SpecDec

Avi Arya & Shreya Singh & Theo Urban
10-423/623 Generative AI Course Project

November 24, 2025

1 Introduction

As large language models have continued to grow, inference latency has become a bigger issue. Improvements in inference latency can significantly affect workflows, and having lower latency may result in faster systems and feedback loops. A recent proposal to address the issue of inference latency is Speculative Decoding, a method of using smaller models to generate a response, then using a larger language model to attempt to either accept or disregard the smaller language model's result. If the larger language model does not accept the smaller language model's result, then the larger language model generates its own result fully. However, the hope is that, oftentimes, the smaller language model and the larger language model will result, resulting in improved latency.

In this project, we hope to apply finetuning to Speculative Decoding to create a faster model for code generation. Specifically, we will use The Stack, a dataset of code files, to finetune minGPT, then use this as the small model in our speculative decoding model, eventually comparing this model to speculative decoding without this additional finetuning.

2 Dataset / Task

For this project, we will use The Stack, a large dataset of source code developed by the BigCode project. The dataset is one of the most extensive collections of code, containing 6.4 terabytes spanning 358 programming languages.

Our use of this dataset will be to finetune our draft model, minGPT 2. We intend to create a training split with code from higher resource languages (Python, C++, Javascript, etc) and perhaps some lower resource languages as well. The lower resource languages can be use to evaluate the finetuning process for its impact on cross lingual performance disparities.

The test set will comprise data from The Stack not used during finetuning. The set will have several code completion prompts where we will use a function signature, comment block, or partial line of code as input. Using this will allow us to compare the base model and finetuned model fairly on their code generation ability.

The primary task at hand is domain-specific accelerated code completion. In other words, given an input context s (a string of source code), we want to generate a plausible and syntatically correct continuation of the code using a speculative decoding framework to potentially accelerate the rate of output.

We define our two primary models as follows:

1. **Verifier Model** (P_θ) represents the probability distribution $p(x \mid s)$ over the next token x given some context s. For this project, we will use the standard GPT 2 model as P_θ.

2. **Draft Model** (Q_ϕ) represents the probability distribution $q(x \mid s)$. For this project, we will use minGPT 2 which is smaller and faster than the verifier model.

Our project will compare two distinct draft models:

1. **Baseline Draft Model** (Q_ϕ) which is the unmodified minGPT 2 model.

2. **Finetuned Draft Model** (Q'_ϕ) which is the minGPT 2 model after it has been finetuned on the training split from The Stack.

The task involves providing the system with a context string s and sampling a fixed length sequence of tokens from the draft model. These tokens are then validated by the verifier in a single pass.

To quantitatively evaluate our model's success, we will look at its acceptance rate, speedup over the baseline model, and crosslingual disparity. These metrics are detailed more in a later section.

3 Related Work

Reducing inference times in LLMs is a significant area of past and current research. Modern LLMs have high associated computation and memory bandwidth costs due to standard decoding, which in turn provide a bottleneck for many real world applications. Our project aims to build on the core technique of speculative decoding, first presented in **?**. The paper demonstrated that using a smaller draft model to generate tokens and then passing them to a larger verifier model could provide speedups in generations without sacrificing generation quality. Then **?** introduced a similar method called speculative sampling that further solidified the technique's potential at reducing overhead. Our baseline model will be a direct implementation of the algorithms presented in these two papers.

The reason for our finetuned model, however, is that the benefits of speculative decoding are not uniformly distributed. This issue is raised and explored by **?**, who point out performance disparities in speculative decoding. They show that it accelerates generation primarily for high resource languages and predictable text sequences. To this extent, we aim to evaluate our model on crosslingual performance as well. We will adopt the disparity metric $U(T)$ that they introduced to measure our approach's success.

Our central hypothesis is that specializing the draft model to a particular task can improve its performance, which is well supported in recent literature. An example is using knowledge distilliation, which is done by **?** via DistillSpec. Here, the draft model is explicitly trained on the verifier's output distribution. However, our approach diverges slightly to finetune on a specialized corpus which can be seen as domain-specific distillation.

Finally, another consideration that we have to make is alignment. As shown by **?**, it is imperative that the verifier and draft model are aligned, especially when the larger model is finetuned in some way (i.e. for chat) because a generic draft model will perform quite poorly in such cases. The solution they use is to finetune the draft model using the verifier model as a "teacher". While our project is not specifically aligning the draft model with the verifier, it does aim to align the draft model to the domain we are testing on.

4 Approach

In this project, our overall goal is to determine the impact of finetuning on speculative decoding models. Our baseline model is a speculative decoding model that uses minGPT 2 as the smaller model and GPT 2 as our larger, verifier model. We first must implement speculative decoding using these two models, and run experiments to determine the speed at which this generates its output on a few sample prompts. Next, we will finetune minGPT 2 on The Stack, a large dataset from Hugging Face containing code files in 300 different languages. By using this dataset, we hope to better inform our smaller model on this particular task. We will then reimplement speculative decoding using this finetuned model alongside GPT2, and will evaluate the inference latency and results on our given prompts.

5 Experiments

5.1 Experimental Setup

We will conduct all of our experiences on an A100 obtained through Google Colab Pro. We use 12B Pythia by EletherAI as our Verifier model and 70M Pythia as our Draft model, following from the initial experiments below. All experiments are conducted with a fixed seed of 42 to ensure consistency across runs.

For our main set of experiemnts, we fix $\gamma =$??? tokens to standardize across runs with a reasonable balance between draft and verification speed.

5.2 Baseline Performance

Our first experiment establishes the baseline performance of a few different models and on different tasks - natural language completions and a few coding tasks in different languages outlined in the appendix.

Model	Completion	Acce. Rate	Speedup
GPT2-Large, distilgpt2	NL	???	???
	Code (Py)	???	
	Code (C)	???	
	Code (go)	???	
	Code (rust)	???	
Qwen2.5 (7B, 0.5B)	NL	???	???
	Code (Py)	???	
	Code (C)	???	
	Code (go)	???	
	Code (rust)	???	
Pythia (12B, 70M)	NL	???	???
	Code (Py)	???	
	Code (C)	???	
	Code (go)	???	
	Code (rust)	???	

Table 1: Caption

5.3 Expected Outcomes

We want to answer a few fundamental research questions:

1. How does domain-specific finetuning of a draft model on code affect the Acceptance rate and speedup of a Speculative Decoding setup in and out of domain.

2. How does finetuning a draft model on a code impact performance on low-resource sets (in our case, languages), within the domain.

3. How does dynamically varying draft length (γ) impact speedup under a fixed compute environment and under different domains

6 Proposed Experiments

In this section, we outline the experiments designed to evaluate how domain-specific finetuning of the draft model influences speculative decoding effectiveness.

6.1 RQ1: Does Domain-Specific Finetuning Improve Speculative Decoding Performance?

We compare two systems:

1. **Baseline SpecDec**: GPT2 (verifier) + unfinetuned minGPT2 (draft)

2. **Finetuned SpecDec**: GPT2 (verifier) + minGPT2 finetuned on The Stack

Both models will be evaluated on:

- **In-domain**: Code completions across Python, C++, Go, Rust, and others

- **Out-of-domain**: Natural language tasks (news, stories, instructions)

Each prompt set will generate 128-token continuations with fixed draft length γ.

Model Pair	Domain	Accept Rate	Speedup
Baseline	Code (Py)	???	???
Finetuned	Code (Py)	???	???
Baseline	NL	???	???
Finetuned	NL	???	???

Table 2: Effect of draft model finetuning on acceptance rate and decoding speed.

Expected Outcome. Finetuning should increase acceptance rate and speedup on code tasks, while maintaining comparable performance on natural language prompts.

6.2 RQ2: Does Finetuning Improve Performance for Low-Resource Languages?

Goal. Determine whether finetuning disproportionately helps high-resource languages or narrows cross-lingual disparities.

Experimental Setup. We construct a language-stratified test set from The Stack with three buckets:**High-resource**: Python, C++, Java, **Medium**: Lua, Haskell, Ruby, and **Low-resource**: Zig, Elixir, OCaml

For each language bucket, we measure (1) acceptance rate, (2) end-to-end speedup, and (3) disparity metric $U(T)$ from **?** before and after finetuning.

Group	Base $U(T)$	Tuned $U(T)$	Δ Disparity
High	???	???	???
Medium	???	???	???
Low	???	???	???

Table 3: Change in cross-lingual performance disparity after finetuning.

Expected Outcome. Finetuning should reduce performance disparity across languages by improving draft alignment with code structure.

6.3 RQ3: How Does Draft Length γ Affect Speedup Under Fixed Compute?

Goal. Characterize how varying γ impacts acceptance rate, rejection bursts, and overall speedup.

Experimental Setup. Using the finetuned draft model, we vary $\gamma \in \{1, 2, 4, 8, 16, 32\}$ For each γ, we record (1) Acceptance rate, (2) rejection burst statistics, and (3) real-time decoding speedup.

γ	Avg. Burst Len.	$P(\text{Burst} > 10)$	Speedup	Accept Rate
1	???	???	???	???
4	???	???	???	???
16	???	???	???	???

Table 4: Effect of draft length on rejection patterns and speedup.

Expected Outcome. We hypothesize a unimodal trend where moderate values of γ maximize speedup without causing excessive rejection cascades.

6.4 Summary

These experiments collectively evaluate:

- Finetuning effects on speculative decoding efficiency (RQ1)

- Cross-lingual disparity and fairness implications (RQ2)

- Sensitivity of speculative decoding to draft length (RQ3)

Together, they form a comprehensive framework for understanding speculative decoding in domain-specialized settings.

7 Compute

8 Plan

We can break this project in to three sequential steps:

1. Set up speculative decoding using minGPT and GPT2 as paired models
2. Create metrics
3. Finetune minGPT and GPT2 on The Stack
4. Evaluate metrics pre and post finetune

We intend to complete through the metrics creation and initial testing by the executive summary deadline. The work is highly sequential so it will involve a lot of pair programming and back-and-forth. However, we will assign directly responsible members for items 1, 2, and 3, to Avi, Shreya, and Theo in that order.