MEDUSA: A Framework for Collaborative Development of Foundation Models with Automated Parameter Ownership Assignment

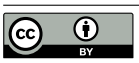
DEZHI RAN, Peking University, China
YUAN CAO, Peking University, China
YUZHE GUO, Beijing Jiaotong University, China
YUETONG LI, University of Chicago, USA
MENGZHOU WU, Peking University, China
SIMIN CHEN, University of Texas at Dallas, USA
WEI YANG*, University of Texas at Dallas, USA
TAO XIE*, Peking University, China

Foundation models (FMs) have become the backbone of intelligent systems. Collaborative development of FMs enables multiple teams to fine-tune different aspects of an FM simultaneously. However, conflicts in model updates across teams, particularly when modifying overlapping parameters, pose significant challenges to maintaining model performance. To address these challenges, in this paper, we propose MEDUSA, a novel framework designed to support collaborative FM development by managing model branches and introducing a structured system of parameter ownership. Medusa tracks fine-tuning efforts as separate branches, similar to Git, allowing developers to work on different tasks without destabilizing the base model. Instead of passively merging parameters from already fine-tuned models, MEDUSA proactively controls the merging process through our novel algorithm for assigning ownership of parameters by generating merging-aware masks to guide the fine-tuning process, ensuring that only specific branches can modify designated parameters. MEDUSA approximates the optimal assignment even as model complexity increases, ensuring scalability in large models. To investigate the efficacy of MEDUSA, we conduct extensive evaluations on five datasets and three models fine-tuned by three popular techniques, and compare our approach against six state-of-the-art approaches for post-training model merging. The evaluation results show that MEDUSA substantially and generally improves the effectiveness of collaborative model development, across different models, fine-tuning techniques, and datasets. Specifically, with automated parameter ownership assignment and masked fine-tuning, MEDUSA outperforms post-training model-merging approaches by improving model performance after merging by 3.19% absolute points. Ablation studies further demonstrate the efficacy of the algorithms in MEDUSA.

CCS Concepts: • **Software and its engineering → Software development methods**; **Collaboration in software development**; **Software version control**; **Software configuration management and version control systems**; • **Computing methodologies → Multi-task learning**.

*Tao Xie and Wei Yang are the corresponding authors.

Authors' Contact Information: Dezhi Ran, Key Lab of HCST (PKU), MOE; SCS, Peking University, Beijing, China, dezhiran@pku.edu.cn; Yuan Cao, Peking University, Beijing, China, cao_yuan21@stu.pku.edu.cn; Yuzhe Guo, Beijing Jiaotong University, Beijing, China, yuzhe.guo@bjtu.edu.cn; Yuetong Li, University of Chicago, Chicago, USA, yuetongli@uchicago.edu; Mengzhou Wu, Peking University, Beijing, China, wmz@stu.pku.edu.cn; Simin Chen, University of Texas at Dallas, Dallas, USA, sxc180080@utdallas.edu; Wei Yang, University of Texas at Dallas, Richardson, USA, wei.yang@utdallas.edu; Tao Xie, Key Lab of HCST (PKU), MOE; SCS, Peking University, Beijing, China, taoxie@pku.edu.cn.



ACM 2994-970X/2025/7-ARTFSE115
https://doi.org/10.1145/3729385

Additional Key Words and Phrases: Model Merging, Parameter-Efficient Fine-Tuning, Model Development Process, Large Language Models, Foundation Models, Ownership Assignment, Mixed Integer Programming

ACM Reference Format:
Dezhi Ran, Yuan Cao, Yuzhe Guo, Yuetong Li, Mengzhou Wu, Simin Chen, Wei Yang, and Tao Xie. 2025. MEDUSA: A Framework for Collaborative Development of Foundation Models with Automated Parameter Ownership Assignment. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE115 (July 2025), 24 pages. https://doi.org/10.1145/3729385

1 Introduction

As the core foundation to build numerous complex intelligent software systems such as machine translation [10, 55, 71, 90], embodied agents [38, 69, 74–76, 78, 95], and programming assistants [9, 11, 46, 80, 88, 108], *foundation models* [4, 24, 77, 89] (FMs) must have comprehensive capabilities (denoted as skills), which come from different training data as shown in Figure 1. Similar to developing any large-scale software [3, 36, 81, 102], in the development of FMs, a collaborative approach is essential [77]. The sheer scale of computational resources required to train these models with heterogeneous data [4, 89] makes it necessary for development to be a team effort, whether within proprietary teams [68, 89] or across contributors in the open-source community [99].

One of the most significant advantages of collaborative development for FMs is that it allows for incremental updates [14] and improvements in a decentralized way [51, 77, 83]. Instead of discarding older models and building entirely new ones, developers can update or enhance specific parts of a model in parallel with others' efforts. This decentralized approach means that multiple teams can simultaneously improve the model's different capabilities, reducing the need for extensive computational resources and centralized training processes [68, 89]. The model, as a result, can evolve gradually, incorporating new capabilities while retaining and improving upon existing ones. This approach of continuous refinement ensures that the model remains up-to-date without the inefficiencies associated with a full-scale redevelopment.

A key concept in collaborative development is to view tasks as compositions of different skills, rather than isolated functions. For example, Figure 1 shows the task of paraphrasing the sentence, "She walk to the market every day and buy some vegetables, she really like the fresh air." This task involves multiple subtasks: grammar correction (e.g., changing "walk" to "walks" and "like" to "likes"), entity relations (e.g., recognizing the link between "she" and "market"), logical reasoning (e.g., inferring that the reason she walks to the market is that she enjoys the fresh air), and common knowledge about daily activities. Instead of finetuning/retraining an FM from scratch to handle the entire paraphrasing task, different previously finetuned models specializing in these subtasks are already available—one for grammar correction, one for entity recognition, and another for logical reasoning. By combining these parallel efforts, a more advanced version of the FM can emerge, capable of performing complex, multi-skill tasks more effectively. Such collaborative development is in great need for building versatile and intelligent systems that address the complexity of real-world problems.

Despite being desirable, achieving such collaborative FM development can be challenging due to conflicts in model updates across different teams. When multiple teams simultaneously fine-tune different aspects of an FM, they may inadvertently modify overlapping parameters [35, 41, 42, 105], resulting in conflicts [105] that can degrade the model's performance [42, 105]. As shown in Figure 2, these conflicts typically arise when multiple branches update the same layers or parameters of the model, making manual resolution time-consuming and error-prone.

Collaborative FM development framework. To address the challenges and apply established software engineering principles [29, 62, 91] to FM development, we propose MEDUSA, the first framework specifically designed to support collaborative FM development by managing the *model*

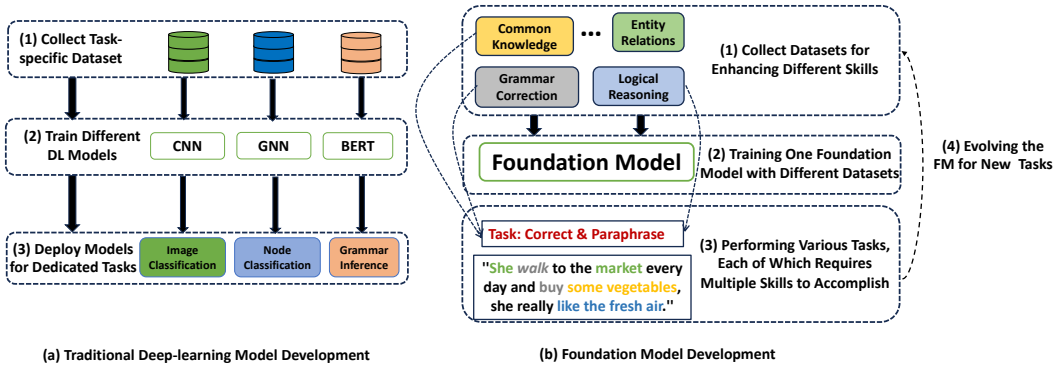


Fig. 1. (a) Traditional deep-learning models are task-specific, and traditional deep-learning model development is a one-shot effort. The developers first collect a dataset for a task, train a model (e.g., CNN, BERT, GNN, depending on specific tasks) from scratch, and deploy the model for the dedicated task. (b) Foundation Models (FMs) are expected to perform more complicated tasks with a single model (usually in the Transformer Architecture), which involves multiple perspectives (i.e., skills). To enhance different skills, FM development requires training the FM on different datasets dedicated to different skills.

branches with the assignment of *model parameter ownership*. Figure 3 presents the design and workflow of MEDUSA for collaborative model development across three stages: model branching, parameter ownership assignment, and collaborative merging for FM updates.

Model branching. MEDUSA tracks the model updates of different finetuning efforts as separate model branches, similar to Git branches [86] for collaborative software engineering. Model branches allow developers to improve different aspects of an FM simultaneously, with each branch focused on specific functional (e.g., domain-specific tasks) or non-functional (e.g., mitigating jailbreaking) requirements, datasets, or environments. This branching structure enables parallel collaboration across teams while keeping the base model stable. Changes made in a model branch can be selectively merged back into the base FM once they have been validated, enabling iterative and modular development across different branches. Such branching structure design allows specialized improvements or task-specific fine-tuning to occur independently, reducing the risk of regressions or unintended behavior in the base FM.

Parameter ownership assignment. To mitigate conflicts of simultaneous model parameter updates [105] across different branches, MEDUSA introduces a structured system of parameter ownership. Instead of allowing unrestricted modification across all branches, MEDUSA assigns exclusive ownership of specific subsets of parameters (e.g., neurons [49], attention heads [92]) to designated branches. This assignment ensures that only the branch with assigned ownership can modify its allocated parameters, preventing concurrent updates across multiple branches. For example, one branch may own the parameters in earlier layers of a neural network, focusing on improving low-level feature extraction, while another branch may manage parameters in later layers to enhance task-specific outputs. By clearly assigning which branches have authority over particular sections of the model, this approach reduces the likelihood of conflicts during collaborative development. If ownership must be shared or transferred, MEDUSA provides mechanisms for conflict resolution, such as weighted averaging [103, 113] and performance-based prioritization [40, 67].

Collaborative merging for FM updates. MEDUSA provides a collaborative merging process, based on the three-way merging algorithm [85, 86] used by version control systems, to integrate updates from multiple branches into a single FM. MEDUSA takes the base FM, the fine-tuned version from branch

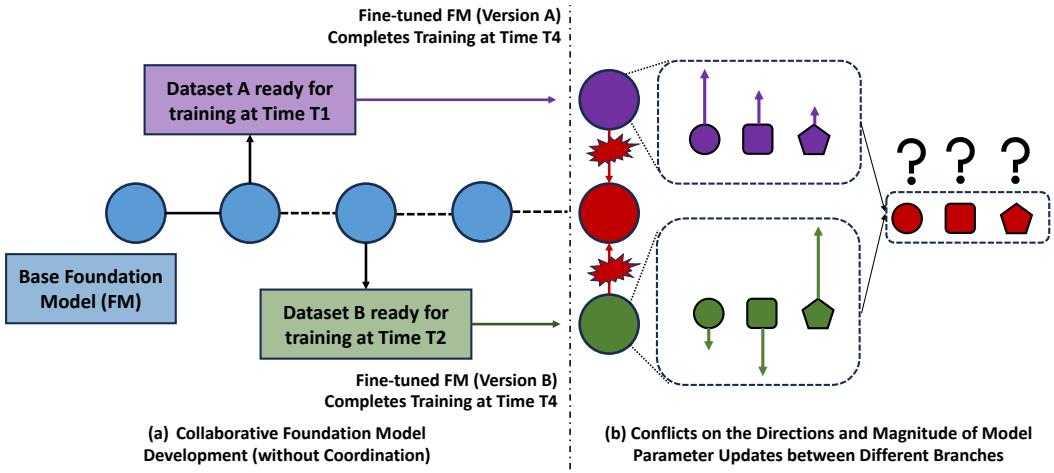


Fig. 2. (a) The datasets to enhance an FM's different skills usually expand time after time, and different datasets get ready for training the FM in an unpredictable order, making collaborative FM development a desirable solution. (b) Different datasets update model parameters in different directions, leading to conflicts between FMs trained on separate datasets and resulting in resource wastage or degraded model performance.

A, and the modified version from branch B as inputs. First, MEDUSA computes the parameter-wise differences between the base FM and the versions from both branches. For parameters that have been modified by both branches, MEDUSA identifies conflicts where the values differ. MEDUSA automatically handles the conflicts with weighted averaging algorithms [103, 113] to blend the updates from the two branches. Non-conflicting parameters are merged automatically into the final FM.

Novel algorithms for automated parameter ownership assignment. To optimize the workflow of MEDUSA, we propose a novel algorithm GPA for parameter ownership assignment. While the optimal assignment of model parameter ownership is NP-hard (formulated as a mixed-integer linear programming problem [101]), GPA provides an efficient, data-driven alternative. By leveraging the gradients generated during model training, the algorithm identifies which parameters are most sensitive to specific datasets or tasks. By analyzing the magnitude and direction of the gradients across different branches, GPA assigns ownership of parameters to branches based on their influence on task-specific performance, ensuring that each branch primarily controls the parameters most critical to its objectives. By using gradient analysis, MEDUSA can efficiently approximate the optimal parameter ownership assignment even for large, fine-tuned models handling diverse tasks without significant computational overhead.

To evaluate the effectiveness and generalization of MEDUSA, we conduct extensive evaluations against six state-of-the-art model merging approaches [40, 103, 105, 109, 113]. We fine-tune three popular pretrained models (T5-Base with 220 million parameters, T5-Large with 770 million parameters, and T0-3B with 3 billion parameters) on five standard NLP tasks [16, 19, 65]. To evaluate the generalization of MEDUSA across fine-tuning techniques, we fine-tune models with both parameter-efficient fine-tuning (i.e., LoRA [37] and IA3 [53]) and full fine-tuning techniques. To investigate the flexibility and adaptability of MEDUSA, we conduct experiments in full and partial collaborative model development settings, where either all developers or only a subset use MEDUSA while the others opt out due to privacy and security concerns [23, 44].

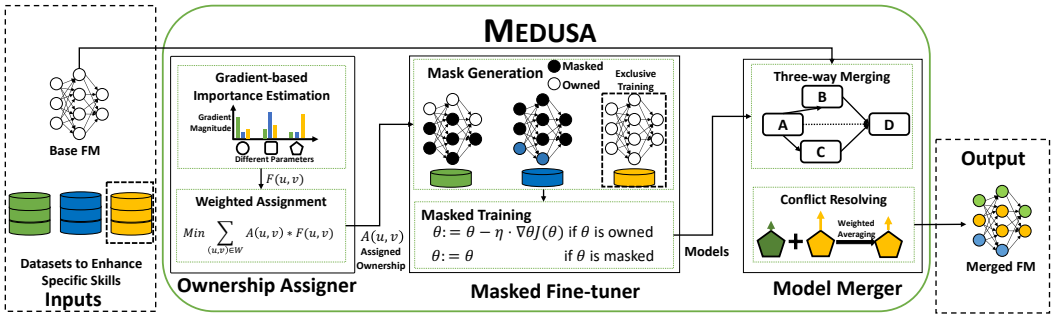


Fig. 3. Inspired by the established collaborative software development process, we propose MEDUSA to support collaborative FM development on (1) model branching to track collaborators' modification simultaneously, and (2) model parameter ownership to avoid parameter update redundancy and conflicts. MEDUSA (1) controls the model parameter ownership in each model branch, (2) assigns masks to freeze the gradient descent and value updates of parameters not owned by each branch, and (3) merges the models developed in different branches to obtain a merged FM, enhanced by different skills without parameter update conflicts.

Findings. Our extensive evaluations reveal three major findings. First, adopting MEDUSA for collaborative FM development substantially improves merged models' performance, even when not all branches adopt MEDUSA. With the awareness of controlling the fine-tuning process, MEDUSA substantially outperforms state-of-the-art model merging approaches, improving the absolute performance of merged models on average by 8.66% if all model developers use MEDUSA and by 2.83% if one developer uses MEDUSA while the others do not. However, the merged models' performance improves as a higher percentage of branches adopt MEDUSA for model development. Second, the mergeability of models seems to be primarily influenced by the number of trainable parameters but not relevant to fine-tuning techniques. As long as the number of trainable parameters is sufficiently large, MEDUSA improves the performance of merged models across all fine-tuning techniques, ranging from full fine-tuning to parameter-efficient fine-tuning. Third, there exist non-mergeable scenarios where the parameter updates from different branches conflict heavily, making merging an undesirable solution, further emphasizing the importance of pre-development scheduling instead of post-development conflict resolution.

In summary, this paper makes the following main contributions:

- **A collaborative development framework of FMs.** We design and implement a framework named MEDUSA to support the collaborative development of FMs, inspired by the established software engineering processes. The implementation of MEDUSA is publicly available [73].
- **Novel algorithms for automated parameter ownership assignment.** We design novel algorithms to automate and optimize the assignment of model parameter ownership in MEDUSA to minimize conflicts between different model branches.
- **Empirical findings.** We conduct extensive evaluations on five datasets with three popular models and three fine-tuning techniques to demonstrate the effectiveness and generalization of MEDUSA, opening up a new research direction on deriving engineering principles for FM development.

2 Background

In this section, we begin by introducing the basics of modern practices in FM development. We then formulate the problem of collaborative FM development.

2.1 FM Development

Modern FMs [4] are typically trained on large, unlabeled datasets [7]. When applied to domain-specific downstream tasks [30, 60, 72, 104] or enhancing specific aspects of FMs [68], a common approach is to further fine-tune [21, 72] these models using labeled datasets. For example, OpenAI's GPT-3.5, one of the most widely used FMs, allows developers to fine-tune it on their private datasets via API access [66], enabling the creation of more specialized and task-specific models.

However, despite advancements in parameter-efficient fine-tuning techniques [37, 53], the process remains challenging. Fine-tuning an FM can be computationally expensive [54], requiring significant storage and processing resources. Moreover, fine-tuned models often struggle to integrate new information dynamically [4], limiting their ability to continuously improve performance without retraining from scratch.

To meet the need to update FMs with newly acquired knowledge, such as dynamically obtained labeled datasets, a common approach is to fine-tune a model based on new data [14]. However, this approach often results in the creation of unnecessary model duplicates [105] and can be impractical due to data privacy concerns [23, 44], as the process may require sharing sensitive information or storing multiple versions of the model.

2.2 Collaborative FM Development

To address the aforementioned limitations, we draw inspiration from traditional software engineering practices [29, 62, 91]. Specifically, we propose a *collaborative FM development framework*, analogous to the traditional collaborative software engineering [85, 86]. We first introduce and define key components related to our proposed framework for FM development and then describe the advantages of our framework.

Key components of our framework. Our framework includes three main concepts: *base model*, *model branch*, and *model merging*.

Definition 2.1 (Base model). A base model B refers to a pretrained FM that has been trained on an unlabeled dataset but has not yet been fine-tuned on any labeled dataset for specific tasks to enhance certain skills.

Definition 2.2 (Model branch). A model branch b is denoted as a tuple (B, δ), where B represents the base model that has been fine-tuned for a specific downstream task, and δ denotes the task-specific parameters.

Definition 2.3 (Model merging). The model merging process can be represented as a function, $O(b_1, b_2)$, which takes two model branches as input and outputs a new, merged model branch. The model in the resulting merged branch should at least preserve the performance of the original models on their respective tasks or exhibit improved generalization across a broader range of tasks or domains.

Advantages of our collaborative FM development framework. Our proposed collaborative FM development framework provides an efficient way for developers to work in parallel while ensuring data integrity and minimizing conflicts in model parameter updates. This approach facilitates collaboration even when different teams are working with private datasets.

First, the framework ensures data integrity throughout the development process. For example, if Team A and Team B are working on an FM but have private datasets that they cannot share, they can still collaborate without compromising data privacy. Each team develops its own branch of the model, fine-tuning it using their private dataset. The key to avoiding conflicts lies in the structured system of *parameter ownership*, where each branch has exclusive control over its designated model parameters. By preventing simultaneous modifications to the same parameters, this approach

guarantees that each team's contributions are integrated smoothly. As a result, the branches can be merged into a unified model that effectively handles tasks from both teams, while preserving the integrity and privacy of their individual datasets.

Second, our framework significantly enhances development efficiency through its structured parameter ownership and model merging capabilities. By assigning control of specific parameters to particular branches, we eliminate conflicts that often arise from concurrent parameter updates. This structure not only reduces development overhead but also allows each branch to specialize in different aspects of the model, such as mitigating jailbreaking attacks or focusing on task-specific outputs. This specialization leads to faster convergence and improved model performance. Furthermore, the model merging technique minimizes computational and storage costs by avoiding the need to train multiple models independently. Instead, the teams can store and maintain only the merged model, streamlining the development process and reducing overall resource usage.

Finally, parallel development is another key advantage of our framework. Often, developers may need to work on different tasks using datasets that are not available at the same time. Without our framework, they would have to either delay development until all datasets are accessible or retrain the model once new data is available. Our framework, however, allows the developers to work independently on separate branches of the model. Each branch focuses on its own dataset, with the parameter ownership structure ensuring that updates remain isolated and do not conflict with one another. This parallelized approach not only accelerates the overall development timeline but also ensures that when the datasets are finally combined, the model can be efficiently updated and merged without the need for extensive retraining.

3 Design of MEDUSA

In this section, we present the design of MEDUSA, a framework to support collaborative FM development with automated parameter ownership assignment, masked fine-tuning, and model merging.

3.1 Key Challenge and Our Idea

Note that the values of branch-specific parameters in two different model branches are often quite distinct [105]. Determining the parameter values for the merged branch thus presents a challenge.

To address this challenge, we propose assigning the ownership of specific parameters to individual branches, meaning that only the designated branch can modify those parameters. Our design minimizes conflicts and simplifies the merging process by construction. Specifically, we introduce the following concept within the context of FM development.

Definition 3.1 (Parameter Ownership). A parameter p_i in the base model B is owned by a model branch b if B_i can be modified exclusively in branch b during fine-tuning.

Formally, let θ be the set of parameters in the base model, where each $p \in \theta$ represents a trainable parameter. Let $\mathcal{B} = \{b_1, b_2, \ldots, b_k\}$ denote the set of model branches. Let a binary mask $M_i^p \in \{0, 1\}$ represent the assignment of parameter ownership, where

$$M_i^p = \begin{cases} 1 & \text{if parameter } p \text{ is owned by branch } b_i \\ 0 & \text{otherwise} \end{cases}$$

By adopting a parameter ownership model, we ensure that branches working on distinct datasets or tasks modify only the parameters that they own. This structured approach minimizes conflicts during model branch merging and preserves the integrity of the base FM as development progresses across multiple tasks.

Algorithm 1 Ownership Assignment with Gradient Estimation

Input: a base model B, tasks $\mathcal{T} = \{T_1, T_2, .., T_n\}$ to be fine-tuned on, boolean value set $\Gamma = \{\gamma_1, \gamma_2, .., \gamma_n | \gamma_i \in \{0, 1\}\}$ indicating whether we can assign parameter ownership in each branch.
Output: ownership assignment $\{A_p(p \in \theta_B)\}$.

▷ Step 1. Estimate parameters' importance.
for each $i \in [1, n]$ **do**
  $B' = train\_one\_epoch(B, T_i)$
  $F_i = keep\_topk\_rest\_to\_zero(\theta_{B'} - \theta_B)$
▷ Step 2. Assign parameters to branches.
for each $p \in \theta_B$ **do**
▷ Step 2.1. Determine the main direction.
  $\rho = sgn(\Sigma_{i=1}^{n}[\neg\gamma_i]F_i^p)$
▷ Step 2.2. Distribute parameters among assignable branches.
  for each $i \in [1, n]$ **do**
    if γ_i **and**$(\rho == 0$ **or** $sgn(F_i^p) == \rho)$ **then**
      $proportion_i = |F_i^p|$
    else
      $proportion_i = 0$
  $A_p = randomly\_select\_one\_proportional\_to(proportion_{1,...,n})$

3.2 Overview of Medusa

Figure 3 presents the design of Medusa, consisting of three major components: *Ownership assigner*, *Masked fine-tuner*, and *Model merger*.

Ownership assigner determines parameter ownership by analyzing the magnitude of parameter changes in each model branch during the fine-tuning process, based on gradient information.

Masked fine-tuner selectively updates only the parameters owned by the current branch during fine-tuning. By masking parameters assigned to other branches, Medusa ensures that each branch modifies only its designated parameters.

Model merger combines different model branches using a three-way merging algorithm [86]. By comparing the base model with the updated parameters from each branch, Medusa integrates changes while resolving conflicts. This approach ensures that only parameters modified according to their assigned ownership are incorporated.

3.3 Workflow of Medusa

Input and output of Medusa. As shown in Figure 3, the input of Medusa is a tuple $(B, \mathcal{T})$, consisting of a base FM (denoted as *base model B*) and a set of datasets $\mathcal{T} = \{T_1, T_2, .., T_n\}$ to fine-tune B on. We denote $\mathcal{B} = \{b_1, ..., b_n\}$ as the model branches, with branch b_i fine-tuning B on dataset T_i. The output of Medusa is a merged model B_{merge} that integrates the fine-tuning efforts from all branches in $\mathcal{B}$.

Parameter ownership assignment (Section 3.4). The ownership assigner of Medusa assigns the ownership of parameters in the base model B to different model branches. The assignment is based on gradient-based importance estimation, which aims to ensure that each model branch b_i owns and modifies only the parameters that contribute the most to its respective dataset T_i. In practice, some model branches may not want to reveal their datasets due to privacy and security concerns [23, 44]. Medusa also supports managing branches with restricted parameter ownership

Algorithm 2 Masked Fine-Tuning with Assigned Parameters

Input: a model branch b_i, boolean value γ_i indicating whether a branch is controllable, the base model B, the dataset T_i for fine-tuning, a learning rate η, and the ownership assignment $\{A_p(p \in \theta_B)\}$ designated in Algorithm 1.

Output: Updated model parameters θ_{b_i} for the branch b_i under ownership assignment $\{A_p(p \in \theta_B)\}$.

▷ Step 1. Generate mask M_i under ownership assignment $\{A_p(p \in \theta_B)\}$.

for each $p \in \theta_B$ **do**
  if γ_i **then**
    $M_i^p = [i == A_p]$
  else
    $M_i^p = 1$

▷ Step 2. For the model branch b_i, update only parameters in the mask M_i.

$\nabla J_i(\theta_B) = gradient\_calculation(B, T_i)$

$\theta_{b_i} = \theta_B - M_i \cdot \eta \cdot \nabla J_i(\theta_B)_p$

control during the fine-tuning process. In such cases, MEDUSA reduces conflicts by modifying only the parameter assignments of controllable branches.

Masked fine-tuning (Section 3.5). After the parameter ownership is assigned, the masked fine-tuner ensures that only the parameters assigned to each branch are updated during the fine-tuning process, while the other parameters remain frozen. The masked fine-tuning optimizes the model for the specific tasks represented by datasets $\mathcal{T}$ while preventing conflicts with other branches. For branches whose parameters are not assignable, MEDUSA does not interfere with their fine-tuning processes.

Model merging (Section 3.6). After all model branches have completed the masked fine-tuning, the model merger of MEDUSA combines the branches into a merged model B_{merge}. Ideally, when MEDUSA controls the parameter ownership in all branches, the updates of parameters experience no conflicts, allowing the merging process to simply combine parameter updates from different branches. When branches with restricted ownership control are managed, conflicts of parameter updates may happen. MEDUSA employs strategies such as weighted averaging [105] to resolve conflicting parameter values, ensuring that B_{merge} effectively integrates the specialized knowledge from each fine-tuned branch.

3.4 Parameter Ownership Assignment with Gradient Estimation

To minimize conflicts of parameter updates in different model branches, MEDUSA assigns each parameter in the base model B to a specific model branch, as illustrated in Algorithm 1. An optimal parameter ownership assignment should (1) minimize the parameter update conflicts, i.e., dedicate each parameter p in the base model B's parameters (θ_B) to only one model branch if possible and (2) minimize the impact of the ownership assignment for the fine-tuning effectiveness. To achieve the aforementioned optimization objectives, MEDUSA assigns each parameter p to the model branch that exhibits the largest change in p after fine-tuning.

However, obtaining the precise parameter change in each model branch is challenging since it can be obtained only after fine-tuning. To avoid the costs of re-training models, we utilize the gradient information to estimate the parameter updates in each model branch. By running a single epoch on each dataset, MEDUSA obtains the gradients and uses the direction and magnitude as an approximation of the direction and magnitude for the final parameter update in the model branch.

Algorithm 3 Merging Model Branches into a Single Model

Input: base model B and fine-tuned model branches $\mathcal{B} = \{b_1, ..., b_n\}$.
Output: Merged model branch b_{merged} with its parameter $\theta_{b_{merged}}$.

$\theta_{b_{merge}} = \theta_B$
for each $(p, p_1, ..., p_n, p_{merged}) \in \theta_B \times \theta_{b_1} \times ...\theta_{b_n} \times \theta_{b_{merged}}$ **do**
▷ Averaging models' changed parameters together after selecting the final sign.
    $\rho = sgn(\Sigma_{i=1}^{n}(p_i - p))$
    $list_p = []$
    for each $i \in [1...n]$ **do**
        if $sgn(p_i - p) == \rho$ **then**
            $list_p.append(p_i)$
    $p_{merged} = Average(list_p)$

Once the direction and magnitude of the parameter update are estimated, the next step is to solve the assignment of parameters to different model branches. When MEDUSA has full parameter ownership control, MEDUSA uses a randomized technique of parameter ownership assignment with probability proportional to the estimated gradient magnitude of every branch and eliminating all parameter-level conflicts. For branch b_i, the probability of assigning the parameter p to b_i is proportional to the estimated gradient change magnitude of p for b_i.

In some cases where MEDUSA cannot control the assignment of specific model branches due to privacy and security concerns [23, 44], MEDUSA first determines a direction for each parameter based on the sum of estimated gradients from uncontrollable branches. MEDUSA then assigns parameters among the assignable branches with the same gradient direction as this sum, with probability proportional to their magnitude.

3.5 Fine-Tuning with Conflict-Avoiding Masks

Once the parameter ownership is assigned, the fine-tuner of MEDUSA generates masks for model branches and uses these masks to control the fine-tuning process of each model branch b_i. As illustrated in Algorithm 2, during the masked fine-tuning phase for model branch b_i, the fine-tuner in MEDUSA first generates a mask M_i based on the parameter ownership assignment. The mask ensures that only the parameters specified in the mask M_i are updated during fine-tuning, while parameters not included in M_i are frozen and not allowed to change in model branch b_i. With these masks, MEDUSA prevents conflicting parameter updates between model branches. For model branches where MEDUSA cannot control parameter ownership (as discussed in Section 3.4), MEDUSA simply assigns a full mask and fine-tunes them without freezing parameters.

3.6 Model Merging with Weighted Averaging

After the development of each model branch is completed, the n branches in $\mathcal{B}$ are merged into a single model using a three-way merge algorithm [86], applied element-wise, as shown in Algorithm 3. When MEDUSA has full parameter ownership control of all model branches, there is no conflict in parameter updates. In this case, we can simply add the parameter updates together as follows:

$$\theta_{b_{merge}} = \theta_B + \Sigma_{i=1}^{n}(\theta_{b_i} - \theta_B)$$

However, in situations where MEDUSA cannot control the fine-tuning process of all branches, parameter update conflicts may occur. MEDUSA resolves these conflicts by first identifying the dominant update direction across branches, and then applying weighted averaging of updates that align with this direction. As shown in Algorithm 3, parameter updates aligned with the dominant

direction are weighted according to their magnitude, averaged, and taken as the merged update. This approach ensures that conflicting updates are resolved in favor of the strongest learning signal while preserving knowledge from multiple branches.

4 Evaluations

To evaluate the effectiveness of MEDUSA, we conduct extensive evaluations to investigate the following three research questions:

- **RQ1:** How effective and general is MEDUSA in helping collaborative FM development in terms of improving the merged model performance given full parameter ownership control?
- **RQ2:** How effective is MEDUSA in helping FM development when MEDUSA cannot control parameter ownership on all model branches?
- **RQ3:** What is the scope of MEDUSA in facilitating collaborative FM development in terms of branch numbers, model scale, and branch conflicts?

To answer RQ1, we analyze the performance of models merged by MEDUSA and state-of-the-art post-training model merging approaches [40, 103, 105, 109, 113] under three popular fine-tuning techniques [37], with different models parameters and different task (i.e., model branch) combinations.

To answer RQ2, we experiment with the setting where MEDUSA can mask the fine-tuning process in only one model branch, i.e., does not update certain parameters in this model branch, while not interfering with the fine-tuning process of another model branch due to privacy and security concerns [23, 44].

To answer RQ3, we conduct experiments and ablation studies to investigate the efficacy and application scope of MEDUSA concerning the number of model branches, the efficacy of gradient estimation for optimizing the parameter ownership assignment, the impact of masked fine-tuning for model performance, and the application scope of model merging concerning the number of model parameters and task combinations.

4.1 Evaluation Setup

Base pre-trained models. Following previous work on model merging [105, 109], we adopt Text-to-Text Transfer Transformer [30, 60, 72, 104] (T5) and T0 [18, 82], two highly popular models fine-tuned by over thousands of developers on HuggingFace [99], as our base pre-trained FMs. T5 [60] employs an encoder-decoder architecture pre-trained on a large corpus of text [60, 72, 104], making it versatile to be fine-tuned for various natural language processing (NLP) applications such as translation [2, 92], summarization [50, 84], and question-answering [39, 61]. T5 contains a series of models, among which T5-base (220M) and T5-large (770M) are the most popular models for academic use. With the same encoder-decoder architecture as T5 and pre-trained on a larger set of NLP tasks, T0 [82] outperforms GPT-3 while being 16x smaller. We use T0-3B (the T0 model with 3 billion parameters) in our evaluation due to the limits of computation resources.

Fine-tuning techniques. We use three standard and popular fine-tuning techniques [37, 53, 107] to evaluate the effectiveness and generalization of MEDUSA in helping different FM development processes.

- **Fully Fine-Tuning (FFT)** [43, 79, 107] involves updating all the parameters of a pre-trained model when adapting it to a specific task.
- **Low Rank Adaptation (LoRA)** [37, 112] introduces low-rank matrices into the layers of the model, allowing only these matrices to be fine-tuned while keeping the original model parameters frozen. LoRA significantly reduces the number of trainable parameters and is one of the most popular fine-tuning techniques.

- **IA3** [53] is another popular parameter-efficient fine-tuning technique designed to modify only a small fraction of the model's layers, specifically targeting the attention [92] and activation components [49], significantly reducing the number of trainable parameters while preserving the model's generalization capabilities.

Datasets. We focus our attention on GLUE [93, 94], a group of classic NLP tasks. GLUE is a widely used benchmark designed to evaluate models' generalization abilities across a variety of NLP tasks such as grammaticality, semantic similarity, sentiment analysis, and natural language inference. We use five tasks in GLUE to evaluate the generalization performance of MEDUSA on a range of well-established NLP challenges. Specifically, CoLA tests the model's ability to classify whether a sentence is grammatically acceptable, SST2 focuses on binary sentiment classification of movie reviews, MNLI and RTE evaluate natural language inference by asking models to predict whether a given premise entails, contradicts, or is neutral with respect to a hypothesis, QNLI is a question-answering task reformulated as sentence-pair classification, and WNLI is a common-sense reasoning task.

Baseline approaches. We compare MEDUSA with six state-of-the-art model merging approaches [40, 103, 105, 109, 113] from the deep learning community. These approaches have been developed to merge already fine-tuned models without altering the fine-tuning process.

- **Linear Averaging** [103, 113] is a simple yet effective approach where the parameters of multiple fine-tuned models are averaged element-wise. This approach assumes that by averaging the weights of fine-tuned models, the resulting model can capture useful features from each model while minimizing the risk of overfitting.
- **Task Arithmetic** [15, 40, 67] leverages task-specific weight vectors, where model weights fine-tuned for different tasks can be added or subtracted. By interpreting task fine-tuning as vector transformations in the model parameter space, Task Arithmetic enables the merging of multiple task-specific models.
- **TIES** [105] tries to resolve parameter conflicts when merging models in addition to averaging parameter updates. The approach trims redundant parameters and elects the most influential sign when there are conflicting weight updates from different models.
- **DARE** [109] proposes a drop-and-rescale technique to improve the merging of fine-tuned models by sparsifying the differences between pre-trained and fine-tuned model parameters.

Among all the baselines, DARE can be integrated with other model-merging approaches. Therefore, we have six baseline approaches in total, namely Linear Average, Task Arithmetic, TIES, DARE+TIES, DARE+Linear Average, and DARE+Task Arithmetic. In contrast to existing approaches, MEDUSA is the first to introduce parameter ownership and merging-aware masks, which are used to alter the fine-tuning process. This proactive approach enables MEDUSA to control the merging process in advance, improving mergeability and ensuring better overall performance.

Experiment platform. All experiments are conducted on an AI server with two Intel Xeon Platinum 8374C 36-core processors, four NVIDIA GeForce RTX 3090 Graphics Cards with 24G memory, CUDA 12.0, and NVLink [25] enabled for training and merging models.

Fine-tuning settings. For fully fine-tuning of T5-base and T5-large, we use Adam optimizer [47] for 1 to 20 epochs on different tasks varied on the size of their dataset with a learning rate of 3e-4, following previous work [105]. We fine-tune models with a batch size of 256 with gradient accumulation [70] to fit our limited GPU resources. During the fine-tuning process, we use bf16, i.e., brain floating point [8], to accelerate the fine-tuning [33]. For fine-tuning T0-3B with LoRA and IA3, we set the learning rate be 1e-4 following previous work [37, 53, 105]. The Adam optimizer is used with a batch size of 16 and gradient accumulation, and the T0-3B models are trained for 2 to 32 epochs across various tasks to fit the GPU resource limits.

Table 1. Effectiveness of MEDUSA and baseline approaches on merging model branches developed from T5-base with fully fine-tuning. Medusa has full control of parameter ownership on both model branches.

Approach	COLA / RTE	COLA / SST2	COLA / QNLI	COLA / WNLI	RTE / SST2	RTE / QNLI	RTE / WNLI	SST2 / QNLI	SST2 / WNLI	QNLI / WNLI	Average
Linear Average	69.1 / 56.3	69.1 / 53.8	69.1 / 50.1	69.1 / 39.4	51.3 / 52.5	66.4 / 80.2	49.8 / 45.1	56.3 / 49.3	60.8 / 43.7	74.4 / 43.7	57.48
DARE+Linear Average	69.1 / 53.8	69.1 / 52.3	69.1 / 51.1	69.1 / 42.3	52.3 / 68.3	66.1 / 50.3	47.3 / 43.7	60.7 / 49.7	59.6 / 39.4	65.8 / 56.3	56.78
Task Arithmetic	69.1 / 51.3	69.1 / 49.4	69.1 / 48.9	69.1 / 45.1	51.6 / 49.2	58.1 / 73.8	65.0 / 43.7	49.1 / 50.1	50.3 / 45.1	50.5 / 43.7	55.07
DARE+Task Arithmetic	69.1 / 54.9	69.1 / 50.3	69.1 / 47.5	69.1 / 40.8	50.5 / 52.8	53.8 / 62.1	58.8 / 43.7	49.1 / 51.3	52.2 / 43.7	49.5 / 43.7	54.05
TIES	71.7 / 58.8	69.1 / 53.6	69.5 / 77.3	74.6 / 47.9	62.5 / 86.4	67.9 / **82.1**	**70.4** / 43.7	87.4 / 67.4	88.3 / **49.3**	**81.5** / 43.7	67.65
DARE+TIES	69.6 / 54.2	69.1 / **85.3**	**78.2** / **80.7**	**78.0** / 40.8	52.7 / 89.9	47.3 / 66.8	47.3 / **56.3**	**90.0** / 57.1	90.5 / 43.7	50.6 / 42.3	64.52
MEDUSA (Ours)	**71.9** / **67.9**	**74.8** / 84.6	76.2 / 79.5	77.9 / **43.7**	**65.0** / **90.6**	**71.8** / 52.7	47.3 / **56.3**	89.1 / **86.3**	**91.9** / 42.3	57.1 / **56.3**	**69.15**

Table 2. Effectiveness of MEDUSA and baseline approaches on merging model branches developed from T5-large with fully fine-tuning. Medusa has full control of parameter ownership on both model branches.

Approach	COLA / RTE	COLA / SST2	COLA / QNLI	COLA / WNLI	RTE / SST2	RTE / QNLI	RTE / WNLI	SST2 / QNLI	SST2 / WNLI	QNLI / WNLI	Average
Linear Average	69.2 / 56.3	36.6 / 53.8	51.5 / 50.1	32.6 / 39.4	45.5 / 52.5	**75.5** / **80.2**	**52.7** / 45.1	49.1 / 49.3	49.1 / 43.7	49.6 / 43.7	51.28
DARE+Linear Average	69.1 / 54.2	69.1 / 49.1	61.7 / 46.3	42.8 / 45.1	46.6 / 49.1	47.3 / 49.5	**52.7** / 43.7	49.1 / 51.2	49.1 / 43.7	49.5 / 43.7	50.61
Task Arithmetic	31.3 / 51.3	30.9 / 49.4	30.9 / 48.9	30.9 / 45.1	45.1 / 49.2	53.8 / 73.8	52.3 / 43.7	49.1 / 50.1	49.1 / 45.1	49.6 / 43.7	46.15
DARE+Task Arithmetic	30.9 / 45.1	30.9 / 49.1	69.1 / **51.9**	30.9 / **49.3**	56.0 / 49.1	47.3 / 49.6	51.3 / 43.7	49.1 / 49.1	49.1 / **54.9**	49.6 / 43.7	47.47
TIES	71.4 / **58.8**	69.2 / 53.6	74.5 / 49.6	73.0 / 47.9	52.7 / 86.4	53.4 / 49.5	**52.7** / 43.7	50.9 / **67.4**	49.7 / 49.3	**60.7** / 43.7	57.91
DARE+TIES	77.1 / 52.7	**76.8** / 49.2	**80.2** / 49.5	**78.7** / 43.7	47.3 / 50.9	47.3 / 49.5	47.3 / 43.7	49.1 / 49.5	49.1 / 43.7	49.5 / 43.7	53.91
MEDUSA (Ours)	**78.7** / 51.6	69.1 / **93.9**	69.1 / 49.7	69.0 / 43.7	**73.3** / **95.4**	35.0 / 43.0	49.8 / **56.3**	**94.7** / 49.4	**95.3** / 43.7	49.3 / **43.7**	**62.69**

Table 3. Parameter update conflicts without parameter ownership assignment.

Model	COLA / RTE	COLA / SST2	COLA / QNLI	COLA / WNLI	RTE / SST2	RTE / QNLI	RTE / WNLI	SST2 / QNLI	SST2 / WNLI	QNLI / WNLI
T5-base	42.87 %	41.19 %	42.74 %	42.21 %	43.11 %	36.67 %	32.51 %	43.36 %	42.42 %	34.80 %
T5-large	47.62 %	45.89 %	47.65 %	47.06 %	47.83 %	41.75 %	40.07 %	47.96 %	47.15 %	42.12 %

Table 4. Task similarity measured by the model's performance fine-tuned on one task and tested on another.

Eval \ Train	COLA	RTE	SST2	QNLI	WNLI
COLA	78.72	52.71	49.08	49.37	43.66
RTE	69.13	70.40	49.08	54.68	46.48
SST2	68.74	52.71	91.97	49.46	43.66
QNLI	69.13	57.76	49.08	89.53	43.66
WNLI	69.13	47.29	49.08	50.54	57.75

4.2 RQ1: Effectiveness with Full Parameter Ownership Control

4.2.1 Main Results on Model Development with Fully Fine-Tuning. Table 1 and Table 2 present the effectiveness of MEDUSA and baseline approaches for collaborative FM development with two model branches, of which MEDUSA can fully control the parameter ownership. These experiments take T5-base and T5-large as base models, respectively, with the fine-tuning technique being fully fine-tuning.

From Table 1 and Table 2, we find that by assigning parameter ownership to different model branches, MEDUSA reduces the parameter update conflicts between model branches by construction, outperforming baseline approaches on average by 1.50% absolute improvement for T5-base and 4.78% absolute improvement for T5-large, respectively.

To investigate whether the improvement really comes from the core contribution of MEDUSA, i.e., reducing update conflicts with parameter ownership assignment, we conduct two additional analyses. First, we investigate the proportion of parameter update conflicts. As shown in Table 3, most task pairs exhibit over 40% parameter update conflict between different model branches. Note that for MEDUSA, the conflicts of model parameter update are zero since MEDUSA assigns different parameters to different model branches. Consequently, we conclude that resolving conflicts generally helps improve the performance of merged models. Among all baseline approaches, TIES outperforms other baseline approaches since TIES tries to resolve conflicts with predefined heuristics such as ignoring minor parameter updates, demonstrating the importance of reducing

Table 5. Effectiveness of MEDUSA and baseline approaches on merging model branches developed from T0-3B with LoRA. MEDUSA has full control of parameter ownership on both model branches.

Approach	COLA / RTE	COLA / SST2	COLA / QNLI	COLA / WNLI	RTE / SST2	RTE / QNLI	RTE / WNLI	SST2 / QNLI	SST2 / WNLI	QNLI / WNLI	Average
Linear Average	61.0 / 52.7	58.1 / 92.9	60.0 / 50.5	61.3 / 45.1	52.7 / 93.2	52.7 / 52.3	52.7 / 43.7	93.1 / 49.6	93.1 / 43.7	55.0 / 45.1	60.42
DARE+Linear Average	58.1 / 52.7	58.4 / 92.4	59.2 / 49.7	58.4 / 43.7	52.7 / 93.0	52.7 / 50.3	52.7 / 43.7	92.7 / 49.6	93.0 / 43.7	51.3 / 45.1	59.65
Task Arithmetic	57.4 / 52.7	56.6 / 92.2	57.3 / 49.7	57.0 / 43.7	52.7 / 92.4	52.7 / 49.7	52.7 / 43.7	92.0 / 49.6	92.4 / 43.7	49.7 / 43.7	59.08
DARE+Task Arithmetic	57.5 / 52.7	56.4 / 92.2	56.5 / 49.7	57.5 / 43.7	52.7 / 92.1	52.7 / 49.7	52.7 / 43.7	92.0 / 49.6	92.0 / 43.7	49.7 / 43.7	59.01
TIES	64.8 / 52.7	58.4 / 93.5	60.7 / 54.1	62.9 / 43.7	52.7 / 93.9	52.7 / 58.1	53.4 / 43.7	92.8 / 50.3	93.9 / 43.7	63.3 / 47.9	61.86
DARE+TIES	**77.6** / 59.2	**73.5** / 95.2	**76.2** / 81.3	**77.4** / **54.9**	**69.0** / **96.7**	49.1 / 90.2	68.6 / 62.0	94.7 / **93.4**	**96.4** / **56.3**	85.7 / 57.7	75.76
MEDUSA (Ours)	77.1 / **64.3**	73.2 / **95.8**	74.9 / **94.7**	76.4 / 50.7	66.4 / 96.1	**62.8** / **94.5**	**74.4** / **67.6**	**95.3** / 89.6	96.1 / 49.3	**94.2** / **59.2**	**77.62**

Table 6. Effectiveness of MEDUSA and baseline approaches on merging model branches developed from T0-3B with IA3. MEDUSA has full control of parameter ownership on both model branches.

Approach	COLA / RTE	COLA / SST2	COLA / QNLI	COLA / WNLI	RTE / SST2	RTE / QNLI	RTE / WNLI	SST2 / QNLI	SST2 / WNLI	QNLI / WNLI	Average
Linear Average	61.1 / 53.4	61.7 / 93.6	60.8 / 50.0	61.1 / 43.7	52.7 / 95.0	54.9 / 76.1	52.7 / 46.5	94.4 / 49.7	95.0 / 43.7	58.8 / 43.7	62.41
DARE+Linear Average	63.9 / 55.2	61.4 / 94.7	61.8 / 50.0	61.2 / 43.7	52.7 / 95.1	**61.0** / 75.3	54.5 / 43.7	94.8 / 49.7	95.5 / 43.7	59.3 / 43.7	63.04
Task Arithmetic	57.9 / 52.7	59.5 / 93.0	57.6 / 49.6	58.4 / 43.7	52.7 / 94.0	53.1 / 55.5	52.7 / 43.7	93.1 / 49.5	93.9 / 43.7	50.1 / 43.7	59.91
DARE+Task Arithmetic	59.9 / 52.7	59.7 / 93.7	58.9 / 49.6	58.5 / 43.7	52.7 / 94.5	54.2 / 54.8	53.1 / 45.1	94.0 / 49.6	94.4 / 43.7	50.1 / 43.7	60.32
TIES	66.0 / 54.5	61.6 / 94.0	65.7 / 50.6	67.9 / 43.7	52.3 / 95.6	57.8 / **85.1**	**56.3** / 50.7	95.2 / 50.0	95.4 / 43.7	79.8 / 45.1	65.55
DARE+TIES	71.6 / **58.1**	68.9 / **96.1**	**75.6** / 82.4	73.6 / 45.1	**69.7** / 96.1	47.3 / 50.7	47.3 / 56.3	95.6 / 56.7	95.6 / 43.7	61.0 / 42.3	66.69
MEDUSA(Ours)	**73.5** / 49.1	**72.5** / 95.8	75.5 / **92.8**	**74.9** / **54.9**	58.1 / **96.4**	48.4 / 53.6	55.2 / **56.3**	**96.4** / **83.9**	**96.4** / **52.1**	**82.0** / **57.7**	**71.29**

parameter update conflicts. However, compared to TIES, MEDUSA avoids parameter update conflicts by construction during the fine-tuning process with the parameter ownership assignment, consequently further improving the performance of the merged models.

Second, we investigate whether the task similarity contributes to the performance of merged models, i.e., training the model on dataset A may improve the model's performance on dataset B. We investigate by fine-tuning T5-base on one dataset and test its performance on another. As shown in Table 4, fine-tuning T5-base on one dataset and testing it on another dataset substantially reduces the model performance. The results indicates that the performance of merged models does not benefit from potential task similarity and that MEDUSA does help collaborative FM development in a practical essence.

4.2.2 Main Results on Model Development with Parameter-Efficient Fine-Tuning. Table 5 and Table 6 present the effectiveness of MEDUSA and baseline approaches for collaborative FM development where MEDUSA can fully control the parameter ownership of the two model branches. Both experiments take T0-3B as the base model, with the fine-tuning technique being LoRA and IA3, respectively. We do not conduct experiments on T5-base and T5-large with PEFT techniques because the number of trainable parameters is relatively small for effective merging. We discuss this scenario further in Section 4.4.

From Table 5 and Table 6, we have two major observations. First, consistent with the results on fully fine-tuning techniques, MEDUSA outperforms baseline approaches by 1.8% and 4.6% on average in the setting of LoRA and IA3, respectively. These results demonstrate the generalization of MEDUSA in helping collaborative FM development concerning model size and fine-tuning techniques. Second, the effectiveness of MEDUSA with LoRA is better than that with IA3. The main reason is that fine-tuning models with LoRA has more trainable parameters than that with IA3. Specifically, fine-tuning T0-3B with LoRA has 9.3M trainable parameters while fine-tuning T0-3B with IA3 has only 0.54M trainable parameters. In Section 4.4, we also find that small trainable parameters can lead to failures of model merging. Nevertheless, larger FMs have more trainable parameters when using PEFT techniques to fine-tune, and MEDUSA is expected to have better performance on them.

4.3 RQ2: Effectiveness with Partial Parameter Ownership Control

Table 7 and Table 8 present the main results for collaborative FM development where MEDUSA can control the parameter ownership of only partial model branches. These experiments take T5-base and T5-large as base models, respectively, with the fine-tuning technique being fully fine-tuning. From the tables, we have three major observations. First, MEDUSA outperforms all

Table 7. Effectiveness of MEDUSA on merging model branches developed from T5-base with fully fine-tuning. MEDUSA has full control of parameter ownership on the second model branch and no control of parameter ownership on the first model branch.

Approach	COLA / RTE	COLA / SST2	RTE / COLA	RTE / SST2	SST2 / COLA	SST2 / RTE	Average
Linear Average	69.1 / 56.3	69.1 / 53.8	56.3 / 69.1	51.3 / 52.5	53.8 / 69.1	52.5 / 51.3	58.68
DARE+Linear Average	69.1 / 53.8	69.1 / 52.3	53.8 / 69.1	52.3 / 68.3	52.3 / 69.1	68.3 / 52.3	60.82
Task Arithmetic	69.1 / 51.3	69.1 / 49.4	51.3 / 69.1	51.6 / 49.2	49.4 / 69.1	49.2 / 51.6	56.62
DARE+Task Arithmetic	69.1 / 54.9	69.1 / 50.3	54.9 / 69.1	50.5 / 52.8	50.3 / 69.1	52.8 / 50.5	57.78
TIES	71.7 / 58.8	69.1 / 53.6	58.8 / 71.7	62.5 / 86.4	53.6 / 69.1	86.4 / 62.5	67.02
DARE+TIES	69.6 / 54.2	69.1 / 85.3	54.2 / 69.6	52.7 / 89.9	**85.3** / 69.1	89.9 / 52.7	**70.13**
MEDUSA (Full Control)	**71.9** / 67.9	**74.8** / 84.6	**67.9** / 71.9	**65.0** / **90.6**	84.6 / 74.8	**90.6** / **65.0**	**75.80**
MEDUSA (Partial Control)	69.1 / **68.6**	69.1 / **92.0**	52.7 / **79.4**	52.7 / 89.2	49.5 / **79.6**	83.1 / 52.7	69.82

Table 8. Effectiveness of MEDUSA on merging model branches developed from T5-large with fully fine-tuning. MEDUSA has full control of parameter ownership on the second model branch and no control of parameter ownership on the first model branch.

Approach	COLA / RTE	COLA / SST2	RTE / COLA	RTE / SST2	SST2 / COLA	SST2 / RTE	Average
Linear Average	69.2 / 56.3	36.6 / 53.8	56.3 / 69.2	45.5 / 52.5	53.8 / 36.6	52.5 / 45.5	52.32
DARE+Linear Average	69.1 / 54.2	69.1 / 49.1	54.2 / 69.1	46.6 / 49.1	49.1 / 69.1	49.1 / 46.6	56.20
Task Arithmetic	31.3 / 51.3	30.9 / 49.4	51.3 / 31.3	30.9 / 45.1	49.4 / 30.9	45.1 / 30.9	39.82
DARE+Task Arithmetic	30.9 / 45.1	30.9 / 49.1	45.1 / 30.9	56.0 / 49.1	49.1 / 30.9	49.1 / 56.0	43.52
TIES	71.4 / 58.8	69.2 / 53.6	**58.8** / 71.4	52.7 / 86.4	53.6 / 69.2	86.4 / 52.7	65.35
DARE+TIES	77.1 / 52.7	**76.8** / 49.2	52.7 / 77.1	47.3 / 50.9	49.2 / 76.8	50.9 / 47.3	59.00
MEDUSA (Ours)	**78.7** / 51.6	69.1 / 93.9	51.6 / 78.7	**73.3** / **95.4**	**93.9** / 69.1	**95.4** / 73.3	**77.00**
Partial MEDUSA (Ours)+TIES	69.1 / **86.3**	69.1 / **95.9**	52.7 / **81.6**	52.7 / 92.9	39.1 / **80.2**	51.6 / **84.5**	**71.31**

Table 9. Effectiveness of MEDUSA on merging multiple model branches fully fine-tuned from T5-base.

Approach	3 Tasks			4 Tasks				5 Tasks					Average
	COLA	RTE	SST2	COLA	RTE	SST2	WNLI	COLA	RTE	SST2	WNLI	QNLI	
Linear Average	69.1	54.5	49.1	69.1	49.5	49.1	43.7	69.1	51.6	49.1	46.5	47.3	53.97
Task Arithmetic	69.1	52.3	49.2	69.1	56.7	49.1	43.7	69.1	47.3	49.5	49.3	59.7	55.35
TIES	69.1	58.1	64.4	69.1	45.8	49.7	52.1	69.1	**52.0**	49.4	45.1	52.9	56.41
DARE+Linear Average	69.1	51.3	49.1	69.1	50.2	49.1	43.7	69.1	50.2	49.1	53.5	47.4	54.24
DARE+Task Arithmetic	69.1	48.0	49.2	69.1	49.1	49.1	43.7	69.1	46.6	**56.7**	50.7	59.0	54.95
DARE+TIES	69.0	53.1	71.1	69.1	52.7	68.2	43.7	69.1	47.7	50.9	**59.2**	50.5	58.69
MEDUSA (Ours)	**70.4**	**60.7**	**93.5**	**69.1**	**60.7**	**70.5**	**56.3**	**69.1**	48.5	53.8	49.3	**80.6**	**64.96**

baseline approaches in terms of average performance across multiple tasks. In particular, Table 7 shows that MEDUSA achieves an average performance of 75.80 compared to the best-performing baseline, DARE+TIES, which scores 70.13. Similarly, in Table 8, MEDUSA leads with an average score of 77.00, surpassing all other baseline approaches.

Second, the results from the experiments where MEDUSA has partial control of parameter ownership demonstrate competitive performance. For example, in Table 8, Partial MEDUSA achieves an average score of 71.31, showing that even with limited control over parameter ownership, MEDUSA can significantly improve the performance of merged models compared to baseline approaches.

Third, more control of parameter ownership indicates higher effectiveness. MEDUSA with full control of parameter ownership assignment consistently outperforms MEDUSA with full control of parameter ownership assignment, indicating the usefulness of using MEDUSA for all model branch development.

4.4 RQ3: Ablation Studies and Application Scope Analysis

4.4.1 Scalability to Multiple Branches. Table 9 presents the performance comparison between MEDUSA and multiple baseline approaches for merging model branches fine-tuned on multiple tasks. The results show how each approach scales as the number of tasks increases from 3 to 5.

For each task configuration (3, 4, and 5 tasks), MEDUSA consistently outperforms the baseline approaches, especially as the number of tasks increases. While baseline approaches such as Linear Average and TIES show a gradual decline in performance when merging more than 3 tasks, MEDUSA

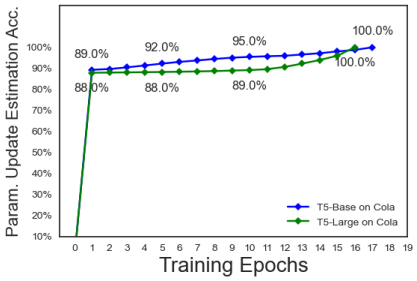


Fig. 4. Efficacy of gradient-based estimation of model parameter updates.

Table 10. Failures when the number of trainable parameters is small.

Approach	T5-base 1.7M[1]	T5-large 4.5M[1]
	COLA / RTE	COLA / RTE
Base Model	69.1 / 52.3	30.9 / 52.7
Finetuned	80.0 / 69.7	78.8 / 66.1
Linear Average	69.1 / 48.7	30.9 / 53.1
DARE+Linear Average	69.1 / 50.9	30.9 / 45.5
Task Arithmetic	69.1 / 52.7	30.9 / 52.0
DARE+Task Arithmetic	69.1 / 52.0	30.9 / 51.6
TIES	69.1 / 52.0	30.9 / 51.6
DARE+TIES	69.1 / 51.3	69.3 / 52.3
MEDUSA (Ours)	69.1 / 55.6	75.3 / 52.7

[1] # of trainable parameters.

maintains its robustness, demonstrating higher average performance across all task sets and its effectiveness in collaborative model merging as the number of tasks and model branches grows.

4.4.2 Efficacy of Gradient-based Estimation. We conduct an ablation study with gradient information at different training epochs to evaluate the efficacy of MEDUSA in balancing the cost and accuracy when using gradient information to estimate the parameter updates in different model branches. Specifically, we investigate the COLA dataset by measuring the consistency of the model parameter update directions with the update direction reflected by gradients. As shown in Figure 4, with the gradient at the first epoch, MEDUSA can correctly predict the update directions of around 90% parameters, demonstrating the efficacy of using gradient information for accurate and cheap parameter update prediction.

4.4.3 Impact of Masked Fine-Tuning. MEDUSA uses masked fine-tuning to avoid conflicts of parameter updates in different model branches. To investigate whether fine-tuning a model with masks harms the performance of the fine-tuned models, we collect the performance of fine-tuned models with and without masks. Table 11 presents the performance of models fine-tuned with and without masks. From Table 11, we have two major observations. First, masked fine-tuning does not compromise the effectiveness of the fine-tuning process for all models. Second, combining the model parameters by averaging the deltas of parameter updates incurs the model performance drop, appealing for a more effective mechanism for incorporating the parameters of two fine-tuned models. However, as MEDUSA focuses on parameter ownership assignment, the merging approach is not the major focus, left for our future work to explore.

4.4.4 Failure Cases of Collaborative FM Development. **Mergeability between conflicting model branches.** While masked fine-tuning does not compromise the model's effectiveness, we observe that on certain task pairs, specifically COLA/RTE and COLA/SST2 on T5-base, all baseline approaches make the fine-tuning invalidated, i.e., the merged model performs comparably to the base model on COLA, despite the base model not being fine-tuned on the task. After investigating the CoLA and SST-2 datasets, we find that the fundamental nature of the two tasks appears incompatible: CoLA requires strict grammatical judgment based on linguistic rules, while SST-2 focuses on semantic sentiment understanding, leading to conflicting optimization objectives during merging if not proactively prevented as Medusa does. This phenomenon also indicates that some tasks may be inherently not suitable for merging, making the mergeability prediction an important direction, left for our future work.

Emergent mergeability of trainable parameters. While we demonstrate the generalization of MEDUSA across different models including T5-base, T5-large, and T0-3B, and different fine-tuning

Table 11. Performance of models fine-tuned with/without masks. FT represents fine-tuning.

Model	COLA	RTE	SST2	QNLI	WNLI
	Base / FT / Masked-FT	Base / FT / Masked-FT	Base / FT / Masked-FT	Base / FT / Masked-FT	Base / FT / Masked-FT
T5-base	69.1 / 78.7 / 79.6	52.3 / 70.4 / 70.8	49.1 / 92.0 / 91.4	49.4 / 89.5 / 88.8	43.7 / 57.7 / 56.3
T5-large	30.9 / 80.8 / 79.8	52.7 / 87.7 / 83.8	49.1 / 95.6 / 96.2	49.5 / 94.0 / 92.3	43.7 / 57.7 / 57.8
T0-3B (LoRA)	57.5 / 84.9 / 84.5	52.7 / 89.2 / 88.4	88.8 / 97.1 / 97.0	49.5 / 94.6 / 94.6	43.7 / 74.6 / 69.0
T0-3B (IA3)	57.5 / 80.8 / 77.5	52.7 / 71.8 / 60.6	88.8 / 96.8 / 96.6	49.5 / 92.4 / 93.1	43.7 / 54.9 / 52.1

techniques including fully fine-tuning, LoRA, and IA3, we also notice that MEDUSA performs better for LoRA than IA3. Given that LoRA has more trainable parameters than IA3, we investigate the impact of the number of trainable parameters on merged models' performance with small models on COLA and RTE datasets, as shown in Table 10. From Table 10, we find that when merging these models, all existing baselines fail to produce meaningful models, i.e., the performance of the models after merging drops to the base model, indicating that the fine-tuning process is wasted after merging. Despite MEDUSA still outperforming all baseline approaches, it also suffers from a performance drop. Consequently, we conclude that MEDUSA and model merging are not suitable for small models without enough trainable parameters to accommodate skills obtained from different sources.

5 Discussion and Future Work

In this section, we discuss future work that could address the current limitations of MEDUSA.

5.1 White-box Model Merging

MEDUSA makes no assumptions about the internal architecture of the models that it merges, missing opportunities to exploit architecture-specific characteristics of the models for improved merging performance. Future work could involve developing white-box merging techniques that consider the specific layers, activation functions, and architectural nuances of the models. By understanding and leveraging these details, we could potentially enhance the efficiency and effectiveness of the merging process, leading to better performance of the merged models across tasks.

5.2 Model Merging across Architectures

Currently, MEDUSA is limited to merging models that have been fine-tuned from the same pre-trained model. Future extensions of MEDUSA could explore techniques for merging models across different architectures. Doing so would involve developing techniques to align and reconcile differences in model structure, parameterization, and feature representations, thereby enabling the combination of models with diverse origins.

5.3 Mergeability Prediction

As revealed in our evaluation in Section 4.4, some tasks are inherently not mergeable due to huge differences and conflicts between two model branches. Mergeability prediction could involve developing metrics or machine learning models that assess the likelihood of successful model merging based on pre-fine-tuning characteristics. Implementing these features would streamline the model merging process, making it more robust and less reliant on manual intervention.

6 Threats to Validity

The main threat to the external validity concerns the representativeness of the models, datasets, training techniques, and baselines selected for our evaluations. To mitigate the impact of the bias introduced by the model and dataset selection, we use three popular pre-trained models and five datasets widely used by related work on model merging. To mitigate the impact of the bias introduced by training technique selection, we use three different and popular fine-tuning techniques. To mitigate the impact of the bias introduced by baseline selection, we choose recent

and state-of-the-art model merging approaches. Evaluating MEDUSA with more different datasets, models, and training techniques will further alleviate the threat.

The threats to internal validity are instrumentation effects that can bias our results, including faults in our implementation of MEDUSA, parameter selection in MEDUSA, and hyper-parameter selection during fine-tuning models. To mitigate the impact of the bias introduced by the parameter selection in MEDUSA, we conduct extensive ablation studies with different parameters in MEDUSA. To mitigate the impact of the hyper-parameter selection during fine-tuning models, we follow previous work [105] and use the same evaluation metrics in our evaluations.

7 Related Work

7.1 Post-training Model Merging

There is a growing body of work on post-training model merging [15, 40, 67, 103, 105, 109, 113], which focuses on combining pre-trained models that have been fine-tuned on different tasks. Linear averaging approaches [103, 113] average parameters of multiple fine-tuned models element-wise. Task arithmetic [15, 40, 67] leverages task-specific weight vectors, to represent weights for specific tasks to facilitate direct manipulation of multi-task learning by adding or subtracting on task vectors. These approaches are vulnerable to parameter update conflicts [13, 48]. TIES [105] tries to reduce parameter update conflicts with heuristics such as trimming minor updates to avoid redundant updates [111]. DARE [109] improves upon TIES by re-scaling the remaining parameters to keep the distributional stability. Complementing the existing work focusing on resolving parameter update conflicts *after* fine-tuning, MEDUSA is the first to resolve parameter update conflicts *before* fine-tuning, with the novel concept of parameter ownership assignment. In addition, MEDUSA can be combined with any existing or future post-training model merging techniques to further improve the effectiveness of collaborative FM development.

7.2 Masked Training of Deep Learning Models

Masked training [52, 63], a technique aimed at enhancing model efficiency and generalization [59], has become increasingly relevant in the context of deep learning. Sung et al. [87] demonstrate that applying fixed masks during training [98] can significantly reduce the number of parameters while maintaining model performance. This approach has been particularly impactful in environments where computational resources are limited, such as in edge computing [115] and federated learning scenarios [51]. In distributed training [83] and federated learning [57, 96, 97], masked training techniques help reduce the communication overhead [1] by limiting the number of parameters that need to be updated and shared across different nodes. This training technique not only improves training efficiency but also enhances privacy by ensuring that only a small, non-sensitive subset of the model parameters is exchanged between nodes. Our work benefits from the existing principles for masked training of deep learning models. By using masks during the fine-tuning process, we ensure that only the most important parameters in each model branch are updated, substantially reducing the parameter update conflicts while maintaining the efficacy of the fine-tuning process and the performance of the merged model.

7.3 Version Control Systems of FMs

The application of version control systems (VCS) [17] to deep learning models is an emerging research field. Git-Theta [45] has pioneered extending Git's large file system [56] to optimize large file management [58] associated with machine learning models [99, 110]. By tracking changes to model weights and configurations, Git-Theta reduces the storage costs for large models. In contrast to Git-theta, which focuses on the storage optimization of FMs, MEDUSA focuses on facilitating

the collaborative development of FMs by reducing parameter update conflicts with automated parameter ownership assignment.

7.4 Ensemble Learning over Multiple Models

It is important to acknowledge that ensemble learning is also a closely related topic to MEDUSA. Ensemble learning [31] has a rich history in machine learning, dating back several decades [5, 26, 100]. Traditional ensemble approaches combine multiple models trained on the same or similar tasks to improve performance and generalization. Some ensemble approaches require maintaining and running multiple models during inference [5, 26]. For example, bagging [5] and boosting [26] have a rich history of combining multiple models to improve performance on a single task [12, 27, 64, 114]. These approaches typically operate by aggregating outputs from multiple models during inference, whether through voting [6], averaging [20], or weighted combinations [32]. Notably, some ensemble approaches, such as parameter averaging and model distillation, do produce a single unified model [100, 103]. For example, parameter averaging [100, 103] combines multiple models trained with different random seeds by averaging their weights, resulting in a single model with improved generalization. Knowledge distillation [34] ensembles multiple teacher models into a single student model [28].

MEDUSA shares the foundational insights from ensemble approaches, particularly parameter averaging. While there are similarities in the technical approach of combining model parameters, our work explores a different and complementary direction. Where traditional ensemble learning excels at combining models trained on the same task to improve performance [22, 106], we investigate the possibility of merging models trained on different tasks to enable capability sharing across domains, yet reducing the conflicts among different tasks. Complementing the existing work focusing on resolving parameter update conflicts *after* fine-tuning, MEDUSA is the first to resolve parameter update conflicts *before* fine-tuning, with the novel concept of parameter ownership assignment. In addition, MEDUSA can be combined with any existing or future post-training model merging/ensemble techniques to further improve the effectiveness of collaborative FM development.

8 Conclusion

In this paper, we have introduced MEDUSA, a pioneering framework designed for collaborative FM development. By managing model branches and introducing a structured system of parameter ownership, MEDUSA tracks different fine-tuning efforts as separate branches, similar to Git, allowing developers to work on different tasks without destabilizing the base model. MEDUSA proactively optimizes the merging process through our parameter-ownership-assignment algorithm to generate merging-aware masks. These masks guide the fine-tuning process on different branches, ensuring that only specific branches can modify designated parameters. Extensive evaluations on five datasets, three fine-tuning techniques, and three popular models against six state-of-the-art post-training model-merging approaches demonstrate the efficacy and generalization of MEDUSA, opening up a new direction for deriving engineering principles for FM development, similar to those used in traditional software engineering.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China under Grant No. 623B2006 and Grant No. 92464301, and an Amazon Trust AI Research Award.

References

[1] Ahmed A Al-Saedi, Veselka Boeva, and Emiliano Casalicchio. 2021. Reducing communication overhead of federated learning through clustering analysis. In *ISCC*. 1–7.

[2] Dzmitry Bahdanau. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Kent Beck. 1999. Embracing change with extreme programming. *Computer* 32, 10 (1999), 70–77.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[5] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24 (1996), 123–140.

[6] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[7] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[8] Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. 2019. Bfloat16 processing for neural networks. In *ARITH*. 88–91.

[9] Simin Chen, Xiaoning Feng, Xiaohong Han, Cong Liu, and Wei Yang. 2024. PPM: Automated generation of diverse programming problems for benchmarking code generation models. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 1194–1215.

[10] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 2022. Nmtsloth: understanding and testing efficiency degradation of neural machine translation systems. In *ESEC/FSE*. 1148–1160.

[11] Simin Chen, Pranav Pusarla, and Baishakhi Ray. 2025. Dynamic benchmarking of reasoning capabilities in code large language models under data contamination. *arXiv preprint arXiv:2503.04149* (2025).

[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *SIGKDD*. 785–794.

[13] Weiyu Chen and James Kwok. 2024. You only merge once: Learning the pareto set of preference-aware model merging. *arXiv preprint arXiv:2408.12105* (2024).

[14] Zhiyuan Chen and Bing Liu. 2022. *Lifelong machine learning*. Springer Nature.

[15] Rajas Chitale, Ankit Vaidya, Aditya Kane, and Archana Ghotkar. 2023. Task arithmetic with LoRA for continual learning. arXiv:2311.02428 [cs.CV] https://arxiv.org/abs/2311.02428

[16] KR1442 Chowdhary and KR Chowdhary. 2020. Natural language processing. *Fundamentals of artificial intelligence* (2020), 603–649.

[17] Catarina Costa and Leonardo Murta. 2013. Version control in distributed software development: A systematic mapping study. In *ICGSE*. 90–99.

[18] Francesco De Toni, Christopher Akiki, Javier De La Rosa, Clémentine Fourrier, Enrique Manjavacas, Stefan Schweter, and Daniel Van Strien. 2022. Entities, dates, and languages: Zero-shot on historical texts with t0. *arXiv preprint arXiv:2204.05211* (2022).

[19] Li Deng and Yang Liu. 2018. *Deep learning in natural language processing*. Springer.

[20] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems*. 1–15.

[21] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *NIPS* 32 (2019).

[22] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. 2020. A survey on ensemble learning. *Frontiers of Computer Science* 14 (2020), 241–258.

[23] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *TAMC*. 1–19.

[24] Xiaoning Feng, Xiaohong Han, Simin Chen, and Wei Yang. 2024. LLMEffiChecker: Understanding and testing efficiency degradation of large language models. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–38.

[25] Denis Foley and John Danskin. 2017. Ultra-performance Pascal GPU and NVLink interconnect. *IEEE Micro* 37, 2 (2017), 7–17.

[26] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139.

[27] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[28] Takashi Fukuda, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. 2017. Efficient knowledge distillation from an ensemble of teachers.. In *Interspeech*. 3697–3701.

[29] Tom Gilb, Susannah Finzi, et al. 1988. *Principles of software engineering management*. Vol. 11. Addison-wesley Reading, MA.

[30] Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. LongT5: Efficient text-to-text transformer for long sequences. *arXiv preprint arXiv:2112.07916* (2021).

[31] Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *TPAMI* 12, 10 (1990), 993–1001.

[32] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

[33] Greg Henry, Ping Tak Peter Tang, and Alexander Heinecke. 2019. Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations. In *ARITH*. 69–76.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[35] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *JMLR* 22, 241 (2021), 1–124.

[36] Ellis Horowitz and Barry W Boehm. 1975. *Practical strategies for developing large software systems*. Citeseer.

[37] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-rank adaptation of large language models. arXiv:2106.09685 [cs.CL] https://arxiv.org/abs/2106.09685

[38] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*. PMLR, 9118–9147.

[39] Shuning Huo, Yafei Xiang, Hanyi Yu, Mengran Zhu, and Yulu Gong. 2024. Deep learning approaches for improving question answering systems in hepatocellular carcinoma research. arXiv:2402.16038 [cs.CL] https://arxiv.org/abs/2402.16038

[40] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic. arXiv:2212.04089 [cs.LG] https://arxiv.org/abs/2212.04089

[41] Jingjing Jiang and Nanning Zheng. 2023. MixPHM: redundancy-aware parameter-efficient tuning for low-resource visual question answering. In *CVPR*. 24203–24213.

[42] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2022. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849* (2022).

[43] Hayeon Jo, Hyesong Choi, Minhee Cho, and Dongbo Min. 2024. iConFormer: Dynamic parameter-efficient tuning with input-conditioned adaptation. arXiv:2409.02838 [cs.CV] https://arxiv.org/abs/2409.02838

[44] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and trends® in machine learning* 14, 1–2 (2021), 1–210.

[45] Nikhil Kandpal, Brian Lester, Mohammed Muqeeth, Anisha Mascarenhas, Monty Evans, Vishal Baskaran, Tenghao Huang, Haokun Liu, and Colin Raffel. 2023. Git-theta: A git extension for collaborative development of machine learning models. In *ICML*. PMLR, 15708–15719.

[46] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an LLM-based programming assistant that balances student and educator needs. In *CHI*. 1–20.

[47] Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[48] Fanshuang Kong, Richong Zhang, and Ziqiao Wang. 2024. Activated parameter locating via causal intervention for model merging. *arXiv preprint arXiv:2408.09485* (2024).

[49] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[50] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv:1910.13461 [cs.CL] https://arxiv.org/abs/1910.13461

[51] Ang Li, Jingwei Sun, Xiao Zeng, Mi Zhang, Hai Li, and Yiran Chen. 2021. Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking. In *SenSys*. 42–55.

[52] Siyuan Li, Luyuan Zhang, Zedong Wang, Di Wu, Lirong Wu, Zicheng Liu, Jun Xia, Cheng Tan, Yang Liu, Baigui Sun, et al. 2023. Masked modeling for self-supervised representation learning on vision and beyond. *arXiv preprint arXiv:2401.00897* (2023).

[53] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. arXiv:2205.05638 [cs.LG] https://arxiv.org/abs/2205.05638

[54] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[55] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *ACL* (2020), 726–742.

[56] Yucheng Low, Rajat Arya, Ajit Banerjee, Ann Huang, Brian Ronan, Hoyt Koepke, Joseph Godlewski, and Zach Nation. 2023. Git Is For Data. In *CIDR*.

[57] Priyanka Mary Mammen. 2021. Federated learning: Opportunities and challenges. *arXiv preprint arXiv:2101.05428* (2021).

[58] Udi Manber et al. 1994. Finding similar files in a large file system.. In *USENIX Winter*, Vol. 94. 1–10.

[59] Yuzhu Mao, Siqi Ping, Zihao Zhao, Yang Liu, and Wenbo Ding. 2024. Enhancing parameter efficiency and generalization in large-scale models: A regularized and masked low-rank adaptation approach. *arXiv preprint arXiv:2407.12074* (2024).
[60] Antonio Mastropaolo, Simone Scalabrino, Nathan Cooper, David Nader Palacio, Denys Poshyvanyk, Rocco Oliveto, and Gabriele Bavota. 2021. Studying the usage of text-to-text transfer transformer to support code-related tasks. In *ICSE*. 336–347. doi:10.1109/ICSE43902.2021.00041
[61] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. arXiv:1806.08730 [cs.CL] https://arxiv.org/abs/1806.08730
[62] Harlan D Mills. 1980. The management of software engineering, Part I: Principles of software engineering. *IBM Systems Journal* 19, 4 (1980), 414–420.
[63] Amirkeivan Mohtashami, Martin Jaggi, and Sebastian Stich. 2022. Masked training of neural networks with partial gradients. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 5876–5890.
[64] Dominik Müller, Iñaki Soto-Rey, and Frank Kramer. 2022. An analysis on ensemble learning optimized medical image classification with deep convolutional neural networks. *IEEE Access* 10 (2022), 66467–66480.
[65] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. 2011. Natural language processing: an introduction. *Journal of the American Medical Informatics Association* 18, 5 (2011), 544–551.
[66] OpenAI. 2024. Fine-tuning OpenAI models available through the API. https://platform.openai.com/docs/guides/fine-tuning
[67] Guillermo Ortiz-Jimenez, Alessandro Favero, and Pascal Frossard. 2023. Task arithmetic in the tangent space: Improved editing of pre-trained models. *NIPS* 36 (2023), 66727–66754.
[68] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *NIPS* 35 (2022), 27730–27744.
[69] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. 2022. Teach: Task-driven embodied agents that chat. In *AAAI*, Vol. 36. 2017–2025.
[70] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
[71] Jeff Pitman. 2021. Google Translate: One billion installs, one billion stories. https://blog.google/products/translate/one-billion-installs/
[72] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv:1910.10683 [cs.LG] https://arxiv.org/abs/1910.10683
[73] Dezhi Ran, Yuan Cao, Yuzhe Guo, Yuetong Li, Mengzhou Wu, Simin Chen, Wei Yang, and Tao Xie. 2025. Medusa Artifacts. doi:10.5281/zenodo.15203401
[74] Dezhi Ran, Zongyang Li, Chenxu Liu, Wenyu Wang, Weizhi Meng, Xionglin Wu, Hui Jin, Jing Cui, Xing Tang, and Tao Xie. 2022. Automated visual testing for mobile apps in an industrial setting. In *ICSE-SEIP*. 55–64.
[75] Dezhi Ran, Hao Wang, Zihe Song, Mengzhou Wu, Yuan Cao, Ying Zhang, Wei Yang, and Tao Xie. 2024. Guardian: A Runtime Framework for LLM-Based UI Exploration. In *ISSTA*. 958–970.
[76] Dezhi Ran, Hao Wang, Wenyu Wang, and Tao Xie. 2023. BADGE: Prioritizing UI Events with Hierarchical Multi-Armed Bandits for Automated UI Testing. In *ICSE*. 894–905.
[77] Dezhi Ran, Mengzhou Wu, Wei Yang, and Tao Xie. 2025. Foundation model engineering: engineering foundation models just as engineering software. *TOSEM* (2025).
[78] Dezhi Ran, Mengzhou Wu, Hao Yu, Yuetong Li, Jun Ren, Yuan Cao, Xia Zeng, Haochuan Lu, Zexin Xu, Mengqian Xu, et al. 2025. Beyond pass or fail: A multi-dimensional benchmark for mobile UI navigation. *arXiv preprint arXiv:2501.02863* (2025).
[79] Olesya Razuvayevskaya, Ben Wu, João A Leite, Freddy Heppell, Ivan Srba, Carolina Scarton, Kalina Bontcheva, and Xingyi Song. 2024. Comparison between parameter-efficient techniques and full fine-tuning: A case study on multilingual news article classification. *PLOS One* 19, 5 (2024), e0301738.
[80] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *IUI*. 491–514.
[81] Winston W Royce. 1987. Managing the development of large software systems: concepts and techniques. In *ICSE*. 328–338.
[82] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207* (2021).

[83] Simone Scardapane and Paolo Di Lorenzo. 2017. A framework for parallel and distributed training of neural networks. *Neural Networks* 91 (2017), 42–54.
[84] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. arXiv:1704.04368 [cs.CL] https://arxiv.org/abs/1704.04368
[85] Diomidis Spinellis. 2005. Version control systems. *IEEE Software* 22, 5 (2005), 108–109.
[86] Diomidis Spinellis. 2012. Git. *IEEE Software* 29, 3 (2012), 100–101.
[87] Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *NIPS* 34 (2021), 24193–24205.
[88] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938* (2023).
[89] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[90] Barak Turovsky. 2016. Ten years of Google Translate. https://www.blog.google/products/translate/ten-years-of-google-translate/
[91] Hans Van Vliet, Hans Van Vliet, and JC Van Vliet. 2008. *Software engineering: principles and practice*. Vol. 13. John Wiley & Sons Hoboken, NJ.
[92] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017).
[93] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2020. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. arXiv:1905.00537 [cs.CL] https://arxiv.org/abs/1905.00537
[94] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv:1804.07461 [cs.CL] https://arxiv.org/abs/1804.07461
[95] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).
[96] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1205–1221.
[97] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3454–3469.
[98] Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. 2022. Should you mask 15% in masked language modeling? *arXiv preprint arXiv:2202.08005* (2022).
[99] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *EMNLP: system demonstrations*. 38–45.
[100] David H Wolpert. 1992. Stacked generalization. *Neural Networks* 5, 2 (1992), 241–259.
[101] Laurence A Wolsey. 2007. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering* (2007), 1–10.
[102] Ray W Wolverton. 1974. The cost of developing large-scale software. *IEEE Trans. Comput.* 100, 6 (1974), 615–636.
[103] 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. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*. PMLR, 23965–23998.
[104] L Xue. 2020. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934* (2020).
[105] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. *NIPS* 36 (2024).
[106] Yongquan Yang, Haijun Lv, and Ning Chen. 2023. A survey on ensemble learning under the era of deep learning. *Artificial Intelligence Review* 56, 6 (2023), 5545–5589.
[107] Peng Ye, Yongqi Huang, Chongjun Tu, Minglei Li, Tao Chen, Tong He, and Wanli Ouyang. 2023. Partial fine-tuning: A successor to full fine-tuning for vision transformers. arXiv:2312.15681 [cs.CV] https://arxiv.org/abs/2312.15681
[108] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. 2024. CoderEval: A benchmark of pragmatic code generation with generative pre-trained models. In *ICSE*. 1–12.
[109] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2024. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *ICML*.

[110] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. 2018. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.* 41, 4 (2018), 39–45.
[111] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. 2021. Barlow twins: Self-supervised learning via redundancy reduction. In *ICML*. PMLR, 12310–12320.
[112] Yuchen Zeng and Kangwook Lee. 2024. The expressive power of low-rank adaptation. arXiv:2310.17513 [cs.LG] https://arxiv.org/abs/2310.17513
[113] Xinyu Zhang, Yanyuan Ma, and Raymond J Carroll. 2019. MALMEM: model averaging in linear measurement error models. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 81, 4 (2019), 763–779.
[114] Changming Zhao, Dongrui Wu, Jian Huang, Ye Yuan, Hai-Tao Zhang, Ruimin Peng, and Zhenhua Shi. 2022. BoostTree and BoostForest for ensemble learning. *TPAMI* 45, 7 (2022), 8110–8126.
[115] Tong Zhou, Jiahui Zhao, Yukui Luo, Xi Xie, Wujie Wen, Caiwen Ding, and Xiaolin Xu. 2024. Adapi: Facilitating dnn model adaptivity for efficient private inference in edge computing. *arXiv preprint arXiv:2407.05633* (2024).

Received 2024-09-13; accepted 2025-04-01