WHEN DOES SCHEDULING MATTER? AN EMPIRICAL STUDY OF DISTRIBUTED LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Distributed large language model (LLM) inference requires coordinating parallel execution with the scheduling of requests that vary in arrival time, prompt length, and generation length. These decisions jointly shape throughput, time to first token, time per output token, and completion latency, yet their interaction remains insufficiently understood. We present a systematic study of parallelism and scheduling in distributed LLM inference using SGLang, spanning tensor, pipeline, and expert parallelism, five scheduling policies, diverse workloads, and full- and hybrid-attention models across model sizes and GPU counts. We find that scheduling sensitivity depends on the execution regime established by parallelism, model architecture, and workload. Median performance variation across policies ranges from 1.7% to 35.8% across model–scale groups, while variation across parallelism configurations is frequently larger. Policy gains can reverse across layouts, and configurations that maximize throughput can differ from those that minimize latency. Quantitative and qualitative analyses connect these differences to communication costs, state memory, effective concurrency, and queueing. Separating admission waiting from post-admission execution explains how scheduling affects time to first token, while greater queue pressure accompanies stronger policy effects across deployments. Across the 32 evaluated contexts, our staged selection strategy matches the recorded exhaustive optimum for all five latency objectives, with a maximum throughput loss of 3.84%, while reducing configuration evaluations by 26.7–48%, including screening.

1 INTRODUCTION

Large language models (LLMs) place substantial demands on accelerator memory and computation, making distributed execution essential for serving large models. Distributed training has developed a rich set of techniques for partitioning computation and model state, including pipeline parallelism, tensor parallelism, memory sharding, and automatic parallelization (Huang et al., 2019; Narayanan et al., 2021; Rajbhandari et al., 2020; Zheng et al., 2022). These techniques provide the foundations for distributed inference, whose performance also depends on how online requests share the available resources.

Inference introduces several sources of variability. Training frameworks such as Megatron-LM (Narayanan et al., 2021) typically repeat a planned computation over batches, with throughput as a central systems objective. In serving, requests arrive over time with heterogeneous prompt and generation lengths, continually changing the active batch and resident state. Inference engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) manage this evolving work through continuous batching and state management. Performance must be understood across four metric families: throughput, time to first token (TTFT), time per output token (TPOT), and end-to-end latency, and their preferred configurations can differ. Request scheduling adds another decision: which waiting requests should enter execution next? This choice changes admission waiting and the requests that execute together. Attention architecture further shapes this interaction. Full attention maintains a KV cache that grows with context length, while hybrid designs such as Qwen3.5 and Kimi Linear combine attention with recurrent state (Qwen Team, 2026c; Kimi Team, 2025). Their different state requirements change how much concurrent work a parallel layout can support as the workload changes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Recent serving systems and performance studies address selected dimensions of this problem. Sarathi-Serve studies batching and pipeline execution to improve the throughput–latency tradeoff (Agrawal et al., 2024b); DistServe separates prefill and decoding and allocates their resources according to latency objectives (Zhong et al., 2024). Vidur simulates inference configurations across parallelization, scheduling, and workloads (Agrawal et al., 2024a). However, these studies fail to provide a complete view of the joint design space, particularly how request-ordering policies interact with tensor, pipeline, and expert parallelism across full- and hybrid-attention models as workloads, model sizes, and GPU counts change. Understanding these combined effects requires paired policy comparisons within each deployment and measurements that connect aggregate performance to queueing, state capacity, and execution.

To that end, we present a comprehensive measurement study of this joint design space of scheduling and parallelism. Our experiments (using SGLang (Zheng et al., 2024)) span full- and hybrid-attention mixture-of-experts models, heterogeneous workloads and arrival intensities, diverse request-ordering policies, and deployments ranging from a single node to multiple nodes. We compare tensor, pipeline, expert, and mixed parallelism across model sizes and GPU budgets, measuring throughput, response time, token spacing, and completion latency, including tail behavior. By combining these measurements with communication, state-memory, and request-trace analyses, we investigate when scheduling matters, why its effects change across deployments, and how to select parallelism and policies for a serving objective. Our contributions are:

- **Characterizing the interplay between parallelism and scheduling.** We quantify how parallelism shapes the performance impact of scheduling policies and, conversely, how scheduling changes the relative performance of parallelism configurations across workloads, arrival intensities, architectures, and serving objectives. Parallelism frequently produces larger variation, while median policy spreads range from 1.7% to 35.8% across model–scale groups, revealing substantial differences in when scheduling matters.
- **Explaining the observed performance differences.** We provide quantitative and qualitative analyses linking scheduling sensitivity to queueing, batch composition, and parallel execution. Communication measurements, state-memory accounting, and request traces explain policy-dependent TTFT and the distinct parallelism preferences of the evaluated full- and hybrid-attention model families.
- **Developing and evaluating a staged configuration-selection strategy.** We use FCFS to shortlist parallelism layouts, then evaluate scheduling policies on the shortlist for the target objective. Across the evaluated settings, the two-layout shortlist retains the best layout under other policies in 96.9% of throughput and 98.4% of mean-TTFT comparisons, supporting its use to reduce configuration-search costs.

2 Related Work

Distributed Inference for LLMs. Tensor, pipeline, expert, and context parallelism distribute model state and computation across accelerators to support larger models and parallel execution (Narayanan et al., 2021; NVIDIA; Rajbhandari et al., 2022; Kimi Team, 2026; DeepSeek-AI, 2025b; Liu et al., 2023; Yang et al., 2025). These techniques enable larger models to fit within aggregate device memory while also exposing additional computational parallelism. Prefill–decode disaggregation separates the two phases onto different accelerators for independent optimization (Patel et al., 2024; Zhong et al., 2024; Qin et al., 2025; Jin et al., 2024).

Scheduling policies. Scheduling remains an active topic in LLM serving. Beyond FCFS, systems exploit prefix locality through policies such as LPM, while recent work explores size-aware scheduling using learned rankings or output-length estimates Fu et al. (2024); Zheng et al. (2026), locality-aware fair scheduling Cao et al. (2025), and tail-aware scheduling under burstiness and memory pressure Li et al. (2026). Together, these studies show that request ordering can substantially affect throughput, latency, fairness, and cache efficiency, making scheduling policy a key serving-system design choice.

Full Attention and Hybrid Attention. Conventional LLMs such as LLaMA and Qwen3 primarily rely on full softmax attention, including MHA and GQA variants (Grattafiori et al. (2024); Qwen Team (2025b)). Recent models increasingly adopt hybrid or compressed attention to reduce long-

context cost: Qwen3.5 combines Gated DeltaNet with softmax attention, Kimi Linear mixes Kimi Delta Attention with MLA, and recent GLM models incorporate sparse or hybrid attention (Qwen Team (2026b); Kimi Team (2025); GLM-5 Team (2026)). DeepSeek-V3 uses MLA to compress KV state, while newer DeepSeek designs further introduce sparse mechanisms such as CSA and HCA (DeepSeek-AI (2025b; 2026)). These architectures expose different memory, computation, and communication tradeoffs to inference systems.

3 BENCHMARK COMPOSITION AND MEASUREMENT METHODOLOGY

Given that existing studies leave scheduling and parallelism interaction incompletely characterized, our objective is to characterize distributed inference under various factors to better understand the interplay of parallelism and serving policy. We assemble existing trace workloads, open-source models, and serving configurations into a comprehensive measurement study. The selection spans parallel execution layouts, traces, arrival intensity, and attention architectures while retaining paired policy comparisons within each deployment.

3.1 MODELS AND PARALLEL EXECUTION

Model coverage. We study four mixture-of-experts (MoE) models spanning smaller and larger deployments. Appendix A.1, Table 5, gives their specifications and deployment coverage. Qwen3-30B-A3B and Qwen3-235B-A22B-Thinking-2507 use full attention throughout. Qwen3.5-35B-A3B and Qwen3.5-122B-A10B interleave three Gated DeltaNet blocks with one full-attention block (Qwen Team, 2025c;a; 2026d;a;c). We abbreviate these models as Q3-30B, Q3-235B, Q3.5-35B, and Q3.5-122B. The two smaller models Q3-30B and Q3.5-35B share a one-node four-GPU budget and a three-parallelism candidate set. The larger models extend the study to eight and sixteen GPUs and mixed parallelism across nodes. Deployment parameters use defaults or are adjusted to avoid out-of-memory (OOM) errors.

Parallelism definitions. Tensor parallelism (TP) shards within-layer computation and synchronizes partial results through collectives. Pipeline parallelism (PP) partitions layers into stages that exchange hidden states. In this study, expert parallelism (EP) specifically denotes the all-to-all DeepEP MoE path with data-parallel attention (DPA): attention processes disjoint request/token shards, dispatch sends tokens to expert owners, and combine returns expert outputs to the originating shards (SGLang Contributors, 2026; DeepSeek-AI, 2025a). We evaluate TP, PP, and EP on four GPUs, adding TP+PP and EP+PP on eight and sixteen GPUs, and Table 6 lists all layouts. Mixed layouts use PP between nodes. Appendix A.3 reports operation counts and measured communication regimes.

Serving stack and placement. The experiments use SGLang v0.5.9 (Zheng et al., 2024) on a cluster with four GPUs per node. The cluster is equipped with node-local NVLink and inter-node InfiniBand. For mixed TP/EP+PP layouts, its placement rules assign one four-GPU stage per node while TP8/TP16 and EP8/EP16 span nodes. Appendix A.1 reports hardware and software details.

3.2 WORKLOAD ASSEMBLY AND SCHEDULING POLICIES

Trace selection and arrival scales. We use Trace A, Trace B, Coder, and Thinking from the Qwen–Bailian trace repository[1] (Wang et al., 2025; Zhang et al., 2026) (Table 1). Arrival scale s multiplies trace interarrival times: $s = 1.0$ preserves the original timing, while $s = 0.6$ compresses it and increases the offered load. Appendix A.2 gives the sampling, metadata, and replay details.

Policy coverage. We compare five policy labels: FCFS, SJF, LOF, LPM, and RANDOM (Appendix A.2, Table 7). They cover arrival ordering, size-based priorities, prefix-locality preference, and a randomized reference. We assume the scheduler has perfect knowledge of each request's decoding length to investigate policy effects on inference. Appendix A.2 summarizes the waiting-queue implementation and its handling of retracted requests.

[1] https://github.com/alibaba-edu/qwen-bailian-usagetraces-anon

Table 1: Workload composition. Scale-0.6 interarrivals are derived by multiplying the scale-1.0 values by 0.6. Input/output entries are mean / median.

				Mean interarrival time (ms)	
Workload	Requests	Input tokens	Output tokens	Scale 1.0	Scale 0.6
Trace A	1,000	1,695 / 829	358 / 291	309.9	185.9
Trace B	1,000	832 / 708	88 / 49	40.2	24.1
Coder	1,000	5,364 / 4,370	881 / 490	225.0	135.0
Thinking	500	3,233 / 1,357	3,688 / 1,122	258.2	154.9

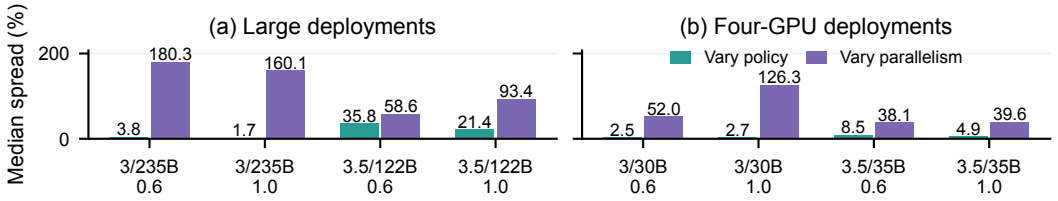


Figure 1: Median policy and parallelism variations, computed with Eq. 1. Each bar summarizes dimensionless within-metric comparisons.

3.3 Measurements and comparisons

Measurement grid and metrics. The selected grid contains $2(5 + 5 + 3 + 3) \times 4 \times 5 = 640$ complete result records, each with six objectives: token throughput, mean and p99 time to first token (TTFT), mean time per output token (TPOT), and mean and p99 end-to-end latency (E2EL).

Unitless comparisons. For a metric m and a controlled set of alternatives $\mathcal{A}$, define the observed spread

$$V_m(\mathcal{A}) = 100 \left(\frac{\max_{a \in \mathcal{A}} x_m(a)}{\min_{a \in \mathcal{A}} x_m(a)} - 1 \right). \tag{1}$$

Policy spread varies five policies while fixing model, scale, workload, parallelism, and metric. Parallelism spread varies the candidate parallelisms while fixing the other factors, including policy. We take medians after forming these dimensionless within-metric spreads. Each model/scale policy median covers $4 \times 6 \times C$ cells, where C is three or five parallelisms. Each parallelism median covers $4 \times 6 \times 5 = 120$ cells. For improvement plots, throughput is candidate/reference and latency is reference/candidate, so *values above one* are better.

4 The Interplay of Parallelism and Scheduling

Parallelism determines how requests execute, and scheduling determines which requests receive service. We first compare the performance variation induced by these two dimensions, then examine how policy effects depend on parallelism and how workload, serving objective, and arrival intensity shape layout preferences. These interactions motivate the staged selection procedure in Section 6.1.

4.1 Effect of Parallelism on Scheduling

Parallelism dominates aggregate variation, but policy effects can be large within a layout. Figure 1 shows that parallelism creates substantially more variation than scheduling for the full-attention models. At scale 0.6, Q3-235B has a median parallelism spread of 180.3%, compared with 3.8% for policy; Q3-30B shows the same ordering. Scheduling is more influential for the hybrid-attention models, reaching a 35.8% spread for Q3.5-122B. These patterns persist at scale 1.0.

The aggregate hides layout-specific opportunities. On Coder, Figure 2 shows that SJF under TP16 improves mean TTFT by 2.61× over FCFS with a 2.0% throughput loss; under TP4+PP4, all five SJF/FCFS ratios stay within 3.2% of one. Changing from TP16 to TP4+PP4 under FCFS yields a much larger mean-TTFT improvement (226×) and higher throughput, at the cost of slower TPOT.

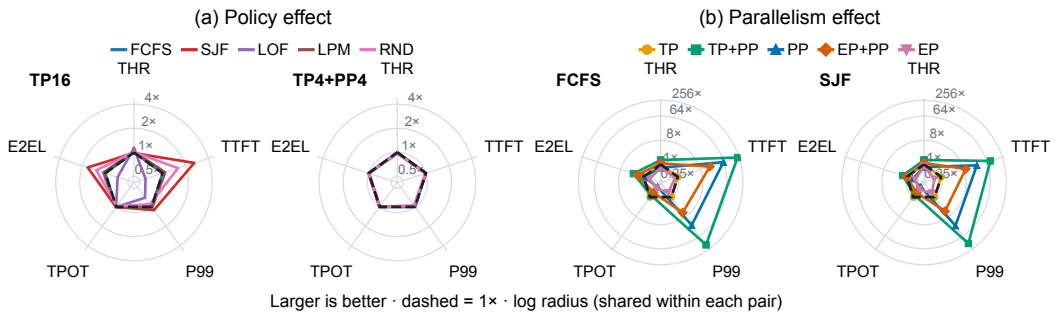


Figure 2: Q3-235B, Coder, scale 0.6, 16 GPUs. (a) All policies relative to same-parallelism FCFS; (b) all parallelisms relative to same-policy TP16. Ratios are candidate/reference for throughput (THR), inverted for latency. TTFT, TPOT, and E2EL are means; P99 is P99 TTFT; RND is RANDOM. Radial scales differ between pairs.

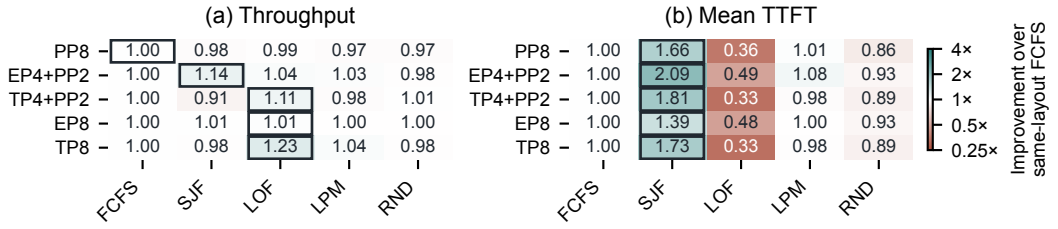


Figure 3: Q3.5-122B, Thinking, scale 0.6. Each row uses its own parallelism+FCFS reference, and values above one indicate improvement. Black outlines mark the best recorded policy within a row/objective. RND denotes RANDOM. Colors encode the same log-ratio scale in both panels. FCFS-normalized rows reveal policy sensitivity.

Parallelism determines the baseline regime, and scheduling determines the remaining opportunity within that regime.

4.2 Scheduling Effects Depend on Parallelism

A policy's effect changes with the parallelism layout. Figure 3 shows this interaction for Q3.5-122B on Thinking. Relative to FCFS, SJF increases throughput by 14.1% under EP4+PP2 but decreases it by 8.5% under TP4+PP2, while reducing mean TTFT in both. Under EP8, the highest policy throughput exceeds the lowest by only 1.5%, while LOF's mean TTFT is 2.88× SJF's. Scheduling can therefore reshape latency even when throughput remains stable, and its throughput effect can reverse across parallelisms.

4.3 Workload and Objective Shape the Preferred Parallelism

Workload and objective often determine the winning parallelism. For Q3-235B at scale 0.6, 21 of 24 workload–metric cells select the same parallelism under all five policies (Figure 4). TP4+PP4 wins every throughput comparison, while latency objectives select differently: on Trace A, every policy chooses TP4+PP4 for TTFT and TP16 for TPOT and E2EL; on Thinking, every policy chooses TP4+PP4 for mean E2EL and EP16 for P99 E2EL. The target objective therefore drives parallelism choice more strongly than policy in these cells.

4.4 Arrival Intensity Changes the Operating Regime

Higher offered load exposes workload-specific saturation. Figure 5 holds Q3-30B and TP4 fixed while compressing arrivals from scale 1.0 to 0.6, a 66.7% increase in offered rate. Trace A converts most of this load into throughput, with a 62.7% gain. Trace B is near saturation: throughput rises by only 4.6%, while mean TTFT grows by 4.78×. Coder and Thinking also gain limited

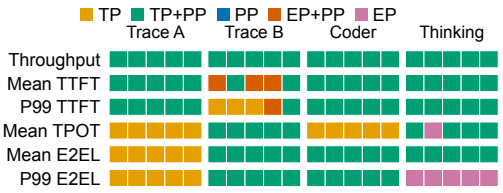


Figure 4: Q3-235B, scale 0.6: best parallelism for each workload and objective. Each workload–objective cell contains five equal mini-cells, ordered left to right by policy: FCFS, SJF, LOF, LPM, and RANDOM. Here TP, TP+PP, PP, EP+PP, and EP denote TP16, TP4+PP4, PP16, EP4+PP4, and EP16, respectively.

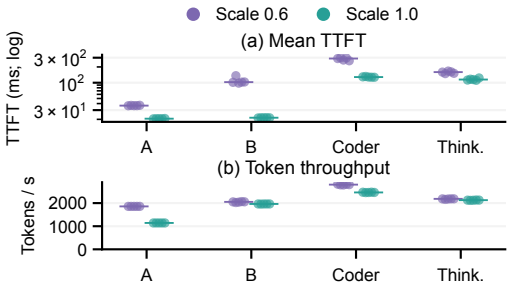


Figure 5: Q3-30B with TP4: each dot is one of five policy runs; short horizontal marks indicate their median. The spread of dots represents variation across policies.

throughput as TTFT rises. Arrival intensity therefore changes the balance between useful throughput and queueing for each workload.

5 Attention Architecture and Deployment Scale

We compare full- and hybrid-attention models at a common four-GPU scale, then track their parallelism preferences in larger deployments. Each model is evaluated as a complete architecture, including its attention, expert geometry, request limits, and kernels. We focus on two mechanisms behind the EP–TP tradeoff: per-request state and per-prompt execution cost.

5.1 Architecture-Dependent EP–TP Tradeoffs

EP–TP tradeoffs differ under a common GPU budget. Figure 11 in Appendix B.1 compares Q3-30B and Q3.5-35B under the same four-GPU budget. Q3.5-35B favors EP across more workloads and objectives. On Trace A, Q3-30B favors TP4 throughout, while Q3.5-35B gains throughput and mean E2EL with EP4. On Trace B, EP4 improves Q3-30B throughput but increases mean TTFT; for Q3.5-35B, it improves both by $1.40\times$ and $1.51\times$, respectively. On Thinking, Q3.5-35B reaches near-parity in throughput, with slower mean TTFT and faster mean E2EL under EP4. The preferred parallelism therefore depends on the architecture, workload, and objective.

State geometry determines the available concurrency. Let m_K be the full-model KV bytes per context token, R the recurrent state per request, and c a representative resident context length. With K KV heads and attention TP size t, the KV replication factor is $r = \max(1, t/K)$; EP has $r = 1$. TP increases per-request state by

$$\frac{rm_K c + R}{m_K c + R} = 1 + (r-1)\varphi_K, \qquad \varphi_K = \frac{c}{c + c_*}, \qquad c_* = \frac{R}{m_K}. \tag{2}$$

Here φ_K is the KV fraction of per-request state. Under balanced placement, active concurrency satisfies

$$b_\ell \le \min\left\{ B_{\text{req}}, \frac{NM}{r_\ell m_K c + R} \right\}, \tag{3}$$

where N is the GPU count, M is the residual state memory per GPU, B_{req} is the global request cap, and r_ℓ is the KV-replication factor of parallelism ℓ ($\max(1, t/K)$ for TP and one for EP). State savings increase concurrency when memory limits admission. Appendix B details the accounting assumptions.

Q3-30B has $K = N = 4$, so TP4 adds no KV replication over EP4; Q3.5-35B has $K = 2 < N = 4$, giving EP4 a state advantage. Equation 3 therefore predicts more room for concurrency when KV capacity limits admission. Figure 6 shows how this opportunity translates into observed occupancy. For Q3.5-35B on Trace B and Coder, EP4 sustains more running requests with slightly higher used-state shares, consistent with the throughput gains in Figure 11. On Thinking, EP4 has

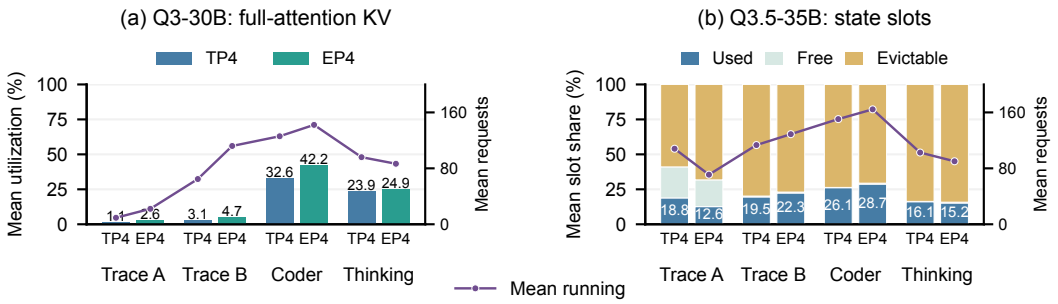


Figure 6: TP4 versus EP4 under FCFS at scale 0.6: mean (a) KV utilization for Q3-30B and (b) state-slot shares for Q3.5-35B. Lines show mean running requests (right axes).

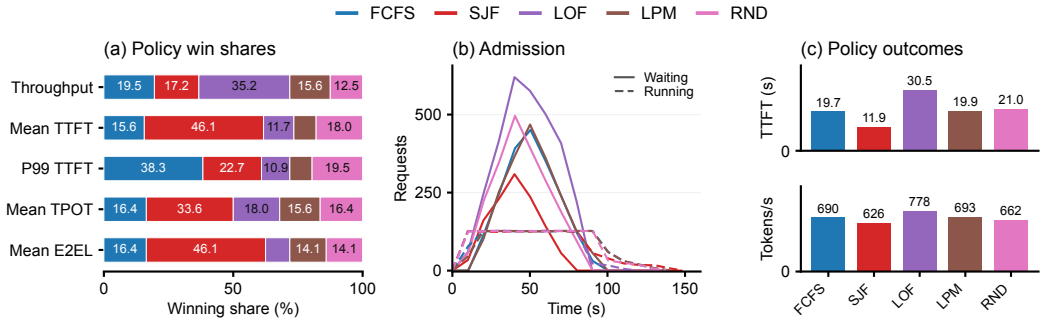


Figure 7: (a) Policy win shares across all models, workloads, arrival scales, and layouts. Each metric pools 128 equally weighted contests (2 scales $\times 4$ workloads $\times(5+5+3+3)$ layouts), comparing five policies per fixed deployment and workload. Throughput is maximized and latencies minimized; ties split one vote. (b–c) Q3.5-122B, TP8, Trace B, arrival scale 1.0. (b) Waiting requests (solid) and running requests (dashed), using rank 0's queue. (c) Mean TTFT above and token throughput below, with zero-based axes. RND denotes RANDOM.

lower occupancy and near-equal throughput. These patterns connect state capacity to workload-dependent concurrency, and realized occupancy also reflects request residency and retained cache.

State savings compete with per-prompt execution cost. TP distributes a prompt's attention across ranks; EP assigns it to one DPA owner. Under ideal scaling across e ranks, the isolated EP path adds approximately $(e-1)A_0^P$, where A_0^P is the TP reference attention time. At higher load, concurrent requests use multiple DPA owners, allowing EP's state-efficient concurrency to offset this per-prompt cost. Across the FCFS runs for four workloads and two arrival scales, EP4 improves throughput in 5/8 Q3-30B cases and 7/8 Q3.5-35B cases, but improves mean TTFT in 0/8 and 3/8 cases. EP therefore benefits throughput more consistently than TTFT when its concurrency gain competes with a longer prompt execution path.

5.2 Parallelism Preferences Across Deployment Scales

Architecture-family preferences persist at larger scales. Figure 12 in Appendix B.1 shows that EP-oriented layouts remain more common winners for hybrid-attention models across both arrival scales. EP4 wins over half of Q3.5-35B decisions, compared with one-third for Q3-30B. At larger scales, EP8 and EP4+PP2 together win at least 80% of Q3.5-122B decisions, while TP4+PP4 wins most Q3-235B decisions. These family-level patterns narrow the candidate layouts, and the final choice depends on the model, GPU count, and available configurations.

6 Co-Selecting Parallelism and Scheduling Policies

We turn the observed parallelism–policy interactions into a selection procedure. We first assess FCFS as a serving policy and as a baseline for screening layouts, then evaluate policy tuning on

Table 2: FCFS policy ranks. Top-one/two columns give rates over 128 fixed-parallelism contexts per objective.

Objective	Top one	Top two	Worst gap
Throughput	19.5%	41.4%	1.232×
Mean TTFT	15.6%	46.1%	2.889×
P99 TTFT	38.3%	57.0%	3.443×
Mean TPOT	16.4%	42.2%	1.329×
Mean E2EL	16.4%	47.7%	1.707×
P99 E2EL	23.4%	50.0%	1.271×

Table 3: FCFS-based parallelism screening. Top-one/two columns count coverage among 128 comparisons per objective.

Objective	Top one	Top two	Worst gap
Throughput	119/128	124/128	1.048×
Mean TTFT	109/128	126/128	1.342×
P99 TTFT	104/128	121/128	1.151×
Mean TPOT	115/128	128/128	1.000×
Mean E2EL	114/128	126/128	1.284×
P99 E2EL	114/128	128/128	1.000×

Both tables cover four models, four workloads, and both arrival scales. **Left**: FCFS is ranked among five policies at each evaluated parallelism, with the worst gap taken over its nonwinning contexts. **Right**: the FCFS shortlist is checked against each of four other policies ($4 \times 4 \times 2 \times 4 = 128$ comparisons), and the worst gap compares the better of the two shortlisted layouts under the evaluated policy with its optimal layout. Gaps are FCFS/best latency or best/FCFS throughput at the same parallelism (left), and selected/optimal latency or optimal/selected throughput under the same policy (right).

the resulting shortlist. We next compare policies across serving objectives and use queue traces to explain their latency–throughput tradeoffs.

6.1 FCFS as a Baseline for Staged Selection

FCFS is competitive for tail TTFT but usually loses on mean metrics. Table 2 shows that FCFS wins 19.5% of throughput and 15.6% of mean-TTFT contests. It leads the policy win shares for P99 TTFT at 38.3%, while its worst-case gap on that objective reaches $3.443\times$. Final policy selection therefore depends on the target objective.

FCFS provides an effective parallelism shortlist. To evaluate FCFS as a layout-screening baseline, we rank parallelisms within each model, workload, scale, and objective, then check whether its shortlist contains the best parallelism under each other policy (Table 3). Keeping the top two covers 124/128 throughput and 126/128 mean-TTFT comparisons, with complete coverage for mean TPOT and P99 E2EL. In contrast, top-one screening is less reliable for latency, with a worst mean-TTFT gap of $1.698\times$. We therefore use FCFS to prune parallelisms and evaluate the remaining policies on the shortlist.

Use FCFS screening to guide joint layout–policy selection. We propose a staged procedure: screen with FCFS, retain two layouts, and tune their policies. Across the 32 model–workload–scale contexts, FCFS screening yields smaller mean and worst performance losses than randomly retaining two layouts for every objective. Throughput loss averages 0.26% and reaches 3.84%, compared with 4.08% and 42.12% for random-two. For all five latency objectives, staged selection reaches the recorded exhaustive optimum in every context. Including FCFS screening and reusing its results, staged selection reduces profiling runs by 26.7% for three layouts and 48% for five compared with exhaustive search. Appendix A.5 reports the complete evaluation and profiling costs.

6.2 Scheduling Tradeoffs Across Serving Objectives

The preferred policy depends on the serving objective. Figure 7(a) summarizes how often each policy wins each objective. LOF leads throughput with a 35.2% win share. SJF leads mean TTFT and mean E2EL with 46.1% each, and mean TPOT with 33.6%. FCFS leads P99 TTFT with 38.3%. LOF wins only 9.4% of mean-E2EL contests. The distinct leaders across objectives motivate choosing policies according to throughput, mean latency, or tail latency requirements.

Policy changes admission dynamics and couples latency with throughput. For Q3.5-122B/TP8 on Trace B, waiting-queue peaks range from 309 under SJF to 620 under LOF (Figure 7(b–c)). Prioritizing longer decodes, LOF extends residency and delays admission. Relative to FCFS, SJF lowers mean TTFT by 39.7% and throughput by 9.4%; LOF raises them by 55.1% and 12.7%, respectively. LOF also yields fewer waits over 10 seconds but a higher mean queue time (Appendix C, Figure 13).

Table 4: Spearman correlations between queue pressure and mean-TTFT policy spread across 110 configurations, with 95% confidence intervals. Adjusted rows account for model, workload, parallelism family, and arrival scale; bootstrap and measurement details appear in Appendix F.

Association	Correlation	95% interval
Queue share, five-policy spread	0.819	$[0.593, 0.890]$
Adjusted queue share, five-policy spread	0.455	$[0.044, 0.690]$
Queue share, four-policy spread excluding FCFS	0.807	$[0.555, 0.884]$
Adjusted queue share, four-policy spread	0.404	$[-0.032, 0.673]$
Mean queue seconds, five-policy spread	0.817	$[0.587, 0.883]$

Similar running-request peaks (127–128) across policies highlight the effects of admission order and residency beyond peak occupancy.

6.3 Queueing and Post-Admission Execution

We decompose TTFT into admission waiting Q and post-admission delay P. Subscript 0 denotes FCFS on the same parallelism. For matched requests,

$$\bar{T}_\ell = \bar{Q}_\ell + \bar{P}_\ell, \qquad \bar{T}_\ell < \bar{T}_0 \iff \bar{Q}_0 - \bar{Q}_\ell > \bar{P}_\ell - \bar{P}_0. \tag{4}$$

Reducing queueing therefore improves mean TTFT whenever it exceeds any added post-admission delay. With $R_T = \bar{T}_\ell/\bar{T}_0$, $R_P = \bar{P}_\ell/\bar{P}_0$, $R_Q = \bar{Q}_\ell/\bar{Q}_0$, and positive reference means,

$$R_T = (1 - w_Q)R_P + w_Q R_Q, \qquad w_Q = \frac{\bar{Q}_0}{\bar{T}_0}, \tag{5}$$

where w_Q is FCFS queueing time as a fraction of TTFT. At low w_Q, post-admission execution dominates; at high w_Q, queueing dominates. Appendix B.3 covers the zero-queue case.

Queueing explains the policy-dependent TTFT. For the Q3.5-122B/TP8 Trace B case at scale 1.0 in Section 6.2, admission waiting accounts for 99.1% of FCFS mean TTFT. SJF saves 7.83 s in queueing while adding only 0.03 s post admission; LOF's TTFT increase likewise comes almost entirely from queueing. Scheduling can also change P through batch composition, utilization, and prefill–decode interference, and size-aware policies can trade lower mean latency for longer waits among large requests. Appendix C reports the full decomposition for all five policies.

Queue pressure across configurations. We examine whether FCFS queue share, the fraction of TTFT spent waiting for admission, is associated with mean-TTFT variation across scheduling policies. Across 110 configurations, higher queue share accompanies greater scheduling sensitivity (Table 4). The pooled Spearman correlation, a rank-based measure of monotonic association, is 0.819 and remains 0.455 after adjusting for model, workload, parallelism family, and arrival scale. This association links admission pressure to the scope for policy-dependent TTFT changes. Appendix F details the measurements and robustness checks.

7 Summary of key findings

Layout sensitivity usually exceeds median policy sensitivity (1.7%–35.8%, Section 4.1), and policy gains can reverse across layouts (Section 4.2). State savings expand memory-limited concurrency (Section 5.1). Queue pressure tracks TTFT sensitivity (Section 6.3). FCFS's top-two shortlist preserves the best layout under other policies in 96.9% of throughput and 98.4% of mean-TTFT comparisons (Section 6.1).

8 Conclusion

Through matched comparisons of layouts and policies, this study shows how workload and attention architecture influence distributed inference performance. The analysis of state occupancy, admission waiting, and post-admission execution explains why scheduling benefits vary across configurations. These findings support a smaller joint search: shortlist two layouts with FCFS, then tune their policies for the serving objective.

AI USE STATEMENT

We use generative AI tools to assist with manuscript revision, editing the discussion of experimental results, checking numerical consistency, and LaTeX formatting. We take full responsibility for the scientific claims, reported results, and final content of this work.

REPRODUCIBILITY STATEMENT

Section 3 specifies the models, workloads, parallelism layouts, scheduling policies, and comparison metrics. Appendix A.1 records the hardware and software versions, model configurations, and request limits; Appendix A.2 describes trace sampling, prompt reconstruction, arrival scaling, and policy implementation. Appendix A.3 documents metric collection and communication measurements. The state-accounting assumptions and TTFT derivation appear in Appendix B, with supporting queue measurements, validation procedures, capacity comparisons, and statistical checks in Appendices C–F. Appendix G provides the policy and parallelism comparisons across all evaluated models, workloads, and arrival scales. The trace repository and model sources are cited in the experimental setup.

REFERENCES

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 351–366, 2024a. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/b74a8de47d2b3c928360e0a011f48351-Paper-Conference.pdf.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, Santa Clara, CA, July 2024b. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Shiyi Cao, Yichuan Wang, Ziming Mao, Pin-Lun Hsu, Liangsheng Yin, Tian Xia, Dacheng Li, Shu Liu, Yineng Zhang, Yang Zhou, Ying Sheng, Joseph Gonzalez, and Ion Stoica. Locality-aware fair scheduling in llm serving, 2025. URL https://arxiv.org/abs/2501.14312.

DeepSeek-AI. DeepEP: An efficient expert-parallel communication library, 2025a. URL https://github.com/deepseek-ai/DeepEP. Accessed September 10, 2026.

DeepSeek-AI. DeepSeek-V3 technical report, 2025b. https://doi.org/10.48550/arXiv.2412.19437.

DeepSeek-AI. Deepseek-v4 technical report. 2026. URL https://www.deepseek.com/en/news/v4-preview/.

Yichao Fu, Siqi Zhu, Runlong Su, Aurick Qiao, Ion Stoica, and Hao Zhang. Efficient LLM scheduling by learning to rank. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/6c8985579293e0209bdaa4f21bb1d237-Abstract-Conference.html.

GLM-5 Team. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan,

Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jen-

nifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf.

Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, Tiandeng Wu, Xing Chu, Ruizhi Huan, Li Ma, Xiao You, Wenting Zhou, Yunpeng Ye, Wen Liu, Xiangkun Xu, Yongsheng Zhang, Tiantian Dong, Jiawei Zhu, Zhe Wang, Xijian Ju, Jianxun Song, Haoliang Cheng, Xiaojing Li, Jiandong Ding, Hefei Guo, and Zhengyong Zhang. P/d-serve: Serving disaggregated large language model at scale, 2024. URL https://arxiv.org/abs/2408.08147.

Kimi Team. Kimi linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Kimi Team. Kimi k2: Open agentic intelligence, 2026. URL https://arxiv.org/abs/2507.20534.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Yueying Li, Yuanfan Chen, Jiayang Chen, Esha Choukse, Haoran Qiu, G. Edward Suh, Rodrigo Fonseca, Ziv Scully, and Udit Gupta. Beyond prediction: Tail-aware scheduling for llm inference, 2026. URL https://arxiv.org/abs/2606.18431.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023. URL https://arxiv.org/abs/2310.01889.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. URL https://doi.org/10.1145/3458817.3476209.

NVIDIA. TensorRT. URL https://github.com/nvidia/tensorrt. Accessed September 10, 2026.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*, pp. 118–132. IEEE, 2024. doi: 10.1109/ISCA59077.2024.00019. URL https://ieeexplore.ieee.org/document/10609649.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for llm serving, 2025. URL https://arxiv.org/abs/2407.00079.

Qwen Team. Qwen3-235B-A22B-Thinking-2507: Model configuration, 2025a. URL https://huggingface.co/Qwen/Qwen3-235B-A22B-Thinking-2507/blob/main/config.json. Accessed September 10, 2026.

Qwen Team. Qwen3 technical report, 2025b. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3-30B-A3B: Model configuration, 2025c. URL https://huggingface.co/Qwen/Qwen3-30B-A3B/blob/main/config.json. Accessed September 10, 2026.

Qwen Team. Qwen3.5-122B-A10B: Model configuration, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-122B-A10B/blob/main/config.json. Accessed September 10, 2026.

Qwen Team. Qwen3.5, 2026b. URL https://qwen.ai/blog?id=qwen3.5. Accessed September 10, 2026.

Qwen Team. Qwen3.5-35B-A3B: Model card, 2026c. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B. Accessed September 10, 2026.

Qwen Team. Qwen3.5-35B-A3B: Model configuration, 2026d. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B/blob/main/config.json. Accessed September 10, 2026.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL https://arxiv.org/abs/1910.02054.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of Machine Learning Research, pp. 18332–18346. PMLR, 2022. URL https://arxiv.org/abs/2201.05596.

SGLang Contributors. DP, DPA and SGLang DP router, 2026. URL https://docs.sglang.io/docs/advanced_features/dp_dpa_smg_guide. Accessed September 10, 2026;.

Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. Kvcache cache in the wild: Characterizing and optimizing kvcache cache at a large cloud provider. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, July 2025. URL https://www.usenix.org/conference/atc25/presentation/wang-jiahao.

Amy Yang, Jingyi Yang, Aya Ibrahim, Xinfeng Xie, Bangsheng Tang, Grigory Sizov, Jongsoo Park, and Jianyu Huang. Context parallelism for scalable million-token inference. In Matei Zaharia, Gauri Joshi, and Yingyan (Celine) Lin (eds.), *Proceedings of the Eighth Conference on Machine Learning and Systems, Mlsys 2025, Santa Clara, CA, USA, May 12-15, 2025*. OpenReview.net/mlsys.org, 2025. URL https://arxiv.org/abs/2411.01783.

Dingyan Zhang, Jinbo Han, Kaixi Zhang, Xingda Wei, Sijie Shen, Chenguang Fang, Wenyuan Yu, Jingren Zhou, and Rong Chen. Simple is better: Multiplication may be all you need for llm request scheduling, 2026. URL https://arxiv.org/abs/2603.15202.

Haoyu Zheng, Yongqiang Zhang, Fangcheng Fu, Xiaokai Zhou, Hao Luo, Hongchao Zhu, Yuanyuan Zhu, Hao Wang, Xiao Yan, and Jiawei Jiang. Scheduling llm inference with uncertainty-aware output length predictions, 2026. URL https://arxiv.org/abs/2604.00499.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 559–578, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2024. URL https://arxiv.org/abs/2312.07104.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In Ada Gavrilovska and Douglas B. Terry (eds.), *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, pp. 193–210. USENIX Association, 2024. URL https://arxiv.org/abs/2401.09670.

A Experimental Details

A.1 Model Architectures and Deployment Configuration

Table 5 collects the model specifications, deployment budgets, and state coefficients used throughout the study. Table 6 records the configured and effective request limits for each layout.

Table 5: Model architecture, deployment coverage, and live-state coefficients. Parameters are total/active billions; layer counts refer to the main text decoder. State coefficients assume BF16 and unreplicated full-model storage; Appendix B.1 derives m_K and R. The TP-only replication factor is $r_0 = \max(1, N/K)$. KiB and MiB use binary units.

	Q3-30B	Q3.5-35B	Q3.5-122B	Q3-235B
Parameters (B)	30/3	35/3	122/10	235/22
Full / linear layers (L_F/L_L)	48/0	10/30	12/36	94/0
Hidden dimension	2048	2048	3072	4096
Full-attention KV heads (K)	4	2	2	4
Routed experts / top-k	128/8	256/8	256/8	128/8
GPUs (N) / nodes	4/1	4/1	8/2	16/4
Candidate parallelism layouts	3	3	5	5
Runs in the study	120	120	200	200
TP-only KV replication (r_0)	1	2	4	4
KV state m_K (KiB/token)	96	20	24	188
Recurrent state R (MiB/request)	0	30	72	0

Serving stack and placement. Experiments run on a supercomputer whose compute nodes use Intel Xeon Platinum 8468 processors and NVIDIA HGX H100 platforms with four SXM GPUs per node. Nodes are interconnected by 400-Gbps InfiniBand NDR. The software stack comprises SGLang v0.5.9, CUDA 12.8, and NCCL 2.26.2. We select DeepGEMM (DeepSeek-AI (2025b)) for the Qwen DeepEP path and disable custom AllReduce when TP degree exceeds four. Thus a parallelism change is a change of deployed execution strategy, including its compute/communication backend.

Table 6: Configured request limits and effective limits under different parallelism layouts. Microbatch sizes are shown for pipeline-parallel runs.

Model	Configured limit	Parallelism Layout	Effective limit
Q3-30B	`auto`	TP4	Engine-derived
		PP4	Engine-derived
		EP4(DPA4)	Engine-derived
Q3.5-35B	384	TP4	384
		PP4	384 (microbatch 96)
		EP4(DPA4)	384 (each DPA 96)
Q3-235B	`auto`	TP16	Engine-derived
		EP16(DPA16)	Engine-derived
		TP4+PP4	Engine-derived
		EP4(DPA4)+PP4	Engine-derived
		PP16	Engine-derived
Q3.5-122B	128	TP8	128
		PP8	128 (microbatch 16)
		TP4+PP2	128 (microbatch 64)
		EP8(DPA8)	128 (each DPA 16)
		EP4(DPA4)+PP2	128 (each DPA 32, microbatch 16)

A.2 Workload Replay and Scheduling Implementation

Trace sampling and prompt reconstruction. The Qwen–Bailian repository releases anonymized production traces with request-relative timestamps, input/output lengths, chat and parent-chat identifiers, turn information, request types, and hashed prompt blocks (Wang et al., 2025; Zhang et al., 2026). Prompt tokens are grouped into 16-token blocks, hashed with salted SipHash, and remapped to integer identifiers. For replay, we map each hash identifier to a distinct synthetic token block and reuse that mapping across requests, retaining block-level prefix-reuse patterns across conversation turns. Each run starts at trace index zero and uses the same workload prefix: 1,000 requests for Trace A, Trace B, and Coder, and 500 for Thinking.

Output-length control. Replay supplies each request's trace output length as its output-token budget and disables end-of-sequence stopping with `ignore_eos=True`. SJF and LOF receive the trace-derived output budget as their size signal.

Arrival-time scaling. All workloads use arrival scales $s \in \{0.6, 1.0\}$. For trace-relative arrival times t_i, replay schedules requests at $s(t_i - t_0)$, preserving request order and input/output lengths. Scale $s = 1.0$ retains the original timing; $s = 0.6$ places arrivals within 60% of the original duration and raises the offered rate by approximately $1/0.6 = 1.67\times$. Scale 0.6 is therefore the higher-arrival-rate setting.

Throughput window and counting. The measurement window runs from the first request's client-side send timestamp to the last received stream chunk across all requests, including the drain after the final arrival. The client waits for every request to finish and reports $\Theta = N_{\log}/D$, where D is this window in seconds. In the supplied client, $N_{\log}$ counts one send timestamp plus every received stream-chunk timestamp per request, so the archived throughput uses this logging count as its token-count proxy.

Figure 8 complements the workload statistics in Table 1 with input- and output-length distributions. The same request prefix and output-length budgets are used across policies within each workload and arrival scale.

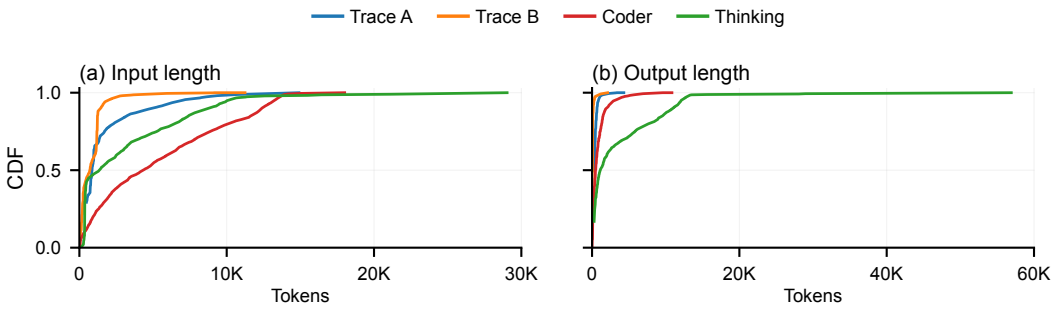


Figure 8: Input- and output-token length CDFs for the workloads in Table 1.

Table 7: Scheduling policies and the priority each label denotes. These descriptions identify the intended ordering signal.

Policy	Priority signal
FCFS: first come, first served	Earlier arrival
SJF: shortest job first	Smaller estimated total job size
LOF: longest output first	Larger output-length signal
LPM: longest prefix match	Greater cached-prefix match
RANDOM	Randomized order

Waiting-queue priorities. The evaluated policies use SGLang's waiting-queue ordering interface. FCFS, LOF, LPM, and RANDOM retain the existing arrival-order, descending output-length,

cached-prefix-match, and randomized ordering rules, respectively. The SJF extension sorts by J_i = `max_new_tokens` in ascending order, where J_i is request i's output-token budget, and breaks ties by waiting-queue entry time. The output-token budget supplies the size signal under the assumption of known output lengths in Section 3.2.

Admission and request re-entry. Priority determines the order in which waiting requests are considered for prefill. Admission then follows the engine's available KV capacity, per-step input-token budget, and running-request limit. Running requests follow the engine's normal execution and memory-pressure retraction mechanisms. A retracted request returns to the waiting queue and is considered for admission again. These semantics connect the policy labels in Table 7 to admission order and batch composition in the measured serving system.

A.3 Measurement Collection and Communication Microbenchmarks

Queue and occupancy measurements. Prometheus snapshots provide queue, running-request, full-attention KV utilization, and recurrent-state occupancy gauges at approximately 10-second intervals. The cache-capacity case study retains only the series whose present rank labels are all zero. The TP8 admission analysis likewise retains rank zero to represent the single logical request queue. Queue-time distributions are reconstructed by differencing cumulative histogram buckets across snapshots and then across bucket boundaries, with monotonicity checks. Each selected TP8 run contains 1,000 queue-time observations, matching its 1,000 client requests. Histogram increments describe newly recorded queue-time observations. Scale windows are reconstructed from collector-session gaps or request-counter boundaries, checked against the JSON request counts and durations, and assigned using the documented 0.6-then-1.0 order.

Table 8: Logical activation-communication operations for one forward microbatch through the model. PP entries count stage-boundary handoffs.

Parallelism	AllReduce	Dispatch	Combine	PP handoffs
TP_R	$2L$	0	0	0
PP_R	0	0	0	$R-1$
EP_R (DPA)	0	M	M	0
TP_q+PP_p	$2L$	0	0	$p-1$
EP_q+PP_p (DPA)	0	M	M	$p-1$

R: total ranks; q: ranks per TP/EP group; p: pipeline stages, with $R = qp$ in mixed parallelisms. L: decoder layers; M: routed-MoE layers. $L = M$ for all the models in our study. The TP/EP degrees in the table exceed one. Each PP handoff carries the microbatch's activation state across one boundary. For μ forward microbatches, multiply counts by μ.

Communication microbenchmarks. Separate communication microbenchmarks measure NCCL collectives and DeepEP dispatch/combine on BF16 hidden-state tensors of width 3,072, using four, eight, and sixteen ranks. After ten warmup iterations, each point averages 20–100 measured iterations; each iteration uses the maximum GPU elapsed time across ranks. For AllReduce, T is the number of token rows *per rank*. For DeepEP, T is the *global source-token count* and each rank starts with T/N rows. The synthetic top-8 routing places all eight selected experts for a token on one destination rank and balances destinations.

Communication establishes different execution regimes. The communication sweep in Figure 9 provides a concrete reason to distinguish parallelisms before comparing policy or other factors. At 1,024 token rows per rank, mean AllReduce latency is 52.8, 115.5, and 231.4 μs for four, eight, and sixteen ranks, respectively. At 32,768 rows it grows to 926.7, 1,523.4, and 2,022.4 μs. Small-message ordering is not uniformly monotonic: at 64 rows the sixteen-rank result is slightly below the eight-rank result.

Figure 10 gives two discrete cross-sections of these regimes and the bars compare measured operation times. DeepEP exhibits a batch-size-dependent crossover. With eight ranks and 64 global source tokens, normal-mode dispatch plus combine sums to 182.9 μs, versus 75.0 μs in low-latency

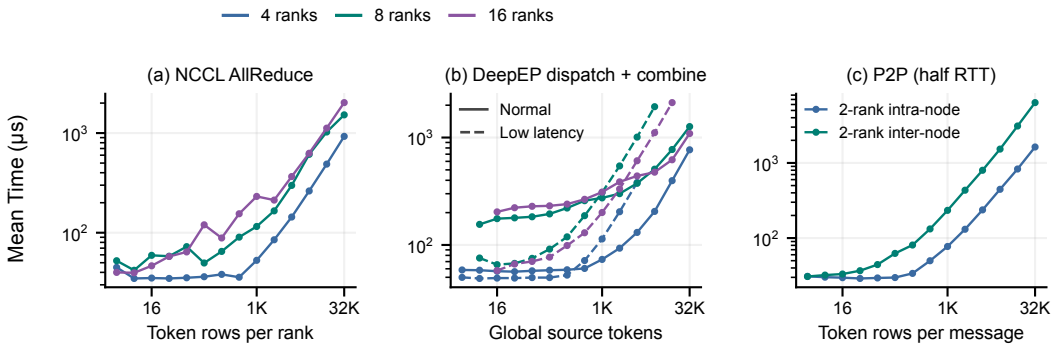


Figure 9: Measured communication regimes for BF16 width-3,072 hidden states. (a) AllReduce varies rank count and node count together. (b) DeepEP dispatch and combine are timed separately and their mean times are summed. Colors identify four, eight, and sixteen ranks in (a,b); solid and dashed lines in (b) denote normal and low-latency modes. DeepEP uses balanced, colocated top-8 routing and global source-token counts, unlike the per-rank token counts in (a). (c) Two-rank intra-node and inter-node P2P times are half of a synchronized ping-pong round trip, measured using pairs 0–1 and 3–4 within the four- and eight-rank benchmark allocations, respectively.

mode. At 8,192 tokens, normal mode is 3.82× faster. This crossover reflects the distinct small- and large-batch operating regimes of the two DeepEP modes (DeepSeek-AI, 2025a).

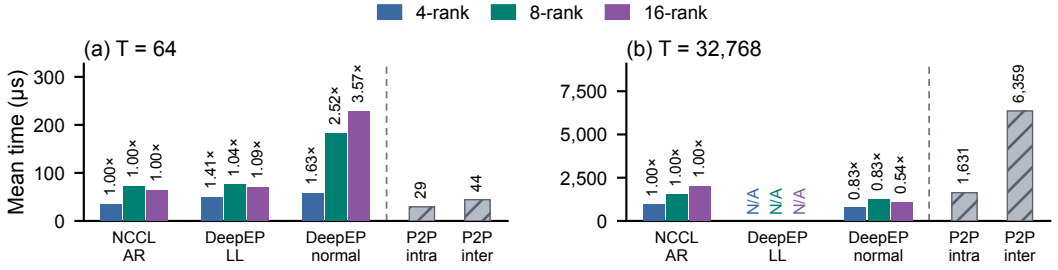


Figure 10: Communication at (a) $T = 64$ and (b) $T = 32{,}768$, BF16 hidden width 3,072. *Every bar height is absolute mean time in microseconds*; panels have different y-axis ranges. Collective annotations give the ratio to same-rank NCCL AllReduce (AR), hence AR is labeled $1\times$ despite different absolute heights. DeepEP bars sum dispatch and combine; LL denotes low latency. Dashed separators isolate two-rank intra-/inter-node P2P, whose labels are absolute times, not ratios. T counts rows per rank for AR, global source tokens for DeepEP (T/N per rank), and rows per transferred P2P tensor. Matching T therefore does not imply equal input volume across operations. P2P uses the half-round-trip measurement of Figure 9. LL is unmeasured at $T = 32{,}768$ because the archived sweep limits it to 1,024 source tokens per rank; N/A is not zero.

A.4 Limitations of Trace Replay

Known output lengths. Size-based policies receive the recorded output-length budget before execution. This gives them advance knowledge that production serving usually lacks. Real requests may terminate early, and length estimates can change policy rankings and admission order. The measured gains therefore describe scheduling with known output budgets; deployment with predicted lengths requires evaluating estimation errors and their effect on policy selection.

Synthetic prompt-token reconstruction. The released traces contain hashed prompt blocks. Replay reconstructs synthetic token sequences with the recorded lengths and block-sharing structure, preserving the intended prefix-reuse relationships. These sequences do not reproduce the original language content or model-generated responses. Content-dependent behavior, including expert routing and activation patterns, can therefore differ from production requests. The experiments char-

Table 9: Staged selection versus exhaustive search. Losses are percentages. Optimum counts exact matches among 32 contexts. Random-two uniformly samples layout pairs; mean loss weights contexts equally and worst loss covers every context and pair.

	FCFS-screened two layouts			Random two layouts	
Objective	Optimum	Mean	Worst	Mean	Worst
Throughput	28/32	0.26	3.84	4.08	42.12
Mean TTFT	32/32	0.00	0.00	100.02	11,659.26
P99 TTFT	32/32	0.00	0.00	239.39	37,462.11
Mean TPOT	32/32	0.00	0.00	17.18	373.70
Mean E2EL	32/32	0.00	0.00	18.16	384.78
P99 E2EL	32/32	0.00	0.00	14.78	273.12

acterize execution under the reconstructed workload; task quality and the effect of natural prompt content remain outside these measurements.

A.5 Evaluation of Staged Layout–Policy Selection

Contexts and performance loss. We evaluate the complete screening-and-tuning procedure described in Section 6.1. Table 3 measures layout coverage while holding each evaluated policy fixed; Table 9 measures the final choice after tuning policies on the two shortlisted layouts. The complete procedure is evaluated retrospectively on the recorded grid, using 32 contexts per objective: four models, four workloads, and two arrival scales. Each context contains three or five layouts and all five policies. Exhaustive search selects the best recorded layout–policy combination. The staged procedure ranks layouts using FCFS for the same objective, retains two, and selects the best combination among their five policies. For exhaustive performance x_* and selected performance x_s, throughput loss is $100(1 - x_s/x_*)$ and latency loss is $100(x_s/x_* - 1)$. Each context receives equal weight when computing mean and worst loss.

Exact random-two baseline. For C candidate layouts, we enumerate all $\binom{C}{2}$ unordered pairs and select the best of each pair's ten layout–policy combinations. Each context has weight $1/32$, split equally among its pairs. We compute the weighted mean over these outcomes; worst loss is the maximum over all contexts and pairs. Thus, the baseline integrates over every possible random shortlist.

Profiling costs. With FCFS screening measurements already available, we count the ten layout–policy evaluations on the two shortlisted layouts. Exhaustive search evaluates $5C$ combinations for C candidate layouts. The reduction is $100(1 - 10/(5C))$: 33.3% for three layouts and 60.0% for five. Averaging these percentages equally over the 32 contexts gives 46.7%. Including screening, the procedure evaluates FCFS on all C layouts and the other four policies on each of the two shortlisted layouts, reusing their FCFS results. This requires $C + 8$ evaluations: 11 instead of 15 for three layouts and 13 instead of 25 for five layouts, reducing configuration evaluations by 26.7% and 48%, respectively.

Comparison with random layout selection. FCFS-screened selection reaches the exhaustive optimum in 28/32 throughput contexts and all 32 contexts for every latency objective. FCFS screening gives smaller mean and worst losses for every objective. Random-two's mean and worst throughput losses are 4.08% and 42.12%, compared with 0.26% and 3.84% for FCFS screening. For mean TTFT, random-two incurs 100.02% mean loss and 11,659.26% worst loss. With the same ten shortlisted evaluations, FCFS screening reduces the risk of excluding a high-performing layout.

B Supporting state accounting, capacity model, and measurement scope

B.1 Architecture and live-state accounting

Figure 11 provides the matched four-GPU comparison supporting the EP–TP analysis in Section 5.1.

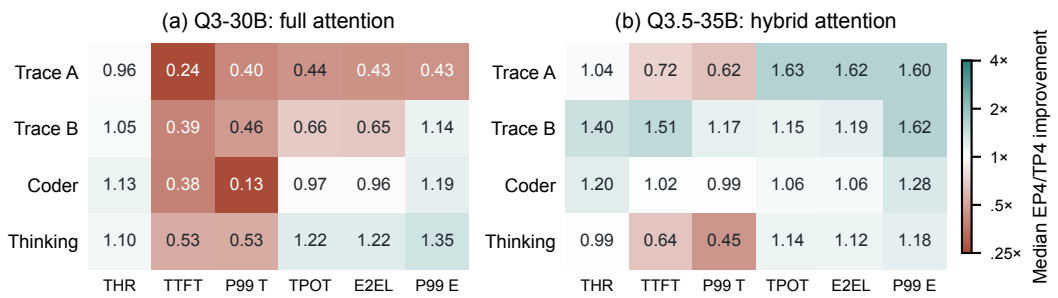


Figure 11: Four-GPU comparison at scale 0.6. Each cell is the median of five policy-paired EP4-versus-TP4 improvement ratios within the same model and workload. Throughput uses EP4/TP4 and latency uses TP4/EP4. Values above one favor EP4. Colors share one log-ratio scale. Models are compared under a matched GPU budget.

Figure 12 complements the matched-budget comparison with layout preferences across deployment scales (Section 5.2).

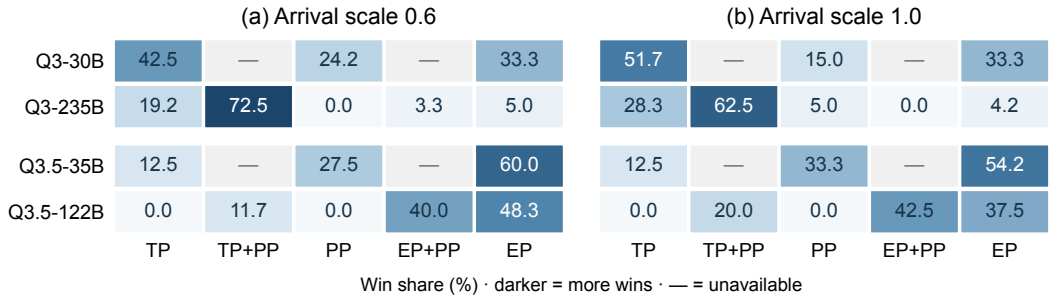


Figure 12: Best-parallelism percentages at arrival scales 0.6 and 1.0. Cells report win shares (%); darker shading indicates larger shares, and dashes mark unavailable families. Full-attention models appear above the gap and hybrid-attention models below it. Each model/scale uses all $4\times5\times6 = 120$ workload–policy–metric decisions.

For Figure 6, observations come from the reconstructed FCFS arrival windows, sampled at approximately 10-second intervals. KV utilization uses rank 0. For EP4 state slots, we sum used, free, and evictable counts across all four owners at each matching timestamp, then average each category's share of the combined 1,152-slot pool. TP4 uses its 1,152-slot rank-0 pool. The three shares sum to 100%, and evictable slots contain reclaimable cached state. KV utilization is normalized by each layout's rank-0 KV-pool capacity. Running-request counts likewise sum the four EP4 owners before averaging, while TP4 uses rank 0. TP4 partitions recurrent state across ranks, while EP4 assigns requests to DPA owners.

The accounting assumes BF16 KV and live recurrent matrices, balanced placement, no prefix reuse, and representative context c. For L_F full-attention layers with K KV heads of dimension d, the key and value caches require

$$m_K = 2 \text{ bytes} \times 2L_F K d = 4L_F K d. \tag{6}$$

For L_L linear-attention layers with h_v value heads and recurrent key/value dimensions $d_k = d_v = 128$, matrix storage is

$$R = 2L_L h_v d_k d_v. \tag{7}$$

Convolution buffers, padding, prefix checkpoints, and unused pool slots are excluded. Table 5 gives the resulting coefficients. The context crossover c_* is where unreplicated KV and recurrent storage are equal. Full-attention models have $R = 0$ and $\varphi_K = 1$ for positive context.

In TP, value heads are partitioned and each head's recurrent matrix remains intact. With t TP ranks and p balanced pipeline stages, per-rank recurrent storage per request is $R/(tp)$. In EP/DPA with e owners and p stages, an owner stores R/p per owned request but owns only b_ℓ/e requests. Hence

per-GPU live state is

$$\underbrace{\frac{b_\ell}{tp}(r_\ell m_K c + R)}_{\text{TP+PP, } N=tp}, \qquad \underbrace{\frac{b_\ell}{ep}(m_K c + R)}_{\text{EP+PP, } N=ep}. \tag{8}$$

Both have recurrent storage $b_\ell R/N$ per GPU. For example, Qwen3.5-35B stores 7.5 MiB per request per TP4 rank and 30 MiB per owned request per EP4 owner, but each EP owner serves one quarter of the requests under balanced assignment. This produces equal aggregate recurrent storage at equal global concurrency. Global request caps are 384 and 128 for Qwen3.5-35B and Qwen3.5-122B, and the idealized full-attention cases use $B_{\text{req}} = \infty$. DPA divides a global cap among owners.

B.2 From architecture to conditional service capacity

To connect these mechanisms to aggregate service, let v_ℓ be bottleneck service cost per completed output token for a representative sustained workload. The qualitative accounting is

$$v_\ell = v_* + C_\ell + S_\ell + E_\ell, \qquad \Gamma_\ell = \frac{v_0}{v_\ell}, \qquad \Gamma_\ell > 1 \iff (C_0 - C_\ell) + (S_0 - S_\ell) > E_\ell - E_0. \tag{9}$$

Here v_* is equal useful arithmetic across parallelisms; C is exposed communication cost, S exposed state-traffic cost, and E the cost of weight rereads, small local batches, pipeline underfill, and unequal utilization. Prefill is amortized over the same completed outputs, and overlapped time is not counted twice.

Communication locality can reduce C: for example, TP4+PP4 replaces TP16's cross-node layer collectives with node-local collectives and stage-boundary transfers. Smaller microbatches or pipeline underfill may increase E. A balanced persistent decode pipeline has ideal utilization approximately $\min(1, q/p)$ for q ready independent microbatches and p stages.

For fixed service profiles and offered output-token rate $\lambda\bar{O}$, an approximation gives

$$\frac{X_\ell}{X_0} = \frac{\min(u, \Gamma_\ell)}{\min(u, 1)}, \qquad u = \lambda\bar{O}\, v_0 > 0, \tag{10}$$

where X is output-token throughput and λ request arrival rate. Throughput is equal when both systems are arrival-limited; a capacity advantage becomes visible under sufficient load.

B.3 Derivation of the relative mean-TTFT decomposition

For each request i under parallelism ℓ, write $T_{\ell i} = Q_{\ell i} + P_{\ell i}$, where queueing ends at admission and all subsequent delay before the first token belongs to $P_{\ell i}$. Averaging over the same request set gives $\bar{T}_\ell = \bar{Q}_\ell + \bar{P}_\ell$ by linearity. The reference likewise satisfies $\bar{T}_0 = \bar{Q}_0 + \bar{P}_0$.

For positive reference means, define

$$R_T = \frac{\bar{T}_\ell}{\bar{T}_0}, \qquad R_Q = \frac{\bar{Q}_\ell}{\bar{Q}_0}, \qquad R_P = \frac{\bar{P}_\ell}{\bar{P}_0}, \qquad w_Q = \frac{\bar{Q}_0}{\bar{T}_0}.$$

The subscripted R_T, R_Q, R_P are dimensionless ratios and are distinct from the unsubscripted recurrent-state size R in Section 5.1. Dividing the candidate's mean TTFT by the reference mean and inserting the reference component means yields

$$\begin{aligned} R_T &= \frac{\bar{Q}_\ell + \bar{P}_\ell}{\bar{T}_0} \\ &= \frac{\bar{Q}_0}{\bar{T}_0}\frac{\bar{Q}_\ell}{\bar{Q}_0} + \frac{\bar{P}_0}{\bar{T}_0}\frac{\bar{P}_\ell}{\bar{P}_0} \\ &= w_Q R_Q + (1 - w_Q) R_P, \end{aligned}$$

which is Equation 5. The weights sum to one because $\bar{P}_0/\bar{T}_0 = 1 - w_Q$: each component ratio is weighted by its share of baseline TTFT.

Zero baseline queueing. If $\bar{Q}_0 = 0$, R_Q is undefined, so the weighted-ratio expression must not be evaluated as a zero weight times an undefined ratio. Instead, using $\bar{T}_0 = \bar{P}_0 > 0$ directly gives

$$R_T = \frac{\bar{Q}_\ell + \bar{P}_\ell}{\bar{P}_0} = R_P + \frac{\bar{Q}_\ell}{\bar{P}_0}.$$

It reduces to $R_T = R_P$ only if the candidate also has zero mean queueing. The additive condition in Equation 4 avoids this ratio boundary case.

C Quantitative Queueing and Concurrency Evidence

Table 10 joins the server-side queue histogram means with client mean TTFT for Q3.5-122B/TP8, Trace B, scale 1.0. Each policy contributes 1,000 completed requests and 1,000 queue-time observations over the same run window. The residual $\bar{P}^{\text{res}} = \bar{T} - \bar{Q}$ combines execution, post-admission interference, and any difference between client and server timing boundaries. It represents aggregate post-admission time.

Table 10: Matched mean delays (seconds), sampled running-request peaks, and the post-admission residual of means.

Policy	Peak running	$\bar{Q}$	$\bar{T}$	$\bar{P}^{\text{res}}$
FCFS	127	19.498	19.678	0.179
SJF	127	11.666	11.875	0.208
LOF	128	30.324	30.517	0.192
LPM	127	19.715	19.899	0.184
RANDOM	127	20.807	20.998	0.192

At the trace-derived representative decode context of 1,738 tokens, Equation 3 gives a 128-request bound for this TP8 layout under the illustrative 10-GiB state budget and configured cap. The observed peaks of 127–128 closely match this bound. Nevertheless, mean TTFT varies by $30.517/11.875 = 2.570\times$ across the five policies. Peak occupancy is nearly constant, so the variation arises from admission order and request residency. This result directly connects policy-dependent queueing to the TTFT spread observed under a fixed execution layout.

Figure 13 shows the matched queue-time distributions. LOF has fewer requests above 10 seconds of queueing while its mean queue time reaches 30.324 seconds, compared with 19.498 seconds under FCFS (Table 10). The mean and the mass above a single threshold capture different aspects of the waiting-time distribution.

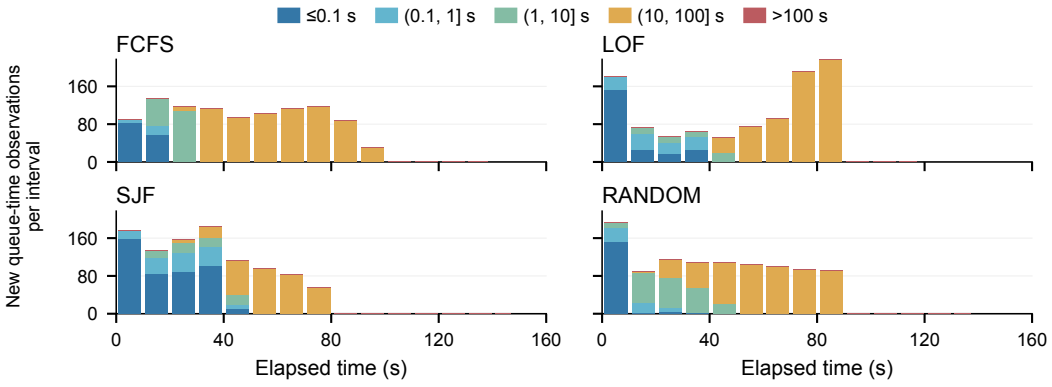


Figure 13: Trace B queue-time distributions over time for Q3.5-122B, TP8, scale 1.0, comparing FCFS, LOF, SJF, and RANDOM. Rank 0 histogram increments account for all 1,000 requests per policy.

D VALIDATION OF THE REGIME DESCRIPTORS

Regime variables. The Q3.5-122B/TP8 case in Section 6.3, with matched measurements in Appendix C, shows how admission waiting creates room for scheduling to change TTFT. We characterize this opportunity with normalized load $\rho_\ell = \lambda\bar{O}/\mu_\ell$, queue share $w_{Q,\ell} = \bar{Q}_\ell/\bar{T}_\ell$, and concurrency utilization $a_\ell = \bar{b}_\ell/b_\ell^{\max}$. Here λ is the request arrival rate, $\bar{O}$ the mean output length, μ_ℓ the sustained output-token capacity, $\bar{b}_\ell$ the mean running-request count, and $b_\ell^{\max}$ the admission bound in Equation 3. Normalized load measures demand relative to capacity, queue share measures the fraction of TTFT spent waiting for admission, and concurrency utilization measures proximity to the admission bound.

The full grid provides estimated load and state descriptors. The predictor therefore combines estimated normalized load with $b_j^{\max}$ and total-state inflation I_j. Appendix F separately analyzes measured queue share in the 110 configurations with complete queue traces.

Prediction target. The 640 archived result records yield 128 cells with five policies each. For cell j, let

$$z_j = \log \frac{\max_\pi \bar{T}_{j,\pi}}{\min_\pi \bar{T}_{j,\pi}} = \log(1 + V_{\mathrm{TTFT},j}/100).$$

This target measures mean-TTFT sensitivity within a cell. The five policy records form one target. We verify all cell metrics against the 640 raw result JSON files.

Features and estimator. We use three log features:

$$x_j = \left(\log \widetilde{\rho}_j,\ \log b_j^{\max},\ \log I_j\right), \quad \widetilde{\rho}_j = \frac{\lambda_j \bar{O}_j}{\widetilde{\mu}_j}, \quad I_j = \frac{r_j m_K c_D + R}{m_K c_D + R}.$$

The capacity $\widetilde{\mu}_j$ is fixed from the earlier uncalibrated FCFS service scenario, which uses assumed batches and service costs. Thus, $\widetilde{\rho}_j$ is a scenario-based load estimate. We recompute $b_j^{\max}$ from Equation 3 using 10 GiB per GPU, BF16 recurrent-state geometry, representative context c_D, and the global hybrid-model caps of 384 and 128. The feature I_j measures total state inflation relative to an unreplicated parallelism. Workload length and rate summaries come directly from the trace-replay inputs.

For each split, we standardize features using the training statistics and fit an unpenalized intercept with ridge-regularized coefficients:

$$\min_{\beta_0,\beta} \sum_{j \in \mathcal{D}_{\mathrm{train}}} (z_j - \beta_0 - \beta^\top x_j^{\mathrm{std}})^2 + \|\beta\|_2^2, \qquad \widehat{z}_j = \max(0, \widehat{\beta}_0 + \widehat{\beta}^\top x_j^{\mathrm{std}}).$$

The penalty is fixed at one. The constant baseline predicts the training mean of z. Training data determine all preprocessing statistics and coefficients.

Grouped validation. Leave-one-workload-out keeps both scales and all parallelisms of a workload in the test fold. Leave-one-model-out keeps every workload, scale, and parallelism of a model together. Scale transfer trains on the 64 scale-1.0 cells and tests the 64 scale-0.6 cells with matched model, workload, and parallelism identities. Table 11 reports pooled cell-weighted mean absolute error (MAE) in z. Figure 14 shows the load association and held-out workload predictions.

Table 11: Grouped validation for mean-TTFT spread. Lower MAE in z is better. The baseline uses only the training mean.

Protocol	Test cells	Ridge MAE	Constant MAE
Leave workload out	128	0.510	0.504
Leave model out	128	0.676	0.609
Scale 1.0 to 0.6	64	0.419	0.450

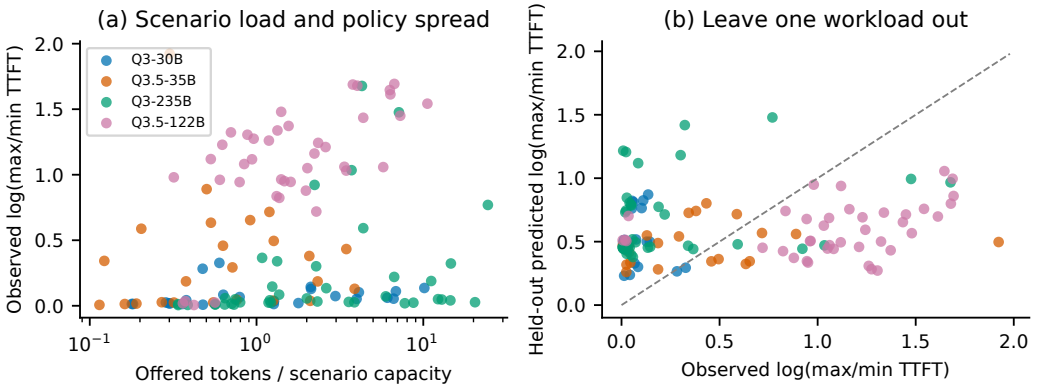


Figure 14: (a) Frozen scenario load versus observed policy sensitivity for 128 cells; colors identify models. The horizontal coordinate is estimated load. (b) Leave-one-workload-out predictions with the identity line dashed.

Table 12: Individual held-out folds. All coefficients and normalization statistics are estimated using the other groups only.

Protocol	Held-out group	Test cells	Ridge MAE	Constant MAE
Leave workload out	Coder	32	0.452	0.505
Leave workload out	Thinking	32	0.656	0.551
Leave workload out	Trace A	32	0.497	0.530
Leave workload out	Trace B	32	0.436	0.432
Leave model out	Q3-235B	40	0.731	0.521
Leave model out	Q3-30B	24	0.662	0.513
Leave model out	Q3.5-122B	40	0.860	0.906
Leave model out	Q3.5-35B	24	0.294	0.357
Scale 1.0 to 0.6	0.6	64	0.419	0.450

Prediction results. The estimator improves scale-transfer MAE from 0.450 to 0.419 but trails the constant baseline in the pooled workload and model holdouts. It improves two of four individual workload folds and two of four model folds, with errors varying substantially across held-out groups. The largest model-fold gap occurs for Q3-235B, where MAE rises from 0.521 to 0.731. These results show that the current load and state descriptors capture scale changes more reliably than workload or model changes.

E Capacity Settings for Full-Attention Models

A matched request-cap intervention. We test whether the conclusions for Q3-30B change when its configured concurrent-request ceiling is reduced from `auto` to 384, matching the ceiling used by Q3.5-35B. The additional experiment covers TP4, PP4, and EP4 on four GPUs, all four workloads, and all five policies at arrival scale 0.6: 60 cap-384 runs paired with 60 default runs. In Equation 3, this intervention sets $B_{\text{req}} = 384$ for the additional Q3-30B runs. PP4 uses 96-request microbatches and EP4 allocates 96 request slots per attention owner. Actual running concurrency also depends on arrivals, state residency, and execution progress.

The relative importance of parallelism persists. Table 13 compares the two settings using the same spread definition as Equation 1. Median policy spread remains small, moving from 2.49% to 2.79%, while median parallelism spread changes from 52.04% to 48.36%. Thus, the substantially larger variation across execution parallelisms persists after matching the hybrid model's configured request cap. The policy-spread median covers 72 workload–parallelism–metric cells; the parallelism-spread median covers 120 workload–policy–metric cells. The six metrics are throughput, mean and P99 TTFT, mean TPOT, and mean and P99 E2EL. These aggregate medians coexist with larger changes in individual TTFT cases.

Table 13: Q3-30B at scale 0.6: sensitivity and best-parallelism counts under the two request caps. Each best-parallelism column counts wins among the same 120 workload–policy–metric comparisons.

Request cap	Policy spread	Parallelism spread	TP4 wins	PP4 wins	EP4 wins
`auto`	2.49%	52.04%	51	29	40
384	2.79%	48.36%	60	20	40

TTFT rankings remain sensitive to the cap. The best parallelism changes in 13 of 120 comparisons (10.8%): 6 of 20 mean-TTFT comparisons and 7 of 20 P99-TTFT comparisons. Every switch is between TP4 and PP4. Throughput, mean TPOT, and both E2EL objectives retain all 20 policy-specific winners per objective. Across the 12 workload–parallelism pairs, median cap-384/default throughput ratios over policies range from 0.985 to 1.020. TTFT changes can be considerably larger: EP4 on Coder has median ratios of 1.348 for mean TTFT and 1.659 for P99 TTFT, alongside a throughput ratio of 1.003. PP4 on Thinking has a P99-TTFT ratio of 1.413 and a throughput ratio of 1.020. Here latency ratios above one indicate longer delay. These observations reinforce the separation between aggregate service efficiency and the request's first-token path, and motivate re-evaluating TTFT rankings whenever admission settings change.

The EP–TP family comparison survives cap matching. For each workload and metric, we take the median of the five policy-paired EP4/TP4 improvement factors, using throughput directly and inverse latency. All 24 median comparisons retain their preferred parallelism under cap 384. EP4's share of best-parallelism decisions remains 40/120 (33.3%). At the same cap and arrival scale, Q3.5-35B has a median policy spread of 8.55% and EP4 wins 72/120 decisions (60.0%). The observed difference between these checkpoints therefore persists after matching the configured request ceiling. Their state geometry, expert structure, layer composition, and kernel paths continue to vary, supporting an interpretation at the level of the evaluated architectures.

FCFS screening benefits from retaining two parallelisms. We rebuild the FCFS shortlist separately for each cap, workload, and objective, then check it against the four other policies. The top-two shortlist covers all 16 policy-specific winners per objective under both caps. Top-one coverage varies for TTFT (Table 14): mean-TTFT coverage stays at 13/16, while P99-TTFT coverage falls from 14/16 to 9/16. At cap 384, choosing only the FCFS-best parallelism can incur a $1.281\times$ P99-TTFT gap to the same policy's optimal parallelism. Throughput, mean TPOT, and both E2EL objectives retain 16/16 top-one coverage. Within this three-parallelism candidate set, the two-parallelism shortlist preserves the screening result across the capacity change; selecting a single parallelism requires greater care for tail latency.

Table 14: FCFS-based screening for Q3-30B TTFT objectives. Each condition has 16 comparisons (four workloads and four non-FCFS policies). The final column is the worst selected/optimal latency ratio using cap 384's FCFS top-one parallelism. Top-two coverage is 16/16 under each cap.

Objective	Top-one, auto	Top-one, 384	Top-two, each cap	Worst gap, 384
Mean TTFT	13/16	13/16	16/16	$1.038\times$
P99 TTFT	14/16	9/16	16/16	$1.281\times$

Implications. The capacity intervention preserves three broad findings: larger aggregate variation across parallelism than policies, the checkpoint-dependent EP–TP preference, and the coverage of FCFS top-two screening. It changes several TTFT-optimal parallelisms and weakens P99 top-one screening. Request capacity therefore belongs in the definition of the serving regime used for latency selection.

F Queue Pressure and Scheduling Sensitivity

The queueing decomposition motivates an observable regime descriptor: FCFS queue share, $w_j = \bar{Q}_{j,\mathrm{FCFS}}/\bar{T}_{j,\mathrm{FCFS}}$, for configuration j, corresponding to w_Q in Equation 5. We examine its associa-

tion with the spread of mean TTFT across the five scheduling policies,

$$V_{\text{sched},j} = 100 \left(\frac{\max_\pi \bar{T}_{j,\pi}}{\min_\pi \bar{T}_{j,\pi}} - 1 \right).$$

Each point fixes model, workload, arrival scale, and parallelism layout. Queue share describes the fraction of baseline first-token latency spent waiting for admission; policy spread describes the variation attainable by changing the ordering rule in that configuration.

Matched observations and coverage. We reconstruct FCFS run windows from arrival-counter boundaries and collector-session gaps and assign the chronological scales 0.6 and 1.0. TP and PP use the stage-zero, rank-zero logical queue; EP(DPA) aggregates the exported attention owners at stage zero, weighting their sums by the total observation count. Every retained window has exactly one queue-time observation per benchmark request, with monotone counters and $0 \le \bar{Q} \le \bar{T}$.

The resulting sample contains 110 of the 128 configurations: 56 full-attention and 54 hybrid-attention cases, with 55 at each arrival scale. Sixteen pure EP8/EP16 cases have partial owner coverage in the archived metrics; two Q3.5-35B/EP4/Trace A cases have incomplete arrival-window coverage. These 18 cases are excluded by the measurement-completeness checks. The retained sample spans TP, PP, EP, TP+PP, and EP+PP. Client TTFT and server queueing use their respective timing boundaries, so the queue share includes that measurement alignment at the level of run means.

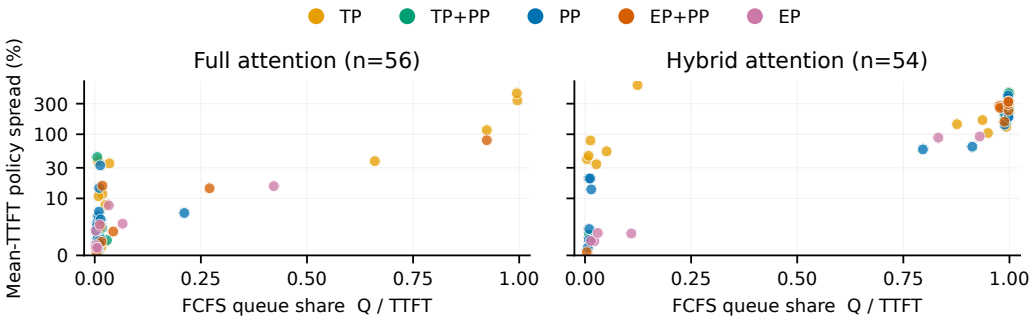


Figure 15: FCFS queue pressure and mean-TTFT scheduling sensitivity across 110 evaluated configurations. Panels separate full and hybrid attention; colors identify execution-layout families. The vertical axis is linear from 0 to 10% and logarithmic above 10%. Each point summarizes five policy outcomes. High queue share accompanies larger policy spread, with substantial variation within and across parallelisms.

Coverage across regimes and subgroups. Figure 15 visualizes the association summarized in Section 6.3. The low ($w_j < 0.1$), intermediate ($0.1 \le w_j < 0.9$), and high ($w_j \ge 0.9$) queue-share groups contain 65, 9, and 36 configurations, respectively. Rank correlations are positive within each parallelism family, ranging from 0.745 for EP to 0.929 for EP+PP, and within both attention families: 0.551 for full attention and 0.847 for hybrid attention. Queue pressure therefore helps identify substantial scheduling opportunity, while execution parallelism, useful concurrency, and request composition help interpret the remaining variation.

Adjustment and robustness checks. Table 4 reports associations across the 110 retained configurations; its 95% intervals use the 16-group clustered bootstrap described below. To account for the configuration mix, we rank-transform queue share and policy spread, regress each rank vector on an intercept and categorical indicators for model, workload, parallelism family, and arrival scale, and correlate the two residual vectors. This partial rank correlation is 0.455, with a clustered interval of $[0.044, 0.690]$. For each of 5,000 bootstrap draws, we sample the 16 model–workload groups with replacement, retain all parallelisms and both scales within each selected group, and recompute ranks and adjustment fits. The intervals are the 2.5th and 97.5th percentiles of these draws and summarize variation across the sampled configuration groups.

FCFS TTFT appears in the queue-share denominator and contributes to the five-policy spread. Recomputing spread using the other four policies gives correlations of 0.807 pooled and 0.404 adjusted (Table 4). Using mean queue seconds as the pressure descriptor gives 0.817. The positive relationship also persists when omitting each model in turn (pooled correlations 0.588–0.862) and each workload in turn (0.773–0.851). The pooled association persists across these checks. Excluding FCFS widens the adjusted correlation's interval to $[-0.032, 0.673]$, indicating greater uncertainty after accounting for the configuration mix.

Execution parallelism shapes the remaining variation. Conditioning on queue-share rank, model, workload, and scale leaves parallelism-associated structure: adding parallelism-family indicators reduces the in-sample residual sum of squares of policy-spread ranks by 12.0%. A concrete example of the low-queue-share variation in Figure 15 is Q3.5-35B/TP4 on Trace A at scale 0.6, which reaches 80.0% policy spread at $w_j = 0.013$. The parallelism-associated residual structure and this example connect scheduling sensitivity to execution effects alongside admission waiting.

Post-admission delay explains the low-queue-share counterexample. For Q3.5-35B/TP4 on Trace A at scale 0.6, SJF's mean TTFT is 242.6 ms, compared with 134.7–140.3 ms for the other policies, and the 80.0% spread therefore corresponds to 107.8 ms. Queue-time counters cover all 1,000 requests per policy and give mean waits of only 1.37–1.76 ms. Relative to FCFS, SJF reduces queueing from 1.764 to 1.416 ms while increasing the post-admission residual from 134.7 to 241.1 ms. Mean TPOT also rises from 81.5 to 121.5 ms, mean E2EL from 29.4 to 43.9 s, and sampled peak running requests from 219 to 326. These observations are consistent with longer execution and residency sustaining a larger active batch with little admission waiting. Batch composition and prefill–decode interference are plausible mechanisms: low queue share leaves room for TTFT variation after admission.

G Policy and Parallelism Comparisons Across Models and Arrival Scales

Each figure covers one model, workload, and arrival scale. The top row compares all five policies with same-layout FCFS, with one panel per layout. The bottom row compares all layouts with same-policy pure TP, with one panel per policy. Ratios are candidate/reference for throughput (THR) and reference/candidate for latency; TTFT, TPOT, and E2EL are means, and P99 denotes P99 TTFT. RND denotes RANDOM. Axes follow the first panel in each row. Larger values indicate improvement; dashed circles mark parity. Logarithmic radial limits are shared within each row and may differ between rows. Layout labels give the parallelism degrees.

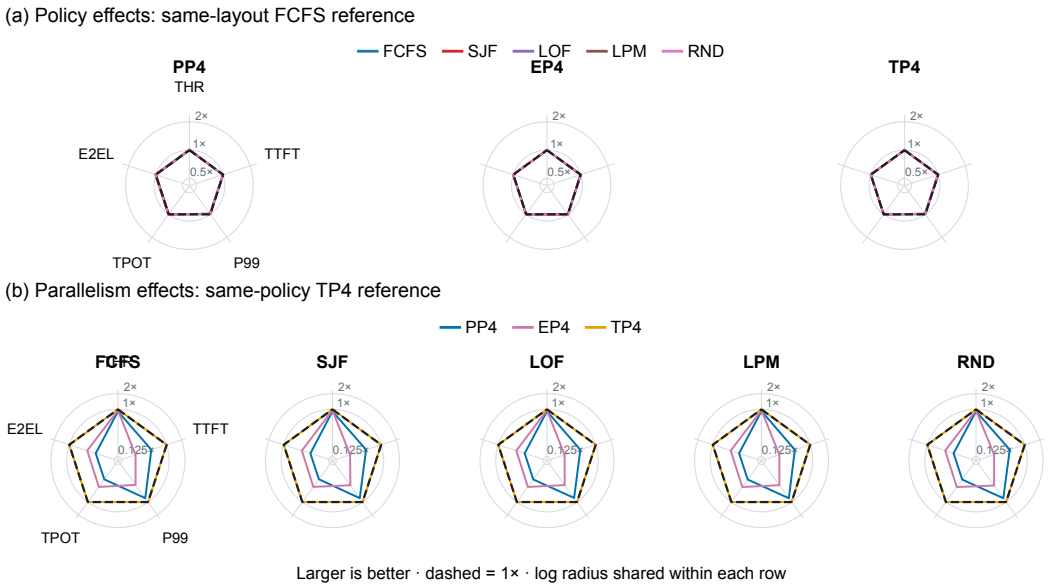


Figure 16: Q3-30B, Trace A, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

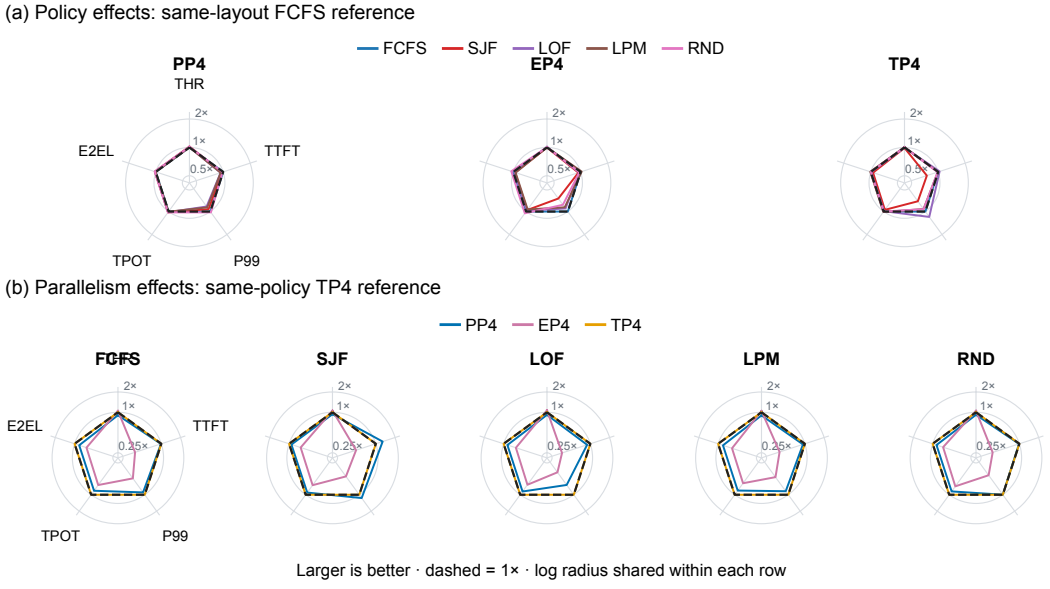


Figure 17: Q3-30B, Trace B, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

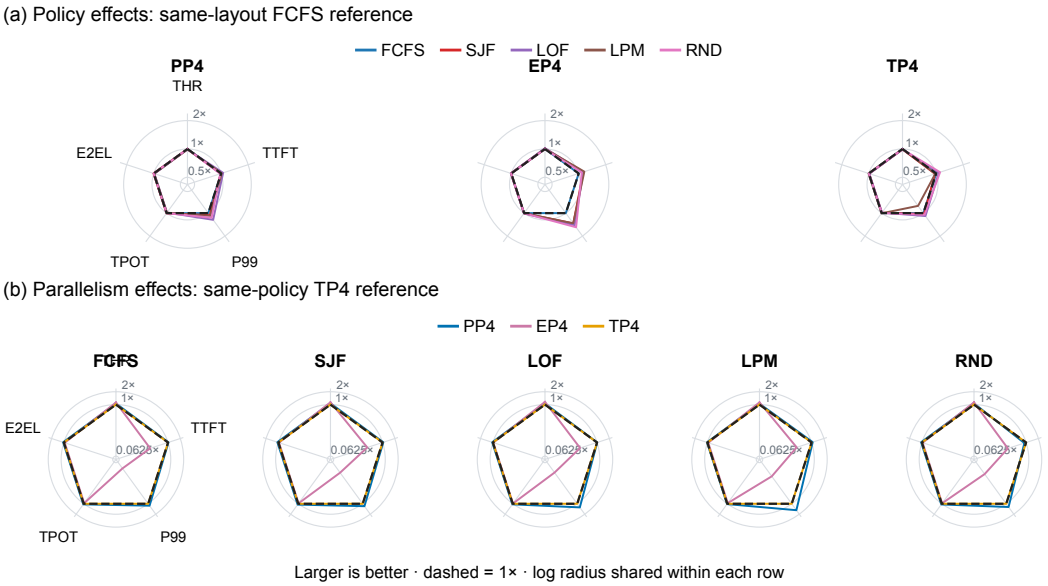


Figure 18: Q3-30B, Coder, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

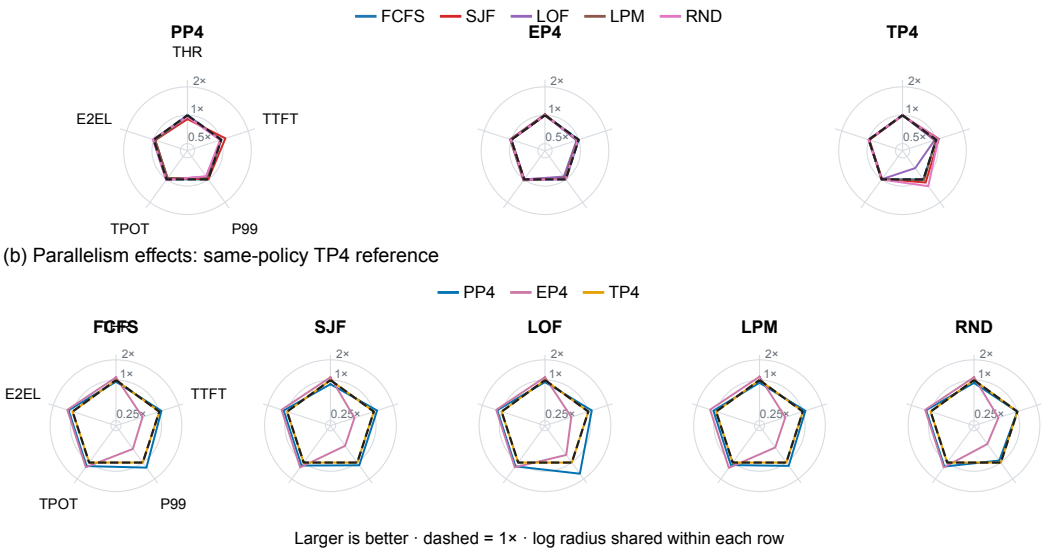


Figure 19: Q3-30B, Thinking, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

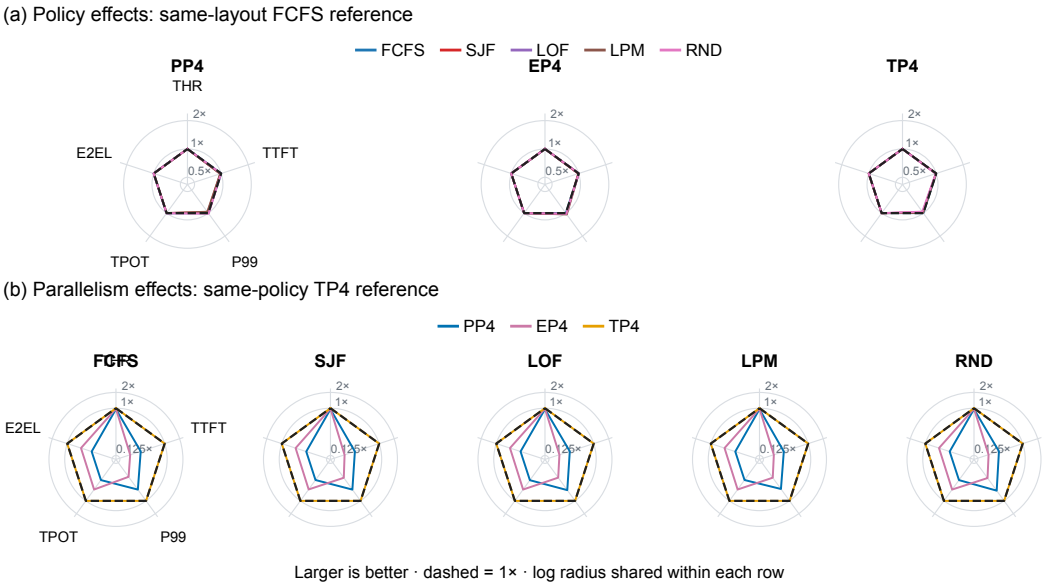


Figure 20: Q3-30B, Trace A, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

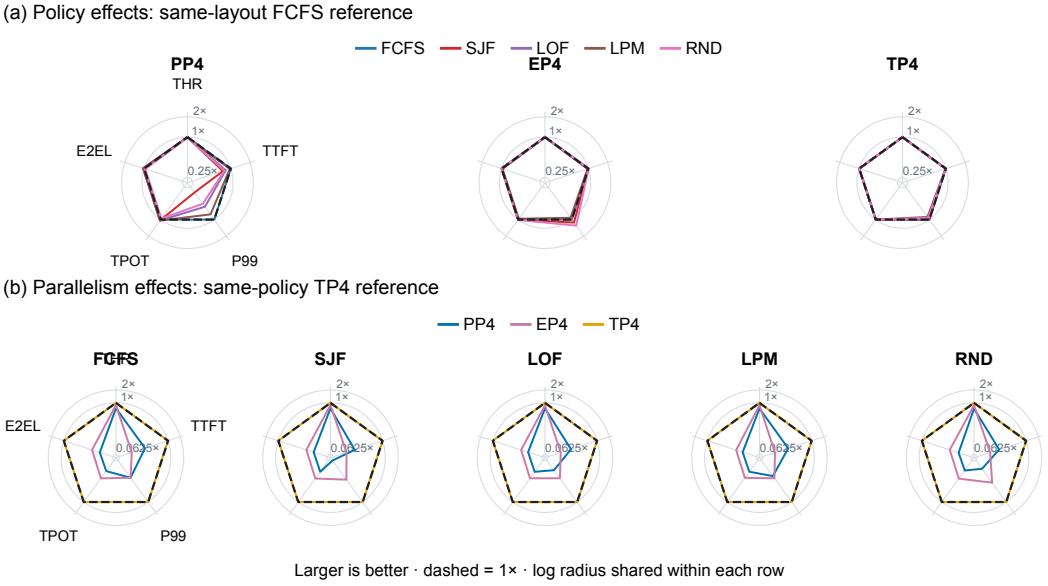


Figure 21: Q3-30B, Trace B, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

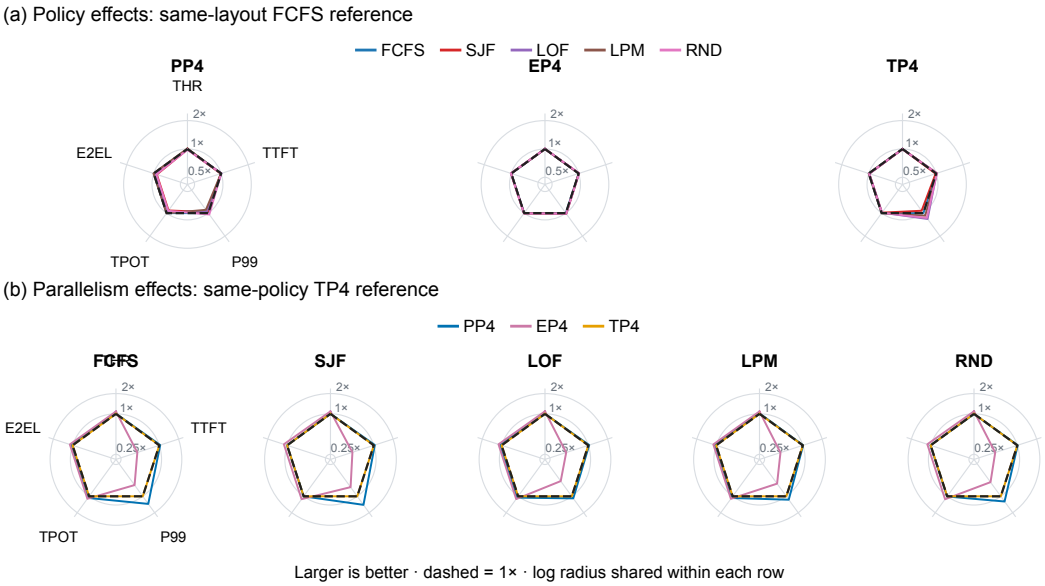


Figure 22: Q3-30B, Coder, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

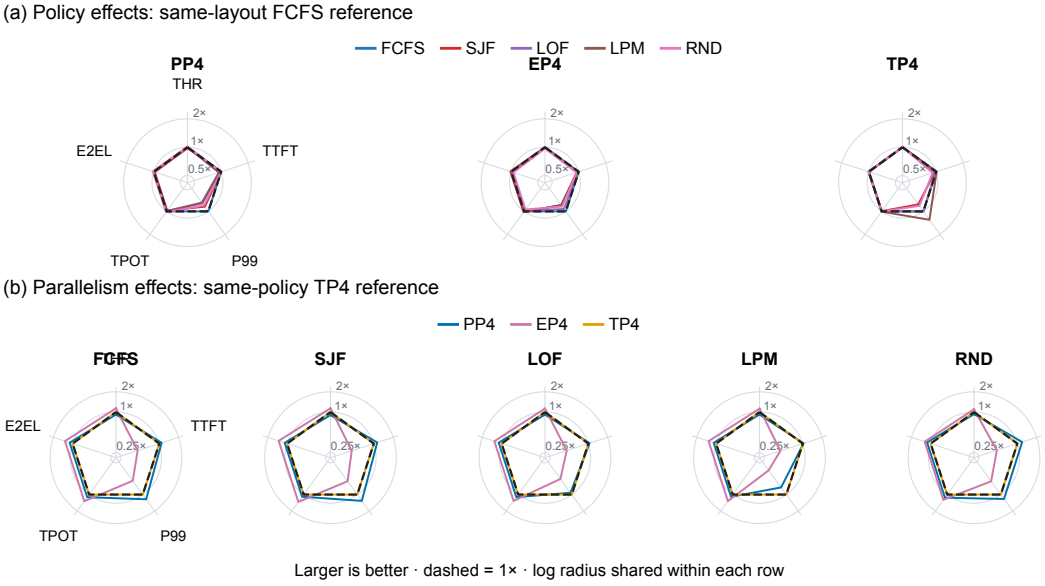


Figure 23: Q3-30B, Thinking, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

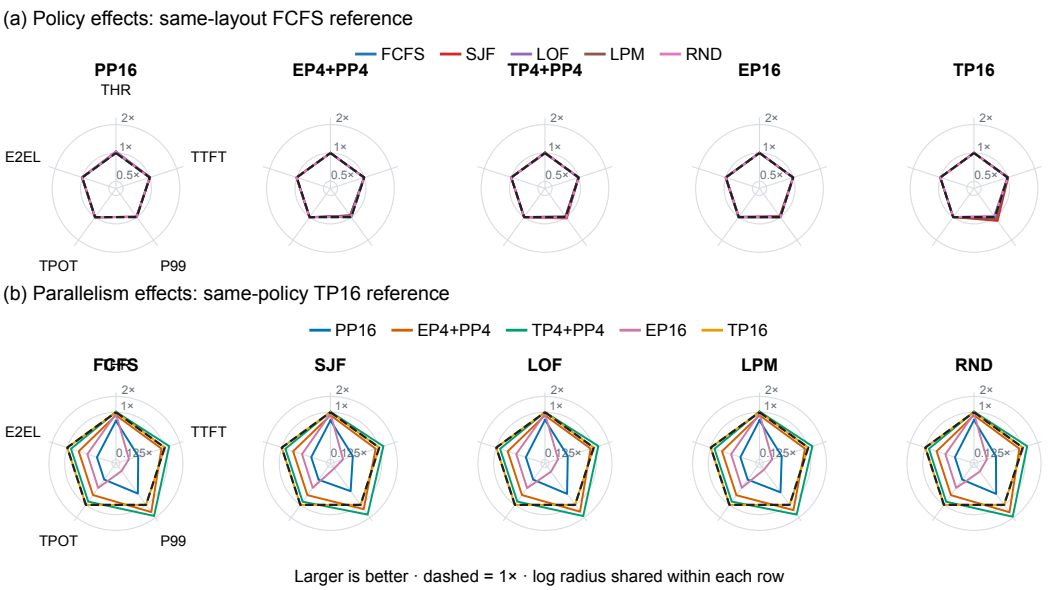


Figure 24: Q3-235B, Trace A, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

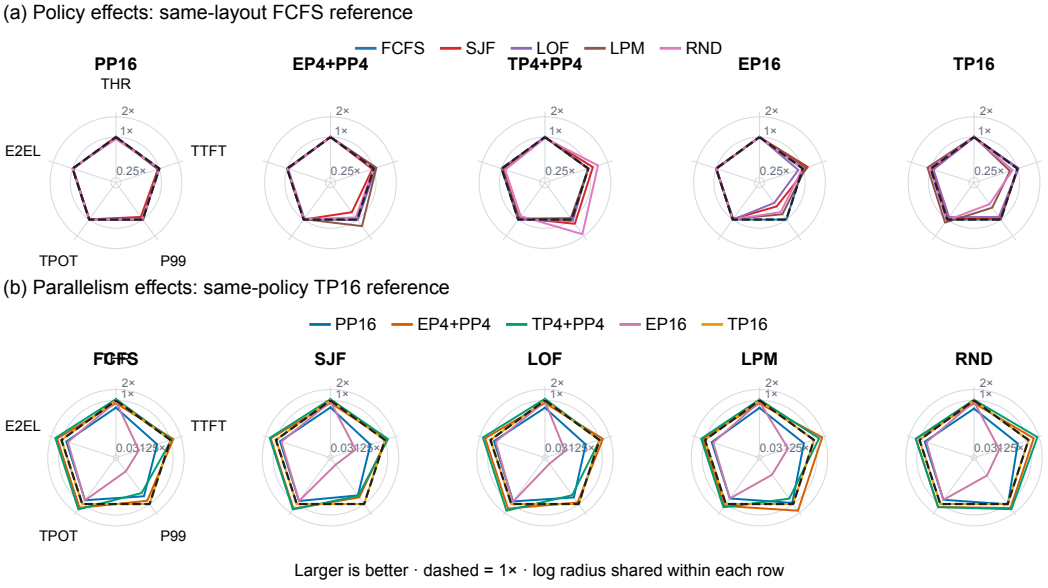


Figure 25: Q3-235B, Trace B, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

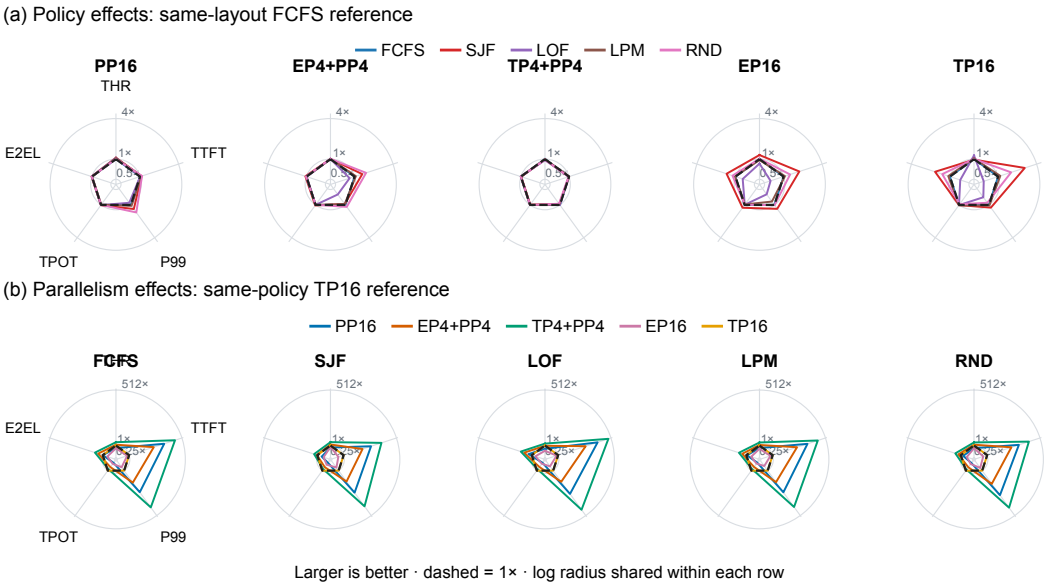


Figure 26: Q3-235B, Coder, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

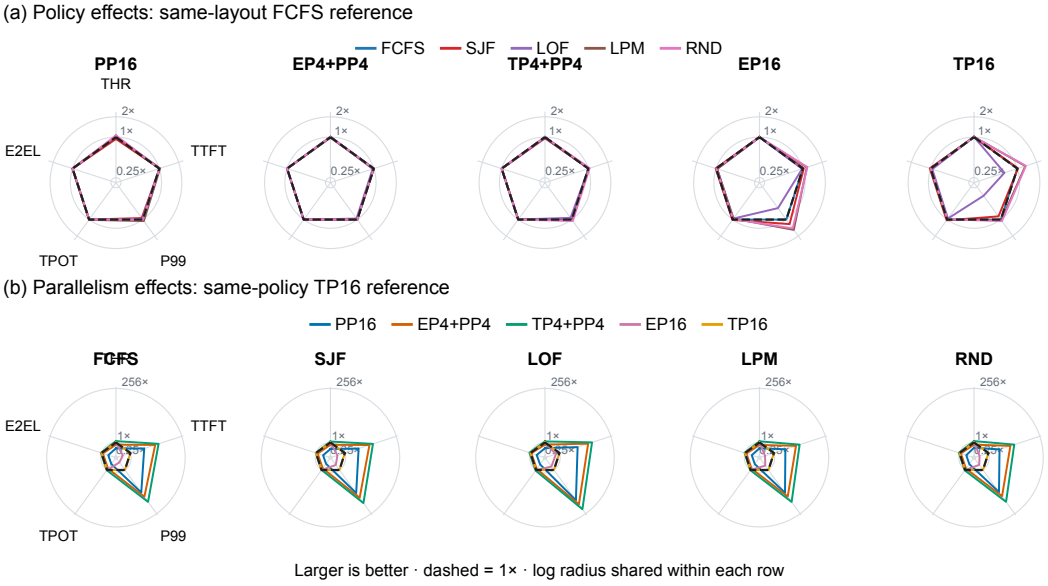


Figure 27: Q3-235B, Thinking, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

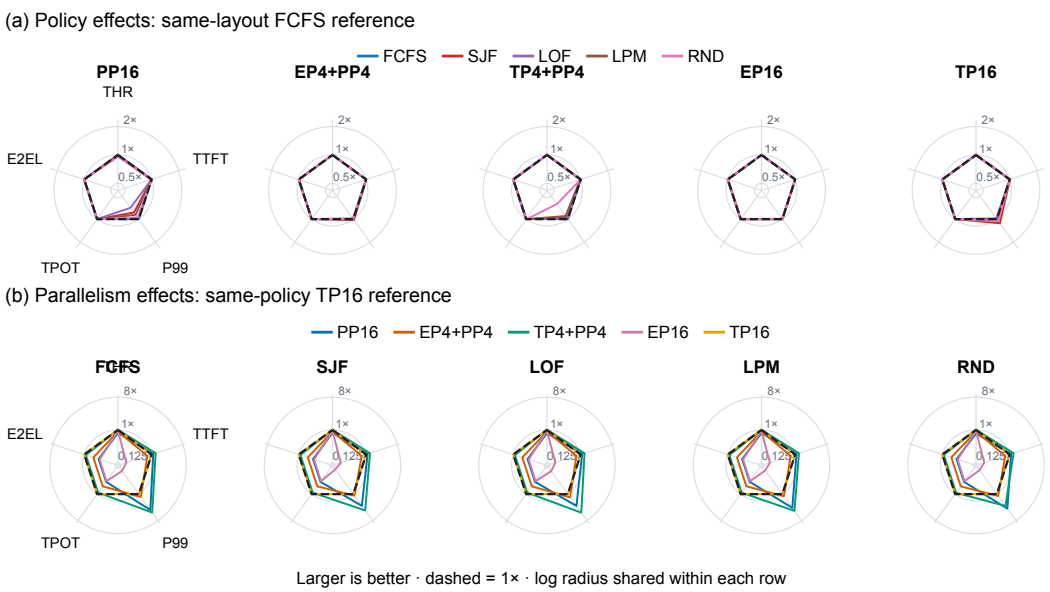


Figure 28: Q3-235B, Trace A, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

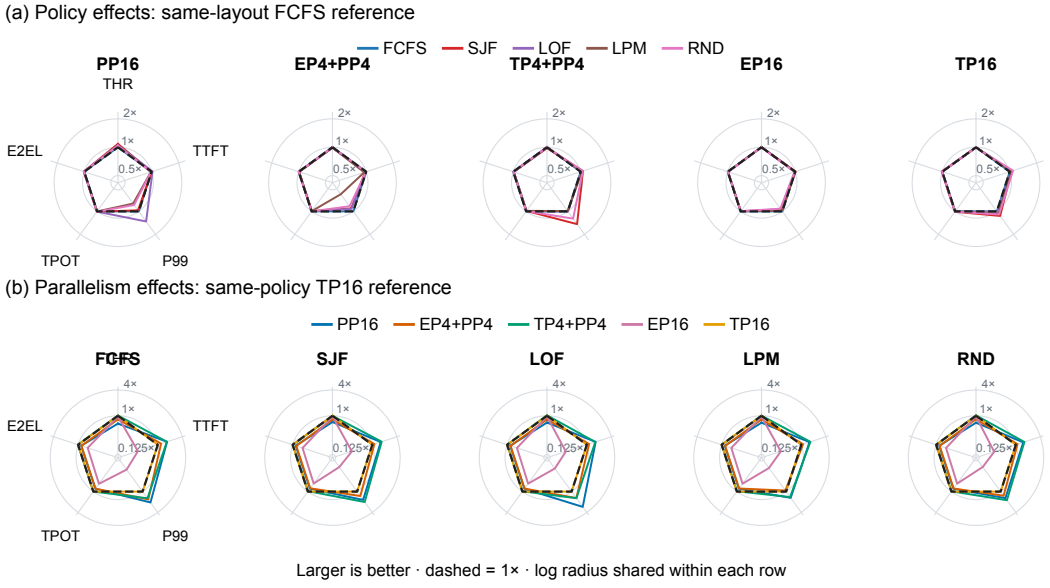


Figure 29: Q3-235B, Trace B, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

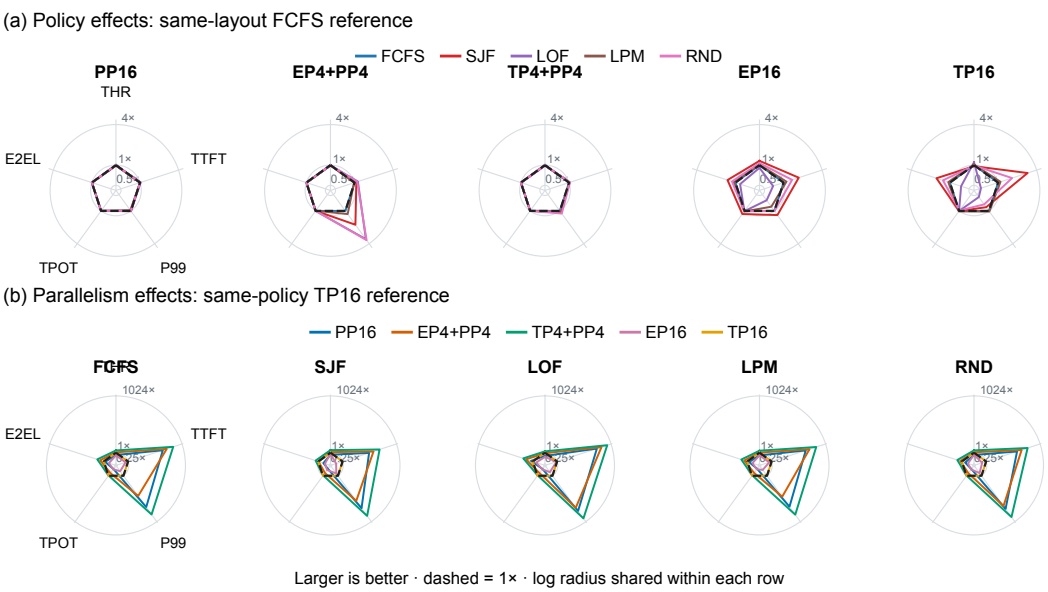


Figure 30: Q3-235B, Coder, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

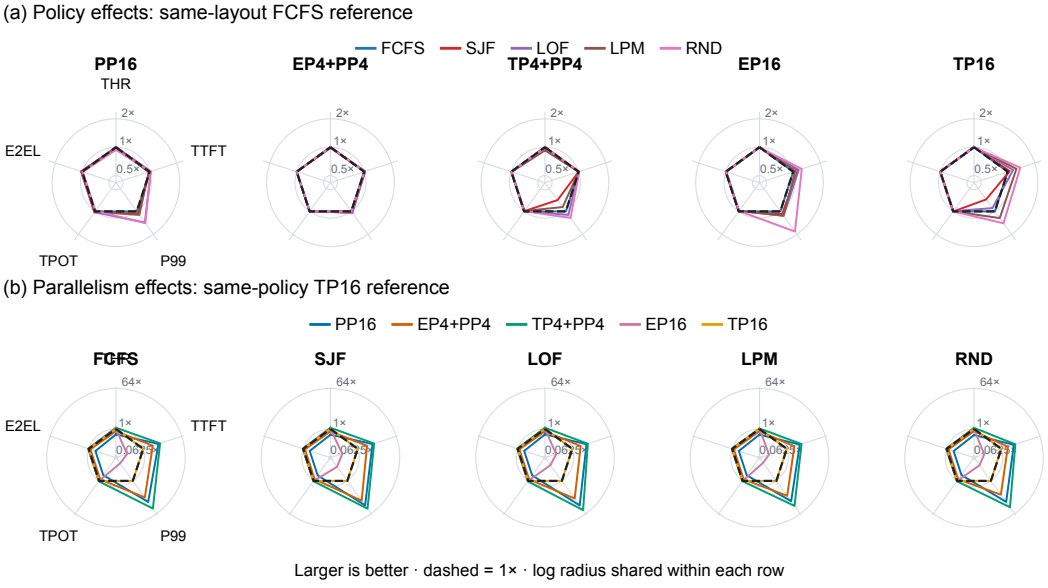


Figure 31: Q3-235B, Thinking, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

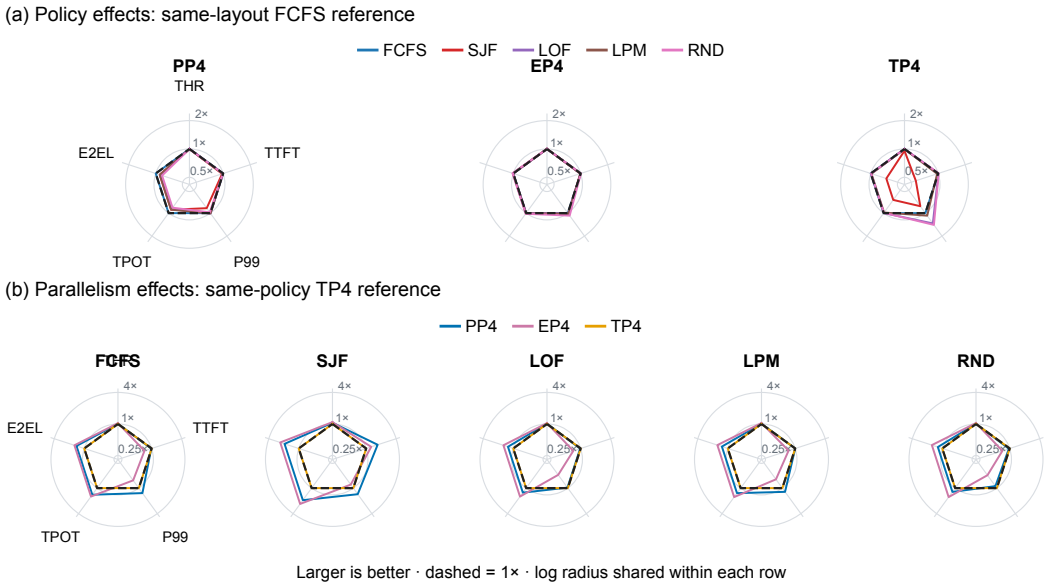


Figure 32: Q3.5-35B, Trace A, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

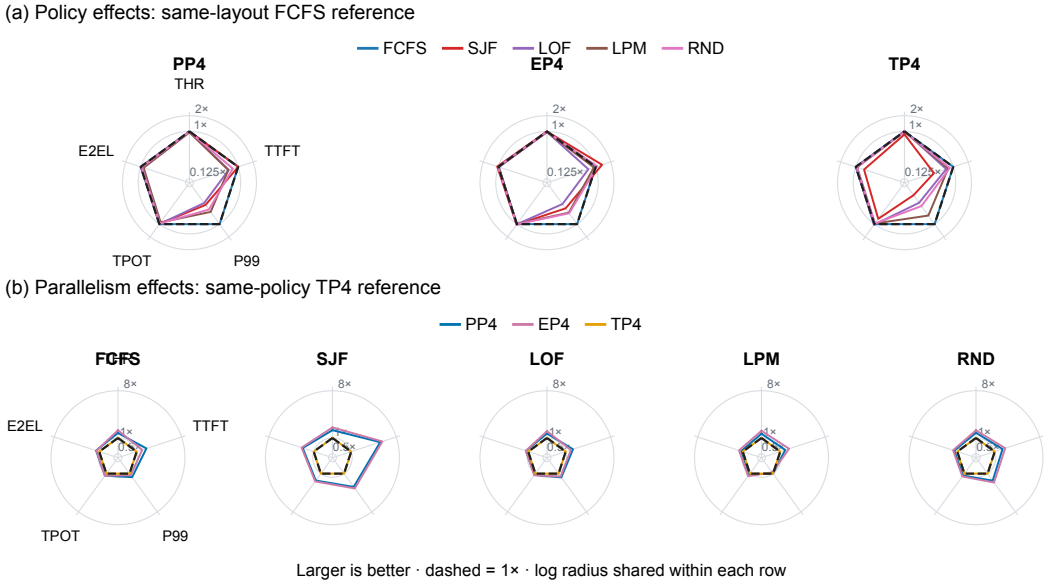


Figure 33: Q3.5-35B, Trace B, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

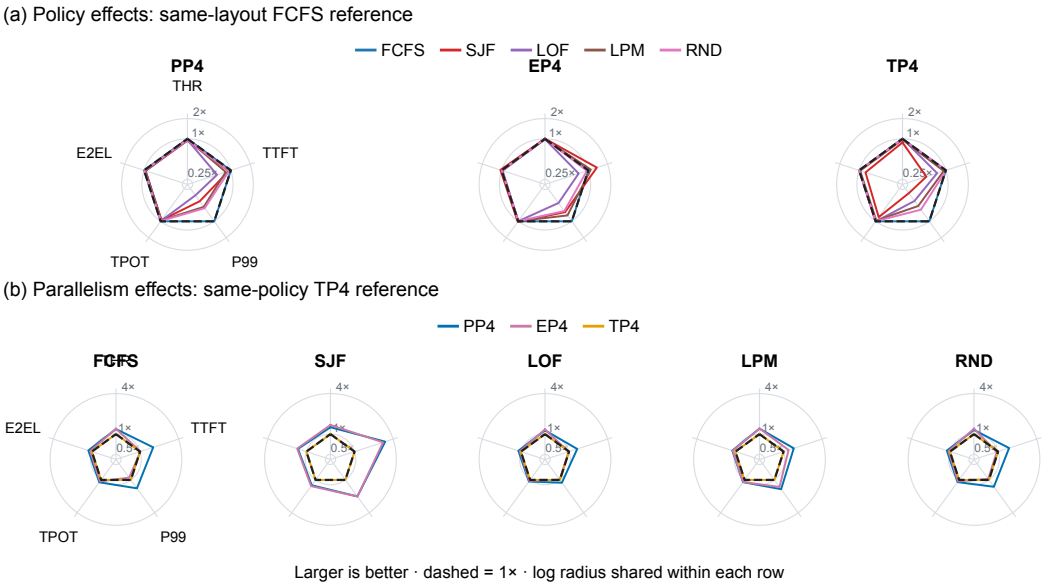


Figure 34: Q3.5-35B, Coder, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

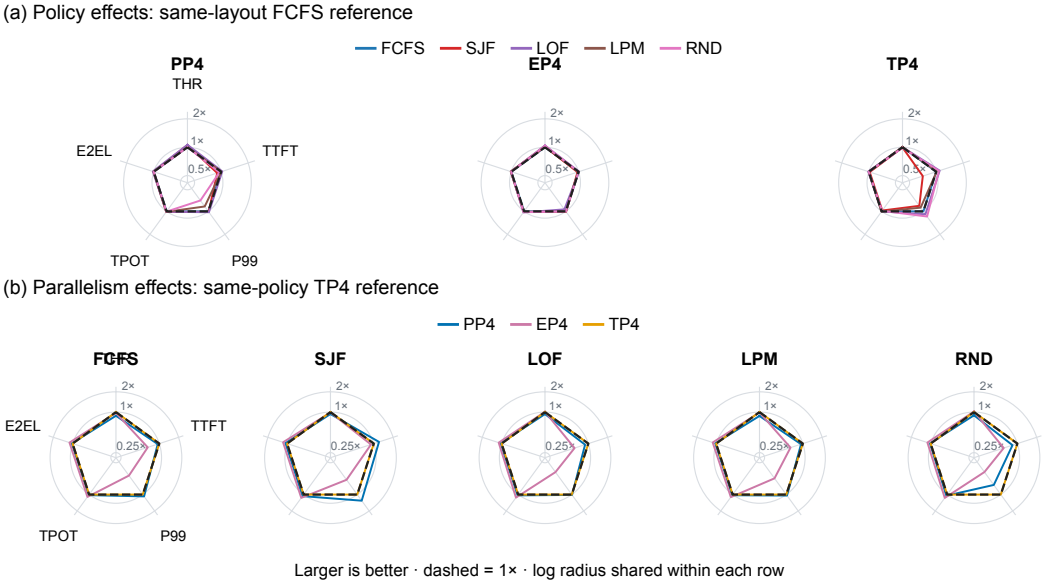


Figure 35: Q3.5-35B, Thinking, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

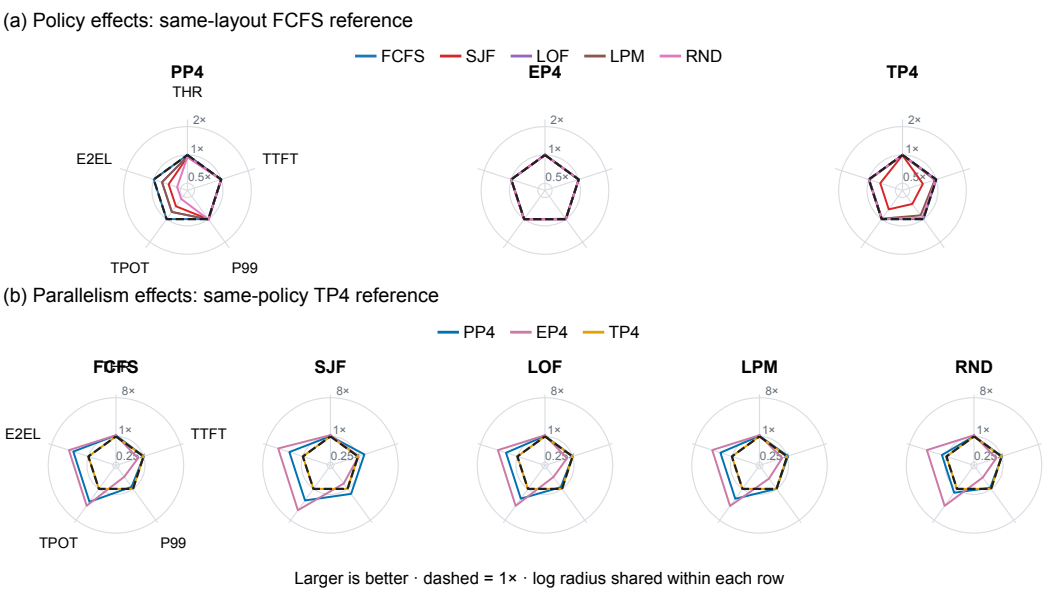


Figure 36: Q3.5-35B, Trace A, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

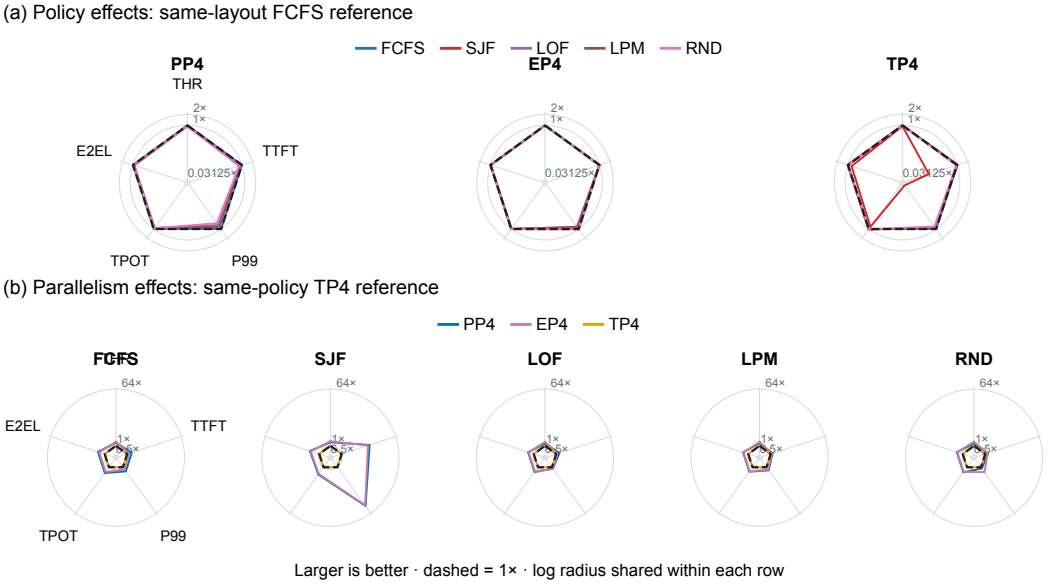


Figure 37: Q3.5-35B, Trace B, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

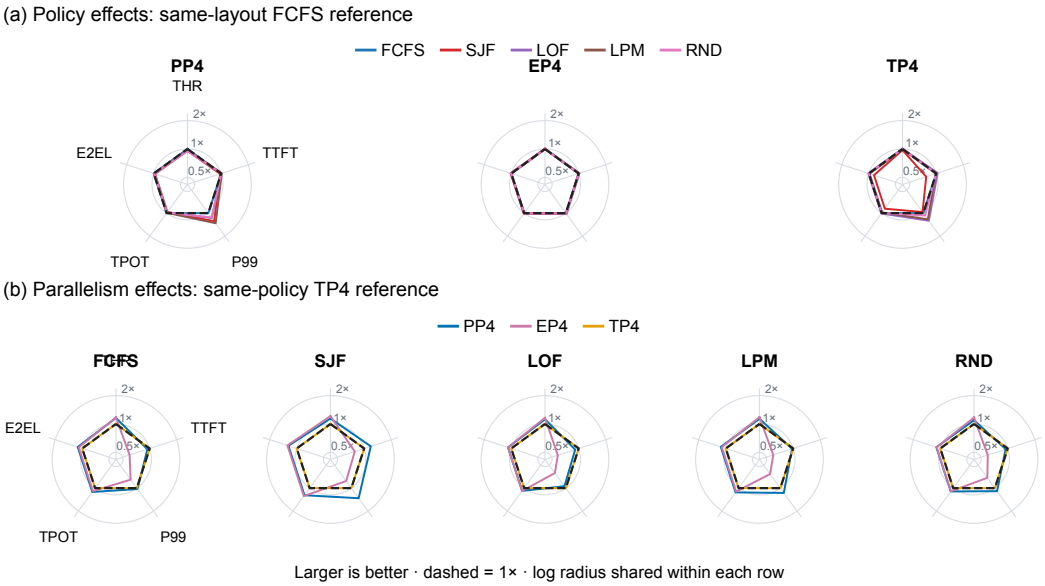


Figure 38: Q3.5-35B, Coder, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

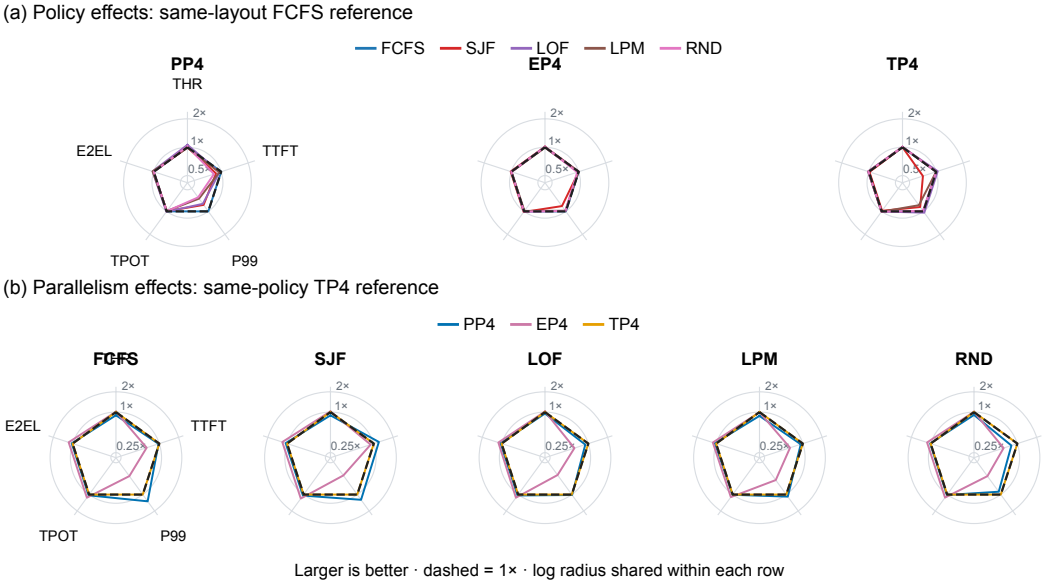


Figure 39: Q3.5-35B, Thinking, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

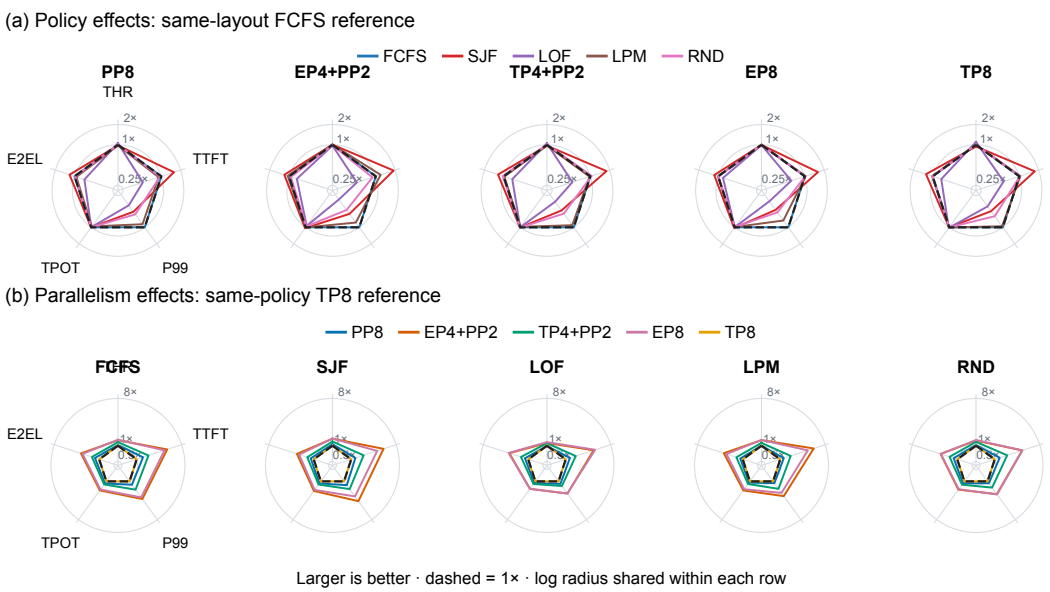


Figure 40: Q3.5-122B, Trace A, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

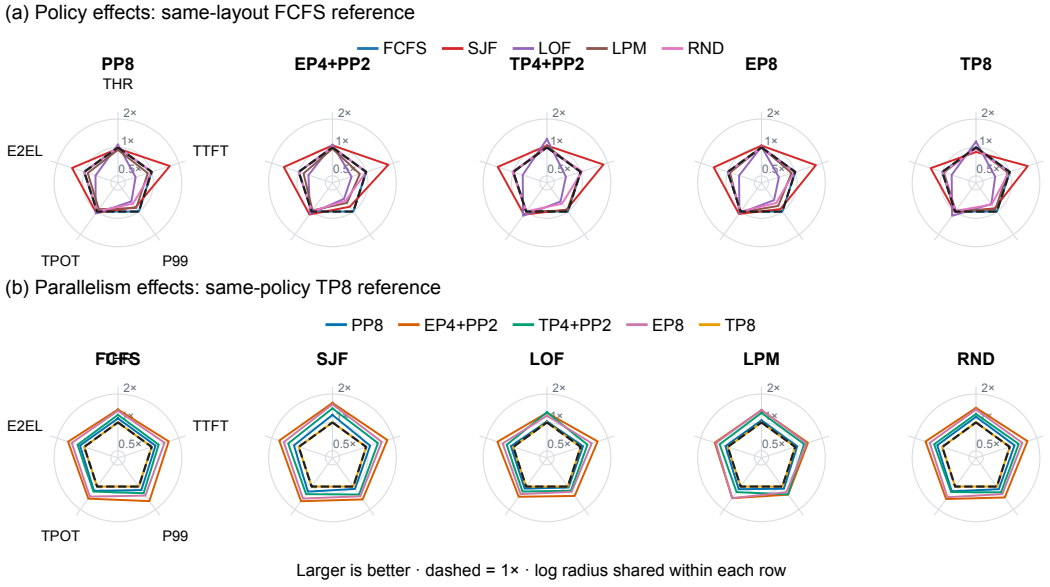


Figure 41: Q3.5-122B, Trace B, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

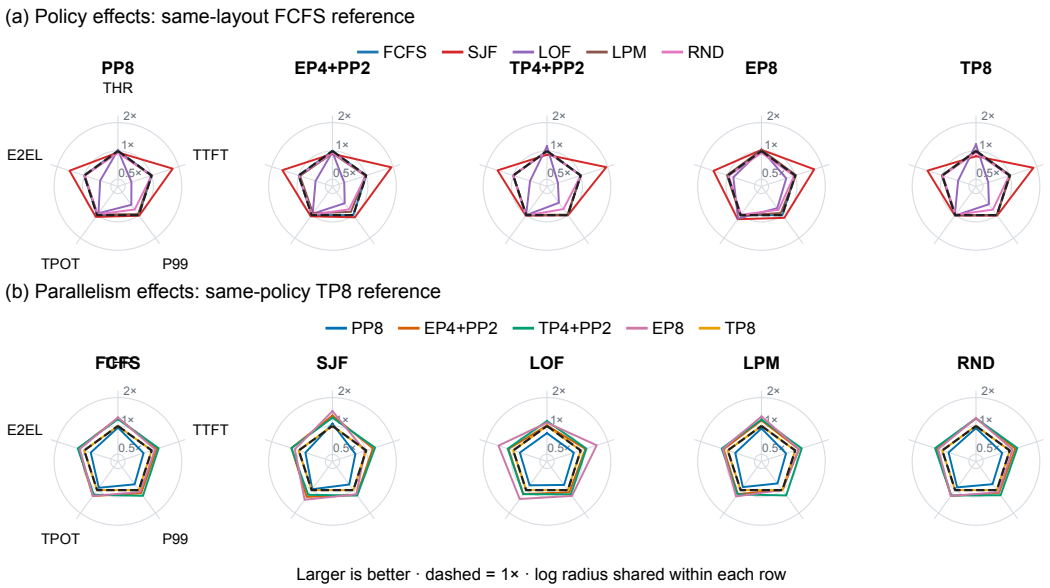


Figure 42: Q3.5-122B, Coder, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

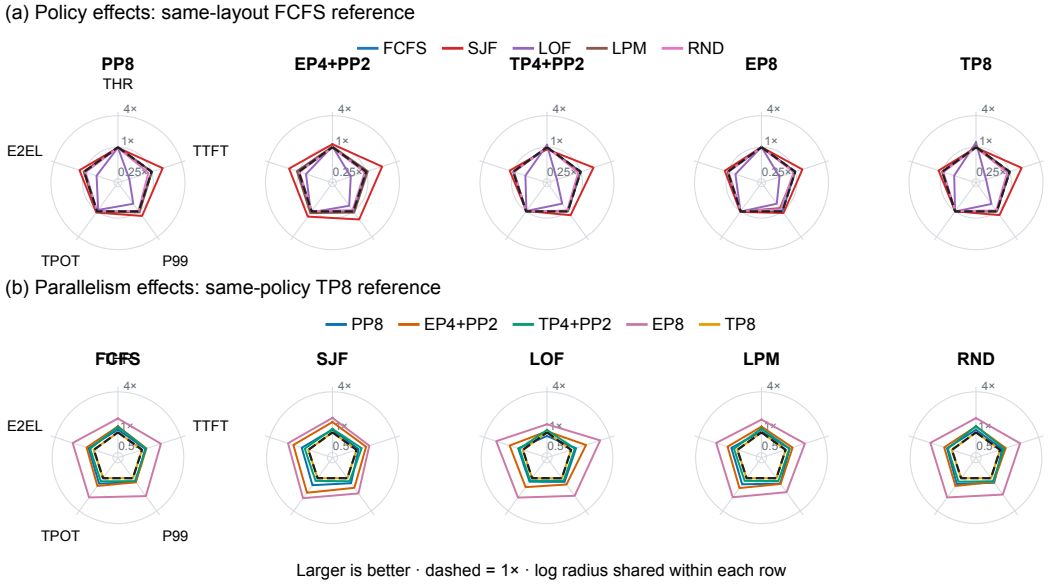


Figure 43: Q3.5-122B, Thinking, arrival scale 0.6: policy effects (top) and parallelism effects (bottom).

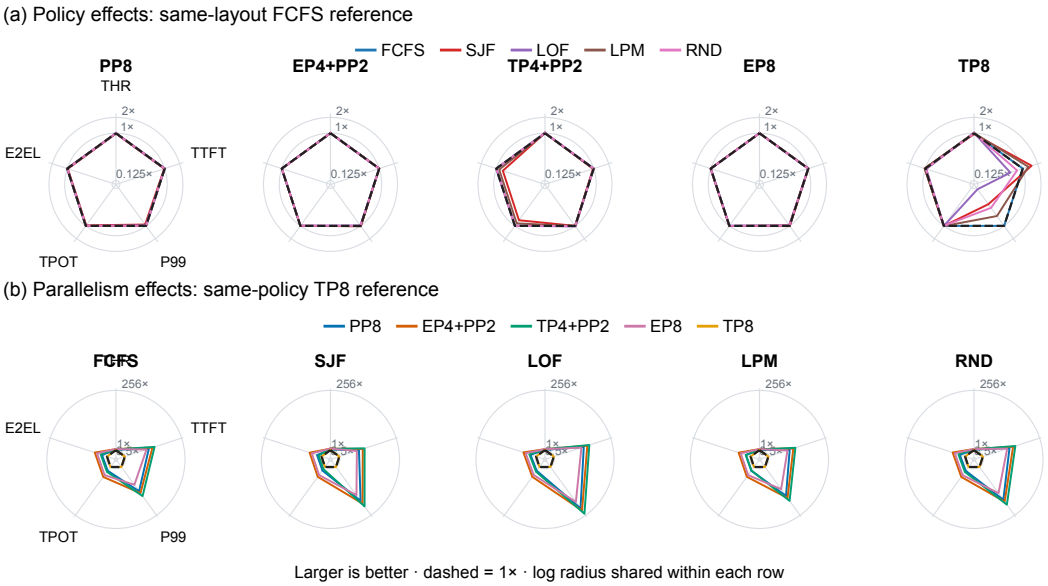


Figure 44: Q3.5-122B, Trace A, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

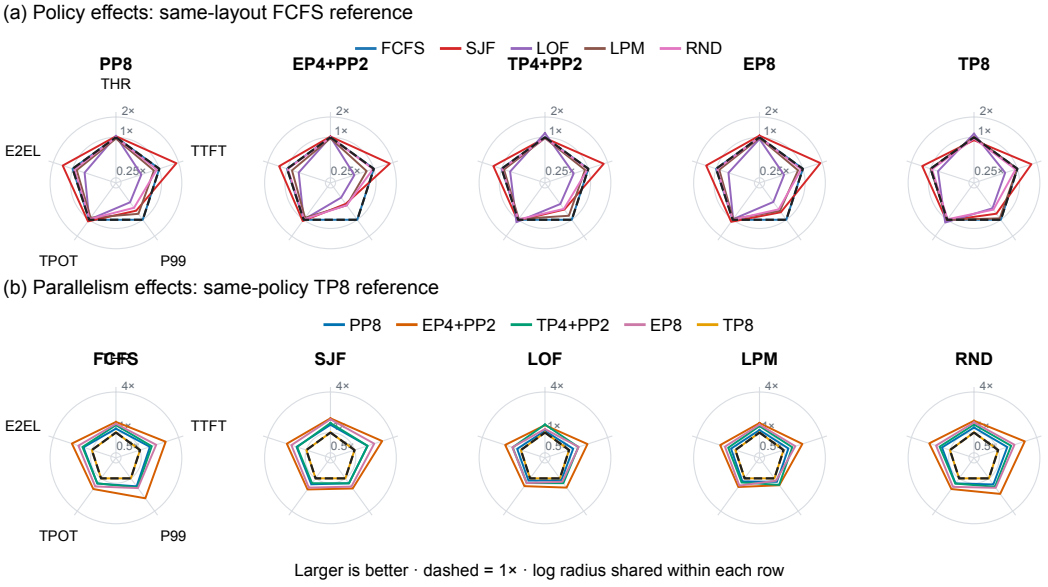


Figure 45: Q3.5-122B, Trace B, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

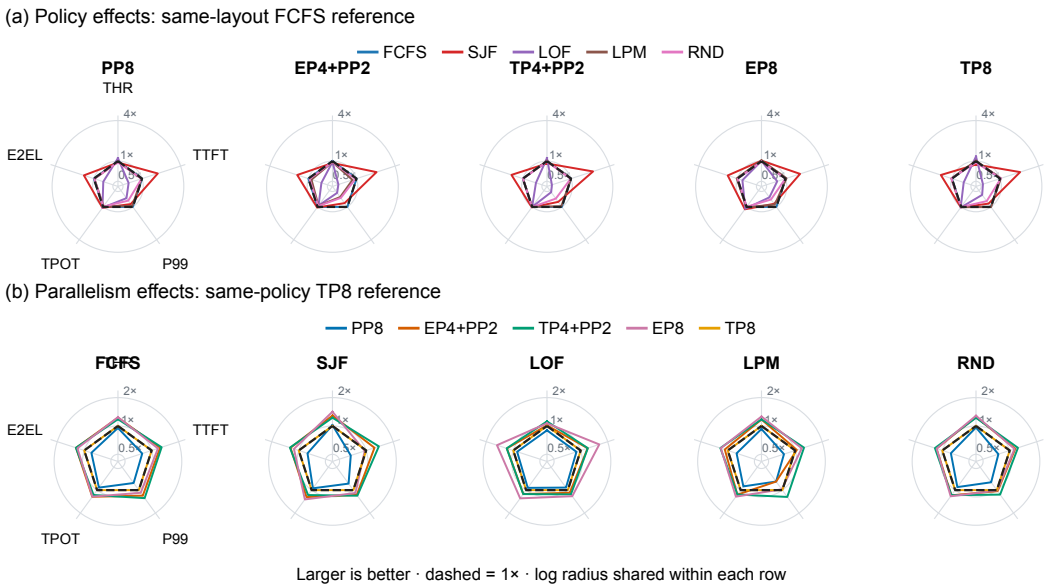


Figure 46: Q3.5-122B, Coder, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).

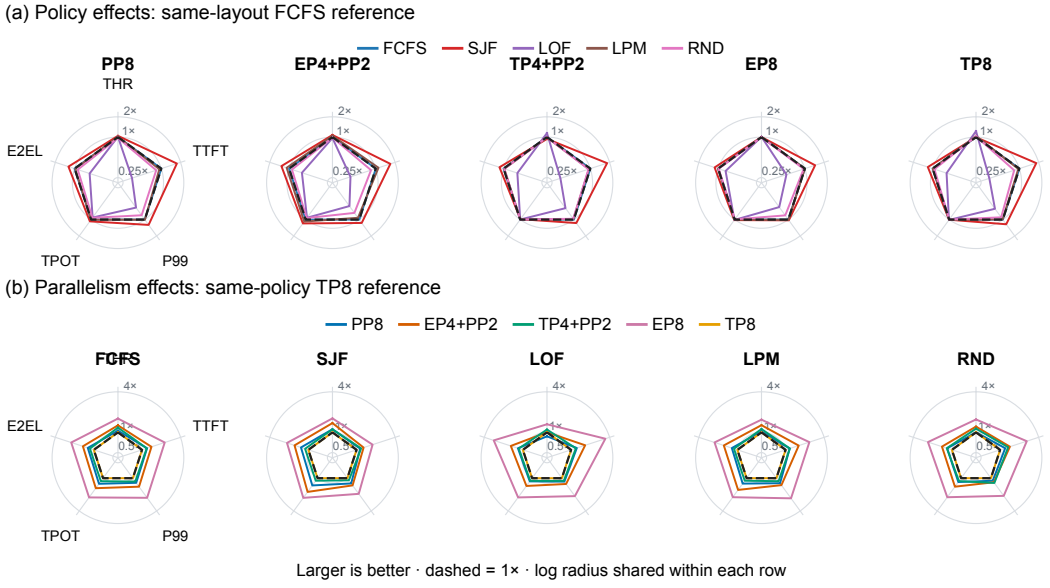


Figure 47: Q3.5-122B, Thinking, arrival scale 1.0: policy effects (top) and parallelism effects (bottom).