Domain Mixture Design via Log-Likelihood Differences for Aligning Language Models with a Target Model

Anonymous ACL submission

Abstract

Instead of directly distilling a language model, this study addresses the problem of aligning a base model with a target model in distribution by designing the domain mixture of training data for pretraining or continued pretraining as a fixed training recipe. We propose a method for determining domain weights by viewing models as points in log-likelihood space and aligning the training update direction with the direction toward the target model. Experiments with NanoGPT show that the proposed method consistently reduces the KL divergence between the trained base model and the target model relative to training with Pile-original weighting. Although knowledge distillation remains more effective when available, the proposed method achieves meaningful alignment, and downstream task performance also tends to become closer to that of the target model.

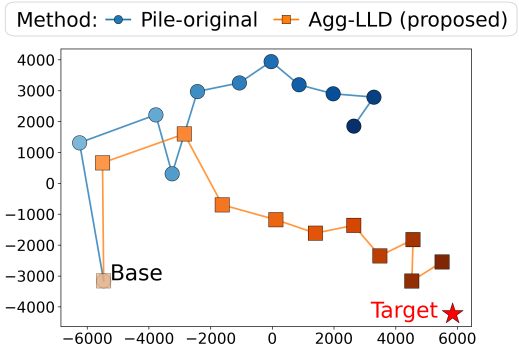


Figure 1: t-SNE visualization of training trajectories in the log-likelihood vector space during continued pretraining from pretrained NanoGPT toward Gemma-2B. The trajectories are obtained using either Pile-original domain weights or weights determined by the proposed method (aggregated-LLD). Each point represents a checkpoint taken every 1k training steps, and color intensity indicates the number of training steps. The star denotes the target model. Compared with Pile-original weighting, the proposed method yields a trajectory that moves more consistently toward the target model. See Section 4.3 for details.

1 Introduction

The properties of large language models depend not only on their architecture and model size but also on the distribution of text used for training (Albalak et al., 2024). In particular, large-scale corpora such as the Pile (Gao et al., 2021) consist of multiple domains, and the mixture weights over these domains strongly affect model behavior (Xie et al., 2023). Therefore, designing the domain mixture of the training data is an important problem for enabling models to acquire desired properties.

In this study, we address the problem of determining the training text distribution for updating a base model so as to align it with a target model in distribution. A common approach is to directly imitate the target model through knowledge distillation (Hinton et al., 2015) or by training on text generated by the target model (Kim and Rush, 2016). However, such methods require access to the teacher model and incur additional generation or inference costs. Moreover, even when the domain weights used in prior training are known, simply reusing them is not necessarily optimal in the current setting. We therefore aim to achieve model alignment by designing the domain weights of the training text as a *fixed training recipe* specified before training begins. The estimated domain weights can serve not only as a target-informed starting point for further training under different settings, but also as an interpretable summary of the domain-level shift from the base model toward the target.

We represent a model as a vector of mean log-likelihoods over multiple text collections (Oyama et al., 2025). In this study, we use the domains of the Pile (Gao et al., 2021) as these text collections. Directly comparing different models in parameter space is difficult because of architectural mismatches and permutation symmetries of network weights induced by hidden-unit reordering (Hecht-

Nielsen, 1990). In contrast, mean log-likelihood vectors computed on a set of text collections provide a common basis for comparison across models. With this representation, model alignment can be viewed as a geometric problem of moving the base model toward the target model, leading to a rule for determining mixture proportions based on the domain-level log-likelihood differences (LLD) between the target and base models.

In experiments using NanoGPT as the base model, determining the domain weights for the Pile using the proposed method consistently reduces the KL divergence between the trained NanoGPT and the target models, Gemma-2B or CodeGemma-2B, compared with training using Pile-original weights (see Fig. 1). For target models that share the same tokenizer, the proposed method provides a useful alternative to knowledge distillation.

2 Background and Related Work

2.1 Data selection and mixing for LLMs

Selecting and mixing training data has become an important issue in LLM training, as performance depends not only on the amount of data but also on its distribution (Albalak et al., 2024). In particular, for multi-domain corpora such as the Pile (Gao et al., 2021), the choice of domain weights substantially affects model properties and can lead to markedly different training trajectories in log-likelihood space (Fig. 2). This has led to active research on optimizing training data selection and domain weights for downstream tasks, domain adaptation, and training efficiency (Zhou et al., 2023; Xie et al., 2023; Ye et al., 2025; Liu et al., 2024; Dohmatob et al., 2025).

As a classical data selection method, Moore and Lewis (2010) proposed ranking texts in a general-domain corpus based on the log-likelihood difference between a language model trained on the target domain and another trained on the general domain. Like this method, our study uses log-likelihood differences between two models on text. The two approaches differ in three important respects: our method estimates domain weights rather than selecting texts, aims to align the base model with the target model in distribution, and derives a rule for determining domain weights from a geometric perspective in log-likelihood space.

Recently, several studies have explored how to choose effective domain weights for language model training. Xie et al. (2023) estimated the difficulty of each domain based on the log-likelihoods of a small model and determined the domain weights for pretraining a larger model. Liu et al. (2024), in a knowledge distillation setting, dynamically updated the domain weights of the distillation data according to the domain-level likelihood ratios between the teacher and the student. In contrast, our study aims to align a base model with a specific target model by designing domain weights as a fixed training recipe, without using distillation. Our study can therefore be viewed as distillation-free data mixture design for distributional alignment to a target model.

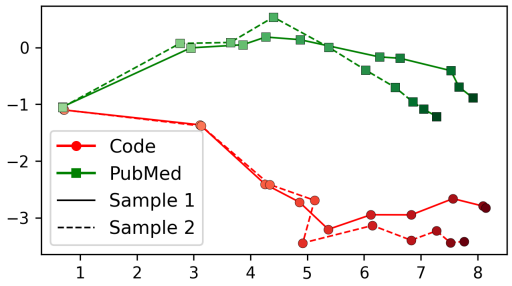


Figure 2: t-SNE visualization of NanoGPT training trajectories for two training datasets independently sampled from each of two domain mixtures over the Pile. Each point represents a checkpoint taken every 1k training steps, and color intensity indicates the number of training steps. Differences in trajectory are much larger across domain mixtures than across datasets sampled from the same mixture. The two mixtures are obtained by quadrupling the weights of either the code-related domains or the PubMed-related domains relative to the original Pile weights. See Appendix B for details.

2.2 Log-likelihood vector

As a framework for comparing models in a shared space, Oyama et al. (2025) introduced, for a language model p, the text-level log-likelihood vector over a set of N texts $\mathcal{D} = \{x_1, \ldots, x_N\}$,

$$\boldsymbol{\ell}^{\text{text}} = (\log p(x_1), \ldots, \log p(x_N))^\top \in \mathbb{R}^N, \quad (1)$$

which serves as a representation of p. For M models (or checkpoints along a training trajectory) $p_1, \ldots, p_M$, let

$$\boldsymbol{L} = (\boldsymbol{\ell}_1^{\text{text}}, \ldots, \boldsymbol{\ell}_M^{\text{text}})^\top \in \mathbb{R}^{M \times N}$$

and define

$$\boldsymbol{Q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_M)^\top \in \mathbb{R}^{M \times N}$$

as the matrix obtained by double-centering[1] $\boldsymbol{L}$. Then, the KL divergence between p_i and p_j is ap-

[1] Centering the log-likelihood matrix row-wise over texts and column-wise over models.

proximated as

$$2\,\mathrm{KL}(p_i, p_j) \approx \|\boldsymbol{q}_i - \boldsymbol{q}_j\|^2/N. \tag{2}$$

This approximation enables comparison of models with different architectures in a shared geometric space. Moreover, log-likelihood vectors have been shown to be useful for predicting downstream task performance. They have also been applied to the analysis of training trajectories during pretraining (Kishino et al., 2026) and to the analysis of how training data contributes to learning efficiency (Harada et al., 2025; Aoki et al., 2025).

Building on this framework, we use the text-level log-likelihood vector to analyze training trajectories and the domain-level log-likelihood vector, consisting of mean log-likelihoods for each domain, to design domain weights for training.

3 Method

In this study, we consider domain weights for distributionally aligning a base model with a target model in log-likelihood space. Our main objective is to design a single set of domain weights fixed once before training. We first formulate the problem and derive a theoretical criterion that determines domain weights iteratively based on update-direction alignment as a first pass. We then aggregate the resulting weights into a form that is easier to use as a fixed recipe in a second pass.

3.1 Text distribution

For each domain $k \in \{1, \ldots, K\}$, let $r_k(x)$ denote the domain-specific text distribution[2]. Let

$$\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)^\top \in \Delta^{K-1}$$

denote the domain weights, where

$$\Delta^{K-1} = \left\{ \boldsymbol{\pi} : \sum_{k=1}^{K} \pi_k = 1,\ \pi_k \geq 0 \right\}$$

is the $(K-1)$-dimensional probability simplex. We then define the corresponding mixture text distribution as

$$r_{\boldsymbol{\pi}}(x) = \sum_{k=1}^{K} \pi_k r_k(x).$$

[2]For simplicity, we refer to the k-th domain simply as domain k. In practice, the domains include, for example, GitHub and arXiv.

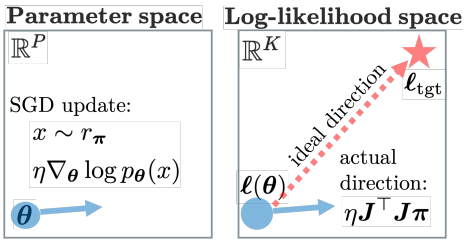


Figure 3: Illustration of the update direction in log-likelihood space when the base model $p_{\boldsymbol{\theta}}$ is updated along the gradient direction in parameter space using a text sampled from the mixture distribution $r_{\boldsymbol{\pi}}$. Our goal is to estimate $\boldsymbol{\pi}$ so that the blue update direction aligns with the red direction toward the target model.

3.2 Domain-level log-likelihood vector

For each k, let $\mathcal{D}_k$ denote a set of texts sampled from r_k, and let

$$\mathcal{D} = \bigcup_{k=1}^{K} \mathcal{D}_k$$

be the union of these sets. We refer to $\mathcal{D}$ as the evaluation corpus.

We consider a base model $p_{\boldsymbol{\theta}}(x)$ and a target model $p_{\mathrm{tgt}}(x)$, where $\boldsymbol{\theta} \in \mathbb{R}^P$ denotes the parameters of the base model. The goal of this study is to align the base model with the target model through pretraining or continued pretraining. The two models need not share the same architecture.

For the base model $p_{\boldsymbol{\theta}}(x)$, we define the domain-level log-likelihood vector on the evaluation corpus $\mathcal{D}$ by

$$\boldsymbol{\ell}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{1}{|\mathcal{D}_1|} \sum_{x \in \mathcal{D}_1} \log p_{\boldsymbol{\theta}}(x) \\ \vdots \\ \frac{1}{|\mathcal{D}_K|} \sum_{x \in \mathcal{D}_K} \log p_{\boldsymbol{\theta}}(x) \end{pmatrix} \in \mathbb{R}^K. \tag{3}$$

Similarly, let $\boldsymbol{\ell}_{\mathrm{tgt}} \in \mathbb{R}^K$ denote the domain-level log-likelihood vector for the target model $p_{\mathrm{tgt}}(x)$.

3.3 Derivation of a rule for determining domain weights

We first derive how training changes the log-likelihood vector. Assuming SGD with learning rate $\eta > 0$, the parameter update with a text $x \sim r_{\boldsymbol{\pi}}$ is given by

$$\Delta\boldsymbol{\theta} := \eta \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x), \quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}.$$

Let the domain-level Jacobian on the evaluation corpus be

$$\boldsymbol{J}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\ell}(\boldsymbol{\theta})^\top \in \mathbb{R}^{P \times K}.$$

Then, the expected change in the log-likelihood vector is approximated as

$$\mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}[\boldsymbol{\ell}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})-\boldsymbol{\ell}(\boldsymbol{\theta})] \approx \eta \boldsymbol{J}(\boldsymbol{\theta})^\top \boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{\pi}. \quad (4)$$

See Appendix A.1 for the proof.

For simplicity, we omit the explicit dependence on $\boldsymbol{\theta}$ and write $\boldsymbol{\ell}$ and $\boldsymbol{J}$. We refer to $\boldsymbol{J}^\top\boldsymbol{J}$ as the Gram matrix[3]. Equation (4) implies that the expected update direction in log-likelihood space is approximated by $\eta\boldsymbol{J}^\top\boldsymbol{J}\boldsymbol{\pi}$, shown as the blue arrow in Fig. 3. Our goal is to align this direction with the target direction $\boldsymbol{\ell}_{\text{tgt}} - \boldsymbol{\ell}$, shown as the red arrow.

If we ignore the constraint $\boldsymbol{\pi} \in \Delta^{K-1}$, we seek $\boldsymbol{\pi}$ such that the expected update direction matches the target direction[4]:

$$\boldsymbol{J}^\top\boldsymbol{J}\boldsymbol{\pi} = \boldsymbol{\ell}_{\text{tgt}} - \boldsymbol{\ell}.$$

The unconstrained weight vector satisfying this relation is

$$\tilde{\boldsymbol{\pi}} = (\boldsymbol{J}^\top\boldsymbol{J})^{-1}(\boldsymbol{\ell}_{\text{tgt}} - \boldsymbol{\ell}).$$

To obtain domain weights in Δ^{K-1}, we consider the optimization problem

$$\max_{\boldsymbol{\pi}\in\Delta^{K-1}} \boldsymbol{\pi}^\top\tilde{\boldsymbol{\pi}} - \tau\,\mathrm{KL}(\boldsymbol{\pi},\boldsymbol{\pi}'). \quad (5)$$

Here, $\tau > 0$ is the temperature parameter controlling the strength of regularization, and $\mathrm{KL}(\boldsymbol{\pi},\boldsymbol{\pi}')$ is a regularization term that encourages domain weights close to the reference weights $\boldsymbol{\pi}'$, thereby avoiding extreme weights [5]. The solution to (5) is the softmax of $\tilde{\boldsymbol{\pi}}/\tau$:

$$\pi_k = \frac{\pi'_k \exp(\tilde{\pi}_k/\tau)}{\sum_{k'=1}^{K} \pi'_{k'} \exp(\tilde{\pi}_{k'}/\tau)}. \quad (6)$$

The proof is given in Appendix A.2.

In practice, computing the Gram matrix $\boldsymbol{J}^\top\boldsymbol{J}$ requires $O(K^2P)$ time, which imposes substantial computational and memory costs for large language models with a huge number of parameters P. We therefore omit $(\boldsymbol{J}^\top\boldsymbol{J})^{-1}$, which provides sensitivity adjustment across domains, and instead use a heuristic approximation given by the softmax of $(\boldsymbol{\ell}_{\text{tgt}} - \boldsymbol{\ell})/\tau$:

$$\pi_k = \frac{\pi'_k \exp((\ell_{\text{tgt},k} - \ell_k)/\tau)}{\sum_{k'=1}^{K} \pi'_{k'} \exp((\ell_{\text{tgt},k'} - \ell_{k'})/\tau)}. \quad (7)$$

In what follows, we use this approximation as a practical mixture design method, while treating the method with the adjustment matrix $(\boldsymbol{J}^\top\boldsymbol{J})^{-1}$ as a theoretical reference method.

3.4 Iterative estimation of domain weights and aggregation

As (7) indicates, the domain weights depend on $\boldsymbol{\ell}(\boldsymbol{\theta})$, so the theoretically desirable weights change during training. We therefore adopt a two-pass procedure. In the first pass, the domain weights are computed at selected update steps and then aggregated into a single set of weights, $\boldsymbol{\pi}^*$, to be used as a fixed training recipe. In the second pass, the model is trained using $\boldsymbol{\pi}^*$, without further reference to the target model.

Here, $\mathcal{D}$ denotes the evaluation corpus used to compute the domain-level log-likelihood vector, and $\mathcal{C}_k$ denotes the training corpus for domain k. Let

$$\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$$

denote the set of training steps at which domain weights are estimated. Given domain weights $\boldsymbol{\pi}^{(t)}$ at step t, we define the fixed domain weights $\boldsymbol{\pi}^*$ by geometric-mean aggregation over $t \in \mathcal{T}$:

$$\pi^*_k \propto \left(\prod_{t\in\mathcal{T}} \pi_k^{(t)}\right)^{1/|\mathcal{T}|}. \quad (8)$$

This coincides with the point in Δ^{K-1} that minimizes the sum of KL divergences[6] to $\boldsymbol{\pi}^{(t)}$:

$$\boldsymbol{\pi}^* = \arg\min_{\boldsymbol{\pi}\in\Delta^{K-1}} \sum_{t\in\mathcal{T}} \mathrm{KL}(\boldsymbol{\pi},\boldsymbol{\pi}^{(t)}). \quad (9)$$

See Appendix A.4 for the proof.

We call adjusted-LLD the theoretical reference method that updates domain weights at each step in $\mathcal{T}$ according to (6), using the sensitivity adjustment $(\boldsymbol{J}^\top\boldsymbol{J})^{-1}$. In contrast, we call iterative-LLD the method that updates domain weights at each step

[3] $\boldsymbol{J}$ may be viewed as a score-like matrix over domains, and $\boldsymbol{J}\boldsymbol{J}^\top$ corresponds to the empirical Fisher information matrix in parameter space.

[4] Since only the direction matters, we omit η.

[5] Hence, (5) can be interpreted equivalently as a one-step update of mirror descent (Nemirovskij and Yudin, 1983) with negative entropy as the distance-generating function (Appendix A.3), or as the multiplicative weights update method (Arora et al., 2012).

[6] When the KL divergence is replaced by the squared Euclidean distance, the solution to $\min_{\boldsymbol{\pi}\in\Delta^{K-1}} \sum_{t\in\mathcal{T}} \|\boldsymbol{\pi} - \boldsymbol{\pi}^{(t)}\|^2$ is $\boldsymbol{\pi}^* = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \boldsymbol{\pi}^{(t)}$. See Appendix G for the difference between arithmetic-mean aggregation and geometric-mean aggregation.

Algorithm 1 First-pass estimation of the fixed domain weights $\boldsymbol{\pi}^*$ for aggregated-LLD

Input: base model $p_{\boldsymbol{\theta}}$, target model p_{tgt}, evaluation corpus $\mathcal{D}$, domain-specific training corpora $\{\mathcal{C}_k\}_{k=1}^{K}$, update steps $\mathcal{T}$, total training steps t_{max}, temperature τ

Output: domain weights $\boldsymbol{\pi}^*$

Compute the domain-level log-likelihood vector $\boldsymbol{\ell}_{\text{tgt}}$ using (3)
for $t = 0, \ldots, t_{\text{max}}$ **do**
  if $t \in \mathcal{T}$ or $t = 0$ **then**
    Compute $\boldsymbol{\ell}(\boldsymbol{\theta})$ using (3)
    Estimate $\boldsymbol{\pi}^{(t)}$ using (7)
  else
    Set $\boldsymbol{\pi}^{(t)} \leftarrow \boldsymbol{\pi}^{(t-1)}$
  end if
  Sample a domain k according to the domain weights $\boldsymbol{\pi}^{(t)}$
  Update $\boldsymbol{\theta}$ using a batch sampled from $\mathcal{C}_k$
end for
Aggregate $\{\boldsymbol{\pi}^{(t)}\}_{t\in\mathcal{T}}$ using (8) to obtain $\boldsymbol{\pi}^*$

according to (7) without the sensitivity adjustment. We call aggregated-LLD the two-pass method that aggregates the iterative-LLD weights over the steps in $\mathcal{T}$ via (8) to estimate fixed domain weights $\boldsymbol{\pi}^*$, and then trains the base model using them. Algorithm 1 summarizes the first pass.

4 Experiments

In this section, we evaluate how well alignment with the target model can be achieved by training the base model using domain weights designed by the proposed method. We examine four aspects: (i) the validity of domain-weight estimation for target models with known domain weights, (ii) alignment with publicly available LLMs, (iii) comparison with knowledge distillation, and (iv) similarity in downstream task performance. Details of the experimental setup are provided in Appendix B.

4.1 Experimental setup

Datasets. Both the evaluation and training texts are taken from the Pile Uncopyrighted[7], which contains $K = 17$ domains[8]. For the evaluation corpus $\mathcal{D}$ used for computing log-likelihood vectors, we use the same 10,000 texts as in Oyama et al. (2025)[9]. These texts are chunked into 1024-byte segments, and the approximate KL divergence is measured on this text set in bits per byte using (2). Unless otherwise specified, we use "KL divergence" in the experimental results to refer to this approximation. By contrast, each domain corpus $\mathcal{C}_k$ used to train the base model is also derived from the Pile Uncopyrighted, but consists of original texts tokenized and stored separately by domain. During training, each batch is constructed from a domain sampled according to the domain weights. The total number of tokens per training step is $2^{19} \approx 500\text{k}$.

Base/target model settings. In our experiments, we use NanoGPT (Karpathy, 2022) with 124M parameters as the base model. To examine whether the proposed method remains effective even when the base model has already acquired general knowledge, we consider not only a randomly initialized NanoGPT but also a NanoGPT pretrained on FineWeb 100BT (Penedo et al., 2024). As target models, we use publicly available models such as Gemma-2B (Gemma Team et al., 2024) and CodeGemma-2B (CodeGemma Team et al., 2024), as well as NanoGPT pretrained on the Pile with controlled domain weights. In particular, we define CodeNanoGPT as a model trained for 100k steps on the Pile with the weights of GitHub, Stack Exchange, Ubuntu IRC, and Hacker News quadrupled relative to their original values, increasing the total proportion of code-related domains from 18.9% to 75.6%. See Appendix B for the hyperparameters used in these pretraining runs.

Computation of the domain-level log-likelihood vector. The domain-specific corpus $\mathcal{C}_k$ used for training and the evaluation text set $\mathcal{D}$ differ in how texts are segmented. The text sequences used to construct training batches have a fixed token length, whereas the evaluation texts vary in token length, as shown in Table 2 in Appendix B. To make log-likelihoods from the two settings comparable, we normalize the domain-level mean log-likelihoods in (3) by the mean token length of each domain[10].

Base model training. The base model is trained for 10k steps in both pretraining and continued pretraining. At each step, one domain is sampled

[7] https://huggingface.co/datasets/monology/pile-uncopyrighted

[8] Five copyrighted domains have been removed from the original 22 domains in the Pile.

[9] https://github.com/shimo-lab/modelmap

[10] Equivalently, this is the same as dividing the total log-likelihood by the total number of tokens for each domain.

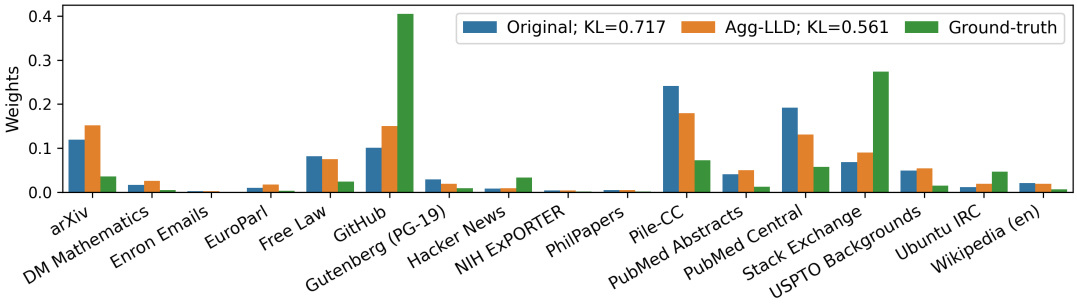


Figure 4: Comparison of domain weights estimated by aggregated-LLD and Pile-original weights against the ground-truth. The target model is CodeNanoGPT, pretrained with known domain weights, and the base model is a randomly initialized NanoGPT. The values in the legend denote the KL divergence from the ground-truth weights.

according to the domain weights, and the batch is constructed from that domain. The domain weights are updated iteratively at steps 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1k, 2k, …, 9k. This follows the Pythia setting (Biderman et al., 2023), in which checkpoints are saved densely during the warmup. For iterative-LLD and adjusted-LLD, training proceeds while iteratively estimating domain weights at these steps using the current base model. In contrast, for aggregated-LLD, we first aggregate the domain weights obtained by iterative-LLD to determine a fixed set of domain weights, and then train the base model using these fixed domain weights. The softmax temperature in (7) for domain-weight estimation is set to $\tau = 1$; ablations are in Appendix E. We use the original domain weights of the Pile as the reference weights $\boldsymbol{\pi}'$. The specific values are listed in Table 3 in Appendix B. We use AdamW (Loshchilov and Hutter, 2019) as the optimizer in all experiments.

4.2 Comparison with ground-truth weights

To examine how aggregated-LLD estimates domain weights, we consider a setting in which the target model is CodeNanoGPT, whose ground-truth weights are known, and the base model is a randomly initialized NanoGPT. Fig. 4 shows the domain weights $\boldsymbol{\pi}^*$ estimated by Algorithm 1.

The KL divergence from the ground-truth domain weights is 0.717 for Pile-original weights and 0.561 for the estimated weights. This indicates that the proposed method yields domain weights closer to the ground-truth than Pile-original weights. In particular, for code-related domains whose weights are increased in the ground-truth, such as GitHub and Stack Exchange, the estimated weights are broadly consistent with the ground-truth trend. Although the estimated weights do not exactly match the ground-truth, they capture the target model's overall domain weighting pattern.

4.3 Distributional alignment via estimated domain weights

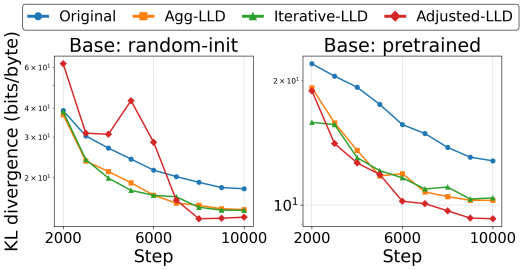
We evaluate how much the KL divergence between the base model and the target model is reduced when the base model is trained with the estimated domain weights. We compare Pile-original domain weights, adjusted-LLD, iterative-LLD, and aggregated-LLD.

We use Gemma-2B and CodeGemma-2B as target models, and randomly initialized or pretrained NanoGPT as base models. Figs. 5a and 5b show KL divergence over training steps, and Table 1 reports final-step KL divergence[11]. Overall, LLD-based domain weights achieve lower KL divergence than Pile-original domain weights in all settings.

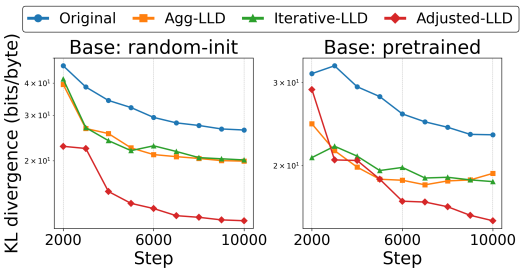
Among the three methods, adjusted-LLD serves as the theoretical reference method and indeed achieves the smallest KL divergence in all settings. Although iterative-LLD and aggregated-LLD do not reach adjusted-LLD, both outperform Pile-original weights in all settings. Moreover, the difference between iterative-LLD and aggregated-LLD is small. Therefore, aggregated-LLD has a practical advantage: it provides fixed domain weights set before training while maintaining performance close to iterative-LLD. We discuss the stability of the weights estimated by iterative-LLD in Appendix G.

Fig. 1 visualizes, using t-SNE (van der Maaten and Hinton, 2008), the training checkpoints of pretrained NanoGPT in the $\boldsymbol{q}$-coordinates obtained by double-centering text-level log-likelihood vectors on the evaluation corpus, following Oyama et al. (2025). By (2), the squared Euclidean distance

[11] For readability, the main plots omit the initial phase of training; the corresponding full curves over all training steps are shown in Appendix C.



(a) Target model: Gemma-2B.



(b) Target model: CodeGemma-2B.

Figure 5: KL divergence between each checkpoint and the target model from step 2k onward for random-init and pretrained NanoGPT trained with different domain weights. Results over all steps are provided in Fig. 9.

Target	Method	Base	
		random-init	pretrained
Gemma-2B	Original	17.9	12.8
	Aggregated-LLD	14.6	10.3
	Iterative-LLD	14.5	10.4
	Adjusted-LLD	13.5	9.26
CodeGemma-2B	Original	26.3	23.2
	Aggregated-LLD	19.9	19.3
	Iterative-LLD	20.1	18.5
	Adjusted-LLD	11.7	15.3

Table 1: Final-step KL divergence (bits/byte) to the target model for random-init/pretrained NanoGPT trained with different domain weights.

in this space approximates the KL divergence between models. The figure shows that training with the domain weights estimated by aggregated-LLD moves closer to the target model than training with Pile-original weights.

Robustness to the evaluation corpus. We also examined the robustness of aggregated-LLD to the choice of the evaluation corpus. Aggregated-LLD improved KL divergence on both another independently sampled Pile evaluation set and texts sampled from SlimPajama (Soboleva et al., 2023), suggesting that the improvement is not specific to the evaluation corpus used in the main experiments. Full results are provided in Appendix D.

Analysis of $(\boldsymbol{J}^\top \boldsymbol{J})^{-1}$. Fig. 6 shows the absolute values of the entries of the adjustment matrix

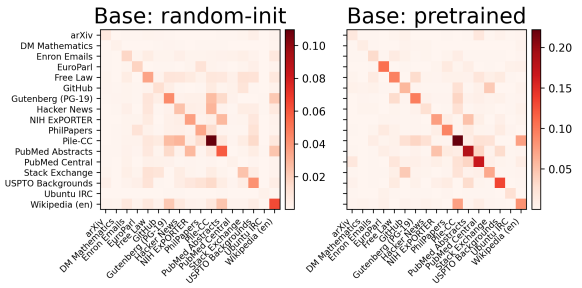


Figure 6: Heatmaps of the absolute values of the entries of the adjustment matrix $(\boldsymbol{J}^\top \boldsymbol{J})^{-1}$ for random-init and pretrained NanoGPT. Rows and columns correspond to domains in the Pile.

$(\boldsymbol{J}^\top \boldsymbol{J})^{-1}$ for randomly initialized and pretrained NanoGPT, computed on the evaluation corpus $\mathcal{D}$ at step $t = 0$. In both cases, the diagonal entries are relatively large, indicating that the adjustment is dominated by each domain's own component. This is consistent with the practical effectiveness of LLD without explicit sensitivity adjustment. For pretrained NanoGPT, however, several off-diagonal entries are also non-negligible, including those between GitHub and Stack Exchange and between Pile-CC and Wikipedia (en). Correcting for such inter-domain correlations may help explain the stronger performance of adjusted-LLD.

Effectiveness of adjusted-LLD. As shown in Table 1, adjusted-LLD performs best when its domain weights are updated stage-wise during training. However, when the adjusted-LLD weights are aggregated into a single fixed mixture, the resulting performance is not consistently better than aggregated-LLD. The final-step KL values are 16.6 and 15.9 for Gemma-2B, and 15.2 and 17.7 for CodeGemma-2B, with randomly initialized and pretrained base models, respectively. Thus, if the goal is to obtain a single fixed training recipe, aggregating the simpler LLD weights without the sensitivity adjustment is a practical choice.

4.4 Comparison with knowledge distillation

We compare aggregated-LLD with knowledge distillation. We use CodeNanoGPT as the target model so that the base model and the target model share the same tokenizer. For distillation, we use Pile-original weights and consider two objectives: KL divergence alone, and the sum of KL divergence and the cross-entropy (CE) loss. Although optimizing domain weights for distillation could yield further improvements (Liu et al., 2024), we adopt this simplest setting to make the comparison

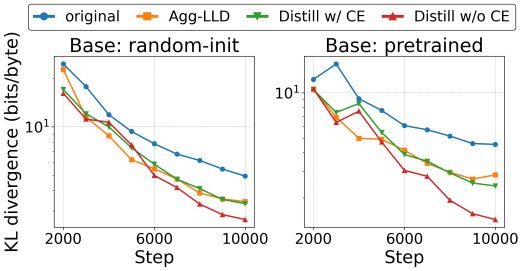
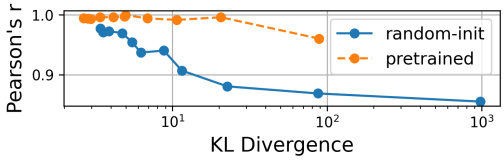


Figure 7: KL divergence between each checkpoint and the target model, CodeNanoGPT, from step 2k onward for random-init and pretrained NanoGPT trained with Pile-original domain weights, domain weights estimated by aggregated-LLD, or distillation. Results over all steps are provided in Fig. 10.

Model	ARC-Easy	HellaSwag	TruthfulQA	PIQA	LAMBADA
Target	0.426	0.283	0.259	0.618	0.284
random-init	0.358	0.267	0.272	0.559	0.113
pretrained	0.425	0.283	0.247	0.613	0.292

(a) Final-step accuracy on each task.



(b) Across checkpoints at different training steps, the relationship between the KL divergence to the target model and the Pearson correlation of the five-task performance profiles.

Figure 8: Downstream task evaluation with CodeNanoGPT as the target model and two base models, trained using aggregated-LLD.

with the proposed method clearer.

The results are shown in Fig. 7. When randomly initialized NanoGPT is used as the base model, the KL divergence from the target model at the final training step is 4.93, 3.42, 3.33, and 2.65 bits/byte for Pile-original, aggregated-LLD, distill with CE, and distill without CE, respectively. When pretrained NanoGPT is used as the base model, the corresponding values are 4.54, 2.84, 2.40, and 1.43 bits/byte. As expected, distillation without CE yields the lowest KL divergence, since it directly optimizes the KL-based objective. However, aggregated-LLD remains close to distillation with CE, even though it uses only pre-estimated domain weights and does not access teacher outputs during training. This indicates that the proposed method can recover much of the alignment achieved by distillation with CE and be a practical distillation-free alternative when teacher outputs are unavailable.

4.5 Similarity in task performance

We examine whether bringing a model closer to the target model in log-likelihood space using the proposed method also makes its downstream performance profile closer to that of the target model.

We use randomly initialized and FineWeb-pretrained NanoGPT as the base models and CodeNanoGPT as the target model. For evaluation, we use AI2 Reasoning Challenge (ARC) (Clark et al., 2018), HellaSwag (Zellers et al., 2019), TruthfulQA (Lin et al., 2022), PIQA (Bisk et al., 2020), and LAMBADA (Paperno et al., 2016), inspired by evaluations of small language models (Penedo et al., 2024). All evaluations are conducted using `lm-evaluation-harness` (Gao et al., 2024).

As shown in Fig. 8, the final checkpoint trained with aggregated-LLD exhibits task performance that generally corresponds to that of the target model across downstream tasks. Moreover, as the KL divergence to the target model decreases, the Pearson correlation between the performance vectors also tends to increase[12].

We also conduct a likelihood-based downstream analysis using BPB on correct completions, following the motivation that accuracy can be insensitive for small models (Heineman et al., 2026). The results show a similar tendency: the proposed method makes the downstream likelihood profiles closer to those of the target model, especially in the random-init setting. Details are provided in Appendix F.

5 Conclusion

We proposed a method for distributionally aligning a base model with a target model during pretraining or continued pretraining by controlling domain weights in the training data. Using a geometric formulation in log-likelihood space, we derived a theoretical criterion for domain weights that steer the update direction toward the target model and approximated it with fixed domain weights. Experiments with NanoGPT showed that the proposed method yields smaller KL divergence than Pile-original domain weights and brings the base model closer to the target model. For target models that share the same tokenizer, the proposed method provides a useful alternative to knowledge distillation. As KL divergence decreases, downstream task performance also tends to become closer to that of the target model.

[12]Note that, in the multiple-choice tasks other than LAMBADA, correlations can arise even at initialization because of the effect of chance rates.

Limitations

- Our experiments are conducted primarily using NanoGPT with 124M parameters. Whether the same behavior also holds for substantially larger models, such as those with several billion parameters or more, has not been examined due to computational resource constraints and remains future work.

- We primarily evaluate our method on specific pretrained models. Its generality to a broader range of architectures, as well as to models that have undergone instruction tuning or alignment through reinforcement learning, has not yet been sufficiently examined.

- This study assumes the Pile Uncopyrighted and thus relies on a large-scale corpus with explicitly defined domains. The proposed framework is not specific to the Pile, and could be applied to other corpora with predefined partitions, such as RedPajama (Weber et al., 2024), or WebOrganizer (Wettig et al., 2025). It also remains unverified whether the same optimization of domain weights is effective for other corpora or for datasets with ambiguous domain boundaries. In particular, the results may vary substantially depending on the granularity of the domain partitioning. Extending the method to finer granularities, such as sentence-level units, to pseudo-domains obtained by clustering in embedding space (Diao et al., 2026), or to settings in which a text belongs to multiple domains remains important future work.

- As described in Section 4.1 and Appendix B, the evaluation text set $\mathcal{D}$ is the same as that used by Oyama et al. (2025) and therefore consists of texts with variable token lengths. Since actual training is performed with a fixed token length, we normalize the log-likelihood by token length for each domain. Ideally, such normalization would be unnecessary if a fixed-token-length evaluation set were used from the outset. At the same time, allowing variable-length texts is itself natural when users may evaluate models on arbitrary text sets. The effect of this particular choice of $\mathcal{D}$ on the results has not been examined.

- For simplicity, the theory assumes SGD as the training algorithm, whereas the experiments use AdamW, which is widely adopted in practice. Although we empirically confirm the alignment effect in this study, the exact correspondence between the internal dynamics of AdamW and the geometric interpretation in log-likelihood space remains unclear.

- Adjusted-LLD uses the adjustment matrix $(\boldsymbol{J}^\top \boldsymbol{J})^{-1}$, but computing it requires $O(K^2P)$ time, making both the computational and memory costs substantial for large language models. For this reason, we treat the criterion based on $(\boldsymbol{J}^\top \boldsymbol{J})^{-1}$ as a theoretical reference method and primarily use its approximations, iterative-LLD and aggregated-LLD, in our experiments. Since what is actually required is the Gram matrix $\boldsymbol{J}^\top \boldsymbol{J}$ rather than the Jacobian itself, approximate computation via partial parameter sampling or stochastic estimation remains an important direction for future work.

- In (7), it is also possible to vary τ across training steps. In particular, since the likelihood is expected to fluctuate substantially in the early stage of training the base model, one possible strategy is to use a higher temperature at the beginning and gradually lower it as training proceeds. It is also conceivable to automatically scale the temperature using the variance of the domain-level log-likelihood differences across domains. In addition, in (8), when aggregating the mixture proportions estimated at each step, one could consider taking a weighted average rather than an unweighted one. Improving the method along these lines remains future work.

- Our method estimates domain weights for distributional alignment to a target model, not for directly optimizing domain-wise performance. Thus, the estimated weights should not be interpreted as predicting per-domain performance improvements. A natural extension is to optimize mixture weights with respect to an explicit objective over domain-wise validation losses or downstream task metrics, possibly using proxy-based mixture optimization frameworks such as RegMix (Liu et al., 2025).

Ethical Considerations

We note a potential misuse risk: because the first pass references a target model but the second pass

can be executed without further access to it, the role of the target model may become obscured. This method should not be used to conceal target-model-dependent training or to circumvent model licenses or usage restrictions. When applying the method, practitioners should disclose whether domain weights were estimated using a target model and ensure that such use complies with the target model's license and terms of use.

References

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. 2024. A survey on data selection for language models. *Trans. Mach. Learn. Res.*, 2024.

Koshiro Aoki, Masaru Isonuma, Yusuke Oda, Hirokazu Kiyomaru, Takashi Kodama, Chaoran Liu, Yohei Oseki, Yusuke Miyao, and Daisuke Kawahara. 2025. Learning-domain decomposition: Interpreting training dynamics via loss vectors.

Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 2397–2430. PMLR.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, and 8 others. 2024. Codegemma: Open code models based on gemma. *Preprint*, arXiv:2406.11409.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan SU, Markus Kliegl, ZIJIA CHEN, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Celine Lin, Jan Kautz, and Pavlo Molchanov. 2026. Nemotron-CLIMB: Clustering-based iterative data mixture bootstrapping for language model pre-training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Elvis Dohmatob, Mohammad Pezeshki, and Reyhane Askari Hemmat. 2025. Why less is more (sometimes): A theory of data curation. *CoRR*, abs/2511.03492.

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. 2021. The pile: An 800gb dataset of diverse text for language modeling. *CoRR*, abs/2101.00027.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, and 89 others. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

Yuto Harada, Yusuke Yamauchi, Yusuke Oda, Yohei Oseki, Yusuke Miyao, and Yu Takagi. 2025. Massive supervised fine-tuning experiments reveal how data, layer, and training factors shape LLM alignment quality. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 22360–22381. Association for Computational Linguistics.

Robert Hecht-Nielsen. 1990. On the algebraic structure of feedforward network weight spaces. In Rolf ECKMILLER, editor, *Advanced Neural Computers*, pages 129–135. North-Holland, Amsterdam.

David Heineman, Valentin Hofmann, Ian Magnusson, Yuling Gu, Noah A. Smith, Hannaneh Hajishirzi,

Kyle Lo, and Jesse Dodge. 2026. Signal and noise: A framework for reducing uncertainty in language model evaluation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

Andrej Karpathy. 2022. NanoGPT. https://github.com/karpathy/nanoGPT.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1317–1327. The Association for Computational Linguistics.

Ryo Kishino, Yusuke Takase, Momose Oyama, Hiroaki Yamagiwa, and Hidetoshi Shimodaira. 2026. Establishing a scale for kullback–leibler divergence in language models across various settings. *Preprint*, arXiv:2505.15353.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics.

Jiaheng Liu, Chenchen Zhang, Jinyang Guo, Yuanxing Zhang, Haoran Que, Ken Deng, Zhiqi Bai, Jie Liu, Ge Zhang, Jiakai Wang, Yanan Wu, Congnan Liu, Jiamang Wang, Lin Qu, Wenbo Su, and Bo Zheng. 2024. DDK: distilling domain knowledge for efficient large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. 2025. Regmix: Data mixture as regression for language model pre-training. In *The Thirteenth International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Robert C. Moore and William D. Lewis. 2010. Intelligent selection of language model training data. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden, Short Papers*, pages 220–224. The Association for Computer Linguistics.

Arkadij Semenovič Nemirovskij and David Borisovich Yudin. 1983. Problem complexity and method efficiency in optimization.

Momose Oyama, Hiroaki Yamagiwa, Yusuke Takase, and Hidetoshi Shimodaira. 2025. Mapping 1, 000+ language models via the log-likelihood vector. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 32983–33038. Association for Computational Linguistics.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin A. Raffel, Leandro von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. Redpajama: an open dataset for training large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. 2025. Organize the web: Constructing domains enhances pre-training data curation. In *Forty-second International Conference on Machine Learning*.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. 2023. Doremi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. LIMO: less is more for reasoning. *CoRR*, abs/2502.03387.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. LIMA: less is more for alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

A Mathematical Details

A.1 Proof of (4)

We show how the coordinates in log-likelihood space change under parameter updates by SGD:

$$\mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}[\boldsymbol{\ell}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})-\boldsymbol{\ell}(\boldsymbol{\theta})] \approx \eta \boldsymbol{J}(\boldsymbol{\theta})^\top \boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{\pi}.$$

First, the expected parameter update direction is approximated as follows:

$$\begin{aligned}
&\mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}[\Delta\boldsymbol{\theta}] \\
&\quad = \mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}\left[\eta \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x)\right] \\
&\quad = \sum_{k=1}^{K} \pi_k \mathbb{E}_{x\sim r_k}\left[\eta \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x)\right] \\
&\quad \approx \eta \sum_{k=1}^{K} \pi_k \nabla_{\boldsymbol{\theta}} \left(\frac{1}{|\mathcal{D}_k|} \sum_{x\in\mathcal{D}_k} \log p_{\boldsymbol{\theta}}(x) \right) \\
&\quad = \eta \boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{\pi}.
\end{aligned}$$

In the last line, we use the fact that the k-th column of the Jacobian $\boldsymbol{J}(\boldsymbol{\theta})$ is

$$\nabla_{\boldsymbol{\theta}} \left(\frac{1}{|\mathcal{D}_k|} \sum_{x\in\mathcal{D}_k} \log p_{\boldsymbol{\theta}}(x) \right).$$

The approximation above replaces the expectation over $x \sim r_k$ with the sample mean over the evaluation text set $\mathcal{D}_k$. This is justified under the assumption that $\mathcal{D}_k$ is sampled from r_k.

Next, we approximate the change in the domain-level log-likelihood vector induced by a small parameter update using the first-order Taylor expansion with respect to $\boldsymbol{\theta}$:

$$\boldsymbol{\ell}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta}) \approx \boldsymbol{\ell}(\boldsymbol{\theta}) + \boldsymbol{J}(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta}.$$

Therefore, we obtain

$$\begin{aligned}
\mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}[\boldsymbol{\ell}(\boldsymbol{\theta}+\Delta\boldsymbol{\theta})-\boldsymbol{\ell}(\boldsymbol{\theta})] &\approx \mathbb{E}_{x\sim r_{\boldsymbol{\pi}}}[\boldsymbol{J}(\boldsymbol{\theta})^\top \Delta\boldsymbol{\theta}] \\
&\approx \eta \boldsymbol{J}(\boldsymbol{\theta})^\top \boldsymbol{J}(\boldsymbol{\theta})\boldsymbol{\pi}.
\end{aligned}$$

A.2 Proof of (6)

We show that the solution to the optimization problem

$$\max_{\boldsymbol{\pi}\in\Delta^{K-1}} \boldsymbol{\pi}^\top \tilde{\boldsymbol{\pi}} - \tau\, \mathrm{KL}(\boldsymbol{\pi}, \boldsymbol{\pi}')$$

is given by

$$\pi_k = \frac{\pi'_k \exp(\tilde{\pi}_k/\tau)}{\sum_{k'=1}^{K} \pi'_{k'} \exp(\tilde{\pi}_{k'}/\tau)}.$$

Let λ be the Lagrange multiplier for the equality constraint $\sum_k \pi_k = 1$, and define

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\pi}, \lambda) = &\sum_{k=1}^{K} \pi_k \tilde{\pi}_k - \tau \sum_{k=1}^{K} \pi_k \log \frac{\pi_k}{\pi'_k} \\
&+ \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right).
\end{aligned}$$

Taking the partial derivative with respect to π_k gives

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \tilde{\pi}_k - \tau \left(\log \frac{\pi_k}{\pi'_k} + 1 \right) + \lambda.$$

Therefore, at the optimum, we have

$$\tilde{\pi}_k - \tau \left(\log \frac{\pi_k}{\pi'_k} + 1 \right) + \lambda = 0.$$

Rearranging this expression yields

$$\log \frac{\pi_k}{\pi'_k} = \frac{\tilde{\pi}_k + \lambda - \tau}{\tau}.$$

Hence,

$$\begin{aligned}
\pi_k &= \pi'_k \exp\left(\frac{\tilde{\pi}_k + \lambda - \tau}{\tau} \right) \\
&= \pi'_k \exp\left(\frac{\tilde{\pi}_k}{\tau} \right) \exp\left(\frac{\lambda - \tau}{\tau} \right) \\
&\propto \pi'_k \exp(\tilde{\pi}_k/\tau) > 0.
\end{aligned}$$

Normalizing over k, we obtain

$$\pi_k = \frac{\pi'_k \exp(\tilde{\pi}_k/\tau)}{\sum_{k'=1}^{K} \pi'_{k'} \exp(\tilde{\pi}_{k'}/\tau)}.$$

A.3 Relation to mirror descent

Let us define the distance-generating function as

$$h(\boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k \log \pi_k,$$

so the primal variable $\boldsymbol{\pi}$ is mapped to the dual variable $\boldsymbol{z} = \nabla h(\boldsymbol{\pi})$, whose components are

$$(\nabla h(\boldsymbol{\pi}))_k = \log \pi_k + 1.$$

Likewise, the reference domain weight vector $\boldsymbol{\pi}'$ is mapped to $\boldsymbol{z}' = \nabla h(\boldsymbol{\pi}')$, whose components are

$$z'_k = \log \pi'_k + 1.$$

Since the gradient of the linear term in the primal objective (5) is $\tilde{\boldsymbol{\pi}}$, one mirror-descent step in the dual space with step size $1/\tau$ is

$$\boldsymbol{z} = \boldsymbol{z}' + \frac{1}{\tau}\tilde{\boldsymbol{\pi}}.$$

Mapping this back to the primal space, we have $\boldsymbol{\pi} = \nabla h^*(\boldsymbol{z})$, and

$$\begin{aligned}\pi_k &\propto \exp(z_k - 1)\\ &= \exp\left(\log \pi'_k + \frac{\tilde{\pi}_k}{\tau}\right)\\ &= \pi'_k \exp(\tilde{\pi}_k/\tau).\end{aligned}$$

Therefore, the update in the primal space is

$$\pi_k = \frac{\pi'_k \exp(\tilde{\pi}_k/\tau)}{\sum_{k'=1}^{K} \pi'_{k'} \exp(\tilde{\pi}_{k'}/\tau)}.$$

Thus, our update can be interpreted as taking one mirror-descent step from $\boldsymbol{\pi}'$.

A.4 Proof of (9)

We show that the solution to the optimization problem

$$\min_{\boldsymbol{\pi} \in \Delta^{K-1}} \sum_{t \in \mathcal{T}} \mathrm{KL}(\boldsymbol{\pi}, \boldsymbol{\pi}^{(t)})$$

is given by

$$\pi^*_k \propto \left(\prod_{t \in \mathcal{T}} \pi_k^{(t)}\right)^{1/|\mathcal{T}|}.$$

Let λ be the Lagrange multiplier for the constraint $\sum_k \pi_k = 1$, and define

$$\begin{aligned}\mathcal{L}(\boldsymbol{\pi}, \lambda) &= \sum_{t \in \mathcal{T}} \mathrm{KL}(\boldsymbol{\pi}, \boldsymbol{\pi}^{(t)}) + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right)\\ &= \sum_{k=1}^{K}\left(|\mathcal{T}|\pi_k \log \pi_k - \pi_k \sum_{t \in \mathcal{T}} \log \pi_k^{(t)}\right)\\ &\quad + \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right).\end{aligned}$$

Taking the partial derivative with respect to π_k, we obtain

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = |\mathcal{T}|(\log \pi_k + 1) - \sum_{t \in \mathcal{T}} \log \pi_k^{(t)} + \lambda.$$

Therefore, at the optimum, we have

$$|\mathcal{T}|(\log \pi_k + 1) - \sum_{t \in \mathcal{T}} \log \pi_k^{(t)} + \lambda = 0.$$

Rearranging this, we obtain

$$\log \pi_k = \frac{1}{|\mathcal{T}|}\sum_{t \in \mathcal{T}} \log \pi_k^{(t)} - \frac{\lambda + |\mathcal{T}|}{|\mathcal{T}|}.$$

That is,

$$\pi^*_k \propto \exp\left(\frac{1}{|\mathcal{T}|}\sum_{t \in \mathcal{T}} \log \pi_k^{(t)}\right).$$

Normalizing over k, we obtain

$$\pi^*_k = \frac{\left(\prod_{t \in \mathcal{T}} \pi_k^{(t)}\right)^{1/|\mathcal{T}|}}{\sum_{j=1}^{K}\left(\prod_{t \in \mathcal{T}} \pi_j^{(t)}\right)^{1/|\mathcal{T}|}}.$$

B Details of Experimental Setup

B.1 Datasets

For the domain names in the Pile, we standardized several labels for readability and consistency. Specifically, ArXiv was changed to arXiv, FreeLaw to Free Law, Github to GitHub, HackerNews to Hacker News, NIH ExPorter to NIH ExPORTER, and StackExchange to Stack Exchange. These changes affect only the presentation of the labels and do not alter the underlying categorization.

We use the evaluation corpus $\mathcal{D}$ from Oyama et al. (2025) without modification. This corpus consists of texts sampled from the Pile, and Table 2

Domain	# Texts	Mean tokens
arXiv	1172	365
DM Mathematics	151	481
Enron Emails	22	242
EuroParl	83	416
Free Law	837	262
GitHub	925	429
Gutenberg (PG-19)	251	270
Hacker News	67	248
NIH ExPORTER	41	179
PhilPapers	51	256
Pile-CC	2353	220
PubMed Abstracts	464	179
PubMed Central	1763	317
Stack Exchange	712	311
USPTO Backgrounds	487	199
Ubuntu IRC	54	406
Wikipedia (en)	567	217

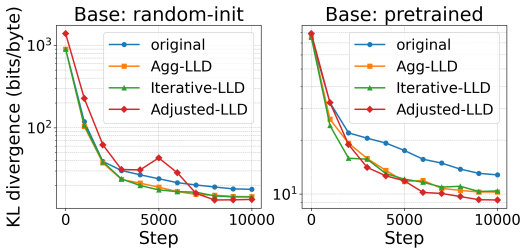
Table 2: Number of texts and mean token length for each domain in the evaluation corpus $\mathcal{D}$.

Domain	Original	Code	PubMed
arXiv	0.119	0.0356	0.0107
DM Mathematics	0.0165	0.00493	0.00148
Enron Emails	0.00187	0.000557	0.000167
EuroParl	0.00973	0.0029	0.000869
Free Law	0.0816	0.0243	0.00729
GitHub	0.101	**0.405**	0.00904
Gutenberg (PG-19)	0.0289	0.00863	0.00258
Hacker News	0.00826	**0.0331**	0.000738
NIH ExPORTER	0.004	0.00119	0.000357
PhilPapers	0.00507	0.00151	0.000452
Pile-CC	0.241	0.072	0.0216
PubMed Abstracts	0.0409	0.0122	**0.164**
PubMed Central	0.192	0.0573	**0.768**
Stack Exchange	0.0684	**0.274**	0.00611
USPTO Backgrounds	0.0487	0.0145	0.00435
Ubuntu IRC	0.0117	**0.0469**	0.00105
Wikipedia (en)	0.0204	0.00608	0.00182

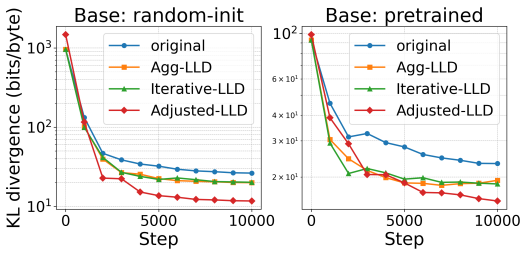
Table 3: Domain weights in the Pile Uncopyrighted. In the Code setting, the weights for GitHub, Stack Exchange, Ubuntu IRC, and Hacker News are quadrupled; in the PubMed setting, the weights for PubMed Abstracts and PubMed Central are quadrupled. The corresponding values are shown in bold.

reports the number of texts and the mean token length for each domain.

In addition, the model used in Fig. 2 in Section 2 and CodeNanoGPT used in Section 4 were both trained from NanoGPT with manually adjusted domain weights on the Pile, in which the weights of particular domains were increased. The corresponding domain weights are shown in Table 3. Note that "Original" is obtained by taking the 17 uncopyrighted domain weights from the 22 domain weights reported in the original Pile paper (Gao et al., 2021) and renormalizing them.



(a) Target model: Gemma-2B.



(b) Target model: CodeGemma-2B.

Figure 9: Full KL-divergence trajectories corresponding to Fig. 5, shown over all training steps. The target models are Gemma-2B and CodeGemma-2B, and the base models are random-init or pretrained NanoGPT.

B.2 Training environment and hyperparameters

The training environment and hyperparameters for the base model are as follows. We use the GPT-2 tokenizer. The NanoGPT (124M) used as the base model has a vocabulary size of 50,304, 12 layers, 12 attention heads, an embedding dimension of 768, and a maximum sequence length of 1,024. The total number of tokens per step is 524,288, and the micro-batch size is 64. We use AdamW for optimization, with weight decay 0.1, betas $(0.9, 0.95)$, and epsilon 1×10^{-8}. For the learning rate schedule, we adopt cosine decay, with the first 10% of the total training steps used for warmup, and decay the learning rate from 6×10^{-4} to 6×10^{-5}. The total number of training steps is 10k for base model training and knowledge distillation, 100k for CodeNanoGPT, and approximately 190k for pretraining on FineWeb 100BT. Training is conducted using four NVIDIA RTX A6000 Ada GPUs. Under these settings, training for 10k steps takes approximately five hours.

C KL Divergence Curves over All Training Steps

This appendix provides the full KL-divergence curves corresponding to the truncated plots in the main text. Whereas Figs. 5 and 7 focus on checkpoints from step 2k onward for readability, Figs. 9

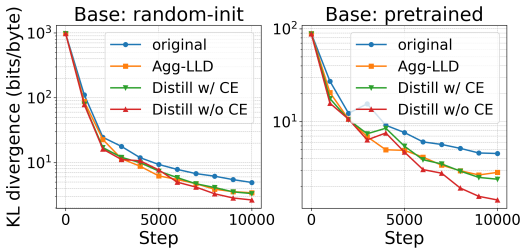


Figure 10: Full KL-divergence trajectories corresponding to Fig. 7, shown over all training steps. The target model is CodeNanoGPT, and the curves compare Pile-original weighting, aggregated-LLD, and distillation-based training for random-init and pretrained NanoGPT.

Target	Method	Base	
		random-init	pretrained
Gemma-2B	Original	18.5	13.0
	Aggregated-LLD	15.4	10.8
CodeGemma-2B	Original	26.5	23.4
	Aggregated-LLD	20.1	19.0

Table 4: Final-step KL divergence (bits/byte) under the same target/base-model settings as Table 1, evaluated on another 10,000-text sample from the Pile.

and 10 show the same experiments over all training steps, including the initial phase.

D Robustness to Evaluation Corpus

In the experiment in Section 4.3, the same evaluation corpus is used for estimating the domain weights and computing the approximate KL divergence between the trained base model and the target model. To examine whether the observed improvements depend on this particular evaluation corpus, we consider two held-out evaluation corpora.

The first is another set of 10,000 texts sampled from the Pile, which follows the same source distribution as the original evaluation corpus but contains different texts. As shown in Table 4, aggregated-LLD achieves lower approximate KL divergence than the Pile-original baseline in all target/base-model settings.

Second, we evaluate the final checkpoints on 10,000 texts sampled from SlimPajama[13]. Unlike the first evaluation, this corpus is not sampled from the Pile, and thus provides a stronger test of robustness to the choice of evaluation corpus. As shown in Table 5, aggregated-LLD again achieves lower approximate KL divergence than the Pile-original baseline in all settings.

[13] https://huggingface.co/datasets/DKYoon/SlimPajama-6B

Target	Method	Base	
		random-init	pretrained
Gemma-2B	Original	15.8	11.3
	Aggregated-LLD	13.9	9.6
CodeGemma-2B	Original	10.8	9.9
	Aggregated-LLD	8.4	7.4

Table 5: Final-step KL divergence (bits/byte) under the same target/base-model settings as Table 1, evaluated on 10,000 texts sampled from SlimPajama.

Base	$\tau = 0.1$	$\tau = 1.0$	$\tau = 10$	$\tau = \infty^*$
Random-init	7.31	**3.42**	4.68	4.93
Pretrained	5.83	**2.84**	3.98	4.54

* $\tau = \infty$ corresponds to the Pile-original weights.

Table 6: KL divergence (bits/byte) to the target model, CodeNanoGPT, under different values of τ.

E Ablation on Temperature

The temperature τ in (7) controls the sharpness of the estimated domain weights. In this appendix, we examine the effect of $\tau \in \{0.1, 1.0, 10, \infty\}$ using randomly initialized or pretrained NanoGPT as the base model and CodeNanoGPT as the target model, where $\tau = \infty$ corresponds to Pile-original weights. As shown in Table 6, when trained with aggregated-LLD, the KL divergence at the final training step from the target model is minimized at $\tau = 1.0$ for both randomly initialized and pretrained NanoGPT.

A low softmax temperature concentrates the domain weights on a few domains, whereas a high temperature makes them closer to Pile-original weights. The difference in mean log-likelihood per token between randomly initialized NanoGPT and CodeNanoGPT ranged from 7.3 (Pile-CC) to 9.1 (DM Mathematics), giving a spread of 1.8. Choosing a temperature on the same scale as this range helps avoid extreme outcomes.

F Likelihood-Based Downstream Evaluation

Accuracy-based downstream evaluation can be noisy or insensitive when models are small and training budgets are limited. Heineman et al. (2026) argue that likelihood-based metrics such as perplexity or BPB can provide a higher signal-to-noise diagnostic for small-scale model development, and compute BPB on correct continuations by normalizing the negative log-likelihood of the correct answer by its number of UTF-8 bytes. Following this motivation, we evaluate downstream likelihood similarity using BPB on correct completions.

For each example, we compute the BPB of the

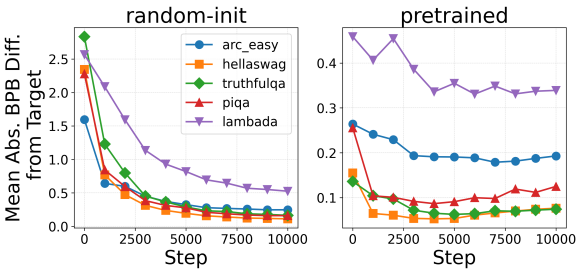


Figure 11: Likelihood-based downstream similarity to the target model. For each task, we compute the BPB of the correct completion and report the mean absolute BPB difference between the trained base model and the target model. Lower values indicate that the trained model assigns likelihoods more similarly to the target model. The left and right panels show randomly initialized and pretrained NanoGPT, respectively.

correct completion by normalizing its negative log-likelihood by the number of bytes in the completion. We then report the mean absolute BPB difference between the base and target model. Lower values indicate that the trained model assigns likelihoods to correct completions more similarly to the target model.

For multiple-choice tasks, BPB is computed only on the answer completion, not on the full prompt. For ARC-Easy, HellaSwag, and PIQA, we use the gold option specified by the task label. For LAMBADA, we treat the target word as the completion. For TruthfulQA, which may contain multiple true answers in the MC2 setting, we compute BPB for each true answer and average over the true answers.

As shown in Figure 11, the mean absolute BPB difference from the target model decreases clearly across most tasks in the randomly initialized setting. This suggests that the proposed mixture also moves the model closer to the target in downstream likelihood profiles, beyond the evaluation corpus used for estimating approximate KL divergence. In the pretrained setting, the initial differences are already much smaller, and the trends are more modest and task-dependent.

G Aggregation of Domain Weights

G.1 Differences across aggregation methods

In the main text, we obtained $\boldsymbol{\pi}^*$ by taking the geometric mean of $\{\boldsymbol{\pi}^{(t)}\}_{t\in\mathcal{T}}$, which are estimated by iterative-LLD, as in (8). This aggregation method is based on the optimization problem in (9). In this appendix, we consider aggregation by the simple

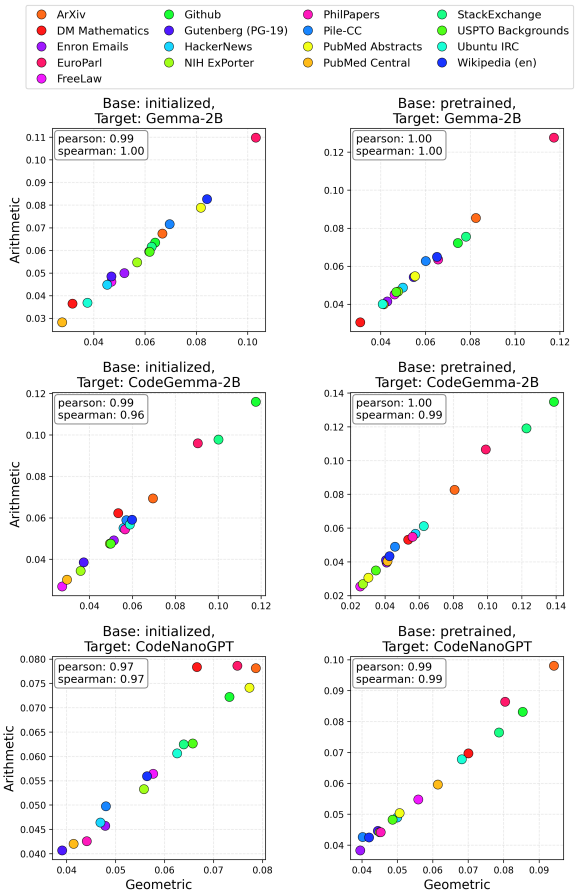


Figure 12: Comparison between the domain weights obtained by geometric-mean aggregation (x-axis) and those obtained by arithmetic-mean aggregation (y-axis). Each point corresponds to a domain in the Pile. Scatter plots are shown for each combination of base model (random-init/pretrained) and target model (Gemma-2B, CodeGemma-2B, and CodeNanoGPT).

arithmetic mean,

$$\bar{\pi}_k = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \pi_k^{(t)},$$

and analyze how the resulting domain weights differ depending on the choice of aggregation method.

The target models are the three models considered in the main text: Gemma-2B, CodeGemma-2B, and CodeNanoGPT. The base models are also the two models considered in the main text: random-init NanoGPT and pretrained NanoGPT. Fig. 12 shows, for each base–target model pair, scatter plots comparing the geometric and arithmetic means of $\{\boldsymbol{\pi}^{(t)}\}_{t\in\mathcal{T}}$ obtained by iterative-LLD. A high correlation is observed between the two methods. This indicates that, regardless of whether geometric-mean aggregation or arithmetic-mean aggregation is used, the relative importance and ranking of the domains are almost unchanged.

Therefore, although the geometric-mean aggregation used in this study is theoretically natural from the viewpoint of minimizing the KL divergence in (9), it is also empirically confirmed to be a stable aggregation method that yields domain weights consistent with those obtained by arithmetic-mean aggregation[14].

[14]When the variation of $\{\pi_k^{(t)}\}_{t\in\mathcal{T}}$ is small for each k, a Taylor expansion of $\log \pi_k^{(t)}$ around its mean shows that the first-order term vanishes when averaged. This suggests that the geometric mean yields a value close to the arithmetic mean.