Fast Inference from Transformers via Speculative Decoding

Presented by Christopher McClanahan

Yaniv Leviathan, Matan Kalman, Yossi Matias

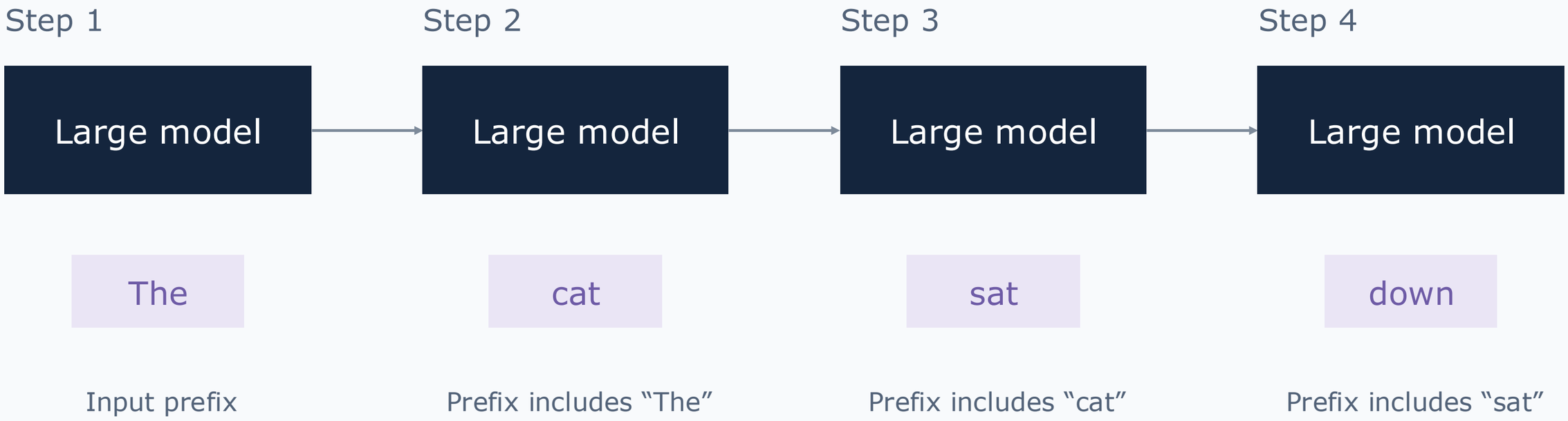
Google Research · ICML 2023

Paper presentation

arXiv:2211.17192v2

Autoregressive decoding has a serial bottleneck

Each new token depends on the tokens generated before it



K output tokens ≈ K expensive serial target-model steps

CPU parallelism meets a control-flow dependency

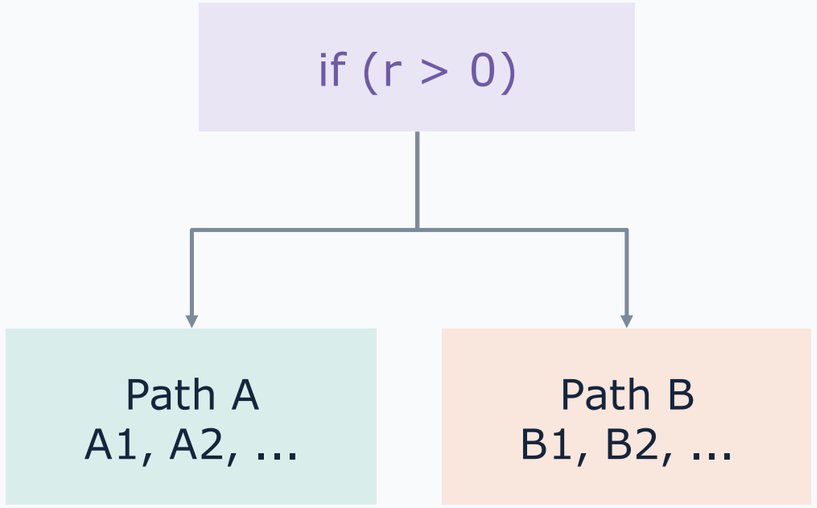
Independent instructions can overlap. A branch makes the next path uncertain

Simplified issue schedule without branch prediction

Cycle	1	2	3	4	5	6
ALU 1	I1	I3	wait	wait	A1	A3
ALU 2	I2	I4	wait	wait	A2	A4
Branch	load	test	wait	resolve		

Useful work overlaps

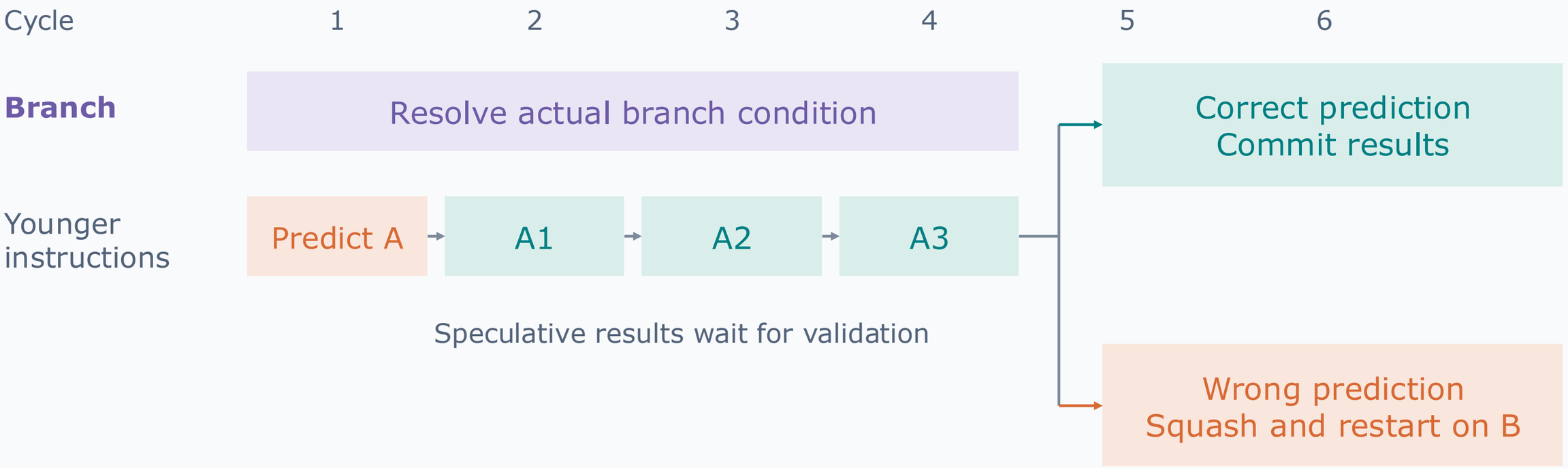
Path-dependent work waits



An unresolved branch can leave execution capacity idle

Speculative execution keeps useful work in flight

Predict a branch path and execute it while the processor resolves the condition



The payoff depends on accurate predictions and cheap recovery

CPU background: Intel, Speculative Execution and Control-Flow Speculation

The problem: faster decoding with exact sampling

Can a cheap predictor reduce expensive serial steps while preserving the target model?

Memory bandwidth often limits decoding, leaving parallel compute unused



01

Speculative sampling

A stochastic accept-and-correct rule

02

Speculative decoding

Existing models, no retraining

03

Analysis and evidence

A speed model and T5 walltimes

Paper §1–2 and Appendix A.1

One speculative decoding iteration

The drafter proposes γ tokens. The target checks the proposed continuation in parallel

1 Draft sequentially

the → cat → slept → soundly

2 Verify once

Target evaluates p_1, p_2, p_3, p_4 and p_5 in one parallel pass

3 Keep the prefix

accept | accept | reject | discard

Output this iteration

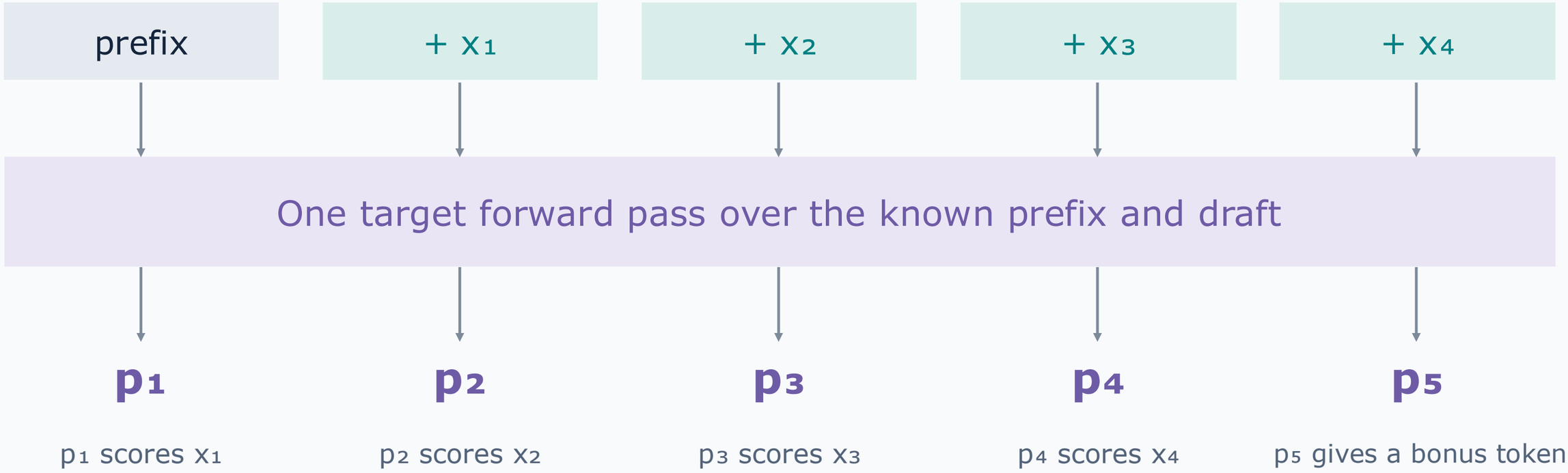
the | cat | sat

Correction from adjusted p_3

Paper §2.1–2.3, Algorithm 1

Parallel target scoring over a known draft

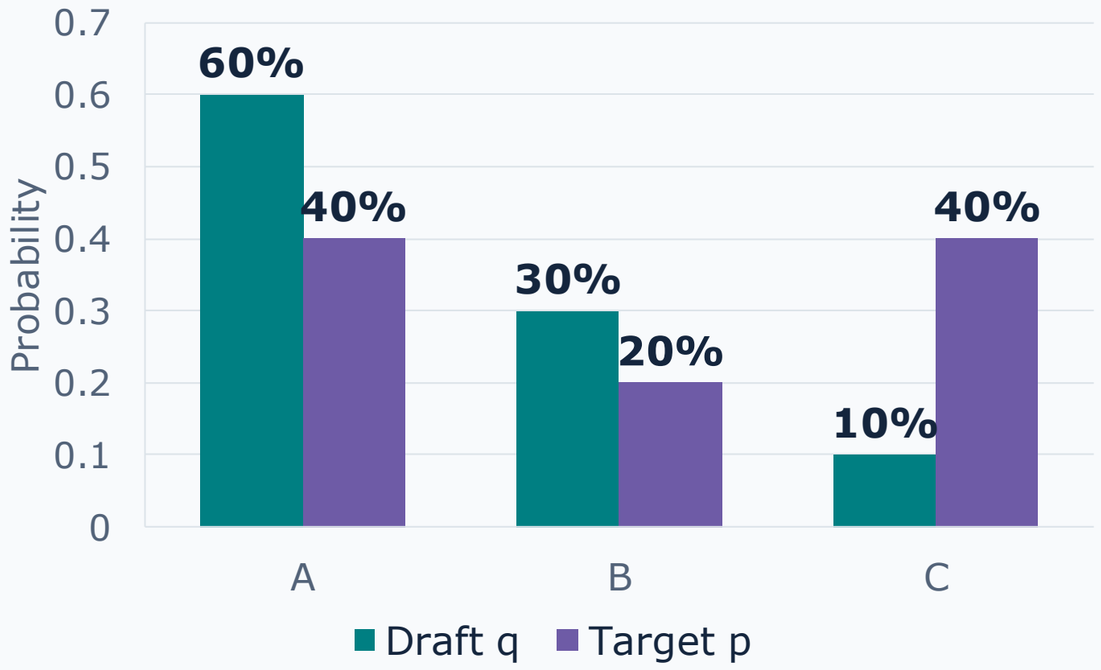
A causal Transformer can score every position once the proposed sequence is known



The large model parallelizes verification across positions

Exact sampling through acceptance and correction

A three-token example shows why both the acceptance test and correction matter



Accept x ~ q with probability

min(1, p(x) / q(x))

On rejection, sample a replacement from

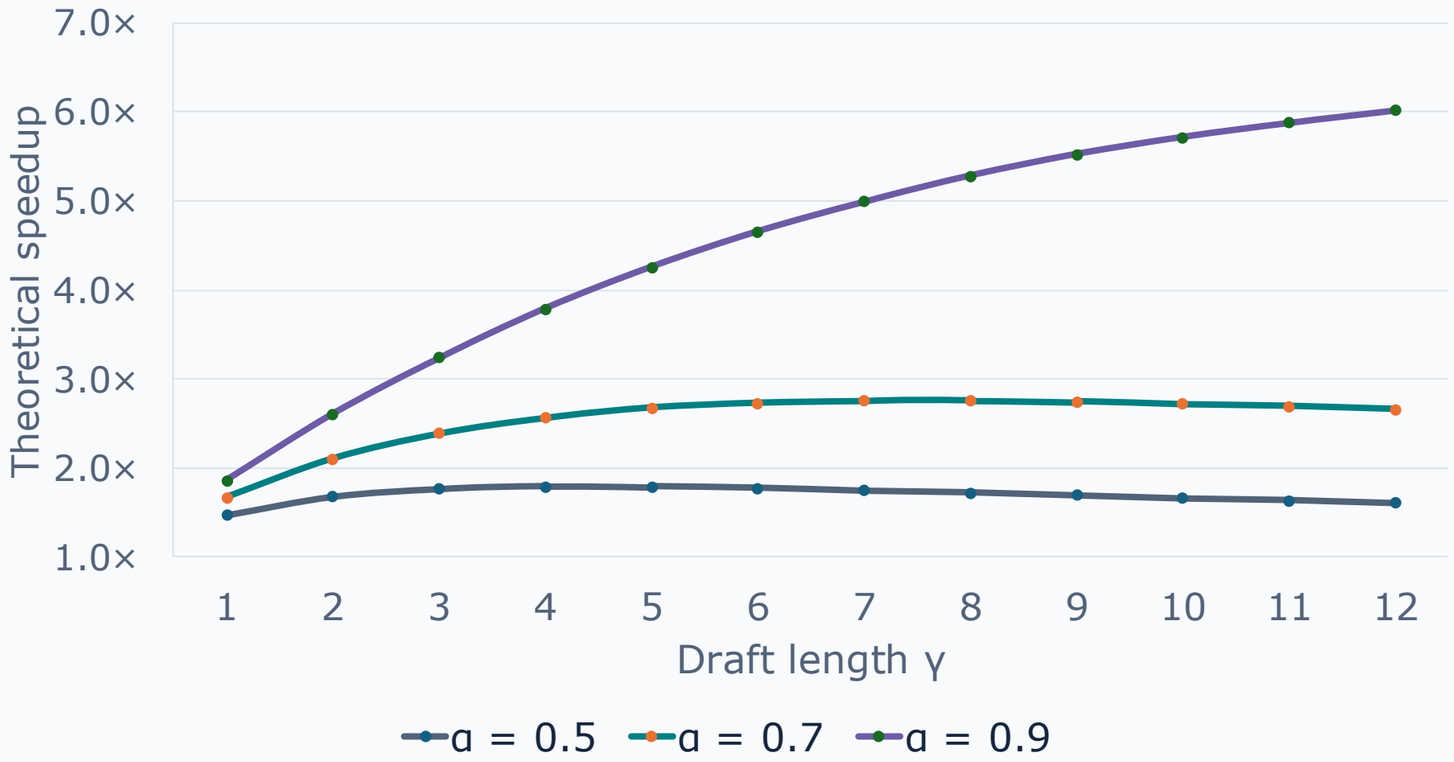
normalize(max(0, p − q))

Here: accept A or B with ⅔ probability.
All rejected mass (30%) returns to C.

Accepted mass (0.4, 0.2, 0.1) **+** **Correction mass** **(0, 0, 0.3) = p**

Paper §2.3 and Appendix A.1

The latency model



α Mean acceptance rate

c Draft-step / target-step time

E[tokens per iteration]

1 + α + ⋯ + α^γ

Expected speedup

E[tokens] / (1 + γc)

Illustrative model: c = 0.02. Longer drafts eventually give diminishing returns

Paper §3.1–3.5, Eq. (1), Theorem 3.8

Experimental setup

Measured walltime: T5 v1.1 checkpoints, batch size 1, one TPU v4

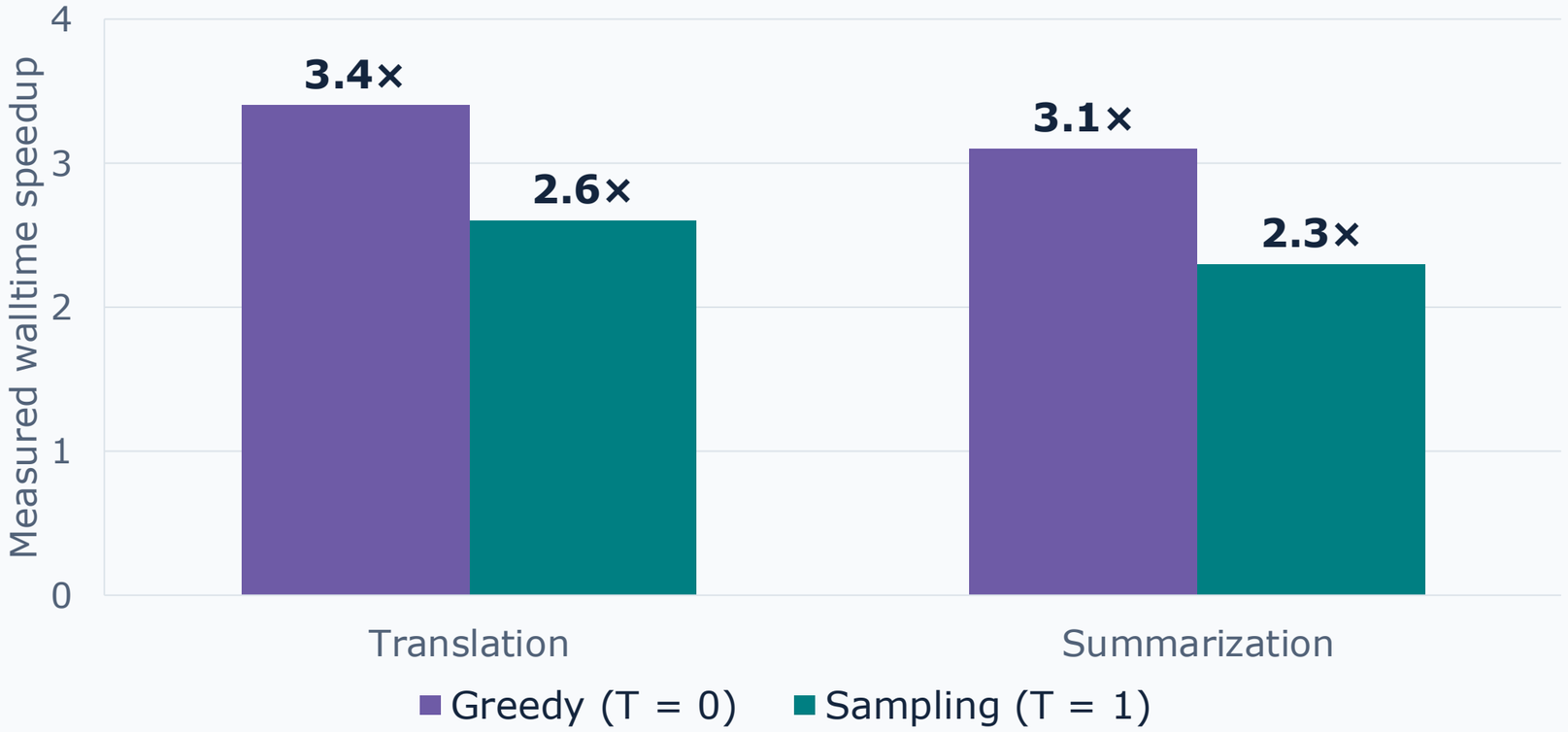


Task	Data	Decoding / baseline
English → German	WMT EnDe	T = 0, T = 1 / T5X
News summarization	CNN / Daily Mail	T = 0, T = 1 / T5X

Paper §4.1–4.2

T5-small gives measured speedups of 2.3× to 3.4×

Speedup over T5X on a single TPU v4, batch size 1



77M / 11B

Draft / target parameters

γ = 7

Translation

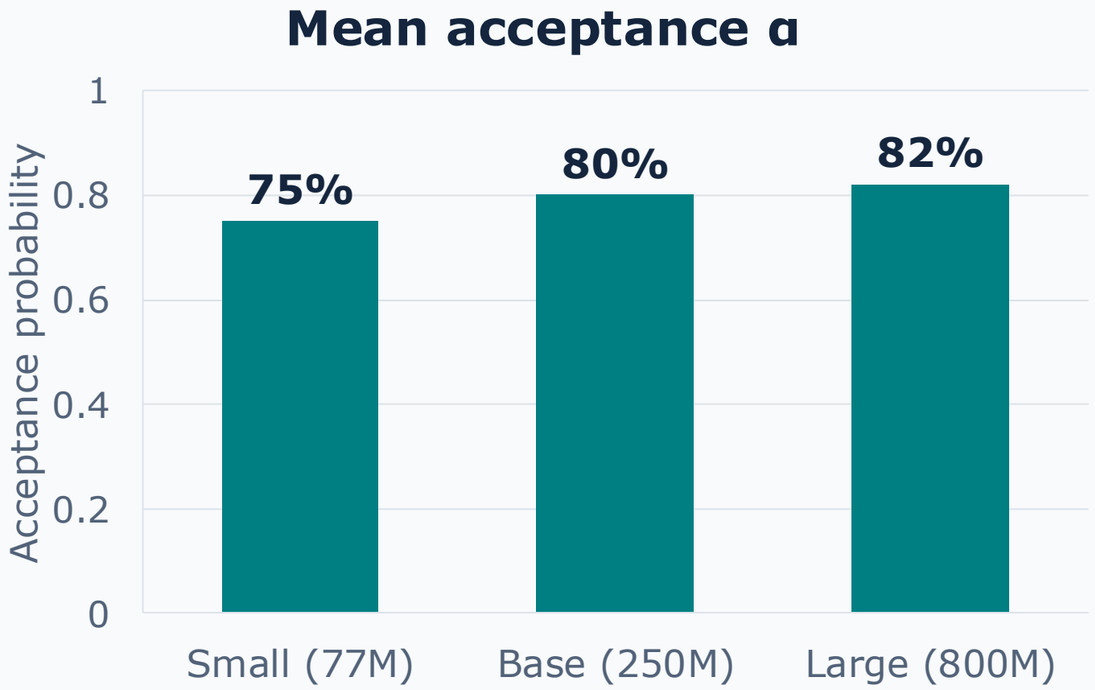
γ = 5

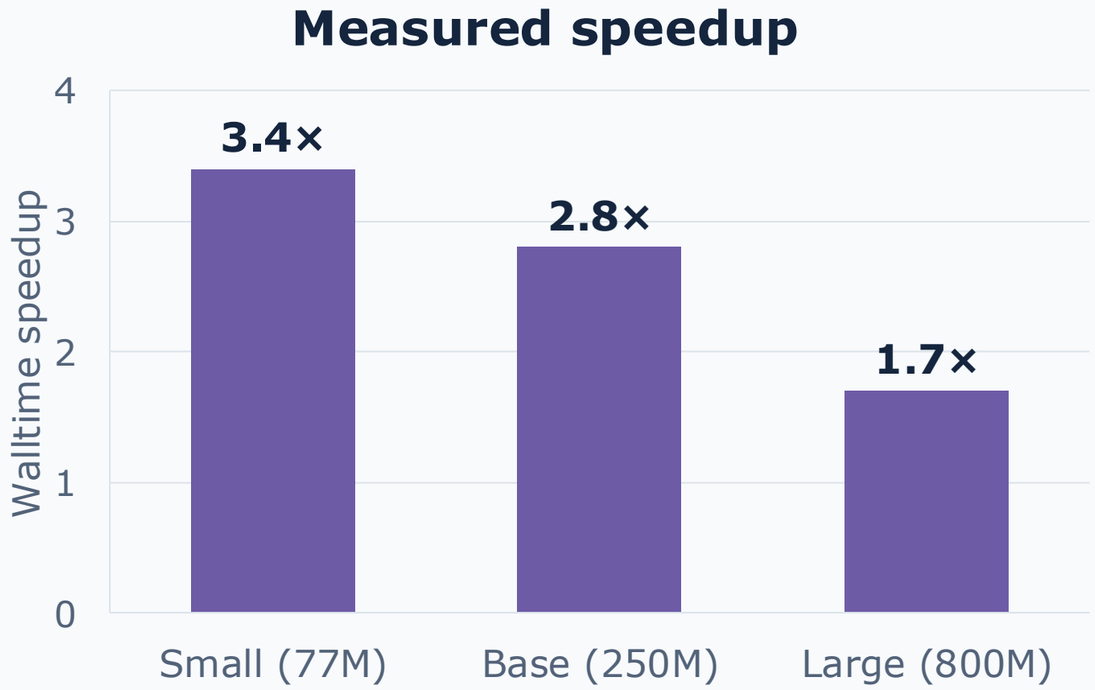
Summarization

Paper Table 2, T5-XXL target with T5-small drafter

Larger drafters can accept more yet run slower

Translation, T = 0, γ = 7 for each drafter



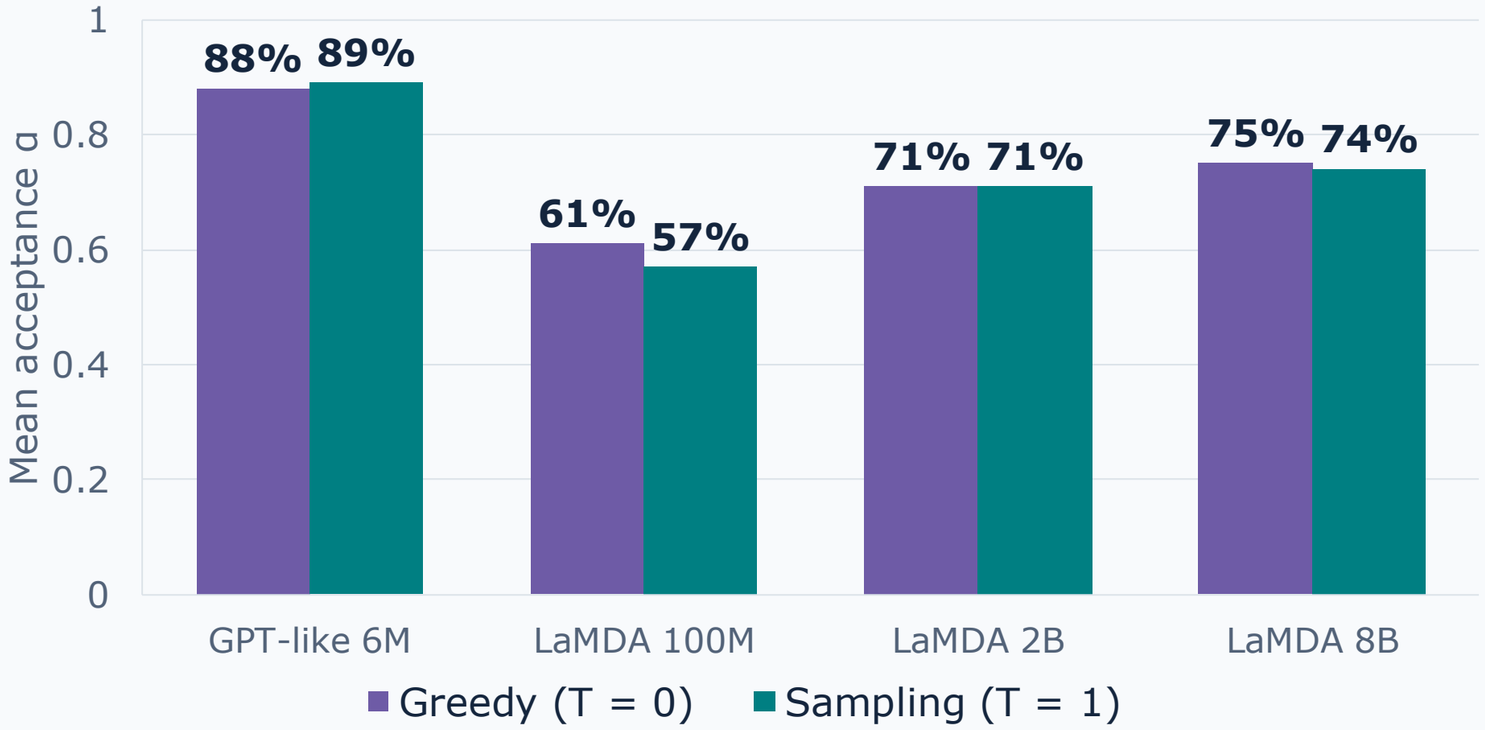


Profiled draft cost c rises from 0.02 to 0.04 to 0.11

Paper Table 2 and Appendix Table 4, EnDe greedy decoding

Acceptance estimates suggest broader potential

These are acceptance rates from 10K target-generated tokens, not measured walltime speedups



Targets

GPT-like: 97M

Unconditional generation on lm1b

LaMDA: 137B

Dialogue task

High acceptance is promising. Hardware cost still determines latency

Paper §4.2 and Table 3

Strengths and weaknesses

Strength	Limit or unresolved issue
Exact output distribution	Numerical implementation must match the sampling rule
No architecture change or retraining	A second model adds memory and integration work
Fewer serial target steps	More arithmetic can erase gains when compute is saturated
Measured acceleration against T5X	Walltime evidence covers batch size 1 on one TPU v4

Best fit: latency-sensitive decoding with spare parallel compute

Assessment based on Paper §3–4, §6 and Appendix A.3

Open questions and applications

Adaptive draft length

Can γ track confidence at each prefix without adding too much overhead?

Better approximation models

Can training directly for acceptance beat a larger generic drafter?

Serving at scale

How do batching, long contexts and shared compute change the gains?

Beyond these text tasks

Could exact speculation help image generation or model-driven simulation?

Immediate uses: interactive translation, summarization and dialogue

Paper §3.5–3.6, §6 and Appendix A.4–A.5. Discussion prompts