Data Selection for Language Models: From the Perspective of Learning to Optimize

Zhonglin Xie

Beijing International Center for Mathematical Research
Peking University

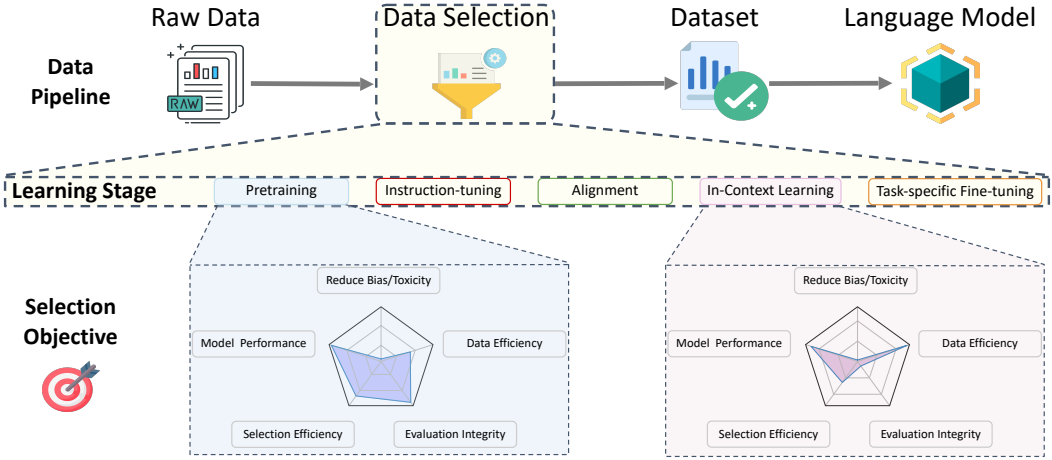
April 10, 2025

Outline

What is Data Selection?

- **Definition:** Designing an *optimal* dataset from raw data under some objective function.
- **Probabilistic View:** Optimal dataset matches the distribution where the model will be evaluated.
- **Goals:**
  - Improve model **performance**.
  - Reduce **cost** (dataset size, training time).
  - Ensure **evaluation integrity** (remove test contamination).
  - Reduce **bias** and **toxicity**.
- Especially important for **Large Language Models (LLMs)** at various stages (pretraining, instruction-tuning, alignment).

Data Pipeline for Language Models



- **Utility Function:** Determines the value/utility of a data point.
- **Selection Mechanism:** Decides how to use a data point based on its utility.

Taxonomy: Background

- **Token:** Smallest unit (byte, char, subword).
- **Data Point ($x^{(i)}$):** Ordered collection of tokens (a single sample).
- **Data Point Characteristics:** Measures describing a data point (length, topic, embedding). Used to determine inclusion/cleaning.
- **Dataset ($\mathcal{D}$):** Collection of data points $\{x^{(1)}, \ldots, x^{(N)}\}$.
- **Dataset Distribution:** Distribution of data points in the data space. Crucial for model generalization (in-distribution vs. out-of-distribution).

Taxonomy: Unified Conceptual Framework

- **Goal:** Filter and select data points from $\mathcal{D}_{\text{raw}}$ to create $\mathcal{D}$ that maximizes a desired objective $f_{\text{obj}}(\mathcal{M})$ for a model $\mathcal{M}$.
- **Formal Definition:** $\mathcal{D} = \phi(\mathcal{D}_{\text{raw}})$, where ϕ is the data selection function.
- Methods can be composed: $\phi = \phi_1 \circ \cdots \circ \phi_n$.
- **Components of Selection Functions ϕ_j:**
  - **Utility Function** $U(x^{(i)}) \to \mathbb{R}$**:** Assigns a score representing data point utility (e.g., quality, relevance, length).
  - **Selection Mechanism:** Uses utility score to decide inclusion/repetition (e.g., thresholding, probabilistic sampling). Requires *selection sensitivity* (e.g., threshold value).

Taxonomy: Dimensions of Variance (1/2)

Distribution Matching vs. Diversification

- **Matching:** Select data similar to a target distribution (e.g., high quality, specific domain, language).
  - Utility = Similarity to target.
  - Uses target data statistics or representations.
- **Diversification:** Prioritize heterogeneity, remove redundancy.
  - Utility = Relation to other points (dissimilarity).
  - Improves efficiency, reduces memorization/bias.

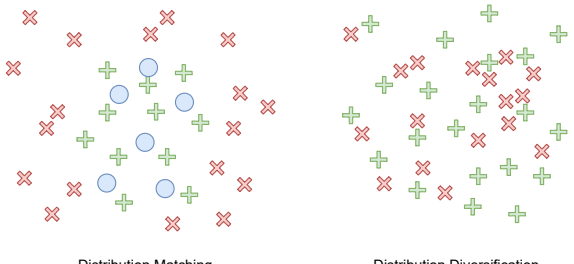


Figure: Conceptual view of Matching vs. Diversification goals.

Taxonomy: Dimensions of Variance (2/2)

- **Altering the Dataset vs. Data Point**
  - *Dataset:* Change frequency of data points (filter, oversample). Map $x^{(i)} \to \mathbb{N}_0$.
  - *Data Point:* Modify content within a data point (e.g., remove HTML tags).
- **Output Space: Binary vs. Natural Number Selection**
  - *Binary (Filtering):* Include or remove ($\{0, 1\}$).
  - *Natural Number (Mixing):* Assign repetition count ($\{0, 1, 2, ...\}$). Often used for weighting sources (e.g., web vs. books).
- **Training Stage**
  - Goals/methods vary: Pretraining, Instruction-Tuning, Alignment, In-Context Learning, Task-Specific Fine-Tuning.
  - Considerations: Target distribution clarity, dataset size, computational budget.

Outline

Overview of Pretraining Data Selection Methods

Selection Method	Distribution **M**atching vs. **D**iversification	**Output Space**	Adjust **D**ataset vs. Data **P**oint
Language Filtering	M	$\{0, 1\}$	D
Heuristic Approaches	M	$\mathbb{N}_0$	D + P
Data Quality	M	$\{0, 1\}$	D
Domain-specific	M	$\{0, 1\}$	D
Deduplication	D	$\{0, 1\}$	D + P
Toxic and Explicit Content	M	$\{0, 1\}$	D + P
Multilingual Filtering	M + D	$\mathbb{N}_0$	D + P
Data Mixing	M + D	$\mathbb{N}_0$	D

Table: Taxonomy of pretraining data selection methods along three axes.

Data Selection Pipeline for Pretraining

Figure: **An overview of the data filtering pipeline for pretraining**. Different works employ different filters, at different stages, and do not necessarily adhere to the order conveyed here.

Common Methodologies in Data Selection

- **URL-based methods:** Filter based on domain names or specific URL patterns.
  - Used for language filtering, toxic content filtering.
  - Simple but effective for certain tasks.
- **Classifier-based methods:** Train models to identify specific properties.
  - Used for language detection, quality filtering, toxicity detection.
  - Example: fastText for language identification.
- **Common utility functions:**
  - Heuristic-based (length, repetition, ratios)
  - Perplexity-based (for quality and domain filtering)
  - Hash-based (for deduplication)

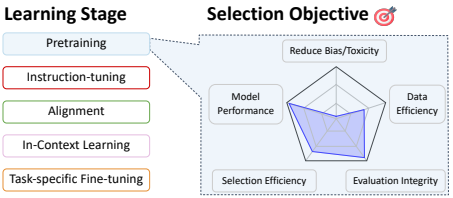


Figure: **Data selection: pretraining**. The first training stage of modern language models. Typically, the most important selection objectives are *model performance*, *evaluation integrity*, and *selection efficiency*.

Data Mixing (1/2)

- **What is Data Mixing?** Optimizing domain weights $\alpha \in \Delta^k$ across k domains to improve performance.
- **Why it's important:**
  - Choice of domain weights significantly impacts downstream accuracy.
  - Optimal mixing can improve training efficiency and model performance.
- **Common Baselines:**
  - **Heuristic/manual** weights (e.g., upweighting books and Wikipedia).
  - **Empirically determined** weights using downstream performance.

Data Mixing (2/2)

- **Principled Approaches:**
  - **Offline methods**: DoReMi, DoGE - optimize static domain weights.
  - **Online methods**: Skill-it, ShearedLLaMA, ODM - adapt weights during training.
- **Challenges:**
  - Trade-off: increasing one domain density decreases others.
  - Weight transferability across tokenizers and training scales.
  - Potential overfitting on smaller domains when scaling to more tokens.

Current Best Practices for Data Selection

- **General workflow:** Start with efficient methods that remove broad quantities, then apply specialized filters.
- **Essential steps:**
  - Language filtering (fastText classifier)
  - Heuristic filtering (length, repetition, format)
  - Deduplication (URL-based, MinHash, Bloom filters)
  - Toxic content filtering (blacklists, classifiers)
- **Domain-specific considerations:**
  - For multilingual models: adjust parameters per language
  - For domain-targeted models: methods like DSIR

Current Landscape of Datasets For Data Selection Research

- **Most popular datasets:**
  - **C4** (750GB): Oldest, web-crawled only, good baseline for improvements
  - **The Pile** (800GB): 28% web scrape, 72% diverse domains, good for data mixing research
  - **RedPajama** (1T tokens): Recreation of LLaMA-2 dataset
  - **RedPajama-2** (30T tokens): Largest with precomputed quality signals
- **Selection research typically uses:**
  - RedPajama: Skill-it, SlimPajama, data mixing laws
  - The Pile: ODM, DSIR, DoReMi
  - C4: DsDm and earlier works

Outline

Algorithm Unrolling (AU)

AU consists of two steps

- Pick a classic iteration and unroll it to an Neural Network (NN)
- Select a set of NN parameters to learn

LASSO example: assume $b = Ax^{\text{true}} + \text{noise}$; recover x^{true} by

$$x^{\text{lasso}} \leftarrow \underset{x}{\text{minimize}} \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

Iterative soft-thresholding algorithm (ISTA):

$$x^{k+1} = \eta_{\lambda\alpha}\left(x^k - \alpha A^T\left(Ax^k - b\right)\right)$$

- convergence requires a proper stepsize α or line search
- the gradient-descent step reduces $\frac{1}{2}\|Ax - b\|^2$
- the soft-thresholding step $\eta_{\lambda\alpha}(\cdot)$ reduces $\lambda\|x\|_1$

Unrolled ISTA

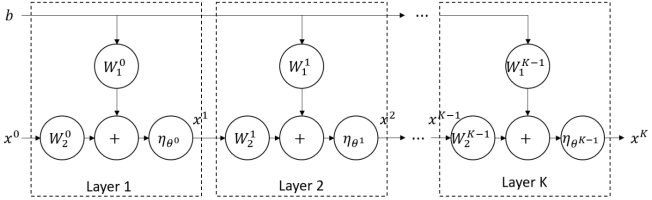
- Introduce scalar $\theta = \lambda\alpha$ and matrices $W_1 = \alpha A^T$ and $W_2 = I - \alpha A^T A$
- Rewrite ISTA as

$$x^{k+1} = \eta_\theta \left(W_1 b + W_2 x^k \right)$$

- Unrolling: introduce $\theta^k, W_1^k, W_2^k, k = 0, 1, \ldots$, as free parameters and re-define

$$x^{k+1} = \eta_{\theta^k} \left(W_1^k b + W_2^k x^k \right)$$

which resembles a DNN:



- Once θ^k, W_1^k, W_2^k are chosen, the algorithm is defined

Train the Unrolled ISTA

- **Objective:** Find θ^k, W_1^k, W_2^k for $k = 0, 1, \ldots$, such that the algorithm converges quickly for LASSO instances with the same matrix A.

- **Setup and Training:**
  - Fix a random matrix A, generate sparse vectors x_i^{true} with varying supports, and compute $b_i = Ax_i^{\text{true}} + \text{noise}_i$. Form the training set $D = \{(x_i^{\text{true}}, b_i)\}$.
  - Fix a small $K > 0$, and train the parameters $\{\theta^k, W_1^k, W_2^k\}_{k=0}^K$ using SGD to minimize:

$$\underset{\{\theta^k, W_1^k, W_2^k\}_{k=0}^K}{\text{minimize}} \sum_{(x^*, b) \in D} \left\| x^K(b) - x^* \right\|_2^2,$$

  where $x^K(b)$ is the K-layer output of the neural network.

Performance of the Learned ISTA (LISTA)

After the NN is trained with $K = 16$, the test performance is shockingly good:

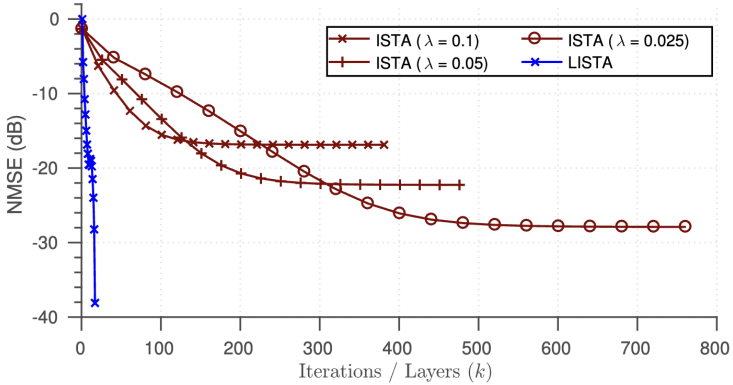


Figure: The trained unrolled ISTA is called Learned ISTA (LISTA)

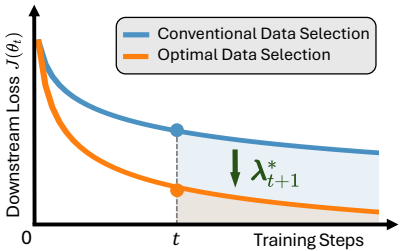
LISTA works much better than ISTA at any λ and using a theoretical stepsize

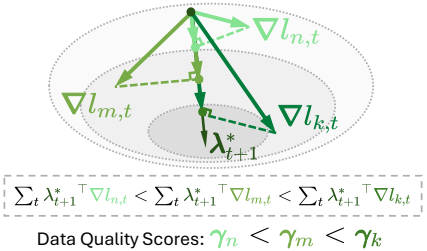
Outline

Data Selection via Optimal Control: Motivation

- **Motivation:** Inspired by learned optimizer literature that focuses on training dynamics
- **Core Insight:** Data selection can be formulated as an *Optimal Control* problem
- **Key Idea:** Optimize data quality scores γ that minimize LM's downstream loss during training
  - Control variables: Data quality scores γ
  - State variables: Model parameters θ
  - Objective function: AUC of the downstream loss $J(\theta)$
- **Pontryagin's Maximum Principle (PMP):** Provides necessary conditions for optimal control

Problem Formulation: Optimal Control Framework





- **Goal**: Find optimal data quality scores γ^* that minimize:

$$\min_{\gamma} \sum_{t=1}^{T} J(\theta_t) \quad \text{subject to: } \theta_{t+1} = \theta_t - \eta \nabla L(\theta_t, \gamma),\ \gamma \in U$$

- **Where:** $L(\theta, \gamma) = \sum_{n=1}^{|\mathcal{D}_{\text{trn}}|} \gamma_n l(x_n, \theta)$, U denotes the $|\mathcal{D}_{\text{trn}}|$-dimensional simplex, and $J(\theta)$ represents the downstream loss function.

Connection Between AUC and Scaling Law Constants

Minimizing the Area Under the Curve (AUC) directly improves model scaling properties:

- **LM Scaling Law**: $L(t) = \frac{C}{t^c} + L^{\text{irre}}$, where $t > T_0$
  - C and c are scaling law constants
  - L^{irre} is the irreducible loss (noise in test set)
  - T_0 is the end of warmup stage
- **Reducible Loss**: $L^{\text{re}}(t) = \frac{C(\gamma)}{t^{c(\gamma)}}$
- **AUC of Reducible Loss**:

$$\text{AUC}(\gamma) = \int_{t=T_0}^{T} \frac{C(\gamma)}{t^{c(\gamma)}} dt = \frac{C(\gamma)}{1 - c(\gamma)} (T^{1-c(\gamma)} - T_0^{1-c(\gamma)})$$

- **Key Result**: Minimizing AUC causes $C(\gamma)$ to decrease and $c(\gamma)$ to increase, improving LM scaling properties

Pontryagin's Maximum Principle for Data Selection

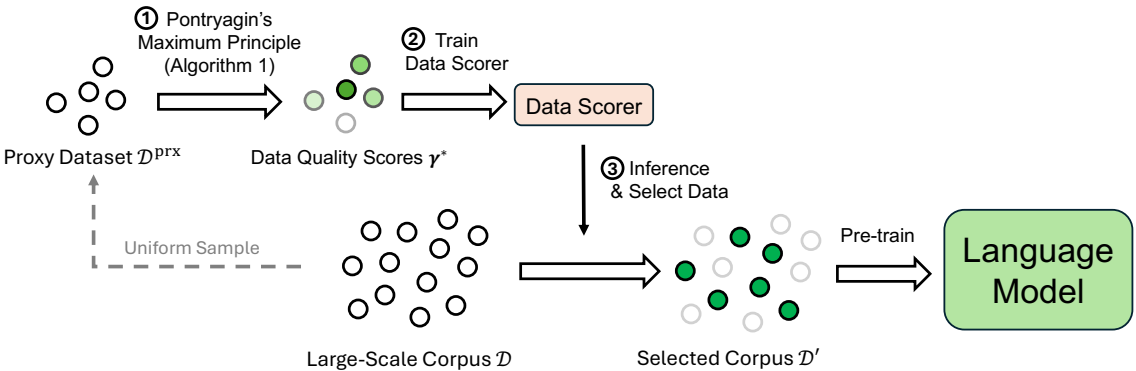
Theorem (PMP Conditions for Data Selection)

Let γ^* solve the optimization problem, and θ_t^* denote the LM parameters trained with γ^*. For $0 \leq t < T$, there exists a vector $\lambda_t^* \in \mathbb{R}^N$ such that:

$$\theta_{t+1}^* = \theta_t^* - \eta \nabla L(\theta_t^*, \gamma^*),\ \theta_0^* = \theta_0$$
$$\lambda_t^* = \lambda_{t+1}^* + \nabla J(\theta_t^*) - \eta \nabla^2 L(\theta_t^*, \gamma^*) \lambda_{t+1}^*,\ \lambda_T^* = \nabla J(\theta_T^*)$$
$$\gamma^* = \arg\max_{\gamma} \sum_{n=1}^{|\mathcal{D}_{trn}|} \gamma_n \left[\sum_{t=0}^{T-1} {\lambda_{t+1}^*}^\top \nabla l(x_n, \theta_t^*) \right],\ \gamma \in U$$

- λ_t^* defines a "target vector" representing ideal gradient direction
- Higher quality scores assigned to data points whose gradients align with the target vector
- This forms a complete equation system that can be solved to derive optimal data weights

PDS Framework Overview



- Compute data quality scores on a proxy dataset via PMP-Solver
- Train a data scorer model to predict quality scores
- Select high-quality data for pre-training LMs

PMP-Solver Algorithm

Algorithm PMP-Solver Algorithm (Height Adjusted)

Require: LM learning rate η, Outer loop learning rate α, Outer epochs T_o, Training data $\mathcal{D}_{\text{trn}}$, Downstream loss $J(\theta)$, Training steps T, Projection function Proj$[\cdot]$, Model initialization θ_0

Ensure: Data quality scores γ^*

Initialize $\gamma = [\gamma_1, \gamma_2, \cdots, \gamma_{|\mathcal{D}_{\text{trn}}|}] \leftarrow \left[\frac{1}{|\mathcal{D}_{\text{trn}}|}, \frac{1}{|\mathcal{D}_{\text{trn}}|}, \cdots, \frac{1}{|\mathcal{D}_{\text{trn}}|}\right]$
for $i = 1$ to T_o **do**
  for $t = 0, 1, \cdots, T-1$ **do** ▷ Forward inner loop
    $\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(\theta_t, \gamma)$
  end for
  $\lambda_T \leftarrow \nabla J(\theta_T)$
  for $t = T-1, T-2, \cdots, 1$ **do** ▷ Reverse inner loop
    $\lambda_t \leftarrow \lambda_{t+1} + \nabla J(\theta_t) - \eta \nabla^2 L(\theta_t, \gamma) \lambda_{t+1}$
  end for
  for $n = 1, 2, \cdots, |\mathcal{D}_{\text{trn}}|$ **do**
    $\gamma_n \leftarrow \gamma_n + \alpha \sum_{t=0}^{T-1} \lambda_{t+1}^\top \nabla l(x_n, \theta_t)$
  end for
  $\gamma \leftarrow$ Proj $[\gamma]$
end for
return γ

- **Bi-level optimization:** Outer loop updates γ while inner loops compute θ_t and λ_t
- Uniform score initialization followed by iterative refinement
- "Soft" update strategy for stability

Computing Target Vectors and Gradient Alignment

- **Forward inner loop:**

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla L(\theta_t, \gamma)$$

Trains the model with weighted loss based on current γ scores

- **Reverse inner loop:**

$$\lambda_t \leftarrow \lambda_{t+1} + \nabla J(\theta_t) - \eta \nabla^2 L(\theta_t, \gamma) \lambda_{t+1}$$

Computes the target vectors from future to current steps

- **Quality score update:**

$$\gamma_n \leftarrow \gamma_n + \alpha \sum_{t=0}^{T-1} \lambda_{t+1}^\top \nabla l(x_n, \theta_t)$$

Increases scores for data points with gradients aligned to target vectors

Efficient Implementation of PMP-Solver

- **Computational challenges:**
  - Computing Hessian matrix for $\nabla^2 L(\theta_t, \gamma)$ is expensive
  - Storing all model parameters θ_t for $t = 0$ to $t = T - 1$ requires large memory
  - Running multiple outer epochs with large models is computationally intensive
- **Practical implementation:**

$$\gamma^* = \frac{1}{M} \sum_{m=1}^{M} \text{PMP-Solver}(\mathcal{D}_{\text{trn}} = \mathcal{D}_{\text{prx}}, T = T^{\text{prx}}, \theta_0 = \theta_0^{(m)}, T_o = 1)$$

  where $\theta_0^{(m)}$ are checkpoints at different stages: $[10K, 20K, 30K, 40K, 50K]$ steps
- **Key optimizations:**
  - Small proxy LM with $N^{\text{prx}} \ll N$ parameters
  - Fewer training steps $T^{\text{prx}} \ll T$
  - Mini-batch SGD with small batch size
  - Just one outer epoch per checkpoint

Data Scorer: Transferring Quality Scores to Full Dataset

- **Data scorer objective:**

$$\phi^*, w^*, b^* = \arg\min_{\phi,w,b} \sum_{n=1}^{|\mathcal{D}_{\text{prx}}|} (w^\top \overline{h}(x_n^{\text{prx}}, \phi) + b - \gamma_n^*)^2$$

- **Implementation details:**
  - Fine-tune a small language model (125M parameter Fairseq-Dense model)
  - Represent each instance by averaging output hidden states: $\overline{h}(x, \phi)$
  - Train linear head w, b to predict quality scores using MSE loss
- **Inference on full dataset:**
  - Apply trained scorer to entire pre-training corpus $\mathcal{D}_{\text{trn}}$
  - Quality score for each instance: $\gamma(x_n) = w^{*\top} \overline{h}(x_n, \phi^*) + b^*$

Final Data Selection Strategy

- **Gumbel-Top-K sampling for diversity:**

$$\mathcal{D}_{\text{select}} = \text{Top-}K\{\gamma(x_n) - \tau \log(-\log(u_n)) \mid x_n \in \mathcal{D}_{\text{trn}}, 1 \leq n \leq |\mathcal{D}_{\text{trn}}|\}$$

  where:
  - $u_n \sim \text{Uniform}(0, 1)$ adds randomness
  - $\tau = 0.1$ controls noise strength
  - $K = r|\mathcal{D}_{\text{trn}}|$ with selection ratio $r = 0.4$

- **Benefits of probabilistic selection:**
  - Ensures diversity in the selected data
  - Prevents overfitting to specific patterns
  - Balances exploration and exploitation

Experimental Setup

- **Data:**
  - Pre-training corpus ($\mathcal{D}_{\text{trn}}$): CommonCrawl from RedPajama
  - Downstream loss function ($J(\cdot)$): LIMA (1,030 diverse instruction-response pairs)
  - Evaluation: OLMo benchmark tasks and MMLU (zero-shot)
- **Models:** Mistral architecture with 160M, 470M, 1B, and 1.7B parameters
- **PDS implementation:**
  - Proxy dataset: 160K instances (0.2B tokens) sampled from CommonCrawl
  - Proxy LM: 160M parameters trained for 50K steps
  - PMP-Solver: Inner loops with 100 steps, batch size 256, learning rate 0.008
  - Data scorer: 125M parameter Fairseq-Dense model
- **Pre-training:** 100K steps, batch size 512, max length 1024 (50B tokens)

Baseline Methods

- **Conventional/RedPajama:**
  - Pre-training LM on 50B tokens uniformly sampled from $\mathcal{D}_{\text{trn}}$
  - No data selection or filtering applied
- **RHO-Loss (Reducible Held-Out Loss):**
  - Selects data with high reducible losses
  - RHO-Loss$(x) = \mathcal{L}_{\text{init}}(x) - \mathcal{L}_{\text{final}}(x)$
- **DSIR (Data Selection using Instance-level Relevance):**
  - Selects data with high n-gram overlap with instances in LIMA
  - Focuses on content similarity to target distribution
- **IF-Score (Influence Function Score):**
  - Selects data with high influence scores
  - IF-Score$(x) = \nabla_\theta \mathcal{L}(x)^T \cdot \nabla_\theta J(D_{\text{val}})$
  - Measures influence through gradient alignment

Detailed OLMo Benchmark Results

Method	HS	LAMB	Wino.	OBQA	ARC-e	ARC-c	PIQA	SciQ	BoolQ	Avg.
Model Size = 470M										
Conv.	36.7	41.4	52.4	**30.4**	44.8	25.2	61.0	70.6	60.4	47.0
RHO-Loss	36.6	42.4	53.0	29.4	43.7	25.2	60.4	72.8	59.8	47.0
DSIR	36.4	42.6	51.7	29.8	46.0	24.7	61.0	72.0	55.8	46.7
IF-Score	36.6	41.8	**53.4**	29.6	44.7	25.1	60.8	68.8	58.7	46.6
PDS	**37.9**	**44.6**	52.3	29.8	**46.5**	**25.8**	**61.8**	**73.8**	**61.4**	**48.2**
Model Size = 1B										
Conv.	39.9	47.6	52.4	30.6	49.3	26.4	63.1	73.7	60.9	49.3
RHO-Loss	39.8	47.0	53.0	30.8	48.0	26.4	62.9	71.1	**61.0**	48.9
DSIR	40.8	47.8	53.0	31.2	49.8	26.8	62.7	76.6	58.0	49.6
IF-Score	39.4	47.0	52.6	28.6	49.4	26.4	63.5	74.0	60.5	49.0
PDS	**42.1**	**48.8**	**54.0**	**33.4**	**51.3**	**28.0**	**64.1**	**78.5**	58.7	**51.0**

Table: Accuracy on OLMo benchmark tasks. PDS consistently outperforms baseline methods.

MMLU Performance and Language Modeling Results

Size	Method	0-shot	PPL
470M	Conv.	27.6	34.8
	RHO-Loss	28.4	33.0
	DSIR	28.0	34.0
	IF-Score	28.4	31.1
	PDS	**28.9**	**27.1**
1B	Conv.	29.7	26.1
	RHO-Loss	30.2	24.9
	DSIR	30.0	25.3
	IF-Score	30.7	23.6
	PDS	**31.4**	**20.5**

Table: MMLU results showing 0-shot accuracy and perplexity (PPL) on ground truth answers.

- **MMLU improvements:**
  - PDS shows +1.3% gain on 470M model
  - PDS shows +1.7% gain on 1B model
  - Lower perplexity indicates higher confidence on correct answers
- **Language modeling quality:**
  - PDS achieves significant reduction in perplexity
  - 1B model: 21% reduction (26.1 $\rightarrow$ 20.5)
  - Indicates better capture of language patterns

Language Modeling Performance on DCLM

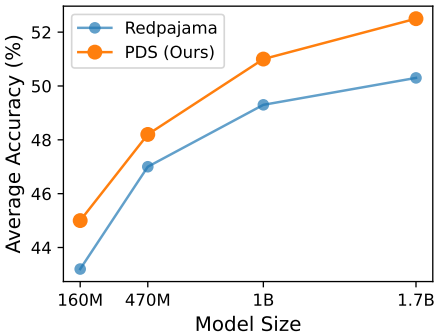


Figure: Test losses on DCLM corpus for models of different sizes, showing consistent improvements with PDS across all model scales.

- **DCLM dataset:**
  - High-quality corpus curated with complex pipelines
  - Verified for diversity and knowledge coverage
  - Challenging benchmark for general language modeling
- **Consistent improvements:**
  - PDS shows lower loss across all model sizes
  - 160M: 9% reduction
  - 470M: 22% reduction
  - 1B: 21% reduction
  - 1.7B: 27% reduction
- PDS provides **principled** alternative to complex data curation pipelines

Extrapolating PDS Benefits to Larger Models

	N	D	Conv.	PDS
GPT-3	175B	300B	2.882	**2.872**
Llama	6.7B	1.0T	2.942	**2.896**
Llama 2	70B	2.0T	2.877	**2.855**
Llama 3.1	405B	15T	2.851	**2.838**

Table: Test loss extrapolation using Scaling Laws. We predict loss when model size N and trained tokens D match those of leading models.

- **Scaling law form:**

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$$

- **PDS improves scaling constants:**
  - Higher α ($0.397 \rightarrow 0.518$): Better parameter scaling
  - Lower B ($7.5 \times 10^5 \rightarrow 1.8 \times 10^5$): Better data utilization
  - Slightly lower β ($0.651 \rightarrow 0.585$): Better starting point but slower data scaling
- **Practical implications:**
  - Benefits consistent across all model scales
  - Extrapolates to largest LLMs in production today
  - Helps mitigate "running out of data" challenge
  - Same test loss with less computation

Computational Efficiency and Training Acceleration

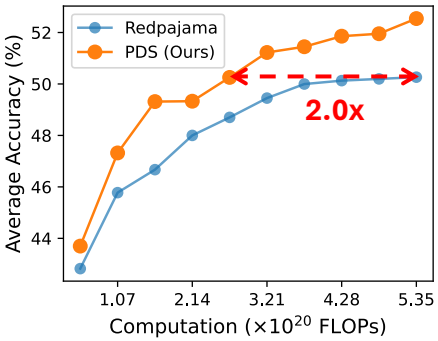


Figure: PDS achieves same performance with 2x fewer training steps, showing significant computational savings.

- **Training acceleration:**
  - 2x faster convergence in terms of training FLOPs
  - Consistent acceleration across model sizes
  - Same final performance with half the compute
- **PDS computational overhead:**
  - Only 1/9 of the computational cost of pre-training
  - Proxy γ-solver: 4.9×10^{19} FLOPs
  - Data scorer: 6.3×10^{18} FLOPs
  - 1.7B pre-training: 5.1×10^{20} FLOPs
- **Benefits:**
  - One-time upfront cost for data selection
  - Selected corpus reusable for multiple models
  - Seamless integration with existing pipelines

Data-Constrained Settings: When Data is Limited

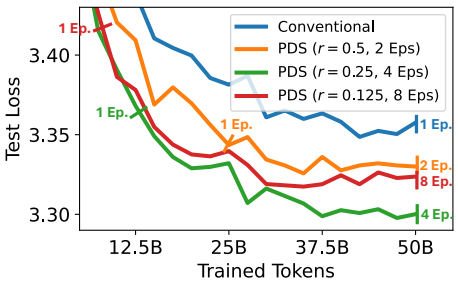


Figure: Test losses on DCLM corpus in the data-constrained setting with different selection ratios and training epochs.

- **Experimental setup:**
  - Restricting $\mathcal{D}_{\text{trn}}$ to 50B tokens
  - Apply PDS with selection ratios $r \in [0.125, 0.25, 0.5]$
  - Training for $[8, 4, 2]$ epochs respectively
- **Key findings:**
  - Selecting $1/4$ data with PDS and training for 4 epochs achieves lowest test loss
  - Consistent with findings from data-constrained LM literature
  - PDS reduces data requirements by 1.8x
  - Conventional training would need additional 42B tokens to match PDS performance
- **Implication:** Critical as high-quality web data becomes exhausted

When in Training is Data Quality Most Important?

Method	Corr.	Acc.
IF-Score	0.32	43.0
PDS ($T^{\text{prx}} = 1$)	0.54	44.6
PDS (50K-100K)	0.48	43.4
PDS (10K-50K)	0.52	**45.0**

Table: Impact of training stages and trajectory length on PDS performance. Correlation with exact solution and downstream accuracy.

- **Early vs. late training stages:**
  - Early stages (10K-50K steps): +1.6% accuracy
  - Late stages (50K-100K): Only +0.2% accuracy
  - Early training involves critical parameter space exploration
- **Trajectory length importance:**
  - Single-step gradients ($T^{\text{prx}} = 1$): Less effective
  - Long-range dynamics ($T^{\text{prx}} = 100$): Captures crucial learning trajectory
  - Multiple checkpoints capture different optimization phases
- **Implication:** Focus data quality efforts on early training for maximum benefit

Efficient Implementation of PMP Solver

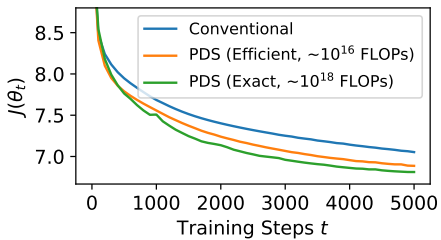


Figure: Comparison between exact and efficient PMP implementations in terms of performance vs. computational cost.

- **Computational challenges:**
  - Exact PMP solution computationally expensive
  - Must balance solution quality with practical overhead
  - Need efficient approximation for production use
- **Efficient implementation:**
  - Single outer epoch instead of convergence
  - Small proxy LM instead of full model
  - Limited training trajectory (100 steps)
  - Selected checkpoints for key training stages
- **Results:**
  - Preserves 92% of exact solution effectiveness
  - Reduces computation by $\sim 95\%$
  - Makes PDS practical for large-scale applications

Summary of Experimental Findings

- **Performance improvements:**
  - Consistent +1-2% accuracy across model sizes and tasks
  - Average gain of 1.8% (470M) and 1.7% (1B) on OLMo tasks
  - Stronger improvement on MMLU (+1.3% for 470M, +1.7% for 1B)
  - 21-27% reduction in perplexity on high-quality text
- **Efficiency gains:**
  - 2x faster convergence in training
  - 1.8x reduction in data requirements
  - Minimal overhead (1/9 of pre-training cost)
  - Benefits extrapolate to largest models
- **Critical insights:**
  - Early training stages most impacted by data quality
  - Long-range dynamics crucial for effective selection
  - Optimal strategy: select 1/4 of data, train for 4 epochs
  - Efficient implementation preserves most benefits

Outline

RegMix: Motivation and Key Idea

- **Challenge:** Data mixture significantly impacts LLM performance, but determining optimal mixtures is difficult
- **Key Insight:** Rank invariance of data mixtures across model scales
  - The ranking of data mixtures by performance holds across model sizes
  - Small-scale models can predict effective mixtures for large-scale models
- **RegMix Approach:**
  - Formulate data mixture selection as a regression task
  - Train many small models on diverse data mixtures
  - Fit regression model to predict performance of unseen mixtures
  - Use best predicted mixture for large-scale model training
- **Advantage:** $10\times$ less computation than prior methods like DoReMi

Key Insight: Small to Large Model Transfer

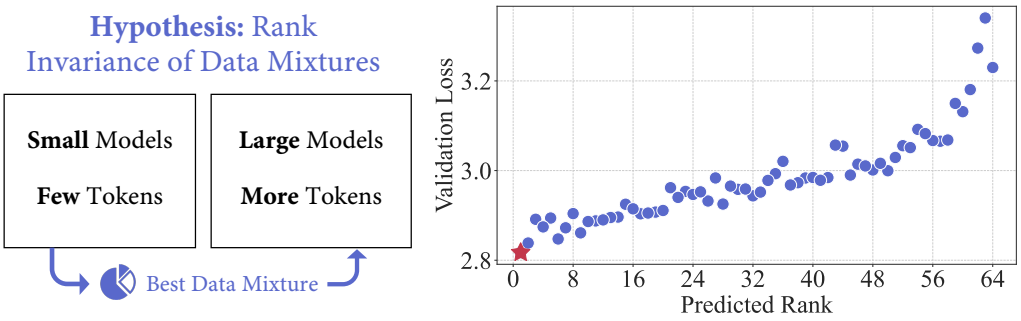
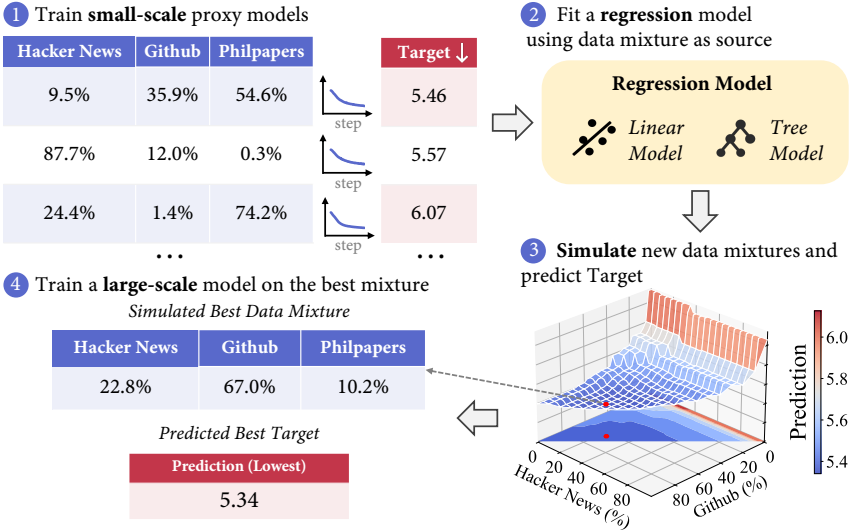


Figure: **Left**: The rank invariance hypothesis across model scales. **Right**: RegMix successfully identifies the best data mixture (red star) for 1B parameter models using only 1M parameter proxy models.

RegMix: Methodology Overview



RegMix: Training Small-Scale Proxy Models

- **Goal:** Sample diverse data mixtures to cover the solution space efficiently
- **Approach:** Use Dirichlet distribution based on token distribution
  - Sample weights from 0% to 100% for each domain
  - Ensure mixtures reflect realistic token availability
  - Apply multipliers (0.1 to 5.0) to create diverse sparsity patterns
- **Implementation:**
  - Train 512 models with 1M parameters on 1B tokens each
  - Evaluate models on downstream tasks or domain losses
  - Collect data mixture weights and corresponding performance

RegMix: Regression Models and Data Domains

- **Regression models:**
  - Linear regression with L2 regularization
  - LightGBM gradient-boosting algorithm
- **Input:** Domain weights of data mixture
- **Output:** Target performance metric

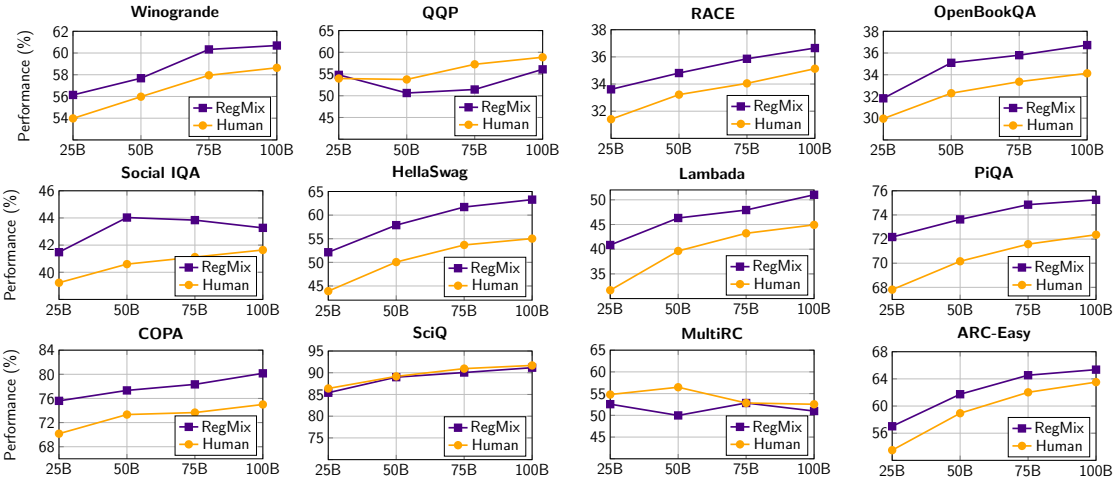
Table: Key domains in the Pile dataset used for experiments

Component	**Size (GiB)**
Pile-CC	227.12
PubMed Central	180.55
ArXiv	112.42
GitHub	95.16
FreeLaw	76.73
Stack Exchange	64.39
Wikipedia (en)	19.13
HackerNews	7.80

RegMix: Simulation and Prediction for Large-Scale Training

- **Simulation process:**
  - Generate millions of potential data mixtures
  - Apply trained regression model to predict performance
  - Identify mixtures with best predicted performance
  - For robust results, average top 100 predicted mixtures
- **Computational efficiency:**
  - Simulation of 1,000,000 data mixtures takes ¡10 CPU seconds
  - Total RegMix compute is 2% of training one 1B parameter model
  - Parallelizable across many small models rather than sequential training
- **Large-scale model training:**
  - Apply predicted optimal mixture to models with 1B-7B parameters
  - Training for 25B-100B tokens (25-100$\times$ more than proxy models)
  - Model size up to 1000$\times$ larger than proxy models

Experimental Results: Performance on Downstream Tasks



Key Findings from RegMix Experiments

- **Data mixture impact:**
  - Up to 14.6% difference in task performance from mixture choice alone
  - Mixture effects persist across model scales (1M to 7B parameters)
- **Domain importance:**
  - Web corpora (e.g., CommonCrawl) show strongest positive correlation with performance
  - Wikipedia is less impactful than commonly believed
  - Complex interactions between domains that defy intuition
- **Scaling behavior:**
  - RegMix consistently outperforms human selection across all scales
  - Matches or exceeds DoReMi with $10\times$ less computation
  - Benefits transcend scaling laws - good mixtures remain beneficial at scale

Practical Implications

- **Efficient resource utilization:**
  - Small models can effectively predict optimal mixtures for large models
  - Parallelizable approach fits well with modern compute infrastructure
  - Total compute cost is just 2% of final model training
- **Better than human intuition:**
  - Automatically discovers non-obvious domain relationships
  - Consistently outperforms mixtures selected by human experts
  - Takes into account complex interactions between domains
- **Practical deployment:**
  - Can be integrated into existing LLM training pipelines
  - Addresses the challenge of optimizing for massive datasets (15T+ tokens)
  - Facilitates principled domain selection as data sources continue to expand

Evaluating RegMix: Regression Prediction

- **Goal:** Evaluate RegMix's ability to predict effects of unseen data mixtures
- **Setup:**
  - **Datasets:** 17 copyright-free domains from the Pile dataset
  - **Models:** Linear and LightGBM regression models (target: Pile-CC validation loss)
  - **Training:** Fit using artifacts from 512 $\times$ 1M parameter models (1B tokens)
  - **Evaluation:** Test on 256 $\times$ unseen mixtures for small models (1M, 60M) and 64 $\times$ unseen mixtures for 1B models
- **Metrics:**
  - Spearman Rank Correlation (ρ) - tests rank preservation
  - Mean Squared Error (MSE) - measures prediction accuracy

Regression Performance Across Model Scales

Test On	*1M*		*60M*	*1B*
Method	ρ (↑)	**MSE (↓)**	ρ (↑)	ρ (↑)
Linear	90.08	0.13	89.26	88.01
LightGBM	98.45	0.04	98.64	97.12

Table: Regression performance on unseen mixtures across model sizes. 1M models trained with 1B tokens achieve 97.12% correlation on mixtures of 1B models with 25B tokens, confirming rank invariance.

Optimizing Small-Scale Training: Models vs. Tokens

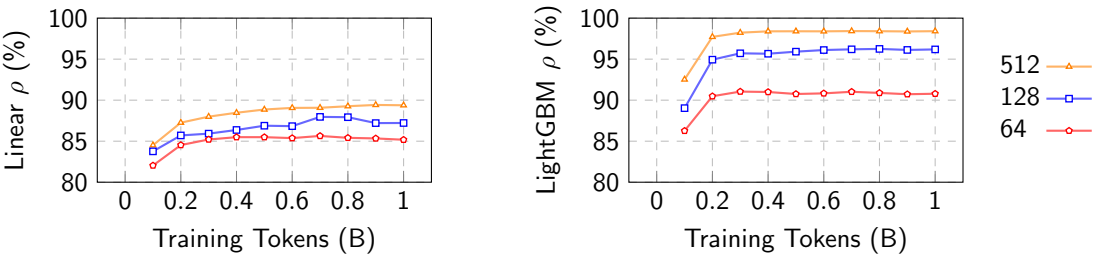


Figure: The plot of Spearman Rank Correlation ρ between the predicted ranks and true ranks of Linear regression (**Left**) and LightGBM regression (**Right**) across different training tokens and different number of proxy models. As shown, increasing the number of proxy models significantly boosts ρ, while adding more training tokens has diminishing returns.

Impact of Data Mixture on Performance

Benchmark	Worst Model	Best Model	Δ
Social IQA	32.4	33.9	1.5
HellaSwag	33.0	43.4	10.4
PiQA	60.2	69.0	8.8
OpenBookQA	25.8	31.2	5.4
Lambada	18.9	33.5	14.6
SciQ	76.7	82.9	6.2
ARC Easy	44.9	52.2	7.3
COPA	61.5	70.5	9.0
RACE	27.9	32.5	4.6
LogiQA	23.2	27.7	4.5
QQP	48.0	59.7	11.7
WinoGrande	50.3	53.2	2.9
MultiRC	47.6	55.7	8.1
Average	43.7	47.9	4.2

Table: Performance comparison of worst vs. best data mixtures.

Domain Importance for Downstream Tasks

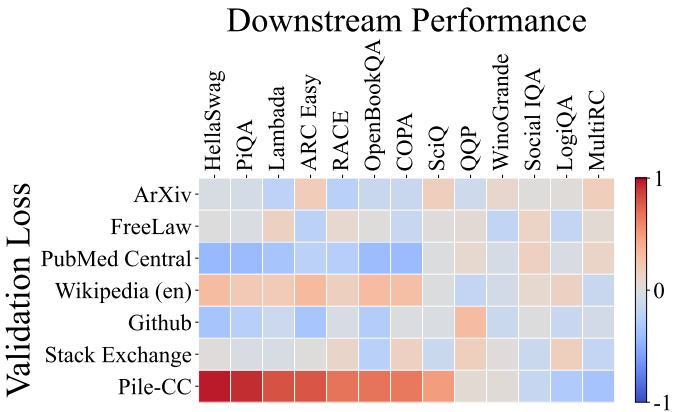


Figure: Correlation between validation loss by domains of the Pile and downstream performance. Pile-CC (web data) shows strongest correlation with most downstream tasks.

Web Domain Correlation with Tasks

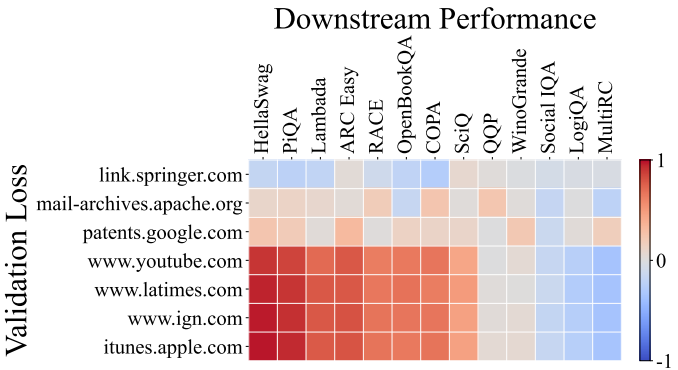


Figure: Correlation between validation loss by URL domain within Pile-CC and downstream performance. Most web domains (e.g., www.ign.com shown here) mirror the correlation pattern of Pile-CC.

RegMix vs. Other Data Selection Methods

Benchmark	Human	DoReMi	PPL	ODM	Pile-CC	RegMix
Social IQA	33.6	33.4	33.3	33.7	33.2	**33.8**
HellaSwag	37.4	43.4	43.1	37.2	44.1	**44.2**
PiQA	65.0	68.3	68.5	64.4	69.2	**69.3**
OpenBookQA	28.2	30.3	30.3	30.0	**31.1**	30.3
Lambada	29.8	32.1	**35.4**	29.6	33.2	34.2
SciQ	80.1	81.6	78.6	79.8	81.8	**82.8**
ARC Easy	49.4	50.6	50.5	47.9	**51.8**	51.7
ARC Challenge	26.3	26.1	25.9	25.6	**26.7**	25.7
COPA	66.7	68.5	69.2	68.2	65.8	**70.2**
RACE	29.0	31.3	31.5	29.7	**31.8**	31.3
LogiQA	25.5	26.4	27.5	25.6	**27.6**	25.8
QQP	52.4	56.6	50.0	53.1	57.0	**58.3**
WinoGrande	**53.1**	52.2	52.8	51.8	52.1	**53.1**
MultiRC	**54.3**	53.8	50.4	53.3	50.3	51.7
Est. FLOPs	0	3.7e19	1.8e19	0	0	3.5e18
Average	45.1	46.8	46.2	45.0	46.8	**47.3**
Best On	2 / 14	0 / 14	1 / 14	0 / 14	5 / 14	**7 / 14**

Table: RegMix achieves best average performance and wins on 7/14 tasks with only 10% of DoReMi's compute.

Out-of-Distribution Effectiveness

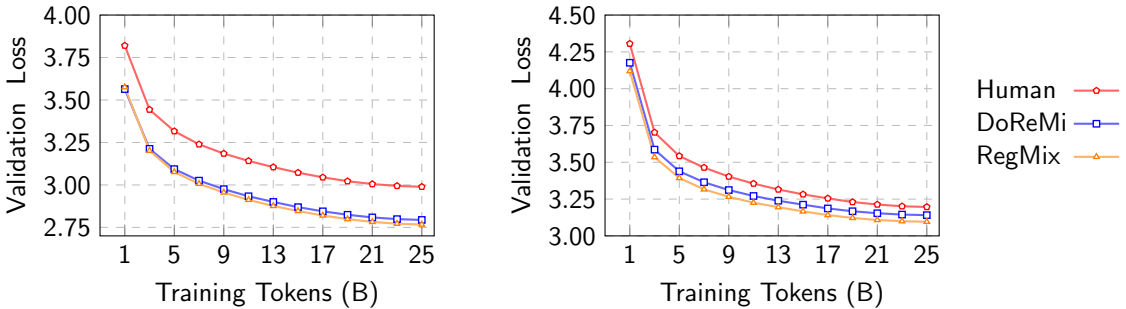


Figure: **Left**: The validation loss on Pile-CC of different methods with Pile-CC in the pre-training corpus. **Right**: The validation loss on Pile-CC excluding Pile-CC in the pre-training.

Code Domain Interaction Example

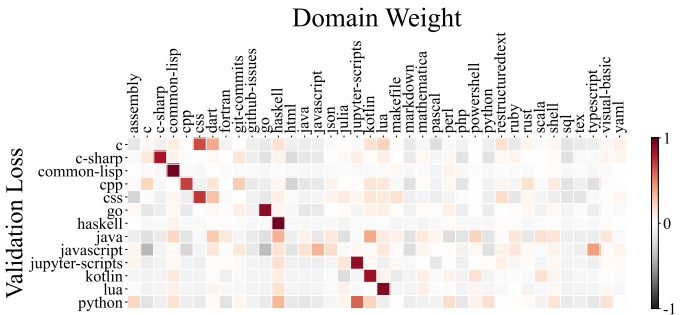


Figure: Correlations between validation losses and training domain weights for code datasets, showing complex relationships difficult for humans to intuit.

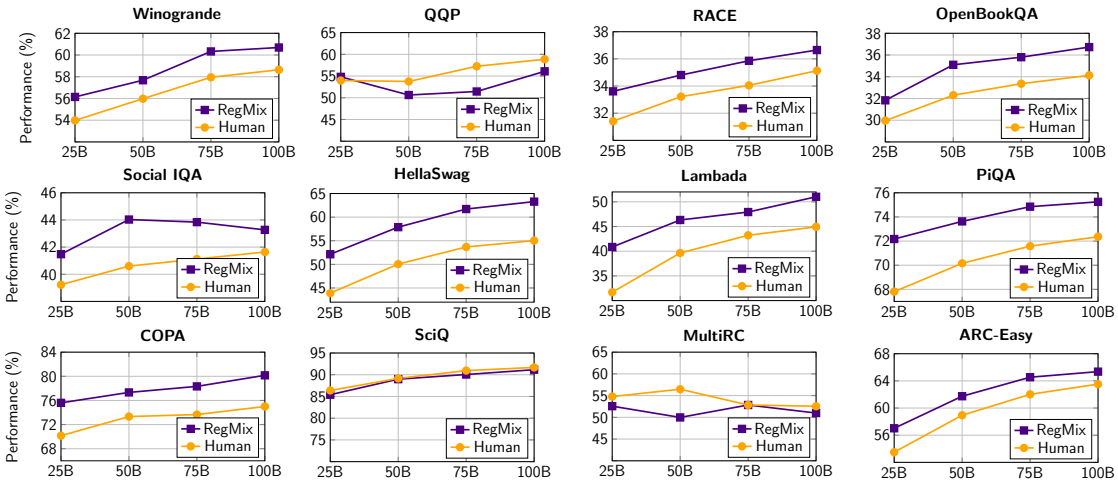
Scaling to 100 Domains

- **Challenge:** Clustering web content into meaningful domain representations
- **Approach:** Define domains by base URLs from FineWeb dataset
- **Examples:** articles.latimes.com, blogs.wsj.com, en.wikipedia.org, etc.
- **Methodology:**
  - Train 1,000 small-scale models (1M parameters) across different mixtures
  - Fit regression model to predict optimal data mixture
  - Evaluate on models with 1M and 60M parameters

Test On	*1M*		*60M*
Method	ρ **(↑)**	**MSE (↓)**	ρ **(↑)**
Linear	90.33	0.12	88.64
LightGBM	99.53	0.02	98.80

Table: RegMix extends effectively to 100 domains with high rank correlation.

Performance on Downstream Tasks on 7B Parameter Models



Outline

Not All Tokens: Overview

- **Key Insight:** Token-level training dynamics reveal not all tokens are equally useful
- **Solution:** Selective Language Modeling (SLM) to focus on valuable tokens

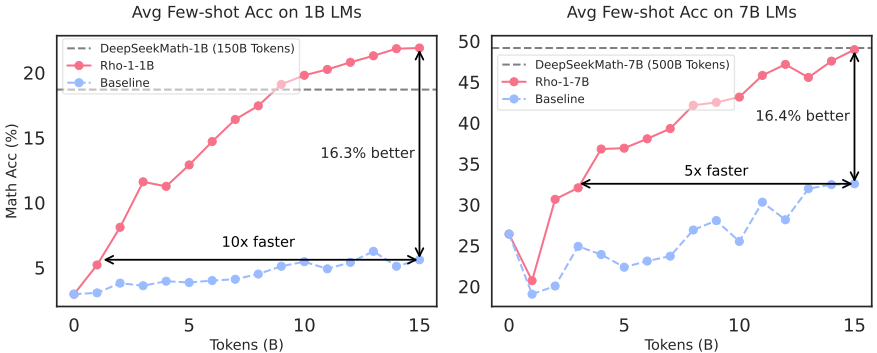


Figure: SLM improves accuracy on GSM8k and MATH.

Token-Level Training Dynamics

- Many tokens exhibit persistent fluctuations and resist convergence

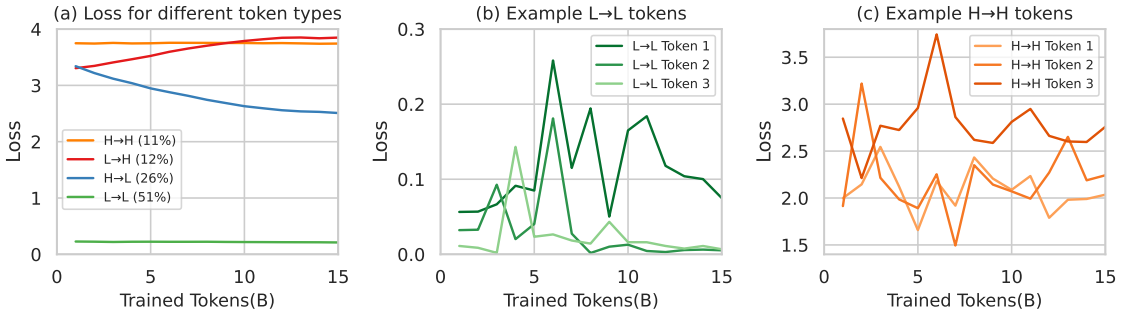


Figure: Training dynamics of four token categories during pretraining. (a) Loss of H→H, L→H, H→L, and L→L tokens. (b) and (c) show cases of fluctuating tokens' loss in L→L and H→H.

Selective Language Modeling (SLM): Pipeline

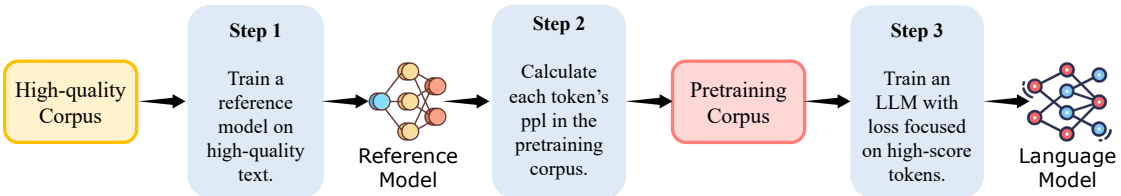


Figure: The pipeline of Selective Language Modeling (SLM): (1) Train a reference model on high-quality data. (2) Score each token's loss using the reference model. (3) Selectively train on tokens with higher scores.

Selective Language Modeling: Reference Model

- Step 1: Train a reference model (RM) on curated high-quality dataset
- Step 2: Compute reference loss for each token in the pretraining corpus

$$\mathcal{L}_{\text{RM}}(x_i) = -\log P(x_i|x_{<i})$$

- This establishes the baseline for token selection

Selective Language Modeling: Token Selection

- Traditional CLM uses cross-entropy loss on all tokens:

$$\mathcal{L}_{\text{CLM}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(x_i|x_{<i};\theta)$$

- SLM focuses on tokens with high excess loss compared to reference model:

$$\mathcal{L}_{\Delta}(x_i) = \mathcal{L}_{\theta}(x_i) - \mathcal{L}_{\text{RM}}(x_i)$$

$$\mathcal{L}_{\text{SLM}}(\theta) = -\frac{1}{N * k\%}\sum_{i=1}^{N} I_{k\%}(x_i) \cdot \log P(x_i|x_{<i};\theta)$$

- Token selection ratio $k\%$ determines proportion of tokens to include
- Easy to implement by ranking tokens in a batch by excess loss

Key Results

- Both 1B and 7B models trained with SLM outperform CLM baselines by 16%+ on GSM8k and MATH
- SLM reaches baseline accuracy up to 10x faster
- ρ-7B matches DeepSeekMath-7B using only 15B tokens vs 500B tokens
- After fine-tuning, ρ-1B achieves 40.6% on MATH (first 1B LM to exceed 40%)
- ρ-7B achieves 51.8% on MATH after fine-tuning
- In general continual pretraining, SLM improves Tinyllama-1B by 6.8% across 15 benchmarks

Experiment Overview

- Mathematical domain experiments
  - Continual pretraining on OpenWebMath (14B tokens)
  - Few-shot CoT reasoning evaluation
  - Tool-integrated reasoning evaluation via fine-tuning
- General domain experiments
  - Continual pretraining on 80B tokens (SlimPajama, StarCoderData, OpenWebMath)
  - Evaluation across 15 benchmarks
- Self-reference experiments and ablation studies

Experimental Setup: Reference Models

- Mathematical reference model
  - 0.5B high-quality math tokens
  - Sources: GPT synthetic data, manually curated datasets
  - Trained for 3 epochs
- General reference model
  - 1.9B tokens from open-source datasets
  - Sources: Tulu-v2, OpenHermes-2.5
  - Trained for 3 epochs
- Learning rates: 5e-5 (1B models), 1e-5 (7B models)
- Sequence lengths: 2048 (1B models), 4096 (7B models)

Experimental Setup: Training Details

- Math pretraining models
  - Base models: Tinyllama-1.1B, Mistral-7B
  - Learning rates: 8e-5 (1.1B), 2e-5 (7B)
  - Hardware: 32 $\times$ H100 80G GPUs
  - Training time: 3.5h (15B tokens, 1.1B model), 18h (15B tokens, 7B model)
- General domain pretraining
  - Learning rate: 1e-4 for Tinyllama-1.1B
  - 80B tokens in 19 hours
- Batch size: 1M tokens (both domains)
- Token selection ratio: 60% (1.1B model), 70% (7B model)

Math Pretraining Results: Few-shot CoT

- SLM outperforms continual pretraining baselines
  - 16.5% average improvement on 1B models
  - 10.4% average improvement on 7B models
- Multiple epochs on OpenWebMath increases accuracy to 40.9%
- ρ-7B (15B tokens) matches DeepSeekMath-7B (500B tokens)

Model	**GSM8k**	**MATH**	**Avg**	**Tokens**
Tinyllama-CT	15.4%	4.8%	10.1%	15B
ρ-Math-1B	32.0%	20.6%	26.3%	15B
Mistral-CT	42.5%	26.3%	34.4%	15B
ρ-Math-7B	53.7%	35.4%	44.6%	15B

Table: Few-shot CoT results (simplified)

Math Pretraining Results: Tool-Integrated Reasoning

- Fine-tuned on 69k ToRA corpus
  - 16k GPT-4-generated tool-integrated reasoning trajectories
  - 53k answer-augmented samples using LLaMA
- State-of-the-art results
  - ρ-1B: 40.6% on MATH (first 1B model exceeding 40%)
  - ρ-7B: 51.8% on MATH
- Strong generalization to unseen tasks
  - 6.2% average improvement on ρ-Math-1B
  - 2.7% average improvement on ρ-Math-7B

General Pretraining Results

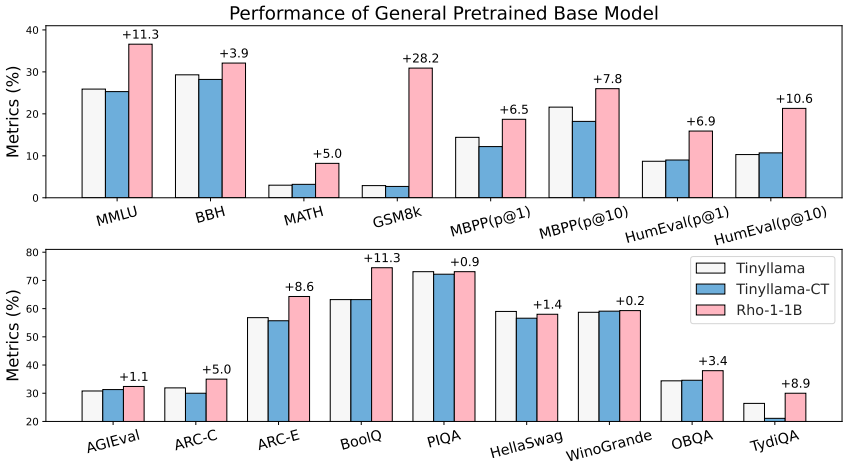


Figure: General pretraining results across 15 benchmarks

Self-Reference Results

- SLM can work without additional high-quality data
- Reference model trained only on OpenWebMath corpus
- Two scoring functions explored:
  - Reference model loss ($\mathcal{L}_{\text{RM}}$)
  - Information entropy of next token ($\mathcal{H}_{\text{RM}}$)
- Key findings:
  - OWM-trained reference model improved performance by $+2.4\%$
  - Information entropy scoring had similar improvement
  - Intersection of both scoring methods: $+3.3\%$ with 40% fewer tokens
  - PPile training with OWM reference: $+1.8\%$ with 30% fewer tokens

Loss Dynamics Analysis

- SLM shows greater loss reduction on selected tokens
- Selected-token pretraining substantially lowers downstream loss
- Traditional pretraining has less pronounced effect on downstream loss

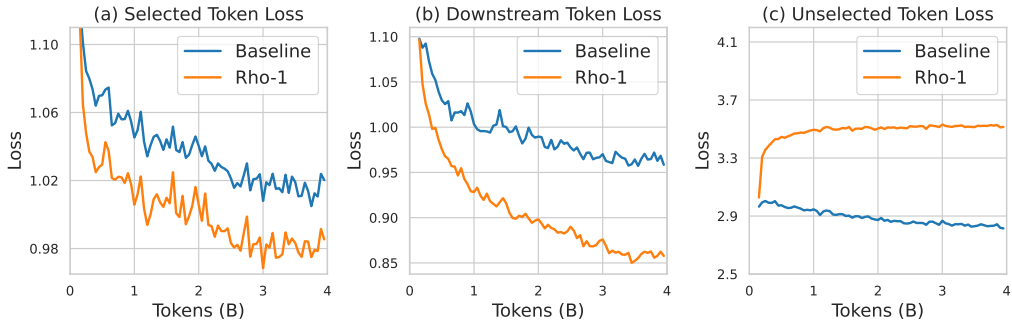


Figure: Loss dynamics during pretraining: (a,c) selected/unselected tokens, (b) downstream task

Loss of selected tokens correlates with downstream task performance

- Selected tokens positively impact performance
- Unselected tokens have negative impact
- Reducing loss across all tokens is not necessary for improved performance

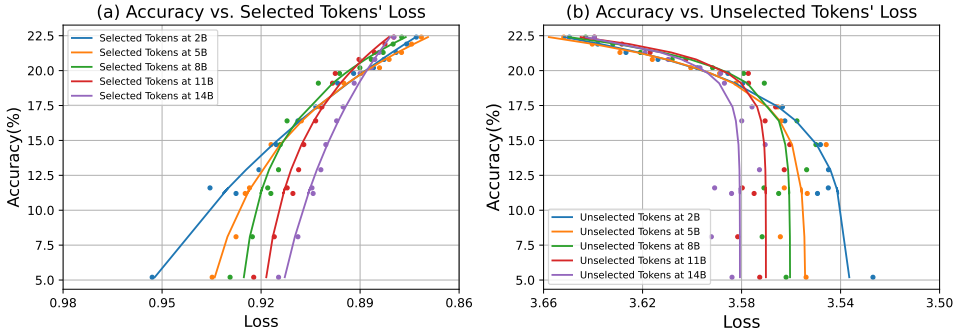
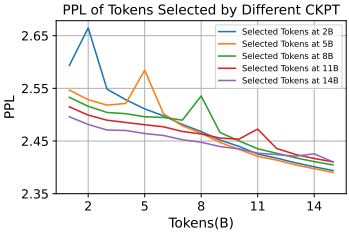
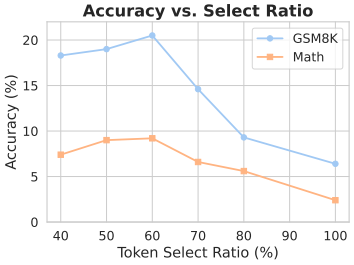


Figure: Relationship between token loss and downstream task performance

Token Selection Analysis





- Majority of selected tokens are closely related to mathematics
- Tokens selected by later checkpoints have:
  - Higher perplexity in later training stages
  - Lower perplexity in earlier stages
- Optimal token selection ratio: 60% of original tokens

Summary of Experimental Findings

- SLM significantly outperforms CLM baselines across domains
- Requires far fewer tokens to reach similar performance levels
- Self-reference approach enables training without external data
- Selected tokens show strong correlation with downstream performance
- Mechanisms explained through loss dynamics and token selection analysis
- Optimal token selection ratio around 60-70%

References

- (TMLR 07/2024) Data Selection Survey: [1]
- (ICLR 2025 Oral) Data Selection via Optimal Control: [3]
- (ICLR 2025 Spotlight) RegMix: [5]
- (ICML 2024 Poster) DOGE: [2]
- (NeurIPS 2024 Oral) Not All Tokens: [4]
- (NeurIPS 2023 Spotlight) DoReMi: [6]

References I


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.
A survey on data selection for language models.
Transactions on Machine Learning Research, 2024.
Survey Certification.


Simin Fan, Matteo Pagliardini, and Martin Jaggi.
DOGE: Domain reweighting with generalization estimation.
In *Forty-first International Conference on Machine Learning*, 2024.


Yuxian Gu, Li Dong, Hongning Wang, Yaru Hao, Qingxiu Dong, Furu Wei, and Minlie Huang.
Data selection via optimal control for language models.
In *The Thirteenth International Conference on Learning Representations*, 2025.

References II

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, yelong shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen.
Not all tokens are what you need for pretraining.
In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

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

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 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Many Thanks For Your Attention!