Velocitune: A Velocity-based Dynamic Domain Reweighting Method for Continual Pre-training

Zheheng Luo[♠†*] **Xin Zhang**[†*] **Xiao Liu**[†]
Haoling Li[◇†] **Yeyun Gong**[†] **Chen Qi**[†] **Peng Cheng**[†]
[♠]The University of Manchester [†]Microsoft [◇]Tsinghua University

Abstract

It is well-known that a diverse corpus is critical for training large language models, which are typically constructed from a mixture of various domains. In general, previous efforts resort to either sampling training data from different domains with static proportions or dynamically adjusting these proportions during training to optimise pretraining performance. However, few methods addressed the complexity of domain-adaptive continual pre-training. To fill this gap, we propose Velocitune, a novel framework that dynamically assesses learning velocity and adjusts data proportions accordingly, favouring slower learning domains while de-emphasising faster learning ones, which is guided by a scaling law to estimate the desired learning goal for each domain with a less associated cost. To evaluate the effectiveness of Velocitune, we conduct experiments on a dataset focused on reasoning tasks with CodeLlama, as well as on a corpus of system commands using Llama3 and Mistral. Velocitune achieves performance gains in both math and code reasoning tasks and command-line generation benchmarks. Further analysis reveals that key factors driving the effectiveness of Velocitune include target estimation and data ordering.

1 Introduction

Datasets used for pre-training language models (LMs) are typically composed of texts of various meta-attributes such as source and focus, referred to as different domains (Du et al., 2022; Azerbayev et al., 2023; Computer, 2023). The distinct characteristics of data from these varying domains, such as focus, quality, and quantity, affect the downstream performance of LMs differently (Rozière et al., 2024; Li et al., 2024). Consequently, numerous studies have explored the optimal combination of data from multiple domains to enhance LM performance. Llama3 (AI@Meta, 2024), GLaM (Du et al., 2022), and Lemma (Azerbayev et al., 2023) employ heuristic methods to iteratively test different ratios by training multiple proxy models and selecting the mixture that demonstrates the best downstream performance. However, these heuristic approaches demand costly large-scale experiments for effective exploration. As a result, recent research is focused on learning an optimal ratio by dynamically adjusting weights during proxy model training (Xie et al., 2024; Fan et al., 2023). For example, Doremi (Xie et al., 2024) implements a method in which a small reference model is initially trained, followed by training a small proxy model using group distributionally robust optimisation (Group DRO) (Sagawa et al., 2019a) to obtain optimised domain weights.

Domain-adaptive[1] continual pre-training, while sharing some similarities with from-scratch training, presents unique challenges that limit the effectiveness of existing domain reweighting methods. Many existing methods utilise small proxy models to estimate optimal domain weights, which are subsequently transferred to a larger model (Xie et al., 2024; Fan et al., 2023; Azerbayev et al., 2023). However, this approach poses challenges in domain-adaptive continual pre-training, as smaller versions of the base model often do not exist, making it difficult to estimate weights from proxy models. Another challenge is how to leverage the learning status. Previous methods rely on the difference between the current loss and the target loss of the model for a domain (Xie et al., 2024; Xia et al., 2023). However, this distance-based approach can result in overemphasis on specific domains, as domains with larger loss disparities may dispropor-

[*]The first two authors contribute equally. Work done during an internship at Microsoft. Correspondence: zhehengluo@gmail.com; xinzhang3@microsoft.com

[1]Here, "domain-adaptive" refers to optimising a model's ability and knowledge in a specific field or area. It should not be confused with the concept of "domain" of data as discussed in this paper.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 16644–16656

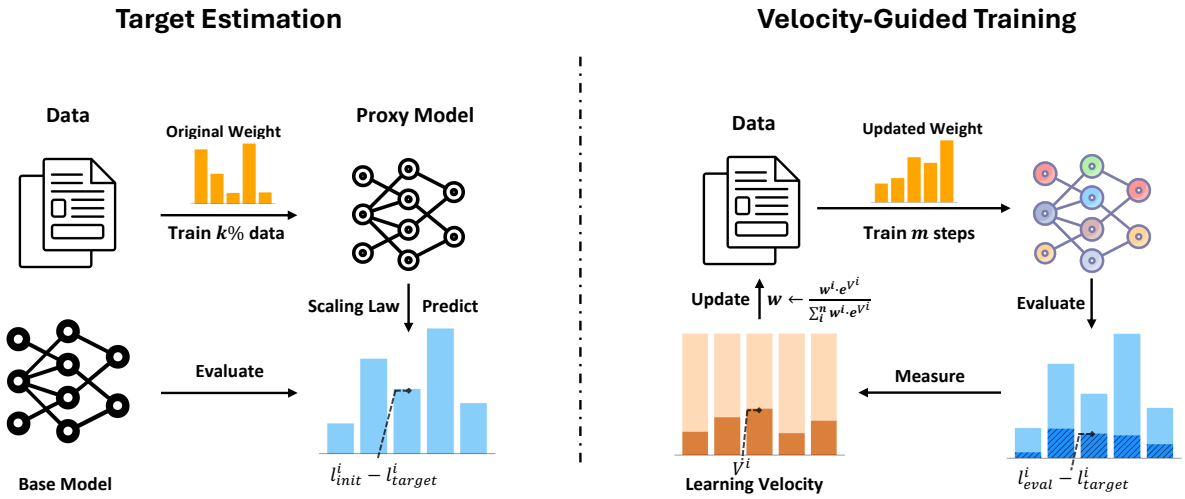

Figure 1: The overall pipeline of Velocitune. Initially, a proxy model is trained using the original domain weights on a subset of the data. Following this, the initial loss is collected by evaluating the base model, while the target loss is determined by extrapolating the evaluation loss of the proxy model. In the second phase, we calculate the learning velocity by rescaling the learning progress between the initial and target losses. This learning velocity is then used to update the domain weights effectively.

tionately influence learning. This can exacerbate imbalances across domains.

To address these issues, we introduce a novel framework, Velocitune, centred on the concept of learning velocity, as illustrated in Figure 1. In contrast to previous approaches that leverage the distance between the current loss and the target loss, Velocitune more effectively captures how fast models learn in each domain by establishing learning velocity. During training, domains exhibiting slower learning velocities are given increased weights, while those with faster velocities receive reduced weights, ensuring balanced learning progress.

To quantify learning velocity, it is crucial to determine both the model's already learnt expertise and its desired learning goal for each domain. In the absence of a smaller proxy model, we leverage the Chinchilla scaling law (Hoffmann et al., 2022), using the loss recorded on sub-sampled training data to cost-effectively predict the learning goal.

We evaluated the performance of Velocitune in two settings: continual pre-training CodeLlama 7B (Rozière et al., 2024) on a math and coding reasoning dataset, as well as Llama3 and Mistral (Jiang et al., 2023) on a system command knowledge corpus. Velocitune demonstrates an average improvement of 1.6% in eight math tasks and 3.8% in two coding tasks compared to the baseline trained with default weights. In addition, Velocitune outperforms the baselines in Llama3, showing improvements of 4.9% and 3.1%, and in Mistral, with gains of 4.4% and 2.6% in the CMDGen-NVIDIA and CMDGen-AMD tasks, respectively. In addition, we conducted an in-depth ablation study to identify key factors contributing to the observed improvements. Our findings indicate that, beyond the contribution of the reweighted data ratios, the sequence of data ordering might also play a significant role in the effectiveness of Velocitune. The results show that incorporating the predicted target loss is critical for the effectiveness of the learning velocity. The contributions of this work can be summarised as follows.

- Introducing Velocitune, a novel framework for dynamically adjusting data ratios in continual pretraining. Velocitune estimates learning velocity to more precisely assess learning progress across domains and leverages scaling laws to optimise data allocation while minimising costs.

- Demonstrating through extensive experiments that Velocitune enhances downstream performance in two continual pre-training settings.

- Providing a detailed analysis revealing that reweighted data ratios, predicted target loss, and data ordering contribute to the effectiveness of Velocitune.

2 Domain-adaptive continual pre-training with Velocitune

In this section, we present a detailed explanation of Velocitune which focusses on adjusting domain weights based on learning velocity. To quantify learning velocity, it is essential to determine the model's existing expertise in each domain, which can be represented by its initial evaluation losses. Additionally, we define a learning target as the expected loss that the model should achieve given a certain amount of training data. This target provides a reference for measuring progress and adjusting domain weights accordingly. During training, we periodically assess learning velocity, increasing the sampling weights of slower domains while reducing the weights of faster ones. This adjustment ensures balanced learning progress across domains. The methodology is detailed in §2.1.

2.1 The Velocitune algorithm

Setup Consider training a language model on a dataset S consisting of n distinct domains, denoted $D_1, D_2, \ldots, D_n$. $S_1, S_2, \ldots, S_n$ represent the subsets of data corresponding to each domain, where each S_i is divided into a train and evaluation set, denoted as $S_{\text{train},i}$ and $S_{\text{eval},i}$, respectively. The weight of the domain $w \in \Delta^n$ represents the sampling weight assigned to the domains.

Target Estimation We first evaluate the LM on each evaluation set $S_{\text{eval},,i}$ before training to obtain the initial loss ℓ_{init}, providing an accurate measure of the model's learnt expertise in each domain. The target loss ℓ_{target} for each domain is derived using the Chinchilla scaling law (Hoffmann et al., 2022). Specifically, we apply the scaling law by first defining the total dataset size and then training the model on subsets of training data using the default domain weights, which correspond to the ratio of tokens in each domain. Throughout the training process, multiple checkpoints are saved and the evaluation losses from these checkpoints are used to fit the scaling law parameters. Finally, we take the fitted function to predict the loss that the model could reach by using the entire dataset. Details of the implementation and analysis of the prediction errors are provided in Appendix B.

Velocity-Guided Training After obtaining the initial loss and target loss in each domain, Velocitune iteratively updates domain weights, denoted by w_t, to minimise the re-weighted training loss effectively. The optimisation problem is formulated as follows, in two sequential steps:
(1) Minimise the weighted training loss:

$$\min_{\theta} \sum_{t=1}^{T} \sum_{i=1}^{n} w_t[i] \cdot \ell_t^i(\theta) \tag{1}$$

where T is the total training steps, $\ell_t^i(\theta)$ denotes the training loss at step t for domain D_i with model parameters θ. The $w_t[i]$ refers to the weight assigned to domain D_i at step t.
(2) Minimise the weighted sum of learning velocities:

$$\min_{w \in \Delta^n} \sum_{i=1}^{n} w_t[i] \cdot V_t[i] \tag{2}$$

$V_t[i]$ is the learning velocity of domain D_i at step t. The first step is to minimise the language modelling training loss given the domain weights. The second step focusses on dynamic adjustment so that the weight in the slowest learning velocity domain is maximised. Unlike Doremi (Xie et al., 2024), which updates domain weights based on the loss difference of each data point between a proxy model and a trained reference model, our method does not rely on a fully trained reference model. Consequently, the loss of each data point in the trained model is unobtainable. Instead, after a set number of training steps, we update the domain weights by first calculating the learning velocity, which reflects how quickly the model is learning in a given domain. The learning velocity is defined as:

$$V_t[i] = \frac{\ell_t^i(\theta, S'_{\text{eval},i}) - \ell^i_{\text{target}}}{\ell^i_{\text{init}} - \ell^i_{\text{target}}} \tag{3}$$

Here S'_{eval} is a subset of S_{eval} to speed up velocity estimation during training and $\ell_t^i(\theta, S'_{\text{eval},i})$ is the evaluation loss on S'_{eval} at step t in domain D_i. Then the domain weights are updated every m steps exponentially following Group DRO (Sagawa et al., 2019b):

$$w_t \leftarrow \frac{w_{t-m}[i] \cdot \exp(V_t[i])}{\sum_{i=1}^{n} w_{t-m}[i] \cdot \exp(V_t[i])} \tag{4}$$

Equation 4 adjusts the weights based on the learning velocity in each domain, ensuring an equitable comparison of the learning efficiency between domains with varying initial and target losses. By doing so, Velocitune prioritises learning in domains where progress towards the target loss is most promising, thereby enhancing overall model

Algorithm 1: Velocitune

Require: Data of n domains $S_1, S_2, \cdots, S_n$, initial data loading weights $w_0 \in \mathbb{R}^n$ initialise as uniform distribution, initial loss $\ell_{\text{init}} \in \mathbb{R}^n$, target loss $\ell_{\text{target}} \in \mathbb{R}^n$, $\ell_t^i(\theta, S'_{\text{eval},i})$ evaluation loss of the model at time t , evaluation interval m, model parameters θ.

for $t = 1, \cdots, T$ **do**
  if $t \mod m = 0$ **then**
    $\Delta_t[i] \leftarrow$
    $\texttt{Clamp}\left\{0, \frac{(\ell_t^i(\theta, S'_{\text{eval},i}) - \ell_{\text{target}}[i])}{(\ell_{\text{init}}[i] - \ell_{\text{target}}[i])}, 1\right\}$
    ▷ Measure learning velocity
    $w_t \leftarrow \texttt{UpdateWeight}(w_{t-m}, \Delta_t)$ ▷ Update data loading proportion
  end
  Sample a batch of data $\mathcal{B}$ from $S_{\text{train},1}, S_{\text{train},2}, \cdots, S_{\text{train},n}$ with proportion w_t;
  Update θ with $\mathcal{L}(\theta, \mathcal{B})$
end
Function $\texttt{UpdateWeight}(w, \Delta)$
  $\alpha \leftarrow w \cdot \exp(\Delta)$ ▷ Update unnormalised weights
  $w \leftarrow \frac{\alpha}{\sum_i^n \alpha[i]}$ **return** w ▷ Normalise domain weight

performance. In summary, this approach dynamically rebalances the learning process across multiple domains, adjusting weights in response to the observed rate of learning progress, and steering them towards achieving optimal and uniform performance improvements. The complete Velocitune algorithm is summarised in Algorithm 1.

3 Experiment

In this section, we apply Velocitune in continual pre-training CodeLlama-7B on the reasoning dataset and Llama3-8B as well as Mistral-7B on the system knowledge dataset.

3.1 Experimental setup

Training Corpus We compile the Reasoning dataset based on Proof-Pile-2 (Azerbayev et al., 2023) which consists of math reasoning text in natural language, format language, and code. The dataset includes three domains: Arxiv, AlgebraicStack, and OpenWebMath (Paster et al., 2023). Following the common practice of adding replay data to prevent catastrophic forgetting, we add two more domains: general code and general language which are composed of the Github subset from SlimPajama (Soboleva et al., 2023) and a blend of Slimpajama except Github and Arxiv. This results in a training set spanning five domains, with 76% of the data in natural language and 24% in code. For system knowledge, we use a dataset **SystemStack** which is collected from three domains Arxiv, Blogs, and Stackoverflow, concentrating on computer system-related knowledge. The statistics of the two datasets are shown in Table 1.

Table 1: Statistics for Reasoning corpus and SystemStack. Token counts were determined using the CodeLlama and Llama3 tokenizers, respectively.

Reasoning	
Domain	**# Tokens** (Billion)
Arxiv	28.70
AlgebraicStack	10.47
OpenWebMath	14.02
GenCode	3.97
GenLangauge	2.92
SystemStack	
Domain	**# Tokens** (Billion)
Arxiv	5.37
Blogs	3.21
StackOverflow	7.64

Training Setup We trained the models using the Negative Log-Likelihood (NLL) loss. Three settings were compared, Velocitune, Dynamic Batch Loading (DBL) (Xia et al., 2023) which updates the weight using the distance between the evaluation loss and the target loss, and a baseline where sampling weights are determined by proportional token counts across domains during the continual pre-training. For DBL, we also apply the predicted target loss. The total number of tokens processed during training was equivalent for the three methods to completing one full epoch using the training dataset. Detailed hyperparameters for the training process are summarised in Appendix C Table 8.

Evaluation To evaluate the math reasoning ability of models, we use math-lm-eval[2] from ToRA (Gou et al., 2023) to evaluate the accuracy on GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022),MMLU-STEM (Hendrycks et al., 2020), ASDiv (Miao et al., 2020), SVAMP (Patel et al., 2021),

[2]https://github.com/ZubinGou/math-evaluation-harness

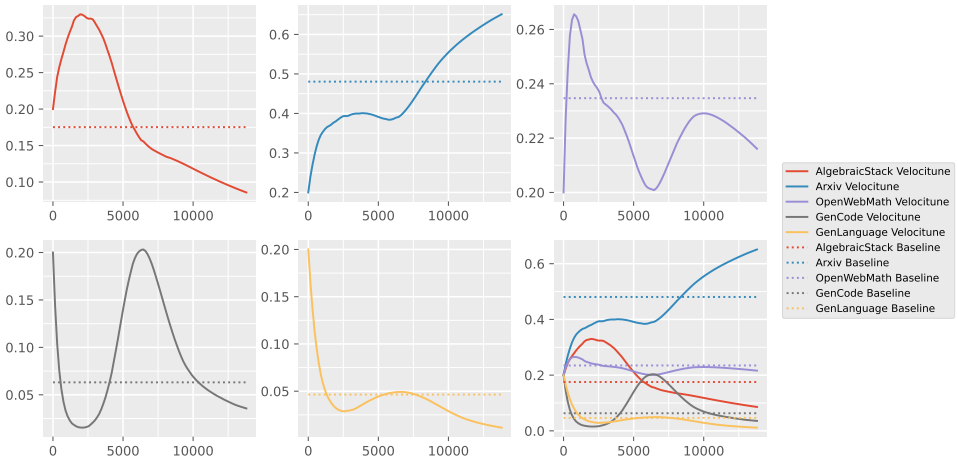

Figure 2: Domain weights dynamic of Velocitune in training CodeLlama 7B on Reasoning. The vertical axis represents domain weights, while the horizontal axis denotes training steps.

Table 2: Performance of CodeLlama-7B, the baseline, and Velocitune on multiple math and code benchmarks. The highest average accuracy for math and code benchmarks is highlighted in bold.

Model	**Math**					
	GSM8K	**MATH**	**Minerva**	**MATHQA**	**ASDiv**	**SVAMP**
CodeLlama	12.4	6.0	5.20	14.1	50.5	44.5
CodeLlama-Baseline	28.9	11.1	9.80	24.0	61.1	56.4
CodeLlama-Velocitune	28.4	11.7	11.4	25.1	60.9	56.1

Model	**Math**			**Code**		
	MMLU-STEM	**SAT**	**Math Avg.**	**HumanEval**	**MBPP**	**Code Avg.**
CodeLlama	20.9	18.8	21.6	30.5	43.2	36.8
CodeLlama-Baseline	36.0	46.9	34.3	26.2	44.8	35.5
CodeLlama-Velocitune	37.3	56.2	**35.9** (+1.6%)	34.1	44.4	**39.3** (+3.8%)

and MathQA (Amini et al., 2019). For coding ability, we use the evaluation kit from DeepSeek-Coder (Guo et al., 2024) to assess the pass@1 accuracy on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021).

For evaluating command generation ability, we use two benchmarks CMDGen-NVIDIA and CMDGen-AMD from the CMDGen series (Lin et al., 2025), which are built to assess the ability of models to provide proper system command when asked a related question. The two benchmarks, which provided a combination of 1.5K instruction-tuning data and 205 and 192 test questions respectively, evaluate model output from six metrics: Similarity of Command(**CMD Sim**): Cosine similarity of embeddings of generated and target commands. Similarity of Execution Output(**Output Sim**): Cosine similarity of embeddings of system outputs. Approximate Accuracy(**Approx Acc**): A value of 1 is assigned if either CMD Sim or Output Sim exceeds 0.9; otherwise 0, the overall Approx Acc is aggregated over test cases. The embeddings are generated by *all-MiniLM-L6-v2*[3]. Exact Match (**EM**): The percentage of generated commands that are identical to the target commands. Success Ratio (**SR**): The percentage of generated commands that produce the same system output as the target command. Accuracy (**Acc**): The union of EM and SR, serving as the primary metric for the CMDGen task. Examples of CMDGen question and answer pair are shown in Appendix A.

3.2 Reasoning training results

Velocitune learns math better and maintains coding ability. In Table 2, we list the benchmarks tested to compare the methods in CodeLlama-7B with Reasoning. Velocitune leads the baseline by 1.6% on average across eight math benchmarks and 3.8% on code average while the baseline's coding performance dropped by 1.3% from before the continued pre-training. It underscores the reason behind Velocitune's effectiveness, which is balancing the learning velocity in each domain, while

[3]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

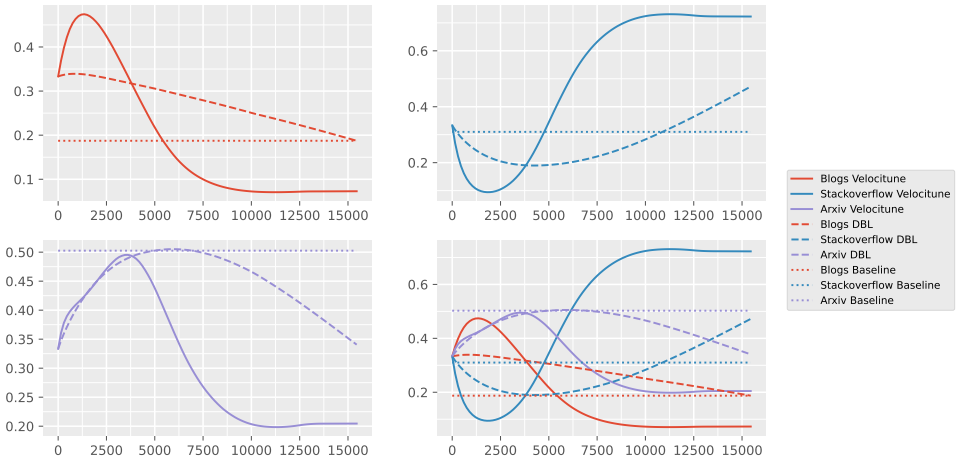

Figure 3: Domain weights dynamic of Velocitune and DBL in training Llama-3 8B on SystemStack. The vertical axis represents domain weights, while the horizontal axis denotes training steps.

Table 3: Results of Velocitune, DBL, and Basline on Llama3 and Mistral on CMDGen-NVIDIA and CMDGen-AMD benchmarks. Best results for each metric are highlighted in bold.

Model	CMDGen-NVIDIA					
	Cmd Sim	Output Sim	Approx Acc	EM	SR	Acc
Mistral	80.57	58.67	61.95	24.88	19.02	30.73
Mistral-Baseline	83.36	65.26	66.34	23.90	21.46	32.20
Mistral-DBL	79.83	65.97	59.02	24.39	19.02	32.20
Mistral-Velocitune	82.85	64.30	68.29	27.32	21.95	36.59 (+4.4%)
Llama-3	86.41	69.09	64.39	41.95	32.68	50.73
Llama-3-Baseline	87.53	72.15	69.27	46.34	37.07	57.07
Llama-3-DBL	83.83	64.87	63.90	38.54	27.80	45.85
Llama-3-Velocitune	**89.37**	**75.59**	**76.59**	**51.21**	**39.02**	**61.95** (+4.9%)
Model	**CMDGen-AMD**					
	Cmd Sim	**Output Sim**	**Approx Acc**	**EM**	**SR**	**Acc**
Mistral	84.25	59.49	61.54	25.64	15.90	29.23
Mistral-Baseline	87.58	65.72	70.77	25.13	18.46	30.77
Mistral-DBL	84.61	67.37	65.13	26.15	17.95	32.82
Mistral-Velocitune	86.84	62.99	69.23	27.18	18.46	33.33 (+2.6%)
Llama-3	88.08	71.22	67.18	41.54	27.69	47.18
Llama-3-Baseline	88.69	69.60	70.26	46.15	29.23	51.79
Llama-3-DBL	88.80	68.00	69.23	39.49	27.69	45.64
Llama-3-Velocitune	**89.23**	**72.97**	**74.36**	**49.74**	**31.28**	**54.87** (+3.1%)

in the baseline due to the static domain mixture, the model might learn some domains well while not saturating for other domains. Velocitune aligns the learning velocity across domains, resulting in more balanced learning progress, thus developing models of better downstream performance.

Velocitune dynamically regularises the learning velocity. As shown in Figure 2, the overall weight adjustments can be divided into three distinct stages. During the initial phase of training, the weights for newly introduced domains—OpenWebMath, Arxiv, and AlgebraicStack—rise, while those for the replay domains, GenCode and GenLanguage, decline sharply. This occurs because LMs are typically less saturated in the new domains, leading to slower learning compared to replay domains. As a result, Velocitune increases the weights of the underperforming domains to balance their learning. In the second stage, the weights for the replay domains begin to increase, while those of the new domains decrease. This shift occurs after the models have made substantial progress in the new domains, prompting Velocitune to reallocate the focus to the replay domains. In the final stage, the weights of the replay domains decrease again, while the downslopes for AlgebraicStack and OpenWebMath become more

Table 4: Performance comparison of CodeLlama-7B trained using Reasoning data with default domain ratios, reweighted domain ratios, and Velocitune, evaluated on math and code reasoning benchmarks.

Model	Math					
	GSM8K	MATH	Minerva	MATHQA	ASDiv	SVAMP
CodeLlama-Baseline	28.9	11.1	9.8	24.0	61.1	56.4
CodeLlama-Reweighted	28.7	11.0	12.4	23.6	60.9	56.3
CodeLlama-Velocitune	28.4	11.7	11.4	25.1	60.9	56.1
Model	**Math**			**Code**		
	MMLU-STEM	**SAT**	**Math Avg.**	**HumanEval**	**MBPP**	**Code Avg.**
CodeLlama-Baseline	36.0	46.9	34.3	26.2	44.8	35.5
CodeLlama-Reweighted	37.7	62.5	**36.6**	30.5	42.8	36.6
CodeLlama-Velocitune	37.3	56.2	35.9	34.1	44.4	**39.3**

Table 5: Performance comparison of Llama3 models trained using SystemStack with original domain ratios, reweighted domain ratios, and Velocitune, evaluated on the CmdGen benchmarks

Model	CmdGen-NVIDIA					
	Cmd Sim	Output Sim	Approx Acc	EM	SR	Acc
Llama-3-Baseline	88.69	69.60	70.26	46.15	29.23	51.79
Llama-3-Reweighted	87.46	72.09	67.32	46.83	35.61	56.10
Llama-3-Velocitune	**89.37**	**75.59**	**76.59**	**51.21**	**39.02**	**61.95**
Model	**CmdGen-AMD**					
	Cmd Sim	**Output Sim**	**Approx Acc**	**EM**	**SR**	**Acc**
Llama-3-Baseline	88.69	69.60	70.26	46.15	29.23	51.79
Llama-3-Reweighted	88.92	**73.16**	69.74	47.18	**32.31**	**54.87**
Llama-3-Velocitune	**89.23**	72.97	**74.36**	**49.74**	31.28	**54.87**

gradual, even showing a slight peak before continuing to decline. This suggests that the model has learnt these domains well. Meanwhile, Velocitune increases the weight of Arxiv. Adjusting weights dynamically, Velocitune aligns the learning velocity across domains, ensuring a balanced learning outcome.

3.3 SystemStack training results

Velocitune brings improvements across benchmarks. On SystemStack, Velocitune improves downstream task performance over DBL and Baseline on Llama3-8B and Mistral-7B. Table 3 shows that Velocitune improves the Accuracy in CMD-NVIDIA by 4.9% and 4.4% compared to baseline on Llama3-8B and Mistral-7B, and 3.1% and 2.6% and in CMDGen-AMD. We also found that Llama3-DBL underperforms the baseline on both benchmarks, which could be attributed to an imbalanced learning brought by updating domain weights based on the distances to the target loss.

Velocitune accelerates weight stabilisation We plot the movement trajectory of the weights in SystemStack comparing Velocitune, DBL, and the baseline in Figure 3. In parallel with Xia et al. (2023), we also observe that the weights stabilise after a few thousand steps. As rescaling accelerates convergence in machine learning (Juszczak et al., 2002), we also observe that Velocitune on SystemStack reaches stabilisation at least 1.5x faster than DBL as DBL does not stabilise at the end of training. For training on Reasoning which is shown in Figure 2, we do not observe a complete stabilisation, but only the curve becoming flat at the end of training.

3.4 Data ordering contributes along with reweighted domain weights

To isolate the key contributors behind the effectiveness of Velocitune, we performed an ablation study to examine the effect of the reweighted data proportion. Specifically, we collect the weights during Velocitune for each evaluation interval and then average the weights to get the overall sampling ratio of each domain during training. A comparison of the original ratio and the reweighted ratio is shown in Tables 9 and 10. Using the reweighted data ratio, we train the model on the dataset using

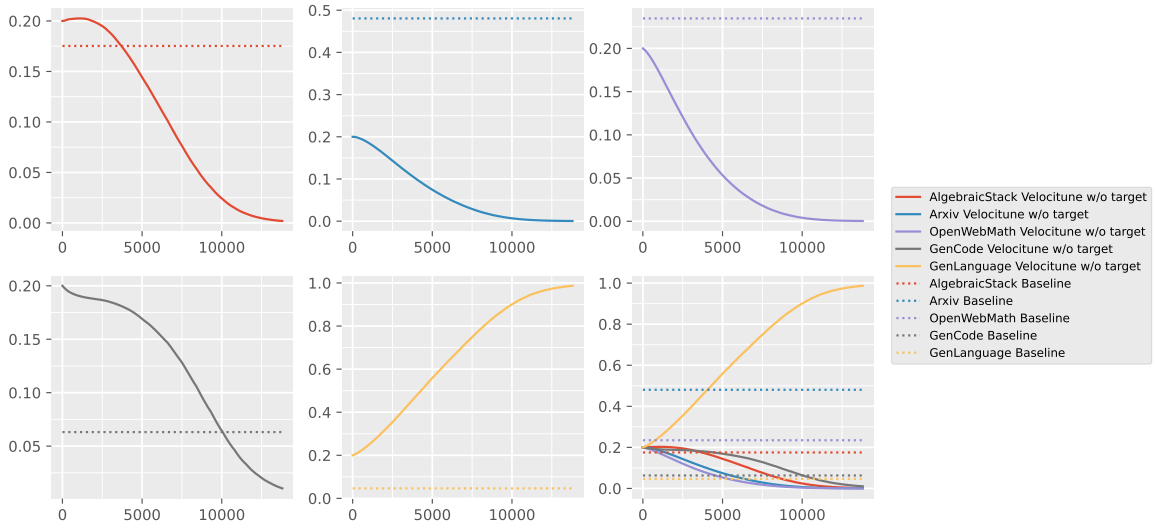

Figure 4: Domain weights dynamics of Velocitune without target loss in training CodeLlama 7B on Math&Code.

Table 6: Performance comparison of CodeLlama-7B trained using Reasoning data with or without target loss and Baseline, evaluated on math and code benchmarks.

Model	**Math**					
	GSM8K	**MATH**	**Minerva**	**MATHQA**	**ASDiv**	**SVAMP**
Velocitune w/o target loss	17.5	8.2	7.4	31.5	43.8	48.8
Baseline	28.9	11.1	9.80	24.0	61.1	56.4
Velocitune	28.4	11.7	11.4	25.1	60.9	56.1
Model	**Math**			**Code**		
	MMLU-STEM	**SAT**	**Math Avg.**	**HumanEval**	**MBPP**	**Code Avg.**
Velocitune w/o target loss	54.2	30.2	30.2	28.0	42.0	35.0
Baseline	36.0	46.9	34.3	26.2	44.8	35.5
Velocitune	37.3	56.2	**35.9**	34.1	44.4	**39.3**

the static mixture for the same training steps. We conducted the experiment for CodeLlama-7B on the Reasoning dataset and Llama3 on SystemStack. The results are shown in Table 4 and Table 5.

On CMDGen benchmarks, Llama-3 trained with reweighted ratio data(denoted as Llama-3-Reweighted) outperforms Llama3-Baseline on both benchmarks, but is inferior to Velocitune on NVIDIA and on par with Velocitune on AMD. In the evaluation of math and code reasoning, CodeLlama trained with the reweighted ratio(denoted as CodeLlama-Reweighted) achieves a slightly higher accuracy than Velocitune on the math benchmarks, but still falls behind Velocitune by 2.7% for the coding benchmark. In general, reweighted data ratios improve downstream task performance compared to the original mixture. However, the models still underperform relative to Velocitune, with the sole exception being the Math average, where the performance is 0.7% higher than Velocitune. This finding sheds new light on the common assumption that data mixture is the primary factor driving downstream performance. Previous studies (Hu et al., 2024; AI@Meta, 2024) have highlighted the effects of presenting different data at various stages of model training. Our comparison further supports the potential of dynamic data weighting during pre-training. We encourage further research into the impact of data ordering and its role in optimising model performance.

3.5 Target loss is critical to Velocitune

To highlight the significance of the target loss estimate in learning velocity, we also explore an alternative method that excludes the target loss in learning velocity when updating the weights. In this setting, Equation 3 is reduced to dividing the evaluation loss by the initial loss for each domain, as given by: $\delta^i = \frac{\ell_t^i(\theta, S'_{\text{eval},i})}{\ell_{\text{init}}^i}$. Using this formulation, we train CodeLlama on the Reasoning dataset. The evaluation results are presented in Table 6 and the dynamic of the domain weight is shown in Figure 4. We see that without the constraints of tar-

get loss, the weight of GenLangauge is driven to nearly one, and others are suppressed to nearly zero. Consequently, the model exhibits a performance decrease of 4.1% in the math benchmarks and 0.5% in the code benchmarks compared to the baseline. We hypothesise that this update function implicitly assumes a target loss of zero for each domain, leading to unrealistic expectations for the model to minimise the loss to zero across all domains. Such impractical targets disrupt the training process, ultimately degrading overall performance.

4 Related Work

GLaM (Du et al., 2022) manually assigns weights to each domain based on the amount of data and the downstream performance of a small model trained on the domain data. Lemma (Azerbayev et al., 2023), an effort to continue pre-training LMs with math-related data from three domains, determines domain weights by iterating through different combinations and selecting those with the lowest perplexity on an evaluation dataset. Doremi (Xie et al., 2024) employs a three-stage process to dynamically adjust domain weights: (1) Train a reference model, (2) Train a proxy model while adjusting the proportion of data based on the proxy model's loss, and (3) Train the final model using the aggregated data weights, demonstrating superior performance compared to models trained with the original weights. Xia et al. (2023) shows that instead of training a reference model, the target loss value can be estimated using scaling laws (Hoffmann et al., 2022), achieving better performance on smaller LMs with the predicted losses.

5 Conclusion

In this work, we introduced Velocitune, a novel approach to dynamically adjust domain weights during the training of large language models. Our method addresses the gap of existing works in domain-adapative continual pre-training. By aligning learning velocity across domains, Velocitune ensures a more balanced learning process, thereby enhancing the model's ability to generalise across diverse downstream tasks. Our experiments on CodeLlama-7B, Llama3-8B, and Mistral-7B demonstrate the effectiveness of Velocitune in improving performance in a variety of tasks, including math reasoning, coding, and system command generation. Furthermore, our ablation study reveals that the benefits of Velocitune are not solely due to the adjustment of the domain weights, but also stem from the synergistic effect of the training dynamics, including the order in which data from different domains are processed. This finding underscores the importance of considering both the weights and the data order of in continual pre-training.

6 Limitations

The limitations of our study are threefold. First, Velocitune is currently designed for continual pre-training and has not yet been evaluated in pre-training from scratch. Second, while supervised fine-tuning (SFT) datasets often span multiple domains, the applicability of Velocitune in the SFT stage remains unexplored. Third, Velocitune incurs higher costs than the baseline due to its evaluation process during training. However, since pre-training primarily prioritises downstream performance, we defer a detailed analysis of computational efficiency and potential optimisations of Velocitune to future work.

7 Ethics Statement

This research is based on open-source models and is intended solely for research purposes. We acknowledge potential risks. While Velocitune improves model efficiency and performance, improved reasoning and code generation capabilities could be misused in misinformation generation, automated decision making, or bias reinforcement. Our work focusses on specific domains (maths, code, and system commands), which may not generalise well to under-represented languages or topics. Our research does not involve sensitive or private data.

References

AI@Meta. 2024. Llama 3 model card.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *North American Chapter of the Association for Computational Linguistics*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *ArXiv*, abs/2108.07732.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang,

Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

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

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR.

Simin Fan, Matteo Pagliardini, and Martin Jaggi. 2023. Doge: Domain reweighting with generalization estimation. *arXiv preprint arXiv:2310.15393*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *ArXiv*, abs/2309.17452.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *ArXiv*, abs/2401.14196.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Xiaodong Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *ArXiv*, abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Xiaodong Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *ArXiv*, abs/2103.03874.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zhen Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chaochao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *ArXiv*, abs/2404.06395.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Piotr Juszczak, D Tax, and Robert PW Duin. 2002. Feature scaling in support vector data description. In *Proc. asci*, pages 95–102. Citeseer.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *ArXiv*, abs/2206.14858.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. 2024. Datacomp-lm: In search of the next generation of training sets for language models. *Preprint*, arXiv:2406.11794.

Zhenghao Lin, Zihao Tang, Xiao Liu, Yeyun Gong, Yi Cheng, Qi Chen, Hang Li, Ying Xin, Ziyue Yang, Kailai Yang, Yu Yan, Xiao Liang, Shuai Lu, Yiming Huang, Zheheng Luo, Lei Qu, Xuan Feng, Yaoxiang Wang, Yuqing Xia, Feiyang Chen, Yuting Jiang,

Yasen Hu, Hao Ni, Binyang Li, Guoshuai Zhao, Jui-Hao Chiang, Zhongxin Guo, Chen Lin, Kun Kuang, Wenjie Li, Yelong Shen, Jian Jiao, Peng Cheng, and Mao Yang. 2025. Sigma: Differential rescaling of query, key and value for efficient language models. *Preprint*, arXiv:2501.13629.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. Openwebmath: An open dataset of high-quality mathematical web text. *ArXiv*, abs/2310.06786.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. 2019a. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *ArXiv*, abs/1911.08731.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. 2019b. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*.

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

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. 2024. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36.

A Examples of CmdGen

We show in the following a question and answer pair from the CMDGen benchmarks.

CMDGen Example 1: Generate a Command to Use "nvidia-smi pmon".

Question.
How to use 'nvidia-smi pmon' to monitor GPU with ID 0, updating every 4 seconds for a total of 10 times, and logging the output to '\/tmp\/nvidia-log.txt'?

Answer.
```
nvidia-smi pmon -i 0 -d 4 -c 10 --filename \tmp\nvidia-log.txt
```

Figure 5: One question and answer pair from CMDGen-Nvidia which prompts the model to generate a command using specific input and output parameters

B Target Loss Prediction Details

To predict the target losses, we continue training the models on the training split, saving a checkpoint at regular intervals. After each new checkpoint is obtained, we update the scaling law by fitting it with the new parameters. This process is repeated until the difference between the predicted target losses at consecutive time steps falls below a predefined threshold, denoted by σ. This stopping criterion ensures the selection of an appropriate value for k during proxy model training. Detailed scaling law prediction errors are given in Table 7.

Table 7: The average error over domain for predicting evaluation loss of models using full training data. We used evaluation loss from checkpoints saved till using half the training data to fit the laws.

	Reasoning	**SystemStack**
Error	2.4e-3	1.84e-3

C Training Details

We run the experiments on 64 Nvidia H100 GPUs. Our distributed training is based on *accelerate*[4] and *FDSP*[5]. Hyperparameters are shown in Table 8.

Table 8: Training hyper-parameters and throughput.

	SystemStack	**Reasoning**
Training tokens	16B	63B
Learning rate	$1e-5$	$5e-5$
LR warmup ratio	0.005	0.005
Batch size (tokens)	1M	4M
Evaluation interval m (steps)	150	150
Steps	$15,482$	$13,807$
# GPUs	64	64
Sample ratio for proxy model	58%	51%
Adam β_2	0.95	0.99

[4] https://huggingface.co/docs/accelerate/en/index
[5] https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/

D Reweighted Data Ratio after Velocitune

The reweighted data ratios for the two training settings are presented in Table 9 and Table 10. In the case of SystemStack training on Llama3, after reweighting with Velocitune, we observe an increase in the weights assigned to Blogs and ArXiv, at the expense of StackOverflow. Similarly, during the training of CodeLlama on Reasoning, the model places less emphasis on ArXiv and OpenWebMath, while allocating more weight to GenCode and GenLanguage.

Table 9: The domain weights of Baseline and Velocitune's reweighted domain weights on SystemStack for Llama3

	Blogs	**Stackoverflow**	**Arxiv**
Baseline	0.187	0.503	0.310
Velocitune-Reweighted	0.197	0.470	0.333

Table 10: The domain weights of Baseline and Velocitune's reweighted domain weights on Reasoning for CodeLlama

	AlgebraicStack	**Arxiv**	**OpenWebMath**	**GenCode**	**GenLanguage**
Baseline	0.189	0.500	0.259	0.029	0.020
Velocitune-Reweighted	0.185	0.463	0.225	0.086	0.040