Fast Inference from Transformers via Speculative Decoding Presentation

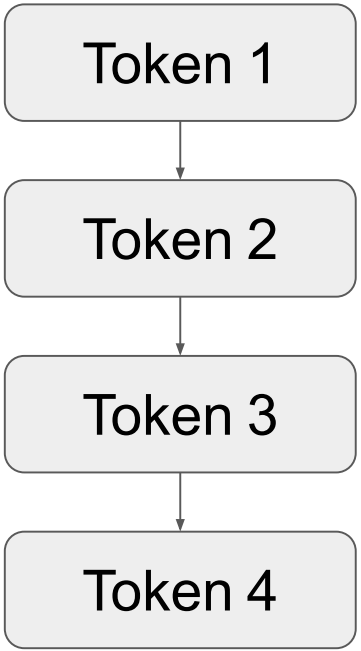
By: Daniel Sarmiento, Abel Varghese

Yaniv Leviathan, Matan Kalman, Yossi Matias Proceedings of the 40th International Conference on Machine Learning, PMLR 202:19274-19286, 2023.

Why This Paper Matters

- Large autoregressive Transformers are powerful, but slow at inference.
- Standard decoding is serial: generating K tokens requires K sequential model runs.
- This paper asks whether we can speed up decoding without retraining models or changing the output distribution.

Standard autoregressive decoding.



Sequential decoding increases latency.

Problem Statement

- Goal: accelerate decoding from a large target model.
- Constraints: no retraining, no architecture changes, and no change to the output distribution.
- Core setup: use a smaller approximation model to propose tokens, then verify them with the target model.

Two-model setup.

Target model $\mathbf{M_p}$
Large
Accurate
Slow

Approximation model $\mathbf{M_q}$
Smaller
Faster
Generates draft tokens

Background: Autoregressive Decoding

• Autoregressive language models generate one token at a time.

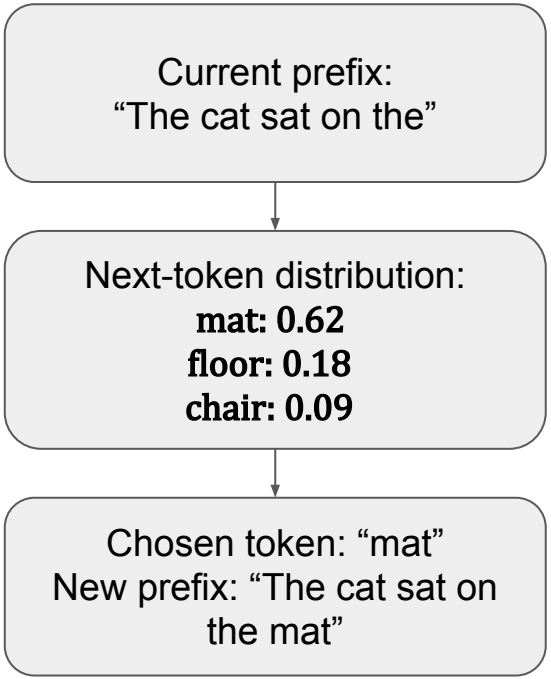
• At step t, the model predicts the next token from the previous prefix $x_{<t}$.

• Because each token depends on earlier tokens, decoding is inherently sequential.

$$p(x_t \mid x_{<t})$$

Probability of the next token given the previous tokens.

Example: next-token prediction.



Background: Prior Approaches to Faster Inference

Efficiency methods

- **Knowledge distillation**: train a smaller student model to imitate a larger one *(Hinton et al., 2015).*
- **Quantization**: reduce precision of model weights/activations to lower compute cost *(Jacob et al., 2018).*
- **Pruning / sparsity**: remove less important weights to shrink computation *(Han et al., 2015).*
- **Architecture changes**: redesign the model itself for faster inference.

Adaptive computation

- **Early exiting**: stop computation early for easier examples *(Graves, 2016; Xin et al., 2020).*
- **Input-dependent computation**: use more compute only when needed.

This paper

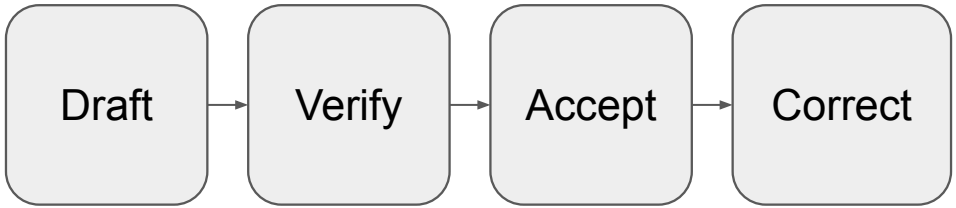
- No retraining.
- No architecture changes.
- Preserves the original output distribution.
- Speeds up decoding through speculative execution.

Key gap: many prior methods improve speed, but often require retraining or change the model's behavior.

Key Idea: Speculative Decoding

- Use a smaller approximation model to draft several candidate tokens.
- Use the large target model to evaluate those guesses in parallel.
- Accept correct guesses and correct the first rejected one, while preserving the target model's output distribution.

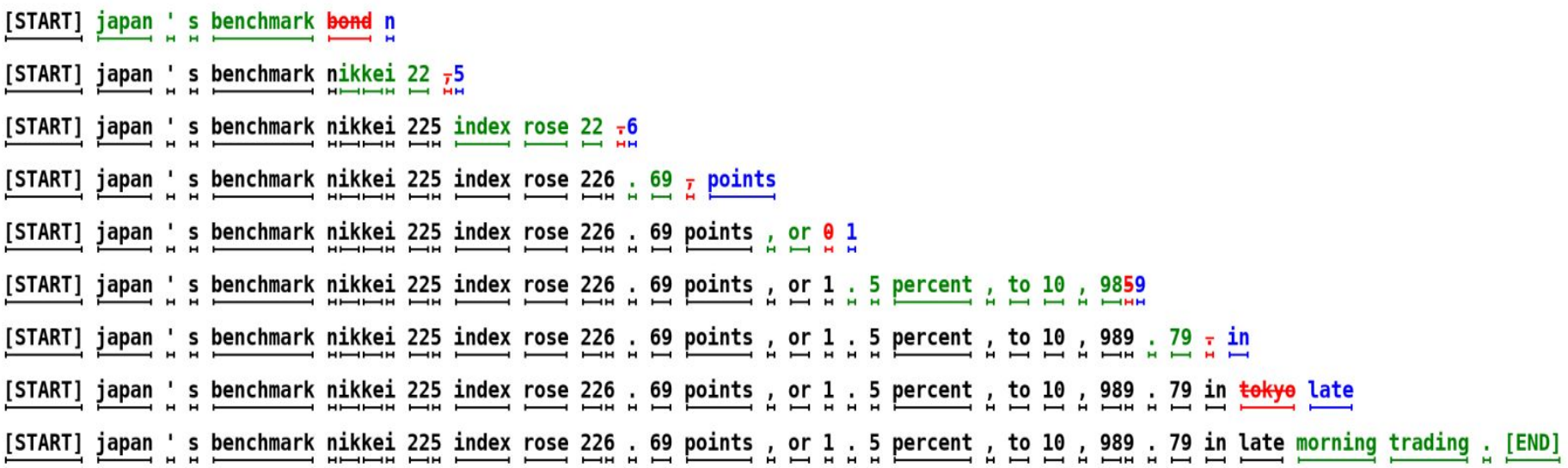
How speculative decoding works.



$\mathbf{Mp}$ = target model | $\mathbf{Mq}$ = approximation model | $\boldsymbol{\gamma}$ = number of guesses

One target-model pass can produce up to $\boldsymbol{\gamma + 1}$ tokens.

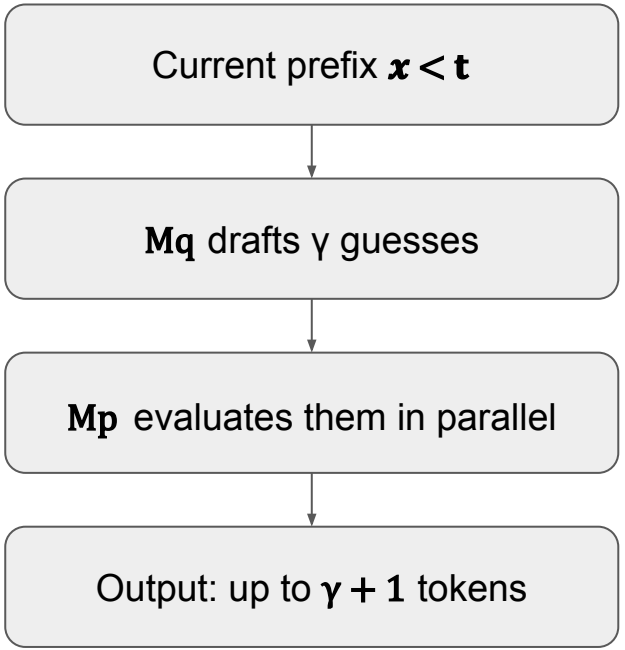
Intuition with Figure 1



Green = accepted guesses, **Red** = rejected guesses, **Blue** = corrections from the target model.
Figure adapted from Leviathan et al., 2023.

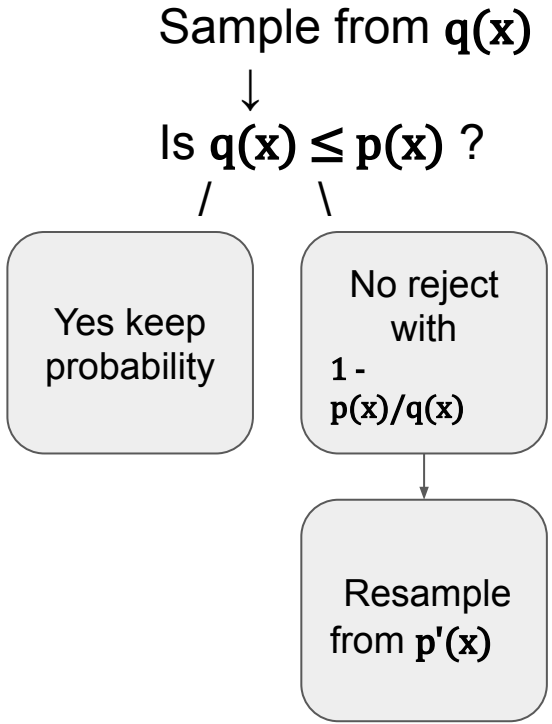
Setup and Notation

- $\mathbf{Mp}$ = large target model.
- $\mathbf{Mq}$ = smaller approximation model.
- $\mathbf{p}(\boldsymbol{x}\square|\boldsymbol{x}<\boldsymbol{t})$ = next-token distribution from Mp.
- $\mathbf{q}(\boldsymbol{x}\square|\boldsymbol{x}<\boldsymbol{t})$ = next-token distribution from $\mathbf{Mq}$.
- $\boldsymbol{\gamma}$ = number of speculative guesses.



Speculative Sampling Rule

- First sample a token from the approximation distribution $\mathbf{q(x)}$.
- If $\mathbf{q(x) \leq p(x)}$, keep the token.
- If $\mathbf{q(x) > p(x)}$, reject it with probability $\mathbf{1 - p(x)/q(x)}$.
- If rejected, resample from an adjusted distribution $\mathbf{p'(x)}$.



Algorithm Walkthrough

1. Draft guesses with $\mathbf{Mq}$

   Use the approximation model to autoregressive generate γ candidate tokens.

2. Evaluate with $\mathbf{Mp}$

   Run the target model on the current prefix and on the guessed continuations in parallel.

3. Count accepted guesses

   Accept guesses until the first rejection.

4. Adjust the distribution

   If a guess is rejected, build an adjusted distribution to correct it.

5. Return output tokens

   Output all accepted guesses plus one final token from the target-side correction step.

Each speculative step always returns at least one new token, and can return several.

Toy Example of Speculative Decoding

Current Prefix: “The cat sat on the”

Small model **Mq** guesses:

mat | floor | chair

Green = accepted guess
Red = rejected guess
Blue = correction
Gray = unused later guess

Target model **Mp** checks:

Accepted | First rejection

Correction from target-side distribution:

Rug

Returned tokens after one speculative step:

mat | Rug

Why the Method Is Exact

• The approximation model first proposes a token using $\mathbf{q(x)}$.

• Accepted guesses contribute the overlap between $\mathbf{q(x)}$ and the target distribution $\mathbf{p(x)}$.

• If a guess is rejected, the remaining probability mass is corrected so the final sample still follows $\mathbf{p(x)}$.

$$\mathbf{[\,Accepted\ part\,]\ +\ [\,Corrected\ part\,]}$$

$$\downarrow$$

Final distribution: $\mathbf{p(x)}$

Why Speculative Decoding Speeds Up Inference

- If many speculative guesses are accepted, one target-model pass can produce multiple tokens.
- Fewer serial target-model passes means lower decoding latency.
- The speedup depends on the acceptance rate $\boldsymbol{\alpha}$ and the approximation-model cost $\mathbf{c}$.

Standard decoding

Pass 1 → Token 1
Pass 2 → Token 2
Pass 3 → Token 3
Pass 4 → Token 4

Speculative decoding

Pass 1 → Token 1, Token 2, Token 3
Pass 2 → Token 4, Token 5
Pass 3 → Token 6, Token 7

Fewer serial target-model passes → lower latency.

Tradeoff: Speed vs Extra Compute

Speculative decoding improves latency, but it is not always a free speedup.

Benefit

- Fewer serial target-model passes.
- Lower decoding latency.
- Can generate multiple tokens per target-model step.

Cost / Limitation

- More parallel verification work.
- Total arithmetic operations can increase.
- Best when additional compute resources are available.

Speculative decoding is mainly a latency optimization, not always a total-compute optimization.

Experimental Setup

Tasks • English → German translation • CNN/DailyMail summarization	**Models** • Target model: T5-XXL (11B) • Approximation models: T5-small (77M) T5-base (250M) T5-large (800M)
Sampling settings • Argmax sampling **(temp = 0)** • Standard sampling **(temp = 1)**	**Evaluation setup** • **Batch size = 1** • Single TPU-v4 • Measured walltime speedup

The experiments test whether speculative decoding speeds up real inference in practice.

Main Results

Speculative decoding gives substantial walltime speedups on T5-XXL.

Task	Approx. Model	Sampling	Speedup
En-De Translation	**T5-small**	**temp = 0**	**3.4×**
En-De Translation	T5-small	temp = 1	**2.6×**
CNN/DailyMail	T5-small	temp = 0	**3.1×**
CNN/DailyMail	T5-small	temp = 1	**2.3×**

Best observed speedups were **3.4×** for translation and **3.1×** for summarization, with strong gains even under standard sampling.

Additional Findings

Acceptance rate insights

• Approximation models much smaller than the target model often achieved acceptance rates between 0.5 and 0.9.

• Sharper sampling distributions tended to produce higher acceptance rates.

• Higher acceptance rates generally led to larger speedups.

Surprising result

• Even trivial unigram and bigram models gave nonzero acceptance rates.

• On English→German translation, a bigram model reached about $\alpha = 0.20$.

• Because its cost is negligible, even this simple model gave about a $1.25\times$ speedup.

Speculative decoding does not require a perfect draft model—even simple approximations can help.

Strengths of the Paper

Exactness Final samples are distributed exactly like the target model.	**Practicality** The method can be added without retraining the large model.
Compatibility It works with existing smaller models rather than requiring a new custom draft model.	**Empirical impact** The reported gains are large enough to matter in real inference settings.

Limitations and Critique

Why the method is strong, but not universally optimal

Method limitations

- The method can increase total arithmetic operations.
- It works best when extra parallel compute is available.
- It is less helpful when hardware is already compute-bound.
- Performance depends on how well the approximation model matches the target model.

Evaluation critique

- The experiments show strong results, but mostly focus on text tasks.
- Real-world gains may vary across hardware systems and software implementations.
- The paper emphasizes latency improvement more than total-compute efficiency.
- More discussion of broader deployment settings would strengthen the paper.

Broader Impact / Later Relevance

Serving relevance

Large language models are often limited by decoding latency, so faster exact inference is practically valuable.

Conceptual impact

The paper introduced a clean draft-and-verify view of decoding: a small model proposes, and a large model confirms.

Research influence

The paper helped motivate later work on speculative decoding and speculative sampling for large-model inference.

The paper’s long-term importance is that it made faster exact decoding feel practical, not just theoretical.

Conclusion / Key Takeaways

• Speculative decoding speeds up autoregressive decoding by drafting tokens with a smaller model and verifying them with a larger one.

• The method preserves the target model's original output distribution exactly.

• It achieves meaningful real speedups, with about $2\times-3\times+$ gains in the paper's experiments.

• Its effectiveness depends on system conditions, especially acceptance rate and available parallel compute.

Looking ahead

Possible future directions include better draft models, adaptive choices of γ, and extensions beyond text generation.

Main message: faster decoding does not have to mean changing what the large model would have generated.

References

- Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast Inference from Transformers via Speculative Decoding. Proceedings of the 40th International Conference on Machine Learning (ICML).
- Raffel, C., Shazeer, N., Roberts, A., et al. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Journal of Machine Learning Research.
- Roberts, A., Chung, H. W., Levskaya, A., et al. (2022). Scaling Up Models and Data with T5X and SeqIO. arXiv:2203.17189.
- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention Is All You Need. NeurIPS.
- Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv.
- Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both Weights and Connections for Efficient Neural Networks. NeurIPS.
- Jacob, B., Kligys, S., Chen, B., et al. (2018). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. CVPR.
- Graves, A. (2016). Adaptive Computation Time for Recurrent Neural Networks. arXiv.
- Xin, J., Tang, R., Yu, J., & Lin, J. (2020). DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. ACL.
- Figures adapted from Leviathan et al. (2023) where noted.