state-of-the-art models by large margins. The code is publicly available at https://github.com/duguodong7/model-evolution.

1 Introduction

Due to the high training costs of large language models, it is common practice to fine-tune already pre-trained language models to adapt them for specific applications. This fine-tuning approach often allows us to achieve excellent performance in specific data domains or tasks at a relatively lower cost (Chen et al., 2021). However, the challenge lies in the fact that fine-tuning (Dodge et al., 2020) the same model in different task scenarios may result in performance variations, meaning that the results may not be satisfactory when testing the same model in different contexts. Therefore, our objective is to integrate knowledge from models trained in different scenarios to enhance the model's performance in cross-domain or cross-task scenarios (Wortsman et al., 2022b), without the need for further training or extra training data.

Mainstream knowledge fusion methods can be divided into two main categories. The first involves extensive training on large datasets across multiple tasks to learn new model parameters with shared representations, such as in multi-task learning. The second relies on fusing existing models from specific scenarios without requiring extensive data. While multi-task learning generally improves overall performance, it has significant drawbacks: the need for abundant annotated data for all tasks and the complexity and time consumption of the training phase, especially with dataset combinations (Ruder, 2017). In contrast, model merging methods do not require retraining models and do not raise concerns about data privacy. In this paper, we primarily delve into the second category of methods and introduce an innovative model evolution approach inspired by Darwinian evolution theory (Shafiee et al., 2018). In short, we compare our model evolution approach with other prevalent knowledge fusion methods, detailing their distinct features in Table 1.

In fact, the problem of model merging can be reformulated as an optimization problem that aims to find the most valuable information for knowledge fusion in order to achieve better outcomes than any individual model alone. For instance, (Jin et al., 2023) employed a simple linear regression approach for optimization, while model soups (Wortsman et al., 2022a) implemented a greedy search method. In this paper, we consider the adoption of a more robust evolutionary algorithm for optimiza-

* Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2024, pages 11727–11742

	Ensemble	Model Merging	Multitask Learning	Federated Learning	Model Soups	**Model Evolution (ours)**
Retraining	✗	✗	✓	✓	✗	✗
High Memory Cost	✓	✗	✗	✗	✗	✗
Round(s)	Single	Single	Single	Multiple	Greedy	Greedy
Data	No	A Few Examples	Train Datasets	Private	Dev Datasets	Dev Datasets
Key Technique	Inference	Matrices Computing	Distribution	Back-Propagation	Search	Evolution
Peak GPU Memory	✗	✓	✓	✓	✗	✗

Table 1: Comparison of different knowledge fusion methods. Round means the number of times the models are edited when implementing a certain knowledge fusion method. Peak GPU memory is used to compare the GPU memory requirements.

tion. Evolutionary algorithms offer several advantages, including their outstanding performance in handling complex, high-dimensional, and nonlinear problems, as well as their relative insensitivity to local optima. Traditionally, evolutionary algorithms are primarily used in neural architecture search (NAS) (Awad et al., 2020). However, in this paper, we pioneer their application to the selection of important weights in language models for knowledge fusion.

As shown in Figure 1, our approach first processes models fine-tuned in different environments as an initial population. We then generate a new population through mutations and recombination among different individuals within the population. Subsequently, we validate the performance of the new population on development environment datasets and preserve elite individuals for updating. After evolving with enough generations, we select individuals with the best performance as the evolved model. Our evolutionary algorithm is firmly rooted in the task vector (Ilharco et al., 2022) methodology, wherein we derive the differential vectors between two individuals during the mutation operation. We delve deeper into the task vector concept from two distinct perspectives. Firstly, we employ a scaling factor (F), to search and regulate the weights of the difference vectors. This is crucial for aligning the expectations of model outputs, as illustrated in (Yu et al., 2023). Secondly, we utilize the crossover ratio (Cr), to maintain a certain level of sparsity, thereby preventing interference among different individuals, as demonstrated in (Yadav et al., 2024). Additionally, by conducting random searches for crossover parameters, we are able to obtain more optimal per-parameter coefficients for model merging.

We conduct knowledge fusion experiments across various difficulty levels, employing different types of models, such as RoBERTa, DeBERTa, T5 and GPT2. These experiments cover sentiment classification tasks in diverse data domains, as well as benchmark tasks from the GLUE dataset. The

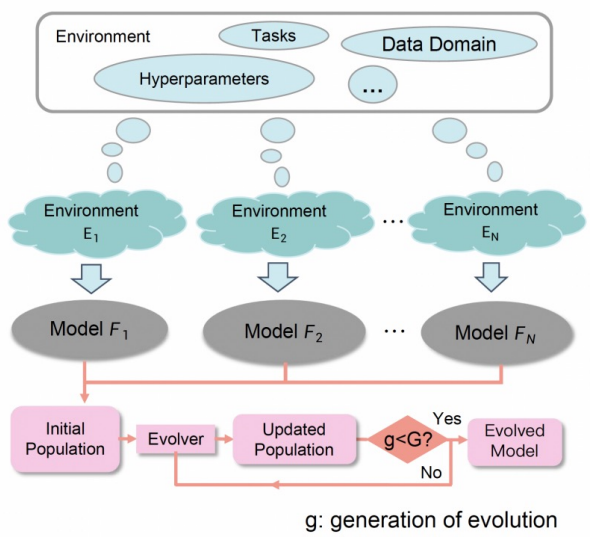


Figure 1: Key idea illustration. The key idea in our model evolution method is to aggregate multiple models $f_{1..N}$ from various environments into a population, which is then iteratively updated through greedy evolutionary rounds.

experimental results consistently demonstrate that our proposed method effectively enhances performance across all experimental settings. Furthermore, our approach can be synergistically combined with previous model merging methods (*e.g.*, *fisher* (Matena and Raffel, 2022), *regmean* (Jin et al., 2023)), resulting in further improvements in knowledge fusion performance. This combined approach significantly outperforms baseline methods and previous techniques. Notably, our method also exhibits superior generalization performance when applied to data domains beyond the scope of multiple datasets. To summarize, our key contributions include:

- **Innovative model evolution method:** We propose a novel knowledge fusion method from an evolutionary perspective, by evolving weights of language models.
- **Improved knowledge fusion performance:** Our method consistently enhances knowledge fusion performance across a broad spectrum of data domains and tasks.

- **Effective integration with existing model merging methods:** Our approach can be effectively enhanced and augmented through the integration of existing model merging techniques.

2 Related Work

2.1 Knowledge Fusion

Numerous studies have shown that aggregating knowledge from multiple datasets can enhance the performance of a single model across various data domains and different tasks (Poth et al., 2021). This approach is also applicable to out-of-domain data (Wang et al., 2020b). (Frankle et al., 2020) demonstrated the effectiveness of simple weight averaging in model fusion, exhibiting better performance than pre-training methods. (Matena and Raffel, 2022) proposed *fisher-weighted averaging* to merge models with different architectures, taking into account the importance of each parameter. (Jin et al., 2023) investigated model fusion using regression mean, re-weighting, and linearly combining rows within the weight matrix. (Wortsman et al., 2022a) introduced *greedy soup*, a technique to obtain robust results by searching for different average weights from multiple fine-tuned models. (Ilharco et al., 2022) proposed the concepts of task vectors to improve pre-trained models on multi-tasks and (Yadav et al., 2023) worked on the interference problems of task vectors to avoid the loss of valuable information. In addition to these knowledge fusion methods that do not require training, there are many knowledge fusion strategies that require complex training environments. Multi-task learning, as explored by (Ruder, 2017), improves performance across various tasks by sharing knowledge. Federated learning (Wang et al., 2020a) is a collaborative decentralized privacy-preserving technology designed to overcome the challenges of data silos and data sensitivity. Our evolutionary algorithm is firmly rooted in the task vector (Ilharco et al., 2022) methodology, wherein we derive the differential vectors between two individuals during the mutation operation. The methodology of task vectors has recently emerged as a promising approach for model merging and knowledge fusion, as evidenced by studies such as (Yadav et al., 2024; Ortiz-Jimenez et al., 2024). These investigations underscore the capacity of differential vectors, acquired through differentiation of pre-trained and fine-tuned models, to facilitate model editing. Overall, our approach harnesses the power of task vectors to enhance model merging and knowledge fusion, offering insights into both the theoretical and practical aspects of this methodology.

2.2 Evolutionary algorithms

Of particular relevance to our work is evolving algorithms (EAs), which provide an alternative path for addressing optimization problems in deep neural networks (DNNs) without relying on gradient information. The fundamental idea behind EAs is to combine the structures and weights of a group of neural networks and continually evolve them in the direction of improved global fitness to enhance model performance. These methods encompass genetic algorithms (Montana et al., 1989), genetic programming (Suganuma et al., 2017), differential evolution (DE) (Pant et al., 2020; Jiang et al., 2024), and evolutionary strategies (Salimans et al., 2017), among others. Neuro-evolution techniques, such as NEAT (Neuro Evolution of Augmenting Topologies) (Stanley and Miikkulainen, 2002), have demonstrated the ability to design simpler neural network architectures for improved performance, particularly in reinforcement learning tasks. However, it's important to note that EA methods typically perform well on small datasets and small-scale DNNs (Piotrowski, 2014). When applied to large-scale datasets, these methods tend to converge slowly and may even fail to converge (Piotrowski, 2014). In this paper, we propose model evolution which is motivated by the fact that model merging is neither amenable to traditional gradient-based optimization methods, nor are simple techniques like grid search sufficient. Therefore, we turn to evolutionary algorithms, which show promise for effectively addressing the model fusion problem.

3 Evolving Weights of Language Models

The goal of model evolution is to combine multiple fine-tuned language models into a more powerful single model. We achieve this by simulating the evolution process of a neural network population, as shown in Figure 2. We use the same pre-trained checkpoint and fine-tune it in different environments to create the initial population. As all individuals share the same model architecture, this enables our evolution algorithm to perform mutations and recombinations among individuals within the parameter space.

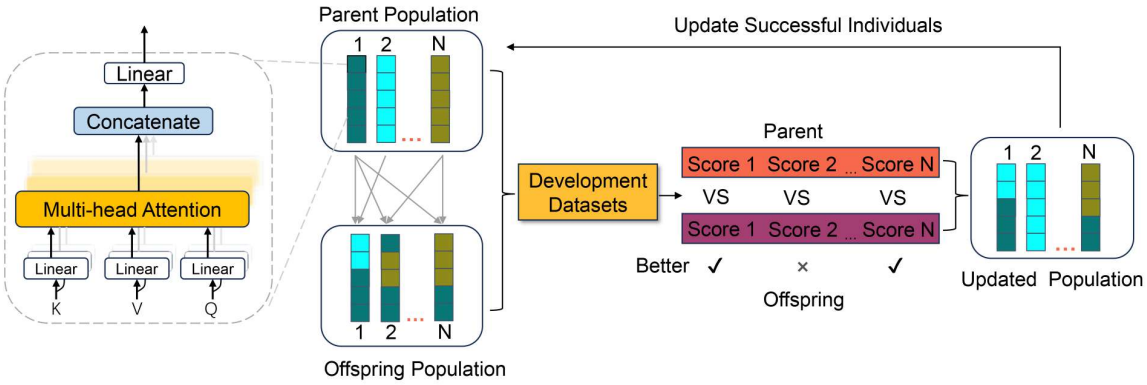


Figure 2: The process of evolving weights of language models. `Evolver` involves aggregating the weights of language models into a population and generating offspring models through mutation and crossover operations.

3.1 Evolutionary Strategy: `Evolver`

Population Initialization. For the optimization problem of model merging, an original set of individuals (population) is initialized. The parameters of each of N models are flattened into a one-dimensional vector, forming a set of candidate solutions. In this way, we obtain a set of candidate individuals represented by $\theta = \theta_i, i = 1, ..., N$. Here, N denotes the size of the population, and $\theta_i = (\theta_{i,1}, \theta_{i,2}, ..., \theta_{i,d})$ represents each candidate individual, where d is the dimension.

Evolution Process. We simulate the evolution process of a population of neural networks using the differential evolution algorithm (Pant et al., 2020). Each generation consists of three main steps: mutation, crossover, and updating.

Mutation: For each candidate individual θ_i, we randomly select two other candidate individual θ_{r_1} and θ_{r_2}, where r_1 and r_2 are two distinct random integers less than or equal to m. We use a scaling factor F to adjust the differences between θ_{r_1} and θ_{r_2}, and then add them to θ_i to obtain the mutated solution $\theta_i^\star = \theta_i + F \times (\theta_{r_1} - \theta_{r_2})$, where F is used to control the weights of the difference vector in the new parameter set.

Crossover: The computation for crossover is as follows:

$$\theta_{i,j}^\star = \begin{cases} \theta_{i,j}^\star & \text{if} \quad \text{rand}(0,1) \leq Cr, \\ \theta_{i,j} & \text{otherwise.} \end{cases} \tag{1}$$

where Cr is the pre-set crossover degree threshold between the new individual and the parent solution, and the setting of the threshold Cr can impact the ratio of elements selected in a mutated solution.

Updating: Throughout the process, we convert the offspring population vectors into models and conduct inference to get performance scores for these models on the development dataset. As demonstrated in the equation below, We sequentially evaluate the performance scores of offspring individuals in comparison to their parent one by one. If an offspring performs better, we then replace the corresponding parent individual with it, thereby updating the parent population.

$$\theta_i = \begin{cases} \theta_i^\star & score(\boldsymbol{\theta}_i^{\star g}) > score(\boldsymbol{\theta}_i^g) \\ \theta_i & \text{otherwise.} \end{cases} \tag{2}$$

3.2 Integration with Other Merging Methods.

Our proposed model evolution method can be integrated with any other model merging technique. Specifically, we can select the optimal individual from the updated population as the outcome of the evolution process, which we refer to as a simple `Evolver`. Furthermore, we can also apply other model merging techniques to the updated population as a further improvement measure. Such integration can be implemented both at the stage of obtaining the final evolved model and during the calculation of the updated population's scores for iterative updates. This process has been summarized as Algorithm 1 and is detailed in the flowchart provided in Appendix A.

3.3 Computation Efficiency

Memory Analysis: The memory expanse during our model evolution is mainly related to the size of the population: $\sum_{i=1}^{N} d$, where N represents the number of populations, d is the dimension of the model parameter space. Since we avoid inner product matrices computing as in previous model

Algorithm 1 Integrating Model Evolver with Other Model Merging Methods

When inference with the updadted population
  if this is a simple evolver
  then
    evaluate the performance of individual $\theta_i^\star$,
  else
    merging $\theta_i^\star$ with other individuals with a specific merging method,
    evaluate the performance of merged model.

merging methods such as *fisher* and *regmean*, and the parameters is updated mainly through forward propagation of greedy models, the peak GPU memory consumption is consequently lower.

Time Consumption: We hereby provide the formula and key definitions required to calculate the runtime. The total evolving time can be calculated as $T = G \times N \times (t_1 + L \times t_2) \approx G \times N \times L \times t_2$, where $t_1 \ll t_2$ in practice. Here, G is the total generations for updating, N is the population size, t_1 is the time for mutation and crossover for each individual, L is the number of samples of development datasets, t_2 is the time for inference of a sample on one model.

4 Experimental Setup

4.1 Evaluation Settings

Problems. We primarily consider the following three main advantages when testing our proposed model evolution method: Firstly, we anticipate that our evolved model f_M, created by integrating knowledge from individual models $f_{1..N}$ finetuned on diverse datasets $D_{1..N}$, will have competitive performance across various data sources without necessitating separate models for each domain or task. Then, by evolving different models excelling in various tasks $D^t_{1..N_t}$, we aim to enhance multi-task handling capacity, avoiding the complexity of retraining as in MTL, while enabling cross-task inferencing within a single model. Lastly, our goal is for the evolved model f_M to excel in generalizing to out of distributio (OOD) test sets $D^o_{1..N_o}$, thereby enabling it to effectively handle new and unforeseen data from domains or tasks not encountered during training. $D_{1..N}$.

Datasets. We use the GLUE datasets (Wang et al., 2018) as the cornerstone of our investigation into the performance of evolved models. This inquiry encompasses two key dimensions: training for non-independent and identically distributed (non-i.i.d.) partitions and training for disparate tasks. Detailed dataset information and additional experimental results are available in Appendix D.

Implementation. We use Hugging Face's transformer library (Wolf et al., 2019) to access pre-trained language models and conduct fine-tuning. All our models, denoted as f_i, follow the same architecture and employ identical pre-trained model weights θ for initialization, as described in (McMahan et al., 2017). Our experiments include various pre-trained models as starting points, such as RoBERTa-base (Liu et al., 2019), the lightweight DistilBert (Khanuja et al., 2021) and well-established model DeBERTa-large-v3 (He et al., 2022). Besides the models with encoder-only architecture, we also conduct experiments with encoder-decoder architecture, T5-base-v1.1 (Raffel et al., 2020) and decoder-only architecture, GPT2 (Radford et al., 2019) and large language model MiniCPM (Hu et al., 2024).

Population Initialization. The initial population for model evolution is created through fine-tuning a model with the same initialization but on different data domains or various tasks. While fine-tuning DistilBERT-base, RoBERTa-base, and DeBERTa-large, we maintained a constant initial learning rate of 1e-5. Throughout our experiments, we consistently utilized the AdamW optimizer with a warm up learning rate during the initial 6% of training. Our model training utilized a batch size of 16 and encompassed 10 epochs for the GLUE task and 30 epochs for the emotion classification task.

4.2 Compared Methods

We mainly compare `Evolver` with existing merging methods, including *simple*, *fisher* (Matena and Raffel, 2022) , *regmean* (Jin et al., 2023) , *greedy soup* (Wortsman et al., 2022a) and *TIES* (Yadav et al., 2023). To gain a better grasp of the advantages of model merging, we show the performance prior to model evolution, the average performance of the population (*Avg.* $f_{1..N}$) and the best-performing individual (*Best.* $f_{1..N}$), more details is shown in Appendix D.3. Moreover, we provide the performance for the model trained on a specific task *domain-specific*. We also compare with *model ensembling*, where the logits from predictions are extracted, averaged, and then subjected

to the argmax operation. Lastly, we use multi-task learning (MTL) as a benchmark.

5 Experimental Results

We assess the performance dynamics of the model evolution method across a range of scenarios with varying levels of complexity. These scenarios include: (1) performance across different data domains used for fine-tuning individual models. (2) performance across different tasks, when individual models are specialized in only one task. (3) OOD generalization performance on datasets from previously unseen domains.

5.1 Model Evolving Across Data Domains

Evolving All Domain-Specific Models. We conduct experiments of evolving five domain-specific models for emotion classification, and the results are recorded in Table 2. Multi-task learning (MTL) achieves performance similar to that of domain-specific models, suggesting that a single model has the capability to acquire knowledge from multiple domains. Besides, *model soup* approach, which greedily selects fusion objects, leads to some improvements over the best individual. However, these improvements are relatively marginal compared to model merging methods.

We compare model evolution with three other knowledge merging methods. The basic version of model evolution outperformed *fisher* method and achieved performance that is comparable to *regmean* on some tasks. Furthermore, we explore the combined use of model evolution and model merging methods, demonstrating that our approach further enhances existing model merging methods and consistently yields improvements across different models. Also, we demonstrate results with shared and different classification head initialization (Same Head Init/Diff Head Init). It can be observed that *fisher* and *regmean* produce unstable and highly variable results with different initialization, while the performance of the model evolution method is less affected by this factor. Therefore, our proposed model evolution method shows more stable performance when deploying and maintaining a single model across multiple domains.

Evolving Pairwise Domain-Specific Models In addition to the fusion experiment involving all the models, we also consider pairwise domain-specific models for knowledge fusion in the context of the emotion classification task. After pairing models from a set of 5, we conduct 10 ($\mathcal{C}_5^2$) runs with all the same methods described in section 5.1. The result of these 10 runs are averaged and recorded in Table 3. We observe clear differences between model evolution and model merging methods, with the *TIES-evolver* achieving the best performance when combining pairwise models finetuned from different domains.

Evolving Models Trained on Non-i.i.d. Partitions. We adopt synthetic data divisions to simulate non-independent and identically distributed (non-i.i.d.) partitions of the same dataset, across the 8 tasks included in the GLUE benchmark. Given the inconsistency in the performance of *regmean* and *fisher* methods under different seeds, we choose to average the result of eight different random seeds. The results in Figure 3 indicate that the implementation of model evolution outperformed previous methods on all tasks, with clear improvements observed particularly on *cola* and *mnli* datasets. Details of this section are shown in Appendix D.4.

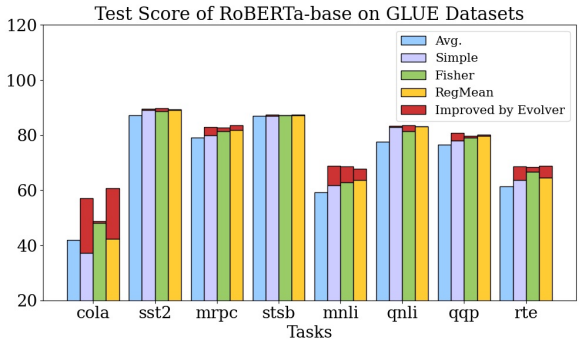


Figure 3: Result of model evolution on non-i.i.d. partitions of GLUE benchmark datasets.

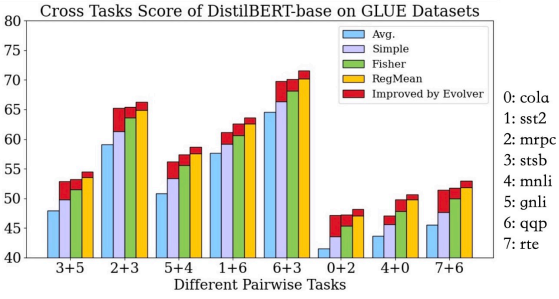


Figure 4: Result of model evolution across different pairwise tasks on GLUE benchmark.

5.2 Evolving Models Across Different Tasks

Here we examine the effectiveness of model evolution in merging models finetuned on different tasks. We use the RoBERTa-base model and train indi-

Method	Encoder-Decoder	Encoder-only			Decoder-only	
	T5-base	RoBERTa-base Same / Diff Head Init.	DistilBERT-base Same / Diff Head Init.	DeBERTa-large Same / Diff Head Init.	GPT2	MiniCPM
Avg. $f_{1..N}$	32.07	26.08	24.55	27.68	23.35	35.12
Best. $f_{1..N}$	34.08	29.27	29.91	31.93	26.76	37.23
Ensemble	33.95	38.77 / 27.73	26.51 / 25.43	29.88 / 29.27	26.82	37.05
Greedy Soup	34.10	30.34	30.11	31.93	26.76	37.31
Simple	39.47	23.18	23.70	3.75	21.54	42.44
`Evolver` (ours)	41.25	33.27 / 30.04	28.95 / 26.29	23.90 / 21.55	23.41	44.76
Fisher	39.12	26.09 / 22.43	26.39 / 22.61	12.83 / 20.42	24.93	\
Fisher_`Evolver` (ours)	40.36	28.41 / 25.71	27.63 / 24.75	17.22 / 22.95	25.66	\
RegMean	40.24	38.74 / 32.58	33.37 / 28.29	38.33 / 18.92	30.14	\
RegMean_`Evolver` (ours)	41.83	39.87 / 34.28	35.67 / 31.11	39.58 / 21.79	32.26	\
TIES	41.24	39.66 / 35.32	35.55 / 30.14	39.22 / 21.67	32.11	45.81
TIES_`Evolver` (ours)	**43.16**	**41.33 / 37.42**	**37.12 / 31.48**	**40.61 / 23.24**	**33.56**	**46.24**
Domain-Specific	49.31	51.38	48.79	52.81	47.62	54.32
MTL	48.98	47.73	45.23	51.77	44.31	52.14

Table 2: In-domain performance when merging emotion classification models. The initial population are all 5 domain specific models or pairwise models. \ indicates the original merging methods can not been conducted due to high GPU cost. **Bold** numbers indicate the best performance across different model merging algorithms. All the results we reported are averages of trials conducted with 5 different random seeds.

Model	Simple	`Evolver`	TIES	**TIES_`Evolver`**
T5-base	38.82	40.21	46.35	**47.92**
RoBERTa-base	37.78	39.13	45.56	**46.89**
DistillBERT-base	36.76	38.85	42.09	**43.22**
DeBERT-large	38.11	39.46	45.87	**46.78**
GPT2	36.33	38.25	41.85	**42.62**
MiniCPM	40.22	42.71	47.02	**48.83**

Table 3: In-domain performance when merging pairwise domain-specific emotion classification models. All the results we reported are averages of 10 (C_5^2) runs after paring models from a set of 5.

vidual models with the complete training data for each task in the GLUE benchmark. Following this, we randomly select two task-biased individuals to conduct pairwise model evolution. Specifically, we exclude the parameters in task-specific headers due to their potential dimension variance depending on tasks. We summarize the results of eight different task pairs in Figure 4, which show that our model evolution strategy performs effectively when fusing knowledge from diverse tasks.

5.3 Model Evolving for Out-of-Domain Generalization

For Out-of-Domain (OOD) generalization, we can obtain the same conclusion as our in-domain experiments, indicating that model evolving leads to improvements in OOD generalization performance, as summarized in Table 4. We notice that in the case of RoBERTa-base and DistilBERT models initialized with different heads, the basic `Evolver` has outperformed *fisher-evolver* and *regmean-evolver*. A plausible explanation for this is that previous methods may suffer from the negative impact of extremely poor-performing individual models. In contrast, model evolution possesses an elimination mechanism that effectively removes poorly-performed individual models during the competition process. This ensures that only models with superior performance are retained and merged. Consequently, model evolution can reduce the negative impact of under-performing models on the merged results.

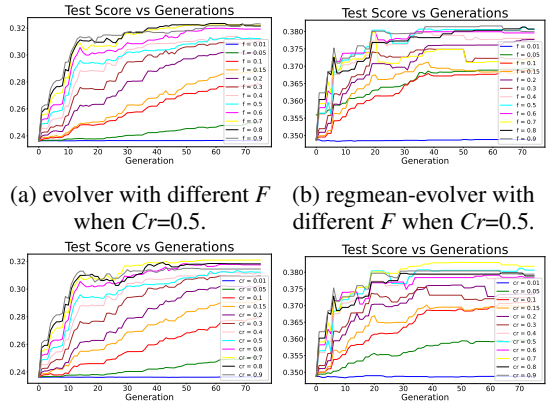
5.4 Mutation and Crossover

We also test the impact of different values of scale factor F for mutation and crossover ratio Cr, as shown in Figure 5. Due to the inherent randomness in the search process of evolutionary algorithms, we conducted each experiment using four different random seeds and then calculated the average results. In general, the study findings indicate that the performance of the evolutionary algorithm improves as the parameters F or Cr increase until reaching 0.5. However, when F or Cr exceeds 0.5, there is minimal improvement in performance, and no clear pattern is observed. Therefore, in all experiments conducted in this paper, we have consistently used $F = 0.5$ and $Cr = 0.5$.

In addition, *regmean* method requires decreasing the non-diagonal items of the inner product matrices by multiplying a scalar α. Since the effectiveness of the *regmean-evolver* method can be influenced by the hyperparameter α, we also test

Method	Encoder-Decoder	Encoder-only			Decoder-only	
	T5-base	RoBERTa-base Same / Diff Head Init.	DistilBERT-base Same / Diff Head Init.	DeBERTa-large Same / Diff Head Init.	GPT2	MiniCPM
Avg. $f_{1..N}$	30.12	20.92	19.69	21.17	18.63	35.24
Best. $f_{1..N}$	37.41	29.46	29.55	31.07	27.88	38.93
Ensemble	27.92	11.36 / 10.90	9.60 / 9.19	11.09 / 9.26	8.77	29.54
Greedy Soup	15.42	13.43	15.26	4.67	11.61	31.70
Simple	38.61	11.56	13.21	0.24	10.25	40.21
`Evolver` (ours)	39.26	17.53 / 17.16	19.02 / 18.42	13.33 / 12.78	15.87	42.48
Fisher	37.72	16.21 / 14.28	17.77 / 15.69	5.57 / 27.61	15.16	\
Fisher_`Evolver` (ours)	38.87	16.98 / 15.44	18.85 / 17.36	15.46 / **30.41**	16.34	\
RegMean	39.46	21.09 / 14.12	18.97 / 16.21	15.92 / 4.88	20.33	\
RegMean_`Evolver` (ours)	41.13	23.41 / 16.45	21.44 / 18.31	18.49 / 11.27	22.07	\
TIES	40.48	22.63 / 16.34	19.76 / 17.52	16.88 / 14.72	21.92	41.35
TIES_`Evolver` (ours)	**42.53**	**24.11 / 19.02**	**22.64 / 19.06**	**18.74** / 16.31	**22.75**	**42.86**
MTL	37.64	27.41	25.63	31.45	25.26	44.62

Table 4: Out-of-domain Performance. All the results we reported are averages of trials conducted with 5 different random seeds.



(a) evolver with different F when Cr=0.5.

(b) regmean-evolver with different F when Cr=0.5.

(c) evolver with different Cr when F=0.5.

(d) regmean-evolver with different Cr when F=0.5.

Figure 5: Result of RoBERTa-base when evolving all domain specific models on emotion datasets with different with different scale factor F and crossover ratio Cr.

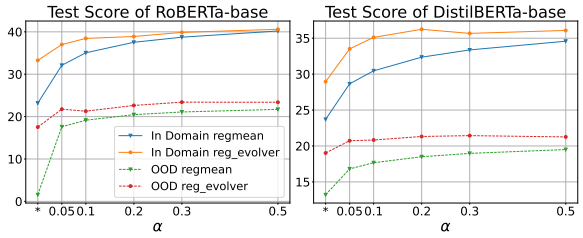


Figure 6: The improvement of *regmean-evolver* with different scale α on the emotion dataset when evolving all domain specific models. $*$ denotes results of simple *evolver*.

the performance of *regmean-evolver* under different α parameters, as shown in Figure 6.

5.5 Integration with Coefficient Search

The coefficient search is a promising scheme to improve the model merging performance, by searching the optimal α. The proposed model evolution can also be integrated with the coefficient search method by searching the optimal scale factor f. We have performed the grid search of α and f with intervals of 0.05 from 0.1 to 0.9. We present the result in Table 5. In the setting of Simple, Fisher and RegMean, the results show that a default version of evolver (with scale factor $f = 0.5, cr = 0.5$) outperforms the coefficient search results. Notably, the integration with scale factor search further boosts the performance of the evolver, which is worth further investigation. Especially, the crossover ratio Cr in model evolution could also be the subject of coefficient search. Many adaptive schemes are also available in the realm of evolution algorithms, like SADE (Qin and Suganthan, 2005) and SHADE (Tanabe and Fukunaga, 2013), providing a possibility of future algorithmic development.

5.6 Analysis

We demonstrate the evolutionary process when combined with other model fusion methods. From Figure 7, we can observe that when model evolver is combined with *fisher* or *regmean* method, the upper bounds of the evolutionary approach can be enhanced. Additionally, we present the test results of the evolutionary algorithm on the development dataset. It is evident that as individual models are trained on the development dataset, their performance on the test set gradually improves. This indicates that our model evolution method indeed has the ability to optimize and learn.

In addition, we visualize a two dimensional slice of the average test accuracy landscape when evolv-

Model	Simple (coefficient search)	Evolver (scale factor search)	Fisher (coefficient search)	Fisher_Evolver (scale factor search)	RegMean (coefficient search)	RegMean_Evolver (scale factor search)
RoBERTa-base	37.78 (**38.83**)	39.13 (**39.98**)	37.11 (**38.96**)	40.34 (**41.23**)	46.56 (**46.82**)	46.89 (**47.03**)
DistillBERT-base	36.76 (**37.63**)	38.85 (**39.67**)	34.52 (**37.54**)	40.37 (**41.31**)	43.09 (**43.14**)	43.22 (**43.31**)
T5-base	38.82 (**39.91**)	40.21 (**41.11**)	38.08 (**39.22**)	41.46 (**42.55**)	47.35 (**47.84**)	47.92 (**48.06**)

Table 5: **Coefficient Search Result** when merging pairwise emotion classification models. Simple, Fisher and RegMean are model merging algorithms for comparison. All the results we reported are averages of 10 (C_5^2) runs after paring models from a set of 5.

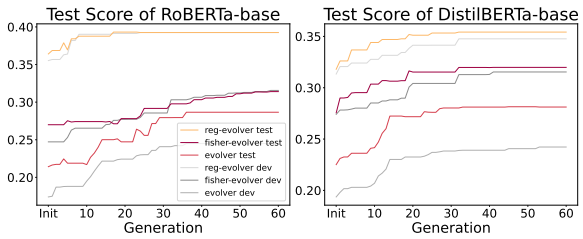


Figure 7: In-domain score of model evolution on the emotion dataset with all domain specific models.

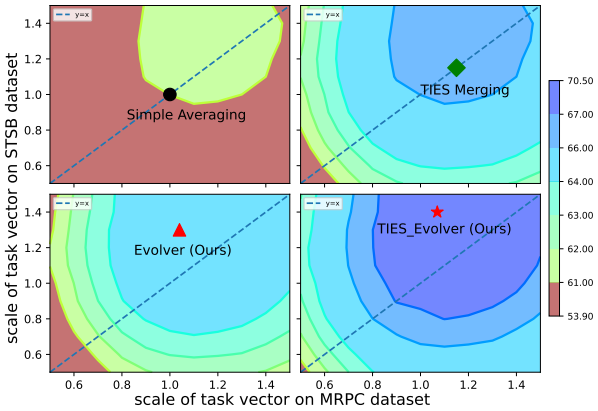


Figure 8: The average performance of merged T5 model on MRPC and STSB test dataset. By observing the density and distribution range of the contour lines, it is evident that our evolver method performs better. Additionally, previous methods typically utilize a uniform scale across different task vectors, whereas the evolved model exhibits a preference for varying scales determined through iterative evolution rounds.

ing pairwise fine-tuned T5 models, as shown in Figure 8. In this experiment, we use the zero-shot initialization and fine-tune twice, independently, to produce task vectors on MRPC and STSB datasets. We analyze the positions of the model results after fusion by different methods in their corresponding landscapes. Methods like TIES and other merging approaches typically process the task vectors of models on different tasks and then perform a direct weighted average. This results in a fixed relative scale among different tasks during model fusion. However, from the landscape, we can see that the optimal fusion performance usually has different preferences for different tasks. Our proposed model evolution method can dynamically update the scales of different tasks to promote better model fusion effects.

The proposed model evolution also has some advantages in other aspects: (1) Model evolution can leverage the benefits of a larger population size without being significantly affected by individuals with extremely poor performance. This advantage is a result of the survival of the fittest mechanism in the model evolution process. By favoring those with most effective performance, we reduce the influence of individuals with extreme differences, leading to a more robust and reliable system. (2) Model evolution method can effectively maintain low peak GPU memory usage. This is primarily attributed to its sequential forward inference of individual models, as opposed to most previous model merging techniques that require additional GPU memory for computing inner product matrices in the model parameter space. This advantage significantly reduces GPU memory consumption and extends the range of feasible solutions for large-scale language models.

6 Conclusions

We introduce a novel knowledge fusion method, called model evolution, inspired by evolutionary algorithms. This approach significantly boosts the performance of model merging in diverse NLP contexts. Model evolution stands out by aggregating model weights into a population and updating it with superior offspring models, all without requiring extra training data. The most significant contribution of our work lies in our experimental discovery that knowledge fusion can benefit from searching per-model and per-parameter coefficients. This type of search-based, gradient-free optimization algorithm proves to be an effective tool for model merging and warrants greater attention in the research community. Our extensive experiments validate its superiority over previous techniques.

Limitation

The limitations of the model evolution method can be summarized in three main aspects: (1) the necessity for a high-quality development dataset with consideration for data privacy, (2) the requirement for a cautious selection of hyperparameters F and Cr, (3) and the significance of conducting further theoretical analysis of evolutionary algorithm principles. Future research offers several promising directions. Firstly, exploring advanced optimization strategies within evolutionary algorithms, including adaptive approaches and hyperparameter selection based on historical performance, holds the potential for enhancing the method's effectiveness. Secondly, extending knowledge fusion to a more complex training environment by considering hyperparameters, multimodal and exploring different training methods like unsupervised or supervised learning can provide a comprehensive understanding of its applicability. Thirdly, arithmetic operations in (Ilharco et al., 2022) for model edit can be analyzed. Lastly, evaluating the approach on larger language models can provide insights into its scalability.

Ethical Considerations

Our approach has been evaluated on GLUE. We explicitly claim that the applicability of our method and findings may be confined to similar datasets or domains. The performance of our method on other specific datasets or domains remains uncertain. Thus, there are potential risks when applying our method to privacy-sensitive or high-risk datasets. We should be cautious and verify whether the method generates correct answers.

Acknowledgements

This work was supported in part by Shenzhen College Stability Support Plan (GXWD20231128103232001), Department of Science and Technology of Guangdong (2024A1515011540), Shenzhen Start-Up Research Funds (HA11409065), National Natural Science Foundation of China (12204130), the Ministry of Higher Education Malaysia through the Fundamental Research Grant Scheme (FRGS/1/2023/ICT02/XMU/02/1), and Xiamen University Malaysia through Xiamen University Malaysia Research Fund (XMUMRF/2022-C10/IECE/0039 and XMUMRF/2024-C13/IECE/0049).

References

Cecilia Ovesdotter Alm, Dan Roth, and Richard Sproat. 2005. Emotions from text: machine learning for text-based emotion prediction. In *Proceedings of conference on empirical methods in natural language processing (EMNLP)*, pages 579–586.

Noor Awad, Neeratyoy Mallik, and Frank Hutter. 2020. Differential evolution for neural architecture search. *arXiv preprint arXiv:2012.06400*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Ben Chen, Bin Chen, Dehong Gao, Qijin Chen, Chengfu Huo, Xiaonan Meng, Weijun Ren, and Yang Zhou. 2021. Transformer-based language model fine-tuning methods for covid-19 fake news detection. In *Combating Online Hostile Posts in Regional Languages during Emergency Situation: First International Workshop, CONSTRAINT*, pages 83–92. Springer.

Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah Smith. 2020. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. 2020. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3259–3269.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2022. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*.

Runhua Jiang, Guodong Du, Shuyang Yu, Yifei Guo, Sim Kuan Goh, and Ho-Kin Tang. 2024. Cade: Cosine annealing differential evolution for spiking neural network. *arXiv preprint arXiv:2406.02349*.

Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2023. Dataless knowledge fusion by merging weights of language models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*.

Simran Khanuja, Melvin Johnson, and Partha Talukdar. 2021. Mergedistill: Merging language models using pre-trained distillation. In *Findings of the Association for Computational Linguistics (ACL-IJCNLP)*, pages 2874–2887.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957*.

Vicki Liu, Carmen Banea, and Rada Mihalcea. 2017. Grounded emotions. In *Proceedings of the Seventh International Conference on Affective Computing and Intelligent Interaction (ACII)*, pages 477–483.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Michael S Matena and Colin A Raffel. 2022. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:17703–17716.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Saif Mohammad. 2012. # emotional tweets. In *Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 246–255.

Saif Mohammad and Felipe Bravo-Marquez. 2017. Wassa-2017 shared task on emotion intensity. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 34–49.

Saif M Mohammad, Xiaodan Zhu, Svetlana Kiritchenko, and Joel Martin. 2015. Sentiment, emotion, purpose, and style in electoral tweets. *Information Processing & Management*, 51(4):480–499.

David J Montana, Lawrence Davis, et al. 1989. Training feedforward neural networks using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 89, pages 762–767.

Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. 2024. Task arithmetic in the tangent space: Improved editing of pre-trained models. *Advances in Neural Information Processing Systems (NeurIPS)*, 36.

Millie Pant, Hira Zaheer, Laura Garcia-Hernandez, Ajith Abraham, et al. 2020. Differential evolution: A review of more than two decades of research. *Engineering Applications of Artificial Intelligence*, 90:103479.

Adam P Piotrowski. 2014. Differential evolution algorithms applied to neural network training suffer from stagnation. *Applied Soft Computing*, 21:382–406.

Clifton Poth, Jonas Pfeiffer, Andreas Rücklé, and Iryna Gurevych. 2021. What to pre-train on? efficient intermediate task selection. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10585–10605.

A Kai Qin and Ponnuthurai N Suganthan. 2005. Self-adaptive differential evolution algorithm for numerical optimization. In *IEEE congress on evolutionary computation*, volume 2, pages 1785–1791.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2383–2392.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Klaus R Scherer and Harald G Wallbott. 1994. Evidence for universality and cultural variation of differential emotion response patterning. *Journal of personality and social psychology*, 66(2):310.

Hendrik Schuff, Jeremy Barnes, Julian Mohme, Sebastian Padó, and Roman Klinger. 2017. Annotation, modelling and analysis of fine-grained emotions on a stance and sentiment detection corpus. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 13–23.

Mohammad Javad Shafiee, Akshaya Mishra, and Alexander Wong. 2018. Deep learning with darwin: Evolutionary synthesis of deep neural networks. *Neural Processing Letters*, 48(1):603–613.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing (EMNLP)*, pages 1631–1642.

Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Carlo Strapparava and Rada Mihalcea. 2007. Semeval-2007 task 14: Affective text. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 70–74.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference (GECCO)*, pages 497–504.

Ryoji Tanabe and Alex Fukunaga. 2013. Success-history based parameter adaptation for differential evolution. In *IEEE congress on evolutionary computation*, pages 71–78.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. 2020a. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*.

Jing Wang, Mayank Kulkarni, and Daniel Preoţiuc-Pietro. 2020b. Multi-domain named entity recognition with genre-aware and agnostic inference. In *Proceedings of the 58th annual meeting of the association for computational linguistics (ACL)*, pages 8476–8488.

Alex Warstadt, Amanpreet Singh, and Samuel Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics (TACL)*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022a. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 23965–23998.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. 2022b. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems (NeurIPS)*, 36.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Appendix

A Explanation of Model Evolution

In this section, we provide a more comprehensive explanation of the principles underlying the Differential Evolution algorithm, furthermore, we offer an in-depth elucidation of the mutation process, providing a visual representation in Figure 9 to enhance clarity and understanding. Overall, the fundamental principle of the differential evolution algorithm involves randomly selecting three distinct individuals, performing a mutation operation to create a new candidate solution, using a crossover operation to refine the solution, and replacing the original solution if the new one performs better. This iterative process continues until certain stopping criteria are met.

To better illustrate our model evolution method, we have created a flowchart as shown in algorithm 2. The details of combining our proposed method with other models are provided in Step 3. The approach involves calculating an overall score by using model merging on the mutated individuals along with the non-mutated individuals in the current evolution. In contrast, the simple evolver determines the success of mutation based on the score of individual entities, while the combined approach assesses it based on the score of the entire population after individual mutation.

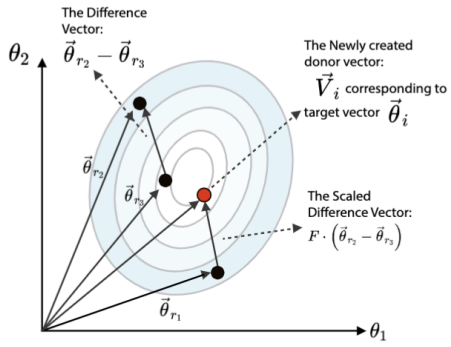


Figure 9: An illustration of the mutation process in difference evolution.

B Preliminaries

Fisher-weighted averaging (*fisher*) (Matena and Raffel, 2022) examines the importance of each weight F_i associated with each label by computing the norm of the logarithmic likelihood gradient. Specifically, the posterior probabilities of each model are interpreted as gaussian distributions $p(\theta|\theta_i, F_i)$, where the parameters θ for model i correspond to the Fisher information matrix F_θ. Finally, the fisher information for each parameter is used to perform a weighted average of the parameters, integrating the parameters of different models into a single model.

Regression mean *(regmean)* (Jin et al., 2023) expands the solution of a linear optimization problem to K models where $W_i, i \in \mathcal{K}$, denoted as $W_M = (\sum_i^{i\in\mathcal{K}} X_i^T X_i)^{-1} \sum_i^{i\in\mathcal{K}} (X_i^T X_i W_i)$. Each transformer model $f^{(j)}$ linear layer's corresponding $X_i^{(j)}$ is captured along with per-weights and its input inner product matrix, to compute the merged weights and produced merged model $f_M(x) = W_M^T x$. The scale of $X_i^T X_i$ exhibits substantial variation across different models. Additionally, a control mechanism is applied by multiplying $X_i^T X_i$ by $\alpha = \frac{1}{1+\gamma}$.

Model soups (*greedy soup*) (Wortsman et al., 2022a). Initially, models are ranked based on their development dataset scores. Subsequently, model parameters θ_i are chosen through a greedy search, and their inclusion in the gradient is determined by comparing the average validation set accuracy before and after their addition. The merged model's parameters can be represented as $\theta_S = \text{average(ingredients)}$.

TIES merging (*TIES*) (Yadav et al., 2023). This method is based on the concept of task vectors (Ilharco et al., 2022) and address the problem of two major sources of interference: (a) interference due to redundant parameter values and (b) disagreement on the sign of a given parameter's values across models. This method proposed TRIM, ELECT SIGN & MERGE (TIES-MERGING), which introduces three novel steps when merging models: (1) resetting parameters that only changed a small amount during fine-tuning, (2) resolving sign conflicts, and (3) merging only the parameters that are in alignment with the final agreed-upon sign

C Impact of Development Dataset

The availability of development datasets directly impacts the effectiveness of our model evolution approach. However, many publicly available datasets either do not provide development sets or widely use them as test sets. In our case, the development set of the GLUE dataset is used as a test set, so we

Algorithm 2 Model Evolution

Step 1 - Initializing the Population
Initialize population Θ ▷ A population of candidate solutions
$generation \leftarrow 0$ ▷ Initialize generation counter
$converged \leftarrow$ **False** ▷ Convergence flag
while not converged **do**
    Step 2 - Evolution Process: Mutation and Recombination
    for each candidate solution θ_i in Θ **do**
        Randomly select θ_{r1} and θ_{r2} ▷ Select random solutions
        $F \leftarrow$ Random scaling factor ▷ Control parameter for mutation
        $Cr \leftarrow$ Random crossover rate ▷ Control parameter for recombination
        Compute mutated solution $\theta_i^\star$ using θ_i, θ_{r1}, θ_{r2}, and F
        Perform recombination of $\theta_i^\star$ based on Cr and θ_i
        Step 3 - Model Inference
        if not combined with other model merging methods **then**
            Evaluate the performance of $\theta_i^\star$ on development data
        else
            Merging $\theta_i^\star$ with other models
            Evaluate the performance of merged model on development data
        end if
    end for
    Step 4 - Updating the Population
    $converged \leftarrow$ **True** ▷ Assume convergence
    for each candidate solution θ_i in Θ **do**
        if $\theta_i^\star$ outperforms θ_i **then** ▷ Comparing performance
            Replace θ_i with $\theta_i^\star$ ▷ Update population
            $converged \leftarrow$ **False** ▷ Reset convergence flag
        end if
    end for
    $generation \leftarrow generation + 1$ ▷ Increment generation counter
end while

utilize a small portion of the training dataset (approximately 5%) for model evolution. For non-i.i.d. partition methods, we also use only a subset of the same training data samples. In the case of the unified emotion dataset, we separately extract 10% of data from each of the five high-resource datasets for model evolution, following the same partitioning method as employed in (Jin et al., 2023).

For our model evolution approach, the quality of the development dataset can significantly impact performance, making the selection of a high-quality development dataset a crucial consideration. To address this, we conducted experiments on the emotion dataset using different lengths of model evolution methods. We performed experiments with both the simple evolver and regmean evolver, evolving all five domain-specific models. The experimental results are shown in Table 6, indicating that even with a short development dataset, model evolution can still be effective. However, as the length of the development dataset increases, the performance of model evolution tends to improve. Additionally, we included the test scores of simple methods as baselines for comparison.

Length	None	1/4	1/2	1
Evolver	23.18	30.14	32.03	33.27
Regmean_Evolver	38.74	39.43	39.57	39.87

Table 6: The performance of model evolution with different length of development dataset. *None* means evolver is not conduct and the test score of *simple averaging* and *regmean* method is recorded.

D Dataset, Metrics and Training Details

D.1 Emotion Classification Datasets

In order to investigate the performance of the sentiment classification task, we selected a diverse and challenging set of datasets. Among them, DailyDialogs (Li et al., 2017), CrowdFlower, TEC (Mohammad, 2012), Tales-Emotion (Alm et al., 2005), and ISEAR (Scherer and Wallbott, 1994) is utilized to train domain-specific model. For acessing OOD generalization performance, we use Emoint (Mohammad and Bravo-Marquez, 2017), SSEC (Schuff et al., 2017), ElectoralTweets (Mohammad et al., 2015), GroundedEmotions (Liu et al., 2017), and AffectiveText (Strapparava and Mihalcea, 2007). For OOD evaluation, we focus exclusively on the fundamental emotions: anger, disgust, fear, joy, sadness, and surprise. A detailed overview of the datasets and statistics is provided in Table 7.

	Train	Dev	Test
In-domain			
DialyDialog	72,085	10,298	20,596
CrowdFlower	27,818	3,974	7,948
TEC	14,735	2,105	4,211
Tales-Emotion	10,339	1,477	2,955
ISEAR	5,366	766	1,534
Out-of-domain			
Emoint			7,102
SSEC			4,868
ElectoralTweets			4,056
GroundedEmotions			2,585
AffectiveText			1,250

Table 7: Statistics of emotion classification datasets.

D.2 GLUE Benchmark Datasets

For the GLUE dataset, we utilized multiple tasks, including CoLA (Warstadt et al., 2019), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), STS-B (Cer et al., 2017), MNLI (Williams et al., 2018),QNLI (Rajpurkar et al., 2016), QQP, and RTE (Giampiccolo et al., 2007). These tasks cover various natural language understanding problems such as text classification, text similarity, and natural language inference. To assess our merged models, we tested them on the official development sets. We performed experiments by training models on non-i.i.d. partitions, creating various partition scenarios through random sampling. Each partition is uniformly sub-sampled to yield a total of 1,000 training examples per partition.

D.3 Definitions and Metrics

The performance of individual models involved in merging are reported: (1) the average performance of all individual models (**Avg.** $f_{1..N}$); (2) the performance of the best *single* individual model (**Best.** $f_{1..N}$), as determined by using the validation set; (3) the performance of the individual models corresponding to the training data set for each test set (**Domain-Specific**).

In evaluating merged models trained for non-i.i.d. partitions of the same dataset, we assessed their performance using a unified test set characterized by a joint distribution of all partitions. For merged models trained across different domains or tasks, we measured their performance across individual domains or tasks incorporated into the

Initial Population for Evolving	T5-base	RoBERTa-base	DistilBERT-base	DeBERTa-large	GPT2	MiniCPM
All Domain Specifis Models on Emotion Datasets	24.5	19.2	18.7	21.3	20.1	28.1
Pairwise Models on Emotion Datasets	9.3	7.3	7.1	8.1	7.6	13.2
None-iid Pairwise Models on GLUE Benchmark	7.2	5.7	5.5	6.2	5.9	10.6
Cross Tasks Pairwise Models on GLUE Benchmark	8.7	7.1	6.8	7.8	7.5	12.6

Table 8: **Time cost** (in the unit of minutes) of RegMean_Evolver on different experiments with 20 generations. T5-base is tested on single A800 GPU and other models are tested on single A6000 GPU. The time cost is mainly related to the size of model and the length of development dataset when conducting model evolution.

merger and derived their macro-average. Similarly, when evaluating out-of-domain performance, we computed the macro-average of their performance across the out-of-domain test set.

D.4 Merging Models Trained on Non-i.i.d. Partitions.

Merging models initially trained on non-i.i.d. partitions of the same dataset is started, which is achieved by simulating synthetic data splits across the 8 tasks within the GLUE benchmark. Each task involves dividing the training data into two partitions, each containing 1,000 training examples with distinct label distributions. Following this, we perform fine-tuning on these two partitions for 8 pairs of individual models and merge each pair of models. The evaluation of these merged models takes place on the official validation sets, which portray a joint distribution of both partitions.

D.5 Time Cost

We report the time cost of the scheme of model evolution, as shown in Table 8. T5-base model and MiniCPM model are tested on single NVIDIA A800 80G GPU and other models are tested on single RTX A6000 48G GPU. We find that all task of model evolution can be completed within half an hour, which is very cost-efficient in improving the model performance without further training.