# CASS: Contribution-Aware Structured Sparsity for Model Merging

**Yan Li**[1] **Guiping Cao**[1] **Meng Xu**[2] **Tao Jiang**[1,3]
**Yaguang Song**[1] **Ming Tao**[1] **Yaowei Wang**[1,4] **Dongmei Jiang**[1,5*]
[1]Pengcheng Laboratory, Shenzhen, China
[2]City University of Hong Kong, Hong Kong, China
[3]Southern University of Science and Technology, Shenzhen, China
[4]Harbin Institute of Technology, Shenzhen, China
[5]Northwestern Polytechnical University, Xi'an, China
liyan4ai@gmail.com, jiangdm@pcl.ac.cn

## Abstract

Model merging integrates task-specific fine-tuned models into a single multi-task model, but often suffers from parameter interference caused by conflicting task-vector updates. Existing methods typically mitigate conflicts by pruning task vectors based on weight magnitude or random heuristics, treating Transformers as unstructured "bags of parameters" and overlooking their inherent modularity. Inspired by insights from mechanistic interpretability, we propose **C**ontribution-**A**ware **S**tructured **S**parsity (**CASS**), a unified framework that reduces parameter interference by identifying and preserving task-specific components. At the core of CASS is a contribution-aware structured mask that localizes task-relevant attention heads and FFN neurons. We instantiate this mask in two settings: **CASS-Merging**, the primary post-hoc setting where masks serve as a plug-and-play denoising filter for existing merging operators, and **CASS-Tuning**, an extension for scenarios with fine-tuning access where masks constrain gradients to reduce structural overlap between task vectors. Our analysis shows that task-relevant components are sparse and partially disjoint, supporting structured component-level filtering as an effective way to reduce merging interference. Extensive experiments across vision (ViT, 20 tasks) and language (RoBERTa, 8 tasks; Qwen2.5, 4 tasks) benchmarks demonstrate that CASS improves a range of representative merging baselines.

## 1 Introduction

Fine-tuning a foundation model for each downstream task produces a collection of task-specific expert models, but deploying all experts independently incurs substantial storage and memory costs. Model merging addresses this problem by integrating multiple fine-tuned models into a single multi-task model, typically by combining their task vectors, i.e., the parameter differences between fine-tuned and pre-trained models [53, 20]. This paradigm enables capability integration without retraining a joint model from scratch and has become a practical tool for reusing task-specific checkpoints [45, 16, 58].

Despite its efficiency, model merging remains limited by parameter interference: task vectors obtained from different tasks may contain incompatible updates, and directly aggregating them can degrade the resulting multi-task model. Existing methods mitigate this issue from different perspectives. Optimization-based and subspace-based methods use second-order statistics, inner-product structure, or singular value decomposition to reweight or transform task vectors [34, 61, 12, 33]. These

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

approaches are often effective but may introduce additional computational or optimization overhead. In contrast, sparsification-based methods improve efficiency by pruning task-vector updates according to magnitude or random criteria [60, 62, 51]. However, such methods usually operate at the individual-parameter level, effectively treating a Transformer as an unstructured "bag of parameters." As illustrated in Fig. 1, they do not explicitly account for the fact that Transformer computations are organized into structured functional components such as attention heads and feed-forward neurons.

This motivates a different view of task-vector filtering. In Transformers, both multi-head attention and feed-forward networks can be decomposed into additive contributions to the residual stream. Attention heads perform distinct information-routing functions [35, 38], while FFN neurons often exhibit sparse and task-dependent activation patterns [13, 55]. Therefore, task-specific knowledge is unlikely to be uniformly distributed over all parameters; instead, it may be concentrated in a sparse subset of activation-prominent components. From this perspective, the importance of a task-vector update should not be determined solely by its static weight magnitude, but by the functional contribution of the corresponding component under task-specific inputs.

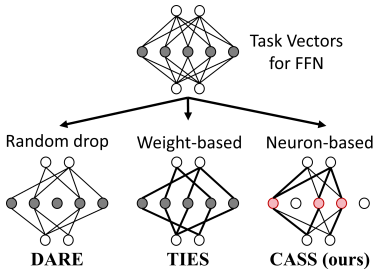


Figure 1: Sparsification-based model merging methods.

In this paper, we propose **C**ontribution-**A**ware **S**tructured **S**parsity (CASS), a framework for reducing model-merging interference by filtering task vectors at the level of attention heads and FFN neurons. CASS first estimates the contribution of each component using a small unlabeled task-specific probe set, combining activation statistics with the component's output projection strength. It then constructs structured binary masks that retain task-relevant components and suppress task-irrelevant updates. Importantly, CASS removes only task-specific parameter updates on inactive components, thereby reverting them to the pre-trained model rather than deleting the components themselves.

We use CASS to refer to the general contribution-aware structured masking framework, and distinguish between two instantiations depending on whether fine-tuned task experts are already available. In the standard post-hoc merging setting, CASS-Merging serves as the primary instantiation: it masks each task vector before applying an existing merging operator such as Task Arithmetic, optimization-based merging, or subspace-based merging. When fine-tuning access is available, CASS-Tuning provides a proactive extension that applies the same structured masks to gradients during task-specific fine-tuning, encouraging different task vectors to occupy structurally separated subspaces by construction.

Our contributions are summarized as follows:

- We formulate Transformer attention heads and FFN neurons as structured additive contributors to the residual stream, providing an algebraic basis for component-level task-vector filtering.
- We propose CASS, a contribution-aware structured sparsity framework that identifies task-relevant heads and neurons using activation-weighted importance metrics and applies the resulting masks to reduce task-vector interference.
- We instantiate CASS primarily as CASS-Merging, a post-hoc pre-filter compatible with existing merging operators, and further extend it to CASS-Tuning, a training-time gradient masking strategy that proactively induces structural separation among task vectors.
- We conduct extensive experiments across vision encoders (ViT), encoder-only (RoBERTa) and decoder-only (Qwen2.5) language models, spanning 20, 8, and 4 tasks respectively, to demonstrate the effectiveness of the proposed CASS.

## 2 Related Work

### 2.1 Model Merging

Model merging aims to integrate the capabilities of multiple task-specific models into a single multi-task model without retraining. We categorize existing methods into four groups. Here, we omit a

detailed discussion of MoE-based methods [32, 46, 27] because the resulting merged model no longer preserves the original model architecture.

**Linear merging** methods combine task vectors via simple arithmetic operations. Task Arithmetic defines task vectors as the weight difference between fine-tuned and pre-trained models, and aggregates them via scaled summation [20]. Model Soups averages the weights of multiple fine-tuned checkpoints directly [53].

**Sparsification-based methods** mitigate interference by pruning conflicting updates. TIES-Merging resolves sign conflicts by retaining only updates with larger magnitudes [60]. DARE randomly drops parameter updates and rescales the remaining updates [62]. PCB-Merging introduces intra-balancing and inter-balancing mechanisms to decide the retention and pruning of parameters [10]. TALL-Masks constructs task-specific binary masks based on weight magnitude differences [51]. While these methods improve upon simple averaging, they operate at the individual parameter level, treating the model as an unstructured "bag of parameters" and ignoring the inherent modular structure of Transformer architectures.

**Subspace-based methods** exploit the geometric structure of task matrices. For example, TSV-Merging applies singular value decomposition to per-layer task matrices and uses a whitening transformation to decorrelate singular vectors across tasks [12]. Iso-C enforces an isotropic singular value spectrum on the merged matrix to improve subspace alignment [33, 30]. These methods aim to identify low-interference directions in the static weight space.

**Optimization-based methods** resolve interference via iterative optimization [21]. Fisher Merging [34] and RegMean [22] employ second-order information (Fisher matrix or Gram matrix) to perform weighted parameter averaging, but incur significant computational overhead. WUDI-Merging minimizes interference within the linear subspace spanned by task vectors via gradient descent, achieving strong performance without requiring additional data [5]. AdaMerging optimizes per-layer merging coefficients via entropy minimization on unlabeled test data, but is limited to classification settings where output entropy is well-defined [61].

In contrast, CASS identifies task-specific components via contribution-based metrics, operating at the structured level of attention heads and FFN neurons rather than individual parameters or weight-space decompositions. This approach is complementary to all four categories above, as demonstrated by our experiments applying CASS as a pre-filter on top of representative methods from each group.

### 2.2 Mechanistic Interpretability

Our work is inspired by the growing field of mechanistic interpretability [11, 29, 43], which views Transformers not as monolithic blocks but as collections of specialized functional components. Extensive research suggests that Feed-Forward Networks (FFNs) act as Key-Value memories [13], where specific neurons store discrete knowledge patterns. These neurons often exhibit high activation sparsity [55]. Similarly, Attention Heads have been shown to perform distinct roles [35], such as induction heads for in-context learning [38] or retrieval heads for factual recall [54].

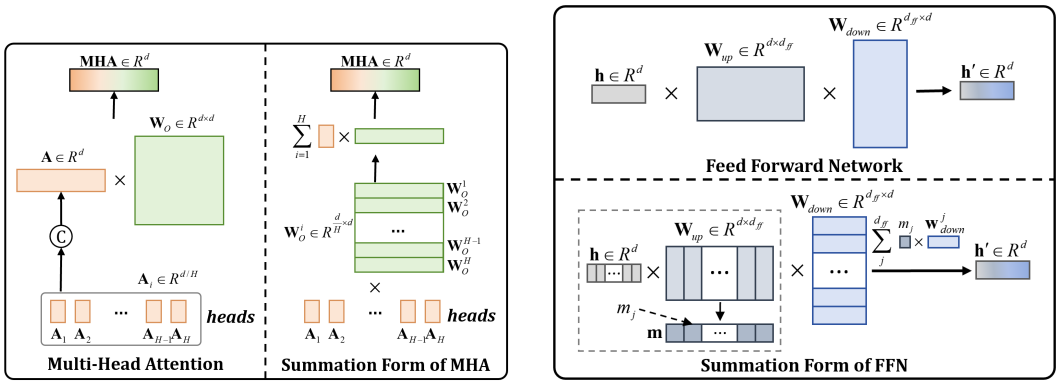
CASS leverages these functional specialization properties directly. Unlike previous merging methods that rely on static weights, we identify task-specific components using contribution-based statistics, aligning the merging process with the model's intrinsic modular structure.

## 3 Motivation and Analysis

### 3.1 Unified Summation Form of Transformer

Common merging methods often treat model parameters as unstructured vectors, ignoring the modular architecture of Transformers. To provide an algebraic grounding for structured masking, we reformulate both MHA and FFN into a unified summation form. Consider a vanilla Transformer layer [48] with input $\mathbf{x} \in \mathbb{R}^d$. We demonstrate that the output of both sub-layers can be decomposed into a sum of independent additive terms.

**MHA as a Sum of Heads.** Let an MHA layer have $H$ heads. By decomposing the output projection $\mathbf{W}_O \in \mathbb{R}^{d \times d}$ into $H$ sub-matrices $\mathbf{W}_O^{(i)} \in \mathbb{R}^{(d/H) \times d}$, as illustrated in Fig. 2, the MHA output




(a) Unified summation form of MHA

(b) Unified summation form of FFN

Figure 2: Unified summation form of Transformer.

becomes:

$$\text{MHA}(\mathbf{x}) = \sum_{i=1}^{H} \mathbf{A}_i(\mathbf{x})\mathbf{W}_O^{(i)} \tag{1}$$

$\mathbf{A}_i(\mathbf{x})$ is the output of the $i$-th head. Therefore, each term $\mathbf{A}_i(\mathbf{x})\mathbf{W}_O^{(i)}$ is an independent additive update to the residual stream. Masking head $i$ in the task vector is thus algebraically equivalent to zeroing out the task-specific parameter updates associated with that head.

**FFN as a Sum of Neurons.** Let $\mathbf{m} = \sigma(\mathbf{x}\mathbf{W}_{up}) \in \mathbb{R}^{d_{ff}}$ be the intermediate activation, where $m_j$ is the scalar activation of neuron $j$. Decomposing $\mathbf{W}_{down}$ into row vectors $\mathbf{w}_{down}^{(j)} \in \mathbb{R}^d$:

$$\text{FFN}(\mathbf{x}) = \sum_{j=1}^{d_{ff}} m_j \cdot \mathbf{w}_{down}^{(j)} \tag{2}$$

Therefore, each neuron $j$ contributes an additive vector scaled by its activation $m_j$; masking neuron $j$ in the task vector is algebraically equivalent to zeroing out the task-specific updates associated with that intermediate channel. Detailed derivations are provided in Appendix A.

### 3.2 Importance Metric via Contribution-Awareness

Having established the summation form, the challenge lies in identifying which components are critical for a specific task. Traditional sparsification-based model merging methods rely on weight magnitude alone. Here, we propose a *Contribution Score* $\mathcal{I}$ that measures the expected magnitude of **a component's additive contribution to the residual stream** over a small unlabeled probe set $\mathcal{D}_{\text{probe}}$. For **FFN neurons**, the absolute contribution magnitude serves as an effective task-specificity proxy:

$$\mathcal{I}_{\text{neuron}}^{(j)} = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}_{\text{probe}}}\left[|m_j| \cdot \left\|\mathbf{w}_{down}^{(j)}\right\|_2\right] \tag{3}$$

For **attention heads**, however, many heads serve as general-purpose mechanisms active across all tasks, making absolute magnitude an unreliable discriminator. We therefore employ a task-selectivity score that captures relative deviation from the global average contribution (see Appendix E for details).

### 3.3 Empirical Analysis

To validate our motivation, we analyze the distribution of functional components on **Qwen2.5-1.5B-Instruct** [40] across four tasks: Coding [52], Instruction Following [25], Mathematics [47], and Safety [15].

The left of Fig. 3 plots the Cumulative Distribution Function (CDF) of normalized neuron contribution scores for the Coding task. The sharp rise near zero suggests that the vast majority (>80%) of neurons have negligible contributions to any specific task, which means task-relevant functional activity is concentrated in a small subset of "loud" components.

The right of Fig. 3 shows the Jaccard similarity of the top-20% most active neurons across tasks (Layer 14). The near-zero off-diagonal values reveal a striking pattern: *the highly active components of different tasks are largely disjoint*. This structural non-overlap suggests that parameter interference in standard merging primarily arises from forcing the "loud" components of one task to mix with the "silent" background of another. By explicitly identifying and preserving these task-specific activation-prominent components via structured masks, our method aims to reduce such conflicts. More analysis on different layers and tasks is provided in Appendix D and E.

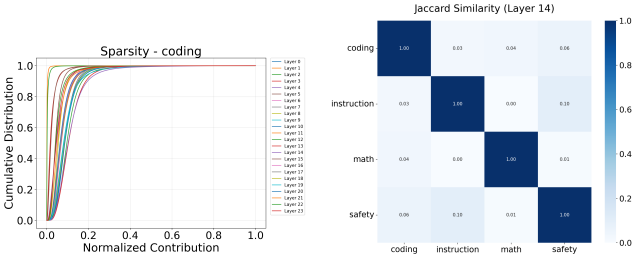



Figure 3: **(Left)** CDF of neuron contribution scores (Coding task). **(Right)** Jaccard similarity of top-20% neurons across tasks (Layer 14).

## 4 Methodology: CASS

Based on the insights in Sec. 3, we propose **CASS**, a framework to mitigate model merging interference by identifying and preserving task-specific components. As illustrated in Fig. 4, CASS consists of a shared mask-generation stage followed by a setting-specific mask-application stage. The shared stage estimates contribution-aware structured masks over attention heads and FFN neurons. The application stage then depends on the available access: in CASS-Merging, masks are applied to task vectors after fine-tuning; in CASS-Tuning, masks are applied to gradients during fine-tuning. This organization emphasizes that both variants share the same structured sparsity principle, while differing only in when the mask is applied.

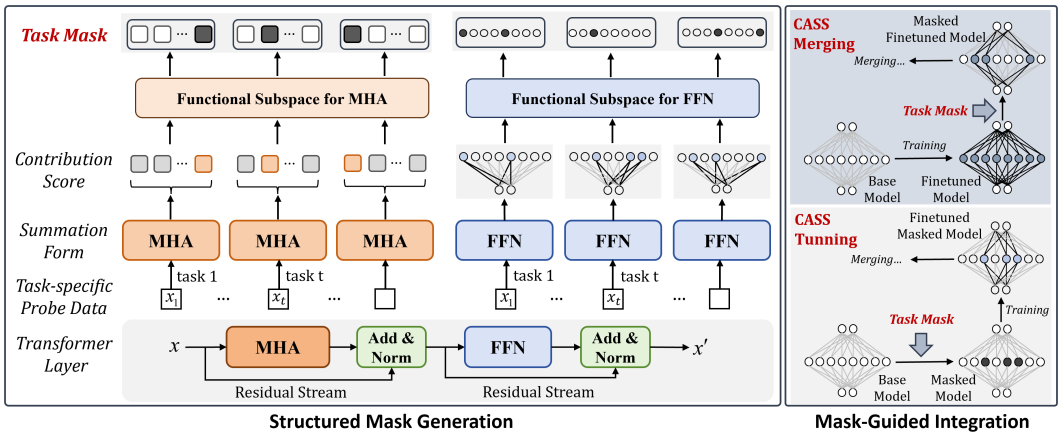



Figure 4: The proposed **C**ontribution-**A**ware **S**tructured **S**parsity (CASS) framework.

### 4.1 Shared Stage: Structured Mask Generation

CASS operationalizes the contribution scores in Sec. 3.2 by selecting task-relevant Transformer components and translating them into structured binary masks that can be applied to task-vector updates or gradients. For a given task $t$ with probe data $\mathcal{D}_t$, we perform a forward pass on a reference model to compute the importance score $\mathcal{I}_t^{(c)}$ for each component $c$, where $c$ denotes either an attention head or an FFN neuron. The choice of reference model and probe data depends on the application scenario and is detailed in Sec. 4.2 and Sec. 4.3, respectively.

Given the set of functional components $\Theta$, we retain the top-$k\%$ components with the highest scores as the task-specific active set. The binary mask is then defined as $\mathbf{M}_t^{(c)} = 1$ if component $c$ is selected and $\mathbf{M}_t^{(c)} = 0$ otherwise. To reflect the different granularities of attention heads and FFN neurons, we rank attention heads globally across layers, while ranking FFN neurons within each layer. This choice is motivated by the relatively small number of attention heads per layer, for which global ranking

yields more stable selection, and the much larger number of FFN neurons, for which layer-wise ranking provides a more balanced and discriminative allocation of active components.

**Choice of retention ratio.** Unless otherwise specified, we use a retention ratio of 20% ($k = 20$). This choice is motivated by the cumulative distribution functions of FFN neuron contribution scores in Fig. 8 in the appendix: more than 80% of the neurons have near-zero scores, suggesting that task-dependent FFN activity is concentrated in a relatively small subset of neurons. Similar sparsity patterns are observed in RoBERTa and ViT, in addition to Qwen. These observations motivate a sparse retention budget, while the sensitivity analysis in Sec. 5.4.3 provides empirical support for 20% as a shared default.

### 4.2 Post-hoc Application: CASS-Merging

CASS-Merging addresses the standard model merging setting, where $K$ independently fine-tuned expert models $\{\theta_t\}_{t=1}^{K}$ are given and the goal is to integrate them into a single multi-task model. Standard methods merge task vectors $\tau_t = \theta_t - \theta_{base}$ directly, which accumulate updates from components irrelevant to task $t$, leading to parameter interference.

**Mask Computation.** In this setting, the fine-tuned checkpoints $\{\theta_t\}$ are available. We therefore compute the structural masks using each fine-tuned model $\theta_t$ as the reference, running a forward pass on unlabeled probe data $\mathcal{D}_t$. This ensures that the identified components faithfully reflect the functional distribution of the fine-tuned model.

**Task Vector Filtering.** We utilize the structural masks as a functional denoising filter. Before merging, we project each raw task vector onto its identified active subspace:

$$\tilde{\tau}_t = \mathbf{M}_t \odot \tau_t \tag{4}$$

where $\odot$ denotes element-wise structured broadcasting multiplication. This operation zeros out the task vector at inactive component positions, effectively reverting non-essential heads and neurons back to the pre-trained state. The filtered vectors $\tilde{\tau}_t$ are then aggregated using standard Merge Operators (e.g., TIES, TSV, or simple summation):

$$\theta_{merged} = \theta_{base} + \text{MergeOp}\left(\{\tilde{\tau}_t\}_{t=1}^{K}\right), \tag{5}$$

By filtering out updates from components that are not prominently activated by task $t$, CASS-Merging reduces the probability of destructive collisions between task-specific updates.

### 4.3 Training-time Extension: CASS-Tuning

While CASS-Merging operates *reactively*—filtering interference after fine-tuning has occurred—we further propose **CASS-Tuning** as a *proactive* extension that mitigates interference at the source by constraining the fine-tuning process itself.

**Mask Computation.** In this setting, the fine-tuned checkpoints are not yet available. We therefore compute the structural masks using the pre-trained base model $\theta_{base}$ as the reference, running a forward pass on unlabeled training data $\mathcal{D}_t$. This is motivated by the latent capability hypothesis: fine-tuning primarily amplifies functional pathways already present in the pre-trained model, rather than creating entirely new circuits [1, 63]. The base model's activation response to task-specific data is therefore sufficient to identify the relevant subspace.

**Gradient Masking.** We use the mask $\mathbf{M}_t$ to constrain the optimization trajectory during fine-tuning of task $t$ by masking gradients for inactive components:

$$\mathbf{g}'_t = \mathbf{M}_t \odot \nabla_\theta \mathcal{L}_t \tag{6}$$

This ensures that parameter updates are strictly confined to the pre-identified active subspace. Since masks restrict updates to task-relevant components and empirically these active sets tend to have low overlap (as shown in Sec. 3.3), the resulting task vectors are encouraged to occupy more structurally separated regions of the parameter space. This structural separation reduces the need for complex conflict resolution during subsequent merging.

# 5 Experiments

## 5.1 Experimental Setup

**Datasets and Models.** We evaluate CASS across three benchmarks spanning vision and language modalities. For **Vision**, we adopt FusionBench [45] with ViT-B/32 and ViT-B/16 architectures pre-trained on CLIP [41], evaluating on 20 diverse image classification tasks. For **Language (Encoder-only)**, we evaluate on RoBERTa [31] across 8 natural language understanding tasks [50]. For **Language (Decoder-only)**, we follow the MergeBench protocol [16] with Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct [40], covering four capabilities: Instruction Following, Mathematics, Coding, and Safety. For brevity, *Instruct* is omitted from the model names in the subsequent tables. More details are provided in Appendix F.

**Baselines.** We evaluate CASS as a plug-and-play filter applied on existing model merging methods, spanning four categories: *(1) Linear merging*: Task Arithmetic (TA) [20]; *(2) Sparsification-based*: TIES-Merging (TIES) [60], DARE [62], PCB-Merging (PCB) [10]; *(3) Subspace-based*: TSV-Merging (TSV) [12], Iso-C [33]; *(4) Optimization-based*: WUDI-Merging (WUDI) [5], AdaMerging (Ada) [61]. AdaMerging is evaluated on vision tasks only, as its entropy-minimization objective is not well-defined for generative and regression tasks.

## 5.2 Main Results: CASS-Merging

**Vision Tasks.** Table 1 reports average accuracy across 20 tasks. CASS-Merging consistently improves all eight baselines on both architectures. Specifically, TA and DARE benefit most significantly, with gains up to $+4.97\%$ (TA, ViT-B/16) and $+4.87\%$ (DARE, ViT-B/16), as these methods lack structural awareness and CASS compensates by preserving functionally coherent units. Notably, CASS-Merging also improves strong baselines including subspace-based TSV-Merging ($+1.24\%$ on ViT-B/32) and optimization-based WUDI-Merging ($+1.40\%$ on ViT-B/16), suggesting that activation-based structured sparsity captures functional information complementary to weight-space decomposition. These results indicate that CASS is particularly effective for merge operators that do not explicitly model Transformer component structure, while remaining beneficial when combined with more advanced weight-space conflict-resolution methods.

Table 1: **Results on Vision Tasks.** Average accuracy (%) on 20 tasks.

| Method | ViT-B/32 | | ViT-B/16 | |
|---|---|---|---|---|
| | Base | w/ CASS-M | Base | w/ CASS-M |
| TA | 61.37 | 65.29 (+3.92) | 65.02 | 69.99 (+4.97) |
| TIES | 63.88 | 65.52 (+1.64) | 66.54 | 69.34 (+2.80) |
| DARE | 61.40 | 65.24 (+3.84) | 65.03 | 69.90 (+4.87) |
| PCB | 63.72 | 65.75 (+2.03) | 67.74 | 69.78 (+2.04) |
| TSV | 73.67 | 74.91 (+1.24) | 78.37 | 78.98 (+0.61) |
| Iso-C | 71.61 | 71.88 (+0.27) | 76.10 | 76.62 (+0.52) |
| Ada | 68.24 | 69.34 (+1.10) | 72.41 | 74.01 (+1.60) |
| WUDI | 72.29 | 73.51 (+1.22) | 75.73 | 77.13 (+1.40) |

Table 2: **Results on RoBERTa and Qwen2.5.** Average performance on 8 tasks for RoBERTa and average normalized performance on 4 tasks for Qwen2.5.

| Method | RoBERTa | | Qwen2.5-0.5B | | Qwen2.5-1.5B | |
|---|---|---|---|---|---|---|
| | Base | w/ CASS-M | Base | w/ CASS-M | Base | w/ CASS-M |
| Base Model | 47.00 | – | 0.720 | – | 0.768 | – |
| Fine-Tuned | 85.91 | – | 1.000 | – | 1.000 | – |
| Post-Mask | 79.33 | – | 0.910 | – | 0.975 | – |
| TA | 66.36 | 70.12 (+3.76) | 0.743 | 0.776 (+.034) | 0.789 | 0.837 (+.048) |
| TIES | 70.05 | 71.15 (+1.10) | 0.695 | 0.773 (+.078) | 0.803 | 0.829 (+.026) |
| DARE | 67.11 | 67.79 (+0.68) | 0.797 | 0.807 (+.010) | 0.842 | 0.871 (+.029) |
| PCB | 70.28 | 72.26 (+1.98) | 0.693 | 0.745 (+.052) | 0.791 | 0.823 (+.032) |
| TSV | 73.38 | 74.19 (+0.81) | 0.776 | 0.777 (+.001) | 0.831 | 0.847 (+.016) |
| Iso-C | 75.72 | 75.98 (+0.26) | 0.738 | 0.729 (-.009) | 0.809 | 0.814 (+.005) |
| WUDI | 73.65 | 74.21 (+0.56) | 0.717 | 0.776 (+.059) | 0.862 | 0.873 (+.011) |

**Language Tasks (RoBERTa).** Table 2 reports results on the 8-task RoBERTa benchmark. The "Post-Mask" row reports the single-task performance of fine-tuned experts after applying CASS

masks, serving as a diagnostic reference for how much task-specific capability is preserved by the selected subspace. CASS-Merging yields consistent improvements across all methods, with the largest gains on TA (+3.76%) and PCB-Merging (+1.98%). The complementary gains over the subspace-based method (TSV: +0.81%) and the optimization-based method (WUDI: +0.56%) confirm the cross-architecture generalization of structured activation-based filtering.

**Language Tasks (Qwen2.5).** Table 2 also reports results on the 4-task decoder-only benchmark. The high Post-Mask performance (0.910 / 0.975) suggests that task-specific knowledge is concentrated in a small subset of activation-prominent components, and that the remaining parameters are largely redundant. CASS-Merging improves most baselines across both model scales, with the only exception of Iso-C on Qwen2.5-0.5B. This suggests that CASS is broadly complementary to existing merging operators, while its interaction with spectrum-regularized subspace methods may depend on model scale and hyperparameter choices. Detailed per-task results are provided in Appendix G.1.

### 5.3 Extension Results: CASS-Tuning

CASS-Tuning constrains the fine-tuning process itself and is evaluated in the decoder-only setting where training access is available. As shown in Table 3, the "Masked FT" row reports the single-task performance when fine-tuning is restricted to the identified subspace, suggesting that the active components are sufficient to preserve most task-specific capabilities (0.954 / 0.924).

CASS-Tuning consistently improves all baselines, with the most notable gains for TIES (+0.121 on Qwen2.5-0.5B, +0.064 on Qwen2.5-1.5B). This is because gradient masking ensures any non-zero parameter update is almost task-relevant, converting TIES's heuristic assumption (the parameter magnitude is approximately equivalent to its importance) into a structural property. TA and WUDI-Merging both yield improvements exceeding 0.06 on Qwen2.5-0.5B. For Iso-C on 0.5B, the improvement is modest (+0.019), as the isotropic spectrum adjustment operates in the weight space and its interaction with subspace-constrained fine-tuning is limited at smaller model scales. Detailed per-task results are in Appendix G.2.

Table 3: **Results of CASS-Tuning** on Qwen2.5. Average normalized performance on 4 tasks.

| Method | Qwen2.5-0.5B | | Qwen2.5-1.5B | |
|---|---|---|---|---|
| | Base | w/ CASS-T | Base | w/ CASS-T |
| Base Model | 0.720 | – | 0.768 | – |
| Fine-Tuned | 1.000 | – | 1.000 | – |
| Masked FT | 0.954 | – | 0.924 | – |
| TA | 0.743 | 0.811 **(+.069)** | 0.789 | 0.820 **(+.031)** |
| TIES | 0.695 | 0.816 **(+.121)** | 0.803 | 0.867 **(+.064)** |
| DARE | 0.797 | 0.810 **(+.013)** | 0.842 | 0.850 **(+.008)** |
| PCB | 0.693 | 0.739 **(+.046)** | 0.791 | 0.826 **(+.035)** |
| TSV | 0.776 | 0.806 **(+.030)** | 0.831 | 0.859 **(+.028)** |
| Iso-C | 0.738 | 0.757 **(+.019)** | 0.809 | 0.825 **(+.016)** |
| WUDI | 0.717 | 0.790 **(+.073)** | 0.862 | 0.883 **(+.021)** |

### 5.4 Analysis

#### 5.4.1 OOD Generalization

Beyond in-distribution task performance, we evaluate whether CASS improves the general capabilities of the fusion model. Table 4 reports OOD results on two settings.

Table 4: **OOD Generalization** on ViT and Qwen.

| Method | 12-task ID | 8-task OOD |
|---|---|---|
| TA (ViT-B/32) | 0.663 | 0.570 |
| w/ CASS-M | 0.715 **(+.052)** | 0.588 **(+.018)** |
| TA (ViT-B/16) | 0.711 | 0.576 |
| w/ CASS-M | 0.780 **(+.069)** | 0.597 **(+.021)** |

(a) Vision OOD (ViT). 12 tasks merged; remaining 8 tasks as OOD.

| Method | MMLU | ↑ |
|---|---|---|
| TA (Qwen-1.5B) | 0.4531 | - |
| w/ CASS-M | 0.4563 | **+0.71%** |
| w/ CASS-T | 0.4603 | **+1.59%** |
| TA (Qwen-0.5B) | 0.4251 | - |
| w/ CASS-M | 0.4388 | **+3.22%** |
| w/ CASS-T | 0.4387 | **+3.20%** |

(b) Language OOD (Qwen2.5).

For **ViT**, we merge 12 in-distribution tasks and evaluate the resulting model on the remaining 8 held-out tasks. CASS-Merging improves both in-distribution and OOD performance on ViT-B/32 and ViT-B/16. This suggests that structured task-vector filtering reduces interference among the merged tasks while better preserving transferable visual representations.

For **Qwen2.5**, we evaluate the merged model on MMLU [18], which measures general knowledge and reasoning ability beyond the four task-specific fine-tuning domains. Both CASS-Merging and CASS-Tuning improve MMLU performance over standard Task Arithmetic on Qwen2.5-0.5B and Qwen2.5-1.5B. The gains are modest but consistent, indicating that restricting task-specific updates to activation-prominent components does not degrade general capability and may help preserve general-purpose knowledge during merging and mask-guided fine-tuning.

### 5.4.2 Probe Sample Size Sensitivity

CASS relies on a small unlabeled probe set to compute contribution statistics. Fig. 5 shows the average accuracy on ViT-B/32 and ViT-B/16 (TA method with CASS-Merging) as the probe size varies from 8 to 1024 samples per task. Performance remains remarkably stable across this range, with less than 0.5% variation on both architectures. This confirms that CASS is robust to the probe set size and that even a minimal probe of 8 samples per task is sufficient for reliable mask generation.

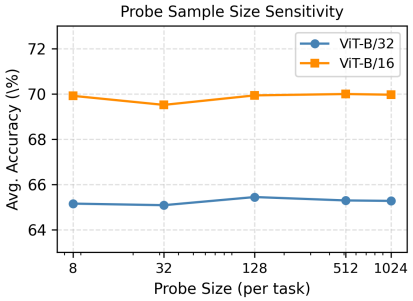


Figure 5: Probe sample size sensitivity.

**Computational Overhead.** The mask generation step of CASS introduces negligible computational overhead. Taking 8 unlabeled samples per task as an example, computing the activation statistics via a single forward pass and generating the binary masks requires only a few seconds on a single NVIDIA A100 GPU. This one-time cost is amortized over the merging process and is orders of magnitude smaller than the fine-tuning cost. Moreover, since CASS is applied as a preprocessing step, it does not introduce additional memory overhead during the subsequent model merging procedure. Once generated, the masks are reusable across different merging operators (e.g., TA, TIES, DARE) without recomputation. This makes CASS a lightweight and practical pre-filter that can be seamlessly integrated into a broad range of existing merging pipelines with minimal additional cost.

### 5.4.3 Retention Ratio Sensitivity

We evaluate the sensitivity of CASS-Merging to the retention ratio using Task Arithmetic on RoBERTa, ViT-B/32, and ViT-B/16. Table 5 reports results for retention ratios ranging from 1% to 100%, where 100% corresponds to Task Arithmetic without CASS filtering.

Retaining 10% or 20% of the components consistently improves performance over the unfiltered baseline across all three model configurations. At the default ratio of 20%, the gains are 3.76, 3.92, and 4.97 points for RoBERTa, ViT-B/32, and ViT-B/16, respectively. The best ratio varies across models: RoBERTa and ViT-B/16 achieve their highest scores at 10%, whereas ViT-B/32 peaks at 5%. Thus, 20% serves as an effective shared default, although it is not optimal for every model.

Reducing the retention ratio to 1% substantially degrades performance relative to 10% and 20% on all three models. This is consistent with overly aggressive filtering discarding task-relevant updates. Overall, these results suggest that the benefits of CASS-Merging persist across a moderate range of retention ratios, rather than depending on a single precise choice.

Table 5: Sensitivity of CASS-Merging to the retention ratio using Task Arithmetic.

| Model | 1% | 5% | 10% | 20% | 30% | 40% | 50% | 100% |
|---|---|---|---|---|---|---|---|---|
| RoBERTa | 52.58 | 62.49 | **70.61** | 70.12 | 70.12 | 69.02 | 68.94 | 66.36 |
| ViT-B/32 | 60.20 | **67.25** | 66.51 | 65.29 | 64.08 | 63.27 | 62.60 | 61.37 |
| ViT-B/16 | 65.99 | 70.95 | **71.49** | 69.99 | 67.99 | 67.11 | 66.37 | 65.02 |

### 5.4.4 Ablation Study

**Contribution-aware masks vs. random masks.** We first compare CASS with random structured masks under the same masking granularity and sparsity budget. As shown in Fig. 6, random masks do not consistently improve over the TA baseline. In particular, on RoBERTa, random masking even leads to a performance drop, indicating that indiscriminately removing task-vector components can discard task-relevant updates and amplify merging instability. In contrast, CASS consistently

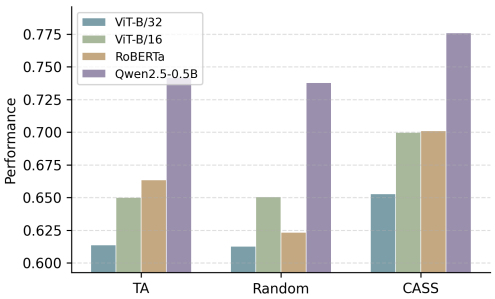



Figure 6: Comparison between TA, random structured masks, and CASS masks. Random masks use the same sparsity budget as CASS.

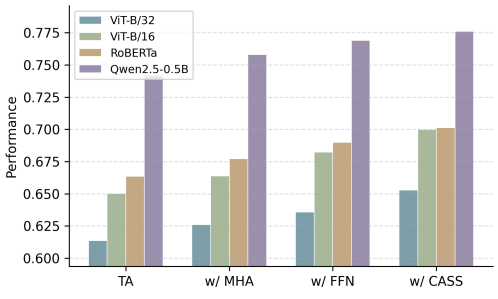



Figure 7: Component-wise ablation study. "w/ MHA" and "w/ FFN" denote applying CASS masks only to attention heads or FFN neurons, respectively.

improves TA across all evaluated benchmarks. This demonstrates that the gains of CASS are not merely caused by reducing the number of merged parameters, but by selectively preserving activation-prominent components that are more likely to carry task-specific functionality. The results support our central hypothesis that contribution-aware structured filtering can reduce inter-task conflicts more effectively than uninformative sparsity.

**Component-wise ablation.** We further isolate the effect of the two types of Transformer components by applying CASS masks only to attention heads (w/ MHA) or only to FFN neurons (w/ FFN). As shown in Fig. 7, both variants improve over TA, suggesting that task-specific activation patterns exist in both attention and FFN components.

Across the four benchmarks, FFN-only masking generally yields larger gains than MHA-only masking. For example, on ViT-B/32, w/ FFN improves TA by +2.20%, whereas w/ MHA improves it by +1.24%. This observation is consistent with the view that FFN neurons often act as sparse key-value memories and therefore contain more task-specific updates. Attention heads, in contrast, tend to serve broader information-routing roles, so head-level filtering provides smaller but complementary improvements. Finally, combining MHA and FFN masks achieves the best performance across all benchmarks. This indicates that attention heads and FFN neurons capture complementary aspects of task-specific functional organization, and that CASS benefits from filtering both types of structured components.

## 6 Conclusion

This paper presents CASS, a framework that mitigates parameter interference in model merging by identifying and preserving task-specific components. Grounded in the algebraic decomposition of Transformer layers, CASS generates structured binary masks via contribution-aware importance metrics, filtering out components irrelevant to each task before merging. CASS operates in two paradigms: CASS-Merging as a lightweight post-hoc filter applicable to any existing merging pipeline, and CASS-Tuning as a proactive training-time constraint designed to inherently reduce structural overlap between task vectors. Extensive experiments show that CASS improves most evaluated baselines across vision and language benchmarks, with particularly consistent gains for sparsification-based and linear merging methods. Limitations are discussed in Appendix H.

## Acknowledgment

This paper is supported by the Major Key Project of Pengcheng Laboratory (PCL2025A12), the National Natural Science Foundation of China (6253000218, 62236006), the China Postdoctoral Science Foundation (2025M781481).

## References


[1] A. Aghajanyan, S. Gupta, and L. Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, pages 7319–7328, 2021.

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[3] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101–mining discriminative components with random forests. In *European conference on computer vision*, pages 446–461. Springer, 2014.

[4] G. Cheng, J. Han, and X. Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

[5] R. Cheng, F. Xiong, Y. Wei, W. Zhu, and C. Yuan. Whoever started the interference should end it: Guiding data-free model merging via task vectors. *arXiv preprint arXiv:2503.08099*, 2025.

[6] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613, 2014.

[7] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[8] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[9] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[10] G. Du, J. Lee, J. Li, R. Jiang, Y. Guo, S. Yu, H. Liu, S. K. Goh, H.-K. Tang, D. He, et al. Parameter competition balancing for model merging. *Advances in Neural Information Processing Systems*, 37:84746–84776, 2024.

[11] L. Gao, A. Rajaram, J. Coxon, S. V. Govande, B. Baker, and D. Mossing. Weight-sparse transformers have interpretable circuits. *arXiv preprint arXiv:2511.13653*, 2025.

[12] A. A. Gargiulo, D. Crisostomi, M. S. Bucarelli, S. Scardapane, F. Silvestri, and E. Rodola. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18695–18705, 2025.

[13] M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, 2021.

[14] I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, et al. Challenges in representation learning: A report on three machine learning contests. In *International conference on neural information processing*, pages 117–124. Springer, 2013.

[15] S. Han, K. Rao, A. Ettinger, L. Jiang, B. Y. Lin, N. Lambert, Y. Choi, and N. Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *Advances in Neural Information Processing Systems*, 37:8093–8131, 2024.

[16] Y. He, S. Zeng, Y. Hu, R. Yang, T. Zhang, and H. Zhao. Mergebench: A benchmark for merging domain-specialized llms. *arXiv preprint arXiv:2505.10833*, 2025.

[17] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[18] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[19] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[20] G. Ilharco, M. T. Ribeiro, M. Wortsman, S. Gururangan, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[21] T. Jiang, X. Yu, C. Yi, Y. Wu, Y. Li, R. Cheng, D. Jiang, and J. Zhang. Evogm: Learning to merge llms via evolutionary generative optimization. *arXiv preprint arXiv:2605.29295*, 2026.

[22] X. Jin, X. Ren, D. Preotiuc-Pietro, and P. Cheng. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849*, 2022.

[23] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

[24] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

[27] Y. Li, G. Cao, Y. Song, M. Tao, H. Gong, J. Liu, Y. Wang, and D. Jiang. Duetmerging: Synergizing dynamic and static strategies for mitigating task interference in model merging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41954–41963, 2026.

[28] Y. Li, W. Gan, K. Lu, D. Jiang, and R. Jain. Aves: An audio-visual emotion stream dataset for temporal emotion detection. *IEEE Transactions on Affective Computing*, 16(1):438–450, 2024.

[29] Y. Li, X. Lan, H. Chen, K. Lu, and D. Jiang. Multimodal pear chain-of-thought reasoning for multimodal sentiment analysis. *ACM Transactions on Multimedia Computing, Communications and Applications*, 20(9):1–23, 2025.

[30] Y. Li, Y. Xing, X. Lan, X. Li, H. Chen, and D. Jiang. Alignmamba: Enhancing multimodal mamba with local and global cross-modal alignment. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 24774–24784. IEEE, 2025.

[31] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[32] Z. Lu, C. Fan, W. Wei, X. Qu, D. Chen, and Y. Cheng. Twin-merging: Dynamic integration of modular expertise in model merging. *Advances in Neural Information Processing Systems*, 37:78905–78935, 2024.

[33] D. Marczak, S. Magistri, S. Cygert, B. Twardowski, A. D. Bagdanov, and J. Van De Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. *arXiv preprint arXiv:2502.04959*, 2025.

[34] M. S. Matena and C. A. Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

[35] P. Michel, O. Levy, and G. Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[36] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 7. Granada, 2011.

[37] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE, 2008.

[38] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[39] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[40] Qwen. Qwen2.5 technical report, 2025.

[41] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[42] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[43] N. Sofroniew, I. Kauvar, W. Saunders, R. Chen, T. Henighan, S. Hydrie, C. Citro, A. Pearce, J. Tarng, W. Gurnee, et al. Emotion concepts and their function in a large language model. *arXiv preprint arXiv:2604.07729*, 2026.

[44] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[45] A. Tang, L. Shen, Y. Luo, H. Hu, B. Du, and D. Tao. Fusionbench: A comprehensive benchmark of deep model fusion. *arXiv preprint arXiv:2406.03280*, 2024.

[46] A. Tang, L. Shen, Y. Luo, N. Yin, L. Zhang, and D. Tao. Merging multi-task models via weight-ensembling mixture of experts. *arXiv preprint arXiv:2402.00433*, 2024.

[47] Y. Tong, X. Zhang, R. Wang, R. Wu, and J. He. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *Advances in Neural Information Processing Systems*, 37:7821–7846, 2024.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[49] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, and M. Welling. Rotation equivariant cnns for digital pathology. In *International Conference on Medical image computing and computer-assisted intervention*, pages 210–218. Springer, 2018.

[50] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pages 353–355, 2018.

[51] K. Wang, N. Dimitriadis, G. Ortiz-Jimenez, F. Fleuret, and P. Frossard. Localizing task information for improved model merging and compression. *arXiv preprint arXiv:2405.07813*, 2024.

[52] Y. Wei, Z. Wang, J. Liu, Y. Ding, and L. Zhang. Magicoder: Empowering code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

[53] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

[54] W. Wu, Y. Wang, G. Xiao, H. Peng, and Y. Fu. Retrieval head mechanistically explains long-context factuality. *arXiv preprint arXiv:2404.15574*, 2024.

[55] C. Xiao, Z. Zhang, C. Song, D. Jiang, F. Yao, X. Han, X. Wang, S. Wang, Y. Huang, G. Lin, et al. Configurable foundation models: Building llms from a modular perspective. *arXiv preprint arXiv:2409.02877*, 2024.

[56] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[57] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119(1):3–22, 2016.

[58] L. Xiao, G. Cao, M. Guo, X. Guan, F. Yang, M. Tao, X. Li, Y. Peng, and Y. Wang. A survey on the green development of large models: From resource-efficient architectures to hardware-software co-design. *arXiv preprint arXiv:2607.09084*, 2026.

[59] M. Xu, Z. Chen, W. Fu, Y. Li, S. Wang, and J. Wang. Expected returns and policy inconsistency-aware offline federated deep reinforcement learning. In *43rd International Conference on Machine Learning (ICML 2025)*. PMLR, 2026.

[60] P. Yadav, D. Tam, L. Choshen, C. A. Raffel, and M. Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023.

[61] E. Yang, Z. Wang, L. Shen, S. Liu, G. Guo, X. Wang, and D. Tao. Adamerging: Adaptive model merging for multi-task learning. *arXiv preprint arXiv:2310.02575*, 2023.

[62] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

[63] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, et al. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021, 2023.

[64] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

# Appendix Table of Contents


- **Appendix A: Detailed Derivation of the Unified Summation Form**
  Provides the detailed derivation of the additive summation forms for MHA and FFN, establishing the algebraic basis for component-level structured masking.

- **Appendix B: Algorithms**
  Summarizes the complete procedures of CASS-Merging and CASS-Tuning, including mask estimation, task-vector filtering, and mask-guided fine-tuning.

- **Appendix C: Mask Construction and Broadcasting**
  Specifies how component-level masks over attention heads and FFN neurons are broadcast to parameter-level task-vector updates or gradients.

- **Appendix D: Extended Analysis on FFN**
  Presents additional analyses of FFN neuron contribution sparsity and cross-task overlap across different task domains and layers.

- **Appendix E: Extended Analysis on MHA**
  Analyzes task-selective attention heads, including selectivity-score distributions, cross-task overlap, and depth-wise localization.

- **Appendix F: Experimental Settings**
  Describes the evaluated tasks, models, datasets, metrics, fine-tuning hyperparameters, normalization protocol, and random seed configuration.

- **Appendix G: Detailed Qwen2.5 Results**
  Reports detailed per-capability results for CASS-Merging and CASS-Tuning on Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct.

- **Appendix H: Limitations**
  Discusses current limitations of CASS, including fixed sparsity ratios and evaluation scale.

# A Detailed Derivation of the Unified Summation Form

In Section 3.1, we introduced a unified summation view of Transformer sub-layers to motivate structured component-level filtering. Here, we provide a more detailed derivation for Multi-Head Attention (MHA) and Feed-Forward Networks (FFNs), explicitly accounting for tensor dimensions and bias terms.

This appendix has two purposes. First, we show that, under a fixed set of model parameters, attention heads and FFN neurons contribute additive terms to the residual stream. This provides the basis for defining contribution-based component scores. Second, we clarify how CASS applies the resulting component masks to task-specific parameter updates. Importantly, CASS does not perform output-level component deletion. Instead, it masks task-vector updates or gradients at the parameter level, so that inactive components retain their pre-trained parameters.

For clarity, we present the derivation using a sequence input $\mathbf{X} \in \mathbb{R}^{L\times d}$, where $L$ is the sequence length and $d$ is the hidden dimension. The single-token formulation used in the main text can be viewed as the special case $L = 1$.

## A.1 Multi-Head Attention

Consider an MHA layer with $H$ attention heads and head dimension $d_h = d/H$. For the $i$-th head, the query, key, and value projections are $\mathbf{W}_Q^{(i)}, \mathbf{W}_K^{(i)}, \mathbf{W}_V^{(i)} \in \mathbb{R}^{d\times d_h}$. The output of head $i$ is given by

$$\mathbf{H}_i(\mathbf{X}) = \mathrm{Attn}\left(\mathbf{X}\mathbf{W}_Q^{(i)}, \mathbf{X}\mathbf{W}_K^{(i)}, \mathbf{X}\mathbf{W}_V^{(i)}\right) \in \mathbb{R}^{L\times d_h}. \tag{7}$$

The standard MHA output is obtained by concatenating all head outputs and applying the output projection:

$$\mathrm{MHA}(\mathbf{X}) = \mathrm{Concat}\left(\mathbf{H}_1(\mathbf{X}), \ldots, \mathbf{H}_H(\mathbf{X})\right)\mathbf{W}_O + \mathbf{b}_O, \tag{8}$$

where $\mathbf{W}_O \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_O \in \mathbb{R}^d$ is broadcast along the sequence dimension.

To expose the additive structure, we decompose the output projection matrix $\mathbf{W}_O$ into $H$ row blocks:

$$\mathbf{W}_O = \begin{bmatrix} \mathbf{W}_O^{(1)} \\ \vdots \\ \mathbf{W}_O^{(H)} \end{bmatrix}, \quad \mathbf{W}_O^{(i)} \in \mathbb{R}^{d_h\times d}. \tag{9}$$

Then, by block matrix multiplication,

$$\begin{aligned} &\mathrm{Concat}\left(\mathbf{H}_1(\mathbf{X}), \ldots, \mathbf{H}_H(\mathbf{X})\right)\mathbf{W}_O \\ &= \begin{bmatrix}\mathbf{H}_1(\mathbf{X}) & \cdots & \mathbf{H}_H(\mathbf{X})\end{bmatrix} \begin{bmatrix} \mathbf{W}_O^{(1)} \\ \vdots \\ \mathbf{W}_O^{(H)} \end{bmatrix} \\ &= \sum_{i=1}^{H} \mathbf{H}_i(\mathbf{X})\mathbf{W}_O^{(i)}. \end{aligned} \tag{10}$$

Therefore, for a fixed model, the MHA output can be written as

$$\mathrm{MHA}(\mathbf{X}) = \sum_{i=1}^{H} \mathbf{H}_i(\mathbf{X})\mathbf{W}_O^{(i)} + \mathbf{b}_O. \tag{11}$$

Equation (11) shows that each attention head contributes an additive term $\mathbf{H}_i(\mathbf{X})\mathbf{W}_O^{(i)}$ to the residual stream under a fixed parameterization. CASS uses this additive term to estimate the functional contribution of a head. For example, the head-level contribution can be measured by

$$\mathcal{A}_{i,t} = \mathbb{E}_{\mathbf{X}\sim\mathcal{D}_t}\left[\left\|\mathbf{H}_i(\mathbf{X})\mathbf{W}_O^{(i)}\right\|_F\right], \tag{12}$$

where the activations and weights are taken from the reference model used for mask estimation.

**Parameter-level masking in CASS.** The above additive decomposition should not be confused with output-level component deletion. In CASS, a head mask is applied to task-specific parameter updates, not to the final head output. Let $\theta_0$ denote the base model and $\theta_t = \theta_0 + \tau_t$ denote the fine-tuned expert for task $t$. For head $i$, the task-vector updates are

$$\begin{aligned}
\Delta\mathbf{W}_{Q,t}^{(i)} &= \mathbf{W}_{Q,t}^{(i)} - \mathbf{W}_{Q,0}^{(i)}, \\
\Delta\mathbf{W}_{K,t}^{(i)} &= \mathbf{W}_{K,t}^{(i)} - \mathbf{W}_{K,0}^{(i)}, \\
\Delta\mathbf{W}_{V,t}^{(i)} &= \mathbf{W}_{V,t}^{(i)} - \mathbf{W}_{V,0}^{(i)}, \\
\Delta\mathbf{W}_{O,t}^{(i)} &= \mathbf{W}_{O,t}^{(i)} - \mathbf{W}_{O,0}^{(i)}.
\end{aligned} \tag{13}$$

Given a binary head mask $M_{\text{head}}^{(i,t)} \in \{0,1\}$, CASS constructs the masked task-specific parameters as

$$\begin{aligned}
\widetilde{\mathbf{W}}_{Q,t}^{(i)} &= \mathbf{W}_{Q,0}^{(i)} + M_{\text{head}}^{(i,t)}\Delta\mathbf{W}_{Q,t}^{(i)}, \\
\widetilde{\mathbf{W}}_{K,t}^{(i)} &= \mathbf{W}_{K,0}^{(i)} + M_{\text{head}}^{(i,t)}\Delta\mathbf{W}_{K,t}^{(i)}, \\
\widetilde{\mathbf{W}}_{V,t}^{(i)} &= \mathbf{W}_{V,0}^{(i)} + M_{\text{head}}^{(i,t)}\Delta\mathbf{W}_{V,t}^{(i)}, \\
\widetilde{\mathbf{W}}_{O,t}^{(i)} &= \mathbf{W}_{O,0}^{(i)} + M_{\text{head}}^{(i,t)}\Delta\mathbf{W}_{O,t}^{(i)}.
\end{aligned} \tag{14}$$

The resulting head output is then computed using these masked parameters:

$$\widetilde{\mathbf{H}}_{i,t}(\mathbf{X}) = \text{Attn}\left(\mathbf{X}\widetilde{\mathbf{W}}_{Q,t}^{(i)}, \mathbf{X}\widetilde{\mathbf{W}}_{K,t}^{(i)}, \mathbf{X}\widetilde{\mathbf{W}}_{V,t}^{(i)}\right). \tag{15}$$

Thus, the CASS-filtered MHA layer is

$$\widetilde{\text{MHA}}_t(\mathbf{X}) = \sum_{i=1}^{H} \widetilde{\mathbf{H}}_{i,t}(\mathbf{X})\widetilde{\mathbf{W}}_{O,t}^{(i)} + \widetilde{\mathbf{b}}_{O,t}. \tag{16}$$

This formulation is the exact parameter-level operation used by CASS. When $M_{\text{head}}^{(i,t)} = 0$, the task-specific update of head $i$ is removed, and the head reverts to its pre-trained parameters. When $M_{\text{head}}^{(i,t)} = 1$, the task-specific update of the head is retained. Because attention is nonlinear in the query and key projections, Eq. (16) is not algebraically equivalent to multiplying the fine-tuned head output by a scalar mask. The mask is applied to parameter updates rather than to the already-computed output contribution.

### A.2 Feed-Forward Networks

We next consider a standard FFN layer with an up-projection $\mathbf{W}_{up} \in \mathbb{R}^{d\times d_{ff}}$, a down-projection $\mathbf{W}_{down} \in \mathbb{R}^{d_{ff}\times d}$, and biases $\mathbf{b}_{up} \in \mathbb{R}^{d_{ff}}$ and $\mathbf{b}_{down} \in \mathbb{R}^{d}$. Given sequence input $\mathbf{X} \in \mathbb{R}^{L\times d}$, the intermediate activation is

$$\mathbf{M} = \sigma\left(\mathbf{X}\mathbf{W}_{up} + \mathbf{b}_{up}\right) \in \mathbb{R}^{L\times d_{ff}}, \tag{17}$$

where $\sigma(\cdot)$ denotes the element-wise nonlinearity. The FFN output is

$$\text{FFN}(\mathbf{X}) = \mathbf{M}\mathbf{W}_{down} + \mathbf{b}_{down}. \tag{18}$$

Let $\mathbf{m}_j \in \mathbb{R}^L$ denote the $j$-th column of $\mathbf{M}$, i.e., the activation of neuron $j$ across all sequence positions. Let $\mathbf{w}_{down}^{(j)} \in \mathbb{R}^d$ denote the $j$-th row of $\mathbf{W}_{down}$. Then the matrix product can be decomposed as

$$\mathbf{M}\mathbf{W}_{down} = \sum_{j=1}^{d_{ff}} \mathbf{m}_j \mathbf{w}_{down}^{(j)}, \tag{19}$$

where each term $\mathbf{m}_j\mathbf{w}_{down}^{(j)} \in \mathbb{R}^{L\times d}$ is the additive residual-stream contribution of neuron $j$ across the sequence.

Therefore, for a fixed model, the FFN output can be written as

$$\text{FFN}(\mathbf{X}) = \sum_{j=1}^{d_{ff}} \mathbf{m}_j \mathbf{w}_{down}^{(j)} + \mathbf{b}_{down}. \tag{20}$$

Equation (20) motivates the FFN neuron contribution score used by CASS:

$$\mathcal{I}^{(j,t)}_{\text{neuron}} = \mathbb{E}_{\mathbf{X}\sim\mathcal{D}_t}\left[\left\|\mathbf{m}_j(\mathbf{X})\mathbf{w}^{(j)}_{down}\right\|_F\right]. \tag{21}$$

In practice, this can be computed using an equivalent activation-weighted output-norm form.

**Parameter-level masking in CASS.** As with attention heads, CASS does not delete FFN neurons from the model. Instead, it masks the task-specific updates associated with the corresponding intermediate channel. Let the task-vector updates for neuron $j$ be

$$\begin{aligned}\Delta\mathbf{W}_{up,t}[:,j] &= \mathbf{W}_{up,t}[:,j] - \mathbf{W}_{up,0}[:,j],\\ \Delta\mathbf{W}_{down,t}[j,:] &= \mathbf{W}_{down,t}[j,:] - \mathbf{W}_{down,0}[j,:].\end{aligned} \tag{22}$$

Given a binary neuron mask $M^{(j,t)}_{\text{neuron}} \in \{0,1\}$, CASS constructs

$$\begin{aligned}\widetilde{\mathbf{W}}_{up,t}[:,j] &= \mathbf{W}_{up,0}[:,j] + M^{(j,t)}_{\text{neuron}}\Delta\mathbf{W}_{up,t}[:,j],\\ \widetilde{\mathbf{W}}_{down,t}[j,:] &= \mathbf{W}_{down,0}[j,:] + M^{(j,t)}_{\text{neuron}}\Delta\mathbf{W}_{down,t}[j,:].\end{aligned} \tag{23}$$

If the up-projection bias is present and updated, the corresponding bias channel is masked in the same way:

$$\widetilde{\mathbf{b}}_{up,t}[j] = \mathbf{b}_{up,0}[j] + M^{(j,t)}_{\text{neuron}}\Delta\mathbf{b}_{up,t}[j]. \tag{24}$$

The resulting CASS-filtered intermediate activation is

$$\widetilde{\mathbf{m}}_{j,t}(\mathbf{X}) = \sigma\left(\mathbf{X}\widetilde{\mathbf{W}}_{up,t}[:,j] + \widetilde{\mathbf{b}}_{up,t}[j]\right), \tag{25}$$

and the CASS-filtered FFN output is

$$\widetilde{\text{FFN}}_t(\mathbf{X}) = \sum_{j=1}^{d_{ff}} \widetilde{\mathbf{m}}_{j,t}(\mathbf{X})\widetilde{\mathbf{W}}_{down,t}[j,:] + \widetilde{\mathbf{b}}_{down,t}. \tag{26}$$

When $M^{(j,t)}_{\text{neuron}} = 0$, neuron $j$ is not removed. Instead, its task-specific update is suppressed, and the neuron uses its pre-trained up- and down-projection parameters. When $M^{(j,t)}_{\text{neuron}} = 1$, the task-specific update associated with that neuron is retained. Because the activation $\widetilde{\mathbf{m}}_{j,t}(\mathbf{X})$ depends nonlinearly on the masked up-projection parameters, this operation cannot in general be written as a scalar multiplication of the fine-tuned neuron's final contribution. The mask is applied to the parameter update that defines the neuron, not to the output contribution after the forward pass.

**Summary.** The summation forms in Eq. (11) and Eq. (20) are used to define contribution scores and identify task-relevant components under a fixed reference model. The actual CASS operation is parameter-level task-vector filtering, as shown in Eq. (14) and Eq. (23). Thus, inactive components retain their pre-trained behavior rather than being deleted from the network.

# B Algorithms

In this section, we provide the detailed procedures of CASS-Merging and CASS-Tuning. One implementation detail is worth noting. FFN neuron scores are computed independently for each task, while attention head scores use a task-selectivity criterion that depends on the global average contribution of each head across the task set. Therefore, head-mask construction is performed after collecting head contribution statistics from all tasks. This introduces a task-set-level dependency, which is natural in the standard model merging setting where all task experts or task datasets are available before merging. The global average is computed only from unlabeled probe activations and does not use task labels.

## B.1 CASS-Merging

Algorithm 1 summarizes the CASS-Merging procedure. The algorithm first collects component statistics for all tasks. It then computes the global head contribution baseline and constructs task-specific masks. This two-stage structure makes explicit that attention head selectivity is defined relative to the current set of tasks being merged.

**Algorithm 1** CASS-Merging

**Require:** Base model $\theta_{base}$; fine-tuned experts $\{\theta_t\}_{t=1}^{K}$; unlabeled probe sets $\{\mathcal{D}_t\}_{t=1}^{K}$; retention ratios $k_{\text{head}}$ and $k_{\text{ffn}}$; merge operator $\text{MergeOp}(\cdot)$.

**Ensure:** Merged model $\theta_{merged}$.

**Stage 1: Collect task-specific component statistics.**

**for** each task $t = 1, \ldots, K$ **do**

Compute the task vector:

$$\tau_t = \theta_t - \theta_{base}.$$

Use the fine-tuned expert $\theta_t$ as the reference model.

Run a forward pass on $\mathcal{D}_t$ and collect component activations.

Compute FFN neuron contribution scores $\mathcal{I}_{\text{neuron}}^{(\ell,j,t)}$ for each layer $\ell$ and neuron $j$.

Compute absolute attention head contributions $\mathcal{A}_{h,t}$ for each attention head $h$.

**end for**

Compute the global average contribution of each attention head:

$$\mu_{h,\text{global}} = \frac{1}{K} \sum_{t=1}^{K} \mathcal{A}_{h,t}.$$

**Stage 2: Construct masks and filter task vectors.**

**for** each task $t = 1, \ldots, K$ **do**

Compute task-selectivity scores for attention heads:

$$\mathcal{S}_{h,t} = \frac{\mathcal{A}_{h,t} - \mu_{h,\text{global}}}{\mu_{h,\text{global}} + \epsilon}.$$

Select the top-$k_{\text{ffn}}$ FFN neurons within each layer according to $\mathcal{I}_{\text{neuron}}^{(\ell,j,t)}$.

Select the top-$k_{\text{head}}$ attention heads across the model according to positive task-selectivity scores $\mathcal{S}_{h,t}$.

Construct the structured binary mask $\mathbf{M}_t$ over selected attention heads and FFN neurons.

Broadcast $\mathbf{M}_t$ to the corresponding parameter slices of $\tau_t$.

Filter the task vector:

$$\tilde{\tau}_t = \mathbf{M}_t \odot \tau_t.$$

**end for**

Merge the filtered task vectors:

$$\theta_{merged} = \theta_{base} + \text{MergeOp}\left(\{\tilde{\tau}_t\}_{t=1}^{K}\right).$$

**return** $\theta_{merged}$.

### B.2 CASS-Tuning

Algorithm 2 summarizes the CASS-Tuning procedure. Since fine-tuned experts are not yet available, masks are estimated using the base model before task-specific fine-tuning. As in CASS-Merging, attention head selectivity is computed relative to the global average contribution across the task set.

**Algorithm 2** CASS-Tuning

**Require:** Base model $\theta_{base}$; task training sets $\{\mathcal{D}_t\}_{t=1}^{K}$; retention ratios $k_{\text{head}}$ and $k_{\text{ffn}}$; learning rate $\eta$; number of training steps $S$; merge operator $\text{MergeOp}(\cdot)$.

**Ensure:** Merged model $\theta_{merged}$.

**Stage 1: Estimate masks from the base model.**

**for** each task $t = 1, \ldots, K$ **do**

  Use the base model $\theta_{base}$ as the reference model.

  Run a forward pass on unlabeled samples from $\mathcal{D}_t$ and collect component activations.

  Compute FFN neuron contribution scores $\mathcal{I}_{\text{neuron}}^{(\ell,j,t)}$ for each layer $\ell$ and neuron $j$.

  Compute absolute attention head contributions $\mathcal{A}_{h,t}$ for each attention head $h$.

**end for**

Compute the global average contribution of each attention head:

$$\mu_{h,\text{global}} = \frac{1}{K} \sum_{t=1}^{K} \mathcal{A}_{h,t}.$$

**for** each task $t = 1, \ldots, K$ **do**

  Compute task-selectivity scores for attention heads:

$$\mathcal{S}_{h,t} = \frac{\mathcal{A}_{h,t} - \mu_{h,\text{global}}}{\mu_{h,\text{global}} + \epsilon}.$$

  Select the top-$k_{\text{ffn}}$ FFN neurons within each layer according to $\mathcal{I}_{\text{neuron}}^{(\ell,j,t)}$.

  Select the top-$k_{\text{head}}$ attention heads according to task-selectivity scores $\mathcal{S}_{h,t}$.

  Construct the structured binary mask $\mathbf{M}_t$.

**end for**

**Stage 2: Fine-tune each task under its structured mask.**

**for** each task $t = 1, \ldots, K$ **do**

  Initialize the task-specific model:

$$\theta_t^{(0)} = \theta_{base}.$$

  **for** training step $s = 0, \ldots, S-1$ **do**

    Sample a mini-batch $\mathcal{B}_t \subset \mathcal{D}_t$.

    Compute the gradient and apply gradient masking:

$$\tilde{\mathbf{g}}_t^{(s)} = \mathbf{M}_t \odot \mathbf{g}_t^{(s)}.$$

    Update the active subspace:

$$\theta_t^{(s+1)} = \theta_t^{(s)} - \eta \tilde{\mathbf{g}}_t^{(s)}.$$

  **end for**

  Obtain the CASS-tuned task vector:

$$\tilde{\tau}_t = \theta_t^{(S)} - \theta_{base}.$$

**end for**

Merge the CASS-tuned task vectors:

$$\theta_{merged} = \theta_{base} + \text{MergeOp}\left(\{\tilde{\tau}_t\}_{t=1}^{K}\right).$$

**return** $\theta_{merged}$.

# C Mask Construction and Broadcasting

CASS defines binary masks at the level of functional components, i.e., attention heads and FFN neurons. However, both task-vector filtering in CASS-Merging and gradient masking in CASS-Tuning are implemented at the parameter level. This section specifies how component-level masks are broadcast to the corresponding parameter slices.

Throughout this section, let $\Delta\mathbf{W}$ denote either a task-vector update in CASS-Merging or a gradient/update tensor in CASS-Tuning. For CASS-Merging, $\Delta\mathbf{W}$ corresponds to the parameter difference between a fine-tuned expert and the base model. For CASS-Tuning, $\Delta\mathbf{W}$ corresponds to the gradient or optimizer update at a given training step. In both cases, a zero mask suppresses only the task-specific update associated with a component. It does not remove the component from the pre-trained model. Therefore, inactive components retain their pre-trained parameters after merging or masked fine-tuning.

## C.1 Attention Head Masks

Consider a Transformer layer $\ell$ with $H$ attention heads and head dimension $d_h = d/H$. For head $i$, CASS assigns a binary mask $M^{(\ell,i)}_{\text{head}} \in \{0,1\}$. The corresponding projection matrices are

$$\mathbf{W}^{(\ell,i)}_{Q}, \mathbf{W}^{(\ell,i)}_{K}, \mathbf{W}^{(\ell,i)}_{V} \in \mathbb{R}^{d\times d_h}, \quad \mathbf{W}^{(\ell,i)}_{O} \in \mathbb{R}^{d_h\times d}, \tag{27}$$

where $\mathbf{W}^{(\ell,i)}_{O}$ is the output-projection block associated with head $i$.

The head mask is broadcast to all parameter slices associated with this head:

$$\begin{aligned}
\Delta\widetilde{\mathbf{W}}^{(\ell,i)}_{Q} &= M^{(\ell,i)}_{\text{head}} \cdot \Delta\mathbf{W}^{(\ell,i)}_{Q},\\
\Delta\widetilde{\mathbf{W}}^{(\ell,i)}_{K} &= M^{(\ell,i)}_{\text{head}} \cdot \Delta\mathbf{W}^{(\ell,i)}_{K},\\
\Delta\widetilde{\mathbf{W}}^{(\ell,i)}_{V} &= M^{(\ell,i)}_{\text{head}} \cdot \Delta\mathbf{W}^{(\ell,i)}_{V},\\
\Delta\widetilde{\mathbf{W}}^{(\ell,i)}_{O} &= M^{(\ell,i)}_{\text{head}} \cdot \Delta\mathbf{W}^{(\ell,i)}_{O}.
\end{aligned} \tag{28}$$

If the attention projections contain bias terms, the same mask is applied to the head-specific slices of the query, key, and value biases. The output-projection bias is not decomposed by head and is therefore left unmasked unless otherwise specified.

This broadcasting rule ensures that when a head is inactive for task $t$, CASS suppresses the task-specific update associated with the entire attention head. In CASS-Merging, the corresponding task-vector slices are zeroed out before applying the merge operator. In CASS-Tuning, the corresponding gradient slices are zeroed out during fine-tuning.

## C.2 FFN Neuron Masks

We next consider a standard FFN layer with an up-projection $\mathbf{W}^{(\ell)}_{up} \in \mathbb{R}^{d\times d_{ff}}$ and a down-projection $\mathbf{W}^{(\ell)}_{down} \in \mathbb{R}^{d_{ff}\times d}$. For neuron $j$ in layer $\ell$, CASS assigns a binary mask $M^{(\ell,j)}_{\text{neuron}} \in \{0,1\}$.

Since neuron $j$ corresponds to the $j$-th intermediate channel, its parameters consist of the $j$-th column of the up-projection and the $j$-th row of the down-projection. The neuron mask is therefore broadcast as

$$\begin{aligned}
\Delta\widetilde{\mathbf{W}}^{(\ell)}_{up}[:,j] &= M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{up}[:,j],\\
\Delta\widetilde{\mathbf{W}}^{(\ell)}_{down}[j,:] &= M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{down}[j,:].
\end{aligned} \tag{29}$$

If the FFN uses an up-projection bias $\mathbf{b}^{(\ell)}_{up}$, we also apply the same mask to its corresponding channel:

$$\Delta\widetilde{\mathbf{b}}^{(\ell)}_{up}[j] = M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{b}^{(\ell)}_{up}[j]. \tag{30}$$

The down-projection bias $\mathbf{b}^{(\ell)}_{down}$ is shared across all FFN neurons and is not naturally attributable to a single intermediate channel. We therefore leave it unmasked in our implementation.

This rule is consistent with the additive FFN decomposition in Appendix A. Masking neuron $j$ suppresses task-specific updates to the parameters that generate and project the corresponding intermediate activation channel, while preserving the base model parameters of that neuron.

### C.3 Gated FFNs

Modern decoder-only language models, including Qwen and Gemma-style architectures, commonly use gated FFNs such as SwiGLU. In this case, the intermediate activation is computed from a gate projection and an up projection:

$$\mathbf{M} = \mathrm{SiLU}\left(\mathbf{X}\mathbf{W}_{gate}\right) \odot \left(\mathbf{X}\mathbf{W}_{up}\right), \tag{31}$$

where $\mathbf{W}_{gate}, \mathbf{W}_{up} \in \mathbb{R}^{d \times d_{ff}}$. The output is then obtained through $\mathbf{W}_{down} \in \mathbb{R}^{d_{ff} \times d}$.

Although gated FFNs use a different mechanism to compute the intermediate activation, each intermediate channel $j$ still contributes to the residual stream through the corresponding row of $\mathbf{W}_{down}$. Therefore, the same neuron-level mask can be applied to the triplet of parameter slices associated with channel $j$:

$$\begin{aligned}
\Delta\widetilde{\mathbf{W}}^{(\ell)}_{gate}[:, j] &= M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{gate}[:, j], \\
\Delta\widetilde{\mathbf{W}}^{(\ell)}_{up}[:, j] &= M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{up}[:, j], \\
\Delta\widetilde{\mathbf{W}}^{(\ell)}_{down}[j, :] &= M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{down}[j, :].
\end{aligned} \tag{32}$$

If gate or up projection biases are present, the same mask is applied to the corresponding bias channel. This ensures that the entire gated intermediate channel is treated as one functional unit.

### C.4 LoRA Implementation

CASS can also be used with parameter-efficient fine-tuning methods such as LoRA. In this case, the task-specific update to a weight matrix is represented by a low-rank update $\Delta\mathbf{W}_{\mathrm{LoRA}}$, rather than by directly fine-tuning the full-rank matrix.

The key principle is that CASS applies the structural mask to the effective update along the same channel dimension as in the full-rank case. For example, for an FFN down-projection, neuron $j$ corresponds to the $j$-th input channel of $\mathbf{W}_{down}$. Therefore, CASS masks the effective LoRA update on the same row:

$$\Delta\widetilde{\mathbf{W}}^{(\ell)}_{\mathrm{LoRA},down}[j, :] = M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{\mathrm{LoRA},down}[j, :]. \tag{33}$$

Similarly, for an up or gate projection, neuron $j$ corresponds to the $j$-th output channel, and the mask is applied to the corresponding column of the effective LoRA update:

$$\Delta\widetilde{\mathbf{W}}^{(\ell)}_{\mathrm{LoRA},up}[:, j] = M^{(\ell,j)}_{\text{neuron}} \cdot \Delta\mathbf{W}^{(\ell)}_{\mathrm{LoRA},up}[:, j]. \tag{34}$$

In practice, this can be implemented in two equivalent ways. One may explicitly apply the mask to the effective low-rank update $\Delta\mathbf{W}_{\mathrm{LoRA}}$ after composing the LoRA factors. Alternatively, for efficiency, one may apply the mask to the LoRA factor whose dimension aligns with the masked channel. Both implementations yield the same masked effective update.

For CASS-Tuning with LoRA, we mask gradients or optimizer updates associated with inactive structural channels. As with full-rank fine-tuning, inactive channels do not acquire task-specific updates. For optimizers with momentum or decoupled weight decay, we ensure that inactive effective updates remain zero so that the corresponding parameters preserve their base-model behavior.

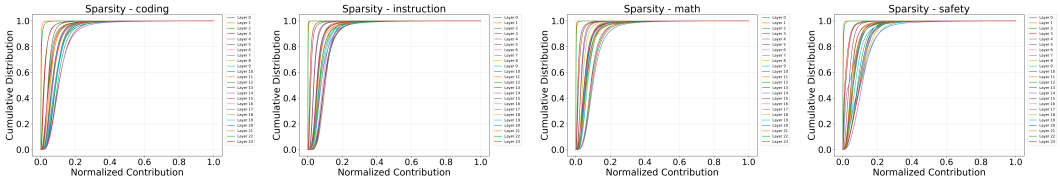
**Summary.** The broadcasting rules above convert component-level masks into parameter-level masks while preserving the structural interpretation of CASS. Attention head masks are applied to the query, key, value, and output-projection slices of each head. FFN neuron masks are applied to the projection slices associated with the corresponding intermediate channel. For gated FFNs and LoRA, the same principle is applied to the effective update along the structurally aligned channel dimension.

# D Extended Analysis on FFN

In Section 3.3, we presented preliminary evidence that task-relevant FFN neurons are sparse and partially separated across tasks. In this section, we provide extended visualizations on Qwen2.5-1.5B-Instruct across four task domains: Coding, Instruction Following, Mathematics, and Safety. The goal is to examine whether the sparsity and cross-task separation observed in the main text persist beyond a single task and a single layer. Similar patterns are also observed on ViT and RoBERTa. To keep the appendix concise, we focus the detailed breakdown on Qwen2.5-1.5B-Instruct results.

## D.1 Universality of FFN Contribution Sparsity

We first examine the distribution of neuron contribution scores across different task domains. For visualization, we follow the same normalized contribution score used in the main text, where larger values indicate stronger additive contribution to the residual stream under task-specific inputs.




(a) Coding (b) Instruction (c) Mathematics (d) Safety

Figure 8: **CDF of FFN neuron contribution scores across tasks.** The CDF curves exhibit a sharp rise near zero across all four task domains, indicating that only a small subset of FFN neurons contributes prominently to each task. This suggests that task-relevant FFN activity is highly concentrated rather than uniformly distributed over all neurons.

Figure 8 shows that the contribution distributions are highly skewed across all four domains. In each case, most neurons have near-zero contribution scores, while a small fraction of neurons forms a high-contribution tail. This observation supports the design choice of CASS: instead of retaining or pruning individual parameters based only on static weight magnitude, we identify task-relevant functional units by their activation-weighted contribution to the residual stream.

## D.2 Layer-wise Separation of Active FFN Neurons

We next analyze whether the active FFN neurons selected for different tasks overlap substantially. For each task and each layer, we select the top-20% neurons according to their contribution scores and compute the pairwise Jaccard similarity between task-specific active sets:

$$\mathrm{Jaccard}(t_1, t_2) = \frac{|\mathcal{N}_{t_1}^{(\ell)} \cap \mathcal{N}_{t_2}^{(\ell)}|}{|\mathcal{N}_{t_1}^{(\ell)} \cup \mathcal{N}_{t_2}^{(\ell)}|}, \tag{35}$$

where $\mathcal{N}_t^{(\ell)}$ denotes the selected active FFN neuron set for task $t$ in layer $\ell$.

Figure 9 shows Jaccard heatmaps from shallow to deep layers. The off-diagonal entries are generally low in the selected layers, indicating limited overlap among the most active FFN neurons of different tasks. We do not interpret this as strict disjointness across the entire model; rather, the visualization suggests that task-relevant FFN activity is often concentrated in partially separated neuron subsets.

Overall, the FFN analysis supports the use of neuron-level structured masks in CASS. Since task-relevant neuron contributions are sparse and the most active neurons of different tasks exhibit limited overlap in representative layers, filtering task-vector updates at the neuron level can suppress task-irrelevant updates while preserving the components most associated with each task.

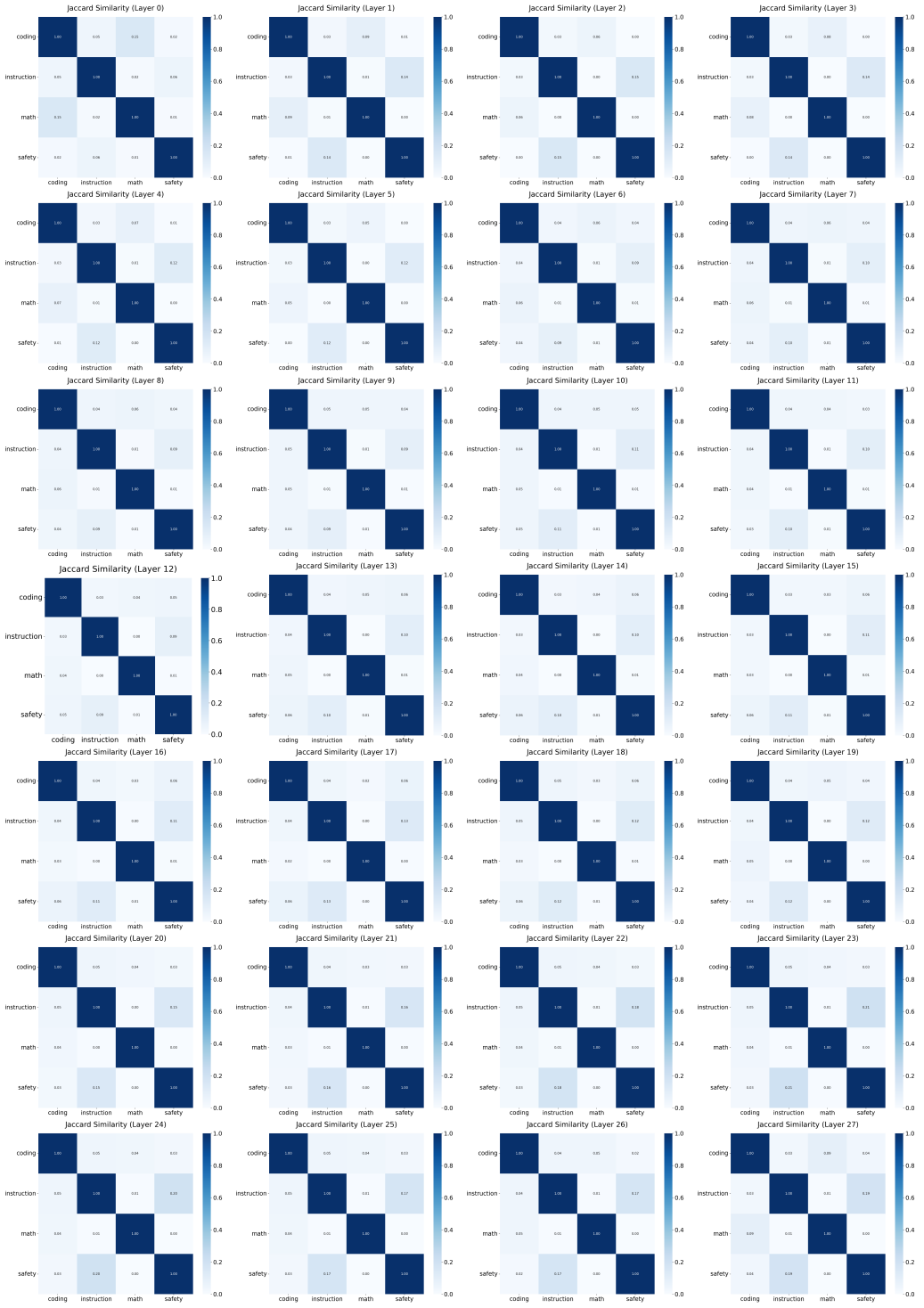



Figure 9: **Layer-wise Jaccard similarity of active FFN neurons across all 28 layers.** Each heatmap shows the pairwise overlap between task-specific active neuron sets in one Transformer layer of Qwen2.5-1.5B-Instruct. The heatmaps are ordered from Layer 0 to Layer 27 in row-major order, from left to right and top to bottom. For each task, we select the top-20% FFN neurons according to their contribution scores and compute the Jaccard similarity between the selected sets. Darker off-diagonal entries indicate lower cross-task overlap. The consistently low off-diagonal values across many layers suggest that task-relevant FFN activity tends to be concentrated in partially separated neuron subsets, supporting the use of neuron-level structured masks in CASS.

# E Extended Analysis on MHA

In addition to FFN neurons, CASS also constructs structured masks over attention heads. Unlike FFN neurons, many attention heads may serve general-purpose functions and remain active across multiple tasks. Therefore, selecting attention heads solely by absolute contribution magnitude may overemphasize generally active heads rather than task-specialized ones. To address this issue, we analyze attention heads using a task-selectivity metric. Similar patterns are also observed on ViT and RoBERTa. To keep the appendix concise, we focus the detailed breakdown on Qwen2.5-1.5B-Instruct results.

## E.1 Task-Selectivity Metric for Attention Heads

For a head $h$, we define its absolute contribution on task $t$ as

$$\mathcal{A}_{h,t} = \mathbb{E}_{\mathbf{X}\sim\mathcal{D}_t}\left[\left\|\mathbf{H}_h(\mathbf{X})\mathbf{W}_O^{(h)}\right\|_F\right], \tag{36}$$

where $\mathbf{H}_h(\mathbf{X})$ denotes the output of head $h$ and $\mathbf{W}_O^{(h)}$ denotes the corresponding output-projection block. The Frobenius norm aggregates the contribution over the sequence dimension.

We then compute the global average contribution of head $h$ across all task domains:

$$\mu_{h,\text{global}} = \frac{1}{K}\sum_{t'=1}^{K}\mathcal{A}_{h,t'}. \tag{37}$$

The task-selectivity score is defined as the relative deviation from this global average:

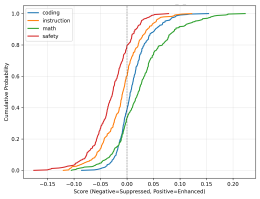
$$\mathcal{S}_{h,t} = \frac{\mathcal{A}_{h,t} - \mu_{h,\text{global}}}{\mu_{h,\text{global}} + \epsilon}, \tag{38}$$

where $\epsilon$ is a small constant for numerical stability.

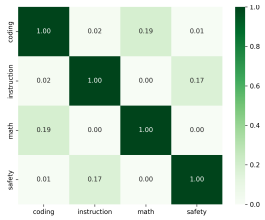
A positive score $\mathcal{S}_{h,t} > 0$ indicates that head $h$ contributes more strongly to task $t$ than its average contribution across tasks. A negative score indicates that the head is less active than its global average under the current task distribution. Therefore, when constructing task-specific head sets for analysis, we rank heads by their positive selectivity scores and select the heads with the largest positive values. We do not use the absolute value of selectivity, since strongly negative scores correspond to suppressed rather than activated task-specific heads.

## E.2 Head-level Sparsity and Cross-task Overlap

Figure 10a shows the CDF of task-selectivity scores. The presence of negative values indicates that many heads are below their global average contribution for a given task. Meanwhile, the right tail shows that only a subset of heads exhibits strong positive selectivity. This pattern motivates using task-selectivity rather than raw contribution magnitude for attention head selection.




(a) **CDF of task-selectivity scores.** Negative scores indicate heads whose contribution is below their global average for a task, while the positive tail corresponds to task-selective heads.




(b) **Global Jaccard overlap of top-20% heads.** For each task, we select the top-20% heads with the largest positive selectivity scores and compute cross-task overlap.

Figure 10: **Attention head specialization across tasks.** Task-selectivity scores reveal that only a subset of heads is strongly specialized for each task, and the selected top heads exhibit limited overlap across different task domains.

Figure 10b reports the Jaccard similarity between the selected top-20% task-selective heads for each task. The overlap is low for many task pairs, suggesting that the most task-selective attention heads are not uniformly shared across domains. We also observe a moderate overlap between Coding and Mathematics, which is consistent with the fact that both tasks involve structured reasoning and may rely on partially shared attention mechanisms. We treat this as evidence of partial sharing rather than strict head-level independence.

### E.3 Depth Distribution of Task-selective Heads

We further examine where task-selective heads are located across the model depth. For each task, we visualize the layer-wise distribution of the top-20% heads selected by positive task-selectivity.

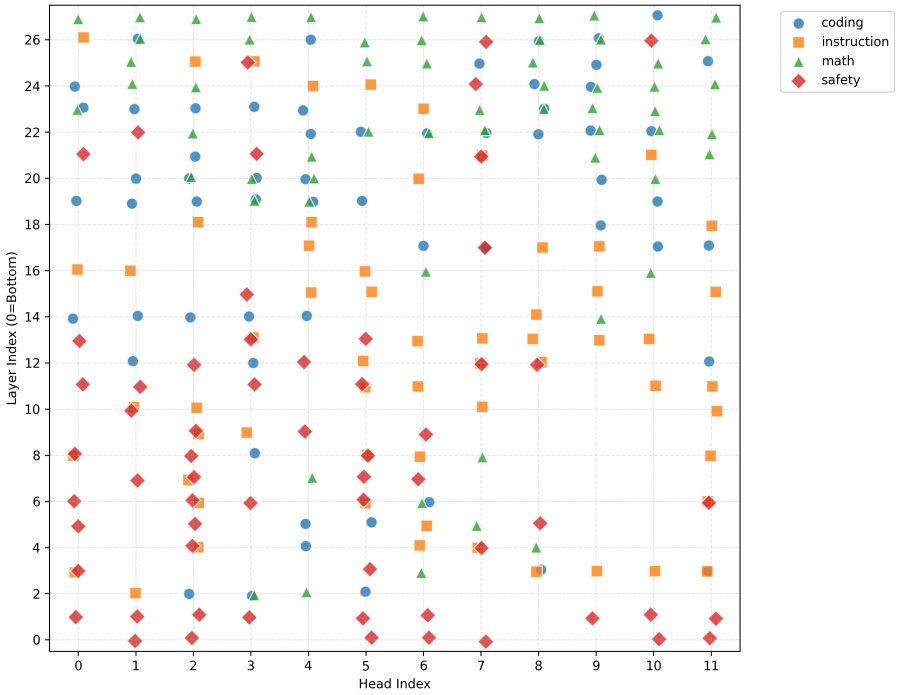



Figure 11: **Layer-wise distribution of top-20% task-selective heads.** Different tasks exhibit different depth preferences. Coding and Mathematics heads are more concentrated in deeper layers, while Safety heads appear more frequently in relatively shallower layers. Instruction-following heads are distributed more broadly across layers.

Figure 11 suggests that task-selective attention heads are not uniformly distributed across depth. Coding and Mathematics show stronger concentration in deeper layers, while Safety-related heads appear more frequently in shallower layers. Instruction-following heads are more broadly distributed. These observations are consistent with the intuition that different tasks may rely on different information-routing patterns, but we avoid interpreting them as strict mechanistic localization.

Overall, the MHA analysis supports the use of task-selectivity-based head masks in CASS. Since many attention heads may be generally active across tasks, raw contribution magnitude alone can overselect generic heads. Positive task-selectivity provides a more targeted criterion for identifying heads whose contributions are relatively amplified under a specific task distribution, enabling CASS to preserve task-relevant head updates while suppressing less relevant ones.

# F Experimental Settings

This section describes the experimental settings used in our vision, encoder-only language, and decoder-only language experiments. We summarize the task collections, evaluation protocols, fine-tuning setup, normalization metrics, and random seed configuration.

## F.1 Vision Benchmarks

For vision experiments, we evaluate CASS on CLIP-based ViT-B/32 and ViT-B/16 models. The benchmark contains 20 image classification tasks:

> DTD [6], EuroSAT [17], GTSRB [44], MNIST [26], RESISC45 [4], Stanford-Cars [23], SUN397 [57], SVHN [36], Oxford-Flowers102 [37], PCAM [49], FER2013 [14], Oxford-IIIT-Pet [39], STL10 [8], CIFAR100 [24], CIFAR10 [24], Food101 [3], Fashion-MNIST [56], EMNIST-Letters [9], KMNIST [7], and Rendered-SST2 [42].

For the main vision experiments, we merge task-specific experts from all 20 tasks and report the average classification accuracy across these tasks following previous study [12].

For the vision OOD experiment, we split the 20 tasks into 12 in-distribution tasks and 8 held-out OOD tasks. The 12 in-distribution tasks are:

> DTD, EuroSAT, GTSRB, MNIST, RESISC45, Stanford-Cars, SUN397, SVHN, Oxford-Flowers102, PCAM, FER2013, and Oxford-IIIT-Pet.

The 8 OOD tasks are:

> STL10, CIFAR100, CIFAR10, Food101, Fashion-MNIST, EMNIST-Letters, KMNIST, and Rendered-SST2.

In this setting, models are merged using only the 12 in-distribution task experts and are then evaluated on both the 12 in-distribution tasks and the 8 held-out tasks.

## F.2 RoBERTa Benchmarks

For encoder-only language experiments, we evaluate RoBERTa on 8 tasks from the GLUE benchmark [50]:

> CoLA, MNLI, MRPC, QNLI, QQP, RTE, SST-2, and STS-B.

We report the average task performance over the 8 tasks. Each task is evaluated using its standard benchmark metric [34]. Since these tasks are evaluated under the same benchmark protocol, we report the direct average of task-level scores without additional normalization.

## F.3 Qwen2.5 Benchmarks

For decoder-only language experiments, we evaluate Qwen2.5 models on four capabilities: Mathematics, Instruction Following, Coding, and Safety. The datasets and metrics are summarized in Table 6.

Table 6: **Qwen2.5 tasks, datasets, and metrics.** We evaluate four decoder-only capabilities covering mathematical reasoning, instruction following, code generation, and safety.

| Capability | Fine-tuning Dataset | Evaluation Dataset | Metric |
|---|---|---|---|
| Math | DART-Math [47] | minerva_math500 [19] | math_verify accuracy |
| Instruction | TULU-3 persona IF [25] | ifeval [64] | prompt-level strict accuracy |
| Coding | Magicoder [52] | mbppplus [2] | pass@1 |
| Safety | WildGuardMix [15] | wildguardtest [15] | Micro Harm Rate (↓) |

Since the four Qwen2.5 capabilities use different evaluation metrics and scales, we report average normalized performance for Qwen2.5 experiments following previous study [16]. The normalization protocol is described in Appendix F.5.

### F.4 Fine-tuning Hyperparameters

For Qwen2.5 task-specific fine-tuning, we use LoRA-based parameter-efficient fine-tuning. The hyperparameters are summarized in Table 7. The same configuration is used across the four decoder-only capabilities unless otherwise specified.

Table 7: **Fine-tuning hyperparameters for Qwen2.5.**

| Hyperparameter | Value |
|---|---|
| Fine-tuning Method | LoRA |
| LoRA Rank ($r$) | 32 |
| LoRA Alpha | 64 |
| LoRA Dropout | 0.1 |
| Learning Rate | $2 \times 10^{-4}$ |
| LR Scheduler | Cosine |
| Warmup Ratio | 0.05 |
| Epochs | 1 |
| Batch Size | 16 |

### F.5 Evaluation and Normalization Metrics

For vision experiments, we report the average classification accuracy across the evaluated tasks [12]. For RoBERTa experiments, we report the average task performance over the 8 GLUE tasks using their standard evaluation metrics [34].

For Qwen2.5 experiments, the four capabilities use heterogeneous metrics. Mathematics, Instruction Following, and Coding are higher-is-better metrics, while Safety is measured by Micro Harm Rate, where lower is better. To aggregate these capabilities into a single score, we normalize each task score relative to the corresponding single-task fine-tuned expert [16].

Let $S_t(\theta)$ denote the raw evaluation score of model $\theta$ on task $t$, and let $\theta_{ft}^{(t)}$ denote the dedicated fine-tuned expert for task $t$. For higher-is-better metrics, the normalized score is defined as

$$P_t(\theta) = \frac{S_t(\theta)}{S_t(\theta_{ft}^{(t)})}. \tag{39}$$

For the Safety task, we denote the Micro Harm Rate of model $\theta$ by $H_{\text{safety}}(\theta)$. Since lower harm rate indicates better safety performance, we convert it into a higher-is-better normalized score:

$$P_{\text{safety}}(\theta) = \frac{1 - H_{\text{safety}}(\theta)}{1 - H_{\text{safety}}(\theta_{ft}^{(\text{safety})})}. \tag{40}$$

The final normalized performance for Qwen2.5 is computed as the arithmetic mean over the four capabilities:

$$P_{\text{avg}}(\theta) = \frac{1}{4} \sum_{t=1}^{4} P_t(\theta). \tag{41}$$

Under this definition, the dedicated fine-tuned experts have normalized performance equal to 1.0 for their corresponding tasks. This allows models to be compared across heterogeneous capabilities using a unified aggregate score.

### F.6 Random Seeds

All experiments use a fixed random seed of 42, including mask construction, fine-tuning, and evaluation procedures whenever randomness is involved.

### F.7 Compute Resources.

All experiments were conducted on NVIDIA GPUs and Ascend NPUs. The computational cost varies substantially across merging baselines. Simple arithmetic or sparsification-based methods such as TA and DARE require only a few seconds per merging run after the single-task checkpoints are available, whereas optimization- or subspace-based methods such as Iso-C may require approximately tens of minutes per run depending on the number of tasks and model size. The additional cost introduced by CASS is small: estimating and applying the CASS mask typically takes only a few seconds per run, and the main overhead comes from collecting activation statistics on a small calibration set.

# G Detailed Qwen2.5 Results

This section provides detailed per-capability results on Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct. While the main text reports average normalized performance, here we further decompose the results into Coding, Safety, Mathematics, and Instruction Following. These results provide a more fine-grained view of how CASS affects different capabilities under both post-hoc merging and mask-guided fine-tuning.

## G.1 Detailed Results of CASS-Merging

Table 8: **Detailed CASS-Merging results on Qwen2.5.** Per-capability normalized performance on Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct. CASS-M is applied as a post-hoc structured task-vector filter before each merge operator.

| Method | Qwen2.5-0.5B-Instruct | | | | | Qwen2.5-1.5B-Instruct | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Code** | **Safe** | **Math** | **Instr.** | **Avg.** | **Code** | **Safe** | **Math** | **Instr.** | **Avg.** |
| Base Model | 0.676 | 0.708 | 0.765 | 0.731 | 0.720 | 0.687 | 0.775 | 0.855 | 0.754 | 0.768 |
| Fine-Tuned | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Post-Mask | 0.964 | 0.836 | 0.961 | 0.878 | 0.910 | 0.953 | 1.015 | 1.051 | 0.882 | 0.975 |
| TA | 0.801 | 0.687 | 0.765 | 0.718 | 0.743 | 0.735 | 0.778 | 0.942 | 0.703 | 0.789 |
| w/ CASS-M | 0.820 | 0.684 | 0.814 | 0.788 | **0.776** | 0.808 | 0.777 | 1.065 | 0.698 | **0.837** |
| TIES | 0.671 | 0.659 | 0.873 | 0.577 | 0.695 | 0.829 | 0.752 | 0.978 | 0.651 | 0.803 |
| w/ CASS-M | 0.753 | 0.731 | 0.863 | 0.743 | **0.773** | 0.846 | 0.767 | 1.094 | 0.610 | **0.829** |
| DARE | 0.857 | 0.697 | 0.892 | 0.743 | 0.797 | 0.933 | 0.722 | 1.072 | 0.641 | 0.842 |
| w/ CASS-M | 0.903 | 0.684 | 0.853 | 0.788 | **0.807** | 0.924 | 0.782 | 1.087 | 0.692 | **0.871** |
| TSV | 0.757 | 0.717 | 0.853 | 0.776 | 0.776 | 0.806 | 0.763 | 1.058 | 0.698 | 0.831 |
| w/ CASS-M | 0.813 | 0.712 | 0.775 | 0.808 | **0.777** | 0.838 | 0.756 | 1.065 | 0.728 | **0.847** |
| Iso-C | 0.704 | 0.701 | 0.725 | 0.820 | **0.738** | 0.743 | 0.782 | 0.935 | 0.774 | 0.809 |
| w/ CASS-M | 0.702 | 0.698 | 0.755 | 0.763 | 0.729 | 0.698 | 0.787 | 1.014 | 0.754 | **0.814** |
| WUDI | 0.589 | 0.674 | 0.843 | 0.763 | 0.717 | 0.971 | 0.713 | 1.058 | 0.708 | 0.862 |
| w/ CASS-M | 0.832 | 0.692 | 0.843 | 0.737 | **0.776** | 0.927 | 0.737 | 1.094 | 0.733 | **0.873** |

Table 8 reports the detailed results of CASS-Merging. The "Post-Mask" row serves as a diagnostic reference by evaluating task-specific experts after applying CASS masks. Its strong average performance on both Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct indicates that the selected components preserve a large fraction of task-specific capability, despite suppressing updates outside the identified active subspace. This supports the premise that task-relevant updates are concentrated in a sparse subset of activation-prominent components.

Notably, on Qwen2.5-1.5B-Instruct, the Post-Mask scores on Safety and Mathematics slightly exceed 1.0. This does not imply that Post-Mask is evaluated against a different reference; rather, the normalized Fine-Tuned score of 1.0 is an empirical single-task baseline rather than a theoretical upper bound. Removing updates outside the selected active subspace can act as a mild denoising or regularization effect, so the masked expert may occasionally outperform the original fine-tuned expert on individual metrics. This observation further suggests that some fine-tuning updates outside the active components can be redundant or mildly detrimental, although Post-Mask is used only as a diagnostic reference rather than as a competing merging method.

Across merge operators, CASS-Merging improves most average scores on both model scales. On Qwen2.5-0.5B-Instruct, the largest gains appear for TIES and WUDI, improving the average normalized performance from 0.695 to 0.773 and from 0.717 to 0.776, respectively. Task Arithmetic also benefits substantially, improving from 0.743 to 0.776. These improvements suggest that structured component-level filtering is particularly effective when the original merge operator does not explicitly distinguish task-relevant functional units from task-irrelevant updates.

On Qwen2.5-1.5B-Instruct, CASS-Merging improves the average performance of all evaluated merge operators. The gains are especially visible for Task Arithmetic, DARE, TIES, and TSV. For example, Task Arithmetic improves from 0.789 to 0.837, while DARE improves from 0.842 to 0.871. The improvements on the larger model indicate that the structured masks remain effective when the model has more capacity and a richer set of functional components.

The per-capability results also show that CASS-Merging does not uniformly improve every individual capability. Some merge operators exhibit trade-offs, where improvements in Mathematics or Coding are accompanied by small decreases in Safety or Instruction Following. This is expected because CASS is applied before a merge operator and does not directly optimize the final multi-task objective. Instead, it filters each task vector according to task-specific component relevance. The consistent average gains indicate that this filtering generally reduces harmful interference, even though individual capabilities may still depend on the downstream merge operator.

The only average-performance degradation occurs for Iso-C on Qwen2.5-0.5B-Instruct, where CASS-Merging changes the average score from 0.738 to 0.729. This suggests that the interaction between structured task-vector filtering and spectrum-based subspace regularization may be sensitive to model scale or hyperparameter choices. In contrast, Iso-C combined with CASS-Merging improves on Qwen2.5-1.5B-Instruct, indicating that this interaction is not uniformly negative but may depend on the capacity and geometry of the underlying model.

Overall, the detailed CASS-Merging results support the conclusion that CASS is broadly complementary to existing merge operators. The benefits are strongest for methods that rely on simple arithmetic, sign conflict resolution, random dropping, or optimization in the original task-vector space, while gains are more moderate for methods that already impose strong subspace or spectrum constraints.

### G.2 Detailed Results of CASS-Tuning

Table 9: **Detailed CASS-Tuning results on Qwen2.5.** Per-capability normalized performance on Qwen2.5-0.5B-Instruct and Qwen2.5-1.5B-Instruct. CASS-T constrains task-specific fine-tuning with structured masks before merging.

| Method | Qwen2.5-0.5B-Instruct | | | | | Qwen2.5-1.5B-Instruct | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Code** | **Safe** | **Math** | **Instr.** | **Avg.** | **Code** | **Safe** | **Math** | **Instr.** | **Avg.** |
| Base Model | 0.676 | 0.708 | 0.765 | 0.731 | 0.720 | 0.687 | 0.775 | 0.855 | 0.754 | 0.768 |
| Fine-Tuned | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| Masked FT | 0.966 | 0.988 | 0.902 | 0.962 | 0.954 | 0.885 | 1.008 | 0.978 | 0.826 | 0.924 |
| TA | 0.801 | 0.687 | 0.765 | 0.718 | 0.743 | 0.735 | 0.778 | 0.942 | 0.703 | 0.789 |
| w/ CASS-T | 0.889 | 0.705 | 0.863 | 0.788 | **0.811** | 0.798 | 0.772 | 1.007 | 0.703 | **0.820** |
| TIES | 0.671 | 0.659 | 0.873 | 0.577 | 0.695 | 0.829 | 0.752 | 0.978 | 0.651 | 0.803 |
| w/ CASS-T | 0.788 | 0.715 | 1.000 | 0.763 | **0.816** | 0.969 | 0.748 | 1.073 | 0.677 | **0.867** |
| DARE | 0.857 | 0.697 | 0.892 | 0.743 | 0.797 | 0.933 | 0.722 | 1.073 | 0.641 | 0.842 |
| w/ CASS-T | 0.853 | 0.690 | 0.922 | 0.776 | **0.810** | 0.933 | 0.743 | 1.022 | 0.703 | **0.850** |
| TSV | 0.757 | 0.717 | 0.853 | 0.776 | 0.776 | 0.806 | 0.763 | 1.058 | 0.698 | 0.831 |
| w/ CASS-T | 0.813 | 0.710 | 0.902 | 0.801 | **0.806** | 0.800 | 0.783 | 1.130 | 0.723 | **0.859** |
| Iso-C | 0.704 | 0.701 | 0.725 | 0.820 | 0.738 | 0.743 | 0.782 | 0.935 | 0.774 | 0.809 |
| w/ CASS-T | 0.784 | 0.708 | 0.775 | 0.763 | **0.757** | 0.743 | 0.787 | 1.022 | 0.749 | **0.825** |
| WUDI | 0.589 | 0.674 | 0.843 | 0.763 | 0.717 | 0.971 | 0.713 | 1.058 | 0.708 | 0.862 |
| w/ CASS-T | 0.901 | 0.695 | 0.794 | 0.769 | **0.790** | 1.003 | 0.764 | 1.036 | 0.728 | **0.883** |

Table 9 reports the detailed results of CASS-Tuning. Unlike CASS-Merging, which filters task vectors after conventional fine-tuning, CASS-Tuning applies structured masks during the fine-tuning process itself. The "Masked FT" row reports the single-task performance of experts fine-tuned under the CASS masks. The high average scores of 0.954 on Qwen2.5-0.5B-Instruct and 0.924 on Qwen2.5-1.5B-Instruct show that restricting updates to the selected active subspace still preserves

most task-specific capability. This confirms that the masked subspace is sufficiently expressive for task adaptation.

CASS-Tuning improves all evaluated merge operators on both Qwen2.5 model scales. On Qwen2.5-0.5B-Instruct, the largest improvement is observed for TIES, which increases from 0.695 to 0.816. WUDI and Task Arithmetic also show strong gains, improving from 0.717 to 0.790 and from 0.743 to 0.811, respectively. These results indicate that constraining fine-tuning to task-relevant components can produce cleaner task vectors before any merge operation is applied.

On Qwen2.5-1.5B-Instruct, CASS-Tuning also consistently improves average performance. TIES improves from 0.803 to 0.867, TSV improves from 0.831 to 0.859, and WUDI improves from 0.862 to 0.883. Compared with the 0.5B model, the gains on the 1.5B model are generally smaller in magnitude but more stable across merge operators. This suggests that larger models may already provide more separable task representations, while CASS-Tuning further regularizes the update subspaces to reduce residual interference.

The per-capability results reveal that the improvements from CASS-Tuning are particularly pronounced for Coding and Mathematics in several merge operators. This pattern is consistent with the intuition that structured reasoning capabilities are sensitive to destructive task-vector interference and can benefit from restricting updates to task-relevant components. Safety and Instruction Following show more mixed per-task changes, but the average results remain consistently positive, indicating that CASS-Tuning improves the overall balance among the four capabilities.

Compared with CASS-Merging, CASS-Tuning provides a stronger form of structural control because it constrains the optimization trajectory rather than filtering task vectors only after fine-tuning. As a result, the generated task vectors are already concentrated in the selected component subspaces before merging. This explains why CASS-Tuning yields consistent average improvements across all evaluated merge operators, including methods such as Iso-C and WUDI, whose interaction with post-hoc filtering can be more method-dependent.

Taken together, the detailed results show that CASS-Merging and CASS-Tuning provide different ways to reduce interference in decoder-only model merging. CASS-Merging acts as a plug-and-play post-hoc filter for existing task vectors, while CASS-Tuning proactively shapes the task vectors during fine-tuning. Both variants improve average normalized performance on Qwen2.5, and CASS-Tuning achieves the most consistent improvements across merge operators.

## H Limitations

We acknowledge two limitations of the current study.

**First**, CASS uses a fixed sparsity ratio for selecting task-relevant components. Although the adopted ratio (20% in our experiments) is motivated by the observed concentration of contribution scores, a uniform threshold may not be optimal for all tasks, layers, or model architectures. Tasks with different levels of complexity may require different numbers of active heads and FFN neurons. Developing adaptive selection strategies, such as task-dependent or layer-dependent sparsity ratios, is an important direction for future work.

**Second**, our decoder-only experiments are conducted on Qwen2.5 models up to 1.5B parameters due to computational resource constraints. Although the consistent improvements across ViT, RoBERTa, and Qwen2.5 suggest that contribution-aware structured sparsity is broadly applicable, validating CASS on larger foundation models, such as 7B-scale or larger decoder-only models, remains future work.

## I Broader Impacts

This work aims to improve the efficiency and effectiveness of model merging and task-aware adaptation. Its potential positive impacts include reducing the computational cost of constructing multi-task models, improving the accessibility of model adaptation, and enabling more efficient reuse of existing task-specific models. These benefits may reduce redundant training costs and lower the barrier for researchers and practitioners with limited compute resources. In future work, we plan to further validate the effectiveness and generalizability of our method across a broader range of tasks and application scenarios [59, 28].

Potential negative impacts may arise from the same efficiency gains. More efficient model merging and adaptation could make it easier to combine or specialize models for harmful applications, or to unintentionally preserve and transfer undesirable biases or unsafe behaviors from task-specific models into a merged model. In addition, if the merged model is deployed without sufficient validation, performance degradation on under-evaluated tasks or safety-sensitive inputs could lead to unreliable behavior. We therefore recommend careful evaluation of merged models, including task-specific robustness and safety checks, before deployment in real-world or high-risk settings.