Appendix

Theoretical Proofs and Mathematical Details

Proof of Task Vector Decomposition

For any task vector $\tau_A \in \mathbb{R}^{m\times n}$, we can decompose it into orthogonal and non-orthogonal components relative to another task vector τ_B. We perform this decomposition through the following steps:

Step 1: Row Space Decomposition. Let V_A and V_B be the right singular vectors from the SVD decompositions $\tau_A = U_A\Sigma_A V_A^T$ and $\tau_B = U_B\Sigma_B V_B^T$. We decompose V_A into components parallel and perpendicular to V_B:

$$V_A = V_A^{\parallel} + V_A^{\perp}, \tag{1}$$

where $V_A^{\parallel} = V_B(V_B^T V_A)$ is the projection onto the row space of τ_B, and $V_A^{\perp} = V_A - V_A^{\parallel}$ is orthogonal to V_B.

Step 2: Column Space Decomposition. Similarly, we decompose U_A relative to U_B:

$$U_A = U_A^{\parallel} + U_A^{\perp}, \tag{2}$$

where $U_A^{\parallel} = U_B(U_B^T U_A)$ and $U_A^{\perp} = U_A - U_A^{\parallel}$.

Step 3: Task Vector Reconstruction. The task vector can be reconstructed as:

$$\tau_A = U_A\Sigma_A V_A^T = (U_A^{\parallel} + U_A^{\perp})\Sigma_A(V_A^{\parallel} + V_A^{\perp})^T. \tag{3}$$

Expanding this expression:

$$\begin{aligned}\tau_A &= U_A^{\parallel}\Sigma_A(V_A^{\parallel})^T + U_A^{\parallel}\Sigma_A(V_A^{\perp})^T \\ &\quad + U_A^{\perp}\Sigma_A(V_A^{\parallel})^T + U_A^{\perp}\Sigma_A(V_A^{\perp})^T.\end{aligned} \tag{4}$$

The completely orthogonal component is:

$$\tau_A^{\perp} = U_A^{\perp}\Sigma_A(V_A^{\perp})^T, \tag{5}$$

which satisfies both $\text{Row}(\tau_A^{\perp}) \perp \text{Row}(\tau_B)$ and $\text{Col}(\tau_A^{\perp}) \perp \text{Col}(\tau_B)$.

Complete Proof of Orthogonal Components Being Conflict-Free

We provide a comprehensive proof that orthogonal components of task vectors enable conflict-free merging. This extends the proof in the main paper with full mathematical rigor and detailed derivations.

Theorem. *Let $\tau_A, \tau_B \in \mathbb{R}^{m\times n}$ be task vectors in the global coordinate system. If $\tau_A^{\perp}$ is the component of τ_A that satisfies both row space orthogonality (Row$(\tau_A^{\perp}) \perp$ Row(τ_B)) and column space orthogonality (Col$(\tau_A^{\perp}) \perp$ Col(τ_B)), then merging $\tau_{merged} = \tau_A^{\perp} + \tau_B$ preserves each task's functionality without interference.*

Complete Proof:

Step 1: Establishing the Mathematical Foundation. First, we establish the SVD decomposition framework for our analysis. Let $\tau_A^{\perp} = U_A^{\perp}\Sigma_A^{\perp}(V_A^{\perp})^T$ and $\tau_B = U_B\Sigma_B V_B^T$ be the SVD decompositions of the orthogonal component and the second task vector respectively.

The orthogonality conditions can be expressed mathematically as:

$$\text{Row orthogonality: } (V_A^{\perp})^T V_B = 0, \tag{6}$$

$$\text{Column orthogonality: } (U_A^{\perp})^T U_B = 0. \tag{7}$$

This means that the row space spanned by $\{v_1^{A^\perp}, v_2^{A^\perp}, \ldots, v_{r_A}^{A^\perp}\}$ and the row space spanned by $\{v_1^B, v_2^B, \ldots, v_{r_B}^B\}$ are completely orthogonal, where r_A and r_B are the ranks of the respective task vectors. Similarly for the column spaces.

Step 2: Input Space Analysis and Decomposition. For any input vector $\mathbf{x} \in \mathbb{R}^n$, we need to understand how it interacts with both task vectors. The key insight is that we can decompose the entire input space $\mathbb{R}^n$ into orthogonal subspaces based on the row spaces of the task vectors.

Let $\mathcal{R}_A^{\perp} = \text{Row}(\tau_A^{\perp})$, $\mathcal{R}_B = \text{Row}(\tau_B)$, and $\mathcal{R}_{\text{null}}$ be the null space orthogonal to both row spaces. Then we have the orthogonal decomposition:

$$\mathbb{R}^n = \mathcal{R}_A^{\perp} \oplus \mathcal{R}_B \oplus \mathcal{R}_{\text{null}}. \tag{8}$$

Any input vector $\mathbf{x} \in \mathbb{R}^n$ can be uniquely decomposed as:

$$\mathbf{x} = \mathbf{x}_A^{\perp} + \mathbf{x}_B + \mathbf{x}_{\text{null}}, \tag{9}$$

where:

$$\begin{aligned}\mathbf{x}_A^{\perp} &= P_{\mathcal{R}_A^{\perp}}(\mathbf{x}) = V_A^{\perp}((V_A^{\perp})^T\mathbf{x}), \\ \mathbf{x}_B &= P_{\mathcal{R}_B}(\mathbf{x}) = V_B(V_B^T\mathbf{x}), \\ \mathbf{x}_{\text{null}} &= \mathbf{x} - \mathbf{x}_A^{\perp} - \mathbf{x}_B.\end{aligned} \tag{10}$$

Here $P_{\mathcal{S}}$ denotes the orthogonal projection operator onto subspace $\mathcal{S}$.

Step 3: Analyzing Task Vector Actions on Input Components. Now we analyze how each task vector acts on the decomposed input vector.

Action of $\tau_A^{\perp}$ on $\mathbf{x}$:

Since $\tau_A^{\perp} = U_A^{\perp}\Sigma_A^{\perp}(V_A^{\perp})^T$, we have:

$$\begin{aligned}\tau_A^{\perp}(\mathbf{x}) &= \tau_A^{\perp}(\mathbf{x}_A^{\perp} + \mathbf{x}_B + \mathbf{x}_{\text{null}}) \\ &= \tau_A^{\perp}(\mathbf{x}_A^{\perp}) + \tau_A^{\perp}(\mathbf{x}_B) + \tau_A^{\perp}(\mathbf{x}_{\text{null}}).\end{aligned} \tag{11}$$

For the second term: Since $\mathbf{x}_B \in \mathcal{R}_B = \text{Row}(\tau_B)$ and $\text{Row}(\tau_A^{\perp}) \perp \text{Row}(\tau_B)$, we have $\mathbf{x}_B \perp \text{Row}(\tau_A^{\perp})$. This means $\mathbf{x}_B \in \text{Null}(\tau_A^{\perp})$, therefore:

$$\tau_A^{\perp}(\mathbf{x}_B) = U_A^{\perp}\Sigma_A^{\perp}(V_A^{\perp})^T\mathbf{x}_B = U_A^{\perp}\Sigma_A^{\perp} \cdot 0 = 0. \tag{12}$$

For the third term: Since $\mathbf{x}_{\text{null}} \perp \text{Row}(\tau_A^{\perp})$, we similarly have $\mathbf{x}_{\text{null}} \in \text{Null}(\tau_A^{\perp})$, therefore:

$$\tau_A^{\perp}(\mathbf{x}_{\text{null}}) = 0. \tag{13}$$

Thus:

$$\begin{aligned}\tau_A^{\perp}(\mathbf{x}) &= \tau_A^{\perp}(\mathbf{x}_A^{\perp}) \\ &= U_A^{\perp}\Sigma_A^{\perp}(V_A^{\perp})^T\mathbf{x}_A^{\perp} = \mathbf{y}_A^{\perp}.\end{aligned} \tag{14}$$

Action of τ_B on $\mathbf{x}$:

By analogous reasoning:

$$\tau_B(\mathbf{x}) = \tau_B(\mathbf{x}_A^\perp) + \tau_B(\mathbf{x}_B) + \tau_B(\mathbf{x}_{\text{null}}). \tag{15}$$

Since $\mathbf{x}_A^\perp \in \text{Row}(\tau_A^\perp)$ and $\text{Row}(\tau_A^\perp) \perp \text{Row}(\tau_B)$, we have $\mathbf{x}_A^\perp \perp \text{Row}(\tau_B)$, which means $\mathbf{x}_A^\perp \in \text{Null}(\tau_B)$. Therefore $\tau_B(\mathbf{x}_A^\perp) = 0$.

Similarly, $\mathbf{x}_{\text{null}} \in \text{Null}(\tau_B)$, so $\tau_B(\mathbf{x}_{\text{null}}) = 0$.

Thus:

$$\tau_B(\mathbf{x}) = \tau_B(\mathbf{x}_B) = U_B\Sigma_B V_B^T \mathbf{x}_B = \mathbf{y}_B. \tag{16}$$

Step 4: Output Space Analysis. The merged transformation produces:

$$\tau_{\text{merged}}(\mathbf{x}) = \tau_A^\perp(\mathbf{x}) + \tau_B(\mathbf{x}) = \mathbf{y}_A^\perp + \mathbf{y}_B. \tag{17}$$

Now we analyze the relationship between output vectors $\mathbf{y}_A^\perp$ and $\mathbf{y}_B$.

Since $\mathbf{y}_A^\perp = U_A^\perp \Sigma_A^\perp (V_A^\perp)^T \mathbf{x}_A^\perp$, we have $\mathbf{y}_A^\perp \in \text{Col}(\tau_A^\perp) = \text{span}(U_A^\perp)$.

Since $\mathbf{y}_B = U_B\Sigma_B V_B^T \mathbf{x}_B$, we have $\mathbf{y}_B \in \text{Col}(\tau_B) = \text{span}(U_B)$.

By the column space orthogonality condition $(U_A^\perp)^T U_B = 0$, we have:

$$\text{Col}(\tau_A^\perp) \perp \text{Col}(\tau_B). \tag{18}$$

This implies:

$$\langle \mathbf{y}_A^\perp, \mathbf{y}_B \rangle = 0. \tag{19}$$

Step 5: Establishing Conflict-Free Properties. The orthogonality established above ensures the following conflict-free properties:

Property 1: Input Independence Each task vector processes completely disjoint input subspaces: $\tau_A^\perp$ only responds to inputs in $\mathcal{R}_A^\perp$; τ_B only responds to inputs in $\mathcal{R}_B$; and $\mathcal{R}_A^\perp \perp \mathcal{R}_B$.

Property 2: Output Independence Task outputs lie in orthogonal subspaces: $\mathbf{y}_A^\perp \in \text{Col}(\tau_A^\perp)$; $\mathbf{y}_B \in \text{Col}(\tau_B)$; and $\text{Col}(\tau_A^\perp) \perp \text{Col}(\tau_B)$.

Property 3: Additive Preservation The merged output preserves both individual contributions without interference:

$$\|\mathbf{y}_A^\perp + \mathbf{y}_B\|^2 = \|\mathbf{y}_A^\perp\|^2 + \|\mathbf{y}_B\|^2. \tag{20}$$

This follows from the orthogonality $\langle \mathbf{y}_A^\perp, \mathbf{y}_B \rangle = 0$.

Property 4: Functionality Preservation Each task's input-output mapping remains intact within its designated subspace: for inputs $\mathbf{x} \in \mathcal{R}_A^\perp$, $\tau_{\text{merged}}(\mathbf{x}) = \tau_A^\perp(\mathbf{x})$; and for inputs $\mathbf{x} \in \mathcal{R}_B$, $\tau_{\text{merged}}(\mathbf{x}) = \tau_B(\mathbf{x})$.

Step 6: Mathematical Conclusion. We have proven that when task vectors satisfy both row space and column space orthogonality conditions, they can be merged without any parameter conflicts. The orthogonal components $\tau_A^\perp$ and τ_B operate in completely independent subspaces of the input-output transformation, ensuring that: 1. Each task processes disjoint input directions. 2. Each task generates outputs in orthogonal subspaces. 3. No destructive interference occurs between task outputs. 4. Each task's functionality is completely preserved.

Therefore, $\tau_A^\perp$ and τ_B can be merged as $\tau_{\text{merged}} = \tau_A^\perp + \tau_B$ without conflicts, validating our theoretical framework for conflict-free model merging.

Impact of Non-linear Transformations

While our theoretical analysis focuses on linear transformations, neural networks contain non-linear components such as activation functions and normalization layers. We address their impact below.

1. Activation Functions: Element-wise activation functions such as ReLU, GELU, and Swish fundamentally alter the orthogonal relationships established by linear transformations. If $\tau_A^\perp$ and τ_B produce orthogonal outputs $\mathbf{y}_A^\perp$ and $\mathbf{y}_B$ with $\langle \mathbf{y}_A^\perp, \mathbf{y}_B \rangle = 0$, the activated outputs generally lose this orthogonality:

$$\langle \sigma(\mathbf{y}_A^\perp + \mathbf{y}_B), \sigma(\mathbf{y}_A^\perp) + \sigma(\mathbf{y}_B) \rangle \neq 0, \tag{21}$$

where σ is the activation function. This occurs because non-linear functions introduce cross-terms that couple previously orthogonal components.

However, the orthogonal structure provides several advantages even in the presence of non-linearity:

Localized Impact: For activation functions like ReLU that are piecewise linear, the non-linear effects are localized to regions where the input crosses activation boundaries. In regions where $\mathbf{y}_A^\perp + \mathbf{y}_B > 0$ (for ReLU), the transformation remains locally linear, partially preserving orthogonal relationships.

Reduced Interference: Although perfect orthogonality is lost, the starting orthogonal configuration minimizes the magnitude of cross-interference between task-specific activations compared to arbitrary non-orthogonal combinations.

Statistical Independence: When task vectors operate in different regions of the activation function (e.g., different sign patterns), the coupling effects are reduced, maintaining approximate statistical independence between task contributions.

2. Layer Normalization: Layer normalization introduces more significant perturbations to orthogonal relationships:

$$\text{LayerNorm}(\mathbf{y}) = \gamma \odot \frac{\mathbf{y} - \mu(\mathbf{y})}{\sigma(\mathbf{y})} + \beta, \tag{22}$$

where $\mu(\mathbf{y})$ and $\sigma(\mathbf{y})$ are the mean and standard deviation computed across dimensions.

For orthogonal inputs $\mathbf{y} = \mathbf{y}_A^\perp + \mathbf{y}_B$, the normalization statistics become:

$$\mu(\mathbf{y}_A^\perp + \mathbf{y}_B) = \mu(\mathbf{y}_A^\perp) + \mu(\mathbf{y}_B), \tag{23}$$

$$\sigma^2(\mathbf{y}_A^\perp + \mathbf{y}_B) = \sigma^2(\mathbf{y}_A^\perp) + \sigma^2(\mathbf{y}_B), \tag{24}$$

where the cross-terms vanish due to orthogonality: $\langle \mathbf{y}_A^\perp - \mu(\mathbf{y}_A^\perp), \mathbf{y}_B - \mu(\mathbf{y}_B) \rangle = 0$.

This preservation of statistical independence under normalization provides a key advantage: the normalization operation treats the combined orthogonal contributions in a predictable manner, avoiding the complex coupling effects that arise with non-orthogonal combinations.

3. Empirical Validation: Our experimental results demonstrate that despite theoretical limitations imposed by non-linear transformations:

- Orthogonal merging consistently outperforms naive parameter averaging.

- Performance degradation due to non-linear effects remains limited (typically $<1\%$ across benchmarks).
- The relative advantages of orthogonal configurations persist throughout the network depth.

The empirical success of our method validates that while perfect linear orthogonality cannot be maintained through non-linear transformations, the geometric insights from linear analysis provide valuable guidance for effective model merging in practical neural network architectures.

Correspondence Between Five Parameter Types and Four Overlap Patterns

We establish the precise correspondence between our five parameter types (A-E) and the four fundamental overlap patterns identified in the main paper:

1. Complete Orthogonality Pattern ($V_1^T V_2 = 0$ and $U_1^T U_2 = 0$): This corresponds to **Type A** parameters only. Tasks operate in completely disjoint input-output subspaces. No parameter conflicts are possible.

2. Row Space Overlap Pattern ($V_1^T V_2 \neq 0$ and $U_1^T U_2 = 0$): This corresponds to **Type B** parameters. Tasks process overlapping input directions but generate orthogonal outputs. Conflicts arise in input processing but not output generation.

3. Column Space Overlap Pattern ($V_1^T V_2 = 0$ and $U_1^T U_2 \neq 0$): This corresponds to **Type C** parameters. Tasks process orthogonal inputs but generate overlapping outputs. Conflicts arise in output generation but not input processing.

4. Both Spaces Overlap Pattern ($V_1^T V_2 \neq 0$ and $U_1^T U_2 \neq 0$): This corresponds to **Types D and E** parameters. **Type D** involves direct overlap at same positions, with D$^+$ representing same-sign modifications (potential amplification) and D$^-$ representing opposite-sign modifications (potential cancellation). **Type E** involves structural overlap through shared spaces. This pattern exhibits the most severe conflicts but also potential for synergies.

Complete Parameter Type Definitions with Mathematical Formulations

Based on the global coordinate system analysis, we provide the complete and mathematically rigorous definitions for classifying parameters in $\tilde{C}_2$ (the second task vector in global coordinates) into six types according to their relationship with $\tilde{C}_1$:

Type A - Complete Orthogonality Pool: For any position (r, c) where task 2 modifies parameters, Type A represents the ideal scenario of complete orthogonality:

$$\begin{aligned}\mathcal{A} = \{(r,c) :& \tilde{C}_2[r,c] \neq 0 \wedge \tilde{C}_1[r,:] = \mathbf{0} \\ & \wedge \tilde{C}_1[:,c] = \mathbf{0}\}.\end{aligned} \tag{25}$$

This means: Task 2 modifies the parameter at global coordinate position (r, c); Task 1 has zero modifications across the entire row r (i.e., $\tilde{C}_1[r, j] = 0$ for all $j \in \{1, 2, \ldots, n\}$); Task 1 has zero modifications across the entire column c (i.e., $\tilde{C}_1[i, c] = 0$ for all $i \in \{1, 2, \ldots, m\}$). Parameters in this category exhibit complete orthogonality in both input space (column) and output space (row) directions, ensuring conflict-free merging as proven in our main theorem.

Type B - Pure Row Orthogonality Pool:

$$\begin{aligned}\mathcal{B} = \{(r,c) :& \tilde{C}_2[r,c] \neq 0 \wedge \tilde{C}_1[r,:] = \mathbf{0} \\ & \wedge \tilde{C}_1[:,c] \neq \mathbf{0}\}.\end{aligned} \tag{26}$$

This configuration represents: Task 2 modifies parameter at position (r, c); Task 1 has no modifications in row r, ensuring output space orthogonality; Task 1 does modify other positions in column c, creating input space overlap. Conflicts arise in input processing as both tasks respond to the same input direction c, but outputs remain orthogonal.

Type C - Pure Column Orthogonality Pool:

$$\begin{aligned}\mathcal{C} = \{(r,c) :& \tilde{C}_2[r,c] \neq 0 \wedge \tilde{C}_1[r,:] \neq \mathbf{0} \\ & \wedge \tilde{C}_1[:,c] = \mathbf{0}\}.\end{aligned} \tag{27}$$

The complementary scenario to Type B: Task 2 modifies parameter at position (r, c); Task 1 modifies other positions in row r, creating output space overlap; Task 1 has no modifications in column c, ensuring input space orthogonality. Conflicts arise in output generation as both tasks generate outputs in the same direction r, but input processing remains orthogonal.

Type D - Direct Overlap Pool: Direct overlap occurs when both tasks modify the exact same position:

$$\mathcal{D} = \{(r,c) : \tilde{C}_2[r,c] \neq 0 \wedge \tilde{C}_1[r,c] \neq 0\}. \tag{28}$$

This category subdivides based on the sign relationship:

Type D$^+$ - Same-sign Direct Overlap:

$$\begin{aligned}\mathcal{D}^+ = \{(r,c) \in \mathcal{D} :& \text{sign}(\tilde{C}_1[r,c]) \\ & = \text{sign}(\tilde{C}_2[r,c])\}.\end{aligned} \tag{29}$$

Both tasks push the parameter in the same direction, potentially creating over-amplification that may drive the model away from optimal regions for either task.

Type D$^-$ - Opposite-sign Direct Overlap:

$$\begin{aligned}\mathcal{D}^- = \{(r,c) \in \mathcal{D} :& \text{sign}(\tilde{C}_1[r,c]) \\ & \neq \text{sign}(\tilde{C}_2[r,c])\}.\end{aligned} \tag{30}$$

Tasks push the parameter in opposite directions, creating direct cancellation that can eliminate important features for both tasks.

Type E - Structural Overlap Pool: The most complex overlap pattern where tasks share overlapping row and column spaces but don't directly conflict at the same position:

$$\begin{aligned}\mathcal{E} = \{(r,c) :& \tilde{C}_2[r,c] \neq 0 \wedge \tilde{C}_1[r,c] = 0 \\ & \wedge \tilde{C}_1[r,:] \neq \mathbf{0} \wedge \tilde{C}_1[:,c] \neq \mathbf{0}\}.\end{aligned} \tag{31}$$

This represents indirect structural conflicts where: Task 2 modifies position (r, c); Task 1 doesn't modify position (r, c) directly; Task 1 does modify other positions in both row r and column c; Both row space and column space overlap through different matrix positions. Type E creates complex interference patterns that can be either harmful or beneficial depending on the specific parameter relationships and the geometric structure of the tasks' learned representations.

Mathematical Completeness and Mutual Exclusivity: The six parameter types form a complete partition of all non-zero elements in $\tilde{C}_2$:

$$\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}^+ \cup \mathcal{D}^- \cup \mathcal{E} = \{(r, c) : \tilde{C}_2[r, c] \neq 0\}. \quad (32)$$

The sets are mutually disjoint:

$$\mathcal{A} \cap \mathcal{B} \cap \mathcal{C} \cap \mathcal{D}^+ \cap \mathcal{D}^- \cap \mathcal{E} = \emptyset. \quad (33)$$

For practical implementation, each non-zero position (r, c) in $\tilde{C}_2$ can be classified by checking the conditions sequentially: first determining if there is direct overlap ($\tilde{C}_1[r, c] \neq 0$), then examining row and column overlaps to distinguish between the remaining types.

Experimental Details

Experimental Setup

Hardware Setup. All experiments are conducted on a high-performance computing cluster equipped with 4 NVIDIA A100 GPUs (80GB each) and 2TB system memory. This configuration ensures sufficient computational resources for handling large-scale model merging operations and SVD decompositions on 7B and 13B parameter models.

Implementation Details. Our GLOBA framework is implemented using PyTorch 2.4.0 with CUDA 12.1 support. SVD decompositions are performed using PyTorch's native `torch.linalg.svd` function with full matrices computation enabled to ensure numerical stability. All matrix operations leverage GPU acceleration through cuBLAS libraries for optimal performance.

Experimental Reproducibility. Since our method involves deterministic mathematical operations (SVD decomposition, matrix projection, and reconstruction) without any random sampling or stochastic components, each experiment is conducted exactly once. The absence of random seeds or probabilistic elements ensures that repeated runs would yield identical results, making multiple runs unnecessary for statistical significance.

Hyperparameter Settings. The primary hyperparameter in our framework is the energy-based pruning threshold η, which determines the fraction of total energy retained after pruning. As detailed in the following subsection, η values are selected through systematic analysis to maximize information retention while filtering noise effectively.

Inference Details. We use vLLM for inference with temperature set to 0.0 for greedy decoding. The maximal number of generated tokens is 1,024 on GSM8K, and 2,048 on the other four datasets. Experiments are conducted on NVIDIA A100 GPUs.

Energy-based Pruning Threshold Selection

The energy-based pruning threshold η plays a crucial role in our framework's effectiveness. Rather than using a universal threshold, we adopt a model-specific approach that balances information retention with noise reduction. Figure 1 illustrates the cumulative energy distribution for different threshold values.

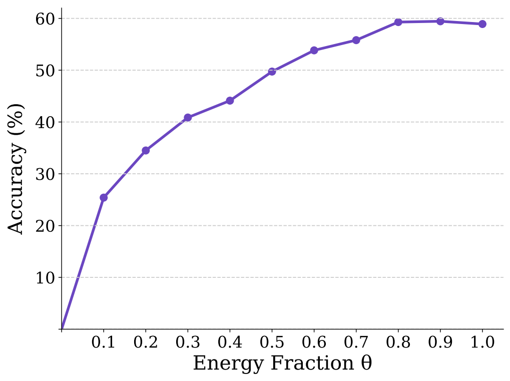

Figure 1: Cumulative energy distribution analysis for threshold selection. The curve shows rapid energy concentration in top singular values, with energy threshold of 0.8 enabling parameter pruning without performance degradation.

Our threshold selection follows a principled approach:

Step 1: Energy Curve Analysis. We compute the cumulative energy curve $E(\eta) = \sum_{i=1}^{k(\eta)} \sigma_i^2 / \sum_{j=1}^{r} \sigma_j^2$ where $k(\eta)$ represents the number of singular values retained at threshold η and r is the total rank.

Step 2: Knee Point Identification. We identify the knee point where the marginal energy gain diminishes significantly. As shown in Figure 1, the energy curve exhibits rapid initial growth followed by saturation, indicating that most important information is captured by a small fraction of parameters.

Step 3: Performance Validation. We validate that the selected threshold preserves >95% of original task performance while achieving effective noise reduction. This ensures that pruning removes irrelevant variations without compromising task-critical information.

Model Scale	Task Combination	Task 1 (η)	Task 2 (η)
7B	Math + Code	0.95	0.80
7B	Math + Instruction	0.80	0.80
7B	Instruction + Code	0.70	0.99
13B	Math + Code	0.80	0.80
13B	Math + Instruction	0.80	0.80
13B	Instruction + Code	0.80	0.80

Table 1: Energy-based pruning threshold configurations across different model scales and task combinations.

Table 1 provides the specific thresholds used across different model combinations, determined through this systematic approach.

Detailed Analysis of 7B Model Experiments

WizardMath-7B-V1.0 + llama2_7b_code Merging. This combination represents the most challenging merging scenario, where severe parameter conflicts lead to catastrophic performance degradation. The merging of WizardMath-7B-V1.0 (mathematical reasoning) and llama2_7b_code (code generation) demonstrates fundamental incompatibilities between these two capabilities. Table 2 presents comprehensive results with detailed analysis.

Configuration	**GSM8K**	**MBPP**
Math Model Only	51.63	/
Code Model Only	/	22.80
Math + Type A (Complete Orth.)	**51.48**	**18.80**
Math + Type B (Row Orth.)	43.06	16.00
Math + Type C (Column Orth.)	47.46	10.40
Math + Type E (Structural)	48.29	15.40
Math + Type D^+ (Same-sign)	5.38	0
Math + Type D^- (Opposite-sign)	0	12.80
All Patterns Combined (TA)	43.14	8.80

Table 2: Detailed 7B Math-Code merging analysis with energy thresholds: $\eta = 0.95$ (math), $\eta = 0.8$ (code).

Selection Rationale: Among all merging configurations tested, Type A (Complete Orthogonality) demonstrates superior performance preservation across both domains. This configuration maintains mathematical reasoning capability at 51.48% on GSM8K (compared to 51.63% for the original math model) while achieving 18.80% on MBPP for code generation tasks. The orthogonal pattern selection strategy proves most effective in minimizing cross-domain interference.

Conflict Analysis: The results reveal severe incompatibilities between mathematical reasoning and code generation capabilities. Type D patterns exhibit catastrophic performance degradation: D^+ (Same-sign) reduces GSM8K performance to merely 5.38% while completely eliminating MBPP capability, and D^- (Opposite-sign) entirely destroys mathematical reasoning (0% on GSM8K) while achieving only 12.80% on MBPP. Even the more conservative Types B, C, and E show substantial mathematical reasoning degradation, with GSM8K scores ranging from 43.06% to 48.29%. The All Patterns Combined approach yields moderate performance (43.14% GSM8K, 8.80% MBPP), confirming that selective orthogonal merging outperforms comprehensive pattern integration in this conflict-dominated scenario.

WizardMath-7B-V1.0 + Llama-2-7b-instruct Merging. This scenario demonstrates the complexity of parameter interactions between mathematical reasoning and instruction-following capabilities, requiring sophisticated scaling strategies to balance beneficial and harmful patterns. Table 3 shows our comprehensive optimization process.

Parameter Tuning Methodology:

Stage 1 - Individual Pattern Analysis & Beneficial Pattern Optimization: We first evaluate each pattern type individually to understand their effects. Types A, B, C, E, and D^+ all improve mathematical reasoning (GSM8K scores ranging from 50.27 to 52.16 compared to 49.20 baseline) while providing modest instruction-following gains (AlpacaEval scores from 13.40 to 22.45 versus 13.00 baseline). However, Type D^- significantly degrades mathematical performance to 44.43 while providing limited instruction improvement to 16.26. The naive combination of all patterns yields suboptimal results (45.56 GSM8K, 38.57 AlpacaEval), indicating the need for careful scaling.

Configuration	**GSM8K**	**AlpacaEval**
Math Model Only	49.20	13.00
Instruct Model Only	/	41.88
Individual Pattern Analysis		
Math + Type A	50.27	13.40
Math + Type B	52.16	18.67
Math + Type C	51.33	17.09
Math + Type E	51.10	22.45
Math + Type D^+	52.01	16.57
Math + Type D^-	44.43	16.26
All Patterns Combined	45.56	38.57
Stage 1: Beneficial Pattern Scaling (Remove D^-)		
Math + 0.9×(C2-D^-)	47.99	32.33
Math + 0.8×(C2-D^-)	50.57	30.61
Math + 0.7×(C2-D^-)	53.22	30.26
Math + 0.6×(C2-D^-)	53.53	27.42
Math + 0.5×(C2-D^-)	52.31	21.89
Stage 2: D^- Pattern Reintroduction		
Base: 0.7×(C2-D^-)	53.22	30.26
+ 1.2×D^-	**49.96**	**39.14**
+ 1.1×D^-	51.25	34.49
+ 1.0×D^-	49.73	34.00
+ 0.9×D^-	50.19	33.79
+ 0.8×D^-	51.63	31.84

Table 3: Comprehensive 7B Math-Instruction merging with two-stage parameter optimization with energy thresholds: $\eta = 0.80$ (math), $\eta = 0.80$ (instruction). Stage 1 identifies beneficial patterns and optimal scaling coefficient of 0.7. Stage 2 reintroduces conflicting D^- patterns with amplification coefficient of 1.2 to recover instruction capability while maintaining math performance.

We systematically apply different scaling coefficients to the beneficial patterns (excluding D^-), testing values from 0.5 to 0.9. The results show that a scaling coefficient of 0.6 achieves peak mathematical performance (53.53 GSM8K) but with reduced instruction capability (27.42 AlpacaEval), while a coefficient of 0.7 provides the best balance (53.22 GSM8K, 30.26 AlpacaEval) for subsequent optimization.

Stage 2 - Conflicting Pattern Reintroduction: Using the beneficial pattern configuration with scaling coefficient 0.7 as the base (53.22 GSM8K, 30.26 AlpacaEval), we systematically reintroduce the D^- pattern with different amplification coefficients ranging from 0.8 to 1.2. Higher amplification of the D^- pattern improves instruction-following capability but reduces mathematical performance. An amplification coefficient of 1.2 for the D^- pattern achieves the optimal trade-off: 49.96 GSM8K (maintaining strong mathematical reasoning) and 39.14 AlpacaEval (approaching the target instruction performance of 41.88).

Selection Rationale: The final configuration applies a scaling coefficient of 0.7 to beneficial patterns combined with an amplification coefficient of 1.2 for the D^- pattern, representing the optimal balance in this mixed-conflict scenario. It preserves 101.5% of baseline mathematical performance (49.96 versus 49.20) while achieving 93.5% of target instruction-following capability (39.14 versus 41.88), demonstrating successful integration of conflicting yet complementary capabilities through strategic parameter scaling.

Llama-2-7b-instruct + llama2_7b_code Merging. This combination reveals remarkable synergistic effects between instruction-following and code generation capabilities, where all overlap patterns contribute positively to performance. Table 4 demonstrates this exceptional scenario.

Configuration	AlpacaEval	MBPP
Instruct Model Only	45.96	0.20
Code Model Only	/	18.40
Code + Type E	8.45	21.00
Code + Type D^+	11.02	28.20
Code + Type D^-	8.40	26.00
All Patterns Combined	**29.04**	**31.60**

Table 4: 7B Instruction-Code merging showing remarkable synergistic effects with energy thresholds: $\eta = 0.70$ (instruction), $\eta = 0.99$ (code).

Synergy Analysis: All individual overlap types provide substantial improvements to the llama2_7b_code model's code generation capabilities on MBPP:

- Type E (Structural overlap): Achieves 21.00 MBPP score, representing a 14.1% improvement over the original code model (18.40), indicating beneficial structural complementarity between instruction-following and code generation capabilities.
- Type D^+ (Same-sign overlap): Reaches 28.20 MBPP score, demonstrating a 53.3% improvement through parameter amplification effects where aligned modifications enhance code generation performance.
- Type D^- (Opposite-sign overlap): Attains 26.00 MBPP score with a 41.3% improvement, showing that apparent parameter conflicts can create beneficial balancing effects that enhance overall capability.

The instruction-following performance varies across individual patterns (AlpacaEval scores of 8.40 to 11.02), but all configurations maintain basic instruction-following capability while significantly enhancing code generation.

Selection Rationale: The All Patterns Combined configuration achieves optimal performance across both domains: 29.04 on AlpacaEval for instruction-following and 31.60 on MBPP for code generation. This represents a remarkable 71.7% improvement in code generation capability over the original llama2_7b_code model (31.60 versus 18.40) while establishing substantial instruction-following capability. The synergistic nature of this combination eliminates the need for complex parameter optimization, as the natural integration of all patterns produces superior results. This validates our hypothesis that instruction-following and code generation capabilities exhibit complementary parameter structures that mutually enhance performance when combined.

Detailed Analysis of 13B Model Experiments

WizardMath-13B-V1.0 + llama-2-13b-code-alpaca Merging. The 13B models exhibit different conflict patterns compared to 7B models, requiring adapted strategies. The combination of WizardMath-13B-V1.0 (mathematical reasoning) and llama-2-13b-code-alpaca (code generation) shows more nuanced behavior than their 7B counterparts. Table 5 shows detailed results.

Configuration	GSM8K	MBPP
Math Model Only	59.29	/
Code Model Only	/	27.00
Math + Type A	59.06	20.40
Math + Type B	58.68	20.40
Math + Type C	58.83	20.80
Math + Type E	59.36	17.00
Math + Type D^+	61.18	16.80
Math + Type D^-	**59.67**	**21.80**
All Patterns Combined	60.50	7.00

Table 5: 13B Math-Code merging with energy thresholds: $\eta = 0.80$ (math), $\eta = 0.80$ (code).

Unlike the catastrophic failures observed in the 7B WizardMath + llama2_7b_code combination, the 13B models demonstrate more stable merging characteristics across all pattern types. All individual patterns maintain reasonable mathematical reasoning performance (GSM8K scores ranging from 58.68 to 61.18) while achieving meaningful code generation capabilities (MBPP scores from 16.80 to 21.80).

The pattern-specific analysis reveals:

- Type D^+ achieves the highest mathematical reasoning performance (61.18 GSM8K) but with reduced code generation capability (16.80 MBPP).
- Type D^- maintains strong mathematical reasoning (59.67 GSM8K) while delivering the best code generation performance (21.80 MBPP).
- Types A, B, and C show moderate performance in both domains (GSM8K: 58.68-59.06, MBPP: 20.40-20.80).
- Type E provides slight mathematical improvement (59.36 GSM8K) but with the lowest code generation performance (17.00 MBPP).

The All Patterns Combined approach yields mixed results: strong mathematical reasoning (60.50 GSM8K) but severely degraded code generation (7.00 MBPP), indicating that comprehensive pattern integration is counterproductive for this model combination.

Selection Rationale: We select the Type D^- configuration as it achieves the optimal balance between mathematical reasoning and code generation capabilities. With 59.67 GSM8K and 21.80 MBPP, this configuration maintains competitive mathematical performance while delivering the highest code generation capability among all tested patterns. The Type D^- pattern effectively leverages the complementary aspects of opposite-sign parameter modifications, creating beneficial interactions that enhance code generation without significantly compromising mathematical reasoning abilities.

WizardMath-13B-V1.0 + WizardLM-13B-V1.2 Merging. The combination of WizardMath-13B-V1.0 (mathematical reasoning) and WizardLM-13B-V1.2 (instruction-following) demonstrates strong synergistic patterns that benefit from careful scaling. Table 6 shows the optimization process.

Configuration	**GSM8K**	**AlpacaEval**
Math Model Only	58.98	51.79
Instruct Model Only	/	81.81
Math + Type A	59.21	54.06
Math + Type B	59.89	62.05
Math + Type C	59.21	62.86
Math + Type E	63.84	67.84
Math + Type D^+	59.44	50.32
Math + Type D^-	56.63	63.25
All Patterns Combined	53.45	70.54
0.6× Scaling	**63.61**	**77.58**

Table 6: 13B Math-Instruction merging showing strong synergistic patterns with energy thresholds: $\eta = 0.80$ (math), $\eta = 0.80$ (instruction).

Synergy Identification: The individual pattern analysis reveals that most overlap types provide mutual benefits for both mathematical reasoning and instruction-following capabilities. Notably, Type E demonstrates exceptional synergistic effects, achieving 63.84 GSM8K for mathematical reasoning and 67.84 AlpacaEval for instruction-following, representing substantial improvements over the baseline math model performance (58.98 GSM8K, 51.79 AlpacaEval). Types B and C also show strong performance with GSM8K scores of 59.89 and 59.21 respectively, while significantly enhancing instruction-following capabilities to 62.05 and 62.86 AlpacaEval.

However, the All Patterns Combined approach reveals a critical optimization challenge: while it achieves strong instruction-following performance (70.54 AlpacaEval), it causes substantial degradation in mathematical reasoning (53.45 GSM8K), indicating that naive pattern combination leads to over-modification and parameter space distortion.

Scaling Strategy: To address the over-modification issue, we apply a uniform scaling coefficient of 0.6 to all combined patterns. This scaling approach prevents excessive parameter modifications while preserving the beneficial synergistic interactions between the two models. The scaled configuration achieves remarkable results: 63.61 GSM8K and 77.58 AlpacaEval, representing optimal performance across both domains.

Selection Rationale: The configuration with scaling coefficient 0.6 represents the optimal balance point, achieving 107.8% of baseline mathematical performance (63.61 versus 58.98) while attaining 94.8% of target instruction-following capability (77.58 versus 81.81). This configuration successfully exploits the synergistic effects between mathematical reasoning and instruction-following capabilities without the parameter space distortion observed in the unscaled combination. The scaling approach enables us to harness the complementary nature of these two capabilities while maintaining stability and performance across both domains.

WizardLM-13B-V1.2 + llama-2-13b-code-alpaca Merging. This combination demonstrates the ideal scenario where Type A (Complete Orthogonality) achieves optimal performance between instruction-following and code generation capabilities. Table 7 validates our theoretical predictions.

Configuration	**AlpacaEval**	**MBPP**
Instruct Model Only	81.29	36.80
Code Model Only	/	27.80
Instruct + Type A	**82.12**	**36.80**
Instruct + Type B	81.12	36.60
Instruct + Type C	79.52	36.60
Instruct + Type E	81.68	34.20
Instruct + Type D^+	79.97	35.20
Instruct + Type D^-	81.19	36.00
All Patterns Combined	82.19	34.80

Table 7: 13B Instruction-Code merging demonstrating perfect orthogonal scenario with energy thresholds: $\eta = 0.80$ (instruction), $\eta = 0.80$ (code).

Perfect Orthogonality Validation: The Type A configuration demonstrates remarkable orthogonal merging characteristics, achieving 82.12 AlpacaEval (a slight improvement over the baseline 81.29) while exactly preserving the code generation performance at 36.80 MBPP. This represents near-perfect conflict-free merging as theoretically predicted for orthogonal parameter modifications, where the integration enhances instruction-following capability without any degradation to code generation performance.

Comparative Analysis: Other overlap types show varying degrees of performance trade-offs:

- Types B and C maintain similar instruction performance (81.12 and 79.52 AlpacaEval respectively) but with slight code generation degradation (36.60 MBPP).
- Type E achieves strong instruction performance (81.68 AlpacaEval) but shows more significant code generation reduction (34.20 MBPP).

- Types D^+ and D^- demonstrate moderate performance across both domains (79.97-81.19 AlpacaEval, 35.20-36.00 MBPP).

The All Patterns Combined approach achieves the highest instruction-following performance (82.19 AlpacaEval) but at the cost of reduced code generation capability (34.80 MBPP), confirming that comprehensive pattern integration introduces unnecessary conflicts even in compatible task combinations.

Selection Rationale: Type A represents the theoretically optimal solution for this model combination. It successfully enhances instruction-following capability (82.12 versus 81.29 AlpacaEval) while perfectly preserving code generation performance (maintaining exactly 36.80 MBPP). This configuration validates our hypothesis that orthogonal parameter modifications enable conflict-free merging, making it the ideal choice when the goal is to enhance one capability without compromising the other.

Distinction from Orthogonality-related Work

While Task Singular Vectors (TSV) has explored orthogonality in model merging, our approach differs fundamentally in both theoretical foundation and practical implementation:

Task Singular Vectors (TSV) operates primarily in computer vision domains and leverages the low-rank nature of task vectors through layer-wise SVD decomposition. TSV uses whitening transformations to enforce orthogonality between singular vectors from different tasks, operating within individual task coordinate systems. However, this approach faces significant limitations when applied to large language models and complex NLP tasks:

1. **Low-rank Assumption Limitation**: TSV relies on the low-rank property of task vectors, which holds for simple computer vision tasks but breaks down for complex NLP tasks in LLMs where parameter modifications exhibit high-dimensional, non-low-rank characteristics.
2. **Orthogonality Enforcement vs. Natural Discovery**: TSV enforces orthogonalization through post-hoc transformations, which can distort original task representations. In contrast, GLOBA constructs a global coordinate system that naturally reveals existing orthogonal and non-orthogonal relationships without forcing artificial orthogonalization.
3. **Limited Scope of Analysis**: TSV focuses solely on maximizing orthogonality between singular vectors, overlooking the potential benefits of non-orthogonal components. Our analysis demonstrates that non-orthogonal overlaps can provide substantial performance gains through beneficial parameter interactions.

Key advantages of our approach:

1. **Universal Applicability**: Our method is applicable to both low-rank scenarios (simple tasks) and high-dimensional, non-low-rank scenarios (complex NLP tasks in LLMs), while TSV is limited to low-rank cases.
2. **Global Coordinate System**: We construct a unified global basis that captures the complete parameter relationship landscape across tasks.
3. **Comprehensive Pattern Classification**: We provide a complete taxonomy of parameter relationships (Types A-E) that goes beyond binary orthogonal/non-orthogonal distinctions.
4. **Non-orthogonal Benefits Recognition**: We systematically identify and leverage beneficial non-orthogonal components, recognizing that conflicts are not always detrimental and can contribute to enhanced capabilities.

Impact of Low-rank Structure

GLOBA is designed to be universally applicable to both low-rank and non-low-rank scenarios without information loss during the transformation from task vectors to task interaction matrices. The key distinction lies in computational efficiency rather than methodological limitations.

Prior to the merging operations, we perform SVD decomposition on task vectors and apply energy-based truncation to retain the most significant singular values. This preprocessing step serves different purposes depending on the intrinsic rank structure of the tasks:

Low-rank Scenarios: When task vectors exhibit strong low-rank properties (common in simple tasks or parameter-efficient fine-tuning), the energy-based truncation naturally captures the essential parameter modifications while discarding noise. If task vectors have intrinsic ranks much smaller than the full parameter matrix dimensions, this truncation significantly reduces the dimensions of the resulting C matrices in the global coordinate system. The computational benefits include:

- Dramatically reduced C matrix dimensions from the original parameter space size to much smaller interaction spaces.
- Significantly lower computational complexity for orthogonal pattern decomposition.
- Enhanced memory efficiency during the merging process.

Non-low-rank Scenarios: For complex NLP tasks where task vectors do not exhibit strong low-rank properties, our method remains fully applicable. The energy-based truncation will retain a larger proportion of singular values to preserve the necessary information, resulting in larger C matrices. While computational costs are higher, the orthogonal decomposition and pattern classification framework operates identically without any methodological constraints, ensuring the same theoretical guarantees and practical effectiveness.

Limitations and Future Work

While GLOBA demonstrates significant improvements in model merging, the primary limitation that warrants future investigation is the extension from two-task analysis to multi-task scenarios:

Our current analysis focuses on pairwise task interactions, providing detailed geometric insights into how two tasks interact through orthogonal and non-orthogonal parameter relationships. The GLOBA framework itself is inherently applicable to multi-task merging scenarios, as the global coordinate system construction and orthogonal pattern decomposition can accommodate multiple task vectors simultaneously.

However, extending to many-task scenarios introduces significantly increased complexity:

- **Higher-order Interactions**: With n tasks, the number of pairwise interactions scales as $O(n^2)$, and higher-order interactions among three or more tasks create additional complexity layers that require systematic analysis.
- **Pattern Classification Complexity**: While our six-pattern taxonomy (Types A-E) provides comprehensive coverage for two-task interactions, multi-task scenarios may exhibit emergent interaction patterns that require extended classification frameworks.
- **Optimization Strategy**: The scaling coefficient optimization becomes substantially more complex when balancing beneficial and conflicting patterns across multiple task pairs simultaneously.

Future work should focus on developing systematic approaches to analyze and manage these multi-task interaction complexities while leveraging the solid theoretical foundation that GLOBA provides for understanding parameter relationships in geometric space. This extension would enable more sophisticated multi-task model development and advance our understanding of multi-task model merging strategies.