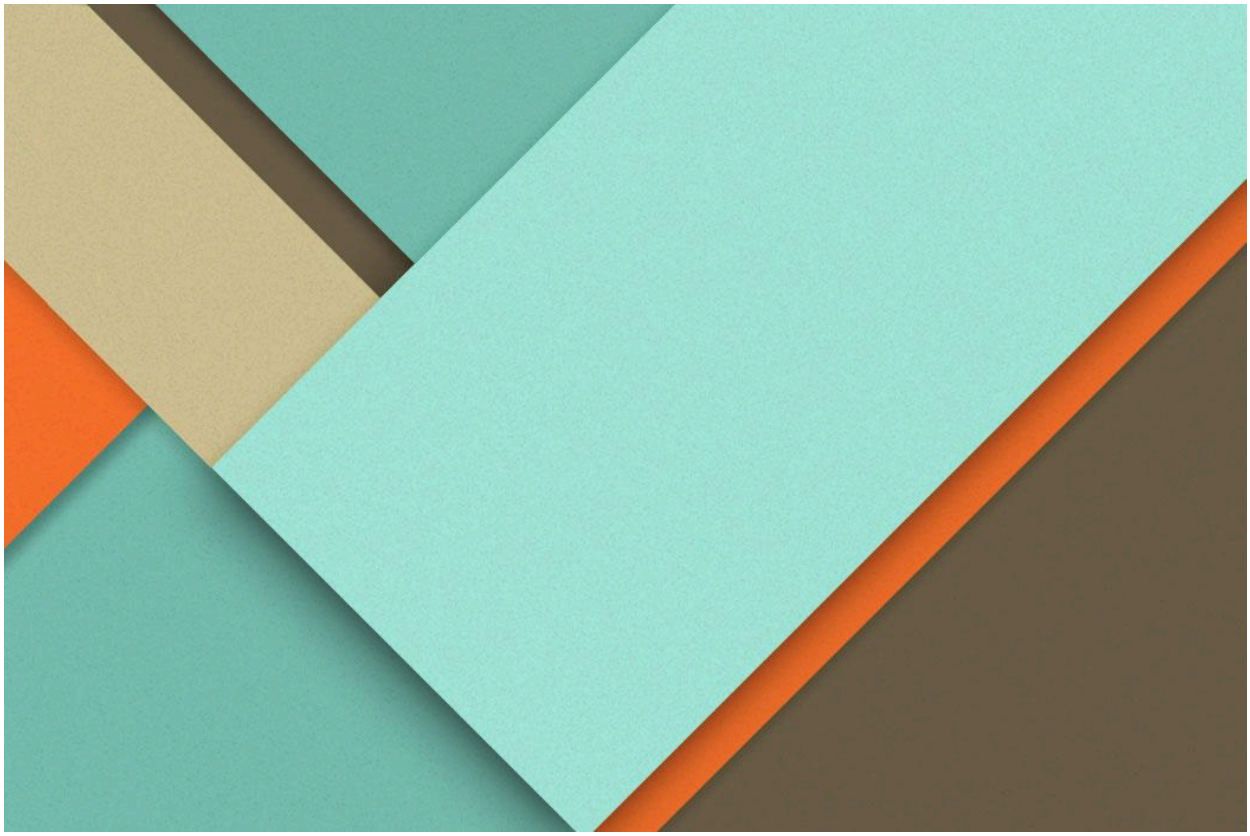

ANLP Project Outline Report

Fast Inference from Transformers via Speculative Decoding

Team Strawberry:

Rhythm Aggarwal 2021101081

Nipun Tulsian 2021101055

Vyom Goyal 2021101099

Niche Problem

The project aims to address the challenge of slow inference times in transformer-based models, which are widely used in natural language processing (NLP) tasks such as text generation, translation, and sentiment analysis. Despite their high accuracy and performance, these models often require significant computational resources, leading to delays that can hinder their application in real-time scenarios. This issue is particularly critical in areas like conversational AI, where rapid response times are essential for user satisfaction.

The niche problem identified is the need for a method that accelerates inference without sacrificing the quality of the generated outputs. The proposed solution is speculative decoding, which allows the model to generate multiple candidate sequences during inference and select the most promising ones efficiently. This approach aims to enhance the responsiveness of transformer models, making them more suitable for applications requiring real-time interaction

Literature Review:

Autoregressive models, particularly those based on the Transformer architecture (Vaswani et al., 2017), have revolutionized various domains, including natural language processing and computer vision. These models, such as GPT-3 (Brown et al., 2020) and LaMDA (Thoppilan et al., 2022), exhibit remarkable capabilities but face significant challenges in inference speed. The traditional decoding process is inherently serial, where generating K tokens requires K sequential runs of the model, leading to substantial latency. The large models are much more capable than smaller models.

Greedy Decoding

In a sequence-to-sequence problem, we are given an input sequence x = (x1 , . . . , xn), and we would like to predict the corresponding output sequence y = (y1, . . . , ym). These sequences might be source and target sentences in the case of machine translation, or low-resolution and high-resolution images in the case of image super-resolution. One common approach to this problem is to learn an

autoregressive scoring model p(y | x) that decomposes according to the left-to-right factorization

$$\log p(y \mid x) = \sum_{j} \log p(y_{j+1} \mid y_{\leq j}, x).$$

The inference problem is then to find $y^* = \arg\max_y p(y \mid x)$.

Since the output space is exponentially large, exact search is intractable. As an approximation, we can perform greedy decoding to obtain a prediction $\hat{y}$ as follows. Starting with an empty sequence $\hat{y}$ and $j = 0$, we repeatedly extend our prediction with the highest-scoring token $\hat{y}_{j+1} = \arg\max_{y_{j+1}} p(y_{j+1} \mid \hat{y}_{<=j}, x)$ and set $j \leftarrow j + 1$ until a termination condition is met. For language generation problems, we typically stop once a special end-of-sequence token has been generated.

Challenges in Inference Speed

The slow inference speed of large autoregressive models has prompted researchers to explore various optimization techniques. Existing methods often focus on reducing the computational cost uniformly across all inputs (Hinton et al., 2015; Jaszczur et al., 2021). In contrast, some strategies recognize that inference steps vary in complexity; simpler tasks can be approximated using less resource-intensive models (Han et al., 2021; Sukhbaatar et al., 2019). These adaptive computation methods aim to optimize resource allocation based on task difficulty. While many of these solutions have proven extremely effective in practice, they usually require changing the model architecture, changing the training-procedure and re-training the models, and don't maintain identical outputs which can prove to be impractical in many applications.

Fast Inference from Transformers via Speculative Decoding

In this paper the authors introduce *speculative decoding* - an algorithm to sample from autoregressive models faster *without any changes to the outputs*, by computing several tokens in parallel.

Their approach is based on the observations that:

(1) hard language-modeling tasks often include easier subtasks that can be approximated well by more efficient models

(2) using speculative execution and a novel sampling method, we can make exact decoding from the large models faster, by running them in parallel on the outputs of the

approximation models, potentially generating several tokens concurrently, and without changing the distribution.

The authors leverage speculative execution, a concept borrowed from computer architecture that allows parallel task execution while verifying their necessity (Burton, 1985; Hennessy & Patterson, 2012). By employing an efficient approximation model alongside the target model, they can generate multiple tokens concurrently which act as speculative prefixes for sl;ower target model and by deploying a novel sampling method they are maximizing the probability for these to be accepted, significantly reducing the number of required serial runs.

So compared to previous approaches used to accelerate inferencing, this method is easy to deploy in actual production settings as it doesn't require training new models and doesn't change the outputs.

Methodology and Implementation

The proposed speculative decoding method operates by:

1. Utilizing a smaller approximation model to generate ɣ speculative token completions.
2. The target model is then run in parallel to evaluate these guesses and their respective probabilities from Mq in parallel, accepting all those that can lead to an identical distribution.
3. Sampling an additional token from an adjusted distribution to fix the first one that was rejected, or to add an additional one if they are all accepted.

This approach not only maintains the output distribution but also achieves a notable acceleration—demonstrated with T5-XXL models showing improvements of 2X-3X over standard implementations without changing outputs (Roberts et al., 2022).

Results and Implications

The results presented by Leviathan et al. indicate that their method is particularly effective in scenarios where memory bandwidth is a bottleneck. The ability to enhance inference speed while preserving output integrity makes this approach highly applicable in real-world settings where computational resources are often constrained. One limitation of speculative execution in general, and of speculative decoding in particular, is that latency is improved through increased concurrency at the cost of an increased number of arithmetic operations. Thus, this method is not helpful for configurations where additional computation resources are not available. However, in common cases

where additional computation resources are available (e.g. when memory bandwidth is the bottleneck) this method provides the speedup with significant benefits.

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 $prefix + [x_1, \ldots, x_n, t]$

Table 2. Empirical results for speeding up inference from a T5-XXL 11B model.

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	**3.4X**
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	**2.6X**
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNDM	T5-SMALL ★	0	5	0.65	**3.1X**
CNNDM	T5-BASE	0	5	0.73	3.0X
CNNDM	T5-LARGE	0	3	0.74	2.2X
CNNDM	T5-SMALL ★	1	5	0.53	**2.3X**
CNNDM	T5-BASE	1	3	0.55	2.2X
CNNDM	T5-LARGE	1	3	0.56	1.7X

This table shows how by using the speculative decoding algorithm while inferencing results in speed up of the inferencing process.

Confident Adaptive Language Modeling

The proposed decoding method, Confident Adaptive Language Modeling (CALM), addresses the computational inefficiencies associated with large language models (LLMs) during inference. With the rapid advancement of Transformer-based architectures, the demand for efficient decoding mechanisms has intensified. CALM introduces a framework that dynamically allocates computational resources based on the complexity of each input, allowing for significant reductions in computational load without sacrificing performance.

Methodology and Implementation

CALM operates on the principle of early-exit decoding, where the model can decide to generate tokens based on intermediate layer representations rather than always using the final layer. This approach is particularly beneficial for autoregressive tasks, where predictions are interdependent. The key components of CALM include:

- Dynamic Compute Allocation: Instead of applying the full model for every token, CALM assesses whether to exit early based on local confidence scores derived from intermediate layers.
- Confidence Measures: The model employs a systematic method to determine when to exit early by calculating confidence scores at each layer and comparing them against predefined thresholds.
- Global Consistency Constraints: CALM ensures that early exits do not compromise the overall quality of the generated sequence by maintaining a high probability of consistency with a full model output.

Instead of always making a prediction based on the representation at the final layer, d_t^L, the key idea in early-exiting is to choose y_{t+1} more quickly, if confident, by computing $P(y_{t+1} \mid d_t^i) = softmax(W_i d_t^i)$ for some intermediate layer $i < L$ based on a confidence score $c_t^i \in [0, 1]$ and a $\lambda_t^i \in [0, 1]$ denoting some local early-exiting threshold

for layer i while processing token t. The model exits early if $c_t^i >= \lambda_t^i$, or otherwise proceeds to compute the next representation d_{t+1}^i.

There are difference confidence measures proposed by the authors, namely: Softmax response, Hidden-state saturation, Early exit classifier that differ in their parameter and compute operation efficiencies.

Blockwise Parallel Decoding for Deep Autoregressive Models

The authors propose a blockwise parallel decoding scheme, allowing multiple time steps to be predicted simultaneously, thereby addressing the inherent sequential nature of generation in autoregressive models. Their approach demonstrates significant speed improvements, achieving up to 4x wall-clock time reductions in real-time applications. In this work, the authors propose a simple algorithmic technique that exploits the ability of some architectures, such as the Transformer (Vaswani et al., 2017), to score all output positions in parallel. They train variants of the autoregressive model to make predictions for multiple future positions beyond the next position modeled by the base model. At test time, the algorithm employ these proposal models to independently and in parallel make predictions for the next several positions. then determine the longest prefix of these predictions that would have generated under greedy decoding by scoring each position in parallel using the base model.

Methodology and Implementation

Let the original model be $p_1 = p$, and suppose that we have also learned a collection of auxiliary models $p_2,\ldots,p_k$ for which $p_i(y_{j+1} \mid y_{<=j}, x)$ is the probability of the (j + i)th token being y_{j+i} given the first j tokens.

The following three substeps are repeated until the termination condition is met:

• **Predict**: Get the block predictions $\hat{y}_{j+i} = \text{argmax}_{yj+i}\ p_i(y_{j+i} \mid \hat{y}_{<=j}, x)$ for i = 1, . . . , k.
• **Verify**: Find the largest k such that $\hat{y}_{j+i} = \text{argmax}_{yj+i}\ p_1(y_{j+i} \mid \hat{y}_{<=j+i-1}, x)$ for all $1 \le i \le k$.

• **Accept**: Extend $\hat{y}$ with $\hat{y}_{j+1},\ldots,\hat{y}_{j+k}$ and set j←j+k.

While Inferencing:

- The method begins with an empty output sequence. For each iteration:
    - Predictions for several future tokens are made in parallel.

- The scoring model evaluates these predictions to determine which ones are valid.
- The output sequence is updated with the valid predictions, allowing for fewer total iterations compared to standard greedy decoding.

So for inferencing a sequence of length m it reduces the number of model invocations from m to 2m/k. To further enhance efficiency, the authors propose merging the scoring and proposal steps into a single model invocation. By modifying the Transformer architecture to compute predictions and scores simultaneously, they reduce the number of model invocations needed during decoding. This will reduce the number of model invocations from 2m/k to m/k. In terms of wall-clock time, fastest models exhibit real-time speedups of up to 4x over standard greedy decoding.

Why Speculative Decoding:

However, this algorithm only supports greedy decoding (temperature=0) and not the general stochastic setting, also it requires additional training of a custom models, and focuses on preserving down-stream task quality, instead of guaranteeing identical outputs. So **Speculative Decoding** can be integrated into existing systems with minimal modifications, as it does not require the training of additional auxiliary models. But **Blockwise Parallel Decoding,** necessitates training multiple proposal models alongside the base model. And since blockwise parallel decoding relies on a fixed number of predictions it may not efficiently handle diverse inputs. The **CALM** decoding approach is also able to achieve speedup of **2-3x** however unlike the speculative decoding approach, the CALM decoding requires confidence calibration and architectural changes. Also speculative decoding achieves better ROUGE scores and BLEU scores on the text summarisation (CNN/DM) task and the machine translation(WMT) task respectively than the other approaches.

Performance metrics for inference from different T5 models (without speculative decoding)

1) The models were run to perform Machine Translation tasks from English to German. The dataset we used was the **Multi30k** dataset. Below are the results obtained for the task performed:

Model	Average Inference Time	Average BLEU score
T5-small	0.3911 seconds	28.45
T5-base	1.0123 seconds	31.98
T5-large	2.9574 seconds	36.87

As can be seen from the above results, there is a very significant time difference in the inference times for the 3 models used above. But also there is a tradeoff with the accuracy as the BLEU score for the larger model is significantly higher than the smaller model. We will implement the speculative decoding method by which we intend to reduce the inference time for the bigger models while reducing the performance tradeoff.

2) The models were run to perform a summarisation task. The dataset used was CNNDM dataset. Below are the inference times:

Model	Inference Time
T5-small	1.1873 seconds
T5-base	3.6613 seconds
T5-large	11.9702 seconds