Learning What Matters: Scalable Data Mixing for LLMs

Francesco Tonin
francesco.tonin@epfl.ch

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

ESAT-STADIUS Seminar, Leuven
February 12, 2026

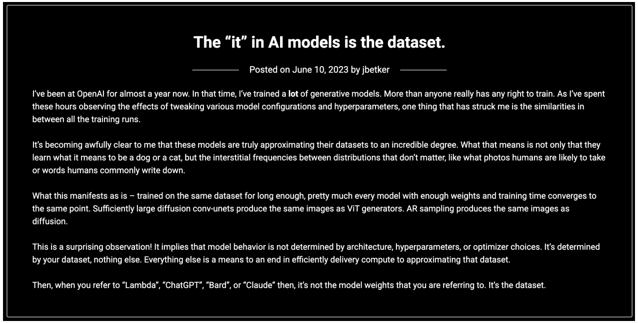
Outline

- What is data mixing?
- Taxonomy
- DoReMi, DoGE, RegMix methods
- The Chameleon framework
- Empirical analysis
- Conclusion

Motivation: Data as a resource

The world's most valuable resource is no longer oil, but data

The data economy demands a new approach to antitrust rules



The "it" in AI models is the dataset.

Posted on June 10, 2023 by jbetker

I've been at OpenAI for almost a year now. In that time, I've trained a **lot** of generative models. More than anyone really has any right to train. As I've spent these hours observing the effects of tweaking various model configurations and hyperparameters, one thing that has struck me is the similarities in between all the training runs.

It's becoming awfully clear to me that these models are truly approximating their datasets to an incredible degree. What that means is not only that they learn what it means to be a dog or a cat, but the interstitial frequencies between distributions that don't matter, like what photos humans are likely to take or words humans commonly write down.

What this manifests as is – trained on the same dataset for long enough, pretty much every model with enough weights and training time converges to the same point. Sufficiently large diffusion conv-unets produce the same images as ViT generators. AR sampling produces the same images as diffusion.

This is a surprising observation! It implies that model behavior is not determined by architecture, hyperparameters, or optimizer choices. It's determined by your dataset, nothing else. Everything else is a means to an end in efficiently delivery compute to approximating that dataset.

Then, when you refer to "Lambda", "ChatGPT", "Bard", or "Claude" then, it's not the model weights that you are referring to. It's the dataset.

Figure: (Left) Economist 2017 article on the importance of data. (Right) The relative importance of data within the model.

- Architectures determine
  - inference resources
  - inference capabilities
- Algorithms determine
  - training resources
  - generalization
- Data determines
  - generalization
  - training resources

Data processing pipeline for LLM pretraining

∘ Modern LLMs (e.g., ChatGPT, Gemini, Claude, DeepSeek,...) employ the following general data pipeline:

1. Data cleaning
   - Deduplication
   - Privacy and copyright
   - Quality filtering
   - Data age
2. Data selection
   - Data masking
   - Document packing
   - Data ordering
   - Synthetic data
3. Data mixing
   - Find optimal domain mixtures
   - Target evaluations
   - Online mixing approaches
   - Offline mixing approaches

Data processing pipeline for LLM pretraining

◦ Modern LLMs (e.g., ChatGPT, Gemini, Claude, DeepSeek,...) employ the following general data pipeline:

1. Data cleaning
   - Deduplication
   - Privacy and copyright
   - Quality filtering
   - Data age
2. Data selection
   - Data masking
   - Document packing
   - Data ordering
   - Synthetic data
3. Data mixing
   - Find optimal domain mixtures
   - Target evaluations
   - Online mixing approaches
   - Offline mixing approaches

Motivating example: Data mixing in practice

- Large-scale pretraining datasets contain many domains (e.g., Wikipedia, Books, etc.) [11, 26].

Data Source	Train Tokens	Validation Tokens	Train (% of Total)
CommonCrawl	200.82B	214.72M	66.99
C4	44.98B	48.06M	15.00
GitHub	13.51B	14.42M	4.51
Books	13.49B	14.39M	4.50
Wikipedia	13.50B	14.41M	4.50
ArXiv	7.49B	8.01M	2.50
StackExchange	5.99B	6.41M	2.00
Total	299.78B	320.42M	100.00

Table: SlimPajama dataset [26].

- Denote k domains for pretraining as $D^{\text{train}} = \{D_1, D_2, \ldots, D_k\}$.
- The sampling distribution is $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_k) \in \triangle^k$, the probability simplex over k domains.
- *How to set $\boldsymbol{\alpha}$ to improve downstream performance?*

What does data mixture optimize?

- Training a language model involves different loss objectives at different stages.

Example (Examples of relevant losses)

- $\mathcal{L}_{\text{train}}$: *Next-token prediction loss* on mixed data sources (e.g., CommonCrawl, Wikipedia).
  - $\mathcal{L}_{\mathrm{train}} = \sum_{i=1}^{k} \alpha_i f(\mathbf{x}; D_i)$ is the weighted cross-entropy loss over the training domains.
- $\mathcal{L}_{\text{val}}$: *Validation perplexity* on held-out set.
  - $\mathcal{L}_{\mathrm{val}} = \sum_{i=1}^{k} f(\mathbf{x}; D_i^{\text{val}})$ is the validation loss on the training domains.
- $\mathcal{L}_{\text{task}}$: *Downstream loss*, e.g., performance on reasoning benchmarks like ARC or LogiQA.
  - $\mathcal{L}_{\mathrm{task}} = f(\mathbf{x}; D^{\text{target}})$ is the loss on a set of interest.

- These losses guide us in choosing data mixture weights $\boldsymbol{\alpha}$.

Data mixing question

How should we sample data from diverse domains to minimize losses that matter?

Illustrative example: A toy data mixture problem

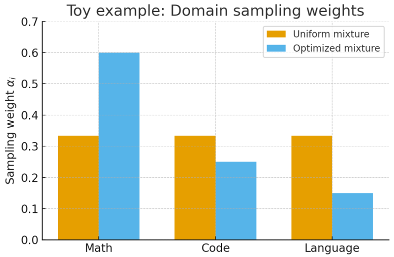
- Suppose we train an LLM on:
  - D_1: Math domain (e.g., problems and proofs)
  - D_2: Code domain (e.g., Python, C++)
  - D_3: Natural language (e.g., Wikipedia, Books)

- We want to optimize model performance on a downstream reasoning benchmark (e.g., LogiQA or ARC).

Question: What is the optimal sampling distribution $(\alpha_1, \alpha_2, \alpha_3)$?

- Uniform might underrepresent math.
- Manual heuristics might overfit and do not scale.

We need a principled optimization method!



Data mixing optimization

∘ Objective: Find optimal data composition over k sources minimizing loss of interest.

Data mixing

The data mixing problem can be written as a bilevel optimization [9]:

$$\min_{\boldsymbol{\alpha} \in \Delta^k} \mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha})) \quad \text{subject to} \quad \mathbf{x}^\star \in \arg\min_{\mathbf{x}} \mathcal{L}_{\text{train}}(\mathbf{x}; \boldsymbol{\alpha}), \tag{1}$$

where $\mathcal{L}_{\text{train}}$ is the next token prediction loss and $\mathcal{L}_{\text{target}}$ is the target loss of interest ($\mathcal{L}_{\text{val}}$ or $\mathcal{L}_{\text{task}}$).

Question

Note that $\mathbf{x}^\star(\boldsymbol{\alpha})$ is the model trained on the dataset re-weighted by $\boldsymbol{\alpha}$. How can we solve this bilevel problem?

Bilevel optimization

◦ We first recall the general bilevel problem:

Bilevel optimization problem [34]

The canonical form of bilevel optimization is given by:

$$\min_{\mathbf{x}_1 \in \mathcal{X}_1} \; f_1(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1)) \quad \text{s.t.} \quad \mathbf{x}_2^*(\mathbf{x}_1) \in \arg\min_{\mathbf{x}_2 \in \mathcal{X}_2} f_2(\mathbf{x}_1, \mathbf{x}_2), \tag{2}$$

where f_1, f_2 are smooth functions.

- $\mathbf{x}_1$ is the outer variable with outer objective f_1.
- $\mathbf{x}_2$ is the inner variable with inner objective f_2.

Example (Data mixing)

The above template captures data mixing with:

- Inner objective: $f_2(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{L}_{\text{train}}(\mathbf{x}; \boldsymbol{\alpha})$.
- Outer objective: $f_1(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1)) = \mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha}))$.

Remarks:

◦ Optimization of f_2 (model training) wrt $\mathbf{x}_2$ depends on $\mathbf{x}_1$ (data mixing).

◦ Hence, (2) is intrinsically hard to solve.

The implicit function approach

◦ Consider the following simplifications:

$$\min_{\mathbf{x}_1 \in \mathcal{X}_1} h(\mathbf{x}_1) \iff \min_{\mathbf{x}_1 \in \mathcal{X}_1} h(\mathbf{x}_1) := f_1(\mathbf{x}_1, \mathbf{x}_2^\star(\mathbf{x}_1)) \quad \text{s.t. } \mathbf{x}_2^\star(\mathbf{x}_1) = \arg\min_{\mathbf{x}_2 \in \mathbb{R}^{p_2}} f_2(\mathbf{x}_1, \mathbf{x}_2),$$

with $f_2(\mathbf{x}_1, \mathbf{x}_2)$ strongly convex in $\mathbf{x}_2$ and unconstrained.

The implicit function approach

∘ Consider the following simplifications:

$$\min_{\mathbf{x}_1 \in \mathcal{X}_1} h(\mathbf{x}_1) \iff \min_{\mathbf{x}_1 \in \mathcal{X}_1} h(\mathbf{x}_1) := f_1(\mathbf{x}_1, \mathbf{x}_2^\star(\mathbf{x}_1)) \quad \text{s.t. } \mathbf{x}_2^\star(\mathbf{x}_1) = \arg\min_{\mathbf{x}_2 \in \mathbb{R}^{p_2}} f_2(\mathbf{x}_1, \mathbf{x}_2),$$

with $f_2(\mathbf{x}_1, \mathbf{x}_2)$ strongly convex in $\mathbf{x}_2$ and unconstrained.

∘ Let's calculate the gradient for $h(\mathbf{x}_1)$:

$$\nabla h(\mathbf{x}_1) = \nabla_{\mathbf{x}_1} f_1(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1)) + \left(\frac{d\mathbf{x}_2^*}{d\mathbf{x}_1}\right)^\top \nabla_{\mathbf{x}_2} f_1(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1)).$$

Implicit function theorem gives:

$$\frac{d\mathbf{x}_2^*}{d\mathbf{x}_1} = -\left(\nabla^2_{\mathbf{x}_2\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1))\right)^{-1} \nabla^2_{\mathbf{x}_1\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2^*(\mathbf{x}_1)).$$

We obtain:

$$\nabla h(\mathbf{x}_1) = \nabla_{\mathbf{x}_1} f_1(\mathbf{x}_1, \mathbf{x}_2^*) - \nabla^2_{\mathbf{x}_1\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2^*)\left(\nabla^2_{\mathbf{x}_2\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2^*)\right)^{-1} \nabla_{\mathbf{x}_2} f_1(\mathbf{x}_1, \mathbf{x}_2^*).$$

The hyper-gradient

- The gradient requires $\mathbf{x}_2^\star(\mathbf{x}_1)$, i.e., the lower-level problem solved to global min $\Rightarrow$ not possible in most cases.
- We assume access to a surrogate for the unknown original gradient of f_1 at any $\mathbf{x}_1$.

The hyper-gradient

Consider the stationary conditions of the bilevel problem as finding $(\mathbf{x}_1^\star, \mathbf{x}_2^\star)$ such that

$$\bar{\nabla} f_1(\mathbf{x}_1, \mathbf{x}_2) = 0, \qquad \nabla_{\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2) = 0,$$

where the gradient approximation of f_1 is defined as

$$\bar{\nabla} f_1(\mathbf{x}_1, \mathbf{x}_2) = \nabla_{\mathbf{x}_1} f_1(\mathbf{x}_1, \mathbf{x}_2) - \nabla^2_{\mathbf{x}_1\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2)\left[\nabla^2_{\mathbf{x}_2\mathbf{x}_2} f_2(\mathbf{x}_1, \mathbf{x}_2)\right]^{-1} \nabla_{\mathbf{x}_2} f_1(\mathbf{x}_1, \mathbf{x}_2).$$

We call $\bar{\nabla} f_1(\mathbf{x}_1, \mathbf{x}_2)$ the *hyper-gradient*.

The bilevel approximation method

Algorithm 1: The bilevel approximation method

```
Input: $\mathbf{x}_1^{(0)} \in \mathbb{R}^{p1}$, $\mathbf{x}_2^{(0)} \in \mathbb{R}^{p2}$, step sizes $\{a^{(m)}\}, \{b^{(t)}\}$, inner iterations $\{t^{(m)}\}$
for $m = 0, 1, 2, \ldots$ do
    // Inner loop: approximate $\mathbf{x}_2^\star(\mathbf{x}_1^{(m)})$
    for $t = 0, 1, \ldots, t^{(m)} - 1$ do
        $\mathbf{x}_2^{(t+1)} = \mathbf{x}_2^{(t)} - b^{(t)} \nabla_{\mathbf{x}_2} f_2(\mathbf{x}_1^{(m)}, \mathbf{x}_2^{(t)})$
    $\bar{\mathbf{x}}_2^{(m)} = \mathbf{x}_2^{(t^{(m)})}$
    // Outer update using approximate hypergradient
    $\mathbf{x}_1^{(m+1)} = \arg\min_{\mathbf{u} \in \mathcal{X}_1} \left\{ \langle \bar{\nabla} f_1(\mathbf{x}_1^{(m)}, \bar{\mathbf{x}}_2^{(m)}), \mathbf{u} \rangle + \frac{1}{2a^{(m)}} \|\mathbf{u} - \mathbf{x}_1^{(m)}\|^2 \right\}$
Output: $\mathbf{x}_1^{(m+1)}$
```

Remarks:

- [12] derives the convergence rate $f_1(\mathbf{x}_1^{(T)}, \mathbf{x}_2^*(\mathbf{x}_1^{(T)})) - f_1(\mathbf{x}_1^*, \mathbf{x}_2^*(\mathbf{x}_1^*)) = \mathcal{O}(1/T)$.
- More advanced algorithms have been developed [34, 3, 18, 23].
- **Key challenge**: Computing second-order derivatives and Hessian inverse is challenging in LLMs.

Data mixing with direct modelling

- Find a mixing function V mapping domain weights to target loss:

$$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}} \mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha})) \approx V(\boldsymbol{\alpha})$$

- Goal: estimate the mixing function V.
  1. Sample mixing weights $\boldsymbol{\alpha}_j$, $j = 1, \ldots, n$, where n is the sampling budget
  2. Train n models with each $\boldsymbol{\alpha}_j$ and evaluate their validation loss $\mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha}_j))$
  3. Fit $V(\boldsymbol{\alpha})$ on $\{(\boldsymbol{\alpha}_j, \mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha}_j)))\}_{j=1}^n$ (e.g., linear regression)

- **Problem**: retraining costs at large model and data scales can be expensive.

Proxy-based algorithms

- Key idea: train smaller proxy models to approximate target performance.
- Fit a mixing function V through the proxy models.
- Train large model with $\alpha^\star$ optimized on proxies.

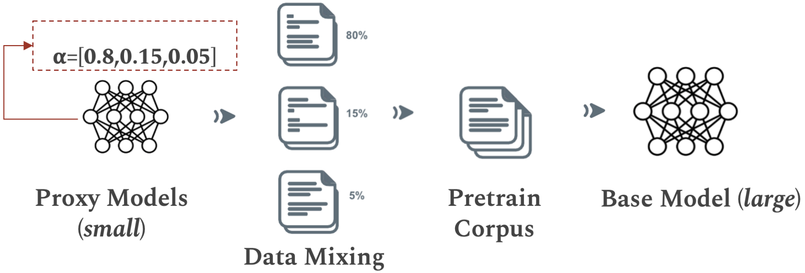


Figure: Proxy-based algorithms' pipeline [9].

What is data mixture *transfer*?

◦ Goal: use mixture $\boldsymbol{\alpha}$ found at a small proxy scale to train a larger model.

Setup

Let s denote the model scale and $\boldsymbol{\alpha} \in \Delta^k$ a domain mixture.

$$\mathbf{x}^\star(\boldsymbol{\alpha}; s) \in \arg\min_{\mathbf{x}} \mathcal{L}_{\text{train}}(\mathbf{x}; \boldsymbol{\alpha}, s), \quad V_s(\boldsymbol{\alpha}) := \mathcal{L}_{\text{target}}\Big(\mathbf{x}^\star(\boldsymbol{\alpha}; s)\Big), \quad \boldsymbol{\alpha}^\star(s) \in \arg\min_{\boldsymbol{\alpha}\in\Delta^k} V_s(\boldsymbol{\alpha}).$$

We ask: how does $\boldsymbol{\alpha}^\star(s)$ change with s?

Two notions of transfer

Strong transfer: $\boldsymbol{\alpha}^\star(s_{\text{small}}) \in \arg\min_{\boldsymbol{\alpha}} V_{s_{\text{large}}}(\boldsymbol{\alpha})$

Effective transfer: $V_{s_{\text{large}}}\Big(\boldsymbol{\alpha}^\star(s_{\text{small}})\Big) \le \min_{\boldsymbol{\alpha}} V_{s_{\text{large}}}(\boldsymbol{\alpha}) + \varepsilon,$ for a small tolerance $\varepsilon > 0$.

Remarks:

◦ In general, $\boldsymbol{\alpha}^\star(s_1) \neq \boldsymbol{\alpha}^\star(s_2)$: capacity, optimization dynamics, and implicit bias vary with s.

◦ Strong transfer is generically *not* expected; in practice, we aim for *effective* transfer.

Optimal weights are scale-dependent...

- Some works [30, 9, 29] empirically show $\alpha^\star$ across model scales.
- Domain weights could change at various scales of proxy model (flatter is more stable).

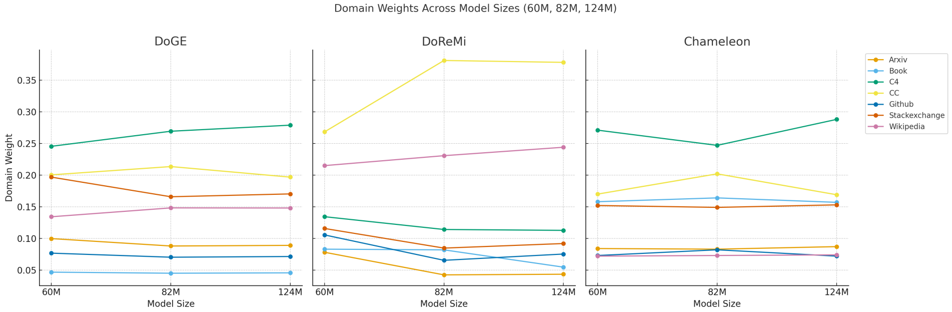
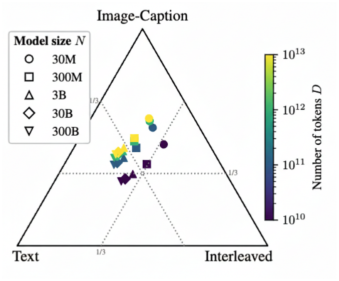


Figure: Optimal domain weights at three proxy model scales for multiple proxy-based methods.

... but effective data mixtures transfer in practice

◦ Can small models reliably select data mixtures for larger models?



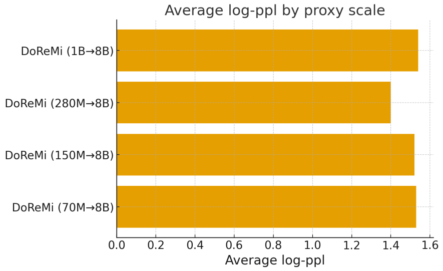


Figure: Figures from [27, 30].

Remarks:

◦ While *optimal* weights change at different scales, the transferred α may still be *effective*.

◦ Optimal α varies with scale, ε-optimal region is broad $\Rightarrow$ proxy mixtures remain effective.

Taxonomy of data mixture methods

Two Main Axes of Classification

- **Temporal Adaptation:** *Offline* (fixed mixture) vs. *Online* (adapt during training).
- **Target Objective:** *General-purpose* (universal generalization $\mathcal{L}_{\text{val}}$) vs. *Task-specific* (downstream target $\mathcal{L}_{\text{task}}$).

Type	General-Purpose	Target Task-Specific
Offline	DoReMi [30] DoGE [9] Data Mixing Laws [33] **Chameleon** [31] ...	RegMix [19] MixMin [28] UtiliMax [14]
Online	ODM [2] ADO [15]	Skill-it! [6] DGA [8]

Proxy-based algorithms: DoReMi [30]

- DoReMi is a three-step algorithm.
  1. Train a small reference model $\mathbf{x}_{\text{ref}}$ with uniform domain weights.
  2. Train proxy model with group distributionally robust optimization to obtain domain weights.
  3. Train large model with new domain weights.

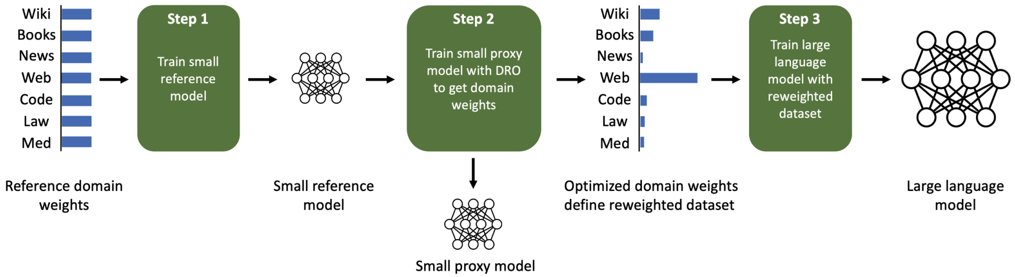

Wiki
Books
News
Web
Code
Law
Med
Reference domain weights
Step 1
Train small reference model
Small reference model
Step 2
Train small proxy model with DRO to get domain weights
Small proxy model
Wiki
Books
News
Web
Code
Law
Med
Optimized domain weights define reweighted dataset
Step 3
Train large language model with reweighted dataset
Large language model

Figure: DoReMi's pipeline [30].

Proxy-based algorithms: DoReMi [30]

◦ Optimize the worst-case loss over domains (Group DRO with *excess loss* [25, 22]):

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha}\in\triangle^k} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha}) := \sum_{i=1}^{k} \alpha_i \cdot [f_i(\mathbf{x}) - f_i(\mathbf{x}_{\text{ref}})]$$

DoReMi intuition

Down-weight redundant and noisy domains, up-weight learnable ones.

$$\begin{aligned}
&\textbf{Redundant}: && \text{low } f_i(\mathbf{x}),\ \text{low } f_i(\mathbf{x}_{\text{ref}}) && \Rightarrow \text{low excess loss } \big(f_i(\mathbf{x}) - f_i(\mathbf{x}_{\text{ref}})\big) && \Rightarrow \textbf{down-weighted};\\
&\textbf{Noisy}: && \text{high } f_i(\mathbf{x}),\ \text{high } f_i(\mathbf{x}_{\text{ref}}) && \Rightarrow \text{low excess loss } \big(f_i(\mathbf{x}) - f_i(\mathbf{x}_{\text{ref}})\big) && \Rightarrow \textbf{down-weighted};\\
&\textbf{Learnable}: && \text{high } f_i(\mathbf{x}),\ \text{low } f_i(\mathbf{x}_{\text{ref}}) && \Rightarrow \text{high excess loss } \big(f_i(\mathbf{x}) - f_i(\mathbf{x}_{\text{ref}})\big) && \Rightarrow \textbf{up-weighted}.
\end{aligned}$$

Remarks:

◦ $\boldsymbol{\alpha}$ highly depends on the performance of reference model.

◦ DoReMi is not efficient enough since it requires two auxiliary models.

◦ Mathematically, one-hot weights can be optimal $\boldsymbol{\alpha}^\star$!

Proxy-based algorithms: DoReMi [30]

∘ DoReMi's core problem corresponds to an inner maximization over domain weights, which is *linear* in α:

$$\max_{\boldsymbol{\alpha}\in\triangle^k} \sum_{i\in[k]} \alpha_i\, g_i^{(t)}, \quad \text{where } g_i^{(t)} = f_i(\mathbf{x}^{(t)}) - f_i(\mathbf{x}_{\text{ref}}).$$

Lemma (Optimality of one-hot weights)

The weights placing all mass on the domain(s) with the largest excess loss solve DoReMi's inner maximization.

∘ No exploration! Idea: treat the maximizer as an online learner updating a distribution $\boldsymbol{\alpha}^{(t)}$ over actions.

Hedge [10] update for DoReMi

For each domain $i \in \{1, \dots, k\}$:

$$\boldsymbol{\alpha}_i' \propto \boldsymbol{\alpha}_i^{(t)} \exp\left(\eta\, g_i^{(t)}\right).$$

Remarks:

∘ Implements entropic regularization to the inner maximization [5, 21].

∘ This prevents collapse to a one-hot solution.

∘ Optimistic update can be applied for further smoothing of the domain weights [7, 16].

Proxy-based algorithms: DoReMi [30]

- In the multiplicative-weights regime, probabilities can collapse: $\boldsymbol{\alpha}_i^{(t)} \to 0$.
- DoReMi enforces stability using uniform smoothing via EXP3 [4].

EXP3 [4] update for DoReMi

DoReMi adds uniform smoothing for each domain $i \in \{1, \ldots, k\}$:

$$\boldsymbol{\alpha}_i^{(t+1)} = (1-\gamma)\frac{\boldsymbol{\alpha}_i'}{\sum_{j=1}^{k} \boldsymbol{\alpha}_j'} + \gamma u_i, \qquad \boldsymbol{u} = \frac{1}{k}\mathbf{1}.$$

Remarks:
- DoReMi formulates domain reweighting as an adversarial online learning problem.
- Uniform exploration prior prevents any domain weight from vanishing.

Proxy-based algorithms: DoGE [9]

- DoGE gets rid of training the reference model.
- DoGE is a two-step algorithm.

1. Train a small-scale proxy model to find optimal domain weights.
   - Formulate domain reweighting as the bilevel problem:

$$\min_{\boldsymbol{\alpha} \in \triangle^k} \sum_{i \in [k]} f_i(\mathbf{x}^\star(\boldsymbol{\alpha})) \quad \text{s.t.} \quad \mathbf{x}^\star(\boldsymbol{\alpha}) \in \arg\min_{\mathbf{x}} \sum_{i \in [k]} \alpha_i f_i(\mathbf{x}). \tag{3}$$

2. Train large-scale base LLM with resampled pretrain corpus according to the finalized domain weights.

Proxy-based algorithms: DoGE [9]

- DoGE gets rid of training the reference model.
- DoGE is a two-step algorithm.

  1. Train a small-scale proxy model to find optimal domain weights.
     - Formulate domain reweighting as the bilevel problem:

$$\min_{\boldsymbol{\alpha} \in \triangle^k} \sum_{i \in [k]} f_i(\mathbf{x}^\star(\boldsymbol{\alpha})) \quad \text{s.t.} \quad \mathbf{x}^\star(\boldsymbol{\alpha}) \in \arg\min_{\mathbf{x}} \sum_{i \in [k]} \alpha_i f_i(\mathbf{x}). \tag{3}$$

  2. Train large-scale base LLM with resampled pretrain corpus according to the finalized domain weights.

Lemma (Optimality of uniform weights)

The uniform domain weight vector $\boldsymbol{\alpha} = \left(\frac{1}{k}, \dots, \frac{1}{k}\right)$ *is a global minimizer of* (3).

Proxy-based algorithms: DoGE [9]

- ◦ DoGE gets rid of training the reference model.
- ◦ DoGE is a two-step algorithm.

1. Train a small-scale proxy model to find optimal domain weights.
   - ▶ Formulate domain reweighting as the bilevel problem:

$$\min_{\boldsymbol{\alpha}\in\triangle^k} \sum_{i\in[k]} f_i(\mathbf{x}^\star(\boldsymbol{\alpha})) \quad \text{s.t.} \quad \mathbf{x}^\star(\boldsymbol{\alpha}) \in \arg\min_{\mathbf{x}} \sum_{i\in[k]} \alpha_i f_i(\mathbf{x}). \tag{3}$$

2. Train large-scale base LLM with resampled pretrain corpus according to the finalized domain weights.

Lemma (Optimality of uniform weights)

The uniform domain weight vector $\boldsymbol{\alpha} = \left(\frac{1}{k}, \ldots, \frac{1}{k}\right)$ is a global minimizer of (3).

Remarks:
- ◦ Uniform weights can be optimal, although we aim to improve over uniform...
- ◦ Alternatively, one can choose outer loss ≠ inner loss, e.g., $\mathcal{L}_{\text{val}}$ or $\mathcal{L}_{\text{task}}$ [8].

Proxy-based algorithms: DoGE [9]

◦ Inner-loop: model update at step t:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \sum_{i \in [k]} \alpha_i^{(t)} \nabla f_i(\mathbf{x}^{(t)}), \qquad \boldsymbol{\alpha}^{(t)} \in \Delta^k. \tag{1}$$

◦ Outer-loop: one-step greedy expansion. Minimize the average loss across domains at the next step $(t+1)$:

$$\begin{aligned} \boldsymbol{\alpha}^{(t)\star} &= \arg \min_{\boldsymbol{\alpha} \in \Delta^k} \sum_{i \in [k]} \left[f_i(\mathbf{x}^{(t+1)}) - f_i(\mathbf{x}^{(t)}) \right] \\ &= \arg \min_{\boldsymbol{\alpha} \in \Delta^k} \sum_{i \in [k]} \left\langle \nabla f_i(\mathbf{x}^{(t)}), \mathbf{x}^{(t+1)} - \mathbf{x}^{(t)} \right\rangle + \epsilon^{(t)} \\ &= \arg \min_{\boldsymbol{\alpha} \in \Delta^k} \sum_{i \in [k]} \left\langle \nabla f_i(\mathbf{x}^{(t)}), -\eta \sum_{j \in [k]} \alpha_j \nabla f_j(\mathbf{x}^{(t)}) \right\rangle + \epsilon^{(t)}. \end{aligned}$$

Remark: ◦ The domain weight update uses approximations based on "influence functions" [13, 17].

Proxy-based algorithms: DoGE [9]

◦ Define the generalization estimation function of domain j as the *gradient alignment*:

$$W_j^{(t)} \triangleq \left\langle \nabla f_j(\mathbf{x}^{(t)}), \sum_{i\in[k]} \nabla f_i(\mathbf{x}^{(t)}) \right\rangle.$$

◦ The outer problem becomes:

$$\boldsymbol{\alpha}^{(t)} = \arg\min_{\boldsymbol{\alpha}\in\Delta^k} -\eta^{(t)}\boldsymbol{\alpha}^\top \boldsymbol{\mathcal{W}}^{(t)} + \mu\, D_\Psi(\boldsymbol{\alpha}\|\boldsymbol{\alpha}^{(t-1)}), \qquad \boldsymbol{\mathcal{W}}^{(t)} = [W_1^{(t)}, \ldots, W_k^{(t)}] \in \mathbb{R}^k,$$

where the Bregman divergence $D_\Psi(\boldsymbol{\alpha}\|\boldsymbol{\alpha}^{(t-1)})$ with $\Psi(\boldsymbol{\alpha}) = \sum_i \alpha_i \log(\alpha_i)$ is used to remain on the simplex.

◦ DoGE's domain weights update rule:

$$\boldsymbol{\alpha}^{(t)} \propto \boldsymbol{\alpha}^{(t-1)} \odot \exp\left(\frac{\eta}{\mu} \boldsymbol{\mathcal{W}}^{(t)}\right).$$

Remark: ◦ The proxy model requires k gradient computations per update, slowing training.

◦ DoGE is sensitive to the choice of μ [31].

Proxy-based algorithms: RegMix [19]

- RegMix grid searches domain proportions and predicts optimal weights via regression.
- RegMix follows four steps.

1. Train a large number of small proxy models.
   - Generate n random mixtures $\boldsymbol{\alpha}_j$.
   - For each $\boldsymbol{\alpha}_j$, train proxy $\mathbf{x}^\star(\boldsymbol{\alpha}_j)$ and compute target metric $y_j = \mathcal{L}_{\text{target}}(\mathbf{x}^\star(\boldsymbol{\alpha}_j))$.
   - Example settings: $n = 512$ of scale $1M$.
2. Fit regression model.
   - Fit a regression model $V(\boldsymbol{\alpha}) : \Delta^k \to \mathbb{R}$ on $\{(\boldsymbol{\alpha}_j, y_j)\}_{j=1}^n$ (e.g., linear regression or tree model).
3. Simulate and predict.
   - Find $\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}} V(\boldsymbol{\alpha})$.
   - Solved by random search over $1,000,000$ candidates.
4. Train large-scale model.

Proxy-based algorithms: RegMix [19]

- Perplexity average may not directly relate to downstream tasks' performance.
  - RegMix finds that Common Crawl (CC) is highly relevant to downstream tasks' performance.
  - It uses the validation loss of Pile-CC [11] as $\mathcal{L}_{\text{target}}$ objective.

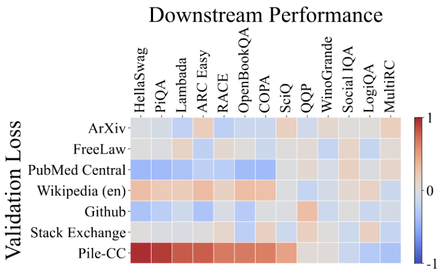


Figure: Correlation between validation loss by domains of the Pile and downstream performance.

From proxy-based methods to a data-centric view

So far: DoReMi, DoGE, RegMix

- All critically rely on proxy training runs and challenging bilevel-style optimization.
- Domain weights are tightly coupled to the training dynamics.
- Any change in domains $\Rightarrow$ retrain proxy models.

Our goal with Chameleon:

- Domain weights from a **single, well-understood least-squares problem**...

A least-squares view of domain importance

We equip each domain with a vector embedding $\boldsymbol{v}_i \in \mathbb{R}^p$. Let $\mathbf{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]^\top \in \mathbb{R}^{k \times p}$.

Min-norm reconstruction problem

For each domain i, consider:

$$S_i = \min_{\mathbf{y} \in \mathbb{R}^k} \left\{ \| \mathbf{y} \|_2^2 + \frac{1}{\lambda} \| \mathbf{V}^\top \mathbf{y} - \boldsymbol{v}_i \|_2^2 \right\}.$$

The S_i is called the *ridge leverage score* of row $\boldsymbol{v}_i$ in matrix $\mathbf{V}$ [20].

Remarks:

- If $\boldsymbol{v}_i$ is a simple linear combination of other rows of $\mathbf{V}$, a small-norm $\mathbf{y}$ can exist $\Rightarrow S_i$ is **small**.
- If $\boldsymbol{v}_i$ is hard to reconstruct from other domains, then $\mathbf{y}$ must have larger norm $\Rightarrow S_i$ is **large**.
- Interpretation: S_i measures how *unique* domain i is in the geometry of the embedded dataset.

Illustrative examples of leverage scores

Example (2D example)

Let $\boldsymbol{v}_1 = (1, 0),\ \boldsymbol{v}_2 = (1, 0.1),\ \boldsymbol{v}_3 = (0, 1)$ be three domain embeddings.

- $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are nearly collinear, $\boldsymbol{v}_3$ is orthogonal to $\boldsymbol{v}_1$ and nearly orthogonal to $\boldsymbol{v}_2$.
- To reconstruct $\boldsymbol{v}_1$: a balanced mix of $\boldsymbol{v}_1$ and $\boldsymbol{v}_2 \Rightarrow$ as $\lambda \to 0$, $\| \mathbf{y}^\star \|_2^2 \approx 0.5$ (low leverage).
- Same for $\boldsymbol{v}_2$.
- To reconstruct $\boldsymbol{v}_3$: one must put almost all weight to itself $\Rightarrow \| \mathbf{y}^\star \|_2^2 \approx 1$ (high leverage).

Example (NLP example)

Consider three domains: D_1 (Common Crawl), D_2 (C4), and D_3 (Arxiv). We expect embeddings $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ to be highly similar, while $\boldsymbol{v}_3$ is relatively orthogonal.

$$\boldsymbol{v}_1 \approx \boldsymbol{v}_2, \quad \boldsymbol{v}_3^\top \boldsymbol{v}_1 \approx 0, \quad \boldsymbol{v}_3^\top \boldsymbol{v}_2 \approx 0$$

Consequently, we anticipate: S_1, S_2 are low and S_3 is high.

Computing domain affinity

∘ We construct the kernel matrix of domain embeddings as $\mathbf{\Omega} = [\kappa(\boldsymbol{v}_i, \boldsymbol{v}_j)]_{i,j=1}^{k} \in \mathbb{R}^{k \times k}$.

Definition: Kernel Ridge Leverage Scores (KRLS)

The KRLS [1] of domain D_i is the i-th diagonal element of the kernel *hat matrix* with regularization $\lambda > 0$:

$$S_i = \left[\mathbf{\Omega}(\mathbf{\Omega} + \lambda \boldsymbol{I})^{-1}\right]_{ii}. \tag{4}$$

∘ Eigendecomposition perspective. Let $\mathbf{\Omega} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^\top$ with $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$ be the reduced SVD, then

$$S_i = \sum_{j=1}^{k} \frac{\sigma_j}{\sigma_j + \lambda} U_{ij}^2.$$

Remark: ∘ For $\lambda > 0$, the contribution of directions with small eigenvalues is damped.

Least-squares view of KRLS

◦ Primal problem of KRLS of domain i:

$$\boxed{\text{P}} \quad \min_{\boldsymbol{w},\boldsymbol{e}} \frac{1}{2}\sum_{j=1}^{k} e_j^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 \quad \text{s.t. } e_j = \delta_j - \boldsymbol{w}^\top \boldsymbol{v}_j \ \ \forall j, \tag{5}$$

where

$$\delta_j = \begin{cases} 1 & \text{if } j = i, \\ 0 & \text{otherwise.} \end{cases}$$

◦ Lagrangian:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{e}; \boldsymbol{\alpha}) = \frac{1}{2}\sum_{j=1}^{k} e_j^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 - \sum_{j=1}^{k} \alpha_j (e_j - \delta_j + \boldsymbol{w}^\top \boldsymbol{v}_j). \tag{6}$$

◦ Elimination of $\boldsymbol{w}, \boldsymbol{e}$ in the optimality conditions gives:

$$\boxed{\text{D}} \quad (\boldsymbol{\Omega} + \lambda \boldsymbol{I})\boldsymbol{\alpha} = \lambda \boldsymbol{\delta},$$

with linear kernel matrix $\boldsymbol{\Omega}$.

KRLS: Model representation

◦ Model-based approach of KRLS:

$$\begin{tikzcd} & \boxed{\text{P}}\ S_i = \boldsymbol{w}^\top \boldsymbol{v}_i \\ \text{KRLS of } \boldsymbol{v}_i \arrow[ur] \arrow[dr] & \\ & \boxed{\text{D}}\ S_i = \boldsymbol{\delta}^\top \left[\boldsymbol{\Omega}(\boldsymbol{\Omega} + \lambda \boldsymbol{I})^{-1}\right] \boldsymbol{\delta} \end{tikzcd}$$

Remarks:

◦ The dual $\boxed{\text{D}}$ recovers the standard definition of KRLS in (4).

◦ The primal $\boxed{\text{P}}$ provides the interpretation of KRLS as finding a vector $\boldsymbol{w}$ that "picks out" $\boldsymbol{v}_i$.

◦ Leads to a natural way to extend KRLS to **multiple modalities** (see our recent preprint [32]).

From data domains to geometric embeddings

- How to learn meaningful embeddings of data domains?
- A proxy model $h_{\mathbf{x}_p}$ is first trained with uniform domain weights.

Definition: Domain embedding

The embedding for a domain D_i is defined as the *centroid* of the representations of its constituent data points, extracted from the L-th hidden layer of the proxy model $h_{\mathbf{x}_p}$.

$$\boldsymbol{v}_i = \frac{1}{|D_i|} \sum_{\mathbf{a} \in D_i} h_{\mathbf{x}_p}^{(L)}(\mathbf{a}) \in \mathbb{R}^p.$$

Remarks:

- In practice, this expectation is estimated using a sufficiently large batch $B_i \subseteq D_i$.
- Proximity between vectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ in the embedding space corresponds to a high degree of semantic similarity between domains D_i and D_j.

The Chameleon pipeline

[Xie, Tonin, Cevher; ICML'25]

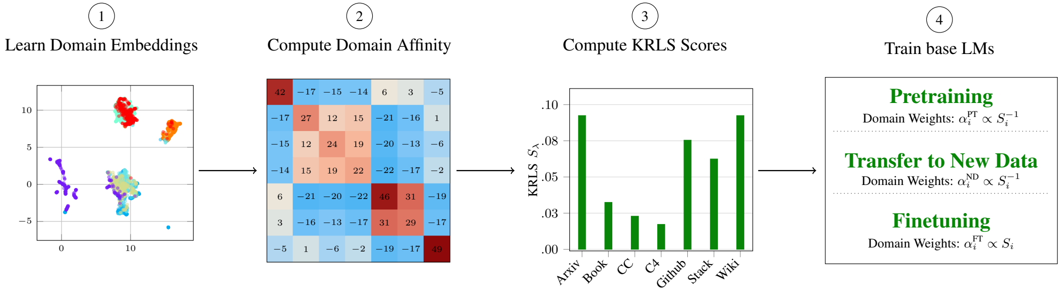


Figure: CHAMELEON's pipeline.

Remarks:

- We use a linear kernel $\kappa(\boldsymbol{v}_i, \boldsymbol{v}_j) = \boldsymbol{v}_i^\top \boldsymbol{v}_j$ as the LLM embeddings are already highly non-linear.
- Chameleon is more stable since domain weights are decoupled from the training process.
- The time for step 1-3 in the above figure is negligible compared to proxy model training.

Using KRLS differently for pretraining vs finetuning

- We propose distinct weighting strategies for pretraining and finetuning.
- One geometric score, two regimes: generalization vs. specialization.

Pretraining: learn broad, shared knowledge.

- Upweight *representative* (low-score) domains.
- Learn general-purpose representations.

$$\alpha_i^{\mathsf{PT}} \propto \frac{1}{S_i}$$

Example

In the SlimPajama dataset, broad domains like 'Common Crawl (CC)' and 'C4' lie in denser regions of the embedding space. Our α^{PT} upweights them.

Finetuning: specialize to a target.

- Upweight *novel* (high-score) domains.
- Focus on features underrepresented in pretraining.

$$\alpha_i^{\mathsf{FT}} \propto S_i$$

Example

When finetuning on the 'Stack' code dataset, the 'Go' domain might be distant from 'Python' or 'Java' Our α^{FT} upweights 'Go' to learn its distinct syntax.

Direct extension to new data

◦ Chameleon allows direct weight computation for new domains.

Out-of-sample extension

1. Given a new set of domains $D' = \{D_1, \ldots, D_{k'}\}$.
2. Generate embeddings for new domains using the *existing* proxy: $\boldsymbol{v}'_j = \mathbb{E}_{\mathbf{a} \in D_j}[h_{\mathbf{x}}(\mathbf{a})]$.
3. Construct the new affinity matrix $\boldsymbol{\Omega}'$.
4. Calculate the new KRLS scores:

$$S'_j = \left[\boldsymbol{\Omega}'(\boldsymbol{\Omega}' + \lambda \boldsymbol{I})^{-1}\right]_{jj} .$$

Remarks: ◦ In existing methods, any change to the dataset D requires costly retraining from scratch.

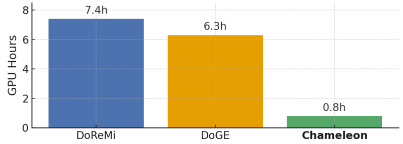
◦ Our method is decoupled from the proxy model's optimization loop.

Experiments: Universal generalization on SlimPajama

Setup: Pretraining a 684M model on SlimPajama (7 domains). Weights derived from an 82M proxy model.

Domain	**Uniform**	DoReMi	DoGE	**Chameleon**	RegMix
Arxiv	8.16	9.16	9.07	8.31	11.35
Book	42.55	46.48	40.30	39.23	41.52
CC	45.26	40.62	38.99	40.11	37.32
C4	49.00	43.92	40.65	42.59	43.85
Github	3.99	4.10	4.09	4.20	4.99
Stackexchange	7.99	8.35	7.39	7.94	10.63
Wikipedia	12.42	10.78	15.74	13.90	20.88
Avg. PPL (↓)	24.20	23.34	22.32	**22.31**	24.36

Perplexity (PPL) results on the SlimPajama dataset. Lower is better.

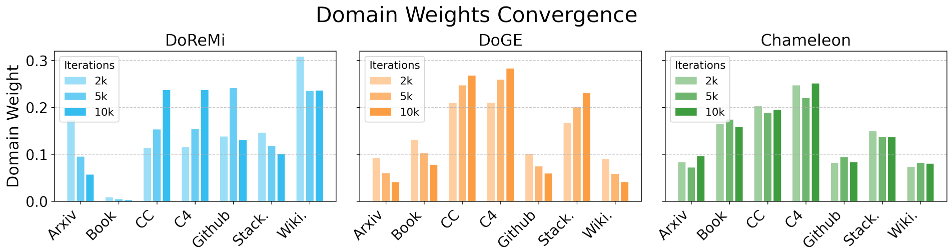


GPU hours for proxy training.

Remarks:
- ○ Chameleon achieves the best perplexity on SlimPajama.
- ○ Chameleon is significantly more efficient, requiring only 1/10th of the FLOPs of DoReMi.

Experiments: Score convergence

- Domain weights across methods during proxy training.



Remarks:

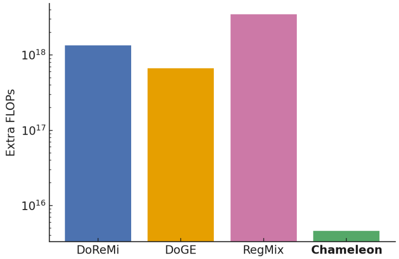
- Chameleon converges quickly.
- DoReMi and DoGE require more iterations to stabilize.
- Chameleon is also robust to training steps, proxy scales, λ, and embedding sample size.

Experiments: Zero-shot transfer to new domains

- The Scalability Test: Can we adapt to a new dataset (The Pile, 17 domains) without retraining?
  - Baselines (DoReMi, DoGE, RegMix) must retrain a proxy from scratch on The Pile.
  - Chameleon simply performs inference using the *same proxy model* trained on SlimPajama.

Benchmark	**Uniform**	DoReMi	DoGE	**Chameleon**	RegMix
ARC-E	37.5	39.3	35.3	39.2	39.5
COPA	56.8	61.5	54.7	60.9	61.2
HellaSwag	26.7	27.3	26.1	27.4	27.3
Lambada	12.5	15.8	9.3	15.9	15.4
LogiQA	22.5	21.2	22.1	23.8	21.9
MultiRC	56.7	56.5	57.2	57.3	56.2
OpenBook	13.3	13.1	13.1	14.2	14.5
PiQA	57.8	59.3	55.9	59.7	60.4
QQP	37.5	36.8	36.8	37.2	37.6
RACE	25.8	27.2	24.9	26.8	27.2
SciQ	64.1	65.7	58.1	66.0	67.1
Social IQA	35.0	36.0	34.2	36.6	36.3
WinoGrande	50.7	51.2	49.8	50.9	49.9
Average (↑)	38.2	39.3	36.7	**39.7**	39.6

Downstream reasoning accuracies. Higher is better (↑).



Extra FLOPs for mixing new domains.

Experiments: Extending data mixing to finetuning

Setup: Finetune a pretrained model on two specialized datasets:

1. Wiki40b: 7 natural languages
2. The Stack: 7 programming languages

Finetuning on Wiki40b

Domain	**Uniform**	**Chameleon**
French	6.86	6.51
German	10.12	8.78
Italian	13.29	12.42
Spanish	8.41	8.04
Portuguese	8.00	7.78
Dutch	13.98	12.30
Polish	5.07	4.21
Average PPL (↓)	9.43	**8.58**
# Domains Over Uniform	-	7/7

Finetuning on The Stack

Domain	**Uniform**	CHAMELEON
Python	19.98	16.53
Java	19.27	15.53
C	28.24	22.58
C++	25.16	21.09
Go	30.25	19.26
Ruby	21.78	17.83
PHP	9.45	7.43
Average PPL (↓)	22.02	**17.18**
# Domains Over Uniform	-	7/7

Remarks:

- ○ Data mixing is effective for finetuning.
- ○ Embeddings can be easily extracted directly from the pretrained model.
- ○ Chameleon demonstrates its flexibility in both pretraining and finetuning phases.

The effect of data mixing on Scion [24]

- Chameleon: optimizes mixtures of pretraining data domains.
- Scion+Chameleon: 40% less steps or 3% better performance on SlimPajama dataset.

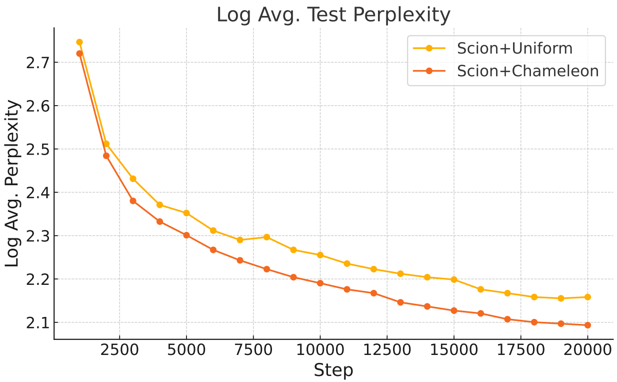


Figure: Significant accelerations in pretraining a 210M LM combining Scion with Chameleon data mixing.

Conclusion

- ◦ Chameleon is a novel, efficient, and flexible data-mixing framework based on Kernel Ridge Leverage Scores.
- ◦ Zero-shot transfer: Adapts to new data domains without costly retraining.
- ◦ Flexibility: Provides a unified, principled approach for both pretraining and finetuning data mixing.

🔔 **NEW** : **Compute grant** – Awarded 50k GPU-hours (NVIDIA GH200) from Swiss AI for:

- ▶ Online adaptation: Extend to online settings where domain weights are adjusted during training.
- ▶ Geometry-aware optimization: Analyze the impact of data mixing on a new class of geometry-aware optimizers — such as Scion and Muon.

Thank you. Questions?

Code: `https://github.com/LIONS-EPFL/Chameleon`

References I

[1] Ahmed El Alaoui and Michael W. Mahoney. Fast Randomized Kernel Methods With Statistical Guarantees, 2015. arXiv:1411.0306. (Cited on page 35.)

[2] Alon Albalak, Liangming Pan, Colin Raffel, and William Yang Wang. Efficient online data mixing for language model pre-training. *arXiv preprint arXiv:2312.02406*, 2023. (Cited on page 20.)

[3] Kimon Antonakopoulos, Shoham Sabach, Luca Viano, Mingyi Hong, and Volkan Cevher. Adaptive bilevel optimization. *ACM/IMS Journal of Data Science*, 2(2):1–29, 2025. (Cited on page 14.)

[4] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002. (Cited on page 24.)

[5] Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003. (Cited on page 23.)

[6] Mayee F. Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models, 2023. (Cited on page 20.)

[7] Constantinos Daskalakis, Maxwell Fishelson, and Noah Golowich. Near-optimal no-regret learning in general games. *Advances in Neural Information Processing Systems*, 34:27604–27616, 2021. (Cited on page 23.)

[8] Simin Fan, David Grangier, and Pierre Ablin. Dynamic gradient alignment for online data mixing. *arXiv preprint arXiv:2410.02498*, 2024. (Cited on pages 20, 25, 26, and 27.)

References II

[9] Simin Fan, Matteo Pagliardini, and Martin Jaggi. DOGE: Domain reweighting with generalization estimation. In *International Conference on Machine Learning*, 2024. (Cited on pages 9, 16, 18, 20, 25, 26, 27, 28, and 29.)

[10] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. (Cited on page 23.)

[11] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. (Cited on pages 6 and 31.)

[12] Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018. (Cited on page 14.)

[13] Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974. (Cited on page 28.)

[14] William Held, Bhargavi Paranjape, Punit Singh Koura, Mike Lewis, Frank Zhang, and Todor Mihaylov. Optimizing pretraining data mixtures with llm-estimated utility. *arXiv preprint arXiv:2501.11747*, 2025. (Cited on page 20.)

[15] Yiding Jiang, Allan Zhou, Zhili Feng, Sadhika Malladi, and J Zico Kolter. Adaptive data optimization: Dynamic sample selection with scaling laws. *arXiv preprint arXiv:2410.11820*, 2024. (Cited on page 20.)

References III

[16] Ehsan Asadi Kangarshahi, Ya-Ping Hsieh, Mehmet Fatih Sahin, and Volkan Cevher. Let's be honest: An optimal no-regret framework for zero-sum games. In *International Conference on Machine Learning*, pages 2488–2496. PMLR, 2018. (Cited on page 23.)

[17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017. (Cited on page 28.)

[18] Junyi Li, Bin Gu, and Heng Huang. A fully single loop algorithm for bilevel optimization without hessian inverse. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7426–7434, 2022. (Cited on page 14.)

[19] 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. *arXiv preprint arXiv:2407.01492*, 2024. (Cited on pages 20, 30, and 31.)

[20] Michael W. Mahoney and Petros Drineas. CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009. (Cited on page 33.)

[21] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009. (Cited on page 23.)

[22] Yonatan Oren, Shiori Sagawa, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust language modeling. *arXiv preprint arXiv:1909.02060*, 2019. (Cited on page 22.)

[23] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746. PMLR, 2016. (Cited on page 14.)

References IV

[24] Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. In *Forty-second International Conference on Machine Learning*, 2025. (Cited on page 46.)

[25] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019. (Cited on page 22.)

[26] Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, et al. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*, 2023. (Cited on page 6.)

[27] Mustafa Shukor, Louis Bethune, Dan Busbridge, David Grangier, Enrico Fini, Alaaeldin El-Nouby, and Pierre Ablin. Scaling laws for optimal data mixtures. *arXiv preprint arXiv:2507.09404*, 2025. (Cited on page 19.)

[28] Anvith Thudi, Evianne Rovers, Yangjun Ruan, Tristan Thrush, and Chris J Maddison. Mixmin: Finding data mixtures via convex minimization. *arXiv preprint arXiv:2502.10510*, 2025. (Cited on page 20.)

[29] Jiachen T. Wang, Tong Wu, Kaifeng Lyu, James Zou, Dawn Song, Ruoxi Jia, and Prateek Mittal. Can small training runs reliably guide data curation? rethinking proxy-model practice, 2025. (Cited on page 18.)

[30] 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. (Cited on pages 18, 19, 20, 21, 22, 23, and 24.)

References V

[31] Wanyun Xie, Francesco Tonin, and Volkan Cevher. Chameleon: A flexible data-mixing framework for language model pretraining and finetuning. In *Forty-second International Conference on Machine Learning*, 2025. (Cited on pages 20 and 29.)

[32] Wanyun Xie, Francesco Tonin, and Volkan Cevher. Mad-mix: Multi-modal data mixtures via latent space coupling for vision-language model training, 2026. (Cited on page 37.)

[33] 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 *The Thirteenth International Conference on Learning Representations*, 2025. (Cited on page 20.)

[34] Yihua Zhang, Prashant Khanduri, Ioannis Tsaknakis, Yuguang Yao, Mingyi Hong, and Sijia Liu. An introduction to bilevel optimization: Foundations and applications in signal processing and machine learning. *IEEE Signal Processing Magazine*, 41(1):38–59, 2024. (Cited on pages 10 and 14.)