感触で始め、データで決める
ローカルLLMハードウェア設計講座
-ベンチマーク編-）

HPCシステムズ株式会社
証券コード：6597（東証グロース）

HPC SYSTEMS
People and Computing Power

HPC事業部
FAEグループ　越前谷靖幸
2025/09/25

注意事項

本資料は、ローカルLLMの導入を検討する初心者の方を対象として作成されたものです。

記載された情報は2025年9月時点のものであり、技術や製品の更新により将来的に内容が変わる可能性があります。

本資料を参考にした構成や運用において不具合が発生した場合、作成者は一切の責任を負いかねますので、自己の責任にてご利用ください。

このセミナーのゴール

- チャットボット/社内RAG向けの構成を、実測**数値（OIlama/vLLMのベンチマーク）で根拠づけることができるようになる**
- このGPU構成にした理由を数字で説明できるようになる
- LLM推論ベンチマークの基本設計がわかる
- システムベンダー、ハードウェアベンダー等と共通の認識、言語で仕様打ち合わせが可能になる

ベンチマークの意義

なぜこの構成にしたのか？に数字で答えるため

実運用に近い条件で実際に測る以外に正解がわからないのでベンチマークを行うべきなのです。
ポイントとしては3点あると考えています。

✅自社に最適化
汎用的なランキングではなく、自社にとっての正しさに整合

✅安全帯を見つける
高スコア競争より、破綻しない運用領域の特定

✅スケールの方向性を決める
GPUを強化するのか増やすのか、ノードを分けてアクセスを分散させるのか（コストの見極め）

ベンチマークは勝敗ではなく、設計の根拠。安全帯を見つけ、**最小コストの拡張経路**を選ぶ方法です。

用語補足：ベンチマーク結果で主に注目する数値

TTFT（Time To First Token）
最初の文字が出るまでの時間（ms）。初動の体感速度。
影響：入力前計算（Prefill）、スケジューラ、I/O。
目安：チャット≦2–3s、RAGでも≦3–5s。

TPOT（Time Per Output Token）
出力1トークン生成にかかる平均時間（ms/token）。デコードの純粋な速さ。
関係式：TPS ＝ 1000 / TPOT。
使いどころ：GPU比較の大きな指標（p50/p99を見る）。

ITL（Inter-Token Latency）
隣り合う出力トークンの間隔（ms）。ストリーミングの滑らかさ＝体感。
備考：ネットワークやホストの影響も乗るのでTPOTより大きく出がち。

QPS（Queries Per Second）/ リクエストスループット
単位時間あたりの完了リクエスト数（req/s）。同時ユーザ耐性の目安。
備考：負荷条件（同時実行・バースト・入出力長）を揃えない比較はNG。

用語補足：ベンチマーク結果で主に注目する数値

指標	意味（要約）	単位	利く場面	目安（例）
TTFT	最初の文字が出るまでの時間	ms や s	チャットやRAGの初動体感	チャット 2～3s RAG 3～5s
TPS	トークンの生成速度	tok/s	運用時の速さ	大きいほど速い （例：200 tok/s）
TPOT	出力1トークンの生成時間	ms/token	GPUの芯の速さ（デコード性能）	TPOT（ms）=1000/TPS
ITL	連続トークン間隔	ms	ストリーミングの滑らかさ（体感）	小さいほど自然 （80 ～120ms未満）
QPS	完了リクエスト数/秒	req/s	同時ユーザー運用耐性（運用）	条件を揃えた比較が必須（長さ/並行/バースト）

用語補足：p99（99パーセントタイル）とは

遅延を小さい順に並べたときの上位1%境界（遅い側の数値 / 100件なら99番目に遅い数値） です。

意味：
　システムの混雑時やバースト時の現実的な数値です。p50（中央値）が速くてもp99が遅いと不満が高まります。

使い方：
　システムの運用の限界（破綻しない限界）を探す場合はp95-99の数値で判断します。

※平常（p50）・混雑（p95）・ピーク（p99）の三段で評価

用語補足：入力前計算（Prefill）と生成（Decode）

いつ起こる？	**入力前計算（Prefill）**：最初に1回だけ	**生成（Decode）**：文字を出すたび毎回
何をしてる？	入力文（質問や資料）をまとめて読み込む	次の文字を1つずつ作る（KVキャッシュ利用）
どこに効く？	**出だしの速さ＝TTFT** total token throughput、**p99_TTFT**	**出力の速さ＝TPS/TPOT** **滑らかさ＝ITL**、p99_tpot
重くなる条件	**入力が長い**（RAGで資料を多く入れる等）	**出力が長い／同時利用が多い**
まずの対策	入力を絞る・要約する・重複を減らす	出力長の上限、量子化/並行度の見直し

長い入力 → Prefillが支配（出だしの速さを見る）

長い出力/多重アクセス → Decodeが支配（出力の速さ・滑らかさを見る）

Ollama と vLLMの位置づけ

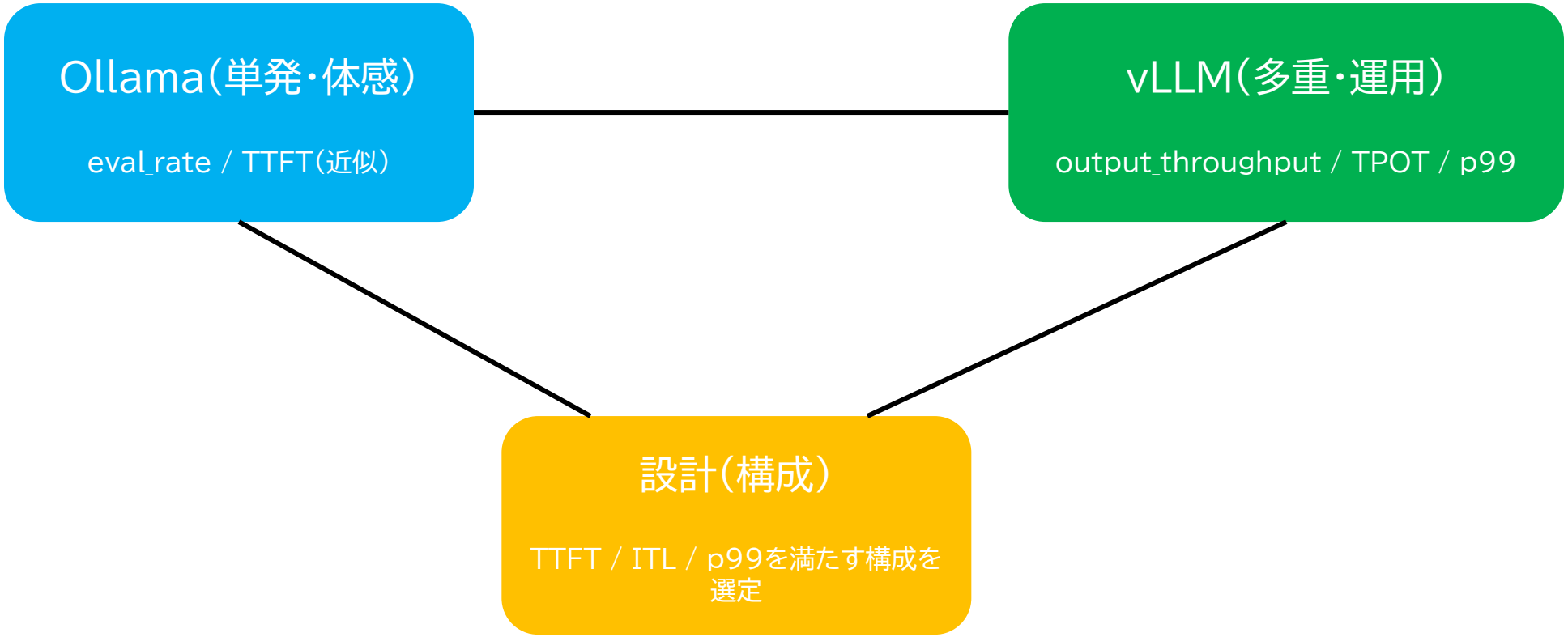
Ollama は、「感触」でモデルやGPUの相性を確かめるためのツールです。
単一リクエストで実行するので「使用したいモデルがこのGPUで動くのか」「GPUの単純な比較」といった直感的な手応えを得るには最適です。

vLLM は「運用の構成」を検討するために使うツールです。
同時実行時のスループットやレイテンシといった数値をもとに、実際にどのようなシステムを組むべきか判断することができます。

技術選定というのは、まず「試して感じる」段階と、その後「測って決める」段階があります。
Ollama で得られるのは、モデルの応答やGPUとの相性といった主観的な手応えです。
一方、vLLM がもたらすのは、スループットやレイテンシのような客観的な構成判断の基準です。

まずは「感触」として目的のLLMが動くGPUを選定し、次に同時実行時のデータで構成やアップデートプランを決める。
これが、LLM運用のハードウェア選定の流れだと考えています。

Ollama と vLLMの位置づけ



Ollamaのベンチマーク

Ollama自体にはベンチマーク機能やプログラムなどは含まれていませんが
verboseオプションを使うこと取得できる数値があります。

例:ollama run モデル名 --verbose
推論処理を行うとログに数値が出力されます。

```
total duration:       44.4774043s
load duration:        51.8639ms
prompt eval count:    70 token(s)
prompt eval duration: 1.3674558s
prompt eval rate:     51.19 tokens/s
eval count:           163 token(s)
eval duration:        43.0572694s
eval rate:            3.79 tokens/s
```

Ollama --verbose で取得できる指標

指標	意味(要約)	補足 / 計算式	利く場面
total_duration	リクエスト開始から最終トークンまでの壁時計時間(ロード/評価/生成/I/Oすべて込み)	内訳の和より長く見えることあり(待ち行列・ストリーム開封・JSON処理等のオーバーヘッドを含む)	エンドツーエンドの体感速度・SLA確認
load_duration	モデル/重みの読み込み(メモリ/VRAMマップ)に要する時間	初回は大きく、その後はキャッシュで短縮/ゼロ近く	コールドスタート対策・常駐/ウォームアップ判断
prompt_eval_count	入力プロンプト側のトークン数(KVキャッシュ構築対象)	長いほど初期応答が遅くなりやすい	コンテキスト量の管理・テンプレ軽量化
prompt_eval_duration	最初のトークンを出すまでの前処理時間(TTFTの主要因)	参考:**prompt tokens/s = prompt_eval_count ÷ prompt_eval_duration**	初期応答の速さ(TTFT)の最適化
eval_count	生成された出力トークン数	出力が長いとコスト/時間増	出力量の管理(要約・語数制限)
eval_duration	生成フェーズに要した時間(プロンプト評価除く)	量子化/GPU/スレッド設定で影響	生成スループット最適化
evals/s(tokens/s)	生成スループットの目安	**eval_count ÷ eval_duration**	ハードウェア/設定比較、最適化効果の評価
(備考)合計が合わない	total_duration が load + prompt_eval + eval より長く見えることがある	待機/I/O/ストリーム送出/ミドルウェア処理が含まれるため	内訳以外のオーバーヘッド把握

Ollama --verbose の読み解きの例(チャットボット向け)

📘 優先して確認したい指標

TTFT(最初のトークンまで)= 主に prompt_eval_duration
ここが短いほど「返事がすぐ始まる」体感が良い。
※OllamaではTTFTを直接計測できません。
コールドスタート時は load_duration も影響(初回のみ長くなりやすい)。

生成スループット = eval_count / eval_duration(tokens/s)
会話がスラスラ出るかの体感に効く。

📘 補助的に確認したい指標

prompt_eval_count

プロンプト側のトークン数。増えると prompt_eval_duration が比例して伸びやすい(TTFT悪化)。
total_duration と内訳(load + prompt_eval + eval)の差
差が大きい場合、I/Oや待ち行列、ストリーミング処理のオーバーヘッドが疑わしい。

Ollama --verbose の読み解きの例(チャットボット向け)

📘 典型的なパターンの読み解き

計測結果	主な原因	まず打つ対処
TTFTだけ遅い / tokens/s は十分	プロンプト肥大(**prompt_eval_count**↑)、システムプロンプト/履歴の積み上がり、KV未活用	履歴圧縮、システム/ツール定義の簡素化、(RAGでなければ)要約でコンテキスト化、**num_ctx**に合わせたトークン管理
tokens/s が低い(eval_duration が長い)	量子化/CPU-GPU/スレッド設定のミスマッチ、温度・ペナルティで探索が重い	軽い量子化へ※品質トレードオフ、GPU/スレッド最適化 出力長(**max_tokens**)を抑制
初回だけ極端に遅い	**load_duration**(モデル読み込み)	モデル常駐・ウォームアップ、 よく使うモデルの集約、 不要モデルの整理

Ollama --verbose の読み解きの例(社内ドキュメント検索など/RAG向け)

RAGは「検索～前処理(分割・埋め込み・再ランク)～プロンプト組み立て～生成」という複合パイプラインです。
Ollama の数値は“生成エンジン内”の区間に限られる点に注意してください。

📘 優先して確認したい指標

コンテキスト負荷の指標:prompt_eval_count と prompt_eval_duration

取得したチャンクを詰め込みすぎるとTTFTが跳ね上がる。ここがボトルネックになりやすい。

生成コスト:eval_count と eval_duration(= tokens/s)

まとめ・要約・根拠提示など出力が長くなりがち。スループットはSLA(Service Level Agreement)に効く。

total_duration −(load + prompt_eval + eval)

ここに「アプリケーション側のI/Oやストリーム開封等のオーバーヘッド」が含まれる。
※実際の「検索時間」はOllama外なので、アプリケーション側で別計測を併用するのが必須。

Ollama --verbose の読み解きの例(社内ドキュメント検索など/RAG向け)

📘 典型的なパターンの読み解き

計測結果	主な原因	まず打つ対処
TTFTが不安定／遅い & prompt_eval_count が大きい	コンテキスト過積載(Top-k過多、長大チャンク、重複引用)	チャンク粒度の最適化(モデルの **num_ctx** 目安で調整)／Top-k・コンテキスト上限の設定／重複除去(dedupe)・要約結合(merge & summarize)／メタデータでフィルタ(部門・日付・バージョン)
tokens/s は十分なのに total_duration が妙に長い	RAG前段(検索・再ランク・テンプレ組み立て)やネットワークが支配	区間別に個別計測しボトルネック特定(検索エンジン応答、ベクタDBの index/ANN パラメータ、再ランクモデルのバッチ化)／I/O・ネットワーク経路の見直し
回答が冗長で eval_count が膨らむ	プロンプト指示が広すぎる 引用方針が曖昧	出力スタイルを固定(箇条書き／最大語数／引用数・根拠URL本数の上限)／段階生成(まず要点→次に詳細)

Ollama --verbose の読み解きの例(社内ドキュメント検索など/RAG向け)

📘 数値に対する具体的なアクション例

計測結果	主な原因	まず打つ対処
prompt_eval_duration 高い / prompt_eval_count 多い	コンテキスト過積載	Top-k縮小、チャンク要約、重複除去、テンプレ軽量化
eval_duration 長い / tokens/s 低い	モデル重い・量子化不整合・GPU未活用	量子化とGPU最適化、スレッド/バッチ設定見直し、出力長制御
total_duration と内訳の差が大きい	前段RAG処理／I/O	検索・再ランク・テンプレ生成の個別計測と並列/キャッシュ
初回遅い(以後は改善)	load_duration	モデル常駐・ウォームアップ・モデル切替の削減

Ollama の最小KPIセット

🟦 チャットボット

prompt_eval_duration（TTFT）

eval_count / eval_duration（tokens/s）

補助：prompt_eval_count、load_duration

🟦 RAG

prompt_eval_count と prompt_eval_duration（コンテキスト負荷）

eval_count / eval_duration（生成スループット）

アプリケーション側で別途：検索～再ランク～プロンプト構築の所要時間（Ollama外）

vLLMのベンチマーク

Ollamaの単発計測では見えにくい「同時実行時のスループット／レイテンシ」を、vLLMの公式ベンチマークで
可視化できます、プログラムは複数種用意されていますが、大きく分けてオフラインとオンラインの2種になります。

オフライン・ベンチマーク（throughput / latency）
単発～バッチ処理の純粋なモデル性能を測る（I/Oやネットワーク、サーバー待ち行列の影響を排除
典型用途：GPUや量子化、テンソル並列の比較／モデル入替前の下見

オンライン・ベンチマーク（serve）
vLLMサーバー（vllm serve）を立て、クライアントから**多重アクセス**を発生させて実運用近似の数値を取る
典型用途：同時ユーザー耐性の上限（QPS）やp99遅延、安全帯の探索

直感：**オフライン＝Ollamaの--verboseに近い“単発の芯の速さ”、オンライン＝“本番運用での行列を含む速さ”。**

vLLMはgithubのプロジェクト内にベンチマークスクリプトを同梱しています。
https://github.com/vllm-project/vllm/tree/main/benchmarks

NVIDIA社のNVIDIA NIM LLMのベンチマークという資料に基づき
LLMのシステム設計に向けた各種データを取得することが可能です。
https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html

測定で見る指標(おさらい)

TTFT
最初のトークンまで

TPOT
出力1トークンあたり時間)/ TPS(= 1000/TPOT)

ITL
トークン間隔

QPS
完了要求/秒

p50/p95/p99
混雑時の現実的な遅延

使い分け:

オフライン:TPOT/TPS中心(Prefill/Decodeの素の速さ)

オンライン:QPS、TTFT_p99、TPOT_p99、ITL(行列込みの体感)

vLLM　オフライン・ベンチマーク

目的：サーバーを立てず、モデル推論コアの**スループット/レイテンシ**を測る

コマンド例：

```
# 例：ランダムデータセットでデコードのスループットを計測
python -m vllm.benchmarks.throughput ¥
--model meta-llama/Meta-Llama-3.1-8B-Instruct ¥
--dataset-name random ¥
--num-prompts 200 ¥
--random-input-len 512 ¥
--random-output-len 128

# 例：単一バッチのレイテンシ特性（Prefill/Decode）を観察
python -m vllm.benchmarks.latency ¥
--model meta-llama/Meta-Llama-3.1-8B-Instruct ¥
--input-len 1024 --output-len 256
```

読み解き方：
TPOT/TPS：GPUや量子化の“芯の速さ”比較
Prefillに効く：--input-len増→TTFT↑
Decodeに効く：--output-len増→TPS低下

vLLM　オンライン・ベンチマーク

目的：vLLMサーバーを起動し、**多重アクセス**下のQPS/レイテンシ（p50/p99）を測る

```
# サーバーを起動する
vllm serve meta-llama/Meta-Llama-3.1-8B-Instruct ¥
--host 0.0.0.0 --port 8000 ¥
--gpu-memory-utilization 0.90
```

```
#クライアントから負荷をかける
python -m vllm.benchmarks.serve ¥
--backend vllm ¥
--base-url http://127.0.0.1:8000 ¥
--endpoint /v1/chat/completions ¥
--model meta-llama/Meta-Llama-3.1-8B-Instruct ¥
--dataset-name random ¥
--num-prompts 1000 ¥
--max-concurrency 32 ¥
--request-rate inf ¥
--random-input-len 256 ¥
--random-output-len 128
```

vLLM オンライン・ベンチマーク

読み解き方：

QPS vs p99_TTFT/TPOTのトレードオフ曲線を確認

安全帯：
p95/p99がSLA閾値（例：TTFT≦3s、ITL≦120ms）を満たす同時実行数

ボトルネック切り分け：
Prefill支配：入力長を下げる／RAGのTop-k/重複を抑制
Decode支配：量子化/テンソル並列/バッチ戦略の見直し
I/O/ネットワーク：サーバー/クライアント間のRTT、CPU負荷、ストリーム処理

vLLM　オンライン・ベンチマーク（ドイル氏制作版）

vLLMのgithub上のベンチマークドキュメントに各種設定方法のガイドはあるのですが
複数のファイルを変更する必要があるなど手順が複雑です。

※簡易化し扱いやすくしたものをドイル・キム氏が公開しているのでそちらを使うと便利です。
GitHub - gjgjos/vllm_benchmark_serving: Benchmarking framework for evaluating vLLM serving performance across different configurations, models, and system settings. Includes tools for automated testing, latency/throughput analysis, and reproducibility.

vLLMのサーバーを起動するときに、最大コンテキスト数などを設定して起動後

使用しているモデル、トークナイザー、同時アクセス数、1アクセスあたりのプロンプト数を設定ファイルに記載し実行するだけです。

vLLM　オンライン・ベンチマーク

それでは実際にvLLMのオンライン・ベンチマークの結果を元に
チャットボット向け（コンテキスト数256）とRAGや長コンテクスト向け（コンテキスト数2048,4096）を想定して

QPS、TTFT、TPOT、ITLの各数値と同時リクエスト数（並列数）の変化
1GPU時と2GPU時の違い（GPUを増やすとどの程度の期待値が持てるのか）
安全に運用できる同時リクエスト数の判定などを見ていきましょう

ベンチマーク実施環境

システム

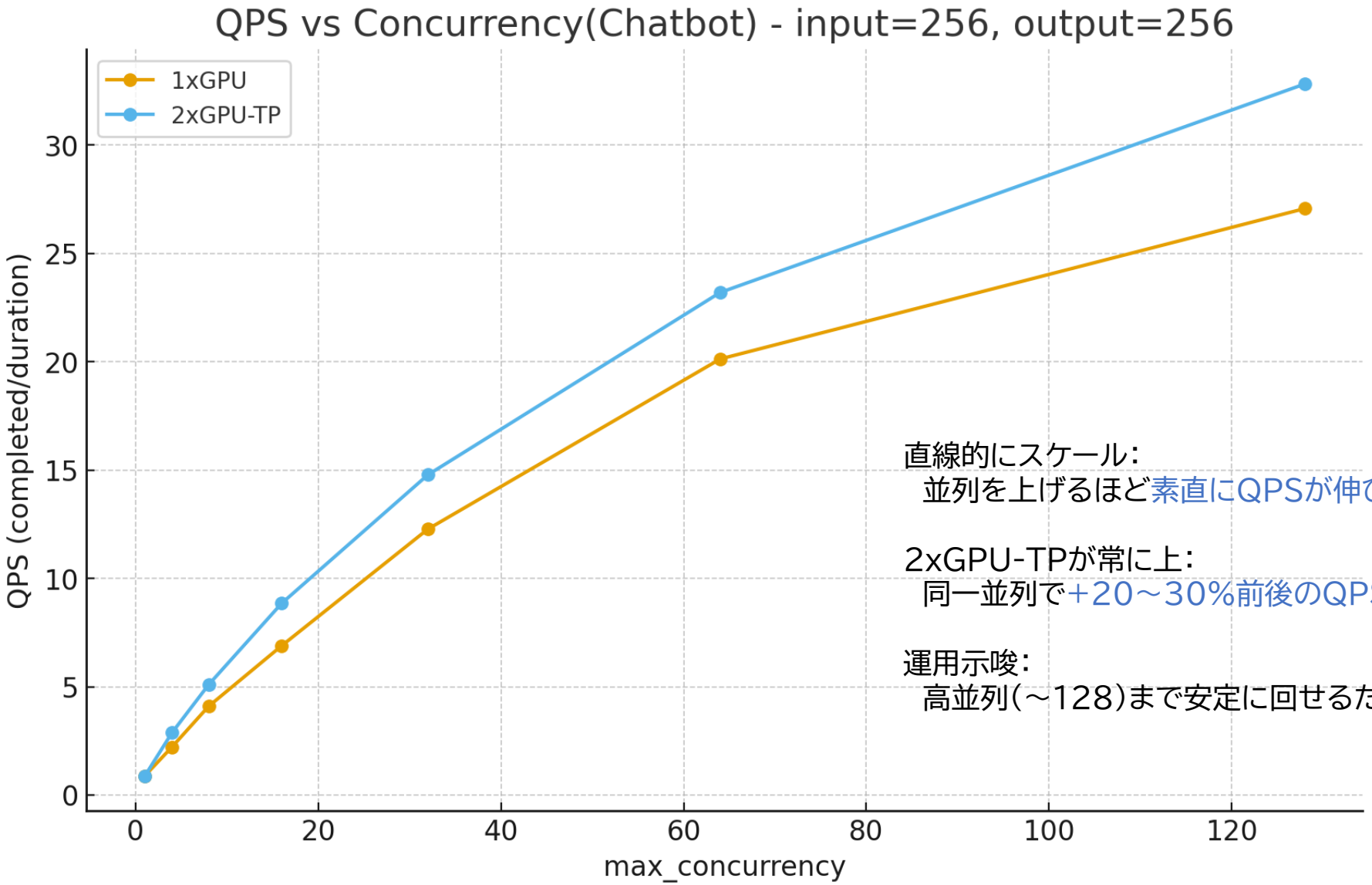
HPC3000-XSRGPU4TP-LC 水冷ワークステーション
CPU: インテル® Xeon® w5-3425 プロセッサー 3.2GHz 12core/24threads
メモリ:DDR5-4800 128GB(16GBx8)
SSD:1TB PCIe Gen4 NVMe
GPU:NVIDIA RTX 5090x2(水冷)
OS:Ubuntu 24.04.1 LTS
CUDA:12.8
vLLM: ver 0.10.1

モデル:gpt-oss 20B

GPUシングル運用時とマルチ運用時(Tensor Parallel)の2種で測定

安全運用の閾値
SLA(TTFT_p99≤3s & ITL_p99≤120ms)

QPS vs 並列数(チャットボット向け) - input=256, output=256

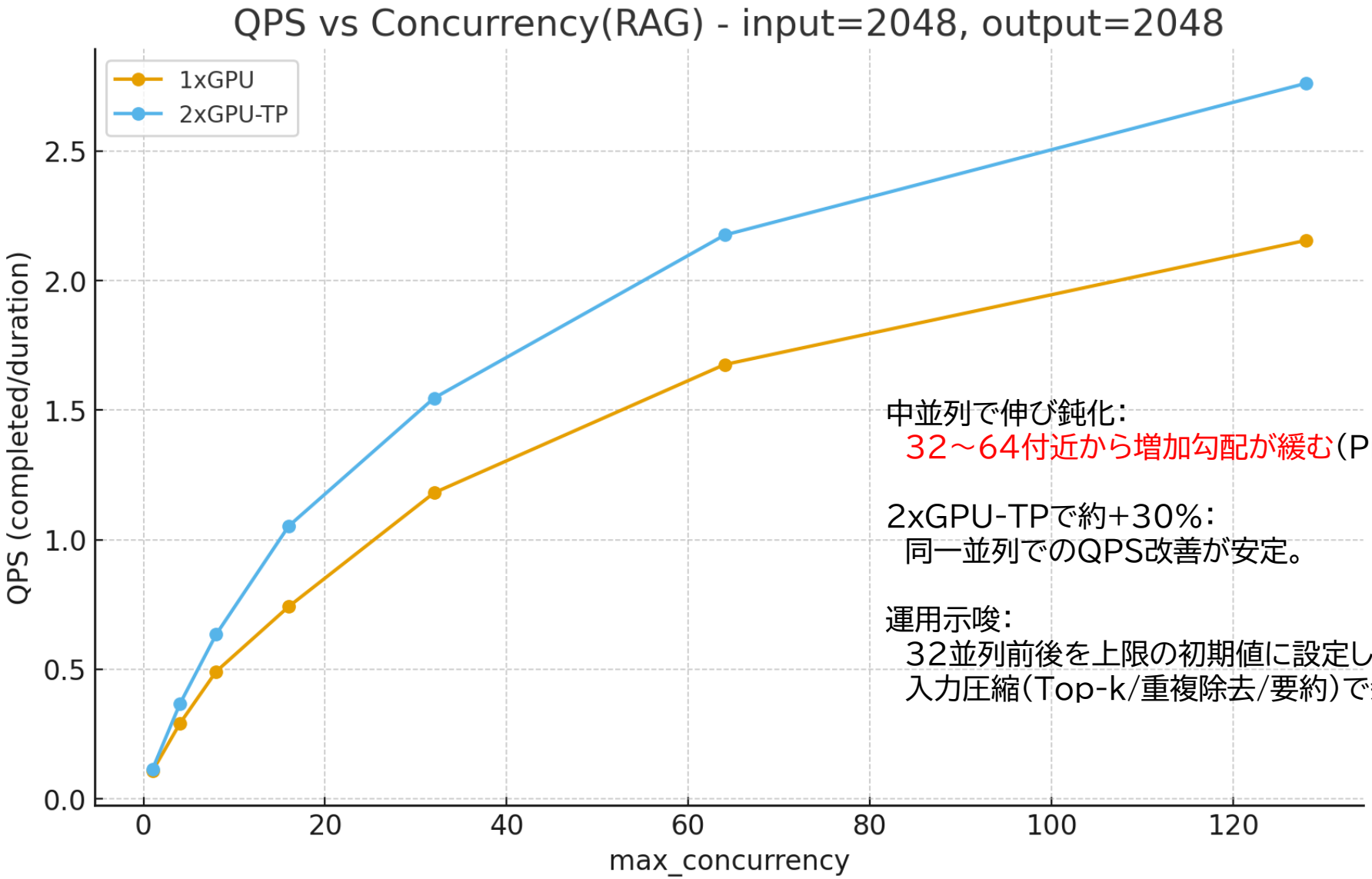


直線的にスケール:
並列を上げるほど素直にQPSが伸びる(頭打ち感は小)。

2xGPU-TPが常に上:
同一並列で+20～30%前後のQPS増。

運用示唆:
高並列(～128)まで安定に回せるため、レート制限緩和やバースト吸収に有利。

QPS vs 並列数(RAG向け) - input=2048, output=2048

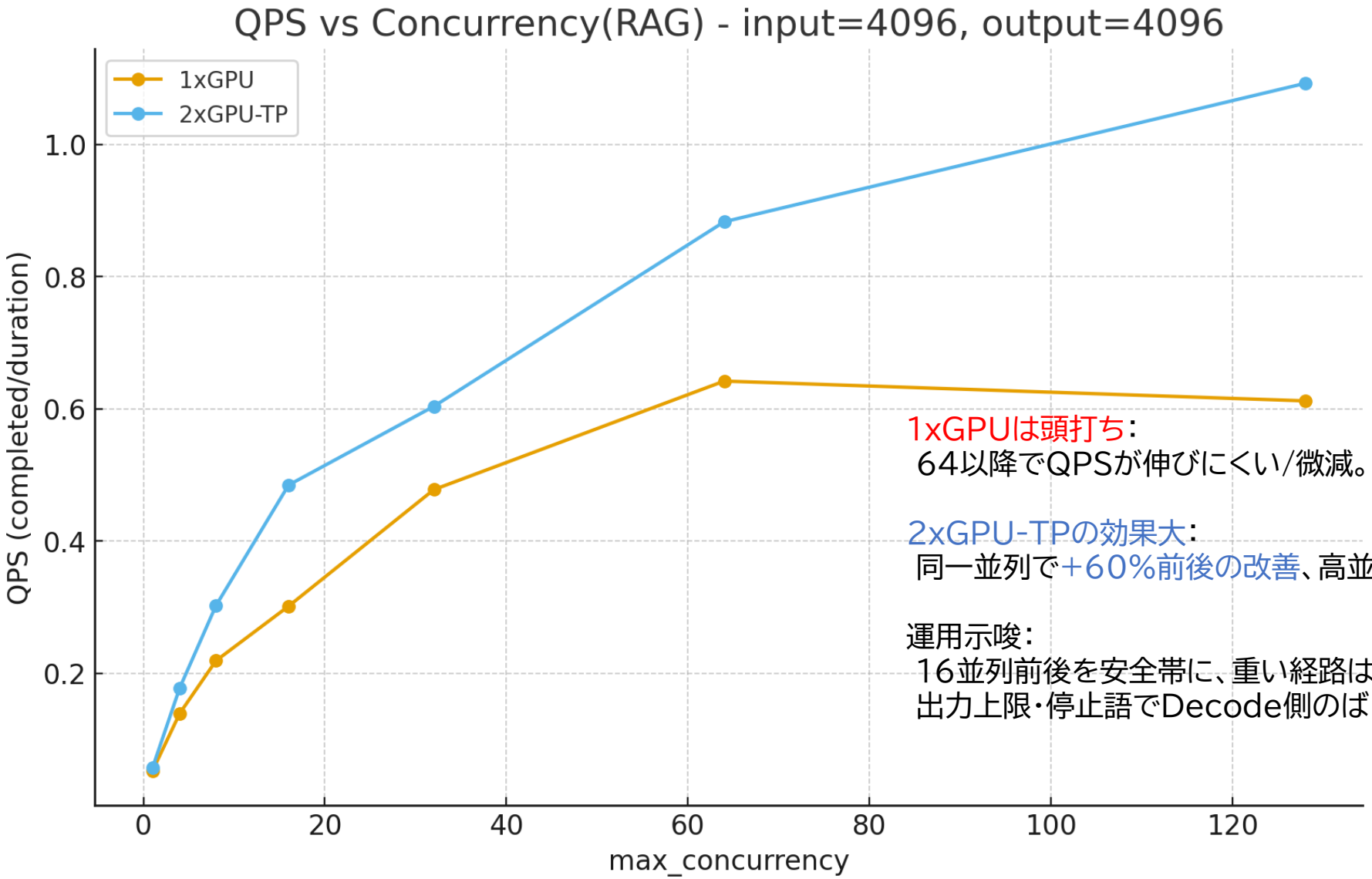


中並列で伸び鈍化:
32～64付近から増加勾配が緩む(Prefill負荷の影響)。

2xGPU-TPで約+30%:
同一並列でのQPS改善が安定。

運用示唆:
32並列前後を上限の初期値に設定し、
入力圧縮(Top-k/重複除去/要約)で余力を作る。

QPS vs 並列数(RAG/長コンテキスト向け) - input=4096, output=4096

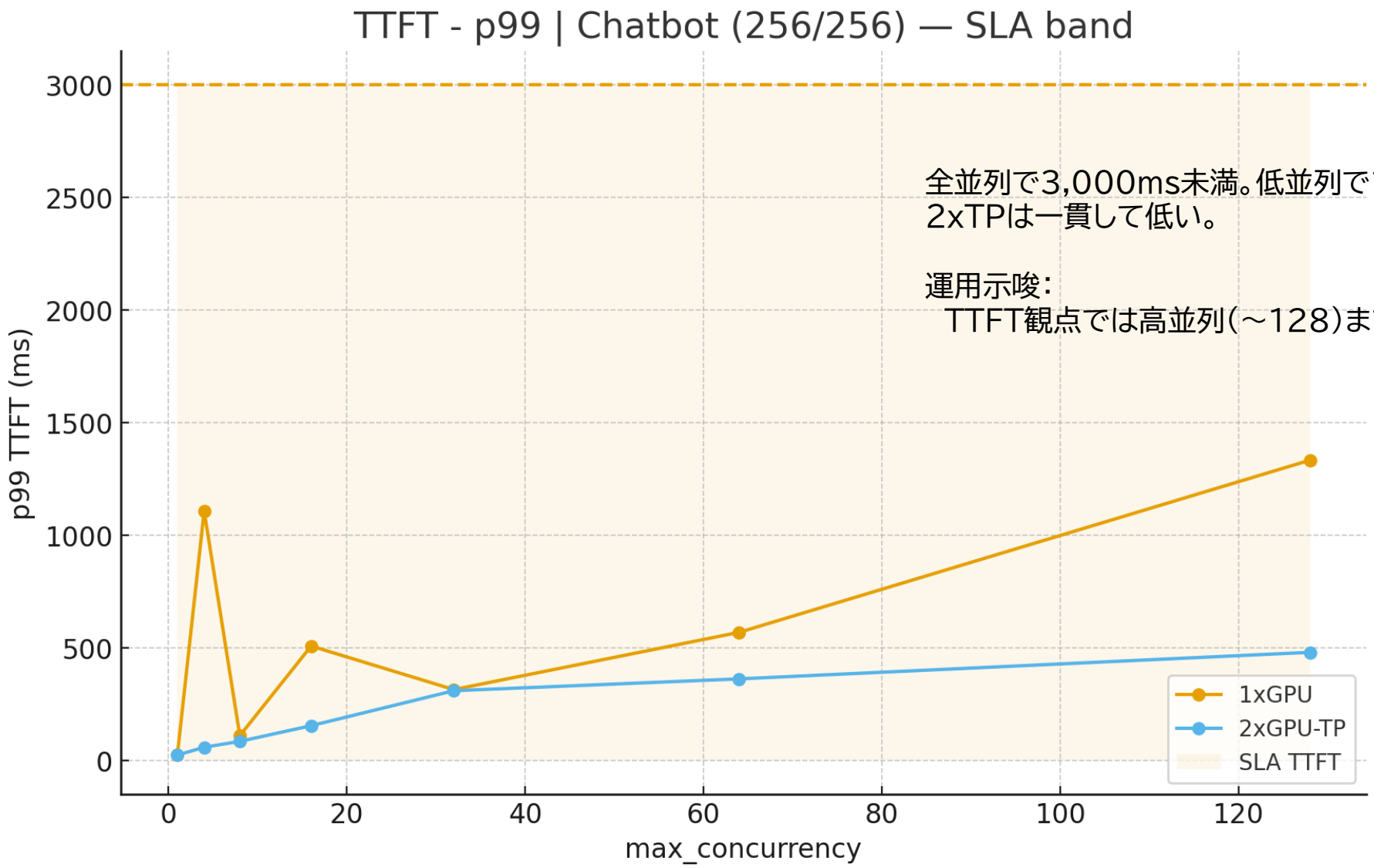


1xGPUは頭打ち:
64以降でQPSが伸びにくい/微減。

2xGPU-TPの効果大:
同一並列で+60%前後の改善、高並列でも上振れが続く。

運用示唆:
16並列前後を安全帯に、重い経路は2xGPU-TPノードへ集約。
出力上限・停止語でDecode側のばらつきを抑える。

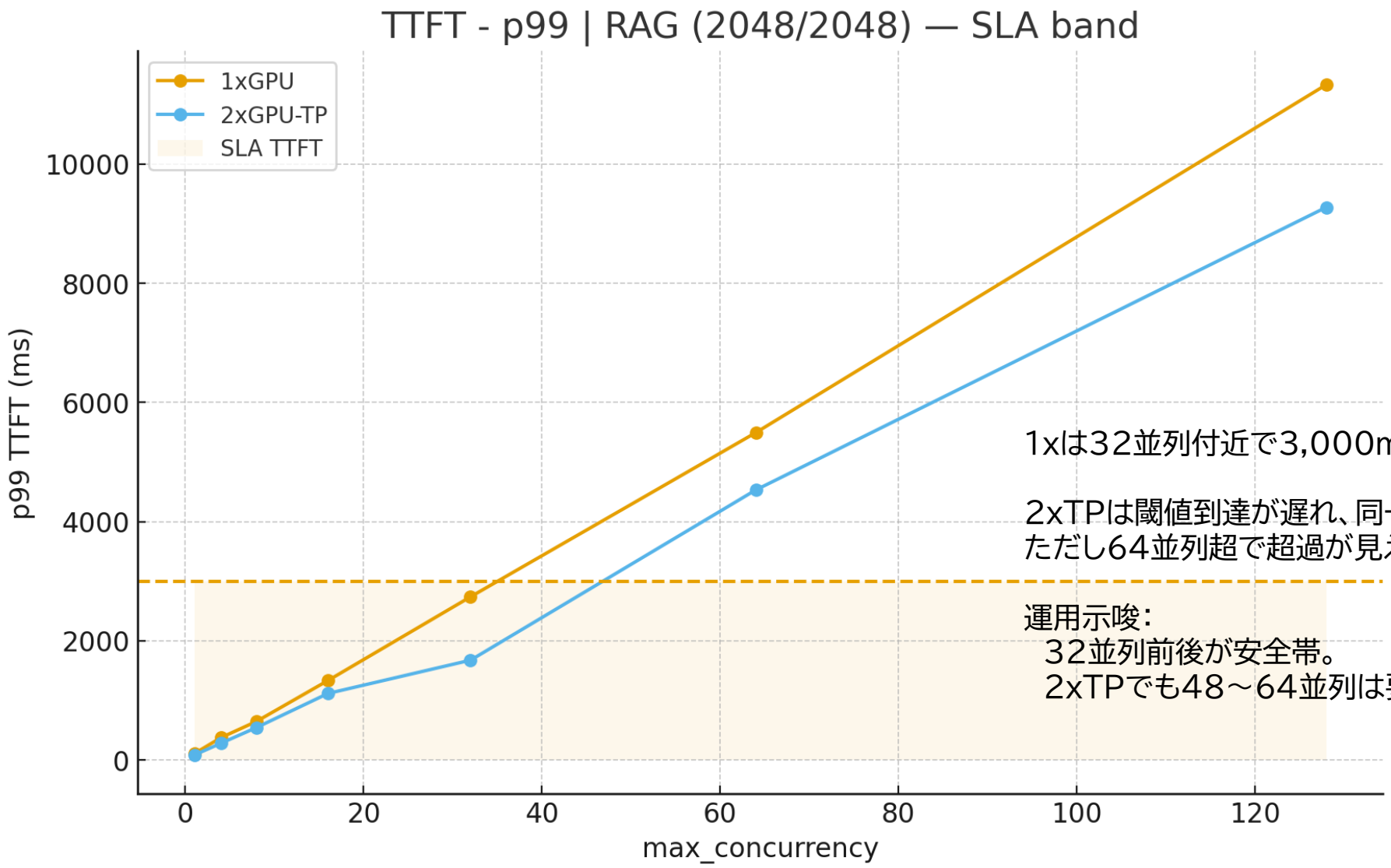
TTFT-P99（チャットボット向け） - input=256, output=256



全並列で3,000ms未満。低並列で1xにスパイクはあるが小さく、
2xTPは一貫して低い。

運用示唆：
TTFT観点では高並列（～128）まで安全。体感の初動は2xGPU-TPが安定。

TTFT-P99（RAG向け） - input=2048, output=2048

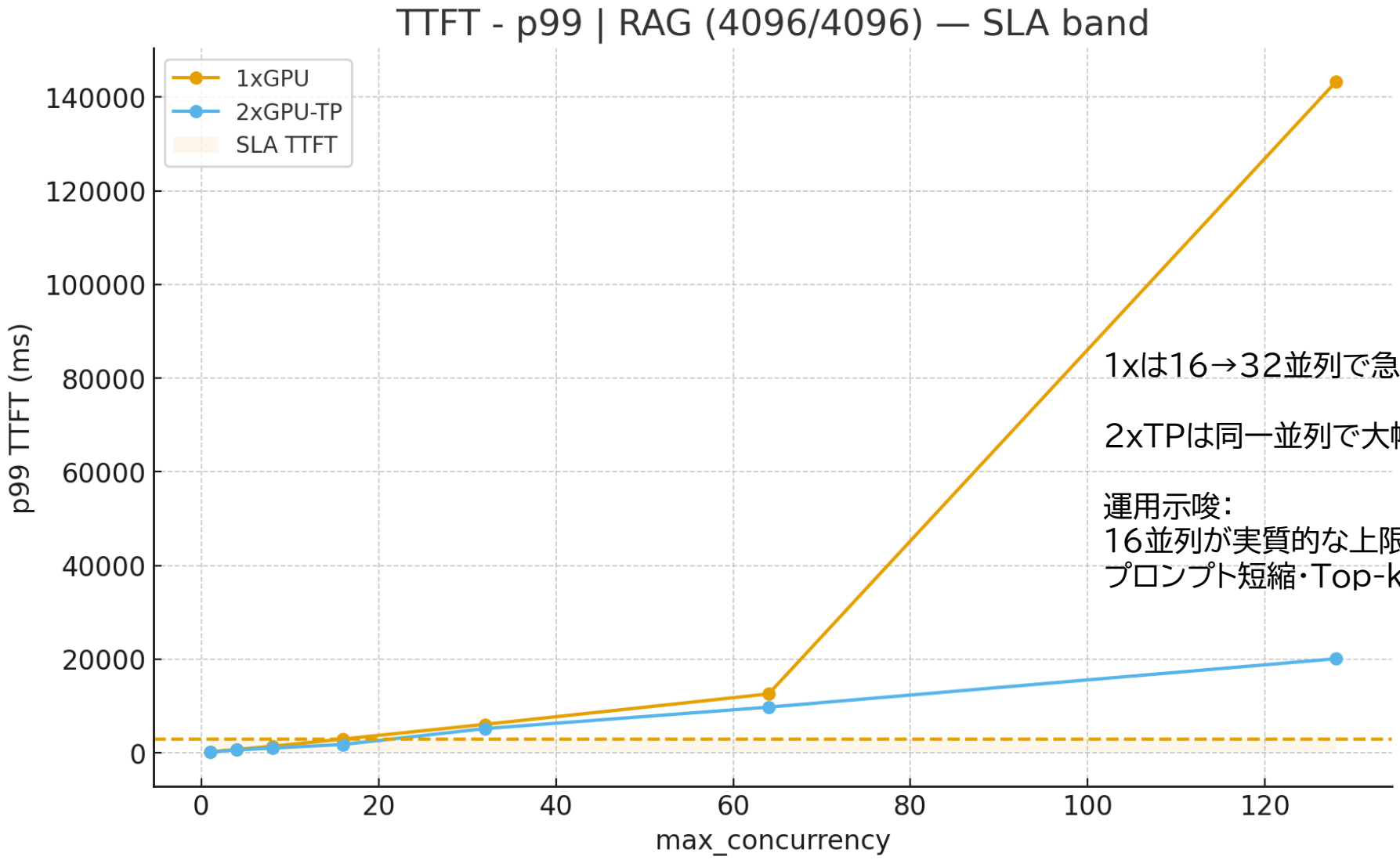


1xは32並列付近で3,000msに到達し、それ以上で超過。

2xTPは閾値到達が遅れ、同一並列で常に低TTFT。
ただし64並列超で超過が見え始める。

運用示唆：
32並列前後が安全帯。
2xTPでも48～64並列は要注意（入力圧縮で余裕づくり）。

TTFT-P99（RAG/長コンテキスト向け）– input=4096, output=4096

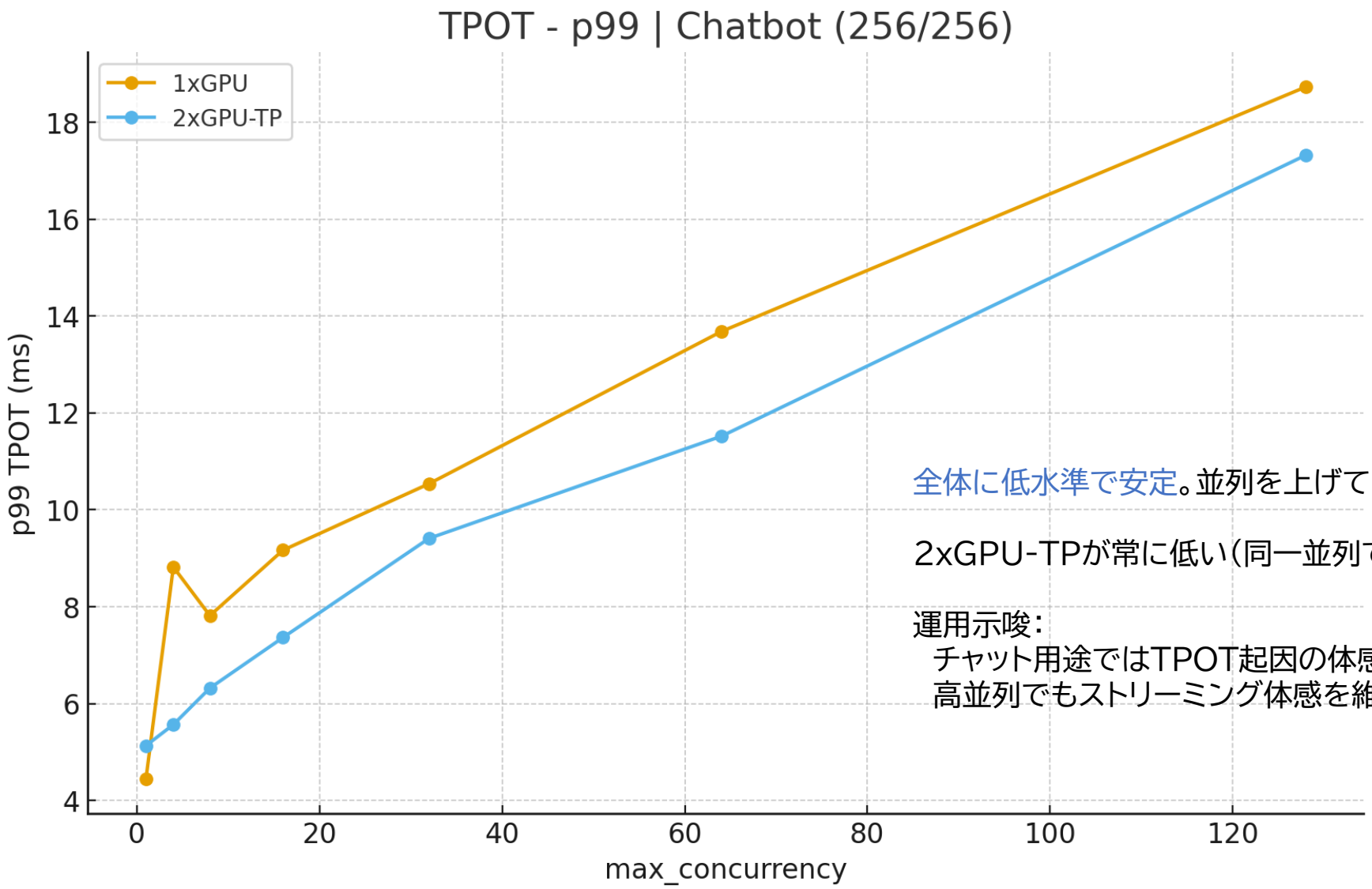


1xは16→32並列で急増し閾値超え。

2xTPは同一並列で大幅に低いが、32並列で閾値近辺～超過に。

運用示唆：
16並列が実質的な上限。重い経路は2xGPU-TPノードに寄せ、プロンプト短縮・Top-k抑制でTTFTを抑える。

TPOT-P99（チャットボット向け） - input=256, output=256

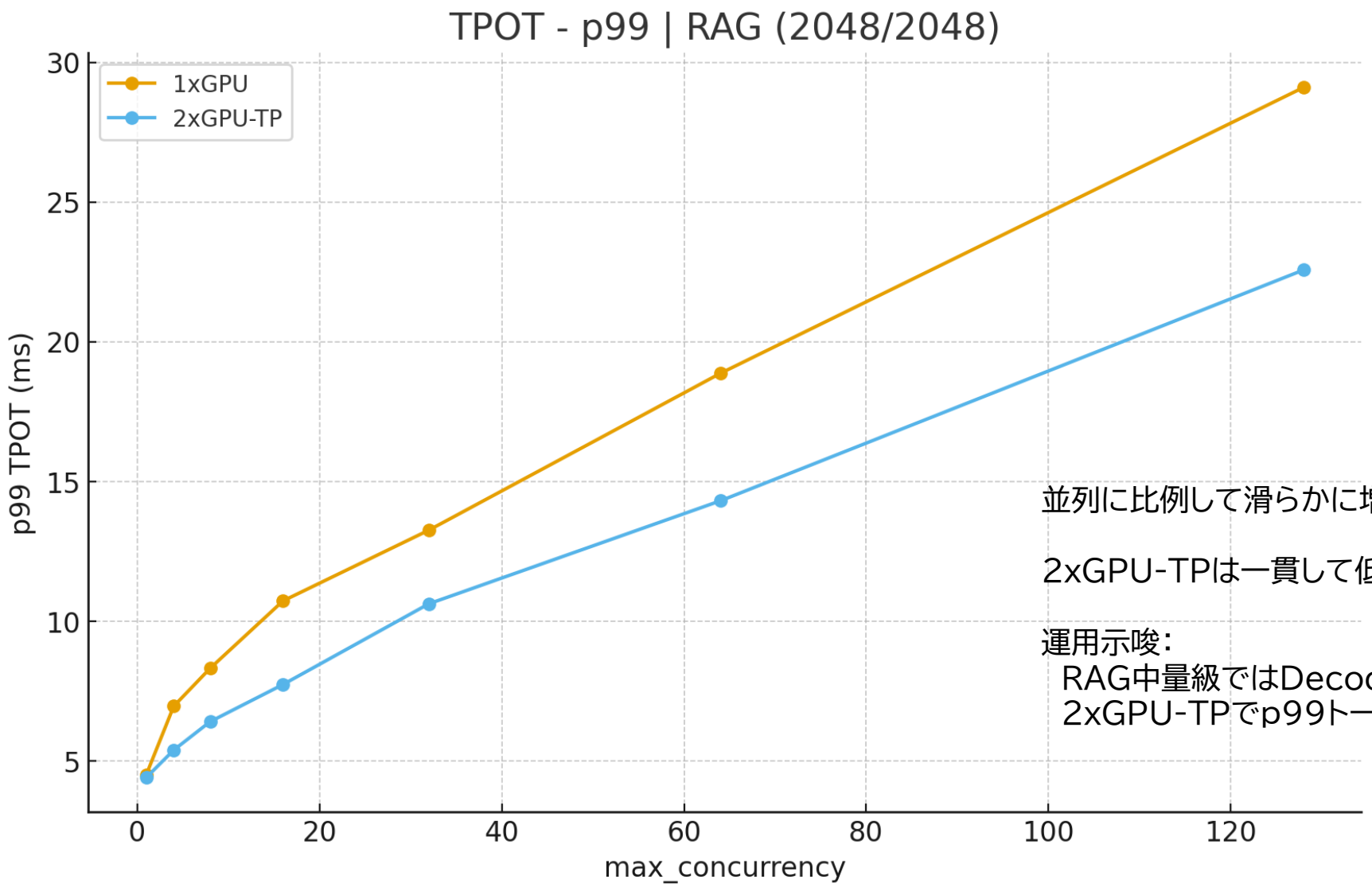


全体に低水準で安定。並列を上げても増加は緩やか。

2xGPU-TPが常に低い（同一並列で数ms差）。

運用示唆：
チャット用途ではTPOT起因の体感劣化は小。
高並列でもストリーミング体感を維持。

TPOT-P99（RAG向け） - input=2048, output=2048

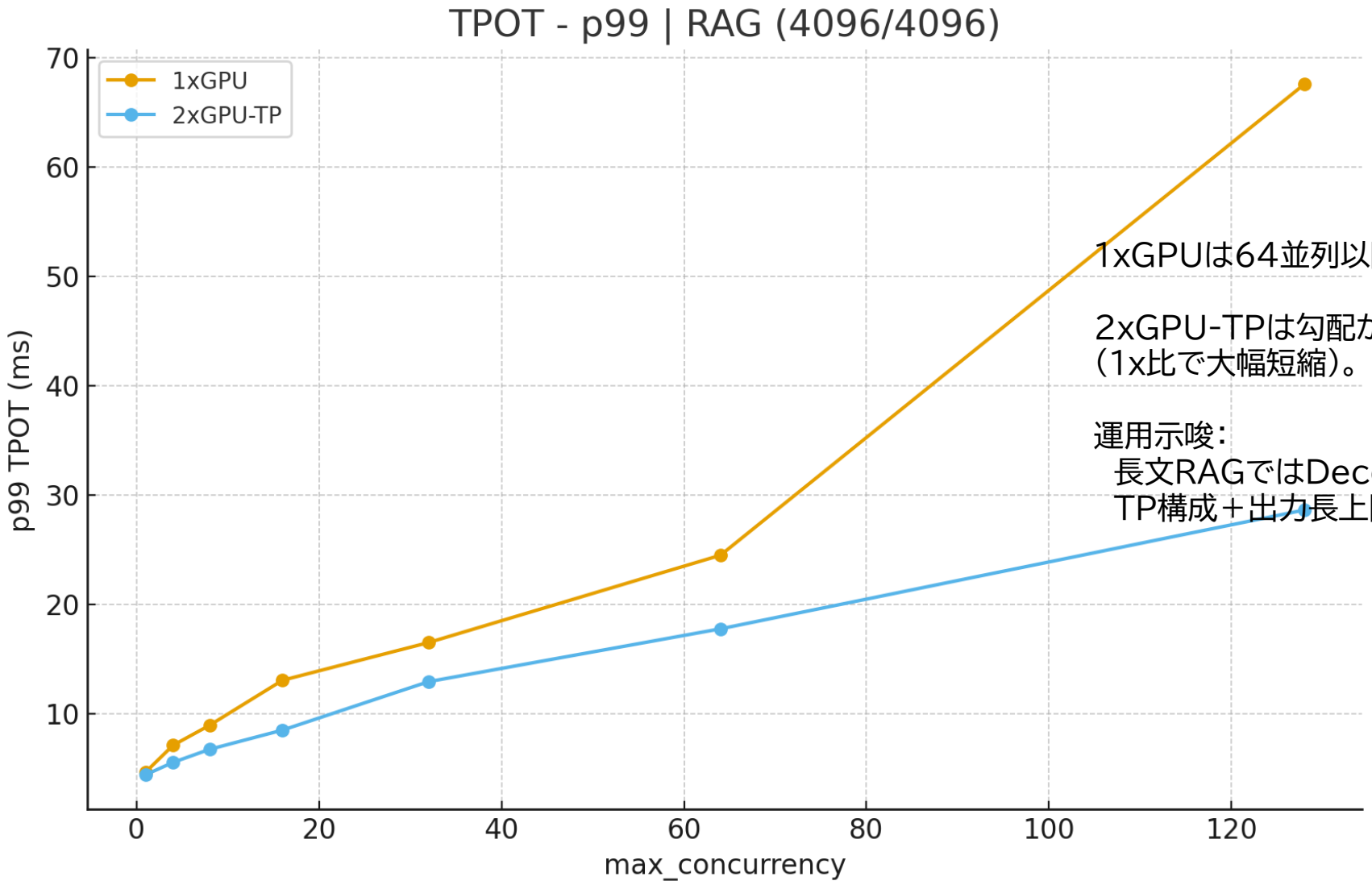


並列に比例して滑らかに増加。

2xGPU-TPは一貫して低位（同一並列で20～30%程度短縮）。

運用示唆：
RAG中量級ではDecode側も効いている。
2xGPU-TPでp99トークン間隔の悪化を抑制可能。

TPOT-P99（RAG/長コンテキスト向け）– input=4096, output=4096

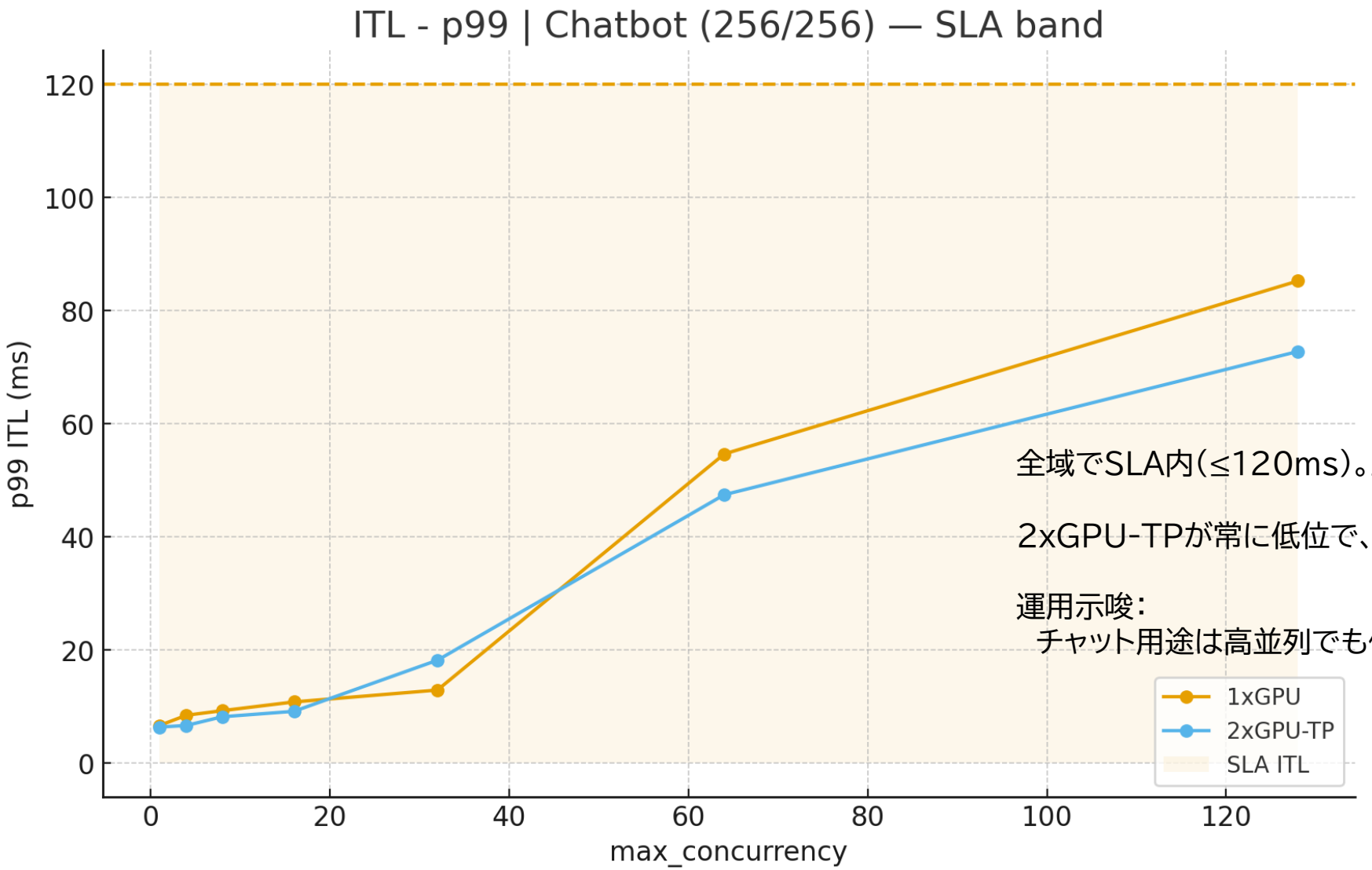


1xGPUは64並列以降で急伸（p99 TPOTが鋭く悪化）。

2xGPU-TPは勾配が緩く、高並列でも上振れが小さい
（1x比で大幅短縮）。

運用示唆：
長文RAGではDecode支配が顕著。
TP構成＋出力長上限でトークン間隔のばらつきを抑えると安定。

ITL-P99(チャットボット向け) - input=256, output=256

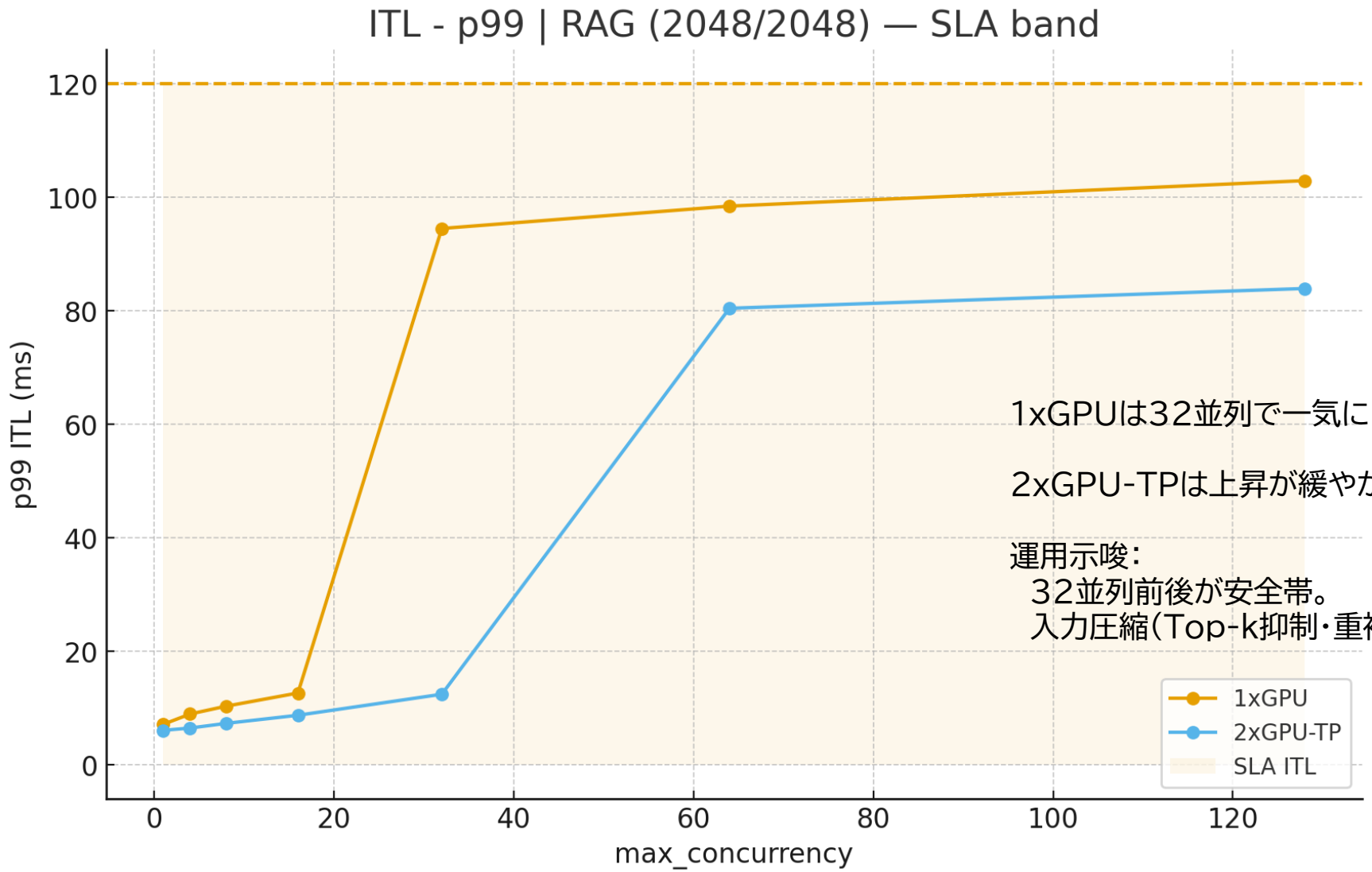


全域でSLA内(≤120ms)。並列↑でも緩やかな増加に留まる。

2xGPU-TPが常に低位で、ストリーミング体感がより滑らか。

運用示唆:
チャット用途は高並列でも体感の劣化が小。TPは余裕づくりに有効。

ITL-P99(RAG向け) - input=2048, output=2048

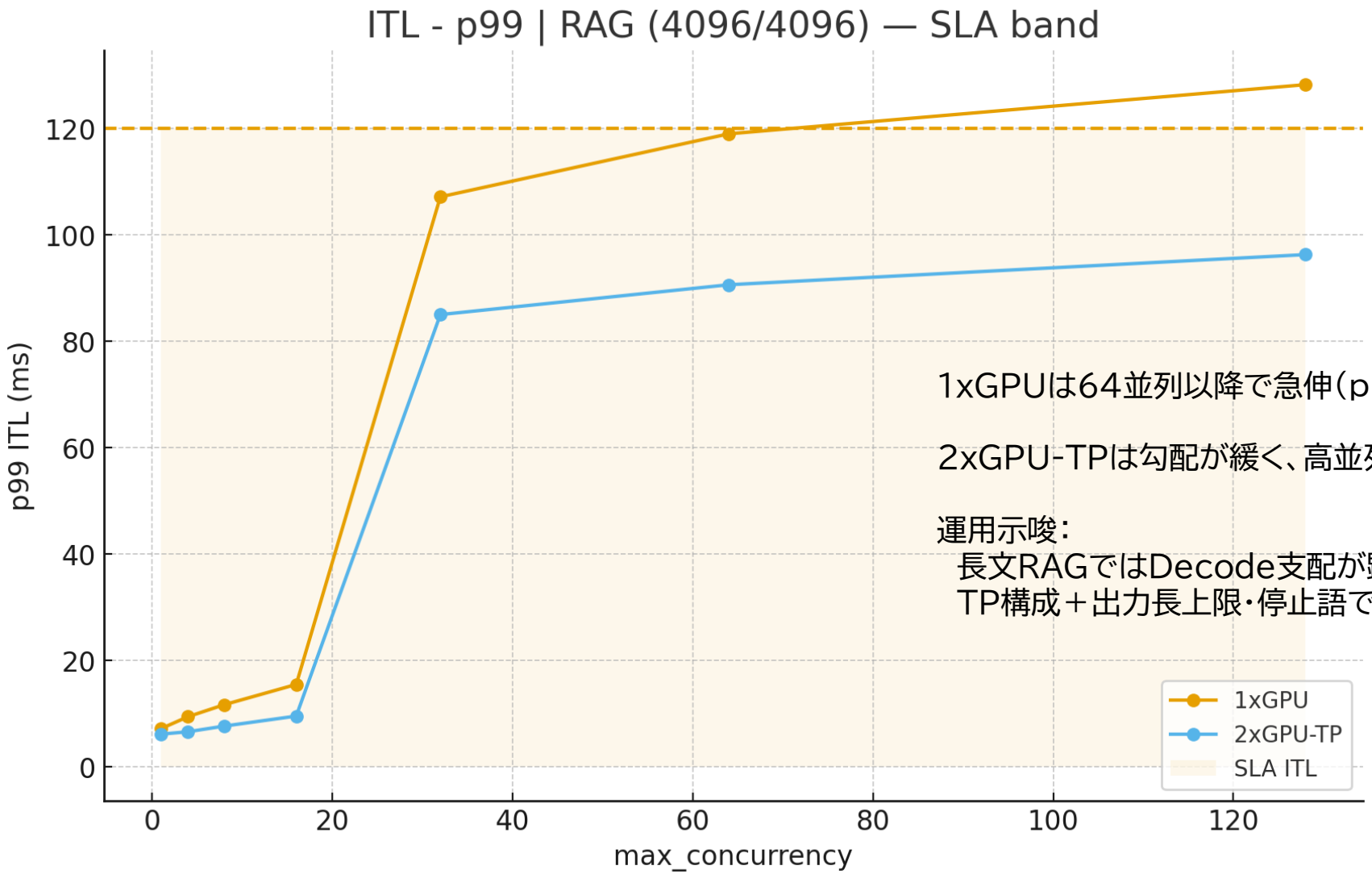


1xGPUは32並列で一気に80～100ms台へ上昇し、その後高止まり。

2xGPU-TPは上昇が緩やかで、同一並列で常に短いITL。

運用示唆:
32並列前後が安全帯。
入力圧縮(Top-k抑制・重複除去・要約)＋TPで体感のカクつき抑制。

ITL-P99（RAG/長コンテキスト向け）– input=4096, output=4096



1xGPUは64並列以降で急伸（p99 TPOTが鋭く悪化）。

2xGPU-TPは勾配が緩く、高並列でも上振れが小さい（1x比で大幅短縮）。

運用示唆：
長文RAGではDecode支配が顕著。
TP構成＋出力長上限・停止語でトークン間隔のばらつきを抑えると安定。

vLLMベンチマークのまとめ　コンテキスト長別の安全帯（SLA内 最大並列 & QPS）

📘 256/256（Chatbot相当）

最大並列 128
1xGPU: QPS = 27.07　/ 2xGPU-TP: QPS = 32.83（+21%）

📘 2048/2048（RAG・中量）

最大並列 32
1xGPU: QPS = 1.18 /　2xGPU-TP: QPS = 1.55（+31%）

📘 4096/4096（RAG・長コンテキスト）

最大並列 16
1xGPU: QPS = 0.30 / 2xGPU-TP: QPS = 0.48（+61%）

vLLMベンチマークのまとめ　設計指針

📘 チャットボット

並列は大きめに張れる（～128）。2xGPU-TPで+20～30% QPS向上、体感（TTFT/ITL）も安定。

運用はレート制限をやや緩める・バースト吸収（キュー/スロット）を厚めに。

たまに長文になる場合は出力上限/停止語で暴れを抑制。

📘 RAG

安全帯の初期値：2k/2k→32並列、4k/4k→16並列（2xTPでも上限は同等、同一並列でQPSが大幅増）。

入力圧縮が最優先：Top-k抑制、重複除去、要約でPrefill負荷とTTFT_p99を下げる。

Decode側の安定化：出力上限/停止語でTPOT・ITLの上振れを抑える。

インフラ：重い経路は2xGPU-TPノードへ集約。規模拡大はモデル複製 or テンソル/パイプ並列の適用でスケール。

RTX 5090を提案するとしたら…

根拠を示す

部門単位での運用を想定
小、中規模までのモデルを扱うことを想定

チャットボットやRAGの PoCや運用環境としてRTX5090を1枚、または2枚でも
十分性能は足りるのではないかということをベンチマーク結果を根拠として説明できますか
（前のセッションのB構成を根拠を持って説明できますか）

RTX 5090を提案するとしたら…

結論

RTX 5090×1枚で、チャットボット用途は実用十分。SLA内で27 QPS（1,620 req/min）。

RAGでも中規模までは現実的：2k/2kで1.18 QPS（≈71 req/min）。

RTX 5090×2枚（Tensor Parallel）にすると、同一SLAのまま+20～60%のQPS増（長文RAGほど伸び大）。
→ 部門規模の同時利用を想定するなら、1～2枚でPoC～初期本番をカバーできます。

RTX 5090を提案するとしたら…

根拠（まとめのページに記載したものです）

📘256/256（チャットボット相当）

最大並列 128
1xGPU: QPS = 27.07 / 2xGPU-TP: QPS = 32.83（+21%）

📘2048/2048（RAG・中量）

最大並列 32
1xGPU: QPS = 1.18 / 2xGPU-TP: QPS = 1.55（+31%）

📘4096/4096（RAG・長コンテキスト）

最大並列 16
1xGPU: QPS = 0.30 / 2xGPU-TP: QPS = 0.48（+61%）

どのケースも p99 TTFT/ITLがSLA内。特に長文RAGで2枚TPの改善幅が大でした。

RTX 5090を提案するとしたら…

運用のイメージに変換した説明例（ざっくりな目安ですが）

📘チャットボット（27 QPS @GPU1枚）

例：ユーザーが平均1リクエスト/3秒でやり取りするなら、同時アクティブ約80～100人でも余裕を持って捌けます
（ピーク時バーストを見込んで7～8割運用）。

📘RAG 2k/2k（1.18 QPS @GPU1枚）

例：検索付きの重め問い合わせが1～2件/秒程度の窓口であれば、1～2枚構成で十分。

📘RAG 4k/4k（0.30 QPS @GPU1枚 / 0.48 QPS @GPU2枚）

超長文の継続照会でも数十件/分規模は維持可能。2枚TPで“待ち”の悪化を抑えつつ+60%増し。

RTX 5090を提案するとしたら…

設計指針の提案例

📘 チャットボット

まずはGPU1枚で開始、並列上限は128近辺を目安。

2枚TPにすると+20～30%のQPS増＆p99遅延が安定 → レート制限を緩めやすい。

たまに長文になる運用は出力上限・停止語でブレを抑制。

📘 RAG

安全帯の初期値：2k/2k→32並列、4k/4k→16並列（2枚にしても上限はほぼ同じ、同一並列のQPSが増）。

入力圧縮が最優先：Top-k抑制・重複除去・要約でTTFT（Prefill）を削る。

出力側の安定化：上限トークン・停止語でTPOT/ITLの上振れを抑える。

重い経路はTPノードに寄せる（長文RAGほどTPのリターンが大）。

RTX 5090を提案するとしたら…

まとめ

今回の実測では、RTX 5090の1～2枚で部門向けチャットボット/RAGのPoC～小中規模本番を
十分に成立させられます。特にRAGの体感（p99）を崩さずQPSを欲張るなら、2枚TPがコスパ良く効きます

最後に

LLM構成は、誰かの成功事例を真似るのではなく、
「自分たちの目的に対しての適した解」を見つける作業です。

そのために必要なのは、高価なGPUでも、複雑な理論でもなく、
「観察する視点」と**「測定する姿勢」**です。

ハードウェア選定は、データに基づき判断することを推奨いたします。

この資料が、あなた自身の“選定軸”を獲得する第一歩になれば幸いです。

付録 vLLMベンチマークスコア 実験メタ情報

項目	説明	単位/備考
date	ベンチマーク実行時刻(実行IDとしても利用可)	例:2025-09-24 10:15:00
backend	推論エンジン名	例:vLLM
model_id	使用モデルの識別子	例:meta-llama/…
tokenizer_id	トークナイザ識別子(モデルと同一のことも)	—
filename	実験の設定・ログ等に対応するファイル名	—

付録 vLLMベンチマークスコア 負荷条件

項目	説明	単位/備考
num_prompts	発行したリクエスト(プロンプト)総数	件
request_rate	リクエスト送出レート	req/s(inf＝可能な限り最大)
burstiness	到着分布の“ばらつき度”(バースト性)	値↑ほどスパイクしやすい／1はほぼ均一
max_concurrency	同時並行実行の上限	
duration	ベンチマーク実行時間	s

付録 vLLMベンチマークスコア 完了状況・スループット

項目	説明	単位/備考
completed	正常終了したリクエスト数	件
request_throughput	完了リクエストの処理レート	req/s
request_goodput	エラー・タイムアウト等を除外した“有効”完了レート	req/s（未集計環境あり。エラー無ければ request_throughput と近い）
total_input_tokens	全リクエストの**入力**トークン合計	tokens
total_output_tokens	全リクエストの**出力**トークン合計	tokens
output_throughput	**出力トークン**のスループット（デコード実効能力）	tokens/s
total_token_throughput	**入出力合計トークン**のスループット（総合処理能力）	tokens/s（＝ 入力前計算＋生成の合算）

付録 vLLMベンチマークスコア レイテンシ(統計・パーセンタイル付き)

項目	説明	単位/備考
ttft_ms	**Time To First Token**: 最初の出力トークンが返るまで(初動体感)	ms
tpot_ms	**Time Per Output Token**: 出力1トークンあたりの所要時間(デコード速度)	ms/token(≈ 1000/TPS)
itl_ms	**Inter-Token Latency**: 連続トークン間隔(ストリーミング体感)	ms(ネットワーク/スケジューラ影響を受けやすい)
e2el_ms	**End-to-End Latency**: リクエスト送出から最終トークンまでの総時間	ms

いずれも ms 単位で、mean(平均)/median(中央値)/std(標準偏差)/p99(99パーセンタイル)が付随します。

付録 vLLMベンチマークスコア 入出力長(平均値)

項目	説明	単位/備考
input_len	1リクエストあたりの**入力**トークン長(平均)	tokens
output_len	1リクエストあたりの**出力**トークン長(平均)	tokens