PACE: Pacing Branch Parallelism in LLM Serving

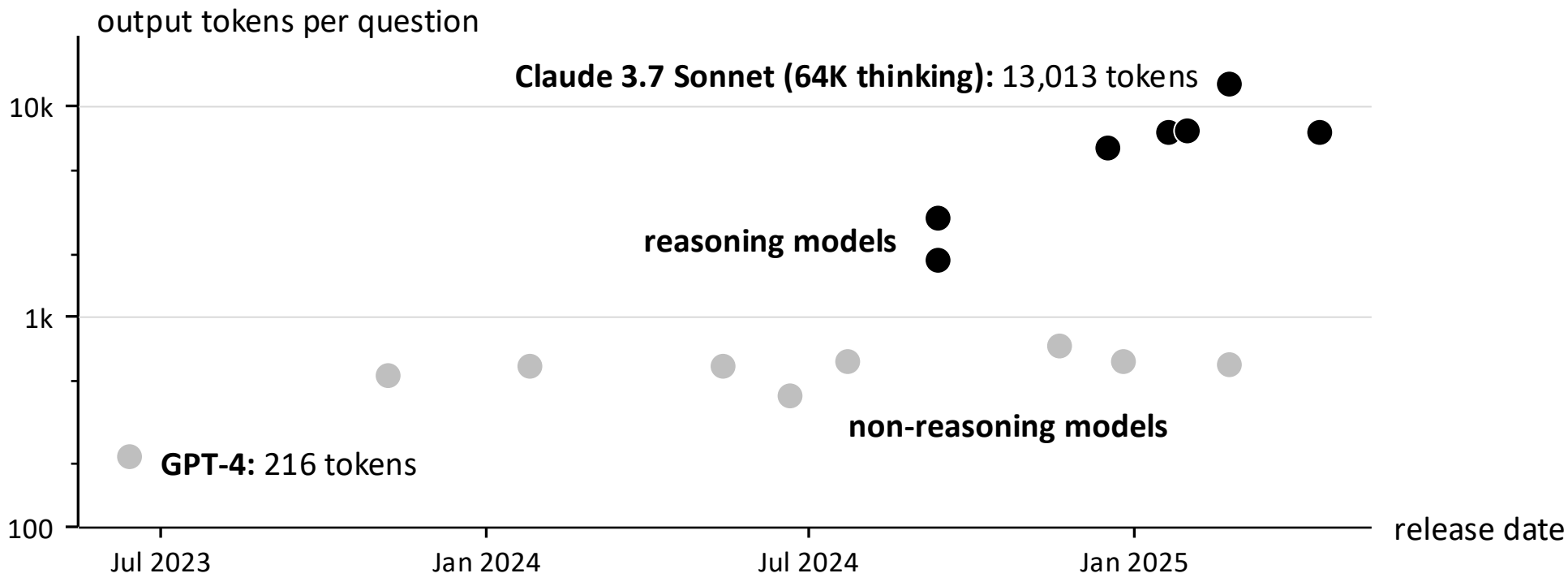
Swapnil Gandhi, Siva Kumar Sastry Hari,
William J. Dally, Christos Kozyrakis





Responses Are Growing **Longer**

Each dot is one model, on the same GPQA Diamond questions (log scale)



Decoding is sequential: latency ≈ **decode steps** × **step latency**

Epoch AI, “LLM responses to benchmark questions are getting longer over time” (2025) · thinking tokens included

Responses Have Inherent **Parallelism**

The highlighted parts of each response do not depend on one another

Chat ShareGPT

Give me three tips for a good night's sleep.

Here are three tips:

- **Keep a regular schedule.** Same bedtime, same wake-up time.
- **Avoid screens.** Put the phone away an hour before bed.
- **Mind the room.** Keep it cool, dark and quiet.

43.5% of requests decompose

Retrieval RAG-12K

What does each of the three retrieved reports conclude?

From the retrieved reports:

- **Report A** finds demand growing fastest in Asia.
- **Report B** expects prices to fall next year.
- **Report C** warns of supply shortages.

67.0% of requests decompose

Math reasoning Math-220K

Solve y = 3x + 2 for x = 5, x = 3, and x = 0.

Substitute each value of x:

- **x = 5:** y = 3(5) + 2 = 17
- **x = 3:** y = 3(3) + 2 = 11
- **x = 0:** y = 3(0) + 2 = 2

84.2% of requests decompose

Independent parts, yet decoded **one after another**

Illustrative examples. Percentages: proportion of decomposable requests in ShareGPT, RAG-12K and Math-220K.

Intra-Request **Parallelism** (IRP)

Prompt **Solve y = 3x + 2 for x = 5, x = 3, and x = 0. Show your steps.**

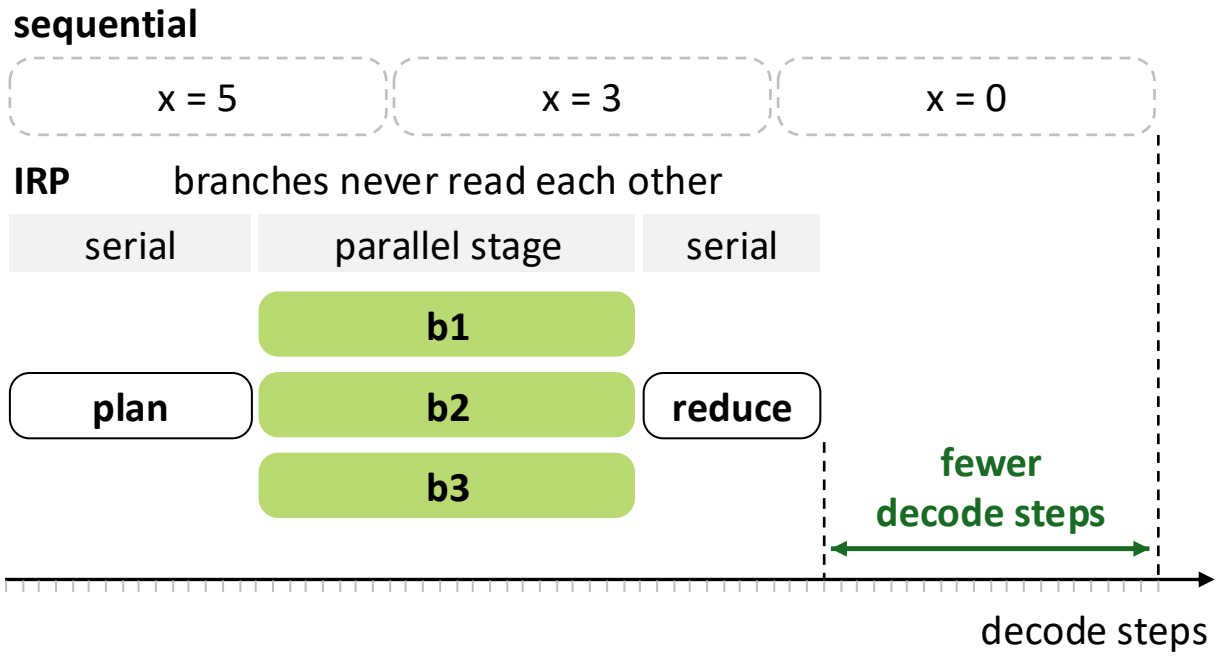
1. Solve x = 5 2. Solve x = 3 3. Solve x = 0

b1 Substitute x = 5:
y = 3(5) + 2 = 15 + 2 = 17
Thus, when x = 5, y = 17.

b2 Substitute x = 3:
y = 3(3) + 2 = 9 + 2 = 11
Thus, when x = 3, y = 11.

b3 Substitute x = 0:
y = 3(0) + 2 = 0 + 2 = 2
Thus, when x = 0, y = 2.

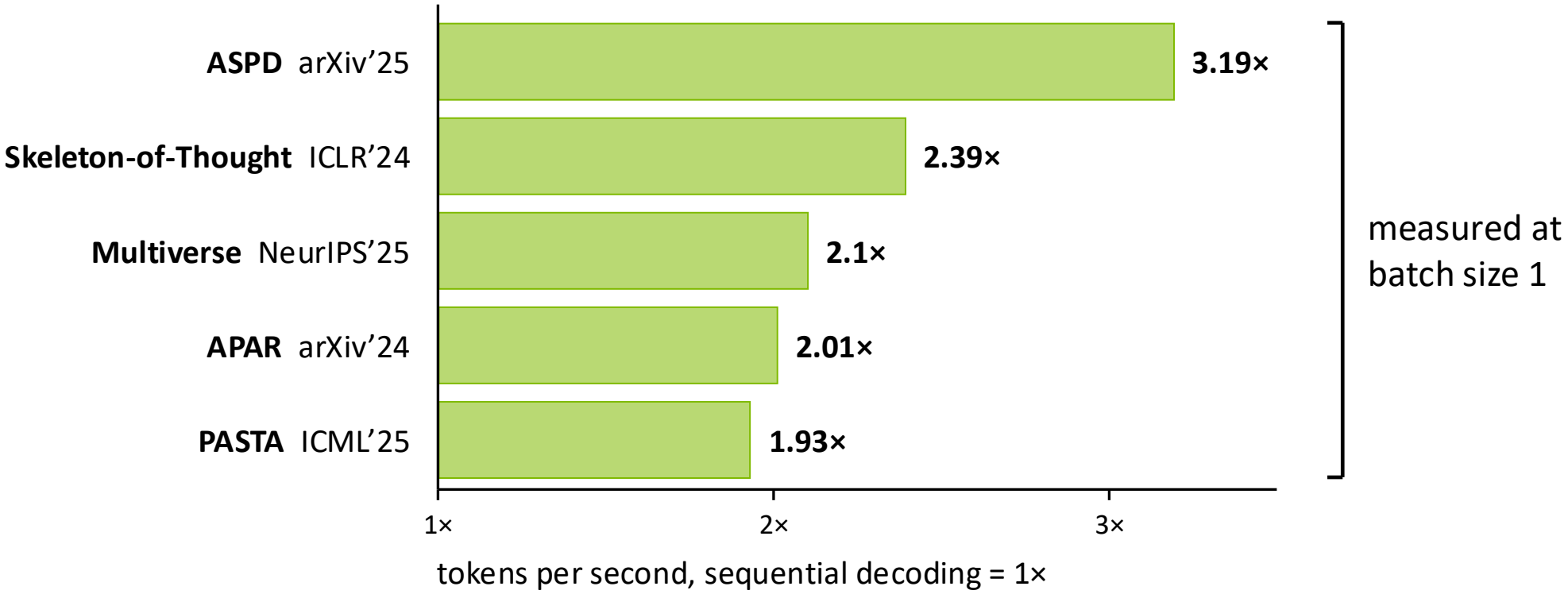
y = 17, 11, 2 for x = 5, 3, 0



IRP cuts **decode steps** to the longest serial path

IRP **Speeds Up** a Single Request

Each paper's reported speedup over sequential decoding

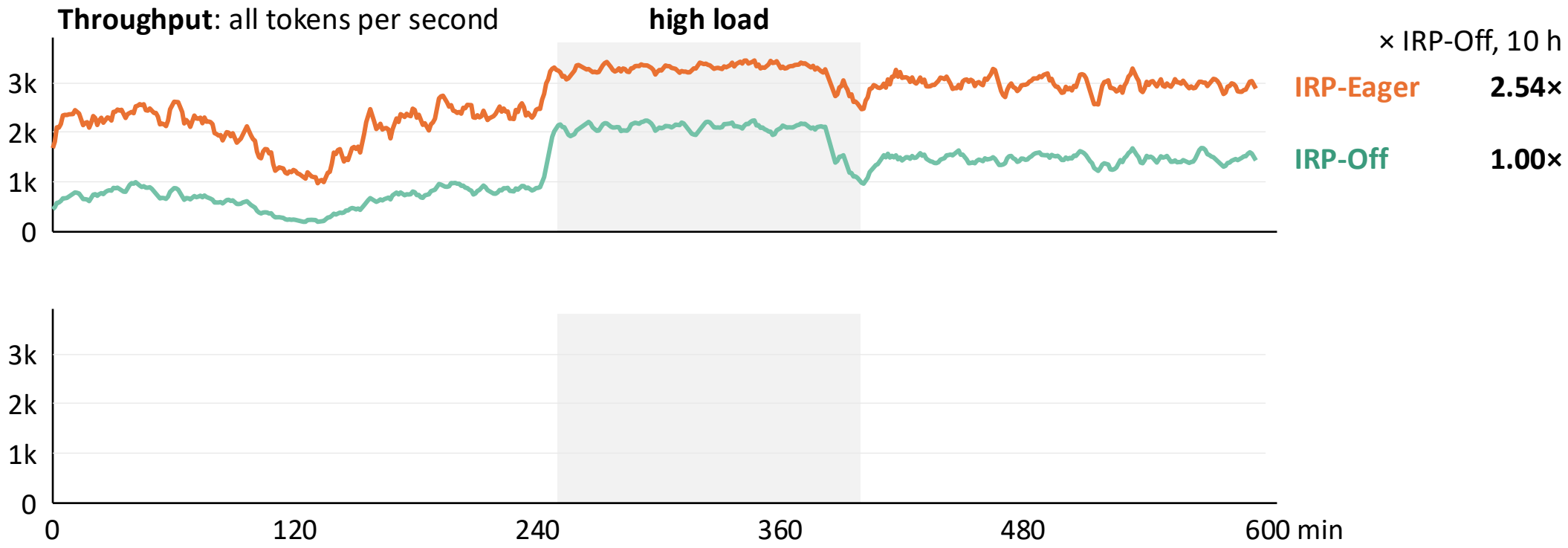


1.9–3.2× faster for the request that branches

ASPD: Chen et al. 2025 · Skeleton-of-Thought: Ning et al. ICLR 2024 · APAR: Liu et al. 2024
Multiverse: Yang et al. NeurIPS 2025 · PASTA: Jin et al. ICML 2025

The Throughput **Trap**

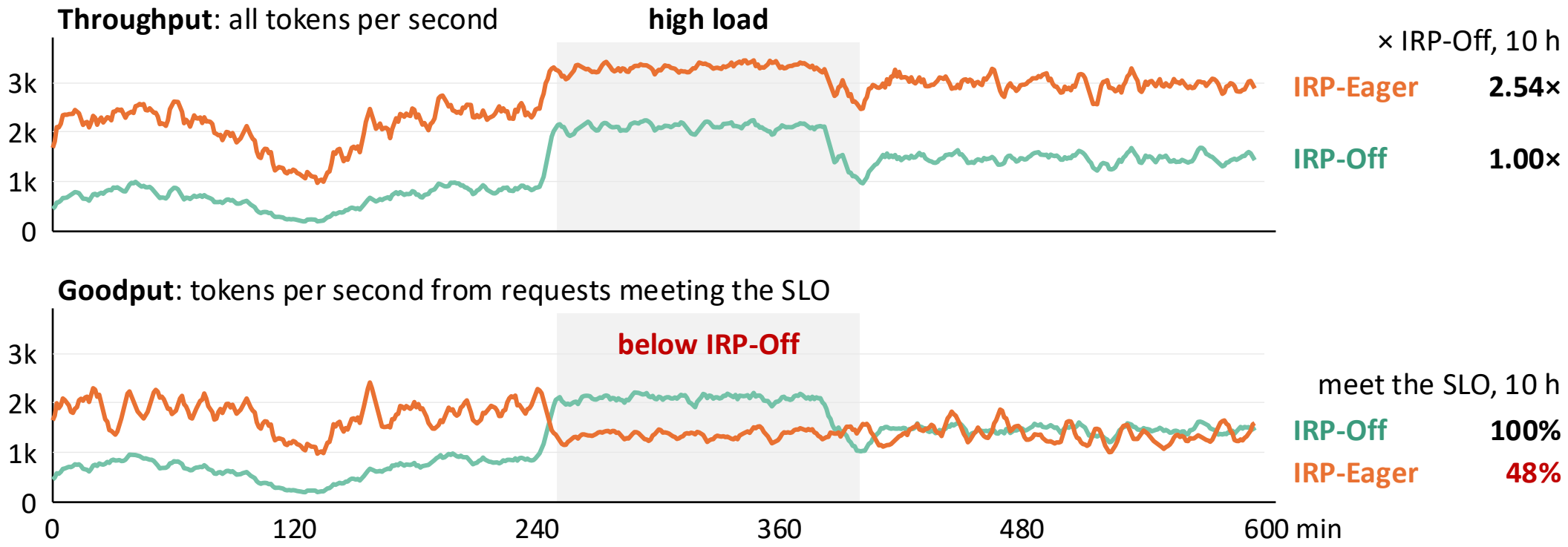
10-hour Azure-derived trace, half the requests branch, SLO: 50 ms per token



IRP-Eager runs every branch: **2.54× IRP-Off's throughput**

The Throughput **Trap**

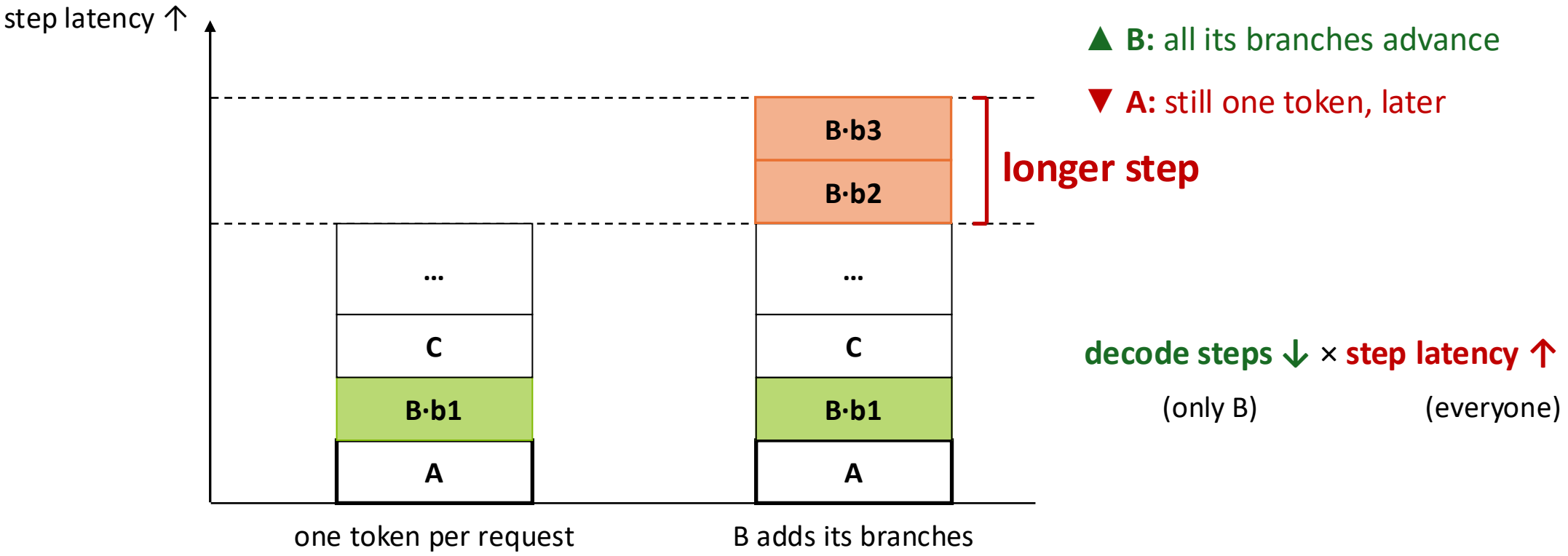
10-hour Azure-derived trace, half the requests branch, SLO: 50 ms per token



At high load, IRP-Eager's goodput falls **below IRP-Off's**

Branches Make Every Step **Longer**

Each decode step adds one token per sequence; more sequences, longer step



Every request pays **the cost**; only B gets extra progress

Practitioners can **prioritize** either…

	Never branch IRP-Off
Throughput	✗
SLO Attainment	✓

No speedup
from branches

Meets SLOs
at every load

All or Nothing

Practitioners can **prioritize** either…

	Never branch **IRP-Off**	**Always branch** **IRP-Eager**
Throughput	✗	✓
SLO Attainment	✓	✗
	No speedup from branches	Full speedup from branches
	Meets SLOs at every load	Misses SLOs at high load
	All or Nothing	All or Nothing

PACE prioritizes both

	Never branch IRP-Off	Always branch IRP-Eager	PACE
Throughput			
SLO Attainment			
	No speedup from branches	Full speedup from branches	Full speedup when there is slack
	Meets SLOs at every load	Misses SLOs at high load	Meets SLOs at every load
	All or Nothing	All or Nothing	Graceful Adaptation

PACE: A Simple Per-Step Planner

```
def PACEStep(requests, T, utility, ρ):

    step  = OneTokenEach(requests)
```

Per-Step Replanning
before every decode step

protected step

condensed from Algorithm 1 in the paper

PACE: A Simple Per-Step Planner

```
def PACEStep(requests, T, utility, ρ):
    step  = OneTokenEach(requests)
    slack = MostUrgentDeadline() - now() - T(step)
    limit = T(step) + ρ * max(0, slack)
```

Per-Step Replanning
before every decode step

protected step

Slack-Budgeted Admission
spend a fraction ρ = 0.8 of the slack

condensed from Algorithm 1 in the paper

PACE: A Simple Per-Step Planner

```
def PACEStep(requests, T, utility, ρ):
    step  = OneTokenEach(requests)
    slack = MostUrgentDeadline() - now() - T(step)
    limit = T(step) + ρ * max(0, slack)
    while some ready b has T(step + b) <= limit:
        step += the fitting b with most utility/ms
    return step
```

Per-Step Replanning
before every decode step

protected step

Slack-Budgeted Admission
spend a fraction ρ = 0.8 of the slack

predictor T: step latency
linear fit, 1.8% mean error

Branch-Level Control
the rest wait: no KV moved

One rule, every step: **the predicted step must fit the slack**

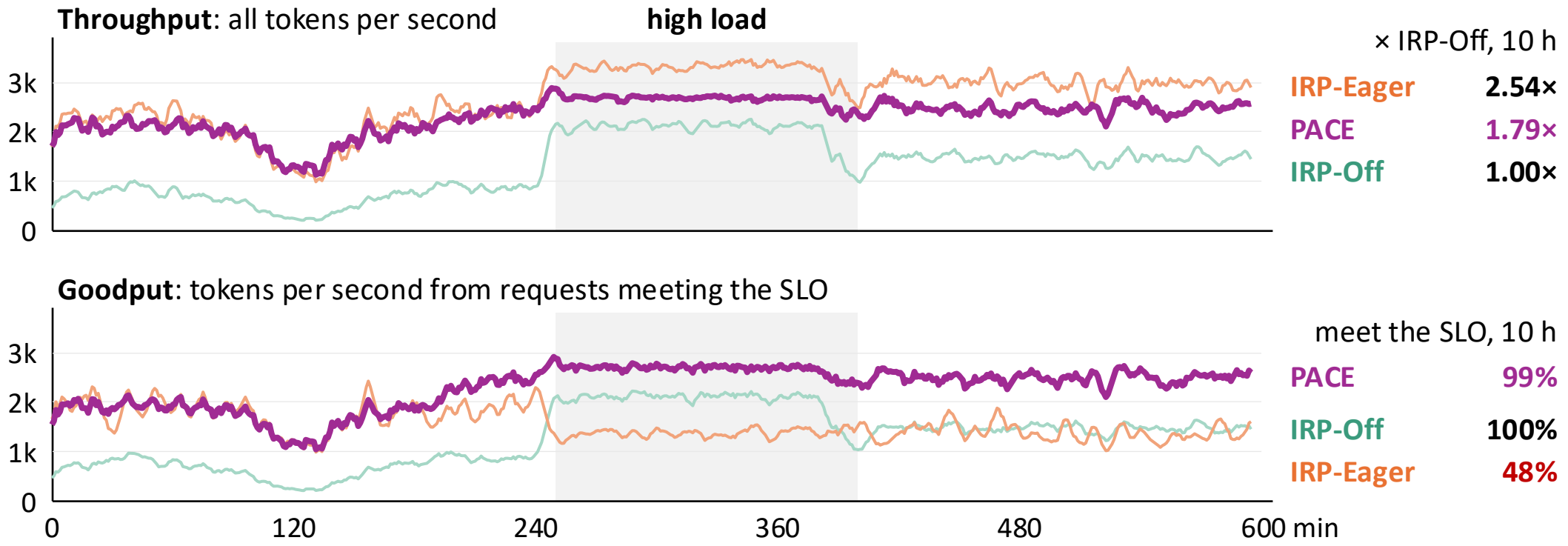
condensed from Algorithm 1 in the paper

PACE: Evaluation Setup

Model	Qwen3-32B; five more, 72B to 671B, for generality
Hardware	8×A100-80GB, served with SGLang
Workload	ShareGPT, RAG-12K, Math-220K; half the requests branch
Trace	10-hour Azure-derived trace: low, high, then moderate load
Baselines	IRP-Off (never branches), IRP-Eager (runs every ready branch)
Metrics	Goodput and SLO attainment at a 50 ms target; raw throughput

PACE Escapes the **Trap**

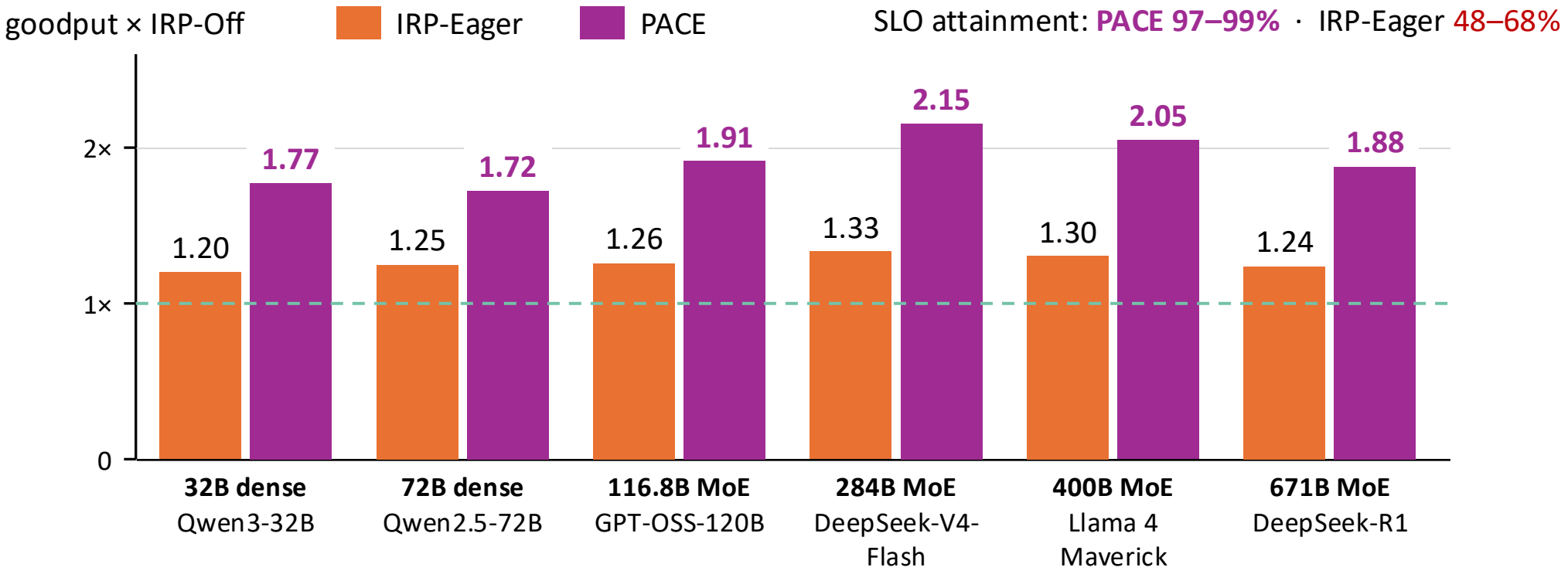
Same trace, same axes; PACE runs as a scheduling hook in SGLang



PACE: **1.77×** IRP-Off's goodput at **99%** SLO attainment

PACE Holds Across **Models**

Same configuration on dense and mixture-of-experts models, 32B to 671B



PACE: **1.72–2.15×** goodput on six models

PACE enables High-Throughput and SLO-Compliant Serving

The problem: branches lengthen every request's step

Slack-Budgeted Admission: admit only what the slack allows

Per-Step Replanning: re-decide at every decode step

Branch-Level Control: defer branches, not requests

IRP cuts **decode steps**; PACE spends only the slack on **step latency**

1.77× IRP-Off's goodput at **99%** SLO attainment

Code: github.com/Swapnil-Gandhi/PACE