FastDraft: How to Train Your Draft

Ofir Zafrir[*], Igor Margulis[*], Dorin Shteyman[*], Shira Guskin, Guy Boudoukh
Intel Labs
{ofir.zafrir,igor.margulis,dorin.shteyman}@intel.com
{shira.guskin,guy.boudoukh}@intel.com

Abstract

Speculative Decoding has gained popularity as an effective technique for accelerating the auto-regressive inference process of Large Language Models. However, Speculative Decoding entirely relies on the availability of efficient draft models, which are often lacking for many existing language models due to a stringent constraint of vocabulary compatibility. In this work we introduce FastDraft, a novel and efficient approach for pre-training and aligning a draft model to any large language model by incorporating efficient pre-training, followed by fine-tuning over synthetic datasets generated by the target model. We demonstrate FastDraft by training two highly parameter efficient drafts for the popular Phi-3-mini and Llama-3.1-8B models. Using FastDraft, we were able to produce a draft model with approximately 10 billion tokens on a single server with 8 Intel® Gaudi® 2 accelerators in under 24 hours. Our results show that the draft model achieves impressive results in key metrics of acceptance rate, block efficiency and up to 3x memory bound speed up when evaluated on code completion and up to 2x in summarization, text completion and instruction tasks. We validate our theoretical findings through benchmarking on the latest Intel® Core™ Ultra, achieving a wall-clock time speedup of up to 2x, indicating a significant reduction in runtime. Due to its high quality, FastDraft unlocks large language models inference on AI-PC and other edge-devices.

1 Introduction

The advent of Transformer architectures has fundamentally reshaped the field of natural language processing (NLP). In recent years, Transformer-based models have achieved remarkable success across a broad spectrum of natural language understanding and generation tasks (Achiam et al., 2023; Team et al., 2023). Their exceptional performance, particularly in large language models (LLMs), has made them highly desirable for deployment in numerous applications, ranging from conversational systems to content generation and beyond. Despite their outstanding performance, LLMs suffer from slow inference speed due to substantial memory bandwidth requirements and the sequential nature of auto-regressive generation (ARG). The introduction of Speculative Decoding (SD) (Leviathan et al., 2023) offers a promising solution for accelerating ARG without sacrificing generation quality, making it a compelling approach for improving LLM inference efficiency. SD utilizes a draft language model (LM) to generate a sequence of tokens auto-regressively, while the target model validates the batched tokens in parallel. In certain applications, SD can achieve a 2-3x speedup in LLM inference without compromising the generation quality of the target model. Achieving significant speedup with SD requires a high-quality draft model that is both efficient and well-aligned with the target. To date, such draft models remain scarce, even for widely used open-source LLMs (Dubey et al., 2024; Abdin et al., 2024) due to vocabulary incompatibility. To address this limitation, we propose FastDraft, a method for producing hardware (HW) efficient draft models that are orders of magnitude smaller than their corresponding target models.

While extensive research has focused on training and data generation for high-quality LLMs (Kaplan et al., 2020; Longpre et al., 2023), these frameworks are not necessarily applicable to training draft models for SD. LLMs are typically trained to generate helpful, high-quality responses, whereas draft models should be trained to generate sequences that are likely to be accepted by the target model. We explore previously unexamined aspects of draft model training for SD. Our contributions include: (1) introducing a method for producing quality and highly efficient draft models with low

[*]Equal contribution

resource requirements for any given target LLM and demonstrate it by training and benchmarking a draft for Phi-3-mini, (2) conducting extensive ablation studies on pre-training data size, pre-training for both code and natural language, target-draft alignment via knowledge distillation (KD) and HW-aware draft architectures, and (3) demonstrating the scalability of FastDraft by training a draft model for Llama-3.1-8B-Instruct, achieving performance improvements comparable to those attained for Phi-3-mini. We demonstrate significant improvements in key metrics using FastDraft, leading to theoretical speedups of up to 3x as measured by the Memory Bound Speedup (MBSU) metric. We further validate our theoretical findings by evaluating the wall-clock time speedup achieved with our approach on the latest Intel® Core™ Ultra. Our results demonstrate an average 1.5x speedup for natural language tasks and an average 2x speedup for code completion tasks. According to our findings, the small size of the draft model and the limited amount of data required to produce a high-quality draft enabled us to successfully train and align a draft model to Phi-3-mini, end-to-end, in under 24 hours using a single node with 8 Intel® Gaudi® 2 accelerators.

2 Related work

The widespread adoption of LLMs in cloud and edge devices has driven a significant body of research focused on developing alternative strategies for ARG to address the low inference efficiency of LLMs (Santilli et al., 2023; Ghazvininejad et al., 2019; Stern et al., 2018). However, many of these approaches compromise generation quality or require additional training data and architectural modifications. The introduction of speculative decoding as a lossless approach for accelerating LLM inference (Leviathan et al., 2023) has inspired a new wave of follow-up research. Some studies propose using plug-in prediction heads as a drafting mechanism (Zhang et al., 2024; Cai et al., 2024), while others focus on improving the serving latency of stand-alone draft models (Sun et al., 2024; Chen et al., 2023; Miao et al., 2023). In contrast, our work focuses on directly enhancing stand-alone draft model's capabilities through pre-training and fine-tuning. We provide more details on the comparison with these methods in Appendix A. Other works apply KD to draft models to improve alignment with the target model. These studies explore various divergence functions for the KD algorithm, rather than relying solely on the commonly used Kullback–Leibler Divergence (KLD). For instance, (Zhou et al., 2023) proposed Total Variation Distance (TVD) based on Corollary 3.6 in (Leviathan et al., 2023), which posits that minimizing TVD maximizes the token-level acceptance rate. (Goel et al., 2024) further developed this approach with TVD++, fine-tuning their model, pre-trained on 600 billion tokens. Another study, (Yan et al., 2024), focuses on enhancing the HW efficiency of draft models by extensively analyzing the trade-off between time latency and acceptance rates, rather than relying solely on the latter. While their work emphasizes the approach for obtaining a draft model by pruning of the shelf models, our method introduces an efficient approach for pre-training and aligning a draft model. (Li et al., 2024) proposes training a compact draft model based on the target model, using a relatively limited dataset. The primary aim of the paper is to develop a draft architecture that effectively utilizes the target model's hidden representations and weights. In contrast, our paper centers on training and aligning any draft architecture that shares only the vocabulary with the target model.

3 Speculative decoding

SD is a lossless decoding paradigm introduced by (Leviathan et al., 2023) for accelerating ARG with LLMs. It is inspired by speculative execution (Hennessy and Patterson, 2017) and aims to mitigate the inherent latency bottleneck caused by the sequential nature of ARG (Pope et al., 2023). SD employs a draft LM to generate a block of γ candidate tokens. The LLM, referred to as the target model, then processes these candidate tokens in parallel. The algorithm examines each token's probability distribution, calculated by both the target and draft models, to determine whether the token should be accepted or rejected. As a result, any LM can function as a draft model, provided it shares the same vocabulary as the target model. However, since an LM's vocabulary is fixed during pre-training, leveraging existing models as draft models is only feasible if they were pre-trained on the same vocabulary. A key metric for benchmarking ARG inference performance is time per output token (TPOT), which represents the expected latency for generating a single output token, excluding pre-filling latency.[1] In

[1] Pre-filling refers to the action of populating the key-value cache with the input tokens' information.

SD, TPOT is a function of the draft model latency l_D, the speculation block size γ, the corresponding expected block efficiency[2] τ^γ, and the target model latency l_T^γ. The equation is expressed as:

$$\mathrm{TPOT_{SD}} = \frac{l_D \times \gamma + l_T^\gamma}{\tau^\gamma} \tag{1}$$

In comparison, for traditional ARG, TPOT is simply $\mathrm{TPOT_{AR}} = l_T^1$. The expected speedup is calculated as follows:

$$\frac{\mathrm{TPOT_{AR}}}{\mathrm{TPOT_{SD}}} = \left(\frac{l_D}{l_T^1} \times \gamma + \frac{l_T^\gamma}{l_T^1}\right)^{-1} \times \tau^\gamma \tag{2}$$

From Equation 2, two key requirements emerge for SD to yield meaningful speedups. First, the latency for generating the speculated tokens block must be negligible compared to the target model's latency. Second, the increase in target model latency for block size γ should be insignificant compared to the latency for block size 1. The latter condition generally holds for most popular LLMs with sufficiently small γ, while the former depends on the availability of draft models that meet the vocabulary constraint.

4 FastDraft: Build your own draft model

4.1 Draft architecture & pre-training

The draft architecture imposes only one strict requirement, it must produce a probability distribution over the target's vocabulary. Beyond this, the design is flexible. Nevertheless, certain factors should be considered when selecting the draft's architecture, with latency being the primary concern, as discussed in Section 3. Then, the chosen draft is trained over a pre-training dataset of natural language for language modeling with the objective of predicting the next token. A common application of LLMs is code completion. While the pre-training dataset may include some code, unless the dataset is specifically focused on code, the draft's performance on tasks requiring an understanding of code is often suboptimal. To address this, (Aryabumi et al., 2024) proposed continued pre-training (CP), where training begins with a pre-trained model and is extended using a combination of code and natural language data. FastDraft adopts this approach for producing drafts for code completion tasks.

[2] The expected number of tokens accepted within a speculation block.

4.2 Target-draft alignment

One of the primary objectives when designing a draft model is to maximize the acceptance rate with the target model, as a higher acceptance rate directly leads to greater speedup in SD. To closely mimic the target model's behavior in real-world scenarios, we investigate two KD strategies that expose the draft model to data samples that closely reflect those generated by the target model.

Strategy 1: sequence-level knowledge distillation (Kim and Rush, 2016) proposed performing KD by training the student model on sequences generated by the teacher model. This approach is widely used in LLM training to improve model quality (Taori et al., 2023; Abdin et al., 2024). To align the pre-trained draft model with the teacher model, we employ sequence-level KD by fine-tuning the draft model on a synthetic dataset generated by the target. In contrast to the sequence-level KD approach proposed by (Kim and Rush, 2016), we differentiate between training on teacher-generated data and training with KD using the teacher's logits. Accordingly, in this work, sequence-level KD refers specifically to training on generated tokens with cross-entropy loss, without incorporating the teacher's soft targets (logits).

Strategy 2: token-level knowledge distillation This strategy aligns with the traditional KD method presented in (Hinton, 2015), which involves calculating the divergence of the token level probability distribution over the vocabulary between the teacher and the draft. Since computing the KD loss depends on the teacher's logits, the teacher is required to perform inference at every training step of the draft, making this process both computationally intensive and time-consuming. An alternative approach is to precompute the teacher's logits beforehand and embed them into the dataset. However, the memory demands of such a dataset can quickly exceed several terabytes, as the number of logits per token equals the vocabulary size[3] posing a substantial memory overhead. To mitigate this issue, only a small subset of the most significant logits per token is extracted, significantly reducing the memory requirements. This optimization results in a 6x-9x reduction in fine-tuning time without noticeable degradation in model quality.

[3] The vocabulary size usually exceeds 30,000 tokens

4.3 Draft evaluation

To scale both our experiments and evaluations, we created the FastDraft evaluation framework, specifically designed for assessing draft models. The metrics and benchmarks we implemented in FastDraft evaluation are detailed in Sections 4.3.1 and 4.3.2. Optimizing these key metrics on the proposed benchmarks is our key objective in this work.

4.3.1 Metrics

Acceptance rate A key metric which reflects the rate at which the target model accepts the draft model's speculated tokens. In this work, we calculate the acceptance rate (AR) α^γ by determining the expected number of tokens accepted per block normalized by the block's size. The formula is expressed as:

$$\alpha^\gamma = \frac{1}{N} \sum_{n=0}^{N} \frac{\#(\text{accepted tokens})}{\gamma} \tag{3}$$

where N is the number of blocks speculated by the draft model during evaluation, and γ is the block size.

Block efficiency The common use of SD with fixed-size blocks motivates the introduction of a more relevant efficiency metric: block efficiency, τ^γ. This is defined as the expected number of accepted tokens per block. For a given block size γ, block efficiency τ^γ serves as a more accurate measure of performance, reflecting the average rate at which tokens are accepted within each block.

$$\tau^\gamma = 1 + \alpha^\gamma \times \gamma \tag{4}$$

Wall-clock time and Memory-Bound Speedup Given the block efficiency τ^γ, the expected speedup achieved by applying Speculative Decoding (SD) is expressed as $\frac{\tau^\gamma}{c\gamma+1}$, where c represents the ratio of latencies between the draft and target models. Since this metric is HW-dependent, it is preferable to use a HW-agnostic measure. Considering that the expected speedup occurs in a memory-bound regime, we utilize the Memory-Bound Speedup (MBSU). We define $\hat{c}$ as the ratio of parameter counts between the draft and target models.

$$\text{MBSU} = \frac{\tau^\gamma}{\hat{c}\gamma + 1} \tag{5}$$

4.3.2 Benchmarks

The most commonly used benchmarks for assessing the quality of draft models, typically open-ended generation and summarization tasks, include the XSum (Narayan et al., 2018), CNN-DailyMail (CNN-DM) (Hermann et al., 2015), and HumanEval (Chen et al., 2021) datasets. In addition to CNN-DM and HumanEval, we include in our evaluation the TinyStories (TS) (Eldan et al., 2023) and Dolly (Conover et al., 2023) datasets, providing a more comprehensive assessment of model performance across diverse tasks. When utilizing the TS dataset, we generate text by starting from a random position within each sample, using the preceding tokens as the input context.

5 Experiments

In this section we report descriptions, setup and results of our experiments. In the following experiments, we report acceptance rate (Section 4.3.1) results using two sampling methods, greedy and multinomial sampling decoding with temperature $T = 0.6$ with block sizes $\gamma = 3$ and $\gamma = 5$ unless stated otherwise. The hyper-parameters used in our draft pre-training experiments are detailed Appendix B.3 unless stated otherwise.

5.1 Experimental setup

We demonstrate FastDraft by training a draft model for Phi-3-mini-4k-Instruct (Abdin et al., 2024) target model. Phi-3-mini, a 3.8 billion parameters LLM, was selected as our case study due to its outstanding performance across multiple open-source LLM benchmarks, while also being capable of running locally on edge devices, such as personal computers and smartphones.

Draft architecture Exploring different architectures for draft models lies beyond the scope of this paper. Therefore, our experiments focus on drafts with the Phi-3 architecture, modified to smaller dimensions. Specifically, we reduced the dimensions of Phi-3 to create drafts in the size of 50M and 120M parameters which are approximately 76x and 32x smaller than Phi-3-mini. Full details of drafts configurations can be found in Table 5 in the Appendix.

Pre-training datasets For our natural language dataset, we use a 10 billion tokens (BT) sample from the FineWeb dataset (Penedo et al., 2024). FineWeb is a widely used open-source, deduplicated, and quality-filtered dataset, comprising 15 trillion tokens derived from 96 Common Crawl snapshots (Crawl). It has been demonstrated to yield better-performing LLMs compared to other open-source pre-training datasets such as C4, Red-Pajama, and The Pile (Section 3.7 in (Penedo et al., 2024)). For our code dataset, we use a 10BT sample from The Stack v2 smol dataset (Lozhkov et al., 2024). This variant of The Stack v2 consists of 17 commonly used programming languages, as well as a substantial collection of documentation languages, configuration languages, and configuration files. An overview of the 10BT dataset composition is provided in Appendix B.2.

Synthetic alignment dataset To perform sequence-level KD as discussed in Section 4.2, we produce an instruction fine-tuning collection using the seed instructions from several open instruction datasets: Alpaca (Taori et al., 2023), OIG-small-chip2 (Huu et al., 2023) and Evol-Instruct (Luo et al., 2023). We collect response sequences generated by Phi-3-mini with greedy sampling along with multinomial sampling with temperature in $\{0.6, 0.8, 1.0\}$ to improve the diversity of the generated sequences. Additionally, we adopt the approach proposed by (Xu et al., 2024), directly soliciting instructions from the target model. These instructions are then used to generate the corresponding responses in the same manner as previously described.

5.2 Pre-training dataset size

We study the impact of the pre-training dataset size on the performance of the draft model, aiming to optimize runtime and resource usage and prevent diminishing returns. Consequently, we uniformly sample subsets of sizes $\{0.1, 0.5, 1, 2, 5, 10\}$BT and pre-train the 50M and 120M draft configurations on these subsets. We report the AR results for the $\{2, 5, 10\}$BT subsets with block size $\gamma = 3$ and multinomial sampling in addition to perplexity results measured on Wikitext2 (Salesforce) in Table 1. The full results are reported in Table 7 in the Appendix.

As anticipated, perplexity values decrease as the models are exposed to more training data, though the rate of improvement slows down. However, when looking at the AR results, it is not the case. In CNN-DM and Dolly AR either plateaus or decreases as the data size grows. In the case of TS, AR increases with the amount of data which can be expected due to the nature of the benchmark of text completion versus instruction following and summarization. Overall, we obtain strong results across all dataset sizes, with the models trained on the 5BT dataset emerging as a promising middle-ground option. When comparing the 50M draft to the 120M draft, the 120M draft demonstrates superior performance in terms of AR. Additionally, while the 120M draft also outperforms the 50M model in MBSU, the margin of improvement is relatively small, see Figure 1. However, it is important to note that training the 120M draft requires twice the time compared to the 50M draft.

For the subsequent experiments, we selected the 50M draft model trained on 5BT tokens as our pre-trained draft, unless otherwise specified.

Draft	Data size	PPL	CNN-DM	TS	Dolly
50M	2BT	297.4	**0.323**	0.264	0.241
	5BT	256.6	0.311	0.277	**0.245**
	10BT	**240.9**	0.312	**0.283**	0.234
120M	2BT	199.6	0.362	0.297	**0.284**
	5BT	167.7	**0.366**	0.327	0.281
	10BT	**147.4**	0.351	**0.331**	0.251

Table 1: Pre-training data size effect on AR and perplexity (PPL). Results for block size $\gamma = 3$ and multinomial sampling with temperature $T = 0.6$

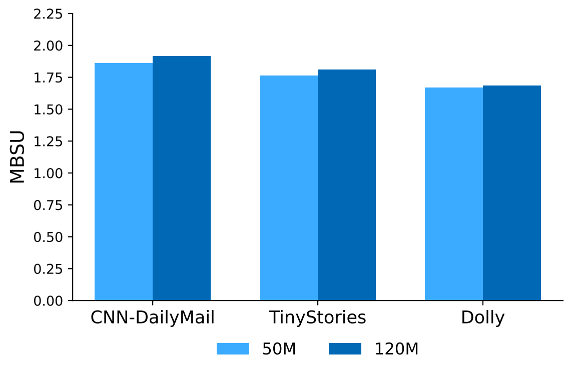

Figure 1: MBSU of 50M and 120M drafts pre-trained on 5BT FineWeb sample.

5.3 Continued pre-training for code

We investigate the continued pre-training (CP) method introduced by (Aryabumi et al., 2024) to refine our draft model on code-related tasks. In this approach, we extend the training of our pre-trained draft model using three distinct combinations of code and natural language datasets. Each combination incorporates 5BT of code data, along with varying amounts of natural language data: specifically,

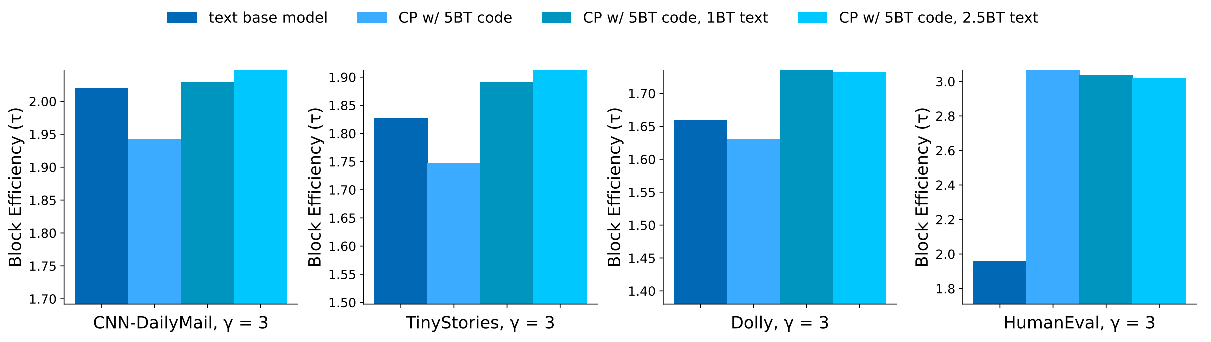

Figure 2: Block efficiency results for continued pre-training with code on tasks: CNN-DM, TS, Dolly and HumanEval using greedy decoding with block size $\gamma = 3$

$0, 1, 2.5$BT. We then evaluate the resulting models on our benchmarks, including the HumanEval benchmark, and present the block efficiency results, τ^{γ}, in Figure 2. Our findings indicate that mixed CP datasets lead to significantly higher block efficiency in natural language tasks compared to using CP with code alone. Additionally, block efficiency improves across natural language benchmarks, surpassing the performance of the base draft model that was trained solely on natural language, with only a slight decline observed in the Dolly benchmark. Notably, the base draft shows a substantial improvement on the HumanEval benchmark. The enhancement in the draft's performance on natural language tasks with CP was anticipated, based on the findings of (Aryabumi et al., 2024). However, unlike the conclusions presented in that study, which advocate for a balanced dataset of natural language and code, our results suggest that CP is the preferred approach for integrating code and natural language datasets when training a draft model.

Given the structured nature of code, it serves as a highly suitable domain for generation with SD. To investigate how to optimize pre-training for code drafts, we conducted a comprehensive ablation study. This study involved pre-training from scratch on a mixture of code and pre-training with CP approach, where the base draft model was pre-trained on code, and CP was applied to adapt the model for natural language instead of code. Detailed results and analysis of this ablation study are provided in Appendix B.5. Overall, our findings indicate that initializing continued pre-training from a text-based model using a mixed CP dataset offers the most effective pre-training strategy for draft models, compared to other configurations tested.

Sampling	Data source	CNN-DM	TS	Dolly
Greedy	None	0.350	0.295	0.259
	Original	0.357	0.282	0.328
	Target	**0.378**	**0.296**	**0.370**
Multinomial	None	0.311	0.227	0.245
	Original	0.328	0.268	0.311
	Target	**0.339**	**0.286**	**0.352**

Table 2: AR results for fine-tuning with original data vs synthetic data. Results for block size $\gamma = 3$

5.4 Fine tuning with synthetic data

Since our objective is to achieve the best draft alignment with the target model rather than the quality of generation, we investigate the impact of using the target's responses instead of the original dataset responses, as described in Section 4.2. We measure the draft's performance across several tasks. In this experiment, we utilize the OIG-small-chip dataset for fine-tuning. We generate the target's responses with multinomial sampling at temperature of 0.6. Token-level KD was not applied in this experiment. The results are presented in Table 2. An analysis of the data reveals that using the target's answers leads to better draft alignment compared to using the original dataset answers across all tasks.

5.5 Fine tuning with knowledge distillation

To further enhance the alignment of our base draft model, we incorporate KD alongside training on target-generated data, as discussed in Section 4.2. We experimented with various combinations of Cross Entropy (CE) loss, denoted as $\mathcal{L}_{CE}$, applied to the ground truth labels, and KD losses, using the sparse logits collected from the target model: $\mathcal{L}_{KL}$ and $\mathcal{L}_{TVD}$. We utilize the same dataset we generated in Section 5.4 for this experiment. The results of these experiments are presented in Table 3. Our findings indicate that KD in this setup doesn't

Sampling	Loss	CNN-DM	TS	Dolly
Greedy	$\mathcal{L}_{CE}$	0.378	0.296	0.370
	$\frac{1}{2}\mathcal{L}_{CE} + \frac{1}{2}\mathcal{L}_{KL}$	0.376	0.295	0.368
	$\frac{1}{2}\mathcal{L}_{CE} + \frac{1}{2}\mathcal{L}_{TVD}$	0.377	0.297	0.370
	$\mathcal{L}_{KL}$	0.374	0.294	**0.371**
	$\mathcal{L}_{TVD}$	**0.384**	**0.301**	0.362
Multinomial	$\mathcal{L}_{CE}$	0.339	0.286	**0.352**
	$\frac{1}{2}\mathcal{L}_{CE} + \frac{1}{2}\mathcal{L}_{KL}$	0.350	0.289	0.346
	$\frac{1}{2}\mathcal{L}_{CE} + \frac{1}{2}\mathcal{L}_{TVD}$	**0.356**	**0.297**	0.347
	$\mathcal{L}_{KL}$	0.355	0.280	0.343
	$\mathcal{L}_{TVD}$	0.347	**0.297**	0.344

Table 3: KD with KL-Divergence and TVD. Results for block size $\gamma = 3$

provide significant advantage over CE loss on the target-generated data. Although Total Variation Distance (TVD) was shown to negatively correlate with acceptance rates, in our results, it offers only slight benefit on some benchmarks over CE and Kullback–Leibler Divergence (KLD).

5.6 Hardware-aware draft

The high quality of models like Phi-3-mini and Llama-3.1-8B, combined with advances in edge device hardware, has made local deployment of LLMs increasingly appealing, yet still challenging. Typical use cases for locally deployed LLMs share similar characteristics, they are designed for single-user interactions where only one query is processed at a time. This setup presents an excellent opportunity to apply speculative decoding to accelerate LLM generation with minimal resource demands.

The Phi3-mini 50M uses the original Transformers dimensions, however, due to the strict latency requirements of draft models, their architectural design warrants special attention, as their performance may vary depending on the hardware platform. We conduct a series of experiments to examine the performance of draft models derived from different architectural choices. Our focus is on altering the depth of the models by changing the number of layers or the width of the models by modifying the hidden dimension. The target system for our experiments is the latest Intel® Core™ Ultra 7 Processor 268V coupled with Intel® OpenVINO™ runtime engine.

First, we analyze the effect of scaling the width and depth of the draft on latency of the draft model on the target hardware. We fix either the hidden dimension or the model depth to match the corresponding values of Phi3-mini 50M draft and modify the other parameter to observe how latency scales with wider or deeper models. As shown

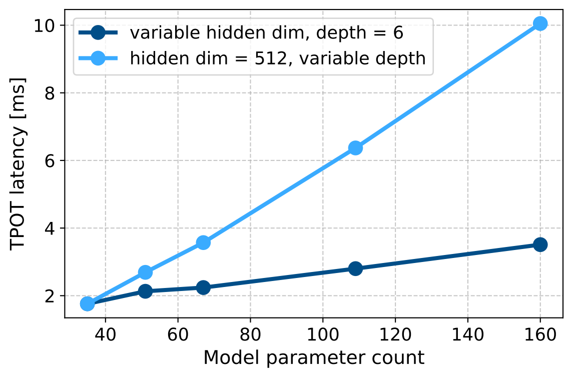

Figure 3: Latency of models obtained by varying either a hidden dimension or a number of layers

in Figure 3, our benchmarking results suggest that increasing the number of layers has the most significant impact on latency, while increasing the hidden dimension leads to only a slight increase.

Next, we evaluate the impact of the draft architecture under a fixed parameter budget[4] by assessing the AR and estimated speedup achieved when varying the width and depth of the draft. By redistributing parameters between these dimensions, we aimed to identify configurations that optimize the trade-off between speculation quality and computational efficiency. We pre-trained multiple draft models for Phi-3-mini, varying the number of layers and hidden dimension sizes while maintaining a fixed parameter count. We then measured their AR on a target dataset and recorded their latency on our target hardware to estimate the speedup achieved through SD via Equation 2. Figure 4 presents the results for the CNN-DM dataset. Our findings indicate that increasing depth does not improve the AR, while excessively shallow drafts degrade the performance. Moreover, the AR–latency trade-off follows an inverted U-shaped speedup curve with a broad base, consistent with our earlier experimental results. The results for the TS and Dolly datasets, presented in Appendix C, exhibited the same pattern.

6 Results & reproducibility

Combining the findings from the ablation studies presented in Section 5, we outline our comprehensive pipeline for draft pre-training and fine-tuning to optimize performance on key metrics for speculative decoding and demonstrate it by producing

[4] We excluded embedding table size from the model parameter budget, as it does not strain memory bandwidth.

Model	Type	CNN-DM		TS		Dolly		HumanEval	
		$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$
Phi3-mini 50M	PT	0.311	0.221	0.277	0.184	0.245	0.163	0.229	0.151
	$PT \to CP$	0.304	0.211	0.287	0.192	0.226	0.149	0.561	0.450
	$PT \to CP \to FT$	**0.369**	**0.267**	**0.306**	**0.208**	**0.370**	**0.265**	**0.562**	**0.472**
Llama3.1 150M	PT	0.280	0.186	0.227	0.147	0.247	0.158	0.248	0.168
	$PT \to CP$	0.280	0.192	0.235	0.155	0.273	0.176	0.606	0.480
	$PT \to CP \to FT$	**0.307**	**0.214**	**0.266**	**0.178**	**0.334**	**0.239**	**0.649**	**0.525**

Table 4: AR results for FastDraft stages, performed in subsequent order: Pre-training (PT), Continued Pre-training (CP) and Fine-tuning (FT) using multinomial sampling with temperature $T = 0.6$

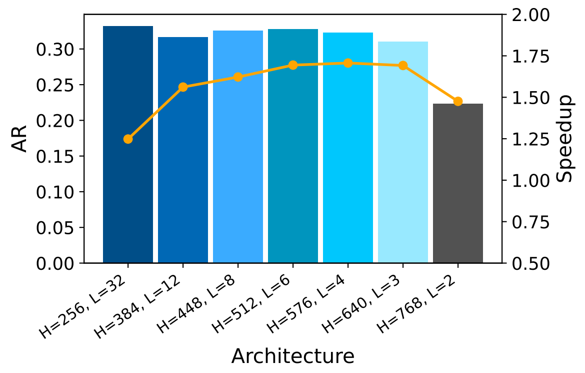

Figure 4: Estimated speedup on CNN-DM for models with constant parameter budget. L, H denote the number of layers and hidden dimension size

a draft for Phi-3-mini. To further illustrate reproducibility, we employ the same pipeline to produce a draft for Llama-3.1-8B-Instruct (Dubey et al., 2024). We present a 50M draft for the Phi-3-mini and a 150M draft for Llama-3.1-8B. Drafts' architecture details are presented in Appendix B.1. Considering both performance and resource efficiency, our findings from sections 5.2, 5.3 suggest pre-training over 5BT of FineWeb text data and continued pre-training on a mixture of 5BT code data from The Stack v2 and 2.5BT FineWeb text data as a best practice. For fine-tuning, we conclude from sections 5.4, 5.5 that sequence-level KD yields significant improvements while token-level KD benefits are not definitive, therefore, for FastDraft we only utilize sequence-level KD. We construct an alignment dataset for both Phi-3 and Llama-3.1 drafts by combining a number of synthetic datasets we generated with the appropriate target model as described in Section 5.1. Table 4 presents the AR improvements achieved through each stage of FastDraft generated with multinomial sampling. Results for greedy sampling are presented in Table 17 in the Appendix. Using both sampling methods, we observe for datasets CNN-DM, TS, and Dolly, a substantial AR increase following fine tuning, with instruction-following dataset Dolly exhibiting an increase of ~10% in AR. For HumanEval the primary performance gains stem from pre-training on code domain knowledge during continued pre-training. These models achieve a MBSU of ~2x for natural language tasks and ~3x for code completion tasks.

Furthermore, we demonstrate the real-world wall-clock time speedup achieved with FastDraft on the Intel® Core™ Ultra 7 Processor 268V as detailed in Section 5.6. We observe an average TPOT speedup of 1.5x for summarization tasks and 2x for code completion. Additionally, memory transaction efficiency improves, with up to 3x reduction in memory bandwidth usage, as indicated by the MBSU metric in these trials. Memory transactions can be highly energy-intensive (Horowitz, 2014), making the use of FastDraft drafts a promising approach for achieving substantial power savings on edge devices. These performance gains stem from the compact nature of our drafts, enabling them to operate in tandem with large target LLMs. Notably, a pre-trained FastDraft draft model can be used either as-is or aligned with any compatible model. For instance, our Llama-3.1 draft is readily compatible with over 1,000 models on the HuggingFace Hub[5] , highlighting FastDraft's scalability. We believe FastDraft has the potential to unlock LLMs inference on AI-PC and other edge-devices.

7 Conclusion & future work

In this paper, we introduced FastDraft, a novel approach for training and evaluating draft models for speculative decoding. Our results demonstrate that FastDraft facilitates the rapid training of high-quality, efficient draft models that are well-

[5] https://huggingface.co/

aligned with target models. We conducted a comprehensive ablation study to examine various aspects of draft training, including pre-training data composition and draft-target alignment. Using our method, we successfully trained a highly efficient 50M-parameter draft model for Phi-3-mini, achieving an acceptance rate of up to 67% and up to a 3x memory-bound speedup. We further demonstrated the scalability of FastDraft by training a draft for Llama-3.1-8B-Instruct, underscoring its effectiveness for larger models. We validated our theoretical findings by benchmarking our drafts on the latest AI-PC HW achieving a wall-clock time speedup of up to 2x. We hope our findings will inspire additional research on efficient draft training, particularly focusing on the development of resource-efficient draft architectures and hardware-aware designs.

8 Limitations

Our training recipe for draft models has been validated exclusively in English, and the reported acceptance rates and speedup benefits may not generalize to non-English languages due to their distinct syntactic structures and linguistic characteristics. Future work should investigate the effectiveness of our approach across different languages and develop language-specific modifications to optimize performance for non-English applications.

While prior work has demonstrated that speculative decoding with multiple candidate sequences can achieve higher acceptance rates and greater speedup compared to single-sequence approaches (Li et al., 2024; Cai et al., 2024), we restricted our implementation to single-sequence speculation due to computational constraints. Multiple-sequence speculation requires substantial parallel compute resources to generate and validate multiple candidate sequences simultaneously. Given our focus on AI-PC, we prioritized developing methods that could run effectively on consumer HW.

Our work primarily investigates optimal training strategies for draft models that share the same architectural design as their target models. While this focused scope allowed us to deeply explore the relationship between architecturally identical draft and target pairs, it also presents notable limitations. First, our findings may not generalize well to scenarios where draft and target models employ different architectures, as the dynamics of knowledge transfer and performance optimization could vary significantly in such cases. Additionally, while our proposed training recipe proved effective for architecturally identical pairs, it's possible that combining our approach with more efficient draft architectures could yield even better results—potentially offering improved speed-quality trade-offs that we have not explored in this work.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. 2024. To code, or not to code? exploring impact of code in pre-training. *arXiv preprint arXiv:2408.10914*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Jie Huang, and Kevin Chen-Chuan Chang. 2023. Cascade speculative drafting for even faster llm inference. *arXiv preprint arXiv:2312.11462*.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm.

Common Crawl. Common crawl. `https://commoncrawl.org/`.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ronen Eldan, Elad Liebman, Colin Cherry, and Yinhan Liu. 2023. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Raghavv Goel, Mukul Gagrani, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. 2024. Direct alignment of draft model for speculative decoding with chat-fine-tuned llms. *arXiv preprint arXiv:2403.00858*.

John L Hennessy and David A Patterson. 2017. *Computer architecture: a quantitative approach*. Morgan kaufmann.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1693–1701.

Geoffrey Hinton. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Mark Horowitz. 2014. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pages 10–14. IEEE.

Nguyen Huu, Suri Sameer, Ken Tsui, Shahules786, Together.xyz team, and Christoph Schuhmann. 2023. Oig-small-chip2. `https://huggingface.co/datasets/0-hero/OIG-small-chip2`.

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. *arXiv preprint arXiv:2001.08361*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2025. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. *arXiv e-prints*, arXiv:2503.01840.

Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, et al. 2023. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. *arXiv preprint arXiv:2305.13169*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

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. *arXiv preprint arXiv:2306.08568*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. 2023. Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification. *arXiv preprint arXiv:2305.09781*.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Xsum: A new dataset for abstractive summarization of news articles. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 701–711.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5.

Salesforce. Salesforce wikitext dataset. `https://huggingface.co/datasets/Salesforce/wikitext`.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31.

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Heming Xia, Yongqi Li, Jun Zhang, Cunxiao Du, and Wenjie Li. 2024. SWIFT: On-the-Fly Self-Speculative Decoding for LLM Inference Acceleration. *arXiv e-prints*, arXiv:2410.06916.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. 2024. Decoding speculative decoding. *arXiv preprint arXiv:2402.01528*.

Aonan Zhang, Chong Wang, Yi Wang, Xuanyu Zhang, and Yunfei Cheng. 2024. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. *arXiv e-prints*, arXiv:2309.08168.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2023. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*.

A Comparison to other methods

To ensure a fair comparison with speculative decoding methods such as MEDUSA (Cai et al., 2024), EAGLE (Li et al., 2024), and self-speculation techniques as presented in (Zhang et al., 2023; Xia et al., 2024), key factors must remain consistent, including hardware, target model, and speculation strategy. In contrast to the aforementioned methods, FastDraft adheres strictly to standard, sequential speculative decoding without employing multi-token windows, multiple drafts, or speculation trees—although such enhancements are complementary to our approach and could be incorporated to achieve additional speed improvements. Moreover, our primary objective is to offer a solution in scenarios where utilizing an existing draft model or a smaller language model (SLM) from the same model family is not feasible (e.g., Phi-3).

As reported in Goel et al., 2024, pretrained draft models have been empirically shown to exhibit significantly better alignment with the target model. Despite the associated cost, our approach enables efficient training from scratch in several key aspects: FastDraft supports architectural flexibility, enabling the construction of compact draft models. In contrast, drafts designed to leverage latent representations of the target model scale with the target's hidden dimension. For example, EAGLE drafts for 7B, 13B, and 70B models require 0.24B, 0.37B, and 0.99B parameters, respectively, while MEDUSA's 3–5 decoding heads correspond to 1.5–2.6B trainable parameters for Llama3.1-8B. In the case of Self-Speculation, the large budget of target parameters—approximately 500M for Llama-3.1-8B—used during speculation also limits its speedup and results in substantial memory consumption. The FastDraft pipeline can be completed within one day using eight accelarators, representing a substantial improvement over prior draft training methods (Goel et al., 2024; Miao et al., 2023). For comparison, EAGLE training requires up to 2 days on four GPUs. Furthermore, EAGLE-3 (Li et al., 2025) was trained on approximately eight times more data than EAGLE. Moreover, FastDraft drafts can be used with any compatible target model without additional training or adjustments, whereas fine-tuning can further improve alignment to the target model. Self-Speculation, by contrast, depends on Bayesian optimization—a computationally intensive process that can take several hours per iteration.

B Pre-training details and results

B.1 Draft model architecture

We work with drafts built on the architecture of "Phi-3-mini-4k-instruct" and "Llama-3.1-8B-Instruct". We use float16 precision for model pre-training. Table 5 below provides a detailed view of the structures of the drafts.

B.2 Details of pre-training code dataset

In Table 6, we summarize the data composition of the code dataset we employ for pre-training.

B.3 Hyper-parameter configuration for datasets size in pre-training experiments

We train all draft variants, for both pre-training and continued pre-training (CP) in float16 precision for 1 epoch with a batch size of 128. We use Adam with learning rate of 1×10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over 5% of the total training steps, and linear decay of the learning rate.

B.4 Pre-training model size ablation

We provide the full results of Table 1, pre-training on varying model sizes of 50 and 120 million parameters in Table 7. For a detailed view of the drafts' structure, see Table 5

B.5 Pre-training on code ablation

Pre-training on a mixed dataset Consistent with current best practices for pre-training large language models (Dubey et al., 2024; Abdin et al., 2024), we experiment with pre-training the draft model on a mixture of code and natural language datasets with varying ratios of the domains. Similar to (Aryabumi et al., 2024), we evaluate a balanced mixture of 5 billion token code and 5 billion token text datasets. Since our datasets are relatively small to begin with, emphasizing on text data can be beneficial for general language understanding across tasks. Accordingly, we also experiment with mixed datasets featuring 4 billion tokens of text and 6 billion tokens of code, as well as 7 billion tokens of text and 3 billion tokens of code.

Our experimental results suggest the following conclusions:

Continued pre-training is better than pre-training on a mixed dataset: Continued pre-training variants with CP mixed dataset of code and text were able to obtain 1-4% higher acceptance rate on CNN-DM, TS and Dolly datasets, over

Draft name	Phi3-mini 50M	Phi3-mini 120M	Llama3.1 150M
Hidden size	512	768	512
Intermediate size	1408	2048	1792
# Layers	6	12	6
#Attention heads	8	12	8
# Key-value heads	8	12	8
Vocabulary size	32064	32064	128256

Table 5: Phi-3-mini and Llama-3.1-8B-Instruct drafts configurations

greedy and multinomial sampling methods (see Table 9, 11) compared to CP datasets of only code or only text. These models have been trained on 1-2.5 billion more tokens compared to pre-trained models on mixed data, but as section 5.2 suggests, the performance gain likely stems from the choice of pre-training strategy rather than additional data. Surprisingly, mixed datasets with higher proportion of text yield better results on HumanEval, a code evaluation dataset, with little to no improvement on natural language tasks 12, 13. This strengths our findings in section 5.2 that additional text data doesn't contribute to our draft performance beyond a certain point.

Text Base dataset is better than code Base dataset for continued pre-training Variants of continued pre-training settings with Base dataset of Fineweb 5 billion token sample achieve higher acceptance rate on HumanEval (Table 8) compared to their code Base dataset counterparts of TheStack-v2 5 billion token sample. This is likely because these models acquire the majority of their code domain knowledge during the initial pre-training stage, and the later introduction of text domain knowledge can cause gradient shifts that are suboptimal for code. While including portions of code in the CP dataset helps mitigate this issue, the performance degrades significantly by up to 50% when the CP dataset lacks code.

C Hardware-aware draft details and results

Tables 14 and 15 contain the full list of configuration used in HW-aware experiments. Table 16 lists the configurations used for estimating the speedup of architectures with varying depth or width satisfying a constant parameter budget.

Figures 5 and 6 present the results for the constant budget pre-trained drafts evaluated on the TS and Dolly datasets.

D FastDraft additional results

Table 4 summarizes results of each stage of the FastDraft scheme using greedy sampling.

Language	Token count (B)	Sample Count	Avg. sample length (tokens)
Python	1.18	963923	1227
C	0.83	309496	2679
C#	0.78	783399	994
C++	1.35	676943	1994
Go	0.18	140831	1285
Java	1.02	1003702	1015
JavaScript	1.27	1109269	1145
Kotlin	0.09	142275	601
Lua	0.1	32335	3212
PHP	1.07	731103	1462
R	0.13	79108	1704
Ruby	0.13	285785	469
Rust	0.07	36811	1926
SQL	0.26	70937	3633
Shell	0.11	175985	613
Swift	0.1	118558	877
TypeScript	0.26	386644	674
Documentation languages	0.96	807360	1194
Configuration languages	0.09	140370	652
Configuration files	0.02	43507	554
Total	10.0	8038341	

Table 6: Overview of the data composition of the 10BT sample of the-stack-v2-train-smol

Draft size	Data size	Perplexity	CNN-DM		TS		Dolly	
			Greedy	Multinomial	Greedy	Multinomial	Greedy	Multinomial
50M	0.1BT	1594.5	0.168	0.152	0.158	0.143	0.173	0.151
	0.5BT	476.3	0.304	0.280	0.245	0.226	0.253	0.233
	1BT	379.0	0.315	0.295	0.267	0.247	0.255	0.236
	2BT	297.4	0.347	**0.323**	0.267	0.247	0.255	0.236
	5BT	256.6	**0.350**	0.311	0.295	0.277	**0.258**	**0.245**
	10BT	**240.9**	0.325	0.312	**0.300**	**0.283**	0.242	0.234
120M	0.1BT	1241.0	0.194	0.180	0.183	0.171	0.199	0.173
	0.5BT	343.2	0.322	0.302	0.279	0.260	0.286	0.266
	1BT	253.8	0.376	0.348	0.302	0.272	**0.314**	**0.294**
	2BT	199.6	0.391	0.362	0.321	0.297	0.295	0.284
	5BT	167.7	**0.393**	**0.366**	0.343	0.327	0.293	0.281
	10BT	**147.4**	0.381	0.351	**0.357**	**0.331**	0.272	0.251

Table 7: Pre-training data size effect on acceptance rate of natural language tasks. Results for block size $\gamma = 3$ on both greedy and multinomial sampling temperature $T = 0.6$

Base dataset	CP mix dataset code	CP mix dataset natural lang.	HumanEval $\gamma = 3$	HumanEval $\gamma = 5$
Fineweb 5BT sample	5B	-	**0.688**	**0.578**
	5B	1B	0.678	0.560
	5B	2.5B	0.672	0.561
The stack-v2 5BT sample	-	5B	0.320	0.238
	1B	5B	0.628	0.519
	2.5B	5B	0.667	0.553

Table 8: Continued pre-training effect on acceptance rate of code using greedy decoding

CP Base dataset	CP dataset code	CP dataset natural lang.	CNN-DM $\gamma = 3$	CNN-DM $\gamma = 5$	TS $\gamma = 3$	TS $\gamma = 5$	Dolly $\gamma = 3$	Dolly $\gamma = 5$
Fineweb 5BT sample	5B	-	0.314	0.216	0.249	0.164	0.210	0.139
	5B	1B	0.343	0.241	0.297	0.200	**0.245**	**0.163**
	5B	2.5B	**0.349**	**0.244**	**0.304**	**0.204**	0.244	**0.163**
The stack-v2 5BT sample	-	5B	0.340	0.238	0.276	0.182	0.220	0.144
	1B	5B	0.339	0.239	0.283	0.190	0.222	0.148
	2.5B	5B	0.344	0.243	0.285	0.191	0.227	0.151

Table 9: Continued pre-training effect on acceptance rate of natural language tasks using greedy decoding

Base dataset	CP mix dataset code	CP mix dataset natural lang.	HumanEval $\gamma = 3$	HumanEval $\gamma = 5$
Fineweb 5BT sample	5B	-	**0.578**	**0.462**
	5B	1B	0.560	0.451
	5B	2.5B	0.561	0.450
The stack-v2 5BT sample	-	5B	0.238	0.161
	1B	5B	0.519	0.400
	2.5B	5B	0.553	0.416

Table 10: Continued pre-training effect on acceptance rate of code using multinomial sampling with temperature $T = 0.6$

CP Base dataset	CP mix dataset code	CP mix dataset natural lang.	CNN-DM $\gamma = 3$	CNN-DM $\gamma = 5$	TS $\gamma = 3$	TS $\gamma = 5$	Dolly $\gamma = 3$	Dolly $\gamma = 5$
Fineweb 5BT sample	5B	-	0.271	0.170	0.232	0.153	0.191	0.126
	5B	1B	0.296	0.210	0.279	0.187	**0.233**	0.148
	5B	2.5B	0.304	0.211	**0.287**	**0.192**	0.226	**0.149**
The stack-v2 5BT sample	-	5B	0.297	0.205	0.254	0.173	0.201	0.135
	1B	5B	0.307	0.201	0.268	0.172	0.205	0.141
	2.5B	5B	**0.312**	**0.213**	0.264	0.176	0.215	0.143

Table 11: Continued pre-training effect on acceptance rate of natural language tasks using multinomial sampling with temperature $T = 0.6$

Mix dataset	Mix dataset	CNN-DM		TS		Dolly		HumanEval	
natural lang.	code	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$
5B	5B	0.354	0.249	0.284	0.189	0.205	0.136	0.639	0.532
6B	4B	0.355	0.248	0.284	0.190	**0.220**	**0.144**	0.656	**0.551**
7B	3B	0.354	0.248	0.285	0.191	0.216	**0.144**	**0.657**	0.550

Table 12: Impact of pre-training on mixed datasets on acceptance rate of natural language and code tasks. Evaluated across window sizes 3 and 5 using greedy decoding

Mix data	Mix data	CNN-DM		TS		Dolly		HumanEval	
natural lang.	code	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$
5B	5B	0.311	0.206	**0.265**	0.175	0.194	0.124	0.506	0.389
6B	4B	0.317	**0.218**	0.261	**0.178**	0.201	0.137	0.530	0.411
7B	3B	**0.326**	0.211	0.258	0.176	**0.205**	**0.137**	**0.533**	**0.424**

Table 13: Impact of pre-training on mixed datasets on acceptance rate of natural language and code tasks. Evaluated across window sizes 3 and 5 using multinomial sampling with temperature $T = 0.6$

# Params (M)	Hidden size	Intermediate size	# Layers	#Attention heads	# Key-value heads
27.91	512	1408	4	8	8
38.95	640	1792	4	10	10
37.10	512	1408	7	8	8
42.67	768	2048	4	12	12
43.23	512	1408	9	8	8
80.32	1024	2816	4	16	16
80.00	512	1408	21	8	8
114.78	1280	3456	4	20	20
113.70	512	1408	32	8	8
34.04	512	1408	6	8	8
48.64	640	1792	6	10	10
49.36	512	1408	11	8	8
64.00	768	2048	6	12	12
64.68	512	1408	16	8	8
104.83	1024	2816	6	16	16
104.51	512	1408	29	8	8
152.60	1280	3456	6	20	20
153.53	512	1408	45	8	8

Table 14: Phi-3-mini draft configurations for latency benchmarks. #Params values exclude the embeddings table parameters

Name	Hidden size	Intermediate size	# Layers	#Attention heads	# Key-value heads
Phi3-draft-256	256	768	32	4	4
Phi3-draft-384	384	1024	12	6	6
Phi3-draft-448	448	1280	8	7	7
Phi3-draft-576	576	1536	4	9	9
Phi3-draft-640	640	1792	3	10	10
Phi3-draft-768	768	2048	2	12	12
Phi3-draft-base	**512**	**1408**	**6**	**8**	**8**

Table 15: Configurations of models sharing the parameter budget with the Phi3-mini 50M draft configuration

Model	CNN-DM		TS		Dolly	
	AR	Speedup	AR	Speedup	AR	Speedup
Phi3-mini-256	0.3319	1.2481	0.3182	1.2223	0.2681	1.1283
Phi3-mini-384	0.3170	1.5616	0.3071	1.5379	0.2097	1.3039
Phi3-mini-448	0.3255	1.6222	0.3009	1.5616	0.2177	1.3568
Phi3-mini-512	0.3277	1.6941	0.3021	1.6284	0.2351	1.4568
Phi3-mini-576	0.3230	1.7066	0.2905	1.6220	0.2222	1.4444
Phi3-mini-640	0.3104	1.6918	0.2836	1.6212	0.2251	1.4676
Phi3-mini-768	0.2236	1.4756	0.2713	1.6020	0.2006	1.4149

Table 16: AR and Estimated speedup results for models obtained by modifying depth and width to stay within the same parameter budget as Phi3-mini-base draft model

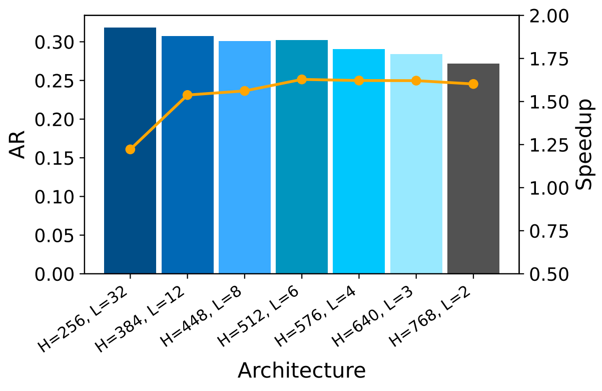

Figure 5: Estimated speedup on TS for models with constant parameter budget. L, H denote the number of layers and hidden dimension size

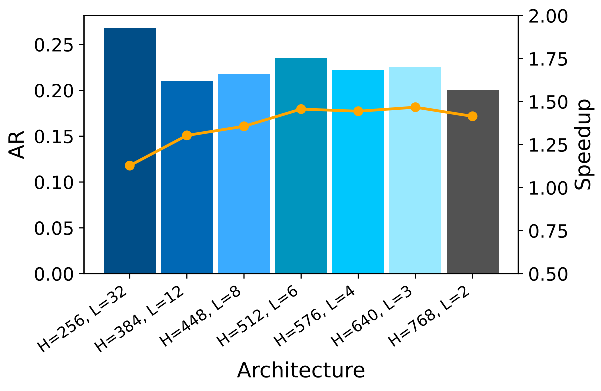

Figure 6: Estimated speedup on Dolly for models with constant parameter budget. L, H denote the number of layers and hidden dimension size

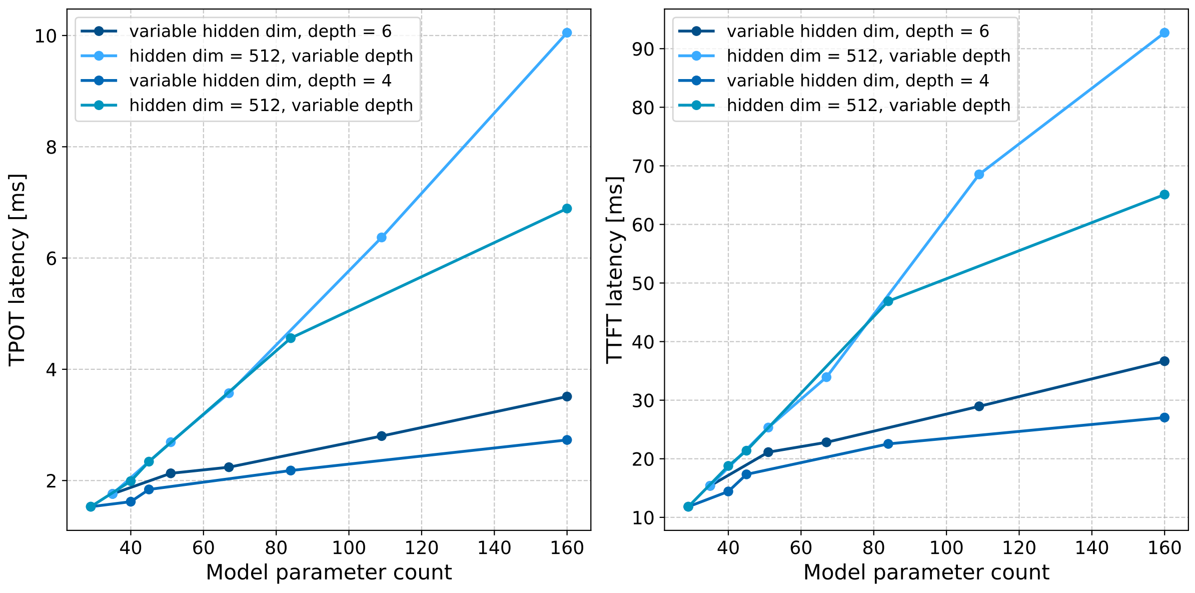

Figure 7: First and second token latencies of models obtained by varying either a hidden dimension or a number of layers

Model	Type	CNN-DM		TS		Dolly		HumanEval	
		$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$	$\gamma = 3$	$\gamma = 5$
Phi3-mini 50M	PT	0.350	0.246	0.295	0.196	0.258	0.174	0.312	0.221
	$PT \rightarrow CP$	0.349	0.244	0.304	0.204	0.244	0.163	**0.672**	**0.563**
	$PT \rightarrow CP \rightarrow FT$	**0.399**	**0.289**	**0.321**	**0.217**	**0.390**	**0.279**	0.663	0.553
Llama3.1 150M	PT	0.298	0.204	0.243	0.162	0.257	0.171	0.282	0.198
	$PT \rightarrow CP$	0.300	0.203	0.250	0.166	0.284	0.193	0.658	0.546
	$PT \rightarrow CP \rightarrow FT$	**0.327**	**0.228**	**0.271**	**0.181**	**0.350**	**0.247**	**0.700**	**0.593**

Table 17: AR results for FastDraft stages, performed in subsequent order: Pre-training (PT), Continued Pre-training (CP) and Fine-tuning (FT) using greedy decoding