$\mathcal{X}$FT: Unlocking the Power of Code Instruction Tuning by Simply Merging Upcycled Mixture-of-Experts

Anonymous ACL submission

Abstract

We introduce $\mathcal{X}$**FT**, a simple yet powerful training scheme, by simply merging upcycled Mixture-of-Experts (MoE) to unleash the performance limit of instruction-tuned code Large Language Models (LLMs). While vanilla sparse upcycling fails to improve instruction tuning, $\mathcal{X}$FT introduces a shared expert mechanism with a novel routing weight normalization strategy into sparse upcycling, which significantly boosts instruction tuning. After fine-tuning the upcycled MoE model, $\mathcal{X}$FT introduces a learnable model merging mechanism to compile the upcycled MoE back to a dense model, achieving upcycled MoE-level performance with only dense-model compute. By applying $\mathcal{X}$FT to a 1.3B model, we create a new state-of-the-art small LLM (<3B) for code with 67.1 and 64.6 pass@1 on HumanEval and HumanEval+ respectively. Furthermore, with the same data and model architecture, $\mathcal{X}$FT improves supervised fine-tuning (SFT) by 13% on HumanEval+, along with consistent improvements from 2% to 13% on MBPP+, MultiPL-E, and DS-1000, demonstrating its generalizable effectiveness. $\mathcal{X}$FT is fully orthogonal to existing techniques such as Evol-Instruct and OSS-INSTRUCT, opening a new dimension for improving code instruction tuning.

1 Introduction

Program synthesis (or code generation) is a long-standing problem explored since the early days of computer science (Manna and Waldinger, 1971). Recently, instruction tuning of code Large Language Models (LLMs) has been used to improve many coding tasks (Chaudhary, 2023; Luo et al., 2023; Wei et al., 2023), such as text-to-code generation (Chen et al., 2021; Austin et al., 2021), code completion (Cassano et al., 2022), and data science engineering (Lai et al., 2022).

A typical instruction tuning flow involves two steps (Zhang et al., 2023): (i) curating an instruction dataset of instruction-output pairs, where the instruction reflects human intents in natural language and the output includes explained code snippets that correspond to the intent; and (ii) supervised fine-tuning of pre-trained LLM on the instruction dataset. In the realm of code, multiple instruction-tuning methods have been proposed to curate high-quality instruction datasets. For example, *Code Evol-Instruct* (Luo et al., 2023) uses ChatGPT to obtain complex synthetic code instructions with heuristic prompts, while OSS-INSTRUCT (Wei et al., 2023) prompts ChatGPT to generate new coding problems by drawing inspiration from open source code snippets. While existing work focuses on the data perspectives of instruction tuning, they all follow the standard SFT, leaving room for exploring advanced training schemes.

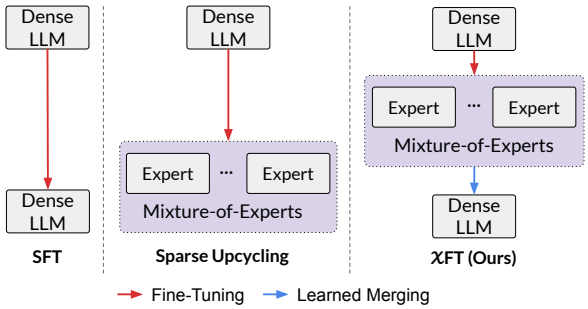

Figure 1: Overview of SFT, sparse upcycling, and $\mathcal{X}$FT.

We argue that prior works largely overlook the possibility of improving the code instruction tuning by advancing the training schemes. Figure 1 depicts the supervised fine-tuning (SFT), which directly starts with the pre-trained weights and architecture for fine-tuning. The model is *dense* here because all parameters are activated to compute the next token (assuming it is a decoder-only LLM). In contrast to fine-tuning a *dense* model, following the scaling laws (Kaplan et al., 2020) (*i.e.,* more parameters, better performance), sparse upcycling (Komatsuzaki et al., 2023) is proposed to efficiently upgrade the model sizes by upcycling a dense LLM to a sparsely activated Mixture-of-Experts (MoE) model. An MoE model is efficient because the

generation of the next token only involves a subset of parameters (*i.e.,* experts) and thus is *sparsely activated*. For example, Mixtral-8x7B (Jiang et al., 2024), compared to a dense 7B model, uses approximately $8\times$ parameters and $2\times$ computation, *i.e.,* only 2 out of 8 experts are dynamically selected to compute the next token. However, there are two key limitations when using sparse upcycling in instruction tuning: (i) *Slow scaling:* Komatsuzaki et al. (2023) show that sparse upcycling improves the dense SFT marginally at the early phase, requiring orders of magnitude of extra compute to achieve decent improvement; and (ii) *Inference cost:* though MoE is more efficient than directly scaling the size of dense LLMs, MoE is still expensive, especially at inference, as it introduces significantly more parameters (*i.e.,* memory) and, more importantly, computes during inference, compared to its dense counterparts.

In this paper, we propose $\mathcal{X}$**FT**: by simply merging upcycled MoE models, we push the performance limit of instruction-tuned code LLMs. While vanilla sparse upcycling fails to improve instruction tuning efficiently (Komatsuzaki et al., 2023), $\mathcal{X}$FT addresses this challenge by isolating one expert as the shared expert among all the other experts in each MoE layer, inspired by DeepSeek-MoE (Dai et al., 2024) and MoCLE (Gou et al., 2024). $\mathcal{X}$FT also includes a novel routing weight normalization strategy to eliminate scale mismatch between the upcycled MoE layer with the shared expert and the original dense layer, which will otherwise lead to performance degradation (Wu et al., 2022). After the upcycled MoE model finishes the SFT phase, motivated by Model Soups (Wortsman et al., 2022), $\mathcal{X}$FT uses a learnable model merging mechanism to output a dense model by merging all the expert networks in the upcycled MoE, *i.e.,* the final dense model is of the same model structure and size as the original pre-trained model, achieving similar performance without paying extra inference cost as the sparse upcycling. With only 1.3B parameters, $\mathcal{X}$FT achieves 67.1 pass@1 on HumanEval and 64.6 pass@1 on HumanEval+, which is the new state-of-the-art for tiny code LLMs (<3B). Compared with SFT, $\mathcal{X}$FT achieves 13% improvement on HumanEval+. Surprisingly, our model merging mechanism can preserve or even further boost the general performance of the upcycled MoE with around $^1/_8\times$ parameters! We conclude our contribution as follows:

- **Dimension:** We open a new dimension of improving instruction tuning of code LLMs by advancing its training scheme, using enhanced sparse upcycling and learnable model merging mechanism, which neither changes the final model structure nor requires more training data.
- **Technique:** We present $\mathcal{X}$FT, a new training scheme for code instruction tuning. $\mathcal{X}$FT involves two steps: *upcycling* and *merging*. A pre-trained dense LLM is first upcycled into an MoE with the shared expert setting and then fine-tuned on the instruction dataset. To avoid the performance degradation caused by the scale mismatch issue, we propose a novel routing weight normalization strategy. In addition, we introduce the first learnable mechanism for merging the upcycled MoE into a dense model, eliminating additional inference overhead while preserving or even improving the MoE performance.
- **Results:** With only 1.3B parameters, $\mathcal{X}$FT achieves 67.1 pass@1 on HumanEval and 64.6 pass@1 on HumanEval+, which is the new state-of-the-art for tiny code LLMs (<3B). Compared with normal supervised fine-tuning (SFT), $\mathcal{X}$FT achieves 13% improvement on HumanEval+! Furthermore, $\mathcal{X}$FT achieves a consistent improvement from 2% to 13% on MBPP, MultiPL-E, and DS-1000 over normal SFT, demonstrating its generalization.

2 Related Work

2.1 Mixture-of-Experts

Mixture-of-Experts (MoE) can efficiently scale up model sizes with only sub-linear increases in computation (Shazeer et al., 2017). Compared with the standard Transformer, MoE replaces each Feed-Forward Network (FFN) layer with an MoE layer, which uses N (*i.e.,* multiple) expert networks that are structurally equivalent to the original FFN layer and uses a router that directs each input token to K out of N expert networks. Formally, for the l-th MoE layer, output hidden state $\mathbf{h}_t^l$ of the t-th input token is computed as follows (Dai et al., 2024):

$$\begin{aligned}
\mathbf{h}_t^l &= \sum_{i=1}^{N} (g_{i,t}\mathrm{FFN}_i(\mathbf{u}_t^l)) + \mathbf{u}_t^l \\
g_{i,t} &= \begin{cases} s_{i,t} & s_{i,t} \in \mathrm{Topk}(s_t, K) \\ 0 & \text{otherwise} \end{cases} \\
s_t &= \{s_{i,t} \mid 1 \le i \le N\} \\
s_{i,t} &= \mathrm{Softmax}_i(\mathbf{u}_t^{l\,T}\mathbf{e}_i^l)
\end{aligned} \tag{1}$$

where N refers to the total number of experts, $g_{i,t}$ refers to the gate value for the i-th expert, $\text{FFN}_i(\cdot)$ refers to the i-th expert, $\mathbf{u}_t^l$ refers to the hidden states of the t-th token which is the input of the l-th MoE layer, $s_{i,t}$ refers to the affinity score between the i-th expert and the t-th token, $\text{Topk}(S, K)$ refers to a function computing K largest scores over S, and $\mathbf{e}_i^l$ refers to the centroid of the i-th expert in the l-th MoE layer. By definition, each token will only be assigned to and computed in the top K experts among all the N experts.

Recently, many works have been proposed to scale model sizes with MoE architecture (Lepikhin et al., 2020; Du et al., 2022; Fedus et al., 2022; Jiang et al., 2024; Xue et al., 2024). While most MoE models are trained from scratch, sparse upcycling (Komatsuzaki et al., 2023) is proposed to initialize MoE models based on the pre-trained dense model, which can efficiently reduce the computational costs of training MoE models, compared with training MoE models from scratch. Specifically, sparse upcycling constructs a new MoE model by initializing each expert of each MoE layer as a copy of the original FFN layer in the dense model, while directly copying the remaining layers from the dense model to the new MoE model.

2.2 Instruction Tuning

Instruction tuning is designed to improve the instruction-following ability of LLMs by fine-tuning them on the instruction datasets in a supervised fashion (Wei et al., 2022). The quality of the instruction dataset is significant for the effectiveness of instruction tuning and researchers have proposed multiple methods to improve data quality. For example, SELF-INSTRUCT (Wang et al., 2023) synthesizes high-quality instruction data by prompting a foundation LLM with specially designed prompts. To improve SELF-INSTRUCT, Evol-Instruct (Xu et al., 2023) improves the complexity and diversity of the instruction dataset by prompting ChatGPT with heuristic prompts. OSS-INSTRUCT (Wei et al., 2023) queries ChatGPT to generate instruction-output pairs by getting inspiration from real-world code snippets.

Recently, some parameter-efficient fine-tuning techniques have been proposed to use MoE for better instruction tuning. For example, LoRAMoE (Dou et al., 2023) and MoCLE (Gou et al., 2024) propose MoE-like modules that are constructed with Low-Rank Adaptations (LoRA) to improve instruction tuning, while PESC (Wu et al., 2024) proposes to integrate adapters into MoE that are upcycled from dense models. Different from these works, $\mathcal{X}$FT focuses on full fine-tuning, which generally performs better than parameter-efficient fine-tuning (Chen et al., 2022).

2.3 Weight Averaging

Weight averaging is a commonly used technique to improve the performance of deep learning models. For example, Model Soups (Wortsman et al., 2022) averages the weights of multiple models that are initialized from the same pre-trained model but finetuned with different hyperparameter configurations to improve the accuracy and robustness of the model. However, only a few works have been proposed to merge expert networks of an MoE layer to a normal FFN layer using weight averaging. For example, OneS (Xue et al., 2022) proposes several simple weight averaging methods to merge expert networks of a BERT-based MoE model. Closely related to our work, Experts Weights Averaging (EWA) (Huang et al., 2023) proposes to convert an MoE model to a dense model with two steps: (i) During MoE training, EWA conducts weighted averaging of all the expert weights after each weight update of MoE, which is based on a manually-crafted hyperparameter β; (ii) After training, EWA converts each MoE layer into an FFN layer by uniformly averaging the experts.

Different from all the aforementioned existing works, $\mathcal{X}$FT is the first work proposing a **learnable** mechanism to merge expert networks in the upcycled MoE model. Furthermore, while the training scheme of EWA is deeply coupled to a specific MoE architecture, $\mathcal{X}$FT can be easily adapted to different MoE architectures by only adjusting the final merging process. In addition, unlike EWA, $\mathcal{X}$FT does not introduce any hyperparameters into the training of the large MoE models, significantly reducing the computational resources for hyperparameter searching. Our empirical results in Section 4 also showcase the clear advantage of $\mathcal{X}$FT.

3 $\mathcal{X}$FT

We describe the details of $\mathcal{X}$FT in this section. There are two steps in our framework: *upcycling* (Section 3.1) and *merging* (Section 3.2). During upcycling, we construct an Mixture-of-Experts (MoE) model from the pre-trained dense model, namely $\textbf{MoE}_{\textbf{DS}}$, which is then fine-tuned on coding instruction data. For merging, we propose a learnable

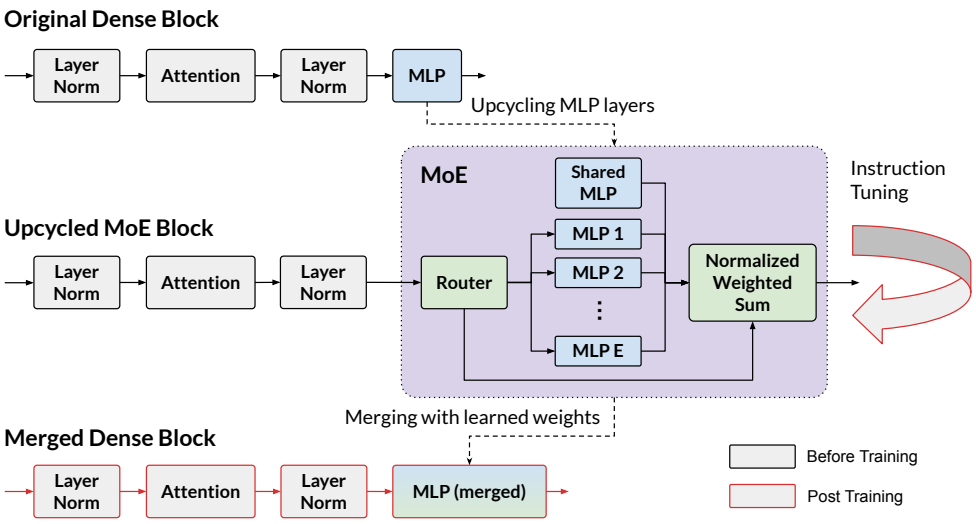

Figure 2: Overview of $\mathcal{X}$FT.

model merging method to convert the instruction-tuned MoE_{DS} back to a normal dense model by merging each MoE layer into an FFN layer through weight averaging while directly copying other remaining layers. Consequently, we can obtain $\mathcal{X}\textbf{FT}_{\textbf{DS}}$ that has the same model architecture and size as the original pre-trained dense model, which eliminates all the additional inference overhead brought by the original sparse upcycling, while preserving or even improving the performance of MoE_{DS}. Our framework is illustrated in Figure 2.

3.1 Upcycling

Inspired by sparse upcycling (Komatsuzaki et al., 2023), we convert the pre-trained dense LLM to a new MoE by initializing each expert of each MoE layer as a copy of the original FFN layer in the dense model, while directly copying the remaining layers from the dense model to the new MoE model. However, the performance gain brought by sparse upcycling is negligible with a very limited extra training budget (Komatsuzaki et al., 2023) – which is exactly the situation we are facing during instruction tuning. Intuitively, it is because each expert in the upcycled MoE model is trained on fewer instruction data than the original dense model does because traditional routers used in sparse upcycling will assign different tokens to different experts and thus reduce the amount of data each expert is trained on (Gou et al., 2024). Consequently, inspired by DeepSeekMoE (Dai et al., 2024) and MoCLE (Gou et al., 2024), $\mathcal{X}$FT introduces the shared expert setting into sparse upcycling to tackle this challenge. We further propose a novel routing weight normalization strategy for $\mathcal{X}$FT to avoid the potential performance degradation caused by the scale mismatch problem (Wu et al., 2022).

3.1.1 Shared Expert for Upcycling

During upcycling, we isolate one shared expert among all the other normal experts in each MoE layer, where the shared expert will be deterministically assigned to handle all the tokens while other normal experts are assigned by the router. By doing so, the upcycled MoE model can achieve a clear performance boost in instruction tuning, where the shared expert can learn general knowledge across the whole instruction dataset while other normal experts learn specific knowledge among different instructions assigned by the router. Formally speaking, following the definition in Eq. (1), the output hidden state $\mathbf{h}_t^l$ of the l-th MoE layer when processing the t-th token can be expressed as:

$$
\begin{aligned}
\mathbf{h}_t^l &= \sum_{i=1}^{N} (g_{i,t}\text{FFN}_i(\mathbf{u}_t^l)) + \mathbf{u}_t^l \\
g_{i,t} &= \begin{cases} 1 - s_{t\max} & i = 1 \\ \text{Softmax}_i(s_{i,t}) \cdot s_{t\max} & s_{i,t} \in S_{tK} \\ 0 & \text{otherwise} \end{cases} \\
S_{tK} &= \text{Topk}(\{s_{i,t} \mid 1 \le i \le N\}, K-1) \\
s_{t\max} &= \max(\{s_{i,t} \mid 1 \le i \le N\}) \\
s_{i,t} &= \begin{cases} -\infty & i = 1 \\ \text{Softmax}_i({\mathbf{u}_t^l}^T \mathbf{e}_i^l) & i \ge 2 \end{cases}
\end{aligned}
\tag{2}
$$

where FFN_1 is chosen as the shared expert in each MoE layer, and each token will be assigned to top K experts including one shared expert and $K-1$ other normal experts. Compared with the original sparse upcycling, there are two major differences:

- **Weighted Shared Expert**. Following MoCLE (Gou et al., 2024), with the token-to-expert affinity score $s_{i,t}$, we get the maximum affinity score $s_{t\max}$ and use its complement $1 - s_{t\max}$ as the routing weight of the shared expert.
- **Routing Weight Normalization**. Although the shared expert setting is also used in recent works (Dai et al., 2024; Gou et al., 2024), we cannot directly follow their routing strategy because they cannot handle a scale mismatch problem that is unique for sparse upcycling. The scale mismatch problem is that differences between the scale of the output of the upcycled MoE layer and the original FFN layer can cause performance degradation (Wu et al., 2022). To handle this problem, we need to make sure the sum of $g_{i,t}$ equals 1, so that the output of the MoE layer matches that of the FFN layer in scale. To do so, we normalize the affinity scores of top $K-1$ normal experts with Softmax and scale their sum to $s_{t\max}$ to make sure that the sum of the $g_{i,t}$ of top K experts, including one shared expert and $K-1$ normal experts, equals 1.

3.2 Merging

We propose a learnable model merging method to convert the large MoE model, namely MoE_{DS}, back to a dense model $\mathcal{X}\text{FT}_{\text{DS}}$. By doing so, we expect $\mathcal{X}\text{FT}_{\text{DS}}$ to keep the boosted performance gained during upcycling while keeping its model size the same as the original dense model to avoid any additional inference overhead. Inspired by Model Soups (Wortsman et al., 2022), we choose to merge MoE_{DS} by learning the mixing coefficients that can be used to average the parameters of all experts in each MoE layer to obtain a normal FFN layer, while directly copying other remaining layers.

Formally speaking, given the weights of N experts at the l-th layer $W_1^l, W_2^l, \cdots, W_N^l$, the process of merging each MoE layer to an FFN layer can be stated as below:

$$\overline{W^l} = \sum_{i=1}^{N} \alpha_i^l W_i^l \tag{3}$$

where $\overline{W^l}$ denotes the merged parameter of all N experts and α_i^l denotes the learnable mixing coefficient of expert W_i^l. We consider a neural network $f(x;\theta)$ with input x and parameters θ. For loss $\mathcal{L}$ and instruction dataset $\{(x_i, y_i)\}_{i=1}^m$, such mixing coefficients α of all the L layers can be learned via:

$$\arg\min_{\alpha} \sum_{j=1}^{m} \mathcal{L}(f(x_j;\theta_o, (\sum_{i=1}^{N} \alpha_i^l W_i^l)_{1:L}), y_i) \tag{4}$$

where θ_o refers to all the remaining layers of MoE_{DS} other than MoE layers and α is parameterized as the output of a softmax, so that each α_i^l is positive and $\sum_{i=1}^{N} \alpha_i^l = 1$.

While the learning process defined in Eq. (4) is the most intuitive way of learning α, our experiment in Section 5.2 shows that, due to the shared expert setting, it tends to simply increase the mixing coefficient of the shared expert at each layer as much as possible to decrease the loss. It is not helpful because, although the shared expert has learned general knowledge across the whole instruction dataset and needs a relatively large mixing coefficient, we still need to keep the scale of the mixing coefficient of other normal experts at a certain level also to keep some specific knowledge learned by other normal experts in the merged parameter $\overline{W^l}$.

To solve this issue, we introduce a *shared expert rate* λ to fix the mixing coefficient of the shared expert and learn the mixing coefficients of the remaining normal experts which sums to $1-\lambda$ in each layer. By doing so, we can easily control the scale of the mixing coefficient of the shared expert, while still being able to learn the optimal layer-wise mixing coefficients of other normal experts. Let's say W_1^l is the shared expert of the l-th layer, then Eq. (3) and Eq. (4) can be reformulated as below:

$$\overline{W^l} = \lambda W_1^l + \sum_{i=2}^{N} \alpha_i^l W_i^l \tag{5}$$

$$\arg\min_{\alpha} \sum_{j=1}^{m} \mathcal{L}(f(x_j;\theta_o, \overline{W^l}_{1:L}), y_i) \tag{6}$$

In practice, we uniformly initialize the mixing coefficients α of all the normal experts as $\frac{1-\lambda}{N-1}$, which is then trained on the same instruction dataset as upcycling.

4 Main Evaluation

4.1 Experimental Setup

Training. We use DeepSeek-Coder-Base 1.3B (Guo et al., 2024) as the main base code LLM. `evol-codealpaca-v1`, an open-source Evol-Instruct (Luo et al., 2023) dataset containing 110K samples, is used as our instruction dataset. **MoE_{DS}**, our MoE model upcycled from

Model	Size	Instruction Dataset	Dataset Size	Benchmark	
				HumanEval (+)	MBPP (+)
GPT-3.5 (May 2023)	-	Private	-	73.2 (66.5)	-
STABLE-CODE	3B	-	-	28.7 (25.6)	53.6 (44.1)
DeepSeek-Coder-Base	1.3B	-	-	28.7 (25.6)	55.6 (46.9)
Phi-2	2.7B	-	-	48.8 (45.1)	62.7 (52.9)
DeepSeek-Coder-Instruct	1.3B	Private	2B	65.2 (59.8)	63.9 (53.1)
SFT_{DS}	1.3B	Public	0.3B	61.6 (57.3)	59.6 (49.1)
EWA_{DS}	1.3B	Public	0.3B	**67.1** (63.4)	58.9 (48.4)
MoE_{DS}	8×1.3B	Public	0.3B	65.2 (62.2)	60.4 (50.1)
$\mathcal{X}\text{FT}_{\text{DS}}$	1.3B	Public	0.3B	**67.1** (**64.6**)	**60.4** (**50.1**)

Table 1: Pass@1 (%) results of different LLMs on HumanEval (+) and MBPP (+) computed with greedy decoding, following the setting of prior works (Wei et al., 2023; Liu et al., 2023). We report the results consistently from the EvalPlus (Liu et al., 2023) Leaderboard. Note that numbers in bold refer to the highest scores among all 1.3B models fine-tuned on public datasets, which is the same for all the other tables.

the base model, is implemented following Llama-MoE (LLaMA-MoE Team, 2023). It is constructed with 8 experts in one expert layer and the top 6 experts[1] are activated for each token, including one shared expert. As such, we denote the model size of $\textbf{MoE}_{\textbf{DS}}$ as 8×1.3B. Other hyperparameter settings are detailed in Appendix A.1. We finally obtain $\mathcal{X}\textbf{FT}_{\textbf{DS}}$ by using the learned mixing coefficients to merge MoE layers inside MoE_{DS} as normal FFN layers. Note that $\mathcal{X}\textbf{FT}_{\textbf{DS}}$ is the final instruction-tuned LLM we produce, while $\textbf{MoE}_{\textbf{DS}}$ is only an intermediate product of $\mathcal{X}$FT framework.

Baselines. To study the effectiveness of $\mathcal{X}$FT, we build a baseline model, namely $\textbf{SFT}_{\textbf{DS}}$, by directly performing SFT for DeepSeek-Coder-Base 1.3B on `evol-codealpaca-v1`. To compare $\mathcal{X}$FT with EWA (Huang et al., 2023), we also implement a baseline $\textbf{EWA}_{\textbf{DS}}$ and instruction-tune it using the same hyperparameter setting as SFT_{DS}, which is described in Appendix A.1. More implementation details of EWA_{DS} can be seen in Appendix A.2. Furthermore, we incorporate multiple small open-source models (<3B) as our baselines, including DeepSeek-Coder-Base 1.3B, DeepSeek-Coder-Instruct 1.3B (Guo et al., 2024), Phi-2 2.7B, and STABLE-CODE 3B (Pinnaparaju et al.).

4.2 Python Text-to-Code Generation

HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) benchmarks are the two most widely-used collections of Python code generation tasks. We further employ HumanEval+ and MBPP+, which use more tests automatically generated by EvalPlus (Liu et al., 2023) for more rigorous evaluation. We leave the details in Appendix A.3.

Table 1 shows the pass@1 results of different LLMs. $\mathcal{X}$FT achieves 67.1 pass@1 on HumanEval and 64.6 pass@1 on HumanEval+, which makes it the new state-of-the-art small code LLM (<3B). We can also observe that $\mathcal{X}\text{FT}_{\text{DS}}$ has a clear improvement over the SFT_{DS} on both benchmarks, with 13% and 2% improvement on HumanEval+ and MBPP+ respectively, while EWA_{DS} even performs worse than SFT_{DS} on MBPP(+). $\mathcal{X}\text{FT}_{\text{DS}}$ also outperforms EWA_{DS} on both benchmarks. Surprisingly, $\mathcal{X}\text{FT}_{\text{DS}}$ even surpasses MoE_{DS} on HumanEval and HumanEval+, despite only using around $1/8\times$ parameters and around $1/6\times$ computations, which showcases the effectiveness of our simple learnable merging technique.

4.3 Multilingual Code Generation

We use MultiPL-E (Cassano et al., 2022), a multi-programming benchmark that supports 18 programming languages in addition to Python, to evaluate the multilingual ability and generalizability of $\mathcal{X}$FT. Among these, we choose 6 representative programming for their distinct language features: Java, JavaScript, C++, PHP, Swift, and Rust, following Wei et al. (2023). Table 2 shows, among all 1.3B models, $\mathcal{X}\text{FT}_{\text{DS}}$ achieves the best average multilingual performance and performs the best on 5 (out of 6) individual programming languages, overall largely improving SFT_{DS} which uses standard SFT. Notably, the overall performance of EWA_{DS} is on par with SFT_{DS}, indicating

[1] 6 is the best-performing number of activated experts per our HumanEval+ experiments using top $\{2, 4, 6\}$ experts.

Model	Size	Programming Language						**Average**
		C++	PHP	Java	JS	Swift	Rust	
DeepSeek-Coder-Base	1.3B	28.1	22.9	27.2	28.7	10.9	18.0	22.6
SFT_{DS}	1.3B	40.4	38.5	**40.2**	46.2	16.4	27.7	34.9
EWA_{DS}	1.3B	39.4	38.4	37.3	45.2	20.9	28.6	35.0
MoE_{DS}	8×1.3B	42.2	42.2	35.4	49.8	24.7	30.6	37.5
$\mathcal{X}\text{FT}_{\text{DS}}$	1.3B	**42.7**	**41.5**	36.0	**49.7**	**25.3**	**32.1**	**37.9**

Table 2: Pass@1 results on MultiPL-E (Cassano et al., 2022) following the same hyperparameter settings as prior works (Wei et al., 2023; Luo et al., 2023): `temperature` $= 0.2$, `top_p` $= 0.95$, `max_length` $= 512$, and `num_samples` $= 50$. All models are evaluated using `bigcode-evaluation-harness` (Ben Allal et al., 2022).

Model	Size	Data Science Library							**Overall**
		np	pd	plt	py	scp	tf	sk	
DeepSeek-Coder-Base	1.3B	25.1	5.8	34.5	12.7	9.8	11.1	12.7	16.4
SFT_{DS}	1.3B	30.9	17.0	40.5	32.7	18.3	21.1	24.4	25.9
EWA_{DS}	1.3B	32.9	19.4	**41.8**	25.7	17.7	**22.2**	33.0	27.8
MoE_{DS}	8×1.3B	33.2	21.3	38.4	41.8	21.8	23.5	37.5	30.0
$\mathcal{X}\text{FT}_{\text{DS}}$	1.3B	**32.9**	**20.2**	38.9	**41.4**	**21.1**	16.9	**37.5**	**29.3**

Table 3: Pass@1 results on DS-1000 (completion format) with `temperature` $= 0.2$, `top_p` $= 0.5$, `max_length` $= 1024$, and `num_samples` $= 40$, following the same hyperparameter setting used in prior works (Wei et al., 2023).

that EWA_{DS} may not improve SFT on multilingual coding. Appendix A.4 further studies whether each expert in MoE_{DS} specializes differently in these programming languages.

4.4 Code Generation for Data Science

The DS-1000 dataset (Lai et al., 2022) is a collection of 1000 realistic data science coding problems ranging from 7 popular data science libraries in Python, including Matplotlib (plt), NumPy (np), Pandas (pd), SciPy (scp), Scikit-Learn (sk), PyTorch (py), and TensorFlow (tf). We evaluate $\mathcal{X}$FT on DS-1000 to understand its effectiveness for practical data science engineering. We follow the evaluation setting of prior works (Guo et al., 2024; Wei et al., 2023). In Table 3, $\mathcal{X}\text{FT}_{\text{DS}}$ achieves the best overall performance among all the evaluated 1.3B models. Specifically, $\mathcal{X}\text{FT}_{\text{DS}}$ consistently surpasses SFT_{DS} among all the seven studied libraries and also outperforms EWA_{DS} in general.

5 Ablation Study

5.1 Effect of Shared Expert with Routing Weight Normalization

We demonstrate the importance of the shared expert of $\mathcal{X}$FT by comparing its performance with the sparse upcycling (Komatsuzaki et al., 2023) baseline that does not employ any shared expert.

Model	HumanEval	HumanEval+
SFT_{DS}	61.6	57.3
MoE_{DS}	**65.2**	**62.2**
MoE_{DS} - Normalization	63.4	59.1
MoE_{DS} - Shared Expert	61.6	56.7

Table 4: Ablation over the design of MoE_{DS}. "- Normalization" removes the routing weight normalization from the router, making it the same design as MoCLE (Gou et al., 2024). "- Shared Expert" removes the shared expert setting, making MoE_{DS} the same architecture as original sparse upcycling (Komatsuzaki et al., 2023).

This baseline has the same hyperparameter setting and training scheme as $\mathcal{X}$FT. As shown in Table 4, the performance of the original sparse upcycling (with the "- Shared Expert" label) drops greatly compared with MoE_{DS}. Notably, the sparse upcycling model even performs worse than SFT_{DS} on HumanEval+, showing its ineffectiveness for instruction tuning.

While the shared expert setting is also employed in most recent works (Dai et al., 2024; Gou et al., 2024), their routing strategy will cause performance degradation due to the scale mismatch between the outputs of the upcycled MoE layer and the original FFN layer. To understand the impor-

Model	HumanEval	HumanEval+
MoE_{DS}	65.2	62.2
$\mathcal{X}\text{FT}_{\text{DS}}$ (INIT)	66.5	64.0
$\mathcal{X}\text{FT}_{\text{DS}}$	**67.1**	**64.6**
$\mathcal{X}\text{FT}_{\text{DS}}$ - Shared Expert Rate	66.5	64.0

Table 5: Ablation over the design of $\mathcal{X}\text{FT}_{\text{DS}}$. "(INIT)" refers to directly using the initialized mixing coefficients to merge experts without training. "- Shared Rate" removes the shared rate setting from $\mathcal{X}\text{FT}_{\text{DS}}$, which is the same as the learned soup (Wortsman et al., 2022).

tance of routing weight normalization, we conduct an ablation by excluding it from $\mathcal{X}$FT. Table 4 shows that, after removing routing weight normalization, the performance substantially decreases, despite being still better than the original sparse upcycling that does not use the shared expert setting.

5.2 Effect of Merging Strategy

In this section, we demonstrate the effectiveness of our learnable merging technique by comparing it with (1) directly merging experts with initialized mixing coefficients, and (2) the learnable merging technique without the shared rate setting, which is the same setting as the learned soup in Model Soups (Wortsman et al., 2022) and is described in Eq. (3) and Eq. (4). Specifically, we initialize the learnable mixing coefficient of the shared expert as 0.75 and that of the other 7 normal experts as $\frac{1}{28}$ for fair comparison. As is shown in Table 5, trained mixing coefficients outperform the initialized mixing coefficients for merging. Furthermore, removing the shared rate setting will largely degrade the performance of $\mathcal{X}\text{FT}_{\text{DS}}$ on both HumanEval and HumanEval+, demonstrating its importance.

By investigating the learned mixing coefficients without the shared rate setting, we find that most mixing coefficients of the shared experts increase to around 0.753 rather than the original value of 0.75. We argue that such an increase is not helpful because, although the shared expert has learned general knowledge across the whole instruction dataset and needs a relatively large mixing coefficient, we still need to keep the scale of the mixing coefficient of other normal experts at a certain level also to keep some specific knowledge learned by other normal experts.

5.3 Effect of Base Code LLM

In this section, we demonstrate that the effectiveness of $\mathcal{X}$FT is not dependent on the choice of base code LLMs. To show this, we conduct an ablation experiment by applying $\mathcal{X}$FT to STABLE-CODE 3B (Pinnaparaju et al.), whose architecture is different from DeepSeek-Coder-Base 1.3B (Guo et al., 2024), and see whether it can still improve its performance on HumanEval and HumanEval+. Hyperparameter settings are detailed in Appendix A.5. As is shown in Table 6, $\mathcal{X}\text{FT}_{\text{STABLE}}$ significantly improves $\text{SFT}_{\text{STABLE}}$ by 10% on HumanEval and 11% on HumanEval+ respectively. Furthermore, $\mathcal{X}\text{FT}_{\text{STABLE}}$ consistently boosts the performance of $\text{MoE}_{\text{STABLE}}$ while only using $1/4\times$ parameters and $1/2\times$ computations. These results show that the effectiveness of $\mathcal{X}$FT does not depend on any specific choice of base code LLMs, demonstrating the generalizability of $\mathcal{X}$FT.

Model	HumanEval	HumanEval+
$\text{SFT}_{\text{STABLE}}$	62.2	56.1
$\text{MoE}_{\text{STABLE}}$	64.0	59.1
$\mathcal{X}\text{FT}_{\text{STABLE}}$	**68.3**	**62.2**

Table 6: Ablation over the effect of the base model by replacing DeepSeek-Coder-Base 1.3B with STABLE-CODE 3B. $\mathcal{X}$FT can consistently improve the instruction tuning performance of different base code LLMs.

6 Conclusion

This paper introduces $\mathcal{X}$FT to unlock the power of code instruction tuning by simply merging upcycled MoE. Similar to SFT, $\mathcal{X}$FT starts with a dense LLM and produces a fine-tuned dense LLM with the exact size and model structure. Yet, $\mathcal{X}$FT improves SFT by upcycling the pre-trained dense LLM to an MoE model for fine-tuning, after which we compile the MoE model back to an efficient dense LLM with a learnable merging mechanism. As such, we unleash the performance limit of instruction tuning without any additional inference overhead. Using the same dataset, $\mathcal{X}$FT improves SFT on a variety of benchmarks, including HumanEval(+), MBPP(+), MultiPL-E, and DS-1000, from 2% to 13%. By applying $\mathcal{X}$FT to DeepSeek-Coder-Base 1.3B, we create the next state-of-the-art small (<3B) LLM for code. The ultimate dense LLM produced by $\mathcal{X}$FT preserves or even outperforms the full upcycled MoE which uses $8\times$ parameters as much as our final dense LLM. $\mathcal{X}$FT is fully orthogonal to the existing instruction tuners such as SELF-INSTRUCT, Evol-Instruct, and OSS-INSTRUCT, opening a new dimension to maximal code instruction tuning.

7 Limitations

While $\mathcal{X}$FT has proven to be effective through extensive experiments in the paper, we apply our technique to LLMs with no more than 3B parameters due to resource constraints. This limitation hinders our ability to showcase the impact of $\mathcal{X}$FT on larger models. In addition, to balance the general knowledge in the shared expert and the specific knowledge in other normal experts, we introduce a hyperparameter λ in the merging process of $\mathcal{X}$FT, which might slightly increase the efforts for hyperparameter search. It would be interesting to explore other hyperparameter-free techniques to tackle this challenge in the future. Furthermore, while $\mathcal{X}$FT has been empirically proven powerful, it would be interesting to provide a theoretical explanation for its strong performance.

References

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

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A framework for the evaluation of code generation models. https://github.com/bigcode-project/bigcode-evaluation-harness.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2022. Multipl-e: A scalable and extensible approach to benchmarking neural code generation.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet?

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri 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, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew 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.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Loramoe: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. Glam: Efficient scaling of language models with mixture-of-experts.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.

Yunhao Gou, Zhili Liu, Kai Chen, Lanqing Hong, Hang Xu, Aoxue Li, Dit-Yan Yeung, James T. Kwok, and Yu Zhang. 2024. Mixture of cluster-conditional lora experts for vision-language instruction tuning.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. 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.

Yongqi Huang, Peng Ye, Xiaoshui Huang, Sheng Li, Tao Chen, Tong He, and Wanli Ouyang. 2023. Experts weights averaging: A new general training scheme for vision transformers.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian,

Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2023. Sparse upcycling: Training mixture-of-experts from dense checkpoints.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2022. Ds-1000: A natural and reliable benchmark for data science code generation.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

LLaMA-MoE Team. 2023. Llama-moe: Building mixture-of-experts from llama with continual pretraining.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct.

Zohar Manna and Richard J Waldinger. 1971. Toward automatic program synthesis. *Communications of the ACM*, 14(3):151–165.

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, , and Nathan Cooper. Stable code 3b.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time.

Haoyuan Wu, Haisheng Zheng, and Bei Yu. 2024. Parameter-efficient sparsity crafting from dense to mixture-of-experts for instruction tuning on general tasks.

Lemeng Wu, Mengchen Liu, Yinpeng Chen, Dongdong Chen, Xiyang Dai, and Lu Yuan. 2022. Residual mixture of experts.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions.

Fuzhao Xue, Xiaoxin He, Xiaozhe Ren, Yuxuan Lou, and Yang You. 2022. One student knows all experts know: From sparse to dense.

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. 2024. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2023. Instruction tuning for large language models: A survey.

A Appendix for "$\mathcal{X}$FT: Boosted Instruction Tuning of Code Language Models by Simply Merging Upcycled Mixture-of Experts"

A.1 Hyperparameter Settings

We use a batch size of 64 and a learning rare of 5e-5 with a linear scheduler to fine-tune $\mathbf{MoE_{DS}}$ for 4 epochs with 500 warmup steps, following the implementation of previous work (Wei et al., 2023). We further use a batch size of 64, a shared expert rate λ of 0.75, and a learning rare of 1e-5 with a linear schedule to fine-tune the learnable mixing coefficients for each of the experts in the instruction-tuned $\mathbf{MoE_{DS}}$ on the instruction dataset for 1 epoch with 125 warmup steps. Detailedly, we use Softmax to keep the sum of the mixing coefficients of the other 7 normal experts as 0.25. For $\mathbf{SFT_{DS}}$ and $\mathbf{EWA_{DS}}$, we use the same hyperparameter setting as $\mathcal{X}$FT, where the batch size is 64 and

the learning rate is 5e-5 with a linear scheduler. Because $\mathcal{X}$FT is trained for 4 epochs during upcycling and 1 epoch during merging, for a fair comparison, we train SFT_{DS} and EWA_{DS} for 5 (= 4 + 1) epochs with 625 warmup steps.

A.2 Implementation details of EWA

Because EWA (Huang et al., 2023) does not release their implementation, we implemented EWA by ourselves, including constant schedule and linear schedule. We use a share rate β of 0.3, following the original setting of EWA. While EWA with the constant schedule achieves reasonable performance in our evaluation, the training loss of EWA with the linear schedule becomes very unstable, as is shown in Figure 3, and thus cannot achieve reasonable performance. As a result, we report the results of EWA with the constant schedule in Section 4.

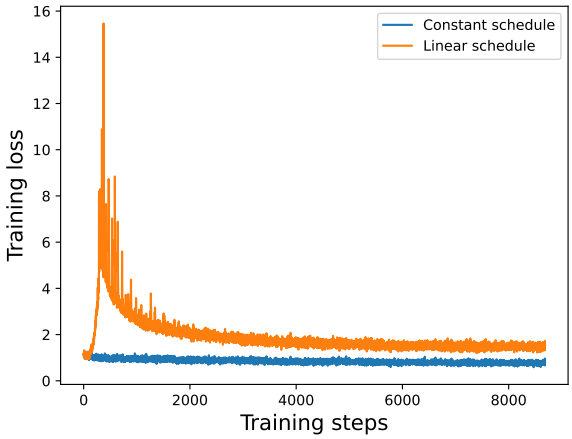

Figure 3: Training loss curve of EWA with constant schedule and linear schedule.

A.3 Details of HumanEval and MBPP

In these benchmarks, each task consists of a task description in English, which is sent to LLMs as the prompt, and LLMs are expected to generate the corresponding code to satisfy the requirements in the description. While these benchmarks provide a handful of test cases to validate the correctness of the generated code, these tests are often insufficient for more rigorous evaluation. As such, HumanEval+ and MBPP+ proposed by EvalPlus (Liu et al., 2023) are usually used to evaluate the correctness of the generated code, which provides 80×/35× more tests compared with the original benchmarks.

A.4 Analysis on Expert Specialization

Inspired by recent works (Jiang et al., 2024; Xue et al., 2024), we analyze whether each expert in MoE_{DS} has different specializations in different programming languages by visualizing the routing decision of the tokens from different programming languages in the MultiPL-E benchmark (including Python). For the MultiPL-E benchmark, we collect the routing decision when conducting experiments in Section 4.3. For Python, we collect the routing decision by reruning HumanEval experiment following the same setting as Section 4.3. Following Mixtral (Jiang et al., 2024), we get the visualization results from layers 0, 11, and 23 in MoE_{DS}, where layer 0 and layer 23 are the first and the last layers of MoE_{DS}. As is shown in Figure 4, we do not observe obvious patterns in the assignment of experts based on the programming language, which is in line with the observation reported by recent works (Jiang et al., 2024; Xue et al., 2024).

A.5 Training Settings for STABLE-CODE 3B

We use `evol-codealpaca-v1` as the training dataset. Since STABLE-CODE 3B is the base model now, we upcycle a new MoE model from the base model, namely $\textbf{MoE}_{\textbf{STABLE}}$. Due to limited computational resources, we construct $\text{MoE}_{\text{STABLE}}$ with 4 experts in one expert layer, where the top 2 experts are activated for each token, including one shared expert. Consequently, the size of $\text{MoE}_{\text{STABLE}}$ can be described as 4×3B. We use a batch size of 64 and a learning rate of 5e-5 with a linear scheduler to fine-tune $\text{MoE}_{\text{STABLE}}$ for 4 epochs with 500 warmup steps. Similar to $\mathcal{X}\text{FT}_{\text{DS}}$, we obtain $\mathcal{X}\textbf{FT}_{\textbf{STABLE}}$ by learning mixing coefficients to merge MoE layers inside $\text{MoE}_{\text{STABLE}}$ as normal FFN layers, which is fine-tuned with a batch size of 64, a shared expert rate λ of 0.85, and a learning rate of 1e-5 with a linear schedule for 1 epoch with 125 warmup steps. We also build a baseline model $\textbf{SFT}_{\textbf{STABLE}}$, which is fine-tuned for 5 epochs under the same settings for a fair comparison.

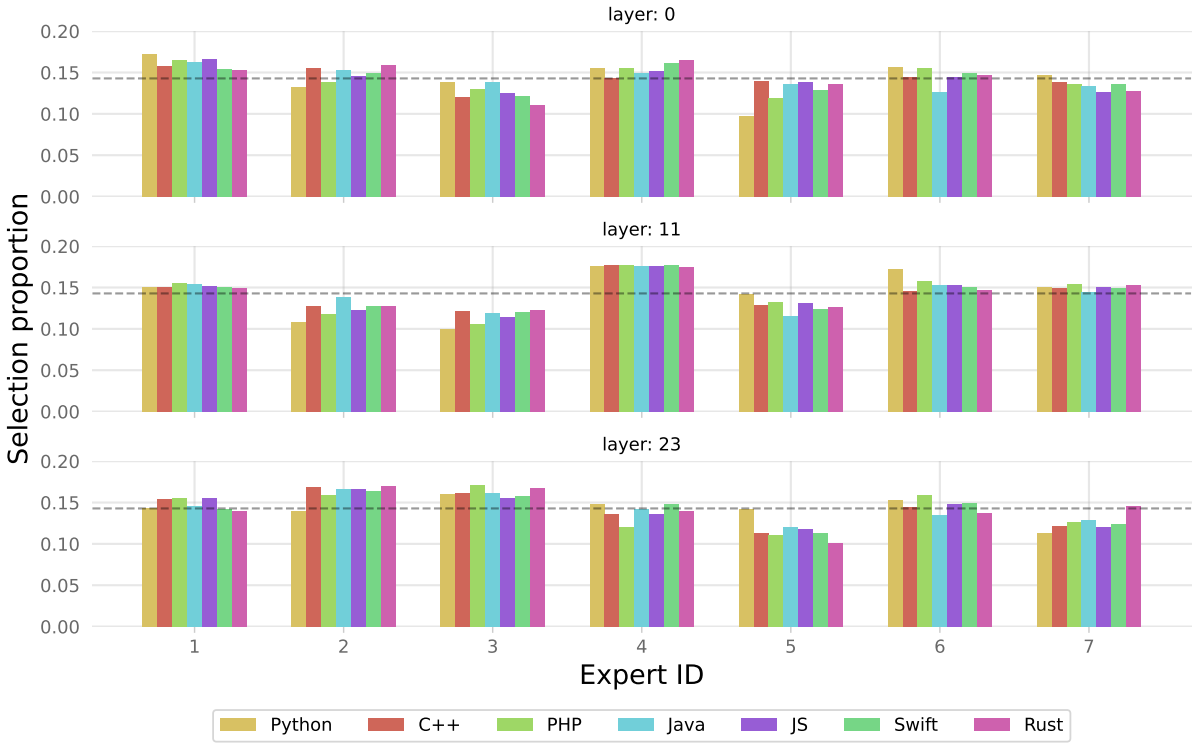

Figure 4: Proportion of tokens assigned to each expert on different programming languages from MultiPL-E (including Python) for layers 0, 11, and 23. The shared expert 0 is excluded from the chart because all the tokens are always assigned to it. The gray vertical line marks $\frac{1}{7}$, which is the proportion expected with the uniform sampling.