Intro

Current Status and Problem:

Neural autoregressive sequence-to-sequence models (e.g., Transformer) have become the standard for various tasks like machine translation, summarization, and speech synthesis.

- While these models offer increased parallel computation capabilities, leading to significantly faster training, **inference still generates outputs one token at a time**, posing a substantial bottleneck for many practical applications.

Proposed Contribution of This Work:

propose a simple algorithmic technique that **exploits the ability of some architectures, like the Transformer, to score all output positions in parallel.**

The prefix length L is determined by:

1. **Following the predictions of the proposal model one by one.**
2. **At each step, checking in parallel whether that prediction matches the base model's most accurate (greedy) next token prediction.**
3. **L then corresponds to the count of tokens up to the point where this match ceases.**

Traditional Sequential Method (Greedy Decoding):
Imagine you're climbing a staircase, and you can only step up one stair at a time.

- You take one step (generate one token).
- Then another step (generate the next token).
- And so on, one by one, until you reach the top.

This process is inherently **sequential**.
Proposed Method (Our Technique):
Now, imagine you're still climbing that staircase, but you have a special ability: sometimes you can **jump** over multiple stairs at once.

- You might take one step (generate one token).
- But then, you might be able to **jump two or three stairs** at once (confirm and skip multiple tokens).
- Then perhaps another single step, and another jump.

This "jump" size is exactly what **L (the length of the validated prefix)** represents.

- If L is large, it means you can jump over more stairs at once.
- The bigger the jump (the larger L is), the faster you can reach your destination (complete the entire sequence).

Key Idea:

- They train **variants of the autoregressive model (proposal models)** to make predictions for multiple future positions beyond the next token.
- During inference, these proposal models independently and in parallel predict the next several tokens.
- Then, the **base model is used to parallelly score each predicted token to determine the longest prefix that would have been generated under greedy decoding.**
- If the length of this validated prefix is greater than one, they are able to **skip one or more iterations of the greedy decoding loop.**

Greedy Decoding = common method for generating output sequences in sequence-to-sequence problems

Goal : to predict an output sequence y given an input sequence x.
This is achieved by learning an **autoregressive scoring model** $p(y|x)$, which determines the probability of the entire sequence y by sequentially calculating the probability of each subsequent token given previous tokens and the input x ($p(y_{j+1}|y_{\leq j},x)$).

Greedy Decoding Works: To overcome this computational limitation, greedy decoding is used as an approximation:

Sequential Selection: Starting with an empty output sequence, at each step, it extends the sequence by **selecting the single token that has the highest probability** ($\mathrm{argmax}_{y_{j+1}} p(y_{j+1}|\hat{y}_{\leq j},x)$) based on the currently generated sequence and the input x

Greedy decoding makes the **locally optimal choice (the highest probability token) at each step**, which allows for fast generation of plausible sequences. However, it does not guarantee a globally optimal output sequence.

Speculative decoding:

Step 1 : A Draft Model (or Proposal Model) quickly and in parallel "speculates" and proposes the next N tokens based on the current context. (e.g., t_1, t_2, t_3, t_4).

Step 2 : A Target Model (or Base Model) then "verifies" these N speculated tokens in parallel. This verification process checks whether each token, in its respective context, was indeed the token with the highest probability according to the Target Model's greedy decoding strategy.

Step 3: During this verification, the length L of the longest continuous sequence (prefix) of speculated tokens that match the Target Model's greedy choices is determined.

Step 4: If L is greater than 1, these L tokens are added to the output sequence all at once, effectively skipping (L-1) decoding iterations to accelerate the process.

Step 5 : If a mismatch occurs during verification, the algorithm stops accepting speculated tokens from that point onwards. Only the tokens up to the point immediately before the mismatch are confirmed. From the point of mismatch, the algorithm reverts to the standard greedy decoding approach, where the Target Model directly predicts the next token.

Blockwise Parallel Decoding)

"Auxiliary Models" (also known as Proposal Models or Draft Models):

- These models are trained to **predict future sequences that closely resemble the outcome of greedy decoding.**
- Based on the current context, they propose **multiple tokens (e.g., k tokens)** all at once. This proposal is a kind of "hypothesis" or "speculation."
- Here, each auxiliary model p_i probabilistically suggests "what the i-th token from the current context might be."

Comparison and Search with Greedy Decoding Results" (Verification):

- The **Base Model (Original Model or Target Model)** then **"verifies"** the tokens proposed by these auxiliary models.
- The verification process checks whether the proposed tokens **match the tokens that the base model would have chosen greedily.**
- For example, if the auxiliary model proposed t_1,t_2,t_3, the base model would check:
- "Is t_1, the first token I would greedily choose from the current context?"
- "Is t_2, the second token I would greedily choose from the current context + t?"
- "Is t_3, the third token I would greedily choose from the current context + t_1+t_2?
- This comparison and verification process can be performed **in parallel**, and the result determines the length of the **longest matching prefix (L)**.

auxiliary models act as a "preview" for greedy decoding, and the base model serves as the "judge" to confirm whether this preview "matches the actual greedy decoding result." By quickly skipping over the matching parts and only reverting to direct greedy decoding by the base model when a mismatch occurs, the overall generation speed is significantly accelerated.

```
# 1. Proposal model predicts k tokens
predicted_tokens = proposal_model.predict_n_tokens(current_context, k=5)
# e.g., predicted_tokens = ['the', 'cat', 'sat', 'on', 'mat']

# 2. Base model verifies these predictions in parallel
# This part is complex due to parallel computation, but conceptually it works like this:
check_results = []
current_sequence_for_verification = current_context # This is conceptual for sequential
understanding

for i in range(k):
   proposed_token = predicted_tokens[i]

   # Calculate what the base model would greedily choose for the next token in the current
sequence
   base_model_greedy_choice =
base_model.get_greedy_choice(current_sequence_for_verification)

   # Check if the proposed token matches the base model's greedy choice
   if proposed_token == base_model_greedy_choice:
      check_results.append(True)
      # For the next verification step, update the sequence (conceptually)
      current_sequence_for_verification += proposed_token
   else:
      check_results.append(False)
      # If a mismatch occurs, subsequent tokens don't need to be verified for the prefix,
      # but in a truly parallel verification, results for all K tokens might still be obtained.
      # This example is sequential for conceptual clarity; actual parallel implementation differs.
      # In a parallel implementation, check_results would be populated in a single operation.

# 3. Determine L using the populated check_results
L = 0
for i in range(k):
   if check_results[i] == True:
      L = i + 1
   else:
      break

print(f"Length of the validated prefix (L): {L}")
```

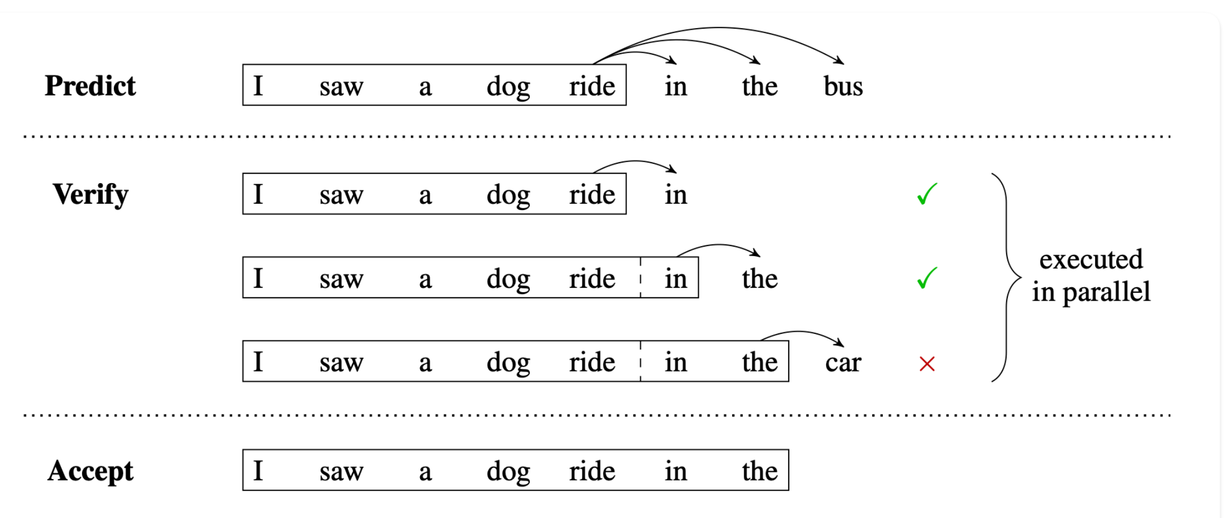


Figure 1: The three substeps of blockwise parallel decoding. In the **predict** substep, the greedy model and two proposal models independently and in parallel predict "in", "the", and "bus". In the **verify** substep, the greedy model scores each of the three independent predictions, conditioning on the previous independent predictions where applicable. When using a Transformer or convolutional sequence-to-sequence model, these three computations can be done in parallel. The highest-probability prediction for the third position is "car", which differs from the independently predicted "bus". In the **accept** substep, $\hat{y}$ is hence extended to include only "in" and "the" before making the next k independent predictions.

Little time lost compare to a single greedy prediction

Draft model : $\hat{y}_{j+i} = \mathrm{argmax}_{y_{j+i}} p_i(y_{j+i} | \hat{y}_{\leq j}, x)$ for $i = 1, \ldots, k$.

$\hat{y}_{j+l} = \mathrm{argmax}_{y_{j+i}} p_i(y_{j+i} | \hat{y}_{\leq j}, x)$ →

refers to an **'auxiliary model' or 'draft model'**, where each p_i predicts the i-th future token. Through argmax, the token with the highest probability is selected to propose (predict)

This process is the step of proposing a block of k tokens quickly and in parallel."

Verify model : Find the largest $\hat{k}$ such that $\hat{y}_{j+i} = \mathrm{argmax}_{y_{j+i}} p_1(y_{j+i} | \hat{y}_{\leq j+i-1}, x)$ for all $1 \leq i \leq \hat{k}$.

Note that $\hat{k} \geq 1$ by the definition of $\hat{y}_{j+1}$."

find the longest prefix of the proposed length-k extension that would have otherwise been produced by p_1.

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

extend our hypothesis with the verified prefix. By stopping early if the base model and the proposal models start to diverge in their predictions, we ensure that we will recover the **same output** that would have been produced by running greedy decoding with p_1.

Flow :

Step 1
p_i = original model, base model

Step 2
$\hat{y}_{j+l}$ = checks if greedy coding matces p_i

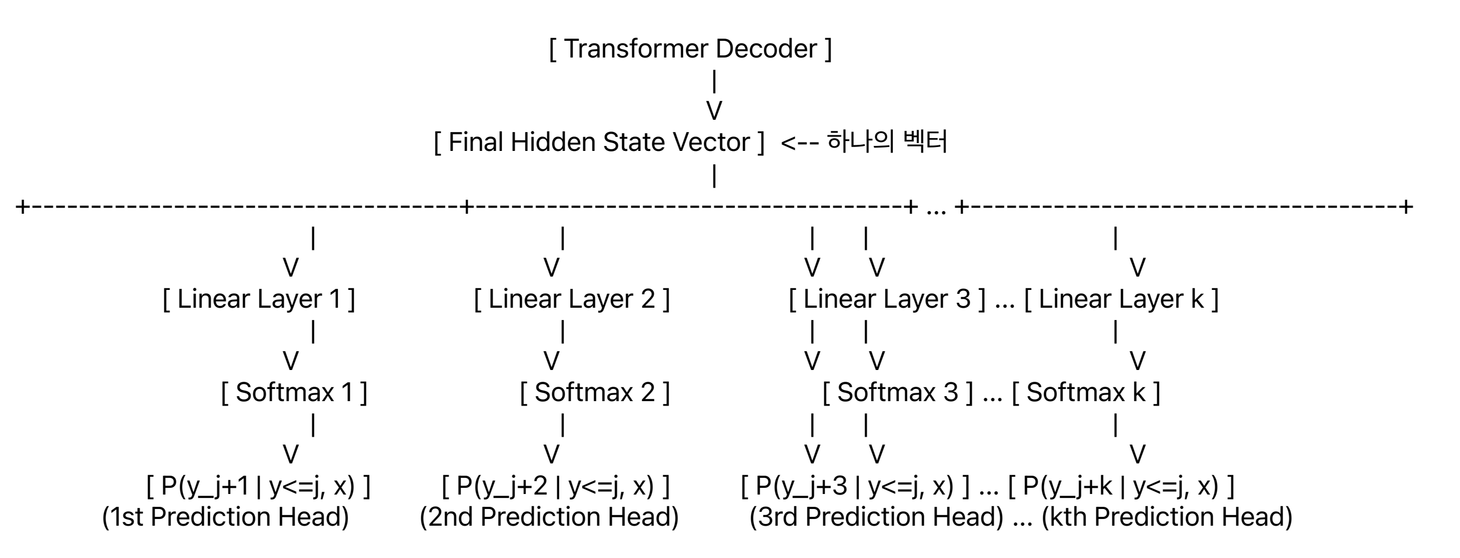
Step 3
$\hat{y}_{\leq j+i-1}$ = refers to the context that includes the tokens verified so far, and p_1 calculates, via argmax, what the next token should be in this context."

Step 4
"This value is compared with the $\hat{y}_{j+l}$ proposed by the draft model to check for a match.
$\hat{k}$ represents the length of the longest contiguous prefix that passed this verification."

The Potential of this Scheme :

The efficiency of Speculative Decoding heavily relies on the main model's ability to verify many proposed tokens at once. Transformer models are perfect for this because, even though they do a lot of calculations overall (quadratic complexity), the parts that *have* to happen one after another (sequential operations) don't get longer as the sequence grows.

This means a Transformer **can check a whole block of k tokens proposed by the draft model in roughly the same amount of real-world time it would take to check just one, leading to significant speed improvements.**



Combined Scoring and Proposal Model = trained to be both the high-quality Base Model *and* to generate proposals.

- The crucial change is in its **final output layer**. Instead of a single "head" that predicts only the very next token (y_{j+1}), it has k **separate " prediction heads."**

instead of two separate models each processing the context, one model processes the context *once*, and then uses that result to generate *all* the necessary predictions (proposals and verification scores) simultaneously through its specialized k-heads. This reduces the number of model invocations per step from two to one.

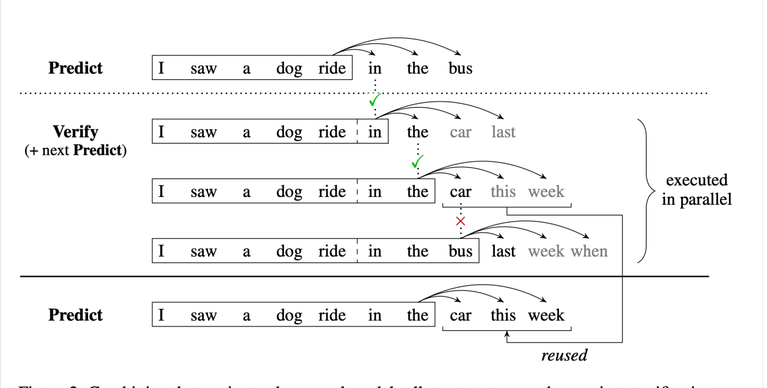


Figure 2: Combining the scoring and proposal models allows us to merge the previous verification substep with the next prediction substep. This makes it feasible to call the model just once per iteration rather than twice, halving the number of model invocations required for decoding.

Approximate Inference

Def: In most complex probabilistic models and large-scale machine learning models (especially deep learning models), exact inference is practically impossible. This is due to finite computational time and resources. Therefore, we need to develop methods for finding sufficiently good (good enough) approximate solutions quickly and efficiently

Approximate Inference/ Top-K selection:
=presents methods for relaxing this strict criterion

even if the draft model's proposal doesn't exactly match p_1's highest probability prediction, it is accepted if it falls within the top k tokens predicted by p_1.

Idea :strategy to **prioritize computational efficiency for faster inference, even if it means slightly compromising accuracy.**"

$$\hat{y}_{j+i} \in \text{top-}k_{y_{j+i}} p_1(y_{j+i} \mid \hat{y}_{\leq j+i-1}, x).$$

Approximate Inference/ Distance-Based Selection:

this method accepts the proposal if the **semantic distance or embedding space distance** between the proposed token and p_1's top prediction is within a certain threshold.

$$d\left(\hat{y}_{j+i}, \operatorname*{argmax}_{y_{j+i}} p_1(y_{j+i} \mid \hat{y}_{\leq j+i-1}, x)\right) \leq \epsilon.$$

It allows the selection of a token that is **not necessarily p_1's highest probability prediction**

It allows the selection of a token that is **not necessarily p_1's highest probability prediction.** However, under the premise that "semantic similarity is good enough," it increases the verification pass rate, thereby boosting speed. It's **approximating accuracy based on semantic similarity.**

Approximate Inference/Minimum Block Size:

When the draft model proposes k tokens, this method **unconditionally accepts at least N tokens** (where $N<k$) and proceeds to the next step, regardless of how many tokens ($\hat{k}$) were actually verified as matching p_1's greedy path.

Knowledge Distillation

- **Core Concept:** It's a technique where a smaller or more difficult-to-train "student model" is trained to mimic the knowledge (often the output probability distributions or predictions) of a larger, more complex, and higher-performing "teacher model."
- **Purpose:** Primarily used for model compression (making the student model smaller for easier deployment) or to stabilize training.
- **Interesting Point:** As cited from Furlanello et al. (2017), distillation can sometimes improve performance even when the teacher and student models have the same architecture and size.

Application to Blockwise Parallel Decoding:

- **Hypothesis:** The authors hypothesize that sequence-level distillation could be particularly beneficial for Speculative Decoding.
- **Reasoning:** This is because distillation tends to result in a training set with "greater predictability due to consistent mode breaking from the teacher model."
- **"Mode breaking":** This refers to the tendency of a model to select one coherent output path among many possibilities. Decoding strategies like beam search explore promising paths to generate more consistent sequences.
- **"Greater predictability in the training set":** Data generated by beam search tends to be more consistent and "predictable" (i.e., less ambiguous for the model) than data generated by greedy decoding or random sampling. Since the drafting part of Speculative Decoding needs to "predict" future tokens, training on such consistent data is expected to lead to better proposals.

How Distilled Data is Produced:

- **Teacher Model:** A pre-trained model with the **same hyperparameters as the baseline but a different random seed** is used. (This aligns with the idea from Furlanello et al. to show distillation effects even with identical architectures/sizes.)
- **Decoding Method: Beam decoding** is used to generate text from the teacher model. Beam search generally produces higher-quality sequences than simple greedy decoding.
- **Beam Search Hyperparameters:** The standard beam search settings from Vaswani et al. (2017) are adopted.
-

by training the draft model on data generated through high-quality beam search predictions, it will be able to make more consistent and predictable proposals, thereby improving overall decoding performance.

Experiments :

Machine Translation

Baseline Setup:

- **Task:** English-German Machine Translation using the WMT 2014 dataset.
- **Model:** A Transformer model trained for 1,000,000 steps on 8 P100 GPUs with transformer_base hyperparameters.
- **Baseline Performance:** Achieved a **BLEU score of 25.56** on the newstest2013 development set using greedy decoding. This serves as the quality benchmark.

Speculative Decoding Model Training:

- **Model Type:** "Combined scoring and proposal Transformer models" (as described in Section 4, featuring k prediction heads for unified proposal and verification).
- **Block Sizes (k):** A collection of these models was trained for various block sizes k to analyze its effect on decoding efficiency.
- **Training Process:** Each model was trained for an **additional 1,000,000 steps** (fine-tuning) on the same hardware, building upon the baseline's weights. Optimizer accumulators and the learning rate schedule were reset for these new training runs.
- **Training Data Variation (incorporating Knowledge Distillation from Section 6.2):**
- **Original Training Data:** Models trained on the standard WMT 2014 dataset.
- **Distilled Training Data:** This data was generated from a *separate* pre-trained Transformer model (with identical hyperparameters to the baseline but a different random seed) acting as a "teacher." The teacher model produced high-quality sequence predictions using **beam decoding** (with parameters from Vaswani et al. 2017). The Speculative Decoding models were then trained on these "distilled" beam-decoded outputs, aiming to improve proposal quality and consistency.

k	Regular ●	Distillation ■	Fine Tuning ▲	Both ♦
1	26.00 / 1.00	26.41 / 1.00		
2	25.81 / 1.51	26.52 / 1.55	25.74 / 1.78	26.58 / 1.88
4	25.84 / 1.73	26.31 / 1.85	25.05 / 2.69	26.36 / 3.27
6	26.08 / 1.76	26.26 / 1.90	24.69 / 2.98	26.18 / 4.18
8	25.82 / 1.76	26.25 / 1.91	24.27 / 3.01	26.11 / 4.69
10	25.69 / 1.74	26.34 / 1.90	23.51 / 2.87	25.60 / 4.95

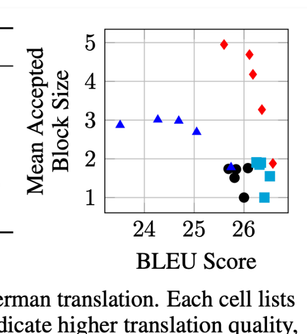


Table 1: Results on the newstest2013 development set for English-German translation. Each cell lists BLEU score and mean accepted block size. Larger BLEU scores indicate higher translation quality, and larger mean accepted block sizes indicate fewer decoding iterations. The data from the table is also visually depicted in a scatter plot on the right.

the results of applying Speculative Decoding to machine translation,

Initial Gains :
Even with a basic setup (original data, frozen parameters), Speculative Decoding improves speed, achieving a mean block size of 1.76 without sacrificing BLEU score.

Knowledge Distillation:
BLEU score increases by 0.43, and the mean block size reaches 1.91 for comparable settings. This confirms that distilled data enhances the model's predictive capability for efficient decoding.

Fine Tuning:

- **Fine-tuning leads to large increases in mean block size**, demonstrating its importance for efficiency.
- The model with the highest mean block size (4.95) still maintains reasonable quality (only 0.81 BLEU points worse than the initial distilled model).

Trade-off & Distillation: Speculative decoding presents a BLEU-vs-speed trade-off, where **knowledge distillation effectively reduces quality degradation at higher speeds**

Approximate Inference Effectiveness: Top-k approximate decoding boosts speed at a quality cost, whereas **minimum block size is inefficient**, yielding poor quality for minimal speed benefits.

Learnings :

It's crucial not to simply use a pre-trained model "as is => align with the specific objective is essential

Actively consider advanced training techniques like Knowledge Distillation.

techniques like "Top-k Selection" are valid methods for achieving additional speed gains. However, this comes with a clear trade-off in quality, which must be acknowledged and managed during application..