CS7.501 : Advanced Natural Language Processing
Project Outline

Nevil Sakhreliya 2023201005
Prakhar Jain 2022121008
Aditya Gupta 2023201009

November 20, 2024

1 Introduction

Problem: Scaling transformer models to 500B+ parameters has led to significant performance improvements in natural language processing (NLP), computer vision, and reinforcement learning tasks. However, transformer decoding in this regime remains costly and inefficient.

Transformer sampling is typically memory bandwidth-bound [1], meaning that for a given hardware configuration, the time to generate a single token is approximately proportional to the size of the model parameters and transformer memory. The large size of these models also necessitates model parallelism, introducing communication overheads [2] and increasing resource requirements. Since generating each new token depends on previous tokens, many transformer calls are required to sample a complete sequence.

2 Autoregressive Sampling

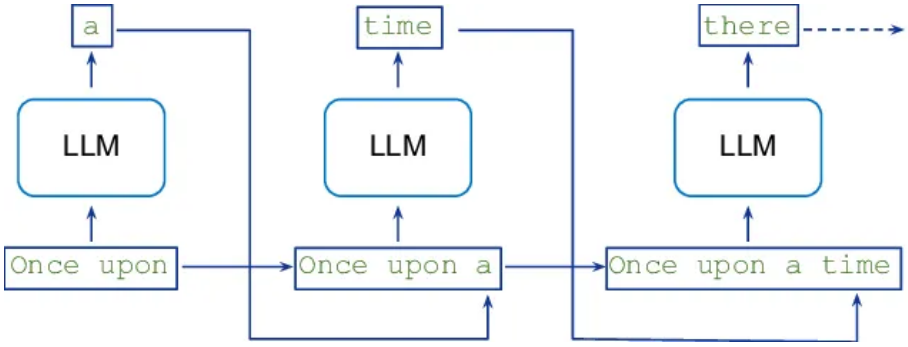

Figure 1: Autoregressive Sampling

The standard method for generating text from language models is **autoregressive sampling**. In this approach, the model generates text one token at a time by conditioning each new token on the previous ones. More formally, given a sequence of tokens, $x_1, x_2, \ldots, x_{t-1}$, the model predicts the next token x_t based on the context of the previous tokens. This can be expressed as:

$$P(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

Autoregressive sampling works by iteratively generating a sequence of tokens until a stopping criterion is met (such as a predefined length or the appearance of a special end-of-sequence token). The process proceeds as follows:

1. The model begins with an initial input, often referred to as a *prompt* or *seed text*.

2. The model generates the first token based on this initial prompt.

3. The generated token is then added to the input context, and the model generates the next token conditioned on the updated context.

4. This process is repeated for K steps, where K is the number of tokens to be generated.

For example, if the model is given the prompt "The cat," it will predict the next word based on the likelihood of various possibilities, such as "sat," "jumped," or "slept." Once a word is chosen, the model updates its context to include the word, and the next prediction is made. This serial process continues for each subsequent word, resulting in a coherent sequence of text.

While autoregressive sampling is effective at generating fluent and contextually relevant text, it is computationally expensive, especially for large models with many parameters. Each token requires a separate forward pass through the model, and generating K tokens requires K individual evaluations, each conditioned on the previous context. This makes the process slower as the model generates longer sequences, as depicted in Figure 1.

For large-scale models, the time complexity of autoregressive sampling becomes significant. As the model size grows, the computational cost per token increases, making it a bottleneck for real-time applications or when generating long passages of text. Researchers have explored various optimizations, such as parallel decoding techniques or faster approximate sampling methods, to mitigate these limitations.

In summary, autoregressive sampling is a foundational technique in natural language generation, where each token is predicted based on the preceding context. While it ensures high-quality text generation, it also comes with notable computational challenges, particularly when dealing with large models and long sequences.

3 Speculative Decoding

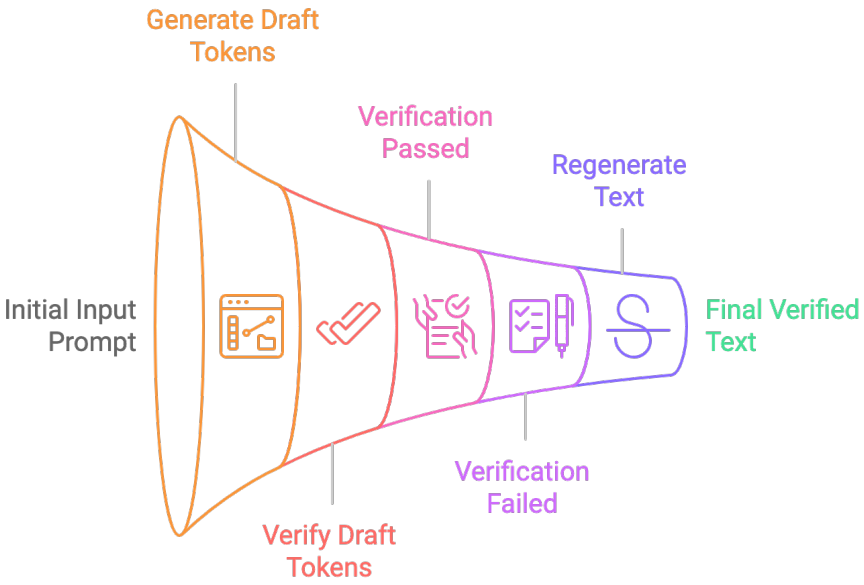

Figure 2: Verificatioin

Speculative decoding is a novel technique designed to accelerate large language models (LLMs) by generating responses faster without compromising quality. It incorporates a smaller, faster "draft" model to produce preliminary predictions. The main, more powerful model evaluates these predictions in parallel, reducing computational overhead.

3.1 Key Ideas

1. **Utilizing a Smaller Model for Easy Tokens:** Predicting simple tokens like `of` can be done by a smaller model, while complex tokens such as `Edinburgh` may require the larger model, refer Figure 3.

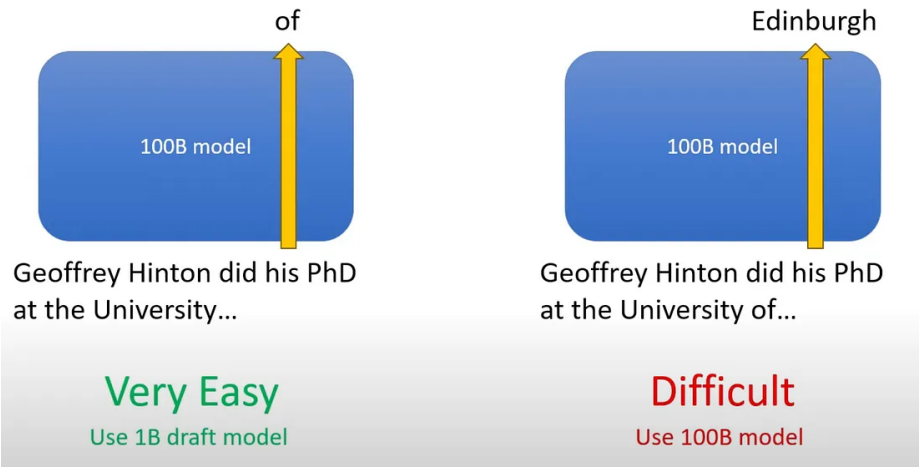

Figure 3: Predicting a token `of` is really easy and can easily predicted by much smaller model while prediction of token `Edinburg` is difficult comparatively which smaller model might not able to predict

2. **Parallel Evaluation:** Although transformers generate one token at a time, they can process multiple tokens simultaneously. The larger model evaluates the draft model's predictions and adjusts them based on probabilities, refer Figure 4.

Figure 4: The smaller model predicts `Toronto` but the correct word is `Edinburgh` the bigger model can see that the probability of `Toronto` is low and correct it to `Edinburgh`.

3.2 Pipeline

Let M_p represent the target model, M_q the draft model. . The speculative decoding pipeline involves three steps:

1. **Draft Generation:** Use the efficient small model M_q to generate γ tokens.
2. **Parallel Verification:** The target model M_p evaluates all guesses from M_q in parallel, accepting those leading to an identical distribution.
3. **Final Output:** Generate additional tokens from an adjusted distribution to fix rejected ones or add more if all tokens are accepted. Each parallel run of M_p produces at least one new token, potentially many, depending on how well M_q approximates M_p.

Mathematically, to sample $x \sim p(x)$, we instead sample $x \sim q(x)$ and keep it if $q(x) \leq p(x)$. If $q(x) > p(x)$, the sample is rejected with probability $1 - \frac{p(x)}{q(x)}$, and x is resampled from the adjusted distribution:

$$p_0(x) = \text{norm}(\max(0, p(x) - q(x))).$$

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

4 Baseline Models

The table provides a comparison of different GPT-2 models based on the average time they take to process each token. These models differ in terms of their size, which influences their performance in both speed and computational demand. Below is a more detailed description of each model:

- **GPT-2 Large (812M parameters)**:
  - **Average Time per Token: 0.0327 seconds**
  - This is the largest version of the GPT-2 model with 812 million parameters. Due to its large size, it requires more computational resources and takes longer to process each token. The larger model can produce higher-quality outputs, but its slower processing time makes it less efficient compared to smaller variants.
- **GPT-2 Medium (380M parameters)**:
  - **Average Time per Token: 0.018791 seconds**
  - With 380 million parameters, this model is smaller than GPT-2 Large, but still relatively large compared to other models. While it offers faster processing than the Large version, it sacrifices some quality in the text generation compared to its larger counterpart.
- **GPT-2 (137M parameters)**:
  - **Average Time per Token: 0.0092 seconds**
  - The standard GPT-2 model with 137 million parameters is the base version of GPT-2. Despite being the smallest among the larger models, it provides a good balance between performance and efficiency. Its faster processing time makes it suitable for tasks requiring speed but at the cost of slightly reduced text quality.
- **DistilGPT-2 (88.2M parameters)**:
  - **Average Time per Token: 0.004570 seconds**
  - DistilGPT-2 is a distilled version of GPT-2 with 88.2 million parameters. Distillation is a process that reduces the size of a model while retaining much of its performance. As a result, DistilGPT-2 is the fastest model on the list, making it an excellent choice for applications where speed is a priority, even if it might not generate the same high-quality output as the larger models.

Increasing the size of the model generally leads to slower processing times, but also typically improves the model's output quality. Smaller models like DistilGPT-2 are much faster but may not achieve the same level of sophistication in text generation.

5 Dataset

The experiments use the **CNN/DailyMail (CNNDM)** dataset, a widely recognized benchmark in natural language processing (NLP) for text summarization tasks. It consists of:

- **287,000 training examples**: Each example pairs a news article with its corresponding human-written summary. This large dataset enables the model to learn the complex task of summarizing articles.
- **13,000 validation examples**: This set is used during training to assess the model's performance on unseen data and adjust hyperparameters to avoid overfitting.
- **11,000 test examples**: After training, the model is evaluated on this separate set to provide an unbiased measure of its real-world performance.

The CNNDM dataset is particularly suited for evaluating summarization models, offering a robust foundation for training and testing their ability to generate concise and accurate summaries from lengthy news articles.

6 Metrics

The *acceptance rate* $\beta_{x_{<t}}$, given a prefix $x_{<t}$, is the probability of accepting $x_t \sim q(x_t|x_{<t})$ by speculative sampling. $E(\beta)$ is a measure of how well M_q approximates M_p. Assuming β values are i.i.d., and letting $\alpha = E(\beta)$, the expected number of tokens generated by the speculative decoding algorithm satisfies Equation (1).

$$E(\#\ generated\ tokens) = \frac{1-\alpha^{\gamma+1}}{1-\alpha} \tag{1}$$

Let c, the *cost coefficient*, be the ratio between the time for a single run of M_q and the time for a single run of M_p.

Theorem 6.1. *The expected improvement factor in total walltime by Algorithm 1 is* $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$.

7 Experiments

Performance metrics for `distilgpt2` and `gpt2-large`

Table 1 summarizes the performance metrics for `distilgpt2` (draft) and `gpt2-large` (target) as the scaling factor γ varies. The key observations are as follows:

1. α **(Efficiency):** The efficiency α decreases as γ increases, starting at 0.48 for $\gamma = 2$ and reducing to 0.18 for $\gamma = 7$. This trend indicates diminishing returns in efficiency with larger scaling factors.

2. **Empirical Time Per Token:** The empirical time per token slightly increases with γ, from 0.0272 at $\gamma = 2$ to 0.0300 at $\gamma = 7$. This suggests a minor degradation in processing speed with increasing scaling factors.

3. **Empirical Speed:** Empirical speed decreases from 1.202 at $\gamma = 2$ to 1.090 at $\gamma = 7$. This reflects a gradual decline in performance as the scaling factor grows.

4. **Theoretical Speed:** Theoretical speed exhibits a more pronounced reduction compared to empirical speed, dropping from 1.338 at $\gamma = 2$ to 0.618 at $\gamma = 7$. This suggests that the model's theoretical assumptions become less aligned with practical performance at higher scaling factors.

γ	α	**Empirical Time Per Token**	**Empirical Speed**	**Theoretical Speed**
2	0.48	0.0272	1.202	1.338
3	0.36	0.0277	1.180	1.084
4	0.30	0.0274	1.193	0.998
5	0.23	0.0291	1.123	0.766
7	0.18	0.0300	1.090	0.618

Table 1: Performance metrics for `distilgpt2` and `gpt2-large`.

Performance Metrics for `gpt2` and `gpt2-medium`

Table 2 summarizes the performance metrics for `gpt2` (draft) and `gpt2-medium` (target) as the scaling factor γ varies. The key observations are as follows:

γ	α	**Empirical Time Per Token**	**Empirical Speed**	**Theoretical Speed**
2	0.56	0.0192	0.958333	0.946
3	0.47	0.0203	0.906404	0.727
4	0.41	0.0216	0.851852	0.566
5	0.34	0.0239	0.769874	0.438
7	0.27	0.0289	0.636678	0.309

Table 2: Performance metrics for `gpt2` and `gpt2-medium`.

1. α **(Efficiency):** The efficiency α decreases as γ increases, starting at 0.56 for $\gamma = 2$ and reducing to 0.27 for $\gamma = 7$. This indicates diminishing efficiency as scaling increases.

2. **Empirical Time Per Token:** The empirical time per token increases with γ, from 0.0192 at $\gamma = 2$ to 0.0289 at $\gamma = 7$, showing a slight slowdown in processing time with higher scaling factors.

3. **Empirical Speed:** Empirical speed decreases as γ grows, from 0.958333 at $\gamma = 2$ to 0.636678 at $\gamma = 7$, highlighting reduced processing efficiency at larger scaling factors.

4. **Theoretical Speed:** Theoretical speed declines significantly, dropping from 0.946 at $\gamma = 2$ to 0.309 at $\gamma = 7$. This suggests that the theoretical model becomes less predictive of practical performance as the scaling factor increases.

Performance metrics for `distilgpt2` and `gpt2`

Table 3 summarizes the performance metrics for `distilgpt2` (draft) and `gpt2` (target) as the scaling factor γ varies. The key observations are as follows:

γ	α	**Empirical Time Per Token**	**Empirical Speed**	**Theoretical Speed**
2	0.57	0.0105	0.904	0.950
3	0.49	0.0112	0.848	0.742
5	0.37	0.0139	0.683	0.454

Table 3: Performance metrics for `gpt2` and `distilgpt2` as γ varies.

1. α **(Efficiency):** The efficiency α decreases as γ increases, starting at 0.57 for $\gamma = 2$ and reducing to 0.37 for $\gamma = 5$. This demonstrates diminishing efficiency at larger scaling factors.

2. **Empirical Time Per Token:** The empirical time per token slightly increases with γ, from 0.0105 at $\gamma = 2$ to 0.0139 at $\gamma = 5$. This suggests minor degradation in processing speed as scaling increases.

3. **Empirical Speed:** Empirical speed decreases from 0.904 at $\gamma = 2$ to 0.683 at $\gamma = 5$. This indicates a gradual performance decline as γ grows.

4. **Theoretical Speed:** Theoretical speed exhibits a more significant reduction compared to empirical speed, dropping from 0.950 at $\gamma = 2$ to 0.454 at $\gamma = 5$. This suggests theoretical predictions become less aligned with practical outcomes at higher γ.

Performance metrics for `distilgpt2` and `gpt2-medium`

Table 4 summarizes the performance metrics for `distilgpt2` (draft) and `gpt2-medium` (target) as the scaling factor γ varies. The key observations are:

1. α **(Efficiency):** Efficiency α decreases as γ increases, starting at 0.46 for $\gamma = 2$ and reducing to 0.26 for $\gamma = 5$, reflecting diminishing returns in efficiency at higher scaling factors.

γ	α	**Empirical Time Per Token**	**Empirical Speed**	**Theoretical Speed**
2	0.46	0.0178	1.033	1.124
3	0.38	0.0180	1.022	0.913
5	0.26	0.0200	0.920	0.609

Table 4: Performance metrics for `gpt2-medium` and `distilgpt2` as γ varies.

2. **Empirical Time Per Token:** The empirical time per token increases slightly, from 0.0178 at $\gamma = 2$ to 0.0200 at $\gamma = 5$, showing minor degradation in processing speed.
3. **Empirical Speed:** Empirical speed shows a slight decrease, from 1.033 at $\gamma = 2$ to 0.92 at $\gamma = 5$, indicating moderate performance decline.
4. **Theoretical Speed:** Theoretical speed drops more substantially, from 1.124 at $\gamma = 2$ to 0.609 at $\gamma = 5$, highlighting greater theoretical degradation compared to empirical performance.

Performance metrics for `distilgpt2`, `gpt2` and `gpt2-medium`

Table 5 summarizes the performance metrics for `distilgpt2` (subdraft), `gpt2` (draft) and `gpt2-medium` (target) as the scaling factor γ varies. The key observations are:

γ	α_{draft}	α_{target}	**Empirical Time Per Token**	**Empirical Speed**
2	0.59	0.49	0.0154	1.194
3	0.51	0.52	0.0169	1.088
4	0.46	0.52	0.0181	1.016
5	0.40	0.52	0.0186	0.989

Table 5: Performance metrics for `distilgpt2`, `gpt2` and `gpt2-medium` as γ varies.

1. $\alpha_{\textbf{draft}}$ **(Efficiency - draft):** The efficiency α_{draft} starts at 0.59 for $\gamma = 2$ and decreases to 0.40 at $\gamma = 5$, showing a reduction in draft model efficiency as the scaling factor increases.
2. $\alpha_{\textbf{target}}$ **(Efficiency - target):** Efficiency α_{target} remains relatively stable, starting at 0.49 for $\gamma = 2$ and holding steady at 0.52 from $\gamma = 3$ to $\gamma = 5$. This indicates a more consistent efficiency for the target model.
3. **Empirical Time Per Token:** The empirical time per token increases slightly as γ rises, from 0.0154 at $\gamma = 2$ to 0.0186 at $\gamma = 5$, reflecting a modest increase in processing time.
4. **Empirical Speed:** Empirical speed decreases gradually from 1.194 at $\gamma = 2$ to 0.989 at $\gamma = 5$, demonstrating a moderate performance decline as the scaling factor increases.

Performance metrics for `distilgpt2`, `gpt2` and `gpt2-large`

Table 6 summarizes the performance metrics for `distilgpt2` (subdraft), `gpt2` (draft), and `gpt2-large` (target) as the scaling factor γ varies. The key observations are:

γ	α_{draft}	α_{target}	**Empirical Time Per Token**	**Empirical Speed**
2	0.59	0.48	0.0227	1.440
3	0.50	0.50	0.0235	1.391
4	0.47	0.48	0.0259	1.262
5	0.41	0.48	0.0251	1.302

Table 6: Performance metrics for `distilgpt2`, `gpt2` and `gpt2-large` as γ varies.

1. $\alpha_{\textbf{draft}}$ **(Efficiency - draft):** The efficiency α_{draft} starts at 0.59 for $\gamma = 2$ and decreases to 0.41 at $\gamma = 5$, indicating a reduction in efficiency as the scaling factor increases for the draft model.

2. $\alpha_{\textbf{target}}$ **(Efficiency - target):** Efficiency α_{target} remains relatively stable, starting at 0.48 for $\gamma = 2$ and holding steady at 0.48 from $\gamma = 3$ to $\gamma = 5$, showing consistent efficiency for the target model.

3. **Empirical Time Per Token:** The empirical time per token increases from 0.0227 at $\gamma = 2$ to 0.0259 at $\gamma = 4$, before slightly decreasing to 0.0251 at $\gamma = 5$. This shows a slight increase in processing time with some fluctuations.

4. **Empirical Speed:** Empirical speed decreases gradually from 1.440 at $\gamma = 2$ to 1.302 at $\gamma = 5$, demonstrating a moderate performance decline as the scaling factor increases.

Performance metrics for `distilgpt2` and `gpt2-large` for Adaptive Gamma

Adaptive Gamma

Instead of using a fixed number of tokens for gamma, the value of gamma can be dynamically adjusted based on model confidence or token acceptance rate. If acceptance rates are high, gamma can be increased to allow more speculative steps; if the rates are low, gamma can be decreased to ensure fewer speculative tokens are generated.

Results

The table below summarizes the performance metrics for `distilgpt2` (draft) and `gpt2-large` (target) with the adaptive gamma (dynamic lookahead window). The key observations are:

Initial Gamma	α	Empirical Time Per Token	Empirical Speed
3	0.45	0.0267	1.224

Table 7: Adaptive Gamma (Dynamic Lookahead Window) for `distilgpt2` and `gpt2-large`.

1. **Initial Gamma:** The initial gamma is set to 3 with an efficiency $\alpha = 0.45$. The empirical time per token is 0.0267, and the empirical speed is 1.224.
2. **Analysis:** Target Model's empirical time per token was 0.0327s and using Adaptive Gamma reduced time per token to 0.0267s giving speed improvement 1.224x. This is slighlty better than static gamma of which gave speed of 1.202x. Here gamma fluctuates around 2.

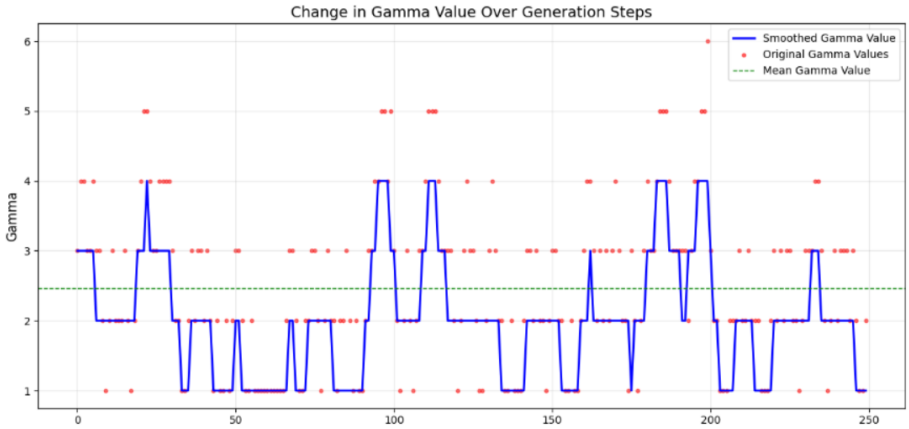

Figure 5: Visualization of Results for Adaptive Gamma with Dynamic Lookahead Window.

8 Conclusion

The experiments demonstrate the impact of scaling factors (γ) on the performance of different GPT models, including `distilgpt2`, `gpt2`, and `gpt2-large`. As the scaling factor increases, both efficiency (α) and empirical speed generally decrease, with diminishing returns in performance. Theoretical speed also tends to diverge from empirical results, especially at higher scaling factors, highlighting the challenges of predictive modeling at larger scales.

The use of adaptive gamma shows promising results, with a slight improvement in empirical speed compared to static scaling, indicating that dynamic adjustments based on model confidence can optimize processing times. These

findings suggest that careful tuning of scaling factors or adopting adaptive strategies may offer a balance between efficiency and performance, particularly in resource-constrained environments.

For Speculative Decoding to be effective, the draft model should me much smaller than the target model. In the original paper they used T5-XXL (11B) as target model and T5-small(77M) draft model which is around 140 times smaller than the target model to achieve speed improvement of 3.4x. The largest target model we used is GPT-Large (812M) and smallest draft model DistilGPT-2 (88.2M) which is just times smaller than the target model. Thus we could only achieve a speed improvement of 1.202x at a γ of 2.

References

[1] N. Shazeer, "Fast transformer decoding: One write-head is all you need," *CoRR*, 2019.

[2] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," 2022.