# CORTEX: LEARNING TO SHARE AND SPECIALIZE IN DENSE LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Large language models are trained on heterogeneous data mixtures, where different knowledge domains require both shared knowledge and specialization. Existing modular approaches typically impose explicit components or discover modules through interpretability analysis after training. In this work, we propose CORTEX, a learning dynamics-inspired framework that learns internal modularization within dense language models. CORTEX partitions trainable matrices into parameter groups and learns module assignments from domain-conditioned gradient and cross-domain gradient similarity. We introduce the selective lesion score and module-domain mutual information to characterize the target-domain lesion effects and alignment, and analyze how module assignment affects the trade-off between assignment bias and update magnitude. Experiments with 160M, Qwen3-8B, and Qwen3-32B backbone models show that CORTEX achieves the highest synthetic-domain exact match and largest average perplexity reduction, while remaining competitive on real-domain evaluations and forming identifiable modules.

## 1 INTRODUCTION

Large language models (LLMs) have demonstrated outstanding capabilities across various domains (Brown et al., 2020; Yang et al., 2025). They are trained on a wide range of data mixtures. Natural language, code, scientific text, legal documents, mathematical reasoning, and instruction-following data all require the model to capture different patterns of prediction and reasoning (Gao et al., 2020; Xie et al., 2023; Ye et al., 2025; Liu et al., 2025; Chen et al., 2025). While these knowledge domains share vocabulary, syntax, world knowledge, and reusable computational structure, they require different abstractions, factual associations, and modes of generalization.

This creates a trade-off between sharing and specialization. On the one hand, a fully shared dense model allows different knowledge domains to use the same parameters. However, gradients from different domains may favor conflicting updates to the same parameters, which may create interference during training. On the other hand, a fully partitioned model assigns separate model parameters to different knowledge domains. This allows domain-specific updates but limits parameter sharing for patterns that are common across domains.

Existing modular approaches usually take modularity as an explicit architectural component to support domain-specific updates when gradients from different domains favor conflicting updates. Mixture-of-experts (MoE) introduces separate expert blocks and uses routing to activate them for different inputs, showing that specialization can improve the capacity and efficiency under heterogeneous training data (Shazeer et al., 2017; Dai et al., 2024; Muennighoff et al., 2025; Shi et al., 2025). Parameter-efficient adaptation approaches, such as low-rank adaptation (LoRA), provide another form of specialization by introducing task-specific parameters (Hu et al., 2022; Liu et al., 2024; Huang et al., 2025). Subnetwork-based approaches identify sparse subnetworks (Frankle and Carbin, 2019) or allocate parameters to tasks (Mallya and Lazebnik, 2018). Although the aforementioned approaches support specialization, they do not specify how to jointly learn shared and domain-specialized assignments over the existing model parameters on a heterogeneous data mixture.

Another related line of work studies the multi-task gradient from the optimization perspective. These works use learning dynamics to balance task losses and to reduce the cross-task interference caused by conflicting updates to shared parameters (Chen et al., 2018; Sener and Koltun, 2018; Yu et al., 2020; Chen et al., 2020; Liu et al., 2021). However, these works modify the global optimization



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

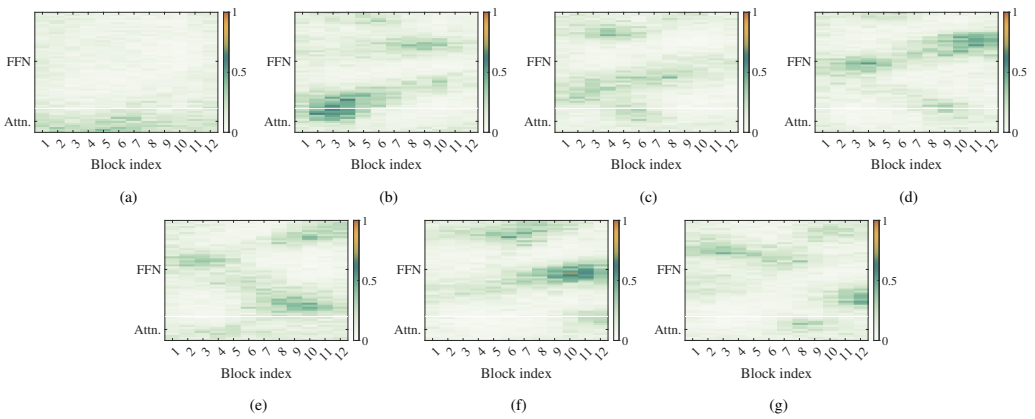



Figure 1: Visualization of CORTEX modularization on the 160M backbone model trained on the synthetic benchmark. The heatmaps correspond to the shared and six knowledge domains, including (a) Shared, (b) COPY, (c) REVERSE, (d) ADD, (e) LOOKUP, (f) LOOKUP→ADD, and (g) REVERSE→LOOKUP, respectively. The columns denote Transformer blocks from 1 to 12, and the rows denote parameter groups within attention heads (Attn.) and the feed-forward network (FFN).

trajectory or task weights and do not learn a persistent modularization. This motivates us not only to adjust the model updates, but also to form modularization inside the dense parameter space.

Furthermore, studies from dense Transformer-based models suggest that internal functional structure can be observed without fully partitioning the model. Prior works have identified specialized attention heads, sparse features, and knowledge circuits inside dense Transformer models (Voita et al., 2019; Elhage et al., 2021; Cunningham et al., 2024; Yao et al., 2024; Gao et al., 2025). These works suggest that dense LLMs contain localized modular structures. However, most of these works discover such structures after training. This leaves a gap between observing modularities in dense LLMs and training an LLM that can be organized as shared and specialized modules.

The aforementioned discussion raises the following question:

*Can modularity be formed inside dense LLMs by shaping learning dynamics?*

Answering this question imposes the following requirement: *The model must identify shared and specialized structures inside the dense parameter space, learn stable module assignments during training, and align the learned modules with heterogeneous knowledge domains.* To achieve this goal, we develop a framework called CORTEX, which supports modularization within dense LLMs. Fig. 1 shows the modularization of CORTEX. Our contributions are summarized as follows:

- We propose CORTEX, a learning dynamics-inspired framework for learning modularization in dense LLMs. CORTEX learns soft assignments of parameter groups, forming a shared module and multiple specialized modules during training.
- We develop a domain-conditioned module assignment mechanism. CORTEX uses domain-conditioned group gradients to determine the module assignment: parameter groups with cross-domain aligned gradients are assigned to the shared module, while domain-concentrated, less-aligned groups are assigned to the corresponding specialized module.
- We introduce the selective lesion score and module-domain mutual information to characterize the target-domain lesion effect and domain alignment. We further show that they characterize module contribution and gradient concentration, while the module assignment weights control the convergence trade-off.
- We evaluate CORTEX on synthetic and real mixtures with 160M, Qwen3-8B, and Qwen3-32B, against Dense, Random, MoE (Shazeer et al., 2017), MoM (Gong et al., 2024), UpIT (Hui et al., 2025), and Self-MoE (Kang et al., 2025). CORTEX achieves the highest exact match across all six synthetic domains, obtains the largest average perplexity reduction, and achieves best or competitive results on real domains while forming identifiable modules.

## 2 Related Work

**Explicit experts and modularity.** MoE introduces expert blocks and activates only a subset of them for each input. Building on the classical structure (Jacobs et al., 1991), modern sparse MoE systems demonstrate that specialization can improve capacity and efficiency at scale (Shazeer et al., 2017; Fedus et al., 2022; Du et al., 2022; Dai et al., 2024; Muennighoff et al., 2025; Li et al., 2026). Recent analyses further study the behavior and modularization degree of MoE-based language models (Lo et al., 2025). Subnetworks (Frankle and Carbin, 2019), packed architectures (Rusu et al., 2016; Mallya and Lazebnik, 2018), and parameter-efficient adaptation approaches such as LoRA (Houlsby et al., 2019; Hu et al., 2022; Dettmers et al., 2023; Liu et al., 2024; Wang et al., 2025; Huang et al., 2025) show that specialization can be achieved without updating all model parameters. Modularizing-while-training (MwT) (Qi et al., 2024) is trained with modularization objectives to obtain reusable class-specific modules, while MODA (Ngo et al., 2026) regulates activation patterns and extracts modules for reuse and replacement. However, their modularization objectives penalize mask or activation similarity across classes. This can discourage beneficial parameter sharing. CORTEX learns shared and specialized modules within dense LLMs by coupling domain-conditioned module assignment with model updates, without introducing explicit experts.

**Multi-task gradient optimization.** Another related line of work balances task losses or adjusts update directions using task-conditioned gradients (Chen et al., 2018; Sener and Koltun, 2018; Yu et al., 2020; Chen et al., 2020; Liu et al., 2021). These approaches do not learn persistent module assignments within the dense parameter space. Recon (Shi et al., 2023) uses gradient conflicts to select shared layers and copies these layers for different tasks. This changes the model structure and introduces additional parameters. In contrast, CORTEX learns shared and specialized assignments over the existing model parameters during training.

**Implicit modularity and mechanistic interpretability.** Prior work has identified specialized attention heads (Voita et al., 2019), feed-forward layers with key-value memories (Geva et al., 2021), knowledge neurons (Dai et al., 2022), and knowledge circuits (Yao et al., 2024). Feed-forward layers can also be converted into expert-like structures (Zhang et al., 2022), and emergent modularity has been observed in pretrained LLMs (Zhang et al., 2023; Qiu et al., 2024). Recent work further studies sparse features and compositional circuits in language models (Elhage et al., 2021; Cunningham et al., 2024; Gao et al., 2025; Mondorf et al., 2025). They do not make module assignment a trainable objective, and the discovered structures are not necessarily aligned with knowledge domains.

## 3 CORTEX

In this section, we introduce the modular parameterization of CORTEX and its domain-conditioned module assignment and training procedure. A list of key notations is presented in Appendix A.

Let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{K} = \{1, 2, \ldots, K\}$ denote set of inputs, outputs, and $K$ knowledge domains, respectively. We represent a training sample by a triplet $(X, Y, C)$, where $X \in \mathcal{X}$ is the input, $Y \in \mathcal{Y}$ is the output, and $C \in \mathcal{K}$ is the label of the knowledge domain. A knowledge domain may represent a domain (e.g., legal text), a skill (e.g., math reasoning), or a fact, as long as it is captured by a measurable conditional distribution over $(X, Y)$. In particular, $C = k$ means that the training sample is from the $k$-th knowledge domain. Let $\pi_k$ denote the probability that a randomly drawn training sample belongs to the $k$-th knowledge domain. We have $\pi_k = \mathbb{P}(C = k) > 0$ and $\sum_{k=1}^{K} \pi_k = 1$. For each knowledge domain $k \in \mathcal{K}$, let $\mathcal{P}_k$ denote its conditional joint distribution of $(X, Y)$ given $C = k$. Then, the overall training data distribution is a mixture of $\mathcal{P}_k$, which is given by $\mathcal{P} = \sum_{k=1}^{K} \pi_k \mathcal{P}_k$.

### 3.1 Modular Parameterization

To learn module assignments inside a dense model, CORTEX first specifies the granularity at which trainable parameters are allocated to the shared and specialized modules. This granularity should distinguish shared and domain-specific knowledge while keeping the assignment stable during training. Therefore, CORTEX divides each trainable weight matrix into *parameter groups* and learns the assignment weights of each group for the shared and specialized modules. The relationship among knowledge domains, parameter groups, and specialized modules is presented in Fig. 2 (a).

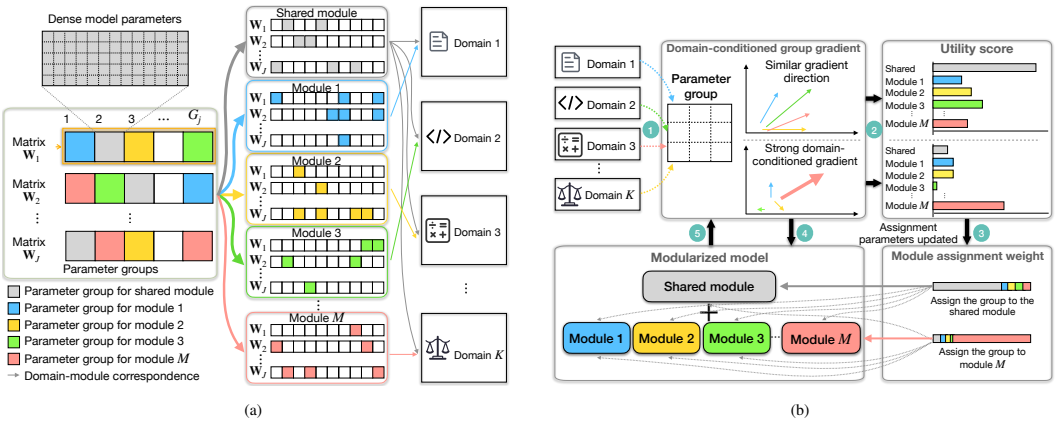



Figure 2: A schematic illustration of CORTEX, including (a) the relationship among knowledge domains, parameter groups, and specialized modules, and (b) the modularization process of CORTEX.

**Parameter group** Let $\mathcal{J} = \{1, 2, \ldots, J\}$ denote the set of $J$ indices of trainable weight matrices in the model. For each index $j \in \mathcal{J}$, let $\mathbf{W}_j \in \mathbb{R}^{d_j^{\text{out}} \times d_j^{\text{in}}}$ denote the $j$-th trainable weight matrix, where $d_j^{\text{out}}$ and $d_j^{\text{in}}$ are its output and input dimensions, respectively. The parameter groups define the allocation granularity within each trainable matrix, and the projection operator selects disjoint groups that can be recombined into the original dense matrix.

**Definition 1** (Parameter group). *For trainable matrix $\mathbf{W}_j$, $j \in \mathcal{J}$, let $\mathcal{G}_j = \{1, 2, \ldots, G_j\}$ denote the index set of $G_j$ corresponding parameter groups. In particular, we represent the projection of the $j$-th trainable parameter matrix onto the $g$-th parameter group by using a linear projection operator as $\Pi_{j,g} : \mathbb{R}^{d_j^{\text{out}} \times d_j^{\text{in}}} \to \mathbb{R}^{d_j^{\text{out}} \times d_j^{\text{in}}}$, $\quad g \in \mathcal{G}_j$. $\Pi_{j,g}(\mathbf{W}_j)$ keeps the entries of the $g$-th parameter group and sets all other entries to zero. It satisfies for arbitrary $g \neq g'$, the supports of two projection operators are disjoint, i.e., $\text{supp}_{\text{str}}(\Pi_{j,g}) \cap \text{supp}_{\text{str}}(\Pi_{j,g'}) = \emptyset$. All parameter groups construct the original matrix, i.e. $\sum_{g \in \mathcal{G}_j} \Pi_{j,g}(\mathbf{W}_j) = \mathbf{W}_j$.*

This projection-based reparameterization allows us to construct each module by combining selected parameter groups. For Transformer backbones, the projections $\Pi_{j,g}$ are chosen along the functional axes. For feed-forward network (FFN) matrices, a parameter group is a block along the intermediate-channel axis. For attention matrices, a parameter group is a block of channels corresponding to one attention head.

**Shared and specialized modules** While there are heterogeneous knowledge domains, they still share a substantial amount of common knowledge. This is because in LLMs, a large fraction of the computation is expected to be common instead of domain-specific. For this reason, we represent the model as a combination of (i) a shared module that captures the common knowledge and (ii) multiple specialized modules that capture heterogeneous knowledge domains. We consider that each specialized module corresponds to a unique knowledge domain. Let $\mathcal{M} = \{1, 2, \ldots, M\}$ denote the index set of specialized modules, and let $m = 0$ denote the shared module. Each specialized module $m \in \mathcal{M}$ is paired with one knowledge domain, denoted by $\kappa(m) \in \mathcal{K}$. This pairing is fixed and one-to-one, with $M = K$.

CORTEX learns soft assignment weights to distribute each parameter group across the shared module and specialized modules. For the $g$-th parameter group in the $j$-th trainable parameter matrix, we define the trainable assignment parameter vector as $\boldsymbol{\psi}_{j,g} = (\psi_{j,g,0}, \psi_{j,g,1}, \ldots, \psi_{j,g,M}) \in \mathbb{R}^{M+1}$. Let $\Psi = \{\boldsymbol{\psi}_{j,g} : j \in \mathcal{J}, g \in \mathcal{G}_j\}$ denote the set of trainable assignment parameter vectors.

**Definition 2** (Module assignment weight). *Let $s_{j,g,m} \in [0, 1], m \in \{0\} \cup \mathcal{M}$ denote the assignment weight of the $g$-th parameter group in the $j$-th trainable parameter matrix assigned to the $m$-th module. $s_{j,g,m}$ is defined as $s_{j,g,m} = \frac{\exp(\psi_{j,g,m})}{\sum_{r=0}^{M} \exp(\psi_{j,g,r})}, j \in \mathcal{J}, g \in \mathcal{G}_j, m \in \{0\} \cup \mathcal{M}$.*

We have $\sum_{m=0}^{M} s_{j,g,m} = 1$. We use this softmax reparameterization since it provides a normalized and differentiable relaxation of discrete module assignment such that the module assignment weight

can be optimized through $\Psi$. We denote $\mathbf{s}_{j,g} = (s_{j,g,0}, s_{j,g,1}, \ldots, s_{j,g,M}) \in \mathbb{R}^{M+1}$ as the assignment weight vector for the $g$-th parameter group of the $j$-th trainable parameter matrix. For the $j$-th trainable parameter matrix $\mathbf{W}_j$, the module-specific component associated with the $m$-th module is defined as $\mathbf{W}_{j,m} = \sum_{g \in \mathcal{G}_j} s_{j,g,m}\, \Pi_{j,g}(\mathbf{W}_j), j \in \mathcal{J}, m \in \{0\} \cup \mathcal{M}$. Each module is a weighted combination of parameter groups, and the same parameter group can contribute to multiple modules. Let $\mathcal{W} = \{\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_J\}$ denote the full model.

### 3.2 Domain-conditioned Module Assignment and Training

CORTEX forms modularization during training by coupling module assignment with model updates. First, we introduce domain-conditioned group gradients to measure how each knowledge domain updates each parameter group. Second, we define the utility scores to determine whether a parameter group is more suitable for the shared module or a specialized module. Finally, the module assignment parameters and backbone model parameters are updated simultaneously using the utility scores and the main task loss, respectively. An overview of CORTEX is presented in Fig. 2 (b).

**Domain-conditioned group gradients** A key question of CORTEX is how to update the assignment weights to form modules. The assignment should reflect how each parameter group represents different knowledge domains. To this end, CORTEX uses domain-conditioned group gradients for module assignment. The intuition is as follows. If several knowledge domains have similar update directions on a certain parameter group, then this parameter group should be assigned to the shared module. On the other hand, if a specific knowledge domain induces a large gradient magnitude on a certain parameter group, then this group is more suitable for a specialized module.

For a mini-batch $\boldsymbol{\xi}$, let $\boldsymbol{\xi}_k = \{(X, Y, C) \in \boldsymbol{\xi} : C = k\}$ denote the subset of samples from the $k$-th knowledge domain. The loss of $\boldsymbol{\xi}_k$ is denoted as $F(\mathcal{W}; \boldsymbol{\xi}_k)$. For the $g$-th parameter group in the $j$-th parameter matrix, its domain-conditioned gradient with respect to (w.r.t.) the $k$-th knowledge domain is defined as $\mathbf{G}_{j,g,k} = \Pi_{j,g}\left(\nabla_{\mathbf{W}_j} F(\mathcal{W}; \boldsymbol{\xi}_k)\right)$, where $\nabla_{\mathbf{W}_j} F(\mathcal{W}; \boldsymbol{\xi}_k) \in \mathbb{R}^{d_j^{\text{out}} \times d_j^{\text{in}}}$ denotes the gradient of $F(\mathcal{W}; \boldsymbol{\xi}_k)$ w.r.t. $\mathbf{W}_j$. The mixture gradient for this parameter group is $\sum_{k \in \mathcal{K}} \pi_k \mathbf{G}_{j,g,k}$.

**Utility score** Based on the domain-conditioned group gradients, we define the utility score as a metric which measures how suitable a parameter group is for the shared module or a specialized module. A higher utility score means that assigning this parameter group to the corresponding module is more consistent with the domain-conditioned gradients.

For the shared module, a parameter group should be useful to multiple knowledge domains, while a specialized module requires that the parameter group can represent a specific knowledge domain. Hence, we define the utility score of the shared module and the $m$-th module as $u_{j,g,0} = \frac{2}{K(K-1)} \sum_{1 \le p < q \le K} \frac{\langle \mathbf{G}_{j,g,p}, \mathbf{G}_{j,g,q} \rangle_F}{\|\mathbf{G}_{j,g,p}\|_F \|\mathbf{G}_{j,g,q}\|_F}$ and $u_{j,g,m} = \frac{\|\mathbf{G}_{j,g,\kappa(m)}\|_F}{\sum_{k' \in \mathcal{K}} \|\mathbf{G}_{j,g,k'}\|_F} - \frac{1}{K-1} \sum_{q \in \mathcal{K} \setminus \{\kappa(m)\}} \frac{\langle \mathbf{G}_{j,g,\kappa(m)}, \mathbf{G}_{j,g,q} \rangle_F}{\|\mathbf{G}_{j,g,\kappa(m)}\|_F \|\mathbf{G}_{j,g,q}\|_F}, m \in \mathcal{M}$, respectively. $\langle \cdot, \cdot \rangle_F$ and $\| \cdot \|_F$ denote the Frobenius inner product and norm, respectively. The inner product is the sum of products of corresponding matrix entries. In particular, $u_{j,g,0}$ measures the average similarity between the domain-conditioned gradient across all pairs of knowledge domains. A larger value of $u_{j,g,0}$ indicates that different domains update this group in similar directions. Hence, the group is better suited to the shared module. For $u_{j,g,m}$, the first term measures the gradient magnitude of the $\kappa(m)$-th domain normalized by the sum across all domains on the same parameter group. The second term measures the average cosine similarity between this domain's gradient and those of the other domains.

**Training procedure** The utility scores defined above provide the training signal for the module assignment. CORTEX updates the module assignments and backbone parameters simultaneously after gradient collection. It calculates the utility scores from the domain-conditioned group gradients accumulated for the current update. We use these scores to define the assignment loss as $\mathcal{L}_{\text{assign}} = -\frac{1}{\sum_{j \in \mathcal{J}} G_j} \sum_{j \in \mathcal{J}} \sum_{g \in \mathcal{G}_j} \sum_{m \in \mathcal{M} \cup \{0\}} s_{j,g,m} u_{j,g,m}$. The assignment parameters $\Psi$ are updated by minimizing $\mathcal{L}_{\text{assign}}$. The backbone parameters are updated according to the main task loss, which is given by $\mathcal{L}_{\text{main}} = \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{P}}\left[F(\mathcal{W}; \boldsymbol{\xi})\right]$.

The shared module is updated using gradients from all knowledge domains, while each specialized module mainly updates its parameters by using the domain-conditioned gradi-

ents. Therefore, the $j$-th trainable parameter matrix is updated by $\mathbf{W}_j \leftarrow \mathbf{W}_j - \eta \sum_{g\in\mathcal{G}_j} \left(s_{j,g,0} \sum_{k\in\mathcal{K}} \pi_k \mathbf{G}_{j,g,k} + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s_{j,g,m} \mathbf{G}_{j,g,\kappa(m)}\right), j \in \mathcal{J}$, where $\eta$ is the learning rate. The algorithm of CORTEX is presented in Algorithm 1 in Appendix B.

## 4 Theoretical Analysis

In this section, we provide theoretical analysis of CORTEX. We first present Proposition 1 to motivate the utility scores. Then, we define two module-level interpretability metrics and introduce Propositions 2 and 3 to characterize specialization and module-domain alignment. In addition, Theorem 1 bounds the average squared gradient norm for the modularized update. We present the detailed assumptions and proof in Appendix C.

**Proposition 1.** *We consider that $\nabla_{\mathbf{W}_j} F(\mathcal{W}; \boldsymbol{\xi}_k)$ is locally Lipschitz continuous in $\mathbf{W}_j$, with all other matrices fixed. For the $g$-th parameter group in the $j$-th trainable matrix, an update using the mixture gradient changes the loss $F(\mathcal{W}; \boldsymbol{\xi}_k)$ by $-\eta \left(\pi_k \|\mathbf{G}_{j,g,k}\|_F^2 + \sum_{q\in\mathcal{K}\setminus\{k\}} \pi_q \langle \mathbf{G}_{j,g,k}, \mathbf{G}_{j,g,q}\rangle_F\right) + O(\eta^2)$.*

Proposition 1 characterizes the first-order loss change when using the mixture gradient. The term $\pi_k \|\mathbf{G}_{j,g,k}\|_F^2$ captures the contribution from the $k$-th domain's own gradient, while positive and negative cross-domain inner products contribute to decreasing and increasing its loss, respectively. These terms motivate the use of gradient magnitude and alignment in the utility scores.

We consider that the number of modules matches knowledge domains, i.e., $M = K$. Let $\mathcal{W}^{\text{out}} = \{\mathbf{W}_1^{\text{out}}, \mathbf{W}_2^{\text{out}}, \ldots, \mathbf{W}_J^{\text{out}}\}$, $\Psi^{\text{out}}$, and $\mathbf{s}_{j,g,m}^{\text{out}}$ denote the model parameters, assignment parameters, and assignment weights after training, respectively. The $m$-th module in the $j$-th trained parameter matrix can be expressed as $\mathbf{W}_{j,m}^{\text{out}} = \sum_{g=1}^{G_j} s_{j,g,m}^{\text{out}} \Pi_{j,g}(\mathbf{W}_j^{\text{out}}), j \in \mathcal{J}, m \in \{0\} \cup \mathcal{M}$. To isolate the contribution of a specialized module, we denote the model without the $m$-th module as $\mathcal{W}^{\text{drop},m} = \{\mathbf{W}_1^{\text{out}} - \mathbf{W}_{1,m}^{\text{out}}, \mathbf{W}_2^{\text{out}} - \mathbf{W}_{2,m}^{\text{out}}, \ldots, \mathbf{W}_J^{\text{out}} - \mathbf{W}_{J,m}^{\text{out}}\}, m \in \mathcal{M}$. We define the empirical risk made with $\mathcal{W}$ on the $k$-th knowledge domain as $R_k(\mathcal{W}) = \mathbb{E}_{\boldsymbol{\xi}_k \sim \mathcal{P}_k}[F(\mathcal{W}; \boldsymbol{\xi}_k)]$.

Then, we adopt the idea of lesion study in neuroscience (Bates et al., 2003; Rorden and Karnath, 2004) to quantify the specialization between a module and a knowledge domain as follows.

**Definition 3** (Selective lesion score)**.** *The lesion effect of removing the $m$-th module on the $k$-th knowledge domain is defined as $\Delta_{m,k}^{\text{les}} = R_k(\mathcal{W}^{\text{drop},m}) - R_k(\mathcal{W}^{\text{out}})$. We define the corresponding selective lesion score as $\text{SLS}(m,k) = \Delta_{m,k}^{\text{les}} - \frac{1}{K-1} \sum_{q\in\mathcal{K}\setminus\{k\}} \Delta_{m,q}^{\text{les}}, k \in \mathcal{K}, m \in \mathcal{M}$.*

Note that a larger selective lesion score indicates that disabling the $m$-th module leads to a larger performance degradation on the $k$-th knowledge domain than on the remaining domains, implying a stronger specialization between the $m$-th module and the $k$-th knowledge domain.

**Proposition 2.** *Removing the $m$-th module increases the empirical risk compared with the trained model if $\text{SLS}(m,k) > -\frac{1}{\pi_k} \sum_{q\in\mathcal{K}\setminus\{k\}} \left(\pi_q + \frac{\pi_k}{K-1}\right) \Delta_{m,q}^{\text{les}}$.*

Proposition 2 shows that the selective lesion score, the lesion effects on the other domains, and the domain mixture weights jointly determine the change in the main task loss after module removal. A larger value of $\text{SLS}(m,k)$ indicates stronger specialization of the $m$-th module for the $k$-th knowledge domain.

In addition, we analyze the degree of alignment between the assigned module and knowledge domains by adopting the idea of mutual information as follows.

**Definition 4** (Module-domain mutual information)**.** *For the $m$-th module and the $k$-th knowledge domain, let $\text{GS}_{m,k} = \frac{\pi_k \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} s_{j,g,m}^{\text{out}} \left\|\mathbf{G}_{j,g,k}^{\text{out}}\right\|_F}{\sum_{k'\in\mathcal{K}} \pi_{k'} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \left\|\mathbf{G}_{j,g,k'}^{\text{out}}\right\|_F}$ denote the gradient share which is the fraction of the total domain-conditioned gradient magnitude that comes from the $k$-th knowledge domain and is assigned to the $m$-th module. We have $\sum_{m\in\{0\}\cup\mathcal{M}} \sum_{k\in\mathcal{K}} \text{GS}_{m,k} = 1$. The module-domain mutual information is defined as $I^{\text{mod}} = \sum_{m\in\{0\}\cup\mathcal{M}} \sum_{k\in\mathcal{K}} \text{GS}_{m,k} \ln \frac{\text{GS}_{m,k}}{\text{GS}_m^{\text{module}} \text{GS}_k^{\text{domain}}}$, where $\text{GS}_m^{\text{module}} = \sum_{k\in\mathcal{K}} \text{GS}_{m,k}, m \in \{0\} \cup \mathcal{M}$, and $\text{GS}_k^{\text{domain}} = \sum_{m\in\{0\}\cup\mathcal{M}} \text{GS}_{m,k}, k \in \mathcal{K}$.*

A larger value of $I^{\mathrm{mod}}$ indicates that the module assignment is more aligned with knowledge domains.

**Proposition 3.** *The upper-bound of $I^{\mathrm{mod}}$ is $-\sum_{k\in\mathcal{K}} \mathrm{GS}_k^{\mathrm{domain}} \ln \mathrm{GS}_k^{\mathrm{domain}}$, which is achieved when each module corresponds to at most one knowledge domain.*

Proposition 3 provides a scale for interpreting the module-domain alignment. The shared module corresponds to multiple knowledge domains, while each module mainly corresponds to a specific knowledge domain. Thus, a larger value of $I^{\mathrm{mod}}$ indicates that the domain-conditioned gradient magnitude is more concentrated on the assigned modules, implying higher module-domain alignment.

While both metrics evaluate the relationship between specialized modules and knowledge domains, the selective lesion score measures whether removing a module causes an increase in the risk for its target knowledge domain than for other domains. The module-domain mutual information measures whether the domain-conditioned gradient magnitude is concentrated on the assigned modules. Then, we present a convergence bound for the modularized update.

**Theorem 1** (Convergence bound). *Let $\mathbf{W}^t$ and $\mathbf{W}^\star$ denote the model at round $t$ and an optimal model. For the modularized update with fixed $\mathbf{W}^1$ and constant learning rate $\eta > 0$, Assumptions 1 and 2 in Appendix C.4 imply $\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla \mathcal{L}_{\mathrm{main}}(\mathbf{W}^t)\|_F^2\right] \le \frac{2}{T\eta}\left(\mathcal{L}_{\mathrm{main}}(\mathbf{W}^1) - \mathcal{L}_{\mathrm{main}}(\mathbf{W}^\star)\right) + \frac{G^2}{T}\sum_{t=1}^{T}\sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}} \pi_{\kappa(m)}\mathbb{E}[A^t_{1,j,g,m}] + \frac{2L\eta(G^2+\sigma^2)}{T}\sum_{t=1}^{T}\sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\mathbb{E}[A^t_{2,j,g}]$, where $A^t_{1,j,g,m} = \left(1 - s^t_{j,g,0} - s^t_{j,g,m}\right)^2$ and $A^t_{2,j,g} = \left(s^t_{j,g,0} + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s^t_{j,g,m}\right)^2$. The expectations are taken over the mini-batches sampled during training.*

In Theorem 1, $L$ is the smoothness constant. For each domain-conditioned group gradient, $G^2$ bounds the squared norm of its conditional expectation, and $\sigma^2$ bounds its conditional variance. The first term is the standard initial-optimality gap term. The second term captures assignment bias. The third term arises from bounding the second moment of the modularized update. Larger shared assignment weights and smaller specialized assignment weights reduce the second term but increase the third term. Hence, module assignment controls the trade-off between the assignment bias and update magnitude.

## 5 Performance Evaluation

We evaluate CORTEX on synthetic and real datasets, with additional results in Appendix D.

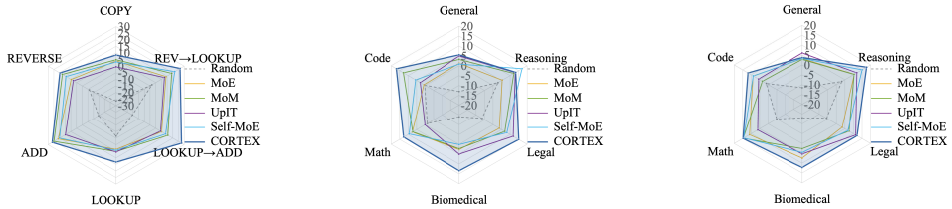
### 5.1 Experimental Setup

**Benchmarks and backbone models** We evaluate CORTEX on two settings for the mixture of knowledge domains. First, we create a synthetic benchmark which contains six knowledge domains: COPY, REVERSE, ADD, LOOKUP, LOOKUP→ADD, and REVERSE→LOOKUP. Domain labels are used during CORTEX training to estimate domain-conditioned group gradients and are used for evaluation. Second, we use a real text knowledge-domain data mixture with general text (C4) (Raffel et al., 2020; Dodge et al., 2021), code (Magicoder-OSS-Instruct-75K) (Wei et al., 2024), math (MathInstruct) (Yue et al., 2024), biomedical text (PubMedAbstractsSubset) (Stuhlmann et al., 2025), legal text (Caselaw Access Project) (Harvard Law School Library Innovation Lab, 2024), and reasoning data (WinoGrande) (Sakaguchi et al., 2020) For the synthetic benchmark, we use a 160M parameter decoder-only model with 12 layers, hidden size 768, and 12 attention heads. For the real-domain benchmark, we use Qwen3-8B and Qwen3-32B (Yang et al., 2025) as dense backbones.

**Baseline schemes** (i) **Dense** trains the same dense model on the entire data mixture without module assignment. (ii) **Random** uses the same domain-conditioned gradient collection and backbone update rule as CORTEX, while keeping the randomly initialized module assignments fixed throughout training. (iii) **MoE** replaces FFN with sparse experts and a router. (iv) **MoM** (Gong et al., 2024) uses token-level routers to select multi-head attention and FFN. It matches **Dense** in the number of backbone parameters and layers applied to each token. (v) We adapt **UpIT** (Hui et al., 2025) using experts from intermediate checkpoints, then train only the router. (vi) We adapt **Self-MoE** (Kang et al., 2025) using the same training data without self-generated samples to train LoRA experts and a router. All baseline schemes use the same data splits, training-token budget, and total number of optimizer updates. The settings are described in Appendix D. CORTEX uses the same backbone as

Table 1: Comparison between CORTEX and the best baseline. We show exact match (EM, in %) for the synthetic benchmark and negative log likelihood (NLL) for the real-domain benchmarks.

| Backbone / Metric | Approach | Domain 1 | Domain 2 | Domain 3 | Domain 4 | Domain 5 | Domain 6 |
|---|---|---|---|---|---|---|---|
| | | **COPY** | **REVERSE** | **ADD** | **LOOKUP** | **LOOKUP→ADD** | **REVERSE→LOOKUP** |
| **160M** | Best Baseline | 98.67±0.21 (MoM) | 97.84±0.27 (MoM) | 96.21±0.33 (MoM) | 96.48±0.32 (UpIT) | 90.82±0.49 (MoM) | 95.71±0.37 (Self-MoE) |
| | **CORTEX** | **99.42±0.18** | **98.83±0.22** | **97.62±0.29** | **98.14±0.26** | **94.76±0.41** | **96.88±0.33** |
| | | **General** | **Code** | **Math** | **Biomedical** | **Legal** | **Reasoning** |
| **Qwen3-8B** | Best Baseline | 1.6257±0.0108 (UpIT) | 1.6578±0.0124 (MoM) | 1.9058±0.0148 (Self-MoE) | 1.7709±0.0134 (UpIT) | 1.8171±0.0140 (UpIT) | **1.7822±0.0128** (Self-MoE) |
| | **CORTEX** | **1.6199±0.0097** | **1.6096±0.0115** | **1.8694±0.0137** | **1.6778±0.0126** | **1.7791±0.0131** | 1.8258±0.0124 |
| | | **General** | **Code** | **Math** | **Biomedical** | **Legal** | **Reasoning** |
| **Qwen3-32B** | Best Baseline | **1.3578±0.0093** (UpIT) | 1.4383±0.0107 (MoE) | 1.5815±0.0122 (MoM) | 1.4878±0.0114 (MoE) | 1.5514±0.0119 (UpIT) | 1.5597±0.0116 (Self-MoE) |
| | **CORTEX** | 1.3820±0.0091 | **1.4051±0.0102** | **1.5798±0.0117** | **1.4344±0.0109** | **1.5426±0.0110** | **1.5279±0.0111** |




(a) (b) (c)

Figure 3: Comparison of the PPL reduction over **Dense** on (a) the 160M model, (b) Qwen3-8B, and (c) Qwen3-32B. The PPL reduction of **Dense** is zero.

Table 2: Comparison of downstream scores between CORTEX and the best baseline on each domain. We show the average ± standard deviation over four seeds.

| Backbone | Approach | General | Code | Math | Biomedical | Legal | Reasoning |
|---|---|---|---|---|---|---|---|
| **Qwen3-8B** | Best Baseline | 0.5785±0.0062 (UpIT) | **0.3097±0.0082** (MoM) | 0.3414±0.0098 (Self-MoE) | 0.7220±0.0059 (UpIT) | 0.6219±0.0069 (UpIT) | **0.4992±0.0091** (Self-MoE) |
| | **CORTEX** | **0.5900±0.0058** | 0.3074±0.0085 | **0.3754±0.0102** | **0.7295±0.0055** | **0.6472±0.0064** | 0.4929±0.0088 |
| **Qwen3-32B** | Best Baseline | **0.6732±0.0050** (UpIT) | 0.3930±0.0068 (MoE) | 0.5322±0.0083 (MoM) | 0.7950±0.0053 (MoE) | 0.6939±0.0059 (UpIT) | 0.6211±0.0076 (Self-MoE) |
| | **CORTEX** | 0.6641±0.0047 | **0.4253±0.0071** | **0.5471±0.0080** | **0.8280±0.0049** | **0.6974±0.0056** | **0.6494±0.0072** |

**Dense** and calculates assignment utilities from the domain-conditioned gradients. CORTEX has a lower training step time than MoE and MoM. In Appendix D, we compare the training step time and peak memory per GPU across five schemes.

### 5.2 Experimental Results

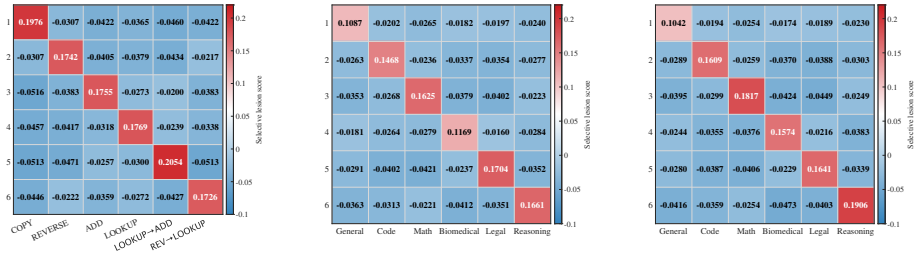
**Learning performance comparison** Table 1 shows that CORTEX consistently outperforms the best baseline on exact match (EM) over all synthetic domains. On the real-domain benchmark, CORTEX achieves the best or competitive negative log likelihood (NLL) for both Qwen3-8B and Qwen3-32B when compared with the baseline schemes, which shows the effectiveness of CORTEX. Fig. 3 compares the perplexity (PPL) reduction of different schemes over **Dense** across datasets and backbones. CORTEX achieves a larger reduction in most cases.

We further perform downstream evaluation for generalization. After training, we evaluate frozen checkpoints without fine-tuning, using zero-shot prompts and greedy decoding with Qwen3 thinking disabled (Appendix D.4). We use MMLU-Pro (Wang et al., 2024) for general knowledge, LiveCodeBench (Jain et al., 2025) for code, MATH/GSM8K (Hendrycks et al., 2021; Cobbe et al., 2021) for mathematics, PubMedQA (Jin et al., 2019) for biomedical QA, LegalBench (Guha et al., 2023) for legal reasoning, and BIG-Bench Hard (BBH) (Suzgun et al., 2023) for reasoning. Table 2 shows that CORTEX achieves higher average scores than the best baseline in nine of the twelve backbone–domain settings. Across four training seeds, the paired gains on mathematics with Qwen3-8B and biomedical QA with Qwen3-32B are $0.0340 \pm 0.0075$ and $0.0330 \pm 0.0042$, respectively. These results show that CORTEX improves the learning performance.

**Shared module analysis** We investigate the effect of the shared module in CORTEX using the 160M backbone on the synthetic benchmark. Table 3 shows that introducing the shared module reduces PPL and NLL and improves EM when compared with CORTEX without the shared module. It also increases $I^{\mathrm{mod}}$ and SLS across all six knowledge domains, indicating stronger domain alignment and selectivity. These results support the shared module design for improving prediction performance and functional specialization.

Table 3: Effect of the shared module on the synthetic benchmark.

| Variant | PPL | NLL | EM | $I^{mod}$ | SLS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | COPY | REVERSE | ADD | LOOKUP | LOOKUP→ADD | REV→LOOKUP |
| CORTEX w/o Shared | 1.1302 | 0.1224 | 95.56 | 1.1182 | 0.1482 | 0.1667 | 0.1694 | 0.1318 | 0.1975 | 0.1558 |
| **CORTEX** | **1.0762** | **0.0734** | **97.61** | **1.4560** | **0.1976** | **0.1742** | **0.1755** | **0.1769** | **0.2054** | **0.1726** |




(a) (b) (c)

Figure 4: Selective lesion scores (SLS) of different modules on (a) 160M model, (b) Qwen3-8B, and (c) Qwen3-32B. Each entry measures the difference between the risk change on a domain and the average change on the remaining domains after removing a module.

**Modularization analysis** Fig. 4 shows positive diagonal SLS values across all three backbones. Appendix D compares learned and shuffled assignments with the same norm of removed weights. The learned assignments achieve higher paired-domain SLS on both 160M and Qwen3-8B. Table 4 shows that CORTEX achieves lower NLL and higher module-domain mutual information than Random across all three backbones. Removing a CORTEX module on its paired domain causes a larger NLL increase, while the average increase on the other domains remains comparable or smaller. Table 5 reports the average maximum assignment weight and average normalized entropy. They are calculated as $\frac{\sum_{j,g} \max_{0\le m\le M} s_{j,g,m}}{\sum_j |\mathcal{G}_j|}$ and $-\frac{\sum_{j,g}\sum_{m=0}^{M} s_{j,g,m} \ln s_{j,g,m}}{\ln(M+1)\sum_j |\mathcal{G}_j|}$, respectively. $\ln(M+1)$ is the maximum entropy over the shared and specialized modules. Results show that CORTEX achieves high average maximum weights and low average normalized entropy across all three backbones, which indicates concentrated soft module assignments.

Table 4: NLL, $I^{\mathrm{mod}}$, and lesion effects. **Target** and **Other** denote the average NLL changes after module removal on the paired domain and other domains, respectively. Results are averaged across seeds.

| Backbone | Method | NLL ↓ | $I^{\mathrm{mod}}$ | $\Delta^{\mathrm{les}}_{m,k}$ | |
|---|---|---|---|---|---|
| | | | | Target | Other |
| 160M | Random | 0.4082 | 1.3860 | 0.0097 | 0.0086 |
| | **CORTEX** | 0.0734 | 1.4560 | 0.1901 | 0.0064 |
| Qwen3-8B | Random | 1.9179 | 1.0238 | 0.0374 | 0.0108 |
| | **CORTEX** | 1.7303 | 1.1534 | 0.1557 | 0.0106 |
| Qwen3-32B | Random | 1.6571 | 1.1996 | 0.0298 | 0.0097 |
| | **CORTEX** | 1.4786 | 1.3331 | 0.1657 | 0.0059 |

Table 5: Assignment concentration of CORTEX, averaged across seeds.

| Backbone | Average max. weight | Average normalized entropy |
|---|---|---|
| 160M | 0.8755 | 0.2783 |
| Qwen3-8B | 0.8429 | 0.3542 |
| Qwen3-32B | 0.9001 | 0.2581 |

## 6 Conclusion

In this work, we proposed CORTEX to form shared and specialized modules within dense language models. CORTEX keeps the original dense parameter space and divides trainable matrices into parameter groups. It learns module assignment weights from domain-conditioned gradients while preserving the dense architecture. We further introduced two module-level interpretability metrics to evaluate whether the modules are functionally specialized and aligned with knowledge domains. Our theoretical analysis relates module assignment weights to the convergence and connects the proposed metrics to modularization. Results on the synthetic and real knowledge-domain mixtures demonstrate that CORTEX reduces NLL and PPL and learns domain-specific module assignments while preserving the dense architecture.

### AI USE STATEMENT

We used LLMs to assist with language polishing, grammar correction, and improving the clarity of the manuscript. For the synthetic benchmark, LLMs were used only to draft candidate task templates. The final task definitions, labels, data splits, and all synthetic instances were produced by the deterministic rule-based generator described in Appendix D.2.

### ETHICS STATEMENT

This work provides a framework for forming and evaluating internal modularization in dense language models. On the positive side, CORTEX can improve transparency by making shared and domain-specific computation more distinctive, and may support more targeted adaptation and diagnosis across different knowledge domains. At the same time, stronger specialization can also improve knowledge-specific capabilities in areas such as code, biomedical text, legal text, or security-related content. If knowledge-domain labels or training data contain biases, the learned modules may also reflect or reinforce these biases. Therefore, applications of CORTEX to high-risk data or models should include careful data governance, privacy protection, bias evaluation, and appropriate safeguards.

### REPRODUCIBILITY STATEMENT

The algorithm of CORTEX is presented in Appendix B. The assumptions and proofs are provided in Appendix C. Hyperparameter settings, dataset preparation, domain-conditioned gradient calculation, and downstream evaluation protocols are described in Appendix D.

## REFERENCES

Elizabeth Bates, Stephen M. Wilson, Ayse P. Saygin, Frederic Dick, Martin I. Sereno, Robert T. Knight, and Nina F. Dronkers. Voxel-based lesion-symptom mapping. *Nature Neuroscience*, 6(5):448–450, 2003. doi: https://doi.org/10.1038/nn1050.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Mayee F. Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Ré. Aioli: A unified optimization framework for language model data mixing. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=sZGZJhaNSe.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning*, pages 794–803, 2018. URL: https://proceedings.mlr.press/v80/chen18a.html.

Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. In *Advances in Neural Information Processing Systems*, volume 33, pages 2039–2050, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/16002f7a455a94aa4e91cc34ebdb9f2d-Abstract.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL: https://arxiv.org/abs/2110.14168.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *Proceedings of the International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=F76bwRSLeK.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 8493–8502, 2022. URL: https://aclanthology.org/2022.acl-long.581/.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 1280–1297, 2024. URL: https://aclanthology.org/2024.acl-long.70/.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, 2023. URL: https://arxiv.org/abs/2305.14314.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1286–1305, 2021. URL: https://aclanthology.org/2021.emnlp-main.98/.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, pages 5547–5569, 2022. URL: https://proceedings.mlr.press/v162/du22c.html.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL: https://transformer-circuits.pub/2021/framework/index.html.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL: https://jmlr.org/papers/v23/21-0998.html.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=rJl-b3RcF7.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. URL: https://arxiv.org/abs/2101.00027.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=tcsZt9ZNKD.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer 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. URL: https://aclanthology.org/2021.emnlp-main.446/.

Zhuocheng Gong, Ang Lv, Jian Guan, Wei Wu, Huishuai Zhang, Minlie Huang, Dongyan Zhao, and Rui Yan. Mixture-of-modules: Reinventing transformers as dynamic assemblies of modules. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20924–20938, 2024. URL: https://aclanthology.org/2024.emnlp-main.1164/.

Neel Guha, Julian Nyarko, Daniel E. Ho, Christopher Ré, Adam Chilton, Aditya Narayana, Alex Chohlas-Wood, Austin Peters, Brandon Waldon, Daniel N. Rockmore, Diego Zambrano, Dmitry Talisman, Enam Hoque, Faiz Surani, Frank Fagan, Galit Sarfaty, Gregory M. Dickinson, Haggai Porat, Jason Hegland, Jessica Wu, Joe Nudell, Joel Niklaus, John Nay, Jonathan H. Choi, Kevin Tobia, Margaret Hagan, Megan Ma, Michael Livermore, Nikon Rasumov-Rahe, Nils Holzenberger, Noam Kolt, Peter Henderson, Sean Rehaag, Sharad Goel, Shang Gao, Spencer Williams, Sunny Gandhi, Tom Zur, Varun Iyer, and Zehua Li. LegalBench: A collaboratively built benchmark for measuring legal reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/89e44582fd28ddfea1ea4dcb0ebbf4b0-Abstract-Datasets_and_Benchmarks.html.

Harvard Law School Library Innovation Lab. Caselaw Access Project. 2024. URL: https://case.law/.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2790–2799, 2019. URL: https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations*, 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

Qiushi Huang, Tom Ko, Zhan Zhuang, Lilian Tang, and Yu Zhang. HiRA: Parameter-efficient Hadamard high-rank adaptation for large language models. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=TwJrTz9cRS.

Tingfeng Hui, Zhenyu Zhang, Shuohuan Wang, Yu Sun, Hua Wu, and Sen Su. Upcycling instruction tuning from dense to mixture-of-experts via parameter merging. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 13012–13031, 2025. URL: https://aclanthology.org/2025.acl-long.637/.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. URL: https://direct.mit.edu/neco/article/3/1/79/5560/Adaptive-Mixtures-of-Local-Experts.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=chfJJYC3iL.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. PubMedQA: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 2567–2577, 2019. URL: https://aclanthology.org/D19-1259/.

Junmo Kang, Leonid Karlinsky, Hongyin Luo, Zhen Wang, Jacob A. Hansen, James R. Glass, David D. Cox, Rameswar Panda, Rogerio Feris, and Alan Ritter. Self-MoE: Towards compositional large language models with self-specialized experts. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=IDJUscOjM3.

Houyi Li, Ka Man Lo, Shijie Xuyang, Ziqi Wang, Wenzhen Zheng, Haocheng Zhang, Zhao Li, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Mixture-of-experts can surpass dense LLMs under strictly equal resource. In *Proceedings of the International Conference on Learning Representations*, 2026. URL: https://openreview.net/forum?id=oIdzliJAeA.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 18878–18890, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/9d27fdf2477ffbff837d73ef7ae23db9-Abstract.html.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. RegMix: Data mixture as regression for language model pre-training. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=5BjQOUXq7i.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32100–32121, 2024. URL: https://proceedings.mlr.press/v235/liu24bn.html.

Ka Man Lo, Zeyu Huang, Zihan Qiu, Zili Wang, and Jie Fu. A closer look into mixture-of-experts in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 4427–4447, 2025. URL: https://aclanthology.org/2025.findings-naacl.251/.

Arun Mallya and Svetlana Lazebnik. PackNet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018. URL: https://openaccess.thecvf.com/content_cvpr_2018/html/Mallya_PackNet_Adding_Multiple_CVPR_2018_paper.html.

Philipp Mondorf, Sondre Wold, and Barbara Plank. Circuit compositions: Exploring modular structures in transformer-based language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 14934–14955, 2025. URL: https://aclanthology.org/2025.acl-long.727/.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open mixture-of-experts language models. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=xXTkbTBmqq.

Tuan Ngo, Abid Hassan, Saad Shafiq, and Nenad Medvidović. DNN modularization via activation-driven training. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, 2026. URL: https://doi.org/10.1145/3744916.3773190.

Binhang Qi, Hailong Sun, Hongyu Zhang, Ruobing Zhao, and Xiang Gao. Modularizing while training: A new paradigm for modularizing DNN models. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering*, 2024. URL: https://arxiv.org/abs/2306.09376.

Zihan Qiu, Zeyu Huang, and Jie Fu. Unlocking emergent modularity in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2638–2660, 2024. URL: https://aclanthology.org/2024.naacl-long.144/.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL: https://jmlr.org/papers/v21/20-074.html.

Chris Rorden and Hans-Otto Karnath. Using human brain lesions to infer function: A relic from a past era in the fMRI age? *Nature Reviews Neuroscience*, 5(10):813–819, 2004. doi: https://doi.org/10.1038/nrn1521.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. URL: https://arxiv.org/abs/1606.04671.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, number 5, pages 8732–8740, 2020. doi: https://doi.org/10.1609/aaai.v34i05.6399.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL: https://papers.nips.cc/paper/2018/hash/432aca3a1e345e339f35a30c8f65edce-Abstract.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=B1ckMDqlg.

Guangyuan Shi, Qimai Li, Wenlong Zhang, Jiaxin Chen, and Xiao-Ming Wu. Recon: Reducing conflicting gradients from the root for multi-task learning. In *Proceedings of the International Conference on Learning Representations*, 2023. URL: https://arxiv.org/abs/2302.11289.

Weijia Shi, Akshita Bhagia, Kevin Farhat, et al. FlexOLMo: Open Language Models for Flexible Data Use. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL: https://proceedings.neurips.cc/paper_files/paper/2025/hash/f2621bfa8b39cd3c5a361273aa7ec4db-Abstract-Conference.html.

Linus Stuhlmann, Michael Alexander Saxer, and Jonathan Fürst. Efficient and reproducible biomedical question answering using retrieval augmented generation. In *Proceedings of the 2025 IEEE Swiss Conference on Data Science*, pages 154–157, 2025. doi: https://doi.org/10.1109/SDS66131.2025.00029.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, 2023. URL: https://aclanthology.org/2023.findings-acl.824/.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, 2019. URL: https://aclanthology.org/P19-1580/.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=y10DM6R2r3.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-Pro: Are low-rank adapters properly optimized? In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=gTwRMU3lJ5.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-Instruct. In *Proceedings of the 41st International Conference on Machine Learning*, pages 52632–52657, 2024. URL: https://proceedings.mlr.press/v235/wei24h.html.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems*, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/dcba6be91359358c2355cd920da3fcbd-Abstract-Conference.html.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL: https://arxiv.org/abs/2505.09388.

Yunzhi Yao, Ningyu Zhang, Zekun Xi, Mengru Wang, Ziwen Xu, Shumin Deng, and Huajun Chen. Knowledge circuits in pretrained transformers. In *Advances in Neural Information Processing Systems*, 2024. URL: https://papers.nips.cc/paper_files/paper/2024/hash/d6df31b1be98e04be48af8bedb95b499-Abstract-Conference.html.

Jiasheng Ye, Peiju Liu, Tianxiang Sun, Jun Zhan, Yunhua Zhou, and Xipeng Qiu. Data mixing laws: Optimizing data mixtures by predicting language modeling performance. In *Proceedings of the International Conference on Learning Representations*, 2025. URL: https://openreview.net/forum?id=jjCB27TMK3.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 5824–5836, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/3fe78a8acf5fda99de95303940a2420c-Abstract.html.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *Proceedings of the International Conference on Learning Representations*, 2024. URL: https://openreview.net/forum?id=yLClGs770I.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, 2022. URL: https://aclanthology.org/2022.findings-acl.71/.

Zhengyan Zhang, Zhiyuan Zeng, Yankai Lin, Chaojun Xiao, Xiaozhi Wang, Xu Han, Zhiyuan Liu, Ruobing Xie, Maosong Sun, and Jie Zhou. Emergent modularity in pre-trained transformers. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4066–4083, 2023. URL: https://aclanthology.org/2023.findings-acl.250/.

# A List of Key Notations

Table 6 summarizes the key notations used throughout the paper.

Table 6: List of key notations.

| Notation | Meaning |
|---|---|
| $C$ | Knowledge-domain label. |
| $d_j^{\text{in}}, d_j^{\text{out}}$ | Input and output dimensions of the $j$-th trainable weight matrix. |
| $\mathcal{G}_j, G_j$ | Index set and the number of parameter groups in $\mathbf{W}_j$. |
| $\mathbf{G}_{j,g,k}$ | Domain-conditioned gradient of the $g$-th parameter group in the $j$-th trainable parameter matrix w.r.t. the $k$-th knowledge domain. |
| $\mathcal{J}, J$ | Set and the number of trainable weight matrices in the backbone model. |
| $\mathcal{K}, K$ | Set and the number of knowledge domains. |
| $\mathcal{L}_{\text{main}}, \mathcal{L}_{\text{assign}}$ | Main task loss and module assignment loss. |
| $\mathcal{P}, \mathcal{P}_k$ | Overall training and conditional distribution of $(X, Y)$ given $C = k$. |
| $\mathbf{s}_{j,g}, s_{j,g,m}$ | Assignment weight vector and assignment weight of the $g$-th parameter group in $\mathbf{W}_j$ assigned to the $m$-th module. |
| $u_{j,g,0}, u_{j,g,m}$ | Utility scores of the $g$-th parameter group in $\mathbf{W}_j$ for the shared module and the $m$-th specialized module. |
| $\mathbf{W}, \mathbf{W}_j, \mathbf{W}_{j,m}$ | Trainable model parameters, the $j$-th trainable weight matrix, and its module-specific component associated with module $m$. |
| $X, Y, \mathcal{X}, \mathcal{Y}$ | Input and output variables, together with their corresponding spaces. |
| $\eta$ | Learning rate. |
| $\boldsymbol{\xi}, \boldsymbol{\xi}_k$ | Mini-batch of training samples and its subset from the $k$-th knowledge domain. |
| $\pi_k$ | Probability that a randomly drawn sample belongs to the $k$-th domain. |
| $\Pi_{j,g}$ | Projection operator that extracts the $g$-th parameter group from $\mathbf{W}_j$. |
| $\boldsymbol{\psi}_{j,g}, \psi_{j,g,m}, \Psi$ | Assignment parameter vector, its $m$-th entry, and the set of all trainable assignment parameter vectors. |

# B Algorithm of CORTEX

**Algorithm 1** CORTEX

**Input:** Training mixture $\mathcal{P} = \sum_{k\in\mathcal{K}} \pi_k \mathcal{P}_k$; trainable matrices $\mathcal{W} = \{\mathbf{W}_j\}_{j\in\mathcal{J}}$; parameter-group projections $\{\Pi_{j,g}\}_{j,g}$; assignment parameters $\Psi = \{\boldsymbol{\psi}_{j,g}\}_{j,g}$; learning rate.

Initialize $\mathcal{W}^1$ from the dense backbone and initialize $\Psi^1$.

Calculate module assignment weights: $s^1_{j,g,m} = \frac{\exp\left(\psi^1_{j,g,m}\right)}{\sum_{r\in\{0\}\cup\mathcal{M}} \exp\left(\psi^1_{j,g,r}\right)}, \quad j \in \mathcal{J},\ g \in \mathcal{G}_j, m \in \{0\}\cup\mathcal{M}.$

**for** $t = 1, \ldots, T$ **do**

Sample mini-batches covering all knowledge domains.

DOMAIN-CONDITIONED GROUP GRADIENTS

**for** each $k \in \mathcal{K}$ **do**

Calculate the domain-conditioned group gradients: $\mathbf{G}^t_{j,g,k} = \Pi_{j,g}\left(\nabla_{\mathbf{W}_j} F(\mathcal{W}^t; \boldsymbol{\xi}_k)\right), \quad j \in \mathcal{J},\ g \in \mathcal{G}_j.$

**end for**

UTILITY SCORES

Calculate the utility scores:

$$u^t_{j,g,0} = \frac{2}{K(K-1)} \sum_{1\le p<q\le K} \frac{\langle \mathbf{G}^t_{j,g,p}, \mathbf{G}^t_{j,g,q}\rangle_F}{\|\mathbf{G}^t_{j,g,p}\|_F \|\mathbf{G}^t_{j,g,q}\|_F}, \quad j \in \mathcal{J},\ g \in \mathcal{G}_j,$$

$$u^t_{j,g,m} = \frac{\|\mathbf{G}^t_{j,g,\kappa(m)}\|_F}{\sum_{k'\in\mathcal{K}} \|\mathbf{G}^t_{j,g,k'}\|_F} - \frac{1}{K-1} \sum_{q\in\mathcal{K}\setminus\{\kappa(m)\}} \frac{\langle \mathbf{G}^t_{j,g,\kappa(m)}, \mathbf{G}^t_{j,g,q}\rangle_F}{\|\mathbf{G}^t_{j,g,\kappa(m)}\|_F \|\mathbf{G}^t_{j,g,q}\|_F},$$
$$j \in \mathcal{J},\ g \in \mathcal{G}_j, m \in \mathcal{M}.$$

MODULE ASSIGNMENT UPDATE

Form the assignment loss: $\mathcal{L}^t_{\text{assign}} \leftarrow -\frac{1}{\sum_{j\in\mathcal{J}} G_j} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\{0\}\cup\mathcal{M}} s^t_{j,g,m} u^t_{j,g,m}.$

Update $\Psi^{t+1}$ and update $s^{t+1}_{j,g,m}$ from $\Psi^{t+1}$.

MODULARIZED DENSE-MODEL UPDATE

**for** each $j \in \mathcal{J}$ **do**

Update $\mathbf{W}^{t+1}_j = \mathbf{W}^t_j - \eta \sum_{g\in\mathcal{G}_j} \left(s^t_{j,g,0} \sum_{k\in\mathcal{K}} \pi_k \mathbf{G}^t_{j,g,k} + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s^t_{j,g,m} \mathbf{G}^t_{j,g,\kappa(m)}\right).$

**end for**

**end for**

**Output:** Trained model $\mathcal{W}^{\text{out}}$.

# C PROOFS

## C.1 PROOF OF PROPOSITION 1

For the $k$-th knowledge domain, we consider the change of the loss $F(\mathcal{W};\boldsymbol{\xi}_k)$ caused by updating the $g$-th parameter group in the $j$-th trainable matrix. The corresponding update is

$$\mathbf{W}_j' - \mathbf{W}_j = -\eta \sum_{q\in\mathcal{K}} \pi_q \mathbf{G}_{j,g,q}, \tag{1}$$

where $\mathcal{W}'$ denotes the model after applying this update, with all other matrices unchanged. By applying the first-order Taylor expansion at $\mathcal{W}$ and substituting this update, we have

$$\begin{aligned} F(\mathcal{W}';\boldsymbol{\xi}_k) - F(\mathcal{W};\boldsymbol{\xi}_k) &= \left\langle \nabla_{\mathbf{W}_j} F(\mathcal{W};\boldsymbol{\xi}_k), \mathbf{W}_j' - \mathbf{W}_j \right\rangle_F + \mathcal{O}\big(\|\mathbf{W}_j' - \mathbf{W}_j\|_F^2\big) \\ &= -\eta \left\langle \nabla_{\mathbf{W}_j} F(\mathcal{W};\boldsymbol{\xi}_k), \sum_{q\in\mathcal{K}} \pi_q \mathbf{G}_{j,g,q} \right\rangle_F + \mathcal{O}(\eta^2), \end{aligned} \tag{2}$$

where $\mathcal{O}(\eta^2)$ denotes a remainder whose absolute value is bounded by a constant times $\eta^2$ as $\eta \to 0$, with the current model and mini-batch fixed. Since $\mathbf{G}_{j,g,k} = \Pi_{j,g}\big(\nabla_{\mathbf{W}_j} F(\mathcal{W};\boldsymbol{\xi}_k)\big)$, we have

$$\left\langle \nabla_{\mathbf{W}_j} F(\mathcal{W};\boldsymbol{\xi}_k), \mathbf{G}_{j,g,q} \right\rangle_F = \left\langle \mathbf{G}_{j,g,k}, \mathbf{G}_{j,g,q} \right\rangle_F. \tag{3}$$

Therefore,

$$\begin{aligned} F(\mathcal{W}';\boldsymbol{\xi}_k) - F(\mathcal{W};\boldsymbol{\xi}_k) &= -\eta \sum_{q\in\mathcal{K}} \pi_q \left\langle \mathbf{G}_{j,g,k}, \mathbf{G}_{j,g,q} \right\rangle_F + \mathcal{O}(\eta^2) \\ &= -\eta \left( \pi_k \|\mathbf{G}_{j,g,k}\|_F^2 + \sum_{q\in\mathcal{K}\setminus\{k\}} \pi_q \left\langle \mathbf{G}_{j,g,k}, \mathbf{G}_{j,g,q} \right\rangle_F \right) + \mathcal{O}(\eta^2). \end{aligned} \tag{4}$$

This completes the proof of Proposition 1.

## C.2 PROOF OF PROPOSITION 2

Since the mixture of knowledge domains satisfies $\mathcal{P} = \sum_{k\in\mathcal{K}} \pi_k \mathcal{P}_k$, the main task loss can be decomposed as

$$\mathcal{L}_{\text{main}}(\mathcal{W}) = \mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{P}}\left[F(\mathcal{W};\boldsymbol{\xi})\right] = \sum_{k\in\mathcal{K}} \pi_k R_k(\mathcal{W}). \tag{5}$$

By applying eqn. (5) to both $\mathcal{W}^{\text{out}}$ and $\mathcal{W}^{\text{drop},m}$, we have

$$\begin{aligned} \mathcal{L}_{\text{main}}(\mathcal{W}^{\text{drop},m}) - \mathcal{L}_{\text{main}}(\mathcal{W}^{\text{out}}) &= \sum_{k\in\mathcal{K}} \pi_k \left( R_k(\mathcal{W}^{\text{drop},m}) - R_k(\mathcal{W}^{\text{out}}) \right) \\ &\overset{\text{(a)}}{=} \sum_{k\in\mathcal{K}} \pi_k \Delta_{m,k}^{\text{les}}, \end{aligned} \tag{6}$$

where equality (a) results from Definition 3. Based on Definition 3, we have

$$\Delta_{m,k}^{\text{les}} = \text{SLS}(m,k) + \frac{1}{K-1} \sum_{q\in\mathcal{K}\setminus\{k\}} \Delta_{m,q}^{\text{les}}. \tag{7}$$

By combining eqns. (6) and (7), we have

$$\begin{aligned} &\mathcal{L}_{\text{main}}(\mathcal{W}^{\text{drop},m}) - \mathcal{L}_{\text{main}}(\mathcal{W}^{\text{out}}) \\ &= \pi_k \Delta_{m,k}^{\text{les}} + \sum_{q\in\mathcal{K}\setminus\{k\}} \pi_q \Delta_{m,q}^{\text{les}} \end{aligned}$$

$$= \pi_k \left( \mathrm{SLS}(m,k) + \frac{1}{K-1} \sum_{q \in \mathcal{K} \setminus \{k\}} \Delta^{\mathrm{les}}_{m,q} \right) + \sum_{q \in \mathcal{K} \setminus \{k\}} \pi_q \Delta^{\mathrm{les}}_{m,q}$$
$$= \pi_k \mathrm{SLS}(m,k) + \sum_{q \in \mathcal{K} \setminus \{k\}} \left( \pi_q + \frac{\pi_k}{K-1} \right) \Delta^{\mathrm{les}}_{m,q}. \tag{8}$$

Therefore, $\mathcal{L}_{\mathrm{main}}(\mathcal{W}^{\mathrm{drop},m}) > \mathcal{L}_{\mathrm{main}}(\mathcal{W}^{\mathrm{out}})$ holds if

$$\pi_k \mathrm{SLS}(m,k) + \sum_{q \in \mathcal{K} \setminus \{k\}} \left( \pi_q + \frac{\pi_k}{K-1} \right) \Delta^{\mathrm{les}}_{m,q} > 0. \tag{9}$$

Since $\pi_k > 0$, inequality (9) is equivalent to

$$\mathrm{SLS}(m,k) > -\frac{1}{\pi_k} \sum_{q \in \mathcal{K} \setminus \{k\}} \left( \pi_q + \frac{\pi_k}{K-1} \right) \Delta^{\mathrm{les}}_{m,q}. \tag{10}$$

This completes the proof of Proposition 2.

### C.3 Proof of Proposition 3

Based on Definition 4, $I^{\mathrm{mod}}$ satisfies

$$\begin{aligned}
I^{\mathrm{mod}} &= \sum_{m \in \{0\} \cup \mathcal{M}} \sum_{k \in \mathcal{K}} \mathrm{GS}_{m,k} \ln \frac{\frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m}}{\mathrm{GS}^{\mathrm{domain}}_k} \\
&= \sum_{m \in \{0\} \cup \mathcal{M}} \sum_{k \in \mathcal{K}} \mathrm{GS}_{m,k} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} - \sum_{m \in \{0\} \cup \mathcal{M}} \sum_{k \in \mathcal{K}} \mathrm{GS}_{m,k} \ln \mathrm{GS}^{\mathrm{domain}}_k \\
&= \sum_{m \in \{0\} \cup \mathcal{M}} \sum_{k \in \mathcal{K}} \mathrm{GS}_{m,k} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} - \sum_{k \in \mathcal{K}} \left( \sum_{m \in \{0\} \cup \mathcal{M}} \mathrm{GS}_{m,k} \right) \ln \mathrm{GS}^{\mathrm{domain}}_k \\
&\overset{(a)}{=} \sum_{m \in \{0\} \cup \mathcal{M}} \sum_{k \in \mathcal{K}} \mathrm{GS}_{m,k} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} - \sum_{k \in \mathcal{K}} \mathrm{GS}^{\mathrm{domain}}_k \ln \mathrm{GS}^{\mathrm{domain}}_k \\
&= \sum_{m \in \{0\} \cup \mathcal{M}} \mathrm{GS}^{\mathrm{module}}_m \sum_{k \in \mathcal{K}} \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} - \sum_{k \in \mathcal{K}} \mathrm{GS}^{\mathrm{domain}}_k \ln \mathrm{GS}^{\mathrm{domain}}_k \\
&= - \sum_{m \in \{0\} \cup \mathcal{M}} \mathrm{GS}^{\mathrm{module}}_m \left( - \sum_{k \in \mathcal{K}} \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \right) - \sum_{k \in \mathcal{K}} \mathrm{GS}^{\mathrm{domain}}_k \ln \mathrm{GS}^{\mathrm{domain}}_k,
\end{aligned} \tag{11}$$

where equality (a) follows from Definition 4. Based on Definition 4, $\frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \in [0,1]$. Hence, we have

$$- \sum_{k \in \mathcal{K}} \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \geq 0. \tag{12}$$

Therefore, $I^{\mathrm{mod}}$ satisfies

$$I^{\mathrm{mod}} \leq - \sum_{k \in \mathcal{K}} \mathrm{GS}^{\mathrm{domain}}_k \ln \mathrm{GS}^{\mathrm{domain}}_k \overset{(a)}{\leq} \ln K, \tag{13}$$

where inequality (a) holds due to the fact that $\mathrm{GS}^{\mathrm{domain}}_k$ is a distribution over $K$ knowledge domains. The equality $I^{\mathrm{mod}} = -\sum_{k \in \mathcal{K}} \mathrm{GS}^{\mathrm{domain}}_k \ln \mathrm{GS}^{\mathrm{domain}}_k$ holds if $-\sum_{k \in \mathcal{K}} \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} \ln \frac{\mathrm{GS}_{m,k}}{\mathrm{GS}^{\mathrm{module}}_m} = 0$. It means each module obtains the gradient from at most one knowledge domain. This completes the proof of Proposition 3.

### C.4 Proof of Theorem 1

Expectations without conditioning are taken over all mini-batches sampled during training. Conditional expectations are taken given the model parameters $\mathbf{W}^t$ and assignment parameters $\Psi^t$ before sampling. We denote the update direction in the $t$-th training round as

$$\mathbf{\Phi}_j^t = \sum_{g\in\mathcal{G}_j}\left(s_{j,g,0}^t \sum_{k\in\mathcal{K}} \pi_k \mathbf{G}_{j,g,k}^t + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s_{j,g,m}^t \mathbf{G}_{j,g,\kappa(m)}^t\right), \quad j\in\mathcal{J}. \tag{14}$$

First, we introduce two assumptions which are widely used to facilitate our proof.

**Assumption 1** ($L$-Smoothness) *$\mathcal{L}_{\text{main}}(\mathbf{W})$ is lower-bounded by $\mathcal{L}_{\text{main}}(\mathbf{W}^\star)$ and $L$-smooth. That is, for two arbitrary matrices $\mathbf{W}$ and $\mathbf{W}'$, we have*

$$\mathcal{L}_{\text{main}}(\mathbf{W}') \le \mathcal{L}_{\text{main}}(\mathbf{W}) + \langle \nabla\mathcal{L}_{\text{main}}(\mathbf{W}), \mathbf{W}'-\mathbf{W}\rangle_F + \frac{L}{2}\|\mathbf{W}'-\mathbf{W}\|_F^2. \tag{15}$$

**Assumption 2** (Bounded gradient and variance) *For each knowledge domain $k$, the collected group gradient is conditionally unbiased: $\mathbb{E}[\mathbf{G}_{j,g,k}^t \mid \mathbf{W}^t, \Psi^t] = \bar{\mathbf{G}}_{j,g,k}^t = \Pi_{j,g}\left(\nabla_{\mathbf{W}_j} R_k(\mathbf{W}^t)\right)$. For all $t, j, g, k$, its conditional mean and variance satisfy*

$$\left\|\bar{\mathbf{G}}_{j,g,k}^t\right\|_F^2 \le G^2, \tag{16}$$

$$\mathbb{E}\left[\left\|\mathbf{G}_{j,g,k}^t - \bar{\mathbf{G}}_{j,g,k}^t\right\|_F^2 \,\middle|\, \mathbf{W}^t, \Psi^t\right] \le \sigma^2. \tag{17}$$

Then, we present the proof of Theorem 1 as follows.

**Proof of Theorem 1** Based on the update rule and (14), we have

$$\mathbf{W}_j^{t+1} = \mathbf{W}_j^t - \eta\mathbf{\Phi}_j^t, \quad j\in\mathcal{J}. \tag{18}$$

Based on Assumption 1, we have

$$\begin{aligned}\mathcal{L}_{\text{main}}(\mathbf{W}^{t+1}) &\le \mathcal{L}_{\text{main}}(\mathbf{W}^t) + \left\langle \nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t), \mathbf{W}^{t+1}-\mathbf{W}^t\right\rangle_F + \frac{L}{2}\left\|\mathbf{W}^{t+1}-\mathbf{W}^t\right\|_F^2 \\ &= \mathcal{L}_{\text{main}}(\mathbf{W}^t) - \eta\underbrace{\sum_{j\in\mathcal{J}}\left\langle \nabla_{\mathbf{W}_j}\mathcal{L}_{\text{main}}(\mathbf{W}^t), \mathbf{\Phi}_j^t\right\rangle_F}_{T_1} + \frac{L\eta^2}{2}\underbrace{\sum_{j\in\mathcal{J}}\left\|\mathbf{\Phi}_j^t\right\|_F^2}_{T_2}.\end{aligned} \tag{19}$$

In particular, $T_2$ satisfies

$$\begin{aligned}T_2 &= \sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\left\|s_{j,g,0}^t\sum_{k\in\mathcal{K}}\pi_k\mathbf{G}_{j,g,k}^t + \sum_{m\in\mathcal{M}}\pi_{\kappa(m)}s_{j,g,m}^t\mathbf{G}_{j,g,\kappa(m)}^t\right\|_F^2 \\ &\overset{\text{(a)}}{=} \sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\left\|\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s_{j,g,0}^t + s_{j,g,m}^t\right)\mathbf{G}_{j,g,\kappa(m)}^t\right\|_F^2,\end{aligned} \tag{20}$$

where equality (a) follows from the one-to-one pairing between specialized modules and knowledge domains. The weights $\pi_{\kappa(m)}(s_{j,g,0}^t + s_{j,g,m}^t)$ are nonnegative. We apply the Cauchy-Schwarz inequality to $\sqrt{\pi_{\kappa(m)}(s_{j,g,0}^t + s_{j,g,m}^t)}$ and $\sqrt{\pi_{\kappa(m)}(s_{j,g,0}^t + s_{j,g,m}^t)}\mathbf{G}_{j,g,\kappa(m)}^t$ for each matrix entry, and then sum over all entries to obtain

$$\begin{aligned}&\left\|\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s_{j,g,0}^t + s_{j,g,m}^t\right)\mathbf{G}_{j,g,\kappa(m)}^t\right\|_F^2 \\ &\le \left(\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s_{j,g,0}^t + s_{j,g,m}^t\right)\right)\left(\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s_{j,g,0}^t + s_{j,g,m}^t\right)\left\|\mathbf{G}_{j,g,\kappa(m)}^t\right\|_F^2\right).\end{aligned} \tag{21}$$

Taking total expectations on both sides of inequality (21), we have

$$\begin{aligned}
&\mathbb{E}\left[\left\|\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s^t_{j,g,0}+s^t_{j,g,m}\right)\mathbf{G}^t_{j,g,\kappa(m)}\right\|_F^2\right]\\
&\overset{(a)}{\leq}\mathbb{E}\Bigg[\left(\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s^t_{j,g,0}+s^t_{j,g,m}\right)\right)\\
&\qquad\times\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s^t_{j,g,0}+s^t_{j,g,m}\right)\mathbb{E}\left[\left\|\mathbf{G}^t_{j,g,\kappa(m)}\right\|_F^2\,\middle|\,\mathbf{W}^t,\Psi^t\right]\Bigg]\\
&\overset{(b)}{\leq}2(G^2+\sigma^2)\mathbb{E}\left[\left(\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s^t_{j,g,0}+s^t_{j,g,m}\right)\right)^2\right]\\
&=2(G^2+\sigma^2)\mathbb{E}\left[\left(s^t_{j,g,0}+\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}s^t_{j,g,m}\right)^2\right].
\end{aligned}\tag{22}$$

Inequality (a) follows from (21) by taking expectations over the conditional expectation, since the assignment weights are fixed given $\mathbf{W}^t$ and $\Psi^t$. Inequality (b) follows from Assumption 2 and $\|\mathbf{A}+\mathbf{B}\|_F^2\leq 2\|\mathbf{A}\|_F^2+2\|\mathbf{B}\|_F^2$. By combining eqn. (20) and inequality (22), we have

$$\mathbb{E}[T_2]\leq 2(G^2+\sigma^2)\sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\mathbb{E}\left[\left(s^t_{j,g,0}+\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}s^t_{j,g,m}\right)^2\right].\tag{23}$$

Then, we bound $\mathbb{E}[T_1]$. We first condition on $\mathbf{W}^t$ and $\Psi^t$. Note that $\nabla_{\mathbf{W}_j}\mathcal{L}_{\text{main}}(\mathbf{W}^t)=\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\bar{\mathbf{G}}^t_{j,g,\kappa(m)}$. And we have

$$\begin{aligned}
\mathbb{E}\left[\mathbf{\Phi}^t_j\,\middle|\,\mathbf{W}^t,\Psi^t\right]&=\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(s^t_{j,g,0}+s^t_{j,g,m}\right)\bar{\mathbf{G}}^t_{j,g,\kappa(m)}\\
&=\nabla_{\mathbf{W}_j}\mathcal{L}_{\text{main}}(\mathbf{W}^t)-\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(1-s^t_{j,g,0}-s^t_{j,g,m}\right)\bar{\mathbf{G}}^t_{j,g,\kappa(m)}.
\end{aligned}\tag{24}$$

The squared norm of the full gradient satisfies $\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\|_F^2=\sum_{j\in\mathcal{J}}\|\nabla_{\mathbf{W}_j}\mathcal{L}_{\text{main}}(\mathbf{W}^t)\|_F^2$. Hence, $\mathbb{E}[T_1\mid\mathbf{W}^t,\Psi^t]$ satisfies

$$\begin{aligned}
&\mathbb{E}[T_1\mid\mathbf{W}^t,\Psi^t]\\
&=-\sum_{j\in\mathcal{J}}\left\langle\nabla_{\mathbf{W}_j}\mathcal{L}_{\text{main}}(\mathbf{W}^t),\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(1-s^t_{j,g,0}-s^t_{j,g,m}\right)\bar{\mathbf{G}}^t_{j,g,\kappa(m)}\right\rangle_F\\
&\quad+\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2\\
&\overset{(a)}{\geq}\frac{1}{2}\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2-\underbrace{\frac{1}{2}\sum_{j\in\mathcal{J}}\left\|\sum_{g\in\mathcal{G}_j}\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(1-s^t_{j,g,0}-s^t_{j,g,m}\right)\bar{\mathbf{G}}^t_{j,g,\kappa(m)}\right\|_F^2}_{T_3},
\end{aligned}\tag{25}$$

where inequality (a) follows from $\langle\mathbf{A},\mathbf{B}\rangle_F\leq\frac{1}{2}\|\mathbf{A}\|_F^2+\frac{1}{2}\|\mathbf{B}\|_F^2$. Then, we bound $T_3$ as follows:

$$\begin{aligned}
T_3&\overset{(a)}{=}\sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\left\|\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(1-s^t_{j,g,0}-s^t_{j,g,m}\right)\bar{\mathbf{G}}^t_{j,g,\kappa(m)}\right\|_F^2\\
&\overset{(b)}{\leq}\sum_{j\in\mathcal{J}}\sum_{g\in\mathcal{G}_j}\left(\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left(1-s^t_{j,g,0}-s^t_{j,g,m}\right)^2\right)\sum_{m\in\mathcal{M}}\pi_{\kappa(m)}\left\|\bar{\mathbf{G}}^t_{j,g,\kappa(m)}\right\|_F^2
\end{aligned}$$

$$\overset{(c)}{\leq} G^2 \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} \left(1 - s^t_{j,g,0} - s^t_{j,g,m}\right)^2, \tag{26}$$

where equality (a) is obtained due to the fact that different parameter groups are disjoint as described in Definition 1. Inequality (b) results from Cauchy-Schwarz inequality. Inequality (c) follows from Assumption 2 and the fact that $\sum_{m\in\mathcal{M}} \pi_{\kappa(m)} = 1$. By combining inequalities (25) and (26) and taking total expectations, we have

$$\mathbb{E}[T_1] \geq \frac{1}{2}\mathbb{E}\left[\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2\right] - \frac{G^2}{2} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} \mathbb{E}\left[\left(1 - s^t_{j,g,0} - s^t_{j,g,m}\right)^2\right]. \tag{27}$$

Taking total expectations in (19) and applying (23) and (27), we obtain

$$\begin{aligned}
\mathbb{E}\left[\mathcal{L}_{\text{main}}(\mathbf{W}^{t+1})\right] \leq\ & \mathbb{E}\left[\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right] - \frac{\eta}{2}\mathbb{E}\left[\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2\right] \\
& + \frac{G^2\eta}{2} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} \mathbb{E}\left[\left(1 - s^t_{j,g,0} - s^t_{j,g,m}\right)^2\right] \\
& + L\eta^2(G^2+\sigma^2) \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \mathbb{E}\left[\left(s^t_{j,g,0} + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s^t_{j,g,m}\right)^2\right].
\end{aligned} \tag{28}$$

Rearranging and summing over $T$ training rounds yields

$$\begin{aligned}
& \frac{\eta}{2} \sum_{t=1}^{T} \mathbb{E}\left[\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2\right] \\
\leq\ & \mathcal{L}_{\text{main}}(\mathbf{W}^1) - \mathbb{E}\left[\mathcal{L}_{\text{main}}(\mathbf{W}^{T+1})\right] \\
& + \frac{G^2\eta}{2} \sum_{t=1}^{T} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} \mathbb{E}\left[\left(1 - s^t_{j,g,0} - s^t_{j,g,m}\right)^2\right] \\
& + L\eta^2(G^2+\sigma^2) \sum_{t=1}^{T} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \mathbb{E}\left[\left(s^t_{j,g,0} + \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} s^t_{j,g,m}\right)^2\right].
\end{aligned} \tag{29}$$

Note that $\mathcal{L}_{\text{main}}$ is lower-bounded by $\mathcal{L}_{\text{main}}(\mathbf{W}^\star)$. We multiply $\frac{2}{T\eta}$ on both sides and obtain

$$\begin{aligned}
& \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\left\|\nabla\mathcal{L}_{\text{main}}(\mathbf{W}^t)\right\|_F^2\right] \\
\leq\ & \frac{2}{T\eta}\left(\mathcal{L}_{\text{main}}(\mathbf{W}^1) - \mathcal{L}_{\text{main}}(\mathbf{W}^\star)\right) + \frac{G^2}{T} \sum_{t=1}^{T} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \sum_{m\in\mathcal{M}} \pi_{\kappa(m)} \mathbb{E}\left[A^t_{1,j,g,m}\right] \\
& + \frac{2L\eta(G^2+\sigma^2)}{T} \sum_{t=1}^{T} \sum_{j\in\mathcal{J}} \sum_{g\in\mathcal{G}_j} \mathbb{E}\left[A^t_{2,j,g}\right].
\end{aligned} \tag{30}$$

This completes the proof of Theorem 1.

# D Implementation Settings and Additional Results

## D.1 Hyperparameter Settings

We present the hyperparameter settings of the 160M model, Qwen3-8B, and Qwen3-32B in Tables 7, 8, and 9, respectively. For each backbone, we select the learning rate using Dense validation NLL

Table 7: Training and modularization hyperparameters for the synthetic benchmark.

| Hyperparameter | Synthetic 160M |
|---|---|
| Domains | 6 |
| Backbone | 160M decoder-only |
| Optimizer | AdamW |
| Selected learning rate | $3\times10^{-4}$ |
| Learning-rate candidates | $\{5\times10^{-4}, 4\times10^{-4}, 3\times10^{-4}, 2\times10^{-4}, 1\times10^{-4}\}$ |
| Mini-batch size / samples per update | 32 / 32 |
| Batch-size candidates | $\{8, 16, 32, 64\}$ |
| Weight decay | 0.01 |
| Precision | bf16 |
| Seeds | 10 (2026–2035) |
| Total optimizer updates / epochs | 15360 / 20 |
| Warmup updates / validation interval | 768 / 256 |
| Specialized modules | 6 |
| Parameter grouping | One attention head;<br>128 FFN intermediate channels |

Table 8: Training and modularization hyperparameters for Qwen3-8B.

| Hyperparameter | Qwen3-8B |
|---|---|
| Domains | 6 |
| Backbone | Qwen3-8B |
| Optimizer | Adafactor |
| Selected learning rate | $5\times10^{-5}$ |
| Learning-rate candidates | $\{1\times10^{-4}, 9\times10^{-5}, 8\times10^{-5}, 7\times10^{-5}, 6\times10^{-5},$<br>$5\times10^{-5}, 4\times10^{-5}, 3\times10^{-5}, 2\times10^{-5}, 1\times10^{-5}\}$ |
| Mini-batch size / samples per update | 1 / 48 |
| Precision | bf16 |
| Seeds | 4 (2026–2029) |
| Total optimizer updates / epochs | 5000 / 1 |
| Warmup updates / validation interval | 150 / 250 |
| Specialized modules | 6 |
| Parameter grouping | One attention head;<br>128 FFN intermediate channels |

Table 9: Training and modularization hyperparameters for Qwen3-32B.

| Hyperparameter | Qwen3-32B |
|---|---|
| Domains | 6 |
| Backbone | Qwen3-32B |
| Optimizer | Adafactor |
| Selected learning rate | $2\times10^{-5}$ |
| Learning-rate candidates | $\{1\times10^{-4}, 9\times10^{-5}, 8\times10^{-5}, 7\times10^{-5}, 6\times10^{-5},$<br>$5\times10^{-5}, 4\times10^{-5}, 3\times10^{-5}, 2\times10^{-5}, 1\times10^{-5}\}$ |
| Mini-batch size / samples per update | 1 / 48 |
| Precision | bf16 |
| Seeds | 4 (2026–2029) |
| Total optimizer updates / epochs | 5000 / 1 |
| Warmup updates / validation interval | 150 / 250 |
| Specialized modules | 6 |
| Parameter grouping | One attention head;<br>128 FFN intermediate channels |

and use it for all methods. We average validation NLL equally over the six knowledge domains. The learning rate increases linearly during warmup and then follows cosine decay to zero. Training uses the full update budget without early stopping. We save checkpoints at each validation interval. Among checkpoints with the final model structure, we select the one with the lowest validation NLL. If multiple checkpoints have the same lowest NLL, we select the earliest one. The same selected checkpoint is used for all downstream benchmarks. For AdamW, we use $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and gradient clipping at an $L_2$ norm of 1. For Adafactor, we disable relative steps and parameter scaling. We also disable the first moment and external gradient clipping. We use $\epsilon = (10^{-30}, 10^{-3})$, decay rate $-0.8$, clipping threshold 1, and zero weight decay.

For CORTEX, we learn module assignments for the attention and FFN matrices. For attention matrices, we divide the query, key, and value matrices along the output axis and the attention output matrix along the input axis. A parameter group contains 64 channels for the 160M model and 128 channels for Qwen3-8B and Qwen3-32B, corresponding to one attention head. For FFN matrices, a parameter group contains 128 intermediate channels. The input, gate, and up matrices are divided along the output axis, while the output and down matrices are divided along the input axis. We learn the module assignment parameters separately for each of these matrices. If the remaining channels do not form a complete parameter group, they are kept as a smaller parameter group. The other model parameters remain trainable and are updated using the mixture gradient.

The assignment parameters $\Psi$ are initialized to zero, with all initial module assignment weights equal to $1/7$. We use Adam to update $\Psi$ by minimizing $\mathcal{L}_{\text{assign}}$. The learning rate is fixed at $10^{-3}$, with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-8}$, and zero weight decay. We update $\Psi$ once after gradients from all knowledge domains are available. We use FP32 to store $\Psi$ and calculate the utility scores. We calculate the gradient of $\mathcal{L}_{\text{assign}}$ w.r.t. $\Psi$, with the utility scores fixed.

For **Random**, we independently sample seven assignment logits from $\mathcal{N}(0, 1)$ for each parameter group. We apply softmax with temperature 1. The random seed is the training seed plus 10000. These assignments remain fixed throughout training.

For MoE, MoM, UpIT, and Self-MoE, we use top-1 activation. MoE replaces each FFN with six experts and a linear router. We set the load-balancing loss coefficient to 0.01 and keep all tokens. For Qwen3-8B and Qwen3-32B, each MoE expert is initialized from the corresponding FFN weights in the same pretrained checkpoint used by **Dense** and CORTEX. The router is randomly initialized, and all other layers use the original checkpoint weights.

For MoM, we divide the backbone into sets of six blocks. The 160M model, Qwen3-8B, and Qwen3-32B use 2, 6, and 10 sets, respectively. Qwen3-32B also uses four dense blocks. We use separate routers for attention and FFN. Within each set, each token uses each attention module and each FFN module once over six steps. MoM matches **Dense** in the number of backbone parameters and layers applied to each token. The routers introduce additional parameters.

For UpIT, we construct six FFN experts from equally spaced checkpoints during the first 60% of training. This stage uses 9216 updates for the 160M model and 3000 updates for each Qwen3 model. We initialize the other parameters from the last of these checkpoints. We then freeze the backbone and experts and train the router for the remaining 6144 and 2000 updates, respectively. We first train the router for one pass on a subset of the training data. Each domain contributes 256 samples for the 160M model and 512 samples for each Qwen3 model. This uses 48 and 64 updates, respectively, within the router training budget. The remaining numbers of updates are 6096 and 1936.

For Self-MoE, we use six LoRA experts with rank 8, alpha 16, and dropout 0.05. LoRA is applied to the query, key, value, attention output, and FFN matrices. We use the original training data without self-generated samples. We train the dense backbone during the first 40% of updates. We then fix the backbone and train the domain experts during the next 40%. The remaining 20% train the router. Each domain uses 256 seed samples for the 160M model and 512 seed samples for each Qwen3 model.

### D.2 Dataset Preparation

The synthetic benchmark is produced by a deterministic rule-based generator with fixed random seeds. We define six synthetic knowledge domains: COPY, REVERSE, ADD, LOOKUP, LOOKUP->ADD, and REVERSE->LOOKUP. For each example, the generator first samples a domain label from fixed domain priors, and then instantiates the corresponding symbolic task using a per-example pseudorandom seed. COPY samples a digit sequence and outputs the same sequence. REVERSE outputs the reversed sequence. ADD samples two 2-digit or 3-digit integers and outputs their decimal sum. LOOKUP maps a three-letter key to the weighted sum of its zero-based alphabet positions modulo 10, using weights $(1, 2, 4)$. REVERSE->LOOKUP applies this rule to the reversed key; both tasks retain only keys whose outputs differ under reversal. LOOKUP->ADD outputs the sum of two lookup values. In our implementation, the default synthetic seed is 2026, with domain priors [0.16, 0.16, 0.16, 0.18, 0.17, 0.17]. For the synthetic benchmark, the train/validation/test sizes are 24576/3072/3072, with each of the six domains uniformly contributing 4096/512/512 samples,

respectively. We stop adding samples to a domain after reaching these sample counts. We remove duplicate input–target pairs across splits. We calculate the loss on the answer and EOS tokens. The domain mixture weights used during training follow the stated domain priors. All samples are generated by a deterministic rule-based generator and are truncated or padded to a maximum sequence length of 48 for the 160M decoder-only backbone.

For the real-domain benchmark, the train/validation/test sizes are 240000/7200/10200, with each of the six domains uniformly contributing 40000/1200/1700 samples, respectively. All samples are truncated or padded to a maximum sequence length of 768 for both Qwen3-8B and Qwen3-32B. We split the real-domain data using seed 2026 after removing duplicate source examples. For WinoGrande, we split 41665 labeled source examples into 38765 training, 1200 validation, and 1700 test examples. We exchange the two answer options in 1235 training examples and add the resulting examples to the training set. This produces 40000 training samples. For general, biomedical, and legal text, we calculate the loss on the full text. For code and mathematics, we calculate the loss on the answer and EOS tokens. For WinoGrande, we use the answer label. All methods use the same tokenized samples. Each sequence contains one sample.

### D.3 Implementation of Domain-conditioned Group Gradients

We clarify how the domain-conditioned group gradients in Section 3.2 are calculated in our implementation. The mini-batch size denotes the number of samples used for each gradient calculation. Each update uses 32 samples for the 160M model and 48 samples for each Qwen3 backbone. For the 160M model, each update uses five samples per domain and two additional samples. The two additional samples come from domain pairs $(0, 1)$, $(2, 3)$, and $(4, 5)$ in turn. For both Qwen3 backbones, each update uses eight samples per domain. Qwen3-8B uses 12 mini-batches per GPU across 4 GPUs. Qwen3-32B uses 3 mini-batches per GPU across 16 GPUs. CORTEX collects domain-conditioned group gradients from all samples used for the update. These samples cover all six knowledge domains. During gradient collection, the backbone parameters remain fixed. For each knowledge domain, we weight each mini-batch gradient by the number of tokens used to calculate its loss and divide the sum by the total number of these tokens. Padding tokens and tokens excluded from the loss calculation are not counted. For each knowledge domain, Dense and CORTEX calculate the average loss over the same tokens. For the real-domain benchmark, both methods use equal domain weights, i.e., $\pi_k = 1/6$. CORTEX further weights each domain-conditioned group gradient by the sum of the shared and corresponding specialized module assignment weights.

When the mini-batch size is one, each mini-batch contributes gradients to a single knowledge domain. After collection, CORTEX calculates the utility scores from the average domain-conditioned group gradients and updates $\Psi$ and the backbone parameters simultaneously. The domain-conditioned group gradients are combined before updating the backbone parameters. We use AdamW for the 160M model and Adafactor for Qwen3-8B and Qwen3-32B. The backbone optimizer uses one set of optimizer states for each parameter and performs one update after gradient collection. The accumulated gradients are cleared after each update. All approaches use the same training-token budget and total number of optimizer updates. The learning-rate schedule is based on the number of optimizer updates. For UpIT and Self-MoE, this budget includes expert construction and router training. Domain labels are used only for training and to report two module-level interpretability metrics. During inference, CORTEX uses the trained dense model directly. The learned modules are soft components of the dense model rather than routed experts. For Table 4, we remove each specialized module separately from Random and CORTEX and calculate the NLL changes on its paired domain and the other domains.

### D.4 Downstream Evaluation Protocol

We use the corresponding Qwen3 chat template with one user message and no system message, demonstrations, or instructions for intermediate reasoning. We disable Qwen3 thinking and use greedy decoding in bf16, generating one answer per question. All methods use the same prompts, generation limits, and scorers. The maximum input length is 32768 tokens. We generate at most 32 tokens for MMLU-Pro, 4096 for LiveCodeBench, and 2048 for MATH/GSM8K. The limits are 16 for PubMedQA, 256 for LegalBench, and 256 for BBH. Generation stops at EOS or the output length limit. Unparseable answers are counted as incorrect without regeneration. The

Table 10: Synthetic benchmark EM (%). We show the mean ± standard deviation over 10 different seeds.

| Baseline | COPY | REVERSE | ADD | LOOKUP | LOOKUP→ADD | REVERSE→LOOKUP |
|---|---|---|---|---|---|---|
| Dense | 96.82±0.31 | 91.36±0.44 | 88.27±0.52 | 92.74±0.39 | 84.91±0.61 | 86.38±0.57 |
| Random | 88.54±0.66 | 87.92±0.58 | 80.76±0.73 | 91.18±0.47 | 73.64±0.85 | 86.95±0.60 |
| MoE | 98.12±0.25 | 95.06±0.34 | 94.38±0.41 | 95.71±0.33 | 87.46±0.56 | 91.64±0.46 |
| MoM | 98.67±0.21 | 97.84±0.27 | 96.21±0.33 | 96.14±0.31 | 90.82±0.49 | 94.76±0.39 |
| UpIT | 96.21±0.36 | 93.71±0.42 | 92.04±0.48 | 96.48±0.32 | 86.39±0.59 | 90.43±0.50 |
| Self-MoE | 97.84±0.28 | 96.52±0.31 | 93.91±0.43 | 95.18±0.35 | 89.34±0.52 | 95.71±0.37 |
| **CORTEX** | **99.42±0.18** | **98.83±0.22** | **97.62±0.29** | **98.14±0.26** | **94.76±0.41** | **96.88±0.33** |

Table 11: Real-domain benchmark NLL on Qwen3-8B. We show the mean ± standard deviation over 4 different seeds.

| Baseline | General | Code | Math | Biomedical | Legal | Reasoning |
|---|---|---|---|---|---|---|
| Dense | 1.6766±0.0124 | 1.7949±0.0141 | 2.0047±0.0168 | 1.8220±0.0149 | 1.9425±0.0161 | 1.9683±0.0157 |
| Random | 1.7997±0.0175 | 1.7924±0.0158 | 2.0172±0.0194 | 1.9534±0.0182 | 2.0053±0.0197 | 1.9392±0.0171 |
| MoE | 1.6642±0.0119 | 1.7878±0.0136 | 1.9262±0.0155 | 1.7947±0.0142 | 1.9054±0.0153 | 1.9133±0.0149 |
| MoM | 1.6441±0.0112 | 1.6578±0.0124 | 1.9290±0.0151 | 1.7995±0.0138 | 1.8714±0.0147 | 1.8385±0.0136 |
| UpIT | 1.6257±0.0108 | 1.7658±0.0131 | 2.0039±0.0162 | 1.7709±0.0134 | 1.8171±0.0140 | 1.8288±0.0132 |
| Self-MoE | 1.6684±0.0121 | 1.7373±0.0129 | 1.9058±0.0148 | 1.8208±0.0145 | 1.8583±0.0146 | **1.7822±0.0128** |
| **CORTEX** | **1.6199±0.0097** | **1.6096±0.0115** | **1.8694±0.0137** | **1.6778±0.0126** | **1.7791±0.0131** | 1.8258±0.0124 |

Table 12: Real-domain benchmark NLL on Qwen3-32B. We show the mean ± standard deviation over 4 different seeds.

| Baseline | General | Code | Math | Biomedical | Legal | Reasoning |
|---|---|---|---|---|---|---|
| Dense | 1.4202±0.0108 | 1.5311±0.0122 | 1.7400±0.0145 | 1.5648±0.0129 | 1.6712±0.0136 | 1.7221±0.0140 |
| Random | 1.5298±0.0149 | 1.5193±0.0135 | 1.7889±0.0161 | 1.6876±0.0152 | 1.7272±0.0157 | 1.6897±0.0148 |
| MoE | 1.3884±0.0099 | 1.4383±0.0107 | 1.6240±0.0128 | 1.4878±0.0114 | 1.6364±0.0130 | 1.6111±0.0125 |
| MoM | 1.3918±0.0101 | 1.4967±0.0119 | 1.5815±0.0122 | 1.5384±0.0120 | 1.5942±0.0127 | 1.6129±0.0124 |
| UpIT | **1.3578±0.0093** | 1.4733±0.0112 | 1.6766±0.0134 | 1.5100±0.0117 | 1.5514±0.0119 | 1.5945±0.0121 |
| Self-MoE | 1.3871±0.0098 | 1.4395±0.0108 | 1.5845±0.0120 | 1.5180±0.0118 | 1.6011±0.0126 | 1.5597±0.0116 |
| **CORTEX** | 1.3820±0.0091 | **1.4051±0.0102** | **1.5798±0.0117** | **1.4344±0.0109** | **1.5426±0.0110** | **1.5279±0.0111** |

prompts request an option letter for MMLU-Pro, Python 3 code for LiveCodeBench, a boxed final answer for MATH/GSM8K, and `yes`, `no`, or `maybe` for PubMedQA. LegalBench and BBH retain their task instructions and answer formats. Generated code is executed only for scoring. We use `lcb_runner` scorer. The limits are 6 seconds per test, 60 seconds per problem, and 4 GiB of memory. MMLU-Pro uses all 12032 test questions. PubMedQA uses the 500 official test IDs from `pqa_labeled`. Both use accuracy over all evaluated questions. LiveCodeBench uses all 880 problems in `code_generation_lite`, `release_v5`, from May 2023 to January 2025. We report single-generation pass@1. For Math, we use all 5000 MATH test questions and 1319 GSM8K test questions. We calculate accuracy for each benchmark and average the two scores equally. LegalBench uses revision `6b4b377`. We exclude `rule_qa` and use all 90844 test examples from the remaining 161 tasks. We average the official scores equally across these tasks. BBH uses all 6511 examples from 27 task configurations. We calculate exact match for each configuration and average the scores equally. The object-count variants of logical deduction and shuffled-object tracking are scored separately. Each domain score is averaged over four independently trained seeds, with sample standard deviation. For paired differences, we first subtract the baseline score from the CORTEX score for each seed. We then calculate the average and sample standard deviation.

### D.5 Additional Results

In this section, we present additional experimental results of our work.

Table 10 shows that CORTEX outperforms the baseline schemes on the synthetic benchmark using the 160M model.

In addition, Tables 11 and 12 compare the NLL on the real-domain benchmark using Qwen3-8B and Qwen3-32B, respectively. CORTEX achieves the best or highly competitive NLL across most knowledge domains under both backbone models. Compared with the **Dense** baseline, CORTEX

Table 13: Paired differences in downstream scores (CORTEX minus the best baseline in Table 2). We show the average ± standard deviation of the differences over four paired training seeds.

| Backbone | General | Code | Math | Biomedical | Legal | Reasoning |
|---|---|---|---|---|---|---|
| Qwen3-8B | $+0.0115\pm0.0046$ | $-0.0023\pm0.0065$ | $+0.0340\pm0.0075$ | $+0.0075\pm0.0044$ | $+0.0253\pm0.0051$ | $-0.0063\pm0.0069$ |
| Qwen3-32B | $-0.0091\pm0.0038$ | $+0.0323\pm0.0053$ | $+0.0149\pm0.0061$ | $+0.0330\pm0.0042$ | $+0.0035\pm0.0046$ | $+0.0283\pm0.0058$ |

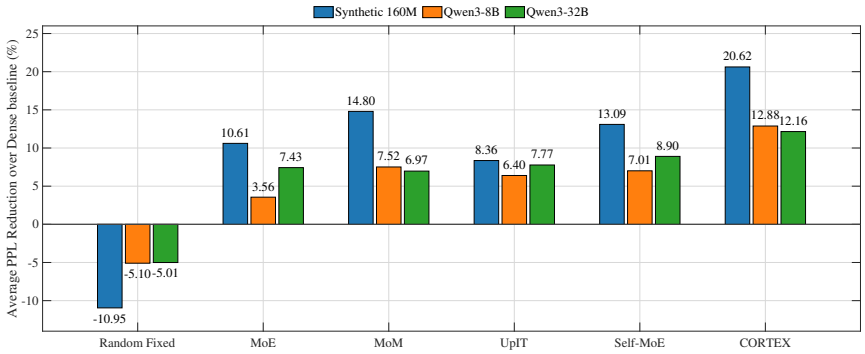



Figure 5: PPL reduction over the Dense baseline, calculated from the average NLL across knowledge domains.

Table 14: Average paired-domain SLS for original and matched-shuffle assignments. Difference denotes original minus matched-shuffle SLS. Values are averages ± standard deviations across training seeds.

| Backbone | Original | Matched shuffle | Difference |
|---|---|---|---|
| 160M | $0.1837\pm0.0120$ | $0.0548\pm0.0100$ | $+0.1289\pm0.0115$ |
| Qwen3-8B | $0.1452\pm0.0150$ | $0.0427\pm0.0120$ | $+0.1026\pm0.0140$ |

consistently reduces NLL on general text, code, math, biomedical text, legal text, and reasoning data, showing that the learned modularization improves the learning performance under heterogeneous real-domain mixtures. Compared with MoE-based and other modular baselines, CORTEX provides a more balanced performance across domains, which indicates that the shared and specialized modules within the dense model can better preserve common knowledge while capturing domain-specific knowledge.

Table 13 reports the paired differences in downstream scores between CORTEX and the best baseline in each domain over four training seeds.

Fig. 5 further shows the average PPL reduction over the Dense baseline across the synthetic benchmark, Qwen3-8B, and Qwen3-32B, respectively. CORTEX obtains the largest average PPL reduction in all three settings, while random assignment consistently performs worse than Dense. Under the same domain-conditioned gradient collection and backbone update rule, CORTEX consistently outperforms fixed random module assignments. The consistent improvement from the 160M model to Qwen3-8B and Qwen3-32B further indicates that CORTEX can scale to larger dense backbones while maintaining its advantage over existing baseline schemes.

We further compare the learned assignments with randomly shuffled assignments at the same trained checkpoints. We shuffle complete assignment vectors among same-shaped parameter groups within each matrix and keep the module–domain pairing and evaluation set fixed. For each matrix and module, we rescale the weights removed by the shuffled assignment to match the Frobenius norm of those removed by the original assignment. This matched-shuffle control accounts for differences in removal magnitude when comparing SLS. We use 10 training seeds with 100 shuffles per seed for 160M and 4 training seeds with 20 shuffles per seed for Qwen3-8B. We average SLS across shuffles within each seed and calculate paired differences before reporting the average ± standard

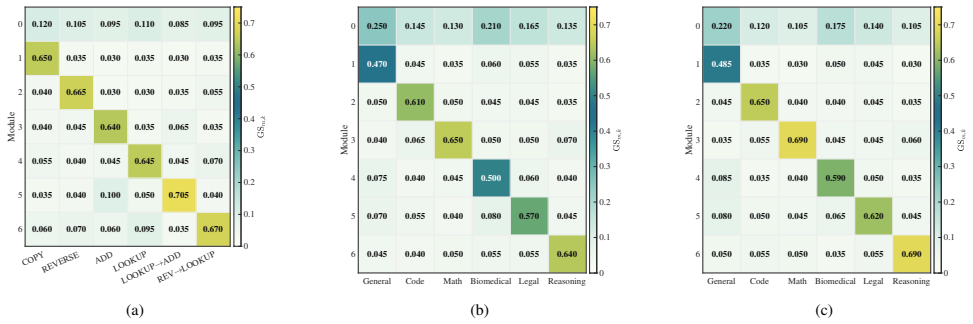



Figure 6: Domain-normalized gradient share ($\mathrm{GS}_{m,k}/\mathrm{GS}_k^{\mathrm{domain}}$) on (a) 160M model, (b) Qwen3-8B, and (c) Qwen3-32B. The columns denote knowledge domains and the rows denote modules. Each column sums to one.

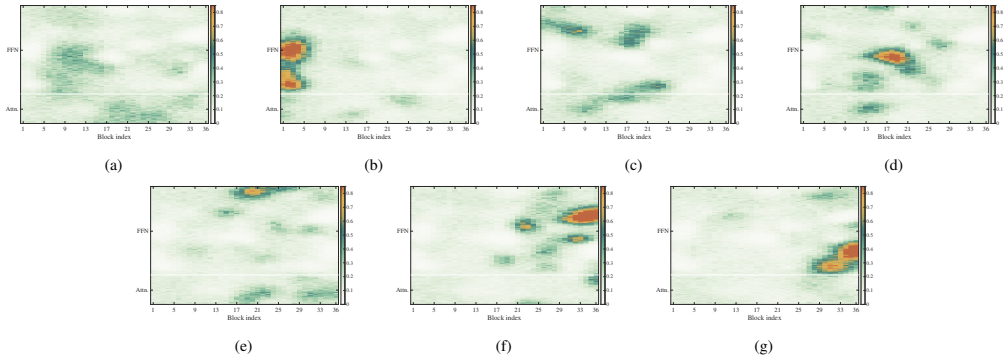



Figure 7: Visualization of modularization of CORTEX on the Qwen3-8B backbone model trained on the real-domain data mixture. The seven heatmaps correspond to the shared knowledge and six knowledge domains, including (a) Shared, (b) General, (c) Code, (d) Math, (e) Biomedical, (f) Legal, and (g) Reasoning, respectively. The columns denote Transformer blocks and the rows denote parameter groups within attention heads (Attn.) and the feed-forward network (FFN).

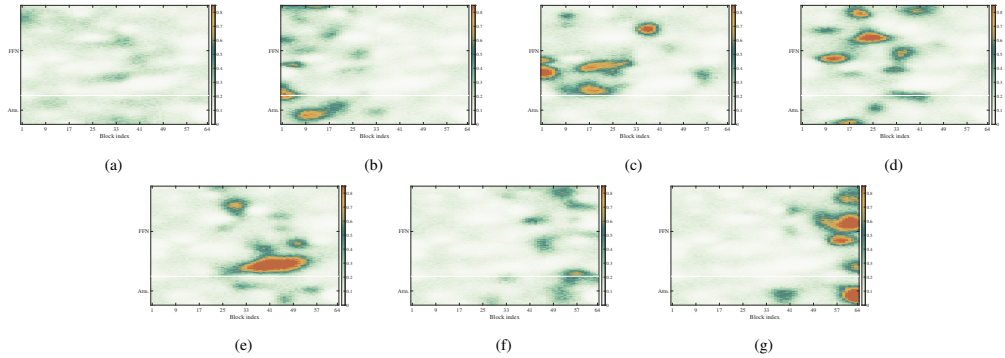



Figure 8: Visualization of modularization of CORTEX on the Qwen3-32B backbone model trained on the real-domain data mixture. The seven heatmaps correspond to the shared knowledge and six knowledge domains, including (a) Shared, (b) General, (c) Code, (d) Math, (e) Biomedical, (f) Legal, and (g) Reasoning, respectively. The columns denote Transformer blocks and the rows denote parameter groups within attention heads (Attn.) and the feed-forward network (FFN).

deviation across training seeds. Table 14 shows higher average SLS for the original assignments on both backbones, supporting domain-selective localization under the matched-shuffle control.

For Fig. 6, we average the joint gradient shares across seeds and then normalize each column to sum to one. For each knowledge domain, the corresponding specialized module receives the largest gradient share, while the shared module receives gradients from all domains. We calculate $I^{\mathrm{mod}}$ from the joint gradient shares in Definition 4 for each seed and report the average in Table 4.

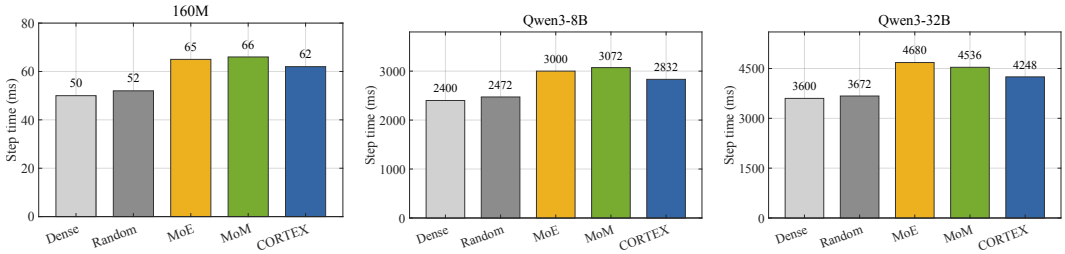



Figure 9: Comparison of training step time on the 160M, Qwen3-8B, and Qwen3-32B backbones.

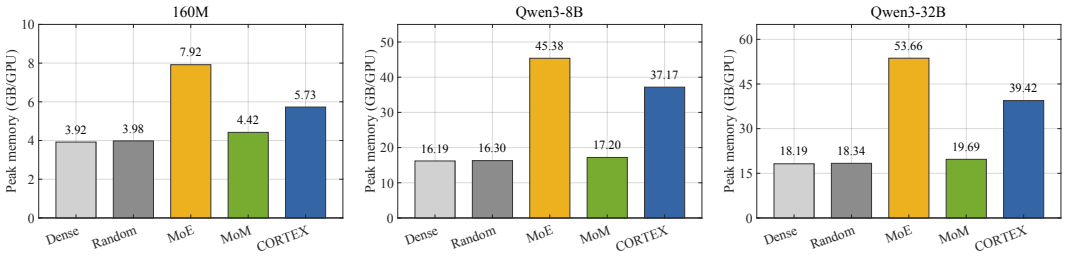



Figure 10: Comparison of peak memory per GPU on the 160M, Qwen3-8B, and Qwen3-32B backbones.

Figs. 7 and 8 show the modularization of CORTEX on Qwen3-8B and Qwen3-32B. The module assignment weight patterns show that CORTEX forms both shared and domain-specialized modules across Transformer blocks, indicating that the modular structure persists when the backbone model scales from the 160M model to larger dense LLMs.

Figs. 9 and 10 compare the training step time and peak memory per GPU across different schemes. Step time is measured over the same token count across methods and includes the complete domain-gradient collection window and both assignment and backbone updates for CORTEX. The 160M, Qwen3-8B, and Qwen3-32B backbones use 1, 4, and 16 H100 80GB GPUs, respectively. When compared with **Dense**, CORTEX increases the step time by 18–24% and uses 1.46–2.30 times the peak memory per GPU. CORTEX has a lower step time than MoE and MoM and lower peak memory per GPU than MoE.

### D.6 Compute Resources

All experiments were conducted on an anonymized institutional H100 GPU cluster using SLURM scheduling. Our GPU experiments were run on NVIDIA H100 SXM5 80GB GPU nodes. For the synthetic benchmark, training and evaluation of the 160M backbone required approximately 523.311 GPU hours, including all baselines and repeated runs. For the real-domain experiments, the Qwen3-8B experiments required approximately 933.486 GPU-hours, and the Qwen3-32B experiments required approximately 8298.656 GPU hours. In total, all experiments in this project used approximately 9755.453 H100 GPU hours. All approaches were evaluated under the same budget within each model scale. These totals include the presented experiments, preliminary experiments, and unsuccessful approach variants that are not reported.

## E Limitations

CORTEX relies on meaningful knowledge-domain labels to estimate domain-conditioned group gradients. This assumption is natural for controlled or curated data mixtures, but may be less accurate when knowledge domains become overlapped or noisy. The learned modularization also depends on the chosen parameter-group granularity and the number of specialized modules: fine-grained parameter groups may result in noisy module assignment, while overly coarse parameter groups may limit the specialization to target knowledge domains. CORTEX introduces overhead from storing and processing domain-conditioned gradients and updating $\Psi$. No inference-time routing overhead is introduced. Finally, CORTEX is not designed as an inference acceleration approach since it aims to improve dense training and provide an interpretable modular decomposition.