PROJECT: Diagnosing the latency–throughput tradeoff in vLLM serving under concurrent loads

- **Experiment 1:** Baseline setup and workload harness for mixed prompt-length serving
- **Experiment 1.1:** Fix max-tokens and context-budget failures by adding dynamic `max_tokens` to prevent request rejections
- **Experiment 2:** BAD (unfair) scheduling characterization with concurrency sweeps to expose head-of-line blocking and tail-latency spikes
- **Experiment 3:** Tail latency and throughput variation with concurrency sweep by holding workload and output length constant to isolate scheduling effects
- **Experiment 4:** Sweep of concurrency and `max-num-seqs` under a deliberately bad configuration to map scaling limits, queueing onset, and short request degradation from long-prefill contention
- **Experiment 5:** Find a strong FAIR scheduling configuration by first sweeping `max-num-batched-tokens` at lower concurrency (c=16), then validating at c=32, and finally running a `max-num-batched-tokens × max-num-seqs` grid at c=32 to pick a stable throughput vs tail-latency sweet spot

Installing libraries

```
!pip -q install datasets transformers
!pip -q install tqdm

!pip -q install "jedi>=0.16"
!pip -q install vllm
```

Importing Libraries

```
from google.colab import drive
import os

import json, random, statistics
from pathlib import Path
from datasets import load_dataset
from transformers import AutoTokenizer

import itertools

import numpy as np
import pandas as pd
from tqdm.notebook import tqdm

import random

import vllm
import torch
from vllm import LLM, SamplingParams
import requests, time

import concurrent.futures as cf
import pandas as pd

import shutil
import math
import subprocess


```

Mounting the Drive

```
# Mounting Google Drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
# Write locally first for faster write, then copy to Drive at the end
OUT_DIR = "/content/workloads"
```

```
os.makedirs(OUT_DIR, exist_ok=True)

# Where to persist outputs in Drive
DRIVE_OUT_DIR = "/content/drive/MyDrive/llm_loadtest_workloads"
os.makedirs(DRIVE_OUT_DIR, exist_ok=True)

print("OUT_DIR:", OUT_DIR)
print("DRIVE_OUT_DIR:", DRIVE_OUT_DIR)

```

```
OUT_DIR: /content/workloads
DRIVE_OUT_DIR: /content/drive/MyDrive/llm_loadtest_workloads
```

```
MODEL = "Qwen/Qwen2.5-3B-Instruct"
tok = AutoTokenizer.from_pretrained(MODEL, use_fast=True)

def tok_len(text: str) -> int:
    return len(tok.encode(text, add_special_tokens=False))

def in_range(n, r):
    return r[0] <= n <= r[1]

random.seed(42)

SHORT_RANGE = (8, 24)        # NQ-friendly short prompts
MED_RANGE   = (128, 256)     # UltraChat-friendly medium prompts
LONG_RANGE  = (2048, 4096)   # long prompts via concatenation


# How many prompts you want in each bucket
WANT_SHORT  = 800
WANT_MEDIUM = 400
WANT_LONG   = 600

# Requests per workload file
TOTAL_REQS_PER_FILE = 400

print("Tokenizer loaded for:", MODEL)

```

```
tokenizer_config.json:   7.30k/? [00:00<00:00, 100kB/s]

vocab.json:   2.78M/? [00:00<00:00, 22.0MB/s]

merges.txt:   1.67M/? [00:00<00:00, 12.0MB/s]

tokenizer.json:   7.03M/? [00:00<00:00, 24.7MB/s]
Tokenizer loaded for: Qwen/Qwen2.5-3B-Instruct
```

Function to enable Dataset Streaming

```
def try_load_streaming(dataset_name, config_split_candidates):
    last_err = None

    for cfg, split in config_split_candidates:
        try:
            if cfg is None:
                ds = load_dataset(dataset_name, split=split, streaming=True)

            else:
                ds = load_dataset(dataset_name, cfg, split=split, streaming=True)
            print(f"Loaded {dataset_name} (config={cfg}, split={split})")

            return ds

        except Exception as e:
            last_err = e
            print(f"Failed {dataset_name} (config={cfg}, split={split}): {type(e).__name__}")

    raise RuntimeError(f"Could not load {dataset_name}. Last error: {last_err}")


```

```
# streaming both the datasets
ultra_stream = try_load_streaming(
    "HuggingFaceH4/ultrachat_200k",
    config_split_candidates=[
        (None, "train_sft"),
        ("default", "train_sft"),
        ("default", "train"),
    ],
)

nq_stream = try_load_streaming(
    "natural_questions",
    config_split_candidates=[
        (None, "train"),
        ("default", "train"),
    ],
)

```

```
README.md:   3.90k/? [00:00<00:00, 123kB/s]
Loaded HuggingFaceH4/ultrachat_200k (config=None, split=train_sft)
README.md:   13.7k/? [00:00<00:00, 239kB/s]

Resolving data files: 100%                      287/287 [00:00<00:00, 33.69it/s]

Resolving data files: 100%                      287/287 [00:00<00:00, 3896.64it/s]
Loaded natural_questions (config=None, split=train)
```

Check if the Dataset loaded properly

```
ex = next(iter(nq_stream))
print("Keys:", ex.keys())
print("Sample:", {k: ex[k] for k in list(ex.keys())[:5]})
```

```
Keys: dict_keys(['id', 'document', 'question', 'long_answer_candidates', 'annotations'])
Sample: {'id': '4549465242785278785', 'document': {'html': '<!DOCTYPE html>\n<HTML class="client-js ve-not-availa
```

```
ex = next(iter(ultra_stream))
print("Keys:", ex.keys())
print("Sample:", {k: ex[k] for k in list(ex.keys())[:5]})
```

```
Keys: dict_keys(['prompt', 'prompt_id', 'messages'])
Sample: {'prompt': "These instructions apply to section-based themes (Responsive 6.0+, Retina 4.0+, Parallax 3.0+
```

Functions to extract prompt from Dataset

```
def ultrachat_user_text(ex):
    prompt = ex.get("prompt")
    if isinstance(prompt, str) and prompt.strip():
        return prompt.strip()

    # This is a fallback: scan the "messages" list for the first user turn
    msgs = ex.get("messages")
    if isinstance(msgs, list):
        for m in msgs:
            if not isinstance(m, dict):
                continue

            role = (m.get("role") or "").lower()
            content = m.get("content") or ""

            if role == "user" and isinstance(content, str) and content.strip():
                return content.strip()

    return None


def nq_question_text(ex):
    q = ex.get("question")
    if isinstance(q, dict):
        t = q.get("text")
        return t.strip() if isinstance(t, str) and t.strip() else None
    if isinstance(q, str):
        return q.strip() if q.strip() else None
    return None

```

Create Small and Medum Buckets

```python
from tqdm import tqdm

def stream_bucket(
    stream,
    text_fn,
    want_n,
    token_range,
    max_read,
    desc,
    print_every=500,
    also_keep_pool=True,
):
    """
    Stream from a HF streaming dataset, extract text, collect:
      - bucket: texts in token_range until want_n
      - pool: all extracted texts (optional)
    """
    bucket = []
    pool = [] if also_keep_pool else None

    for i, ex in tqdm(enumerate(stream), total=max_read, desc=desc):
        if i >= max_read:
            break

        s = text_fn(ex)
        if not s:
            continue

        if also_keep_pool:
            pool.append(s)

        n = tok_len(s)
        if len(bucket) < want_n and in_range(n, token_range):
            bucket.append(s)

        if (i % print_every) == 0:
            tqdm.write(f"read={i:,}, {desc}_bucket={len(bucket)}")

        if len(bucket) >= want_n:
            break

    return bucket, pool

```

```python
MAX_NQ_READ = 50000

short_bucket, nq_pool = stream_bucket(
    stream=nq_stream,
    text_fn=nq_question_text,
    want_n=WANT_SHORT,
    token_range=SHORT_RANGE,
    max_read=MAX_NQ_READ,
    desc="short",
    print_every=200,   # keep your original cadence
    also_keep_pool=True,
)

print("short_bucket:", len(short_bucket))

```

```
short:   0%|          | 2/50000 [00:18<104:09:36,  7.50s/it]read=0, short_bucket=1
short:   0%|          | 208/50000 [00:30<26:42, 31.07it/s]read=200, short_bucket=201
short:   1%|          | 402/50000 [00:42<2:21:13,  5.85it/s]read=400, short_bucket=401
short:   1%|          | 604/50000 [00:58<35:59, 22.88it/s]read=600, short_bucket=601
short:   2%||         | 799/50000 [01:08<1:10:09, 11.69it/s]
short_bucket: 800
```

```python
MAX_ULTRA_READ = 200000

medium_bucket, ultra_pool = stream_bucket(
    stream=ultra_stream,
    text_fn=ultrachat_user_text,
    want_n=WANT_MEDIUM,
    token_range=MED_RANGE,
    max_read=MAX_ULTRA_READ,
    desc="medium",
    print_every=500,   # keep your original cadence
    also_keep_pool=True,
)

print("medium_bucket:", len(medium_bucket))
```

```
medium:   0%|          | 14/200000 [00:03<9:37:24,  5.77it/s] read=0, medium_bucket=0
medium:   0%|          | 528/200000 [00:04<10:42, 310.50it/s]read=500, medium_bucket=51
medium:   1%|          | 1040/200000 [00:06<09:46, 339.40it/s]read=1,000, medium_bucket=106
medium:   1%|          | 1537/200000 [00:08<12:04, 274.11it/s]read=1,500, medium_bucket=158
medium:   1%|          | 2042/200000 [00:09<09:52, 334.10it/s]read=2,000, medium_bucket=198
medium:   1%||         | 2543/200000 [00:11<10:07, 325.01it/s]read=2,500, medium_bucket=252
medium:   2%||         | 3004/200000 [00:13<23:16, 141.10it/s]read=3,000, medium_bucket=305
medium:   2%||         | 3614/200000 [00:19<07:18, 448.23it/s]read=3,500, medium_bucket=355
medium:   2%||         | 3990/200000 [00:20<17:07, 190.85it/s]medium_bucket: 400
```

Create Large Bucket

```python
def build_long_prompt(target_tokens, rng, snippets_pool, max_unique_before_reset=2000):
    parts = []
    used = set()

    while True:
        idx = rng.randrange(len(snippets_pool))
        if idx in used:
            continue
        used.add(idx)

        parts.append(snippets_pool[idx].replace("\n", " ").strip())
        text = "\n\n".join(parts)

        if tok_len(text) >= target_tokens:
            break

        if len(used) > min(max_unique_before_reset, len(snippets_pool) - 1):
            used.clear()

    ids = tok.encode(text, add_special_tokens=False)[:target_tokens]
    return tok.decode(ids)

def build_long_bucket(
    want_n,
    token_range,
    snippets_pool,
    seed=123,
    desc="long",
    print_every=10,
):
    assert len(snippets_pool) > 0, "snippets_pool must not be empty"

    rng = random.Random(seed)
    bucket = []

    for j in tqdm(range(want_n), desc=f"Building {desc}"):
        target = rng.randint(token_range[0], token_range[1])
        lp = build_long_prompt(target, rng, snippets_pool)
        bucket.append(lp)

        if ((j + 1) % print_every) == 0:
            tqdm.write(f"built={j+1}/{want_n}, last_len={tok_len(lp)} tokens")

    return bucket

```

```python
snippets_pool = short_bucket + medium_bucket
long_bucket = build_long_bucket(
    want_n=WANT_LONG,
    token_range=LONG_RANGE,
    snippets_pool=snippets_pool,
    seed=123,
    desc="long",
    print_every=10,
)

print("long_bucket:", len(long_bucket))

```

Show hidden output

Create Mixed Buckets

```python

from tqdm.auto import tqdm

TOTAL_REQS_PER_FILE = 400
SEED = 999

OVERWRITE_LOCAL = False
OVERWRITE_DRIVE = False

ENABLE_LONG_HEAVY = True
HEAVY_LONG_RANGE = (3500, 4096)
WANT_LONG_HEAVY = 600
USE_LONG_HEAVY_FOR = {"mix_60_40.jsonl", "mix_50_50.jsonl"}

MIX_SPECS = {
    "short_only.jsonl":     {"short": 1.0},
    "medium_only.jsonl":    {"medium": 1.0},
    "long_only.jsonl":      {"long": 1.0},

    "mix_90_10.jsonl":      {"short": 0.9, "long": 0.1},
    "mix_70_30.jsonl":      {"short": 0.7, "long": 0.3},
    "mix_70_20_10.jsonl":   {"short": 0.7, "medium": 0.2, "long": 0.1},

    # these use long heavy
    "mix_60_40.jsonl":      {"short": 0.6, "long": 0.4},
    "mix_50_50.jsonl":      {"short": 0.5, "long": 0.5},
}

assert "tok" in globals(), "Tokenizer 'tok' not found."
assert "short_bucket" in globals() and len(short_bucket) > 0, "short_bucket missing/empty."
assert "medium_bucket" in globals() and len(medium_bucket) > 0, "medium_bucket missing/empty."
assert "long_bucket" in globals() and len(long_bucket) > 0, "long_bucket missing/empty."


def tok_len(text: str) -> int:
    return len(tok.encode(text, add_special_tokens=False))


def build_long_prompt(target_tokens, rng, snippets_pool):
    parts = []
    used = set()
    while True:
        idx = rng.randrange(len(snippets_pool))
        if idx in used:
            continue
        used.add(idx)

        parts.append(snippets_pool[idx].replace("\n", " ").strip())
        text = "\n\n".join(parts)

        if tok_len(text) >= target_tokens:
            break

        if len(used) > min(2000, len(snippets_pool) - 1):
            used.clear()

    ids = tok.encode(text, add_special_tokens=False)[:target_tokens]
    return tok.decode(ids)


def build_long_bucket(want_n, token_range, snippets_pool, seed=1234, desc="long_bucket"):
    rng = random.Random(seed)
    bucket = []

    for _ in tqdm(range(want_n), desc=f"Building {desc} {token_range}"):
        target = rng.randint(token_range[0], token_range[1])
        bucket.append(build_long_prompt(target, rng, snippets_pool))

    return bucket


long_heavy_bucket = None
if ENABLE_LONG_HEAVY and len(USE_LONG_HEAVY_FOR) > 0:
    snippets_pool = short_bucket + medium_bucket
    long_heavy_bucket = build_long_bucket(
        want_n=WANT_LONG_HEAVY,
        token_range=HEAVY_LONG_RANGE,
        snippets_pool=snippets_pool,
        seed=1234,
        desc="long_heavy_bucket",
    )

    print(
        "long_heavy_bucket:",
        len(long_heavy_bucket),
        "example_len_tokens:",
        tok_len(long_heavy_bucket[0]),
    )

def write_jsonl(path, prompts, max_tokens=128, temperature=0.0):
    with open(path, "w", encoding="utf-8") as f:
        for p in prompts:
            row = {
                "messages": [{"role": "user", "content": p}],
                "max_tokens": max_tokens,
                "temperature": temperature,
            }
            f.write(json.dumps(row) + "\n")


def sample_prompts(bucket, k, rng):
    if k <= 0:
        return []
    if len(bucket) < k:
        raise ValueError(f"Bucket too small: need {k}, have {len(bucket)}")
    return rng.sample(bucket, k)


def normalize_ratios(ratios: dict) -> dict:
    r = dict(ratios)
    s = float(r.get("short", 0.0))
    m = float(r.get("medium", 0.0))
    l = float(r.get("long", 0.0))
    total = s + m + l

    if total > 1.000001:
        raise ValueError(f"Ratios sum > 1.0: {ratios}")

    if "long" not in r:
        r["long"] = 1.0 - (s + m)

    return r

def make_mix(total, ratios, rng, short_b, medium_b, long_b):
    ratios = normalize_ratios(ratios)
    k_short  = int(total * ratios.get("short", 0))
    k_medium = int(total * ratios.get("medium", 0))
    k_long   = total - k_short - k_medium

    prompts = []
    prompts += sample_prompts(short_b,  k_short,  rng)
    prompts += sample_prompts(medium_b, k_medium, rng)
    prompts += sample_prompts(long_b,   k_long,   rng)

    rng.shuffle(prompts)

    return prompts

def should_write(path, overwrite=False):
    return overwrite or (not os.path.exists(path))


def safe_copy(local_path, drive_path, overwrite=False):
    if not overwrite and os.path.exists(drive_path):
        return False

    shutil.copy(local_path, drive_path)

    return True

rng = random.Random(SEED)

written = []
skipped = []

for fn, ratios in MIX_SPECS.items():
    local_path = os.path.join(OUT_DIR, fn)
    drive_path = os.path.join(DRIVE_OUT_DIR, fn)

    if (not OVERWRITE_LOCAL) and os.path.exists(local_path) and (not OVERWRITE_DRIVE) and os.path.exists(dr
        skipped.append(fn)
        continue

    use_heavy = (fn in USE_LONG_HEAVY_FOR) and (long_heavy_bucket is not None)
    long_source = long_heavy_bucket if use_heavy else long_bucket

    prompts = make_mix(
        total=TOTAL_REQS_PER_FILE,
        ratios=ratios,
        rng=rng,
        short_b=short_bucket,
        medium_b=medium_bucket,
        long_b=long_source,
    )

    if should_write(local_path, overwrite=OVERWRITE_LOCAL):
        write_jsonl(local_path, prompts)
    else:
        skipped.append(fn)
        continue

    copied = safe_copy(local_path, drive_path, overwrite=OVERWRITE_DRIVE)
    if not copied:
        print("Exists on Drive, skipping copy:", drive_path)

    written.append(fn)

print("Done.")
print("Written:", written)
print("Skipped:", skipped)
print("Local dir:", OUT_DIR)
print("Drive dir:", DRIVE_OUT_DIR)

```

```
Building long_heavy_bucket (3500, 4096): 100%                                    600/600 [07:49<00:00,  1.57it/s]
long_heavy_bucket: 600 example_len_tokens: 3951
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/short_only.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/medium_only.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/long_only.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/mix_90_10.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/mix_70_30.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/mix_70_20_10.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/mix_60_40.jsonl
Exists on Drive, skipping copy: /content/drive/MyDrive/llm_loadtest_workloads/mix_50_50.jsonl
Done.
Written: ['short_only.jsonl', 'medium_only.jsonl', 'long_only.jsonl', 'mix_90_10.jsonl', 'mix_70_30.jsonl', 'mix_
Skipped: []
Local dir: /content/workloads
Drive dir: /content/drive/MyDrive/llm_loadtest_workloads
```

```
WORKLOAD_DIR = "/content/workloads"
FILES = sorted([f for f in os.listdir(WORKLOAD_DIR) if f.endswith(".jsonl")])


def tok_len(text: str) -> int:
    return len(tok.encode(text, add_special_tokens=False))

def read_jsonl(path: str):
    rows = []

    with open(path, "r", encoding="utf-8") as f:
        for line in f:
            if not line.strip():
                continue

            rows.append(json.loads(line))

    return rows

def extract_prompt(row: dict) -> str:
    msgs = row.get("messages", [])

    if isinstance(msgs, list) and len(msgs) > 0 and isinstance(msgs[0], dict):
        return (msgs[0].get("content") or "").strip()

    return ""

def stats(arr):
    arr = np.array(arr, dtype=float)

    if len(arr) == 0:
        return {}

    return {
        "n": int(len(arr)),
        "min": float(np.min(arr)),
```

```python
        "p50": float(np.percentile(arr, 50)),
        "p90": float(np.percentile(arr, 90)),
        "p95": float(np.percentile(arr, 95)),
        "p99": float(np.percentile(arr, 99)),
        "max": float(np.max(arr)),
        "mean": float(np.mean(arr)),
    }

results = []
per_file_details = {}

for fn in tqdm(FILES, desc="Verifying workload JSONLs"):
    path = os.path.join(WORKLOAD_DIR, fn)
    rows = read_jsonl(path)

    prompts = [extract_prompt(r) for r in rows]
    prompts = [p for p in prompts if p]  # drop empties

    tok_lens = [tok_len(p) for p in prompts]
    char_lens = [len(p) for p in prompts]

    s_tok = stats(tok_lens)
    s_chr = stats(char_lens)

    results.append({
        "file": fn,
        "n": s_tok.get("n", 0),
        "tok_min": s_tok.get("min"),
        "tok_p50": s_tok.get("p50"),
        "tok_p90": s_tok.get("p90"),
        "tok_p95": s_tok.get("p95"),
        "tok_p99": s_tok.get("p99"),
        "tok_max": s_tok.get("max"),
        "tok_mean": s_tok.get("mean"),
        "char_p50": s_chr.get("p50"),
        "char_p99": s_chr.get("p99"),
    })

    # store some examples for sanity checks
    if len(tok_lens) > 0:
        idx_sorted = np.argsort(tok_lens)
        smallest = idx_sorted[:2].tolist()
        largest = idx_sorted[-2:].tolist()
        per_file_details[fn] = {
            "smallest_examples": [(tok_lens[i], prompts[i][:200]) for i in smallest],
            "largest_examples": [(tok_lens[i], prompts[i][:200]) for i in largest],
        }

df_stats = pd.DataFrame(results).sort_values("file")
display(df_stats)

print("\n--- Sanity samples (first 200 chars) ---")
for fn, det in per_file_details.items():
    print(f"\n### {fn}")
    print("Smallest:")
    for L, s in det["smallest_examples"]:
        print(f"  tok={L} | {s!r}")
    print("Largest:")
    for L, s in det["largest_examples"]:
        print(f"  tok={L} | {s!r}")

```

```
Verifying workload JSONLs: 100%                                    8/8 [00:20<00:00,  2.03s/it]
```

	file	n	tok_min	tok_p50	tok_p90	tok_p95	tok_p99	tok_max	tok_mean	char_p50	char_p99
0	long_only.jsonl	400	2051.0	3068.5	3931.1	4028.10	4083.01	4092.0	3061.0350	14489.5	19827.18
1	medium_only.jsonl	400	128.0	169.0	231.1	240.05	253.00	256.0	175.9325	819.0	1378.01
2	mix_50_50.jsonl	400	8.0	1761.0	4018.1	4062.10	4090.02	4096.0	1918.2400	7905.0	19973.59
3	mix_60_40.jsonl	400	8.0	12.5	3983.1	4042.10	4089.01	4095.0	1532.5800	56.0	19942.21
4	mix_70_20_10.jsonl	400	8.0	11.0	432.9	3189.05	3898.31	4090.0	357.9650	49.5	18275.65
5	mix_70_30.jsonl	400	8.0	11.0	3525.5	3852.10	4070.13	4092.0	939.5800	50.0	19346.25
6	mix_90_10.jsonl	400	8.0	10.0	239.1	3332.80	3887.60	4090.0	334.2300	46.0	18965.18
7	short_only.jsonl	400	8.0	10.0	14.0	15.00	19.01	23.0	10.5925	45.0	88.01

```
--- Sanity samples (first 200 chars) ---

### long_only.jsonl
Smallest:
  tok=2051 | 'who wrote the song come thou fount of every blessing\n\nCan you summarize the setting and atmospher
  tok=2058 | 'Write a fast-paced, plot-driven thriller novel that chronicles the story of a young woman abducted
Largest:
  tok=4090 | "elvis presley daddy don't you walk so fast\n\nhow old do you have to be to get a tattoo in utah\n\n
  tok=4092 | 'when was the last time a total eclipse happened\n\nWrite a detailed report that analyzes and summar

### medium_only.jsonl
Smallest:
  tok=128 | 'Great way to get your message out there, "Happy Birthday", "Will You Marry Me" and More!\nLet them k
  tok=128 | 'Implement a queue data structure in Java that can enqueue and dequeue elements. The queue should hav
Largest:
  tok=255 | 'such as. The grüffelo) or poems for children. Do you have any tips? Recommendations? Or would you li
  tok=256 | '1. Roast and powder coriander seeds, cumin seeds and red chillies.\n2. Grind together ginger-garlic.

### mix_50_50.jsonl
Smallest:
  tok=8 | 'most commonly used punishment in the united states'
  tok=8 | 'i m in the love with the coco'
Largest:
  tok=4095 | 'who does the voice of shaggy on scooby doo\n\ntom and jerry movie the fast and the furry\n\nIs it p
  tok=4096 | 'deeper than the holler what is a holler\n\nwho played tj on head of the class\n\nis the song fight

### mix_60_40.jsonl
Smallest:
  tok=8 | 'does president have to be born in usa'
  tok=8 | 'who wants to marry a prince tv show'
Largest:
  tok=4095 | 'Can you provide a brief summary of the central problem facing Dell management according to the HBS
  tok=4095 | 'who does the voice of shaggy on scooby doo\n\ntom and jerry movie the fast and the furry\n\nIs it p

### mix_70_20_10.jsonl
Smallest:
  tok=8 | 'properties of red black tree in data structure'
  tok=8 | 'what are the parts of the finger called'
Largest:
  tok=4084 | 'Write a free verse poem that captures the beauty and essence of each season, drawing inspiration fr
  tok=4090 | "elvis presley daddy don't you walk so fast\n\nhow old do you have to be to get a tattoo in utah\n\n

### mix_70_30.jsonl
Smallest:
  tok=8 | 'what number book is mark in the bible'
  tok=8 | 'the legend of the monkey king tv series'
Largest:
  tok=4090 | "elvis presley daddy don't you walk so fast\n\nhow old do you have to be to get a tattoo in utah\n\n
  tok=4092 | 'when was the last time a total eclipse happened\n\nWrite a detailed report that analyzes and summar

### mix_90_10.jsonl
Smallest:
  tok=8 | 'why do ice cubes get smaller over time'
  tok=8 | 'early blues musicians used which combination of instruments'
Largest:
  tok=4032 | "what is the meaning of yanny and laurel\n\nearly blues musicians used which combination of instrume
  tok=4090 | "elvis presley daddy don't you walk so fast\n\nhow old do you have to be to get a tattoo in utah\n\n

### short_only.jsonl
Smallest:
  tok=8 | 'who sang take that look off your face'
  tok=8 | 'where is the us military base in japan'
Largest:
  tok=21 | 'nasa conducted a poll with 9 names to choose the name of the rover which landed on mars recently'
  tok=23 | 'erupted on august 3 1965 as the result of a routine arrest of a drunk driver'
```

Setting up vLLM server and Dataset loading

```python
print("vLLM:", vllm.__version__)
print("torch:", torch.__version__, "cuda:", torch.version.cuda)
print("GPU:", torch.cuda.get_device_name(0))
```

```
vLLM: 0.13.0
torch: 2.9.0+cu126 cuda: 12.6
GPU: NVIDIA L4
```

```python
# Use this cell, when restarting the Server again in the same runtime
# it clears out the previous instances prevents issues in server setup

!pkill -f "vllm.entrypoints.openai.api_server" || true
!pkill -f "EngineCore" || true
!pkill -f "vllm" || true

```

```
^C
^C
^C
```

```python
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"
PORT = 8000

MAX_MODEL_LEN = 4096
GPU_UTIL = 0.70
DTYPE = "half"              # fp16 weights/compute

# Launch server
# run it in the background and log to a file.
cmd = f"""
nohup python3 -m vllm.entrypoints.openai.api_server \
  --model {MODEL_ID} \
  --host 127.0.0.1 \
  --port {PORT} \
  --max-model-len {MAX_MODEL_LEN} \
  --gpu-memory-utilization {GPU_UTIL} \
  --dtype {DTYPE} \
  --enforce-eager \
  --disable-log-stats \
  > vllm_server.log 2>&1 &
"""

print("Launching vLLM server...")
!bash -lc "$cmd"

print("Server process launched. Showing last 30 log lines:")
!tail -n 30 vllm_server.log

```

```
Launching vLLM server...
Server process launched. Showing last 30 log lines:
```

Function to check the successful startup of the server

```python
import time, requests

BASE = "http://127.0.0.1:8000"

def wait_for_server(timeout_s=600):
    t0 = time.time()
    while time.time() - t0 < timeout_s:
        try:
            r = requests.get(f"{BASE}/v1/models", timeout=2)
            if r.status_code == 200:
                return True, r.json()
        except Exception:
            pass
        time.sleep(2)
    return False, None

ok, payload = wait_for_server()
print("Ready:", ok)
if ok:
    print("Models:", [m["id"] for m in payload.get("data", [])])
else:
    print("Not ready yet. Check logs:")
    !tail -n 60 vllm_server.log

```

```
Ready: True
Models: ['Qwen/Qwen2.5-3B-Instruct']
```

```
# Server take almost a minute to start,
# keep checking the logs to confirm successful startup
# before running any experiments

!tail -n 200 vllm_server.log

```

Show hidden output

Sanity check after Server start

```
import requests, json

BASE="http://127.0.0.1:8000"

payload = {
  "model": "Qwen/Qwen2.5-3B-Instruct",
  "messages": [{"role":"user","content":"Say hi in one sentence."}],
  "max_tokens": 32,
  "temperature": 0.0
}

r = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=120)
print(r.status_code)
print(r.json()["choices"][0]["message"]["content"])

```

```
200
Hi there! How can I assist you today?
```

```
# check the GPU consumption once the vLLM server is up
! nvidia-smi
```

```
Sun Jan 11 19:55:23 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA L4                      Off |   00000000:00:03.0 Off |                    0 |
| N/A   56C    P0             30W /   72W |   16225MiB /  23034MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
+-----------------------------------------------------------------------------------------+
```

Load the dataset and run tests

```
from google.colab import drive
drive.mount("/content/drive")

```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
```

Load the dataset from Drive

```
import json, os, random
from pathlib import Path

PROMPT_DIR = Path("/content/drive/MyDrive/llm_loadtest_workloads")

FILES = [
    "short_only.jsonl",
    "short_only.jsonl",
    "medium_only.jsonl",
    "long_only.jsonl",
    "mix_90_10.jsonl",
    "mix_70_30.jsonl",
    "mix_70_20_10.jsonl",
    "mix_60_40.jsonl",
    "mix_50_50.jsonl"
]

def load_jsonl_prompts(path: Path, limit=None, seed=123):
    prompts = []
    with path.open("r", encoding="utf-8") as f:
        for line in f:
            obj = json.loads(line)

            if "prompt" in obj:
                prompts.append(obj["prompt"])

            elif "messages" in obj:
                users = [m["content"] for m in obj["messages"] if m.get("role") == "user"]

                if users:
                    prompts.append(users[-1])

            if limit and len(prompts) >= limit:
                break

    random.Random(seed).shuffle(prompts)
    return prompts

prompt_sets = {}
for fn in FILES:
    p = PROMPT_DIR / fn
    if p.exists():
        prompt_sets[fn] = load_jsonl_prompts(p)
        print(f"Loaded {fn}: {len(prompt_sets[fn])} prompts")
    else:
        print(f"Missing: {p}")

```

```
Loaded short_only.jsonl: 400 prompts
Loaded medium_only.jsonl: 400 prompts
Loaded long_only.jsonl: 400 prompts
Loaded mix_90_10.jsonl: 400 prompts
Loaded mix_70_30.jsonl: 400 prompts
Loaded mix_70_20_10.jsonl: 400 prompts
Loaded mix_60_40.jsonl: 400 prompts
Loaded mix_50_50.jsonl: 400 prompts
```

Checking if the dataset is properly read

```
from transformers import AutoTokenizer
import numpy as np, random

TOK_MODEL = "Qwen/Qwen2.5-3B-Instruct"
tok = AutoTokenizer.from_pretrained(TOK_MODEL, use_fast=True)

def tok_len(s: str) -> int:
    return len(tok.encode(s, add_special_tokens=False))

def summarize_lengths(name, prompts, n=200):
    sample = random.sample(prompts, min(n, len(prompts)))
    lens = np.array([tok_len(p) for p in sample])
    print(f"{name:18s}  n={len(sample):4d}  min={lens.min():4d}  p50={int(np.median(lens)):4d}  p90={int(np.

for fn, prompts in prompt_sets.items():
    summarize_lengths(fn, prompts)

```

```
short_only.jsonl    n= 200  min=   8  p50=  10  p90=  13  p99=  16  max=  16
medium_only.jsonl   n= 200  min= 128  p50= 166  p90= 230  p99= 246  max= 255
long_only.jsonl     n= 200  min=2051  p50=3196  p90=3949  p99=4062  max=4093
mix_90_10.jsonl     n= 200  min=   8  p50=  10  p90=2319  p99=3944  max=4002
mix_70_30.jsonl     n= 200  min=   8  p50=  11  p90=3312  p99=3924  max=4031
mix_70_20_10.jsonl  n= 200  min=   8  p50=  11  p90= 231  p99=3660  max=3833
mix_60_40.jsonl     n= 200  min=   8  p50=  12  p90=3970  p99=4082  max=4090
mix_50_50.jsonl     n= 200  min=   8  p50=3510  p90=4002  p99=4084  max=4096
```

Experiment 1

In this experiment, we stress-test vLLM serving under mixed prompt-length workloads to reproduce two real-world issues:

(1) context-budget failures where long prompts combined with a fixed max_tokens cause requests to error out (HTTP 400 max token limit), and

(2) degraded serving performance as the fraction of long prompts increases. We run controlled workloads (short-only and mixed ratios like 90/10 and 70/30), then compare success rate, throughput (RPS), and latency metrics (p50/p90/p99) to quantify how long prompts amplify tail latency and reduce effective throughput.

This sets up the next experiments where we mitigate the failure mode (dynamic max_tokens / context budgeting) and run experiments to see how can we configure our servers to perform better under mixed loads.

Single-Request Client Helper (Latency + Token Usage)

```
import time, requests

BASE = "http://127.0.0.1:8000"
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"

def one_request(prompt, max_tokens=128, temperature=0.0, timeout=120):
    t0 = time.perf_counter()
    payload = {
        "model": MODEL_ID,
        "messages": [{"role": "user", "content": prompt}],
        "max_tokens": max_tokens,
        "temperature": temperature,
    }

    r = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=timeout)

    latency = time.perf_counter() - t0

    # Hard fail with useful error
    if r.status_code != 200:
        txt = r.text[:500]
        raise RuntimeError(f"HTTP {r.status_code}: {txt}")

    j = r.json()
    usage = j.get("usage", {})
    return {
        "latency_s": latency,
        "prompt_tokens": usage.get("prompt_tokens"),
        "completion_tokens": usage.get("completion_tokens"),
        "total_tokens": usage.get("total_tokens"),
        "status_code": r.status_code,
    }

```

Some checks before starting the experiment

```
# Is the server process actually running?
!ps aux | grep -E "vllm\.entrypoints\.openai\.api_server|api_server" | grep -v grep

# Is anything listening on port 8000, and which process owns it?
!ss -ltnp | grep ":8000" || true

# What did the server log say most recently?
!tail -n 80 vllm_server.log || true


print(requests.get("http://127.0.0.1:8000/v1/models").status_code)
print(requests.get("http://127.0.0.1:8000/v1/models").json())
```

Show hidden output

Server Health Check + Verify vLLM OpenAI Endpoint Is Working

```
import requests, json

BASE = "http://127.0.0.1:8000"
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"
```

```python

payload = {
    "model": MODEL_ID,
    "messages": [{"role": "user", "content": "Say hello in one sentence."}],
    "max_tokens": 32,
    "temperature": 0.0,
}

r = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=120)
print("status:", r.status_code)
print("text:", r.text[:400])

```

```
status: 200
text: {"id":"chatcmpl-bc2f4e8695ca898b","object":"chat.completion","created":1768163326,"model":"Qwen/Qwen2.5-3B-
```

Single-Request Sanity Check (Real Workload Prompt)

```python
test_prompt = prompt_sets["short_only.jsonl"][0]
print("prompt tokens (rough chars):", len(test_prompt))

out = one_request(test_prompt, max_tokens=32)
print(out)

```

```
prompt tokens (rough chars): 49
{'latency_s': 0.8831302409998898, 'prompt_tokens': 39, 'completion_tokens': 32, 'total_tokens': 71, 'status_code'
```

Client-Side Load Generator (ThreadPool) + Latency/RPS Summary

```python
def run_load(prompts, concurrency=8, total_requests=200, max_tokens=128, desc="load"):
    rng = random.Random(123)
    picked = [rng.choice(prompts) for _ in range(total_requests)]

    results = []
    t_start = time.perf_counter()

    def task(i):
        p = picked[i]
        try:
            res = one_request(p, max_tokens=max_tokens)
            res["ok"] = True
        except Exception as e:
            res = {"ok": False, "error": str(e), "latency_s": None,
                   "prompt_tokens": None, "completion_tokens": None, "total_tokens": None}
        return res

    ok_ct = 0
    fail_ct = 0

    with cf.ThreadPoolExecutor(max_workers=concurrency) as ex:
        futs = [ex.submit(task, i) for i in range(total_requests)]

        pbar = tqdm(total=total_requests, desc=desc)
        for f in cf.as_completed(futs):
            r = f.result()
            results.append(r)
            if r.get("ok"):
                ok_ct += 1
            else:
                fail_ct += 1
            pbar.update(1)
            pbar.set_postfix(ok=ok_ct, fail=fail_ct)
        pbar.close()

    t_end = time.perf_counter()

    df = pd.DataFrame(results)
    ok_df = df[df["ok"] == True].copy()

    summary = {
        "concurrency": concurrency,
        "total_requests": total_requests,
        "ok": int(ok_df.shape[0]),
        "fail": int(df.shape[0] - ok_df.shape[0]),
        "wall_time_s": t_end - t_start,
        "rps": ok_df.shape[0] / (t_end - t_start) if (t_end - t_start) > 0 else None,
        "p50_s": float(ok_df["latency_s"].quantile(0.50)) if len(ok_df) else None,
        "p90_s": float(ok_df["latency_s"].quantile(0.90)) if len(ok_df) else None,
        "p99_s": float(ok_df["latency_s"].quantile(0.99)) if len(ok_df) else None,
        "mean_s": float(ok_df["latency_s"].mean()) if len(ok_df) else None,
    }
    return df, summary

```

Reproducing Context-Length Failures Under Load (Max Token Limit)

```python
workloads = [
    ("short_only", "short_only.jsonl"),
    ("mix_90_10", "mix_90_10.jsonl"),
    ("mix_70_30", "mix_70_30.jsonl"),
]

all_summaries = []
all_dfs = {}

for name, fn in tqdm(workloads, desc="Workloads"):
    print("\nRunning:", name)
    df, summ = run_load(
        prompt_sets[fn],
        concurrency=8,
        total_requests=200,
        max_tokens=128,
        desc=f"{name} (c=8)"
    )
    all_dfs[name] = df
    all_summaries.append({"workload": name, **summ})
    print(summ)

summary_df = pd.DataFrame(all_summaries)
summary_df

```

```
Workloads: 100%                                        3/3 [04:25<00:00, 90.89s/it]

Running: short_only
short_only (c=8): 100%                                 200/200 [01:19<00:00,  2.34it/s, fail=0, ok=200]
{'concurrency': 8, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 79.18587238500186, 'rps': 2.525703

Running: mix_90_10
mix_90_10 (c=8): 100%                                  200/200 [01:26<00:00,  2.30it/s, fail=2, ok=198]
{'concurrency': 8, 'total_requests': 200, 'ok': 198, 'fail': 2, 'wall_time_s': 86.622785784999, 'rps': 2.28577271

Running: mix_70_30
mix_70_30 (c=8): 100%                                  200/200 [01:39<00:00,  2.96it/s, fail=5, ok=195]
{'concurrency': 8, 'total_requests': 200, 'ok': 195, 'fail': 5, 'wall_time_s': 99.59230680399924, 'rps': 1.957982
```

	workload	concurrency	total_requests	ok	fail	wall_time_s	rps	p50_s	p90_s	p99_s	mean_s
0	short_only	8	200	200	0	79.185872	2.525703	3.553791	3.576180	3.586765	3.104981
1	mix_90_10	8	200	198	2	86.622786	2.285773	3.615653	4.137271	4.528919	3.433756
2	mix_70_30	8	200	195	5	99.592307	1.957983	4.101179	4.894588	5.376055	4.054638

```python
for name, df in all_dfs.items():
    print("\n===", name, "===")
    print("rows:", len(df))
    print("cols:", list(df.columns))

    if "ok" in df.columns:
        fails = df[df["ok"] == False]
    elif "status" in df.columns:
        fails = df[df["status"] != 200]
    else:
        print("I don't see ok/status columns — tell me df.columns and I'll adapt.")
        continue

    print("fail count:", len(fails))
    if len(fails):
        cols_to_show = [c for c in ["status", "error", "latency_s", "prompt_len", "prompt_tokens"] if c in fa
        display(fails[cols_to_show].head(10))
        if "error" in fails.columns:
            print("\nTop error reasons:")
            display(fails["error"].value_counts().head(10))
```

```
=== short_only ===
rows: 200
cols: ['latency_s', 'prompt_tokens', 'completion_tokens', 'total_tokens', 'status_code', 'ok']
fail count: 0

=== mix_90_10 ===
rows: 200
cols: ['latency_s', 'prompt_tokens', 'completion_tokens', 'total_tokens', 'status_code', 'ok', 'error']
fail count: 2
```

	error	latency_s	prompt_tokens
159	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN
168	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN

```
Top error reasons:
```

error	count
HTTP 400: {"error":{"message":"'max_tokens' or 'max_completion_tokens' is too large: 128. This model's maximum context length is 4096 tokens and your request has 4030 input tokens (128 > 4096 - 4030). None","type":"BadRequestError","param":null,"code":400}}	1
HTTP 400: {"error":{"message":"'max_tokens' or 'max_completion_tokens' is too large: 128. This model's maximum context length is 4096 tokens and your request has 4040 input tokens (128 > 4096 - 4040). None","type":"BadRequestError","param":null,"code":400}}	1

dtype: int64

```
=== mix_70_30 ===
rows: 200
cols: ['latency_s', 'prompt_tokens', 'completion_tokens', 'total_tokens', 'status_code', 'ok', 'error']
fail count: 5
```

	error	latency_s	prompt_tokens
85	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN
90	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN
118	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN
178	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN
189	HTTP 400: {"error":{"message":"'max_tokens' or...	NaN	NaN

```
Top error reasons:
```

error	count
HTTP 400: {"error":{"message":"'max_tokens' or 'max_completion_tokens' is too large: 128. This model's maximum context length is 4096 tokens and your request has 4040 input tokens (128 > 4096 - 4040). None","type":"BadRequestError","param":null,"code":400}}	2
HTTP 400: {"error":{"message":"'max_tokens' or 'max_completion_tokens' is too large: 128. This model's maximum context length is 4096 tokens and your request has 3984 input tokens (128 > 4096 - 3984). None","type":"BadRequestError","param":null,"code":400}}	1
HTTP 400: {"error":{"message":"'max_tokens' or 'max_completion_tokens' is too large: 128. This model's maximum context length is 4096 tokens and your request has 4015 input tokens (128 > 4096 - 4015). None","type":"BadRequestError","param":null,"code":400}}	1

Conclusion

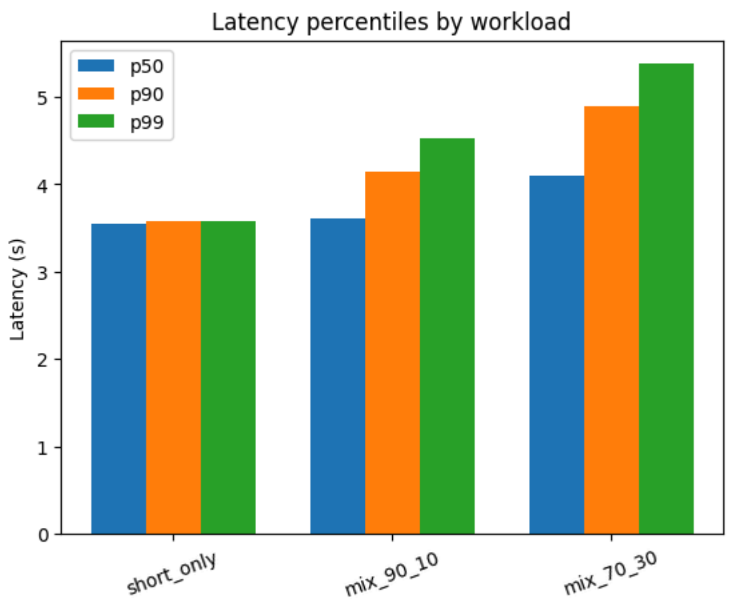
In the mixed workloads (mix_90_10 and mix_70_30), we start seeing HTTP 400 failures even though the same code works fine for short-only prompts. The error message shows why: the model has a fixed maximum context length of 4096 tokens, and some of our long prompts already consume ~3980–4060 input tokens.

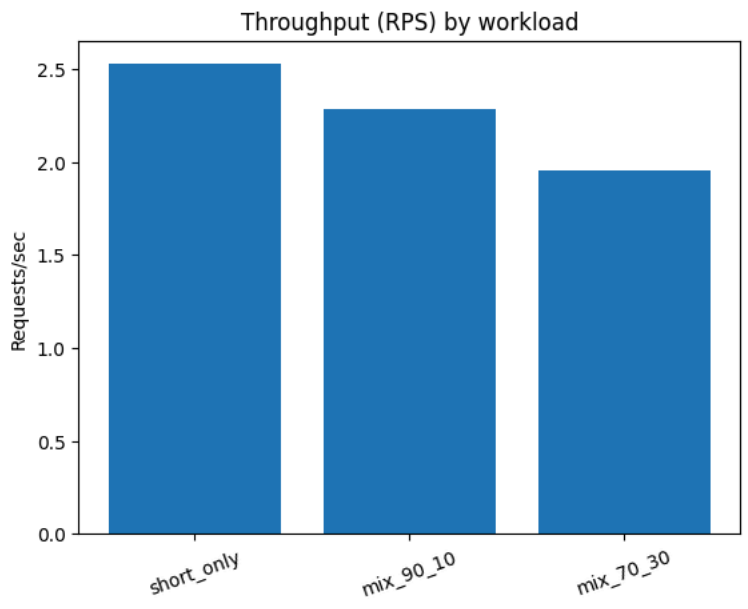
When we blindly request a fixed max_tokens=128 for completion, the request becomes impossible because the remaining context budget (4096 - input_tokens) is smaller than 128. vLLM therefore rejects those requests with a BadRequest error (e.g., "128 > 4096 – 4040").

This confirms a real serving issue under mixed prompt lengths: static max_tokens can cause hard failures whenever inputs approach the context window, motivating the next experiment where we dynamically cap max_tokens based on the available token budget per request.

```python
import matplotlib.pyplot as plt
import numpy as np

```

```

labels = summary_df["workload"].tolist()

# p50/p90/p99
p50 = summary_df["p50_s"].values
p90 = summary_df["p90_s"].values
p99 = summary_df["p99_s"].values

x = np.arange(len(labels))
w = 0.25

plt.figure()
plt.bar(x - w, p50, width=w, label="p50")
plt.bar(x,     p90, width=w, label="p90")
plt.bar(x + w, p99, width=w, label="p99")
plt.xticks(x, labels, rotation=20)
plt.ylabel("Latency (s)")
plt.title("Latency percentiles by workload")
plt.legend()
plt.show()

# RPS
plt.figure()
plt.bar(labels, summary_df["rps"].values)
plt.ylabel("Requests/sec")
plt.title("Throughput (RPS) by workload")
plt.xticks(rotation=20)
plt.show()

```





Plot Interpretation

These two plots summarize the impact of prompt-length mix on both latency tail and throughput at a fixed concurrency (c=8, 200 total requests each).

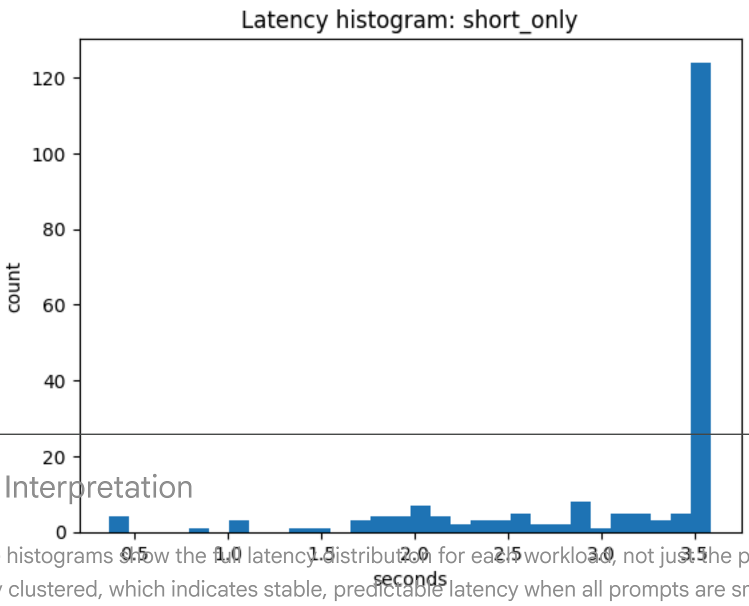
As we move from short_only -> mix_90_10 -> mix_70_30, the proportion of long prompts increases, which pushes up p90/p99 latency and reduces overall RPS. This is the classic “mixed-length serving” effect.

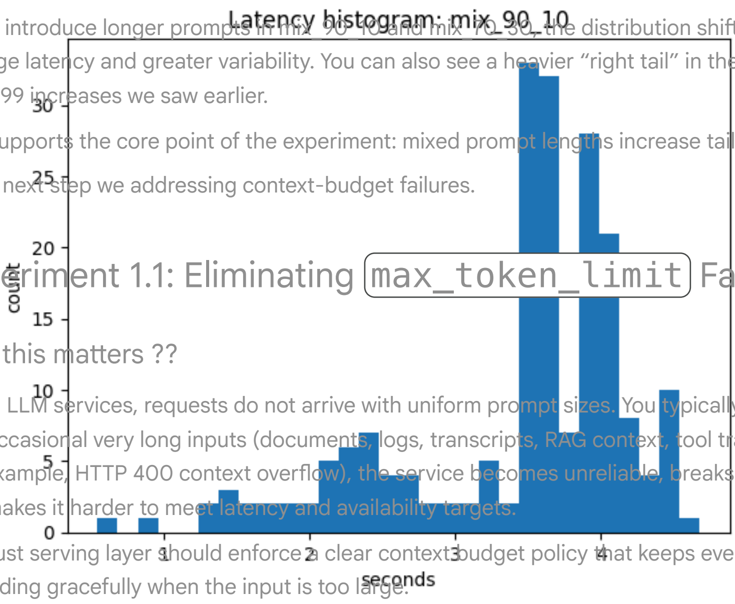
Long prompts spend much more time in prefill and consume more KV-cache and batching budget, so they slow down batch completion and increase queueing for everyone, which shows up most clearly in tail latency.

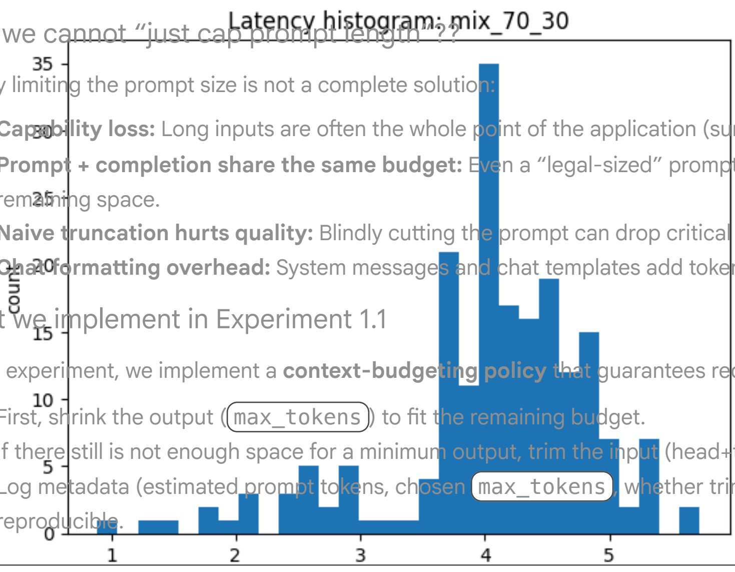
The throughput chart reinforces this: more long requests means fewer requests completed per second. Also note that the latency percentiles here are computed over successful (ok) requests.

The few failed requests from long prompts are not included in the percentile bars, and those failures are exactly what we address next with the max_tokens safety and dynamic max_tokens fix.

```
for name, df in all_dfs.items():
    if "latency_s" in df.columns:
        vals = df[df["latency_s"].notna()]["latency_s"].values
        plt.figure()
        plt.hist(vals, bins=30)
        plt.title(f"Latency histogram: {name}")
        plt.xlabel("seconds")
        plt.ylabel("count")
        plt.show()

```







Plot Interpretation

These histograms show the full latency distribution for each workload, not just the percentile summaries. The short_only workload is tightly clustered, which indicates stable, predictable latency when all prompts are small.

As we introduce longer prompts in mix_90_10 and mix_70_30, the distribution shifts to the right and spreads out, meaning higher average latency and greater variability. You can also see a heavier “right tail” in the mixed workloads, which is exactly what drives the p90/p99 increases we saw earlier.

This supports the core point of the experiment: mixed prompt lengths increase tail latency and make serving behavior less predictable.

In the next step we addressing context-budget failures.

Experiment 1.1: Eliminating `max_token_limit` Failures with Context-Budgeting

Why this matters ??

In real LLM services, requests do not arrive with uniform prompt sizes. You typically see a mix of short questions, medium chat turns, and occasional very long inputs (documents, logs, transcripts, RAG context, tool traces). If the system sometimes returns hard failures (for example, HTTP 400 context overflow), the service becomes unreliable, breaks downstream pipelines (agents, batch jobs, retries), and makes it harder to meet latency and availability targets.

A robust serving layer should enforce a clear context-budget policy that keeps every request within the model’s context window, while degrading gracefully when the input is too large.

Why we cannot “just cap prompt length” ??

Simply limiting the prompt size is not a complete solution:

- **Capability loss:** Long inputs are often the whole point of the application (summarization, document Q&A, log diagnosis).
- **Prompt + completion share the same budget:** Even a “legal-sized” prompt can overflow if `max_tokens` is too large for the remaining space.
- **Naive truncation hurts quality:** Blindly cutting the prompt can drop critical context and silently degrade answer quality.
- **Chat formatting overhead:** System messages and chat templates add tokens that a simple character cap does not account for.

What we implement in Experiment 1.1

In this experiment, we implement a **context-budgeting policy** that guarantees requests fit the model’s context window:

- First, shrink the output (`max_tokens`) to fit the remaining budget.
- If there still is not enough space for a minimum output, trim the input (head+tail) to create room.
- Log metadata (estimated prompt tokens, chosen `max_tokens`, whether trimming occurred) so the behavior is measurable and reproducible.

```
# clean out previous instances of vLLM if you want to start a new server

!pkill -9 -f "vllm.entrypoints.openai.api_server" || true
!pkill -9 -f "EngineCore" || true
!pkill -9 -f "vllm" || true
!sleep 2
!nvidia-smi
```

```
^C
^C
^C
```

```
Sun Jan 11 22:01:00 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA L4                      Off |   00000000:00:03.0 Off |                    0 |
| N/A   54C    P8             18W /   72W |       3MiB /  23034MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

```
!lsof -i :8000 || true


```

```
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 1144 root   79u  IPv4 606432      0t0  TCP localhost:49490->localhost:8000 (CLOSE_WAIT)
```

```

MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"
PORT = 8000

MAX_MODEL_LEN = 4096
GPU_UTIL = 0.70
DTYPE = "half"

cmd = f"""
nohup python -m vllm.entrypoints.openai.api_server \
  --model {MODEL_ID} \
  --host 127.0.0.1 \
  --port {PORT} \
  --max-model-len {MAX_MODEL_LEN} \
  --gpu-memory-utilization {GPU_UTIL} \
  --dtype {DTYPE} \
  --disable-log-stats \
  > vllm_server.log 2>&1 &
"""

print("Launching vLLM server...")
!bash -lc "$cmd"
!tail -n 30 vllm_server.log

```

```
Launching vLLM server...
```

Function to check the successful startup of the server.

```
BASE = "http://127.0.0.1:8000"

def wait_for_server(timeout_s=600):
    t0 = time.time()
    while time.time() - t0 < timeout_s:
        try:
            r = requests.get(f"{BASE}/v1/models", timeout=2)
            if r.status_code == 200:
                return True, r.json()
        except Exception:
            pass
        time.sleep(2)
    return False, None

ok, payload = wait_for_server()
print("Ready:", ok)
if ok:
    print("Models:", [m["id"] for m in payload.get("data", [])])
else:
    print("Not ready yet. Check logs:")
    !tail -n 60 vllm_server.log

```

```
Ready: True
Models: ['Qwen/Qwen2.5-3B-Instruct']
```

Dynamic Output Budgeting and Prompt Trimming to Avoid Context Overflow

```
SAFETY_MARGIN = 16          # keep small headroom to avoid borderline failures
DESIRED_MAX_TOKENS = 128    # same as Experiment 1
MIN_OUTPUT_TOKENS = 32      # minimum output you want to guarantee

def count_prompt_tokens(messages):
    """
    Counts tokens for the full chat-formatted prompt that will be sent to vLLM.
    Qwen tokenizer supports apply_chat_template().
    """
    ids = tok.apply_chat_template(
        messages,
        tokenize=True,
        add_generation_prompt=True,
        return_tensors=None
    )
    return len(ids)

def head_tail_trim_text_by_tokens(text, target_tokens, head_frac=0.35):
    """
    Keep some head + some tail tokens, drop middle.
    """
    ids = tok.encode(text, add_special_tokens=False)
    if len(ids) <= target_tokens:
        return text, False

    head_tokens = int(target_tokens * head_frac)
    tail_tokens = target_tokens - head_tokens

    trimmed_ids = ids[:head_tokens] + ids[-tail_tokens:]
    return tok.decode(trimmed_ids), True

def build_payload_with_budget(prompt_text,
                              desired_max_tokens=DESIRED_MAX_TOKENS,
                              min_output_tokens=MIN_OUTPUT_TOKENS,
                              safety_margin=SAFETY_MARGIN,
                              temperature=0.0):
    """
    Returns (payload, meta).
    - Ensures no context overflow.
    - If needed, trims user prompt (head+tail) to make room for min output tokens.
    """

    messages = [{"role": "user", "content": prompt_text}]
    ptoks = count_prompt_tokens(messages)

    # First attempt: just shrink output tokens if needed
    max_out = min(desired_max_tokens, MAX_MODEL_LEN - safety_margin - ptoks)

    trimmed = False
    original_ptoks = ptoks
    original_prompt = prompt_text

    # If not enough room even for min output -> trim input to create room
    if max_out < min_output_tokens:
        target_prompt_budget = MAX_MODEL_LEN - safety_margin - min_output_tokens
        if target_prompt_budget <= 0:
            raise RuntimeError("Model context too small after safety margin.")

        # We need to reduce prompt tokens from ptoks down to target_prompt_budget.
        # We'll trim only the USER CONTENT tokens.
        #
        # Use a simple search on how many user-content tokens we can keep.
        # (Because chat template adds overhead tokens.)
        lo, hi = 0, len(tok.encode(prompt_text, add_special_tokens=False))
        best_text = ""
        best_ptoks = None

        while lo <= hi:
            mid = (lo + hi) // 2
            candidate_text, _ = head_tail_trim_text_by_tokens(prompt_text, mid)
            candidate_msgs = [{"role": "user", "content": candidate_text}]
            candidate_ptoks = count_prompt_tokens(candidate_msgs)

            if candidate_ptoks <= target_prompt_budget:
                best_text = candidate_text
                best_ptoks = candidate_ptoks
                lo = mid + 1
            else:
                hi = mid - 1

        if best_ptoks is None:
            raise RuntimeError(
                f"Prompt too long to fit even after trimming. "
                f"prompt_tokens={ptoks}, budget={target_prompt_budget}"
            )

        prompt_text = best_text
        trimmed = True
        messages = [{"role": "user", "content": prompt_text}]
        ptoks = best_ptoks

        max_out = min(desired_max_tokens, MAX_MODEL_LEN - safety_margin - ptoks)

    # Final guard
    if max_out <= 0:
        raise RuntimeError(
            f"No room for output tokens. prompt_tokens={ptoks}, "
            f"max_len={MAX_MODEL_LEN}, safety_margin={safety_margin}"
        )

    payload = {
        "model": MODEL_ID,
        "messages": messages,
        "max_tokens": int(max_out),
        "temperature": temperature,
    }

    meta = {
        "prompt_tokens_est": ptoks,
        "max_tokens_used": int(max_out),
        "trimmed": trimmed,
        "prompt_tokens_est_original": original_ptoks,
        "original_prompt_len_chars": len(original_prompt),
        "final_prompt_len_chars": len(prompt_text),
    }
    return payload, meta

```

Request Execution with Dynamic Token Budgeting

```


def one_request(prompt, desired_max_tokens=128, temperature=0.0, timeout=120):
    t0 = time.perf_counter()

    payload, meta = build_payload_with_budget(
        prompt_text=prompt,
        desired_max_tokens=desired_max_tokens,
        min_output_tokens=MIN_OUTPUT_TOKENS,
        safety_margin=SAFETY_MARGIN,
        temperature=temperature,
    )

    r = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=timeout)
    latency = time.perf_counter() - t0

    if r.status_code != 200:
        raise RuntimeError(f"HTTP {r.status_code}: {r.text[:800]}")

    j = r.json()
    usage = j.get("usage", {})

    return {
        "latency_s": latency,
        "prompt_tokens": usage.get("prompt_tokens"),
        "completion_tokens": usage.get("completion_tokens"),
        "total_tokens": usage.get("total_tokens"),
        "status_code": r.status_code,
        # extra debugging info (useful for your writeup)
        **meta
    }

```

```
# smoke test of one_request function

test_prompt = "Explain in 2 sentences what chunked prefill does in vLLM."
out = one_request(test_prompt, desired_max_tokens=128, temperature=0.0, timeout=120)
out

```

```
{'latency_s': 1.3368042790007166,
 'prompt_tokens': 46,
 'completion_tokens': 49,
 'total_tokens': 95,
 'status_code': 200,
 'prompt_tokens_est': 46,
 'max_tokens_used': 128,
 'trimmed': False,
 'prompt_tokens_est_original': 46,
 'original_prompt_len_chars': 57,
 'final_prompt_len_chars': 57}
```

Load Test Harness Using Budget-Aware Requests (Dynamic max_tokens + Optional Prompt Trimming)

```
def run_load(prompts, concurrency=8, total_requests=200, desired_max_tokens=128, desc="load"):
    rng = random.Random(123)
    picked = [rng.choice(prompts) for _ in range(total_requests)]

    results = []
    t_start = time.perf_counter()

    def task(i):
        p = picked[i]
        try:
            res = one_request(p, desired_max_tokens=desired_max_tokens, temperature=0.0, timeout=180)
            res["ok"] = True
        except Exception as e:
            res = {"ok": False, "error": str(e), "latency_s": None,
                   "prompt_tokens": None, "completion_tokens": None, "total_tokens": None}
        return res

    ok_ct = 0
    fail_ct = 0

    with cf.ThreadPoolExecutor(max_workers=concurrency) as ex:
        futs = [ex.submit(task, i) for i in range(total_requests)]
        pbar = tqdm(total=total_requests, desc=desc)
        for f in cf.as_completed(futs):
            r = f.result()
            results.append(r)
            if r.get("ok"):
                ok_ct += 1
            else:
                fail_ct += 1
            pbar.update(1)
            pbar.set_postfix(ok=ok_ct, fail=fail_ct)
        pbar.close()

    t_end = time.perf_counter()

    df = pd.DataFrame(results)
    ok_df = df[df["ok"] == True].copy()

    summary = {
        "concurrency": concurrency,
        "total_requests": total_requests,
        "ok": int(ok_df.shape[0]),
        "fail": int(df.shape[0] - ok_df.shape[0]),
        "wall_time_s": t_end - t_start,
        "rps": ok_df.shape[0] / (t_end - t_start) if (t_end - t_start) > 0 else None,
        "p50_s": float(ok_df["latency_s"].quantile(0.50)) if len(ok_df) else None,
        "p90_s": float(ok_df["latency_s"].quantile(0.90)) if len(ok_df) else None,
        "p99_s": float(ok_df["latency_s"].quantile(0.99)) if len(ok_df) else None,
        "mean_s": float(ok_df["latency_s"].mean()) if len(ok_df) else None,
    }
    return df, summary



```

Run Safe (Budget-Aware) Load Tests Across Workloads

```python
workloads = [
    ("short_only", "short_only.jsonl"),
    ("mix_90_10", "mix_90_10.jsonl"),
    ("mix_70_30", "mix_70_30.jsonl"),
]


all_summaries = []
all_dfs = {}

for name, fn in tqdm(workloads, desc="Workloads (safe)"):
    print("\nRunning:", name)
    df, summ = run_load(
        prompt_sets[fn],
        concurrency=8,
        total_requests=200,
        desired_max_tokens=128,
        desc=f"{name} (c=8, safe)"
    )
    all_dfs[name] = df
    all_summaries.append({"workload": name, **summ})
    print(summ)

summary_df_safe = pd.DataFrame(all_summaries)
summary_df_safe
```

```
Workloads (safe): 100%                                   3/3 [04:30<00:00, 92.91s/it]

Running: short_only
short_only (c=8, safe): 100%                             200/200 [01:19<00:00,  2.30it/s, fail=0, ok=200]
{'concurrency': 8, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 79.25956869500078, 'rps': 2.523354

Running: mix_90_10
mix_90_10 (c=8, safe): 100%                              200/200 [01:28<00:00,  2.34it/s, fail=0, ok=200]
{'concurrency': 8, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 88.17387503600003, 'rps': 2.268245

Running: mix_70_30
mix_70_30 (c=8, safe): 100%                              200/200 [01:42<00:00,  3.26it/s, fail=0, ok=200]
{'concurrency': 8, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 102.89451057299993, 'rps': 1.94373
```

	workload	concurrency	total_requests	ok	fail	wall_time_s	rps	p50_s	p90_s	p99_s	mean_s
0	short_only	8	200	200	0	79.259569	2.523355	3.556058	3.573445	3.581660	3.106944
1	mix_90_10	8	200	200	0	88.173875	2.268246	3.640731	4.242185	4.532153	3.469050
2	mix_70_30	8	200	200	0	102.894511	1.943738	4.175408	4.893263	5.393245	4.073671

Conclusion

In Experiment 1.1 (the budget-aware request builder), the key improvement is reliability: all three workloads now complete with 0 failures (200/200 OK), including mixes that previously hit HTTP 400 context overflow when long prompts left insufficient room for the requested completion (max_tokens).

This confirms the fix works as intended, because each request now dynamically adjusts max_tokens to fit the remaining context window and trims the prompt only when necessary to guarantee a minimum output budget.

Performance trends remain consistent with Experiment 1: as the proportion of long prompts increases (short_only -> mix_90_10 -> mix_70_30), latency rises and throughput (RPS) drops due to heavier prefill and decoding work, but crucially this happens without any request-level hard failures, making the system robust under mixed prompt lengths.

Experiment 2: Diagnosing Tail Latency with a “Bad” vLLM Batching Config

```
!pkill -f "vllm.entrypoints.openai.api_server" || true
!pkill -f "VLLM::EngineCore" || true
!pkill -f "EngineCore" || true
time.sleep(2)

!rm -f vllm_bad.log
print("killed + log removed")

```

```
^C
^C
^C
killed + log removed
```

Creating a Bad Config for LLM Server

To replicate a **“chunked prefill effectively OFF”** situation

Why this is a “bad” config: We set `--max-num-batched-tokens 8192` (very large) while leaving `--max-num-seqs` unconstrained (defaults can be high). This encourages vLLM to pack big prefills into a batch, which can let long prompts dominate compute and delay short requests.

Chunked prefill caveat: Newer vLLM versions have chunked prefill **enabled by default**, but its *practical* benefit depends on admission and batching knobs. If `max-num-batched-tokens` is too permissive (and/or `max-num-seqs` allows over-admission), long prefills can still monopolize batch capacity, effectively **diminishing chunked prefill’s ability to interleave work and protect short requests**.

Head-of-line blocking risk: With mixed workloads (50/50 short and long), long-prefill requests can occupy most of the batch token budget, causing short prompts to wait behind long prefills, inflating **p90/p99** latency.

```
!nohup python3 -m vllm.entrypoints.openai.api_server \
  --model Qwen/Qwen2.5-3B-Instruct \
  --host 127.0.0.1 --port 8000 \
  --dtype half \
  --max-model-len 4096 \
  --gpu-memory-utilization 0.70 \
  --disable-log-stats \
  --enforce-eager \
  --max-num-batched-tokens 8192 \
  --long-prefill-token-threshold 1024 \
  > vllm_bad.log 2>&1 </dev/null &

```

```
BASE = "http://127.0.0.1:8000"

def wait_for_server(timeout_s=600, poll_s=2):
    t0 = time.time()
    last_err = None
    while time.time() - t0 < timeout_s:
        try:
            r = requests.get(f"{BASE}/v1/models", timeout=2)
            if r.status_code == 200:
                return True, r.json()
            last_err = f"HTTP {r.status_code}: {r.text[:200]}"
        except Exception as e:
            last_err = f"{type(e).__name__}: {e}"
        time.sleep(poll_s)
    return False, last_err

ok, models_or_err = wait_for_server()
print("Ready:", ok)

if ok:
    print("Models:", [m.get("id") for m in models_or_err.get("data", [])])
else:
    print("Not ready yet. Last error:", models_or_err)
    print("Check logs:")
    !tail -n 80 vllm_bad.log

```

```
Ready: True
Models: ['Qwen/Qwen2.5-3B-Instruct']
```

```
!tail -n 60 vllm_bad.log
```

Show hidden output

Load Tokenizer

```python
import requests
from transformers import AutoTokenizer

BASE = "http://127.0.0.1:8000"
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"

tok = AutoTokenizer.from_pretrained(MODEL_ID, use_fast=True)

models = requests.get(f"{BASE}/v1/models").json()
model_info = next(m for m in models["data"] if m["id"] == MODEL_ID)
MAX_MODEL_LEN = int(model_info.get("max_model_len", 4096))

print("tok loaded")
print("MAX_MODEL_LEN from server:", MAX_MODEL_LEN)

```

```
tok loaded
MAX_MODEL_LEN from server: 4096
```

Run Load Test on Bad Server Config

```python
import pandas as pd

WL_FILE = "mix_50_50.jsonl"
CONCURRENCY = 32
TOTAL_REQ = 200
DESIRED_MAX_TOKENS = 16


df_bad, summ_bad = run_load(
    prompt_sets[WL_FILE],
    concurrency=CONCURRENCY,
    total_requests=TOTAL_REQ,
    desired_max_tokens=DESIRED_MAX_TOKENS,
    desc=f"BAD_unfair | {WL_FILE} | c={CONCURRENCY}"
)


print("BAD summary:", summ_bad)
df_bad.head()

```

```
BAD_unfair | mix_50_50.jsonl | c=32: 100%                    200/200 [00:42<00:00,  9.61it/s, fail=0, ok=200]
BAD summary: {'concurrency': 32, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 42.19113462899986, '
```

	latency_s	prompt_tokens	completion_tokens	total_tokens	status_code	prompt_tokens_est	max_tokens_used	trim
0	8.678932	37	16	53	200	37	16	
1	8.703615	40	16	56	200	40	16	
2	8.683890	42	16	58	200	42	16	
3	8.700286	37	16	53	200	37	16	
4	8.697547	37	16	53	200	37	16	

Compute Overall and Short-Prompt Tail Latency Metrics (p50/p90/p99)

```python
def tail_metrics(df, label, short_cutoff_tokens=64):
    ok = df[df["ok"] == True].copy()

    # overall
    overall = {
        "label": label,
        "subset": "overall",
        "n": len(ok),
        "p50": float(ok["latency_s"].quantile(0.50)),
        "p90": float(ok["latency_s"].quantile(0.90)),
        "p99": float(ok["latency_s"].quantile(0.99)),
        "mean": float(ok["latency_s"].mean()),
    }

    # short-only subset (inside the mixed workload)
    short = ok[ok["prompt_tokens"] <= short_cutoff_tokens].copy()
    if len(short) == 0:
        short_stats = {"label": label, "subset": f"short<= {short_cutoff_tokens}", "n": 0,
                       "p50": None, "p90": None, "p99": None, "mean": None}
    else:
        short_stats = {
            "label": label,
            "subset": f"short<= {short_cutoff_tokens}",
            "n": len(short),
            "p50": float(short["latency_s"].quantile(0.50)),
            "p90": float(short["latency_s"].quantile(0.90)),
            "p99": float(short["latency_s"].quantile(0.99)),
            "mean": float(short["latency_s"].mean()),
        }

    return overall, short_stats

```

```python
overall_bad, short_bad = tail_metrics(df_bad, "BAD_unfair", short_cutoff_tokens=64)
overall_bad, short_bad

```

```
({'label': 'BAD_unfair',
  'subset': 'overall',
  'n': 200,
  'p50': 6.181310366500156,
  'p90': 10.99196271639994,
  'p99': 11.637151490029915,
  'mean': 6.493085269189996},
 {'label': 'BAD_unfair',
  'subset': 'short<= 64',
  'n': 108,
  'p50': 5.480035559000271,
  'p90': 8.717383960199822,
  'p99': 10.618033962269847,
  'mean': 5.465330376453692})
```

Expermient 3: Tail latency and Throughput variation with Concurrency Sweep

This sweep isolates scheduling effects by holding workload and max batched token length constant, then measuring how tail latency and throughput evolve with concurrency.

```python
from google.colab import drive
drive.mount("/content/drive")

import os, json, shutil, time
import pandas as pd

WL_FILE = "mix_50_50.jsonl"
TOTAL_REQ = 200
DESIRED_MAX_TOKENS = 16
CONCURRENCY_LIST = [1, 2, 4, 8, 16, 32, 48, 64]

SHORT_CUTOFF = 64
LONG_CUTOFF  = 2048

LABEL = "BAD_unfair"

RUN_DIR = "/content/drive/MyDrive/vllm_sweeps/exp2_bad_unfair"
os.makedirs(RUN_DIR, exist_ok=True)

CKPT_CSV      = os.path.join(RUN_DIR, "concurrency_checkpoint.csv")
MANIFEST_JSON = os.path.join(RUN_DIR, "concurrency_manifest.json")

SAVE_RAW_DF = False
RAW_DIR = os.path.join(RUN_DIR, "raw_dfs")
if SAVE_RAW_DF:
    os.makedirs(RAW_DIR, exist_ok=True)

manifest = {
    "label": LABEL,
    "workload_file": WL_FILE,
    "total_requests": TOTAL_REQ,
    "desired_max_tokens": DESIRED_MAX_TOKENS,
    "concurrency_list": CONCURRENCY_LIST,
    "short_cutoff": SHORT_CUTOFF,
    "long_cutoff": LONG_CUTOFF,
}

if os.path.exists(MANIFEST_JSON):
    try:
        old = json.loads(open(MANIFEST_JSON, "r").read())
        if old != manifest:
            print("Manifest differs from existing checkpoint. You may be mixing runs.")
            print("Existing:", old)
            print("New:", manifest)
    except Exception:
        print("Could not read existing manifest. Proceeding.")
else:
    with open(MANIFEST_JSON, "w") as f:
        json.dump(manifest, f, indent=2)

def summarize_run(df, summ, label, workload, concurrency):
    ok = df[df["ok"] == True].copy()

    row = {
        "label": label,
        "workload": workload,
        "concurrency": int(concurrency),

        "ok": summ.get("ok"),
        "fail": summ.get("fail"),
        "rps": summ.get("rps"),
        "wall_time_s": summ.get("wall_time_s"),
        "p50_s": summ.get("p50_s"),
        "p90_s": summ.get("p90_s"),
        "p99_s": summ.get("p99_s"),
        "mean_s": summ.get("mean_s"),
    }

    short = ok[ok["prompt_tokens"] <= SHORT_CUTOFF]
    row["short_n"] = int(len(short))
    row["short_p50_s"] = float(short["latency_s"].quantile(0.50)) if len(short) else None
    row["short_p90_s"] = float(short["latency_s"].quantile(0.90)) if len(short) else None
    row["short_p99_s"] = float(short["latency_s"].quantile(0.99)) if len(short) else None

    longg = ok[ok["prompt_tokens"] >= LONG_CUTOFF]
    row["long_n"] = int(len(longg))
    row["long_p50_s"] = float(longg["latency_s"].quantile(0.50)) if len(longg) else None
    row["long_p90_s"] = float(longg["latency_s"].quantile(0.90)) if len(longg) else None
    row["long_p99_s"] = float(longg["latency_s"].quantile(0.99)) if len(longg) else None

    return row

def load_ckpt_df(path):
    if not os.path.exists(path):
        return pd.DataFrame()
    df = pd.read_csv(path)
    if "concurrency" in df.columns:
        df["concurrency"] = pd.to_numeric(df["concurrency"], errors="coerce").astype("Int64")
    return df

def done_concurrencies(df_ckpt):
    if df_ckpt.empty:
        return set()
    good = df_ckpt[df_ckpt["concurrency"].notna()].copy()
    if "status" in good.columns:
        good = good[good["status"].isna()].copy()
    return set(int(x) for x in good["concurrency"].dropna().tolist())

def append_ckpt_row(row, path):
    df_row = pd.DataFrame([row])
    header = not os.path.exists(path)
    df_row.to_csv(path, mode="a", header=header, index=False)

def maybe_save_raw_df(df, concurrency):
    if not SAVE_RAW_DF:
        return
    outp = os.path.join(RAW_DIR, f"raw_c{int(concurrency)}.parquet")
    try:
        df.to_parquet(outp, index=False)
    except Exception:
        outp = os.path.join(RAW_DIR, f"raw_c{int(concurrency)}.csv")
        df.to_csv(outp, index=False)

ckpt_df = load_ckpt_df(CKPT_CSV)
done = done_concurrencies(ckpt_df)
print("Loaded checkpoint rows:", len(ckpt_df))
print("Completed concurrencies:", sorted(done))

for c in CONCURRENCY_LIST:
    c = int(c)
    if c in done:
        print("Skipping:", c)
        continue

    print("Running:", LABEL, WL_FILE, "c=", c)

    try:
        df, summ = run_load(
            prompt_sets[WL_FILE],
            concurrency=c,
            total_requests=TOTAL_REQ,
            desired_max_tokens=DESIRED_MAX_TOKENS,
            desc=f"{LABEL} | {WL_FILE} | c={c}",
        )

        row = summarize_run(df, summ, label=LABEL, workload=WL_FILE, concurrency=c)
        append_ckpt_row(row, CKPT_CSV)
        maybe_save_raw_df(df, concurrency=c)

        ckpt_df = load_ckpt_df(CKPT_CSV)
        done = done_concurrencies(ckpt_df)

    except Exception as e:
        fail_row = {
            "label": LABEL,
            "workload": WL_FILE,
            "concurrency": c,
            "ok": None,
            "fail": None,
            "rps": None,
            "wall_time_s": None,
            "p50_s": None,
            "p90_s": None,
            "p99_s": None,
            "mean_s": None,
            "short_n": None,
            "short_p50_s": None,
            "short_p90_s": None,
            "short_p99_s": None,
            "long_n": None,
            "long_p50_s": None,
            "long_p90_s": None,
            "long_p99_s": None,
            "status": "run_failed",
            "error": type(e).__name__,
        }
        append_ckpt_row(fail_row, CKPT_CSV)
        print("run_load failed:", type(e).__name__, str(e)[:200])

summary_df = load_ckpt_df(CKPT_CSV).copy()
if "status" in summary_df.columns:
    summary_df = summary_df[summary_df["status"].isna()].copy()

summary_df = summary_df.sort_values("concurrency").reset_index(drop=True)

print("Concurrency sweep done/resumed. Summary DF loaded from Drive:")
display(summary_df)

FINAL_CSV = os.path.join(RUN_DIR, "summary_df_final.csv")
summary_df.to_csv(FINAL_CSV, index=False)
print("Final snapshot saved:", FINAL_CSV)

```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
Loaded checkpoint rows: 0
Completed concurrencies: []
Running: BAD_unfair mix_50_50.jsonl c= 1
BAD_unfair | mix_50_50.jsonl | c=1: 100%                200/200 [02:32<00:00,  1.22it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 2
BAD_unfair | mix_50_50.jsonl | c=2: 100%                200/200 [01:25<00:00,  2.07it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 4
BAD_unfair | mix_50_50.jsonl | c=4: 100%                200/200 [00:55<00:00,  3.03it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 8
BAD_unfair | mix_50_50.jsonl | c=8: 100%                200/200 [00:40<00:00,  6.62it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 16
BAD_unfair | mix_50_50.jsonl | c=16: 100%               200/200 [00:33<00:00,  7.47it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 32
BAD_unfair | mix_50_50.jsonl | c=32: 100%               200/200 [00:33<00:00, 13.21it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 48
BAD_unfair | mix_50_50.jsonl | c=48: 100%               200/200 [00:32<00:00, 10.68it/s, fail=0, ok=200]
Running: BAD_unfair mix_50_50.jsonl c= 64
BAD_unfair | mix_50_50.jsonl | c=64: 100%               200/200 [00:32<00:00, 35.51it/s, fail=0, ok=200]
Concurrency sweep done/resumed. Summary DF loaded from Drive:
```

	label	workload	concurrency	ok	fail	rps	wall_time_s	p50_s	p90_s	p99_s	mean_s	shor
0	BAD_unfair	mix_50_50.jsonl	1	200	0	1.310728	152.586979	0.455211	1.264692	1.302439	0.762749	
1	BAD_unfair	mix_50_50.jsonl	2	200	0	2.350512	85.087858	0.753871	1.356037	1.788700	0.848931	
2	BAD_unfair	mix_50_50.jsonl	4	200	0	3.588961	55.726434	0.974289	1.860058	2.692315	1.103148	
3	BAD_unfair	mix_50_50.jsonl	8	200	0	4.897939	40.833506	1.472894	2.478667	3.694609	1.622580	
4	BAD_unfair	mix_50_50.jsonl	16	200	0	5.902047	33.886548	2.537444	4.032396	6.204649	2.649116	
5	BAD_unfair	mix_50_50.jsonl	32	200	0	5.912537	33.826429	5.829220	7.589303	9.536481	5.360095	
6	BAD_unfair	mix_50_50.jsonl	48	200	0	6.029853	33.168302	7.492978	11.804981	13.755477	7.446991	
7	BAD_unfair	mix_50_50.jsonl	64	200	0	5.817353	34.379898	9.910197	16.760385	21.667766	10.589799	

```
Final snapshot saved: /content/drive/MyDrive/vllm_sweeps/exp2_bad_unfair/summary_df_final.csv
```

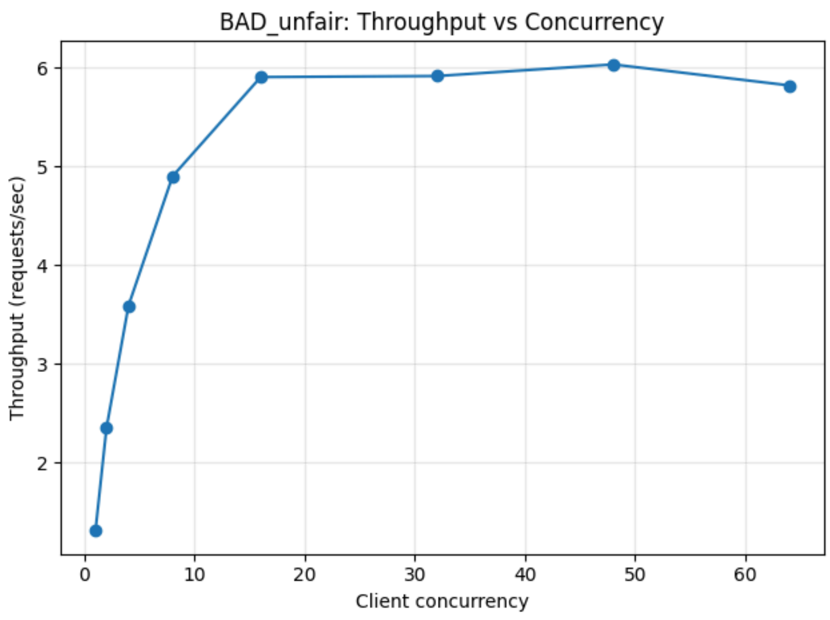
Next steps: Generate code with summary_df New interactive sheet

Plot some meaningful charts from Sweep results

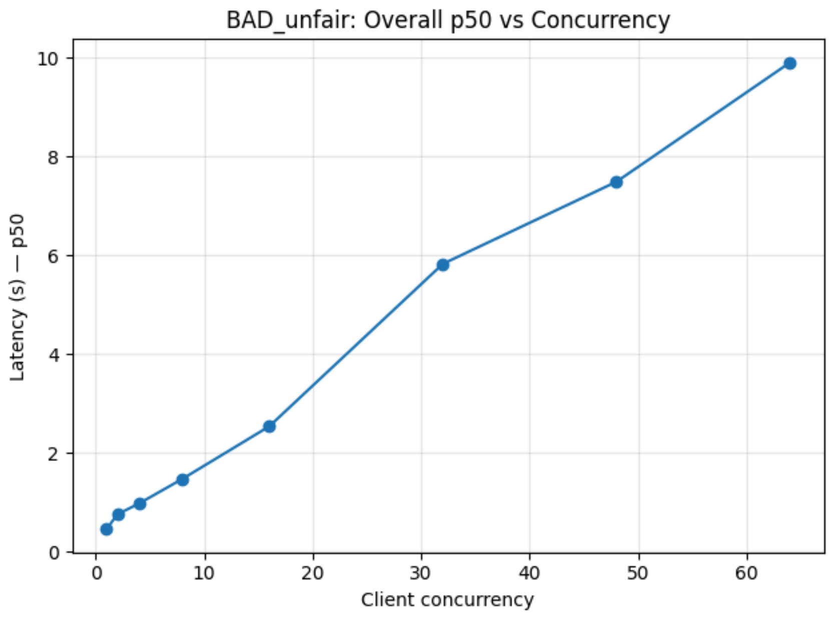
```python
df = summary_df.copy()

num_cols = [
    "concurrency","rps","wall_time_s","p50_s","p90_s","p99_s","mean_s",
    "short_p50_s","short_p90_s","short_p99_s",
    "long_p50_s","long_p90_s","long_p99_s",
    "short_n","long_n"
]
for c in num_cols:
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.sort_values("concurrency").reset_index(drop=True)

# output directory
PLOT_DIR = "plots_bad_sweep"
os.makedirs(PLOT_DIR, exist_ok=True)


def savefig(name):
    path = os.path.join(PLOT_DIR, name)
    plt.tight_layout()
    plt.savefig(path, dpi=200, bbox_inches="tight")
    print("Saved:", path)


# Throughput vs concurrency
plt.figure()
plt.plot(df["concurrency"], df["rps"], marker="o")
plt.xlabel("Client concurrency")
plt.ylabel("Throughput (requests/sec)")
plt.title("BAD_unfair: Throughput vs Concurrency")
plt.grid(True, alpha=0.3)
savefig("bad_throughput_vs_concurrency.png")
```

```python
plt.show()

# Overall tail latency vs concurrency (p50 / p90 / p99)
for metric, title in [
    ("p50_s", "BAD_unfair: Overall p50 vs Concurrency"),
    ("p90_s", "BAD_unfair: Overall p90 vs Concurrency"),
    ("p99_s", "BAD_unfair: Overall p99 vs Concurrency"),
]:
    plt.figure()
    plt.plot(df["concurrency"], df[metric], marker="o")
    plt.xlabel("Client concurrency")
    plt.ylabel(f"Latency (s) — {metric.replace('_s','')}")
    plt.title(title)
    plt.grid(True, alpha=0.3)
    savefig(f"bad_{metric}_vs_concurrency.png")
    plt.show()

# Short vs Long p99 comparison (same plot)
plt.figure()
plt.plot(df["concurrency"], df["short_p99_s"], marker="o", label="short p99")
plt.plot(df["concurrency"], df["long_p99_s"], marker="o", label="long p99")
plt.xlabel("Client concurrency")
plt.ylabel("Latency (s)")
plt.title("BAD_unfair: Short vs Long p99 vs Concurrency")
plt.grid(True, alpha=0.3)
plt.legend()
savefig("bad_short_long_p99_vs_concurrency.png")
plt.show()

# "Knee" visualization: p99 vs RPS (tradeoff curve)
plt.figure()
plt.plot(df["rps"], df["p99_s"], marker="o")
for _, r in df.iterrows():
    plt.text(r["rps"], r["p99_s"], str(int(r["concurrency"])), fontsize=9)

plt.xlabel("Throughput (requests/sec)")
plt.ylabel("Overall p99 latency (s)")
plt.title("BAD_unfair: p99 vs Throughput (labels = concurrency)")
plt.grid(True, alpha=0.3)
savefig("bad_p99_vs_rps.png")
plt.show()

print("Done. Plot folder:", PLOT_DIR)
df[["concurrency","rps","p50_s","p90_s","p99_s","short_p99_s","long_p99_s"]]

```

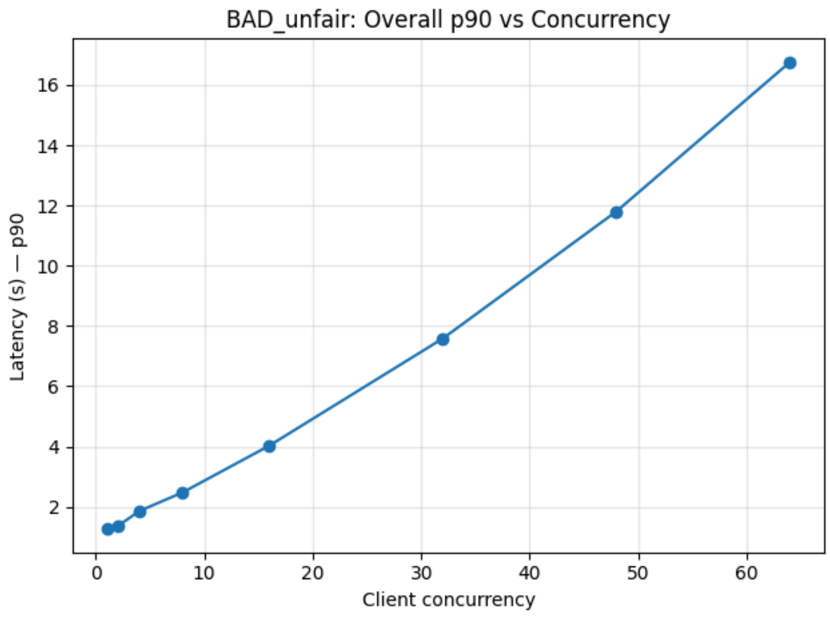
```
Saved: plots_bad_sweep/bad_throughput_vs_concurrency.png
```



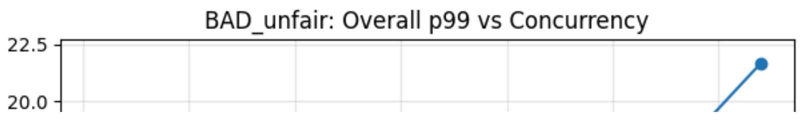
```
Saved: plots_bad_sweep/bad_p50_s_vs_concurrency.png
```

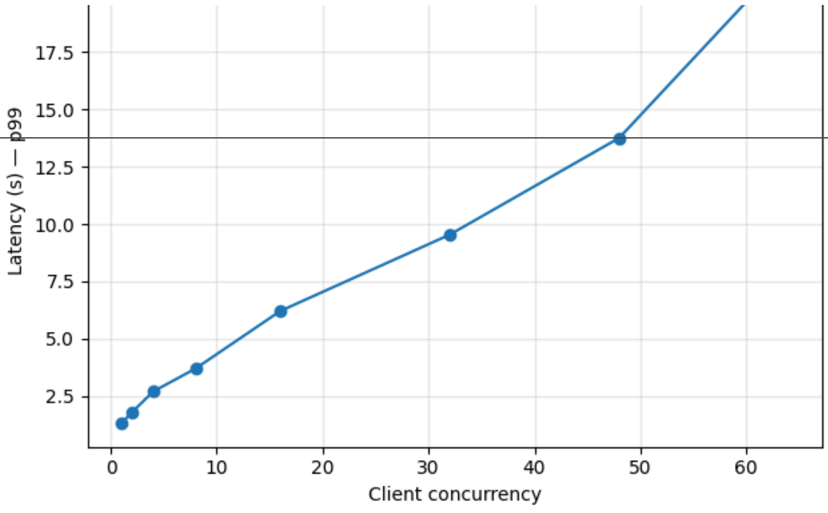


```
Saved: plots_bad_sweep/bad_p90_s_vs_concurrency.png
```

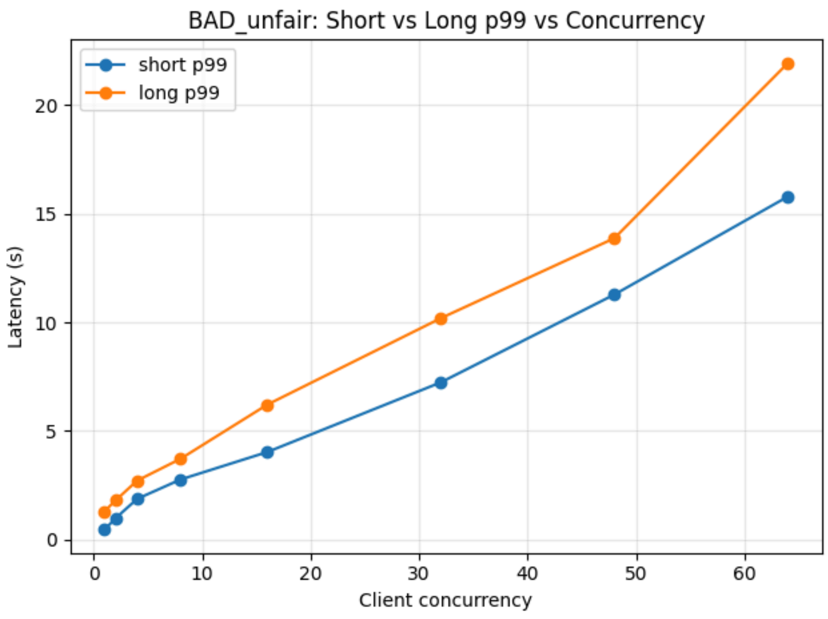


```
Saved: plots_bad_sweep/bad_p99_s_vs_concurrency.png
```

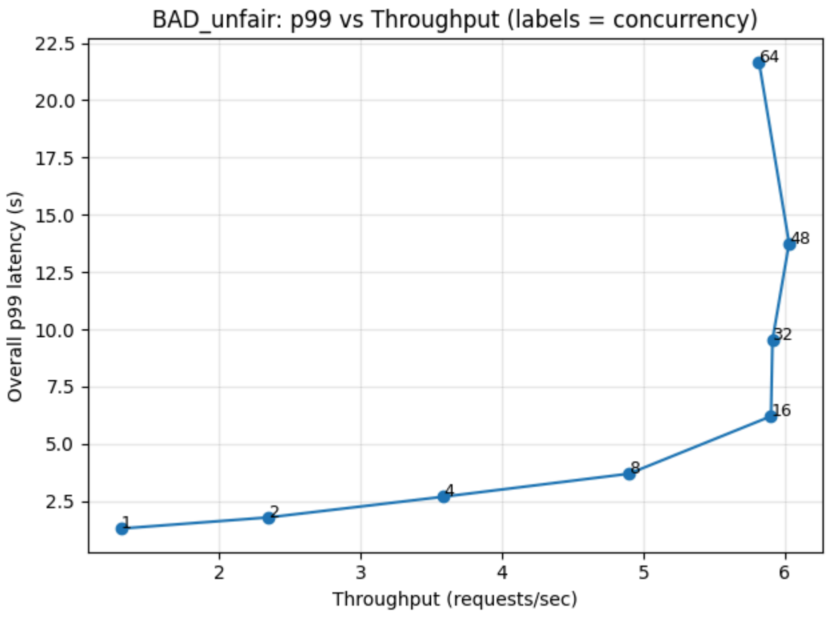




```
Saved: plots_bad_sweep/bad_short_long_p99_vs_concurrency.png
```



```
Saved: plots_bad_sweep/bad_p99_vs_rps.png
```



```
Done. Plot folder: plots_bad_sweep
```

	concurrency	rps	p50_s	p90_s	p99_s	short_p99_s	long_p99_s
0	1	1.310728	0.455211	1.264692	1.302439	0.459093	1.303359
1	2	2.350512	0.753871	1.356037	1.788700	0.975944	1.797350
2	4	3.588961	0.974289	1.860058	2.692315	1.870385	2.711042
3	8	4.897939	1.472894	2.478667	3.694609	2.762490	3.715869
4	16	5.902047	2.537444	4.032396	6.204649	4.027770	6.210381

5	32	5.912537	5.829220	7.589303	9.536481	7.235541	10.189636
6	48	6.029853	7.492978	11.804981	13.755477	11.278168	13.872306
7	64	5.817353	9.910197	16.760385	21.667766	15.790760	21.937133

Conclusion

- **Throughput scales then saturates:** RPS rises from **~1.31 (c=1)** to **~5.90 (c=16)** and then stays around **~5.8–6.0 RPS** through **c=64**, showing the server hits a throughput ceiling.
- **Clear knee and tail blow-up:** overall **p99** grows from **~1.30s (c=1)** to **~6.20s (c=16)**, then jumps to **~9.54s (c=32)** and reaches **~21.67s (c=64)**.
- **Short requests get dragged into the tail:** short **p99** inflates from **~0.46s (c=1)** to **~4.03s (c=16)** and **~15.79s (c=64)**, which indicates strong queueing interference from long prompts.
- **At high concurrency, short and long tails converge:** by **c=64**, short **p99 ~15.79s** and long **p99 ~21.94s**, meaning even short prompts suffer large delays under this bad/unfair configuration.

Saving plots to Drive

```python
LOCAL_PLOT_DIR = "plots_bad_sweep"
DRIVE_PLOT_DIR = "/content/drive/MyDrive/vllm_sweeps/exp2_bad_unfair/plots_bad_sweep"

if os.path.exists(DRIVE_PLOT_DIR):
    shutil.rmtree(DRIVE_PLOT_DIR)
shutil.copytree(LOCAL_PLOT_DIR, DRIVE_PLOT_DIR)

print("Copied plots to:", DRIVE_PLOT_DIR)
```

```
Copied plots to: /content/drive/MyDrive/vllm_sweeps/exp2_bad_unfair/plots_bad_sweep
```

Experiment 4: Sweep of concurrency and `max-num-seqs`

In this sweep, we stress-test a deliberately “bad” vLLM server configuration under a mixed workload (mix_50_50) to quantify how tail latency behaves as system load increases. By sweeping both client concurrency and the server’s admission capacity (max_num_seqs), we expose where throughput stops scaling, where queueing begins to dominate, and how long-prefill requests can degrade the latency of short requests in a shared serving setup. The goal is to establish a clear baseline of failure modes before introducing improved scheduling/configurations in the next steps.

```python
BASE = "http://127.0.0.1:8000"
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"


CHECKPOINT_CSV = "bad_grid_checkpoint.csv"
MANIFEST_JSON  = "bad_grid_manifest.json"


def kill_server():
    os.system('pkill -f "vllm.entrypoints.openai.api_server" || true')
    os.system('pkill -f "VLLM::EngineCore" || true')
    time.sleep(1.5)

def tail_log(log_path, n=60):
    if os.path.exists(log_path):
        print(f"\n--- tail {log_path} (last {n}) ---")
        os.system(f"tail -n {n} {log_path}")
    else:
        print(f"(log missing: {log_path})")

def wait_for_ready(base=BASE, log_path="vllm_bad.log", timeout_s=420):
    t0 = time.time()
    tries = 0
    while True:
        tries += 1
        try:
            r = requests.get(f"{base}/v1/models", timeout=2)
            if r.status_code == 200:
                data = r.json().get("data", [])
                ids = [m.get("id") for m in data]
                print(f" Server ready. Models: {ids}")
                return True
            else:
                print(f"[{tries}] /v1/models status={r.status_code}")
        except Exception as e:
            print(f"[{tries}] not ready yet: {type(e).__name__}")

        alive = os.system('ps -ef | grep -E "vllm.entrypoints.openai.api_server" | grep -v grep > /dev/null
        if not alive:
            print(" vLLM process is NOT running anymore.")
            tail_log(log_path, n=140)
            return False

        if time.time() - t0 > timeout_s:
            print(" Timed out waiting for server.")
            tail_log(log_path, n=140)
            return False

        if tries % 5 == 0:
            tail_log(log_path, n=30)

        time.sleep(2)

def warmup_request():
    payload = {
        "model": MODEL_ID,
        "messages": [{"role": "user", "content": "hi"}],
        "max_tokens": 8,
        "temperature": 0.0,
    }
    try:
        _ = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=30)
    except Exception:
        pass

def start_bad_server(max_num_seqs, log_path="vllm_bad.log"):
    kill_server()
    try:
        os.remove(log_path)
    except FileNotFoundError:
        pass

    cmd = f"""
nohup python3 -m vllm.entrypoints.openai.api_server \
  --model {MODEL_ID} \
  --host 127.0.0.1 --port 8000 \
  --dtype half \
  --max-model-len 4096 \
  --gpu-memory-utilization 0.70 \
  --disable-log-stats \
  --enforce-eager \
  --max-num-batched-tokens 8192 \
  --long-prefill-token-threshold 1024 \
  --max-num-seqs {int(max_num_seqs)} \
  > {log_path} 2>&1 </dev/null &
"""
    os.system(cmd)
    time.sleep(1.0)



def _safe_quantile(series, q):
    if series is None or len(series) == 0:
        return None
    return float(series.quantile(q))

def _safe_mean(series):
    if series is None or len(series) == 0:
        return None
    return float(series.mean())

def subset_metrics(df, short_cutoff_tokens=64, long_cutoff_tokens=2048):
    ok = df[df["ok"] == True].copy()
    lat = ok["latency_s"]

    out = {
        "overall_n": int(len(ok)),
        "overall_p50": _safe_quantile(lat, 0.50),
        "overall_p90": _safe_quantile(lat, 0.90),
        "overall_p99": _safe_quantile(lat, 0.99),
        "overall_mean": _safe_mean(lat),
    }

    short = ok[ok["prompt_tokens"] <= short_cutoff_tokens]
    out.update({
        "short_n": int(len(short)),
        "short_p50": _safe_quantile(short["latency_s"], 0.50),
        "short_p90": _safe_quantile(short["latency_s"], 0.90),
        "short_p99": _safe_quantile(short["latency_s"], 0.99),
        "short_mean": _safe_mean(short["latency_s"]),
    })

    long = ok[ok["prompt_tokens"] >= long_cutoff_tokens]
    out.update({
        "long_n": int(len(long)),
        "long_p50": _safe_quantile(long["latency_s"], 0.50),
        "long_p90": _safe_quantile(long["latency_s"], 0.90),
        "long_p99": _safe_quantile(long["latency_s"], 0.99),
        "long_mean": _safe_mean(long["latency_s"]),
    })
    return out

def row_from_run(label, workload, concurrency, max_num_seqs, df, summ):
    m = subset_metrics(df, short_cutoff_tokens=64, long_cutoff_tokens=2048)
    return {
        "label": label,
        "workload": workload,
        "concurrency": int(concurrency),
        "max_num_seqs": int(max_num_seqs),

        "status": "ok",
        "ok": int(summ.get("ok")) if summ.get("ok") is not None else int((df["ok"] == True).sum()),
        "fail": int(summ.get("fail")) if summ.get("fail") is not None else int((df["ok"] != True).sum()),
        "rps": float(summ.get("rps")) if summ.get("rps") is not None else None,
        "wall_time_s": float(summ.get("wall_time_s")) if summ.get("wall_time_s") is not None else None,
        "summ_p50_s": float(summ.get("p50_s")) if summ.get("p50_s") is not None else None,
        "summ_p90_s": float(summ.get("p90_s")) if summ.get("p90_s") is not None else None,
        "summ_p99_s": float(summ.get("p99_s")) if summ.get("p99_s") is not None else None,
        "summ_mean_s": float(summ.get("mean_s")) if summ.get("mean_s") is not None else None,
    }


def load_checkpoint_df(path=CHECKPOINT_CSV):
    if os.path.exists(path):
        df = pd.read_csv(path)
        # normalize types
        if "max_num_seqs" in df.columns:
            df["max_num_seqs"] = df["max_num_seqs"].astype(int)
        if "concurrency" in df.columns:
            df["concurrency"] = df["concurrency"].astype(int)
        return df
    return pd.DataFrame()

def completed_pairs(df_ckpt):
    # Only treat status=="ok" as completed (you can also include server_failed if you want)
    if df_ckpt.empty:
        return set()
    done = df_ckpt[df_ckpt["status"] == "ok"][["max_num_seqs", "concurrency"]].dropna()
    return set((int(r.max_num_seqs), int(r.concurrency)) for r in done.itertuples(index=False))

def append_row_checkpoint(row, path=CHECKPOINT_CSV):
    df_row = pd.DataFrame([row])
    header = not os.path.exists(path)
    df_row.to_csv(path, mode="a", header=header, index=False)

def write_manifest(payload, path=MANIFEST_JSON):
    # don't overwrite if already exists unless identical
    if os.path.exists(path):
        try:
            old = json.loads(open(path, "r").read())
            if old != payload:
                print(" Manifest differs from existing one. You may be resuming with different sweep params
                print("Existing:", old)
                print("New:", payload)
        except Exception:
            print(" Could not read existing manifest. Proceeding.")
        return
    with open(path, "w") as f:
        json.dump(payload, f, indent=2)



# sweep runner

WL_FILE = "mix_50_50.jsonl"
TOTAL_REQ = 200
DESIRED_MAX_TOKENS = 16

concurrency_list = [1, 2, 4, 8, 16, 32, 48, 64]
max_num_seqs_list = [16, 32, 48, 64, 80, 96]

# save manifest to know what this checkpoint corresponds to
write_manifest({
    "workload": WL_FILE,
    "total_requests": TOTAL_REQ,
    "desired_max_tokens": DESIRED_MAX_TOKENS,
    "concurrency_list": concurrency_list,
    "max_num_seqs_list": max_num_seqs_list,
    "model_id": MODEL_ID,
    "server_flags": {
        "max_num_batched_tokens": 8192,
        "long_prefill_token_threshold": 1024,
        "gpu_memory_utilization": 0.70,
        "max_model_len": 4096,
        "dtype": "half",
        "enforce_eager": True
    }
})

ckpt_df = load_checkpoint_df(CHECKPOINT_CSV)
done = completed_pairs(ckpt_df)
print(f" Loaded checkpoint rows: {len(ckpt_df)} | completed runs: {len(done)}")

for max_num_seqs in max_num_seqs_list:
    # If all concurrencies for this max_num_seqs are done, skip server restart entirely
    all_done_for_this = all((max_num_seqs, c) in done for c in concurrency_list)
    if all_done_for_this:
        print(f"\n Skipping max_num_seqs={max_num_seqs} (all concurrencies already done)")
        continue

    print("\n====================================================")
    print(f"Starting BAD server with max-num-seqs={max_num_seqs}")
    print("====================================================")
    start_bad_server(max_num_seqs=max_num_seqs, log_path="vllm_bad.log")

    ok = wait_for_ready(BASE, log_path="vllm_bad.log", timeout_s=420)
    if not ok:
        # record server_failed for missing concurrencies only
        for c in concurrency_list:
            if (max_num_seqs, c) in done:
                continue
            row = {
                "label": "BAD_unfair",
                "workload": WL_FILE,
                "concurrency": int(c),
                "max_num_seqs": int(max_num_seqs),
                "status": "server_failed",
            }
            append_row_checkpoint(row, CHECKPOINT_CSV)
        # refresh checkpoint view
        ckpt_df = load_checkpoint_df(CHECKPOINT_CSV)
        done = completed_pairs(ckpt_df)
        continue

    warmup_request()

    for c in concurrency_list:
        if (max_num_seqs, c) in done:
            print(f"  Skip (already done): max_seqs={max_num_seqs}, c={c}")
            continue

        print(f"\n--- RUN: BAD_unfair | {WL_FILE} | max_seqs={max_num_seqs} | c={c} ---")

        try:
            df_bad, summ_bad = run_load(
                prompt_sets[WL_FILE],
                concurrency=c,
                total_requests=TOTAL_REQ,
                desired_max_tokens=DESIRED_MAX_TOKENS,
                desc=f"BAD_unfair | {WL_FILE} | max_seqs={max_num_seqs} | c={c}",
            )

            row = row_from_run(
                label="BAD_unfair",
                workload=WL_FILE,
                concurrency=c,
                max_num_seqs=max_num_seqs,
                df=df_bad,
                summ=summ_bad,
            )
            append_row_checkpoint(row, CHECKPOINT_CSV)

        except Exception as e:
            # record run_failed so you can retry later
            row = {
                "label": "BAD_unfair",
                "workload": WL_FILE,
                "concurrency": int(c),
                "max_num_seqs": int(max_num_seqs),
                "status": "run_failed",
                "error": type(e).__name__,
            }
            append_row_checkpoint(row, CHECKPOINT_CSV)
            print(" run_load failed:", type(e).__name__)
            tail_log("vllm_bad.log", n=80)

        # refresh done set so mid-run restarts still work cleanly
        ckpt_df = load_checkpoint_df(CHECKPOINT_CSV)
        done = completed_pairs(ckpt_df)

kill_server()

summary_df = load_checkpoint_df(CHECKPOINT_CSV)
print("\n BAD (concurrency x max-num-seqs) sweep (resume-capable) done.")
summary_df

```

```
✅ Loaded checkpoint rows: 0 | completed runs: 0

====================================================
Starting BAD server with max-num-seqs=16
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=1: 100%     200/200 [02:30<00:00,  1.21it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=2: 100%     200/200 [01:13<00:00,  2.78it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=4: 100%     200/200 [00:42<00:00,  4.44it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=8: 100%     200/200 [00:26<00:00,  9.00it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=16: 100%     200/200 [00:18<00:00, 10.34it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=32: 100%     200/200 [00:15<00:00, 19.97it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=48: 100%     200/200 [00:14<00:00, 19.86it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=16 | c=64: 100%     200/200 [00:13<00:00, 19.43it/s, fail=0, ok=200]

====================================================
Starting BAD server with max-num-seqs=32
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
```

```
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
[25] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=1: 100%    200/200 [02:31<00:00,  1.21it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=2: 100%    200/200 [01:12<00:00,  2.06it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=4: 100%    200/200 [00:42<00:00,  4.55it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=8: 100%    200/200 [00:27<00:00, 10.03it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=16: 100%    200/200 [00:18<00:00, 10.30it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=32: 100%    200/200 [00:12<00:00, 30.93it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=48: 100%    200/200 [00:10<00:00, 22.96it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=32 | c=64: 100%    200/200 [00:10<00:00, 30.86it/s, fail=0, ok=200]

====================================================
Starting BAD server with max-num-seqs=48
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=1: 100%    200/200 [02:31<00:00,  1.22it/s, fail=0, ok=200]
```

```
--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=2: 100%          200/200 [01:12<00:00,  2.77it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=4: 100%          200/200 [00:42<00:00,  5.14it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=8: 100%          200/200 [00:27<00:00,  8.21it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=16: 100%          200/200 [00:17<00:00,  9.75it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=32: 100%          200/200 [00:12<00:00, 21.39it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=48: 100%          200/200 [00:10<00:00, 17.80it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=48 | c=64: 100%          200/200 [00:09<00:00, 38.46it/s, fail=0, ok=200]


====================================================
Starting BAD server with max-num-seqs=64
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=1: 100%          200/200 [02:31<00:00,  1.21it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=2: 100%          200/200 [01:13<00:00,  2.77it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=4: 100%          200/200 [00:42<00:00,  4.33it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=8: 100%          200/200 [00:26<00:00,  8.99it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=16: 100%          200/200 [00:18<00:00, 12.46it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=32: 100%          200/200 [00:12<00:00, 27.29it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=48: 100%          200/200 [00:09<00:00, 41.25it/s, fail=0, ok=200]
```

```
--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=64 | c=64: 100%          200/200 [00:08<00:00, 35.42it/s, fail=0, ok=200]

====================================================
Starting BAD server with max-num-seqs=80
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
[25] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=1: 100%          200/200 [02:31<00:00,  1.21it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=2: 100%          200/200 [01:13<00:00,  2.78it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=4: 100%          200/200 [00:42<00:00,  4.63it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=8: 100%          200/200 [00:27<00:00,  8.01it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=16: 100%          200/200 [00:18<00:00,  9.05it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=32: 100%          200/200 [00:13<00:00, 17.77it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=48: 100%          200/200 [00:10<00:00, 20.46it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=80 | c=64: 100%          200/200 [00:08<00:00, 22.60it/s, fail=0, ok=200]

====================================================
Starting BAD server with max-num-seqs=96
====================================================
[1] not ready yet: ConnectionError
[2] not ready yet: ConnectionError
[3] not ready yet: ConnectionError
[4] not ready yet: ConnectionError
[5] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[6] not ready yet: ConnectionError
[7] not ready yet: ConnectionError
[8] not ready yet: ConnectionError
[9] not ready yet: ConnectionError
[10] not ready yet: ConnectionError
```

```
--- tail vllm_bad.log (last 30) ---
[11] not ready yet: ConnectionError
[12] not ready yet: ConnectionError
[13] not ready yet: ConnectionError
[14] not ready yet: ConnectionError
[15] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[16] not ready yet: ConnectionError
[17] not ready yet: ConnectionError
[18] not ready yet: ConnectionError
[19] not ready yet: ConnectionError
[20] not ready yet: ConnectionError

--- tail vllm_bad.log (last 30) ---
[21] not ready yet: ConnectionError
[22] not ready yet: ConnectionError
[23] not ready yet: ConnectionError
[24] not ready yet: ConnectionError
✅ Server ready. Models: ['Qwen/Qwen2.5-3B-Instruct']

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=1 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=1: 100%          200/200 [02:30<00:00,  1.22it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=2 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=2: 100%          200/200 [01:23<00:00,  2.05it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=4 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=4: 100%          200/200 [00:52<00:00,  2.91it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=8 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=8: 100%          200/200 [00:36<00:00,  6.51it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=16 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=16: 100%          200/200 [00:23<00:00,  7.26it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=32 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=32: 100%          200/200 [00:15<00:00, 12.58it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=48 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=48: 100%          200/200 [00:14<00:00, 32.98it/s, fail=0, ok=200]

--- RUN: BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=64 ---
BAD_unfair | mix_50_50.jsonl | max_seqs=96 | c=64: 100%          200/200 [00:11<00:00, 21.06it/s, fail=0, ok=200]

✅ BAD (concurrency x max-num-seqs) sweep (resume-capable) done.
```

	label	workload	concurrency	max_num_seqs	status	ok	fail	rps	wall_time_s	summ_p50_s	...	sh
0	BAD_unfair	mix_50_50.jsonl	1	16	ok	200	0	1.327445	150.665335	0.456500	...	
1	BAD_unfair	mix_50_50.jsonl	2	16	ok	200	0	2.735775	73.105440	0.729429	...	
2	BAD_unfair	mix_50_50.jsonl	4	16	ok	200	0	4.670095	42.825685	0.773686	...	
3	BAD_unfair	mix_50_50.jsonl	8	16	ok	200	0	7.587554	26.358956	1.061564	...	
4	BAD_unfair	mix_50_50.jsonl	16	16	ok	200	0	11.004723	18.174014	1.386675	...	
5	BAD_unfair	mix_50_50.jsonl	32	16	ok	200	0	13.177642	15.177222	2.401322	...	
6	BAD_unfair	mix_50_50.jsonl	48	16	ok	200	0	13.587360	14.719563	3.413464	...	
7	BAD_unfair	mix_50_50.jsonl	64	16	ok	200	0	13.562172	14.746900	4.487614	...	
8	BAD_unfair	mix_50_50.jsonl	1	32	ok	200	0	1.319701	151.549532	0.458507	...	
9	BAD_unfair	mix_50_50.jsonl	2	32	ok	200	0	2.747514	72.793065	0.732680	...	
10	BAD_unfair	mix_50_50.jsonl	4	32	ok	200	0	4.677001	42.762445	0.775419	...	
11	BAD_unfair	mix_50_50.jsonl	8	32	ok	200	0	7.324900	27.304127	1.088049	...	
12	BAD_unfair	mix_50_50.jsonl	16	32	ok	200	0	10.693156	18.703553	1.404330	...	
13	BAD_unfair	mix_50_50.jsonl	32	32	ok	200	0	15.372674	13.010098	2.089596	...	
14	BAD_unfair	mix_50_50.jsonl	48	32	ok	200	0	17.569705	11.383230	2.574769	...	
15	BAD_unfair	mix_50_50.jsonl	64	32	ok	200	0	15.976706	12.518225	3.737646	...	
16	BAD_unfair	mix_50_50.jsonl	1	48	ok	200	0	1.322014	151.284371	0.458063	...	
17	BAD_unfair	mix_50_50.jsonl	2	48	ok	200	0	2.743948	72.887680	0.731331	...	
18	BAD_unfair	mix_50_50.jsonl	4	48	ok	200	0	4.716243	42.406635	0.777884	...	

19	BAD_unfair	mix_50_50.jsonl	8	48	ok	200	0	7.384087	27.085270	1.056648	...
20	BAD_unfair	mix_50_50.jsonl	16	48	ok	200	0	11.097723	18.021716	1.366025	...
21	BAD_unfair	mix_50_50.jsonl	32	48	ok	200	0	15.919387	12.563298	2.105477	...
22	BAD_unfair	mix_50_50.jsonl	48	48	ok	200	0	17.584097	11.373914	2.540315	...
23	BAD_unfair	mix_50_50.jsonl	64	48	ok	200	0	18.785692	10.646400	2.921785	...
24	BAD_unfair	mix_50_50.jsonl	1	64	ok	200	0	1.319597	151.561409	0.457572	...
25	BAD_unfair	mix_50_50.jsonl	2	64	ok	200	0	2.721553	73.487456	0.733294	...
26	BAD_unfair	mix_50_50.jsonl	4	64	ok	200	0	4.658324	42.933894	0.793960	...
27	BAD_unfair	mix_50_50.jsonl	8	64	ok	200	0	7.434882	26.900224	1.056196	...
28	BAD_unfair	mix_50_50.jsonl	16	64	ok	200	0	11.067443	[illegible]	[illegible]	...
29	BAD_unfair	mix_50_50.jsonl	32	64	ok	200	0	15.559964	12.853500	1.932129	...
42	BAD_unfair	mix_50_50.jsonl	4	96	ok	200	0	[illegible]	[illegible]	[illegible]	...
43	BAD_unfair	mix_50_50.jsonl	8	96	ok	200	0	5.551586	36.025742	1.418485	...
44	BAD_unfair	mix_50_50.jsonl	16	96	ok	200	0	8.342541	23.973511	1.792479	...
45	BAD_unfair	mix_50_50.jsonl	32	96	ok	200	0	12.902239	15.501185	2.657199	...
46	BAD_unfair	mix_50_50.jsonl	48	96	ok	200	0	13.514812	14.798578	3.581036	...
47	BAD_unfair	mix_50_50.jsonl	64	96	ok	200	0	14.880155	13.440721	3.983440	...

48 rows × 28 columns

Conclusion

- **Throughput scaling:** RPS increases with client concurrency, but the gains flatten after roughly `c=32` for most `max_num_seqs` values (diminishing returns region).
- **Tail latency inflation:** As concurrency rises, **p99 grows much faster than p50**, showing queueing and head-of-line effects dominating at high load.
- **Short vs long separation:** In the mixed workload, **short requests stay much faster than long requests**, but short p99 still inflates as concurrency increases because shorts get stuck behind long-prefill work.
- **Impact of max_num_seqs:** Larger `max_num_seqs` generally allows higher throughput at mid/high concurrency, but it can also **worsen tail latency** if the server admits too many concurrent sequences and amplifies contention/queueing under this "bad" configuration.

```
OUT_DIR = "/content/drive/MyDrive/vllm_sweeps/bad"
os.makedirs(OUT_DIR, exist_ok=True)

# Copy any existing on-disk artifacts to Drive (if they exist)
for f in ["bad_grid_checkpoint.csv", "bad_grid_manifest.json", "vllm_bad.log"]:
    if os.path.exists(f):
        shutil.copy(f, f"{OUT_DIR}/{f}")
        print("✅ Copied:", f"{OUT_DIR}/{f}")
    else:
        print("⚠️ Missing locally:", f)

# save the final dataframe we have in memory (summary_df)
try:
    summary_df_path = f"{OUT_DIR}/bad_summary_df.csv"
    summary_df.to_csv(summary_df_path, index=False)
    print("✅ Saved summary_df:", summary_df_path)
except NameError:
    print("⚠️ summary_df not found in memory. If your df has a different name, save that instead.")

```

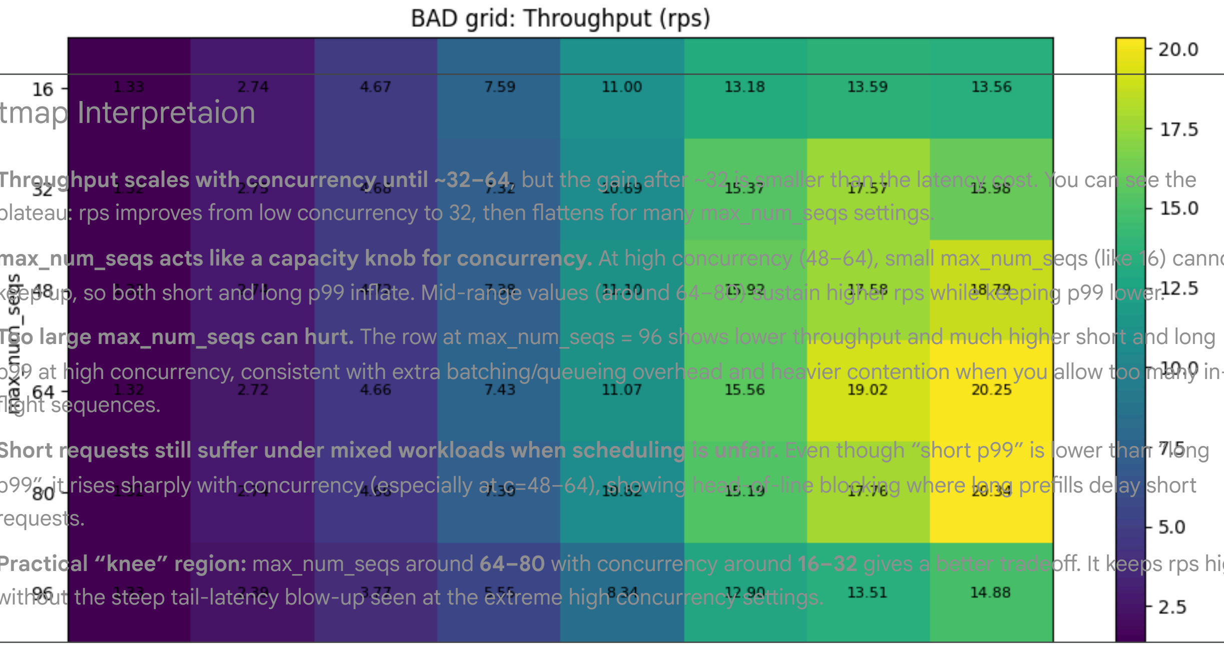
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force
✅ Copied: /content/drive/MyDrive/vllm_sweeps/bad/bad_grid_checkpoint.csv
✅ Copied: /content/drive/MyDrive/vllm_sweeps/bad/bad_grid_manifest.json
✅ Copied: /content/drive/MyDrive/vllm_sweeps/bad/vllm_bad.log
✅ Saved summary_df: /content/drive/MyDrive/vllm_sweeps/bad/bad_summary_df.csv
```

Plots for Bidirectional Sweep in Bad Config

```
df = summary_df.copy()

# Keep only good rows
df = df[df["status"] == "ok"].copy()

# Ensure numeric
for col in ["concurrency", "max_num_seqs", "rps", "short_p99", "long_p99", "overall_p99", "wall_time_s"]:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")

# Sort for consistent plotting
df = df.sort_values(["max_num_seqs", "concurrency"]).reset_index(drop=True)

concurrency_list = sorted(df["concurrency"].unique().tolist())
max_num_seqs_list = sorted(df["max_num_seqs"].unique().tolist())

print("Concurrencies:", concurrency_list)
print("max_num_seqs:", max_num_seqs_list)
print("Rows:", len(df))

```

```
Concurrencies: [1, 2, 4, 8, 16, 32, 48, 64]
max_num_seqs: [16, 32, 48, 64, 80, 96]
Rows: 48
```

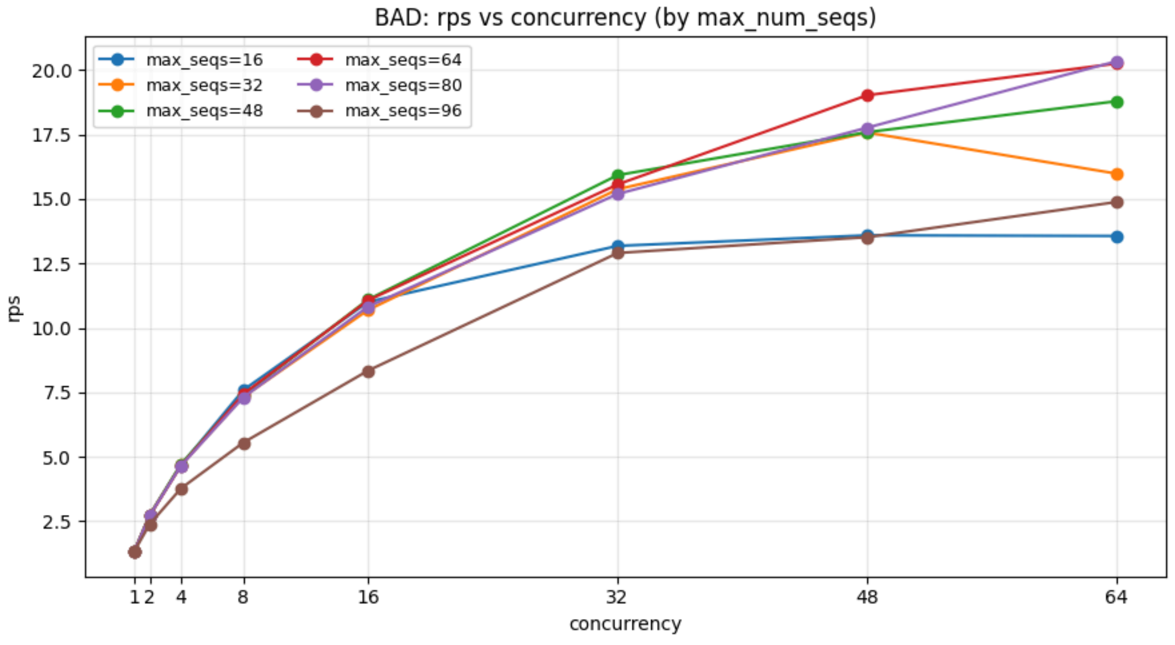
```
def plot_heatmap(pivot_df, title, xlabel="concurrency", ylabel="max_num_seqs", fmt="{:.2f}"):
    """
    pivot_df: index=max_num_seqs, columns=concurrency, values=metric
    """
    data = pivot_df.values.astype(float)

    plt.figure(figsize=(10, 5.5))
    im = plt.imshow(data, aspect="auto", interpolation="nearest")
    plt.colorbar(im)

    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    plt.xticks(ticks=np.arange(pivot_df.shape[1]), labels=pivot_df.columns.tolist())
    plt.yticks(ticks=np.arange(pivot_df.shape[0]), labels=pivot_df.index.tolist())


    for i in range(pivot_df.shape[0]):
        for j in range(pivot_df.shape[1]):
            val = data[i, j]
            if np.isfinite(val):
                plt.text(j, i, fmt.format(val), ha="center", va="center", fontsize=8)

    plt.tight_layout()
    plt.show()

# Heatmap: rps
p_rps = df.pivot_table(index="max_num_seqs", columns="concurrency", values="rps", aggfunc="mean")
plot_heatmap(p_rps, "BAD grid: Throughput (rps)")

# Heatmap: short_p99
p_sp99 = df.pivot_table(index="max_num_seqs", columns="concurrency", values="short_p99", aggfunc="mean")
plot_heatmap(p_sp99, "BAD grid: short p99 (s)")

# Heatmap: long_p99
p_lp99 = df.pivot_table(index="max_num_seqs", columns="concurrency", values="long_p99", aggfunc="mean")
plot_heatmap(p_lp99, "BAD grid: long p99 (s)")

```

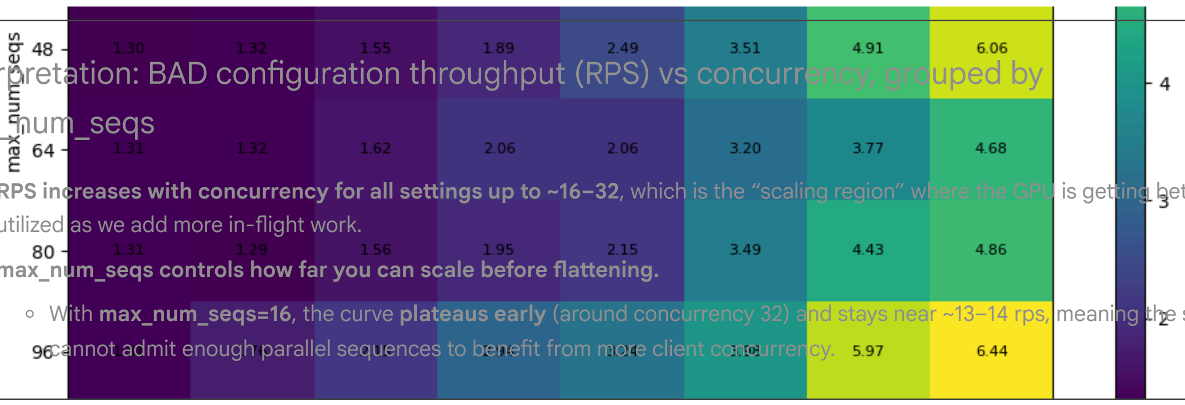


Heatmap Interpretaion

- **Throughput scales with concurrency until ~32–64**, but the gain after ~32 is smaller than the latency cost. You can see the plateau: rps improves from low concurrency to 32, then flattens for many max_num_seqs settings.
- **max_num_seqs acts like a capacity knob for concurrency.** At high concurrency (48–64), small max_num_seqs (like 16) cannot keep up, so both short and long p99 inflate. Mid-range values (around 64–80) sustain higher rps while keeping p99 lower.
- **Too large max_num_seqs can hurt.** The row at max_num_seqs = 96 shows lower throughput and much higher short and long p99 at high concurrency, consistent with extra batching/queueing overhead and heavier contention when you allow too many in-flight sequences.
- **Short requests still suffer under mixed workloads when scheduling is unfair.** Even though “short p99” is lower than “long p99”, it rises sharply with concurrency (especially at c=48–64), showing head-of-line blocking where long prefills delay short requests.
- **Practical “knee” region:** max_num_seqs around **64–80** with concurrency around **16–32** gives a better tradeoff. It keeps rps high without the steep tail-latency blow-up seen at the extreme high-concurrency settings.

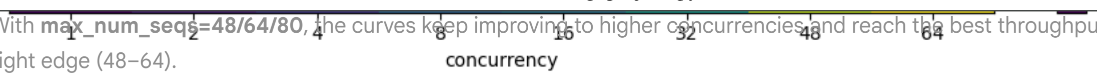
```python
plt.figure(figsize=(9, 5))
for m in max_num_seqs_list:
    sub = df[df["max_num_seqs"] == m].sort_values("concurrency")
    plt.plot(sub["concurrency"], sub["rps"], marker="o", label=f"max_seqs={m}")

plt.title("BAD: rps vs concurrency (by max_num_seqs)")
plt.xlabel("concurrency")
plt.ylabel("rps")
plt.xticks(concurrency_list)
plt.grid(True, alpha=0.3)
plt.legend(ncol=2, fontsize=9)
plt.tight_layout()
plt.show()

```





Interpretation: BAD configuration throughput (RPS) vs concurrency, grouped by max_num_seqs

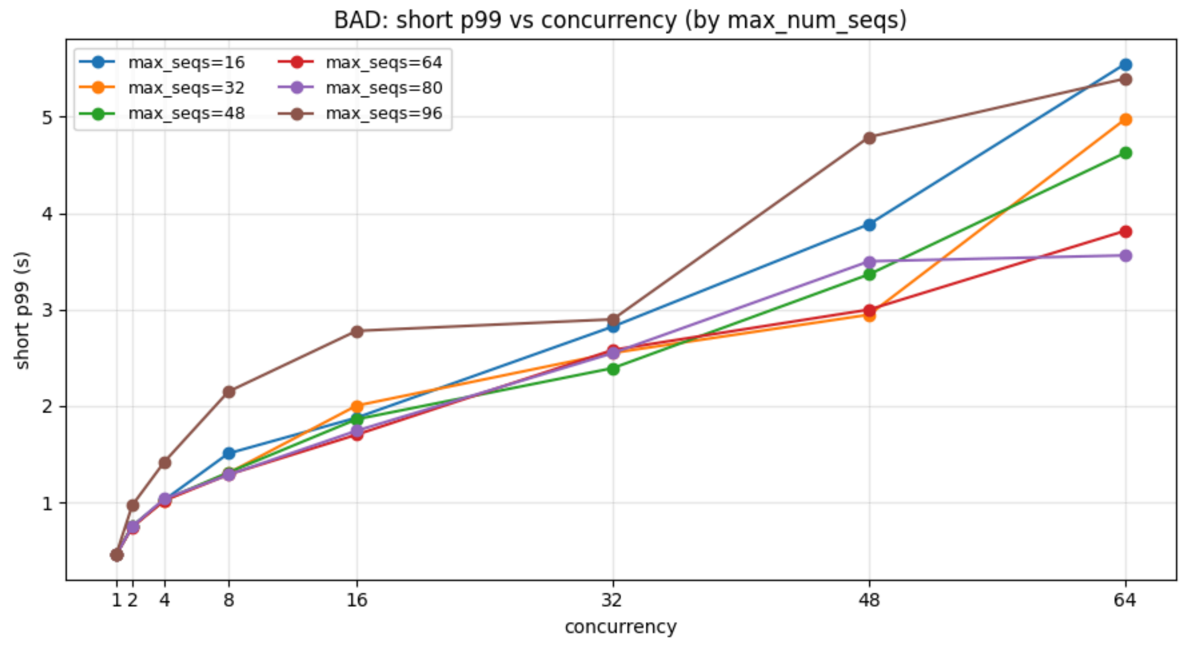
- **RPS increases with concurrency for all settings up to ~16–32**, which is the “scaling region” where the GPU is getting better utilized as we add more in-flight work.
- **max_num_seqs controls how far you can scale before flattening.**
  - With **max_num_seqs=16**, the curve **plateaus early** (around concurrency 32) and stays near ~13–14 rps, meaning the server cannot admit enough parallel sequences to benefit from more client concurrency.

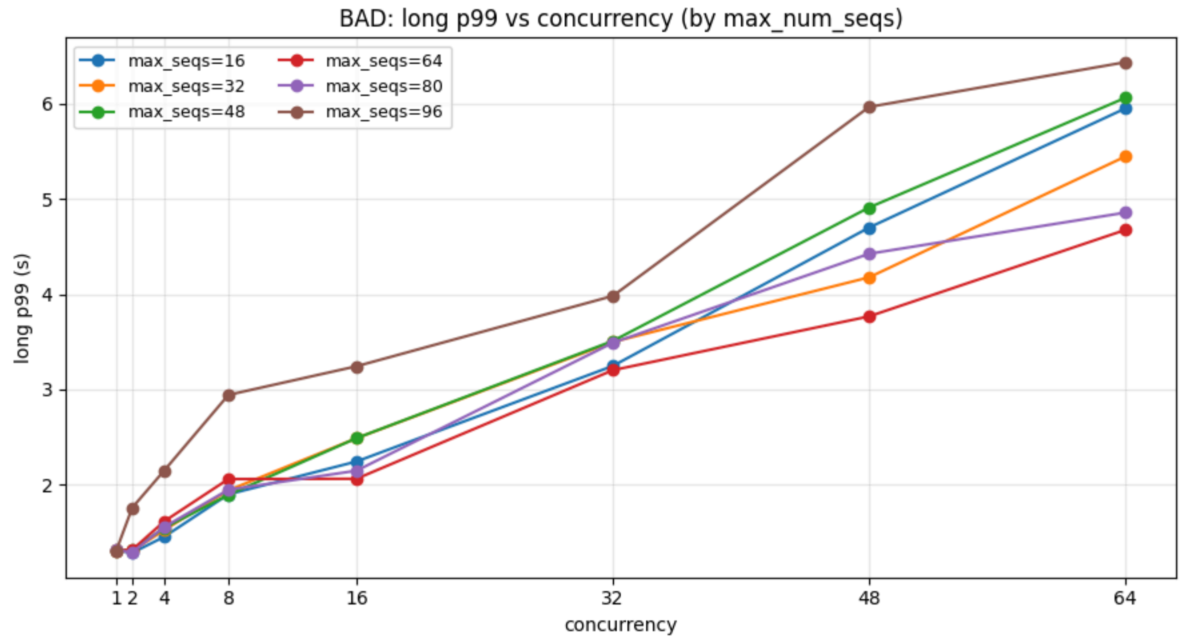


- With **max_num_seqs=48/64/80**, the curves keep improving to higher concurrencies and reach the best throughput at the right edge (48–64).

- **The best throughput in this sweep occurs at high concurrency (48–64) with max_num_seqs around 64–80** (top curves near ~20 rps). That suggests this range is closer to the capacity point of this setup.
- **max_num_seqs=96 underperforms at low and mid concurrency and only partially catches up later.** This is consistent with “too many allowed in-flight sequences” adding overhead (more queueing, scheduling contention, memory pressure), so the system wastes efficiency instead of converting concurrency into throughput.
- **Diminishing returns are visible beyond ~48 concurrency**, even for the best max_num_seqs values. The slope flattens, meaning you are approaching saturation, and extra concurrency mostly increases queueing rather than throughput.

```python
# short p99
plt.figure(figsize=(9, 5))
for m in max_num_seqs_list:
    sub = df[df["max_num_seqs"] == m].sort_values("concurrency")
    plt.plot(sub["concurrency"], sub["short_p99"], marker="o", label=f"max_seqs={m}")

plt.title("BAD: short p99 vs concurrency (by max_num_seqs)")
plt.xlabel("concurrency")
plt.ylabel("short p99 (s)")
plt.xticks(concurrency_list)
plt.grid(True, alpha=0.3)
plt.legend(ncol=2, fontsize=9)
plt.tight_layout()
plt.show()

# long p99
plt.figure(figsize=(9, 5))
for m in max_num_seqs_list:
    sub = df[df["max_num_seqs"] == m].sort_values("concurrency")
    plt.plot(sub["concurrency"], sub["long_p99"], marker="o", label=f"max_seqs={m}")

plt.title("BAD: long p99 vs concurrency (by max_num_seqs)")
plt.xlabel("concurrency")
plt.ylabel("long p99 (s)")
plt.xticks(concurrency_list)
plt.grid(True, alpha=0.3)
plt.legend(ncol=2, fontsize=9)
plt.tight_layout()
plt.show()

```





Interpretation: BAD short p99 vs concurrency and BAD long p99 vs concurrency

- For both short and long requests, **p99 increases monotonically with concurrency**. This is classic queueing behavior: higher offered load increases worst-case wait time.
- **Short p99 inflates sharply at high concurrency** (c=48, c=64), showing that short requests lose their "fast path" when mixed with long prompts under unfair scheduling.
- **Long p99 also climbs steadily**, and it typically climbs faster than short p99 because long prompts contribute more prefill work and hold resources longer.
- **max_num_seqs=64 (and sometimes 80) provides a better tradeoff** than 16/32 (too restrictive) and 96 (too aggressive): higher throughput without blowing up p99 as badly as the overcommitted setting.

Expermient 5: Creating Fair Scheduling to improve the latency and RPS (or the trade-off)

Fair sanity check: lower max-num-batched-tokens (btok=768) at c=32

Before running the full FAIR sweeps, I ran a single controlled trial by reducing `max-num-batched-tokens` from the BAD baseline (8192) down to **768**, while keeping everything else identical and evaluating at the **same client concurrency (c=32)** used for the BAD runs.

- **Purpose:** verify that `max-num-batched-tokens` is a meaningful lever under mixed 50/50 traffic, and that lowering it can reduce the “long-prefill domination” effect that inflates tail latency.
- **Why 768:** it’s a mid-range value expected to encourage more frequent scheduling turns (smaller batches), which can improve responsiveness for short requests in mixed workloads.
- **Why c=32:** match the BAD configuration’s stress level so the comparison is apples-to-apples and any improvement is not just because the system is under-loaded.

```
!pkill -f "vllm.entrypoints.openai.api_server" || true
!pkill -f "VLLM::EngineCore" || true
!rm -f vllm_fair.log
!sleep 1

```

```

!nohup python3 -m vllm.entrypoints.openai.api_server \
  --model Qwen/Qwen2.5-3B-Instruct \
  --host 127.0.0.1 --port 8000 \
  --dtype half \
  --max-model-len 4096 \
  --gpu-memory-utilization 0.70 \
  --disable-log-stats \
  --enforce-eager \
  --max-num-batched-tokens 768 \
  --long-prefill-token-threshold 1024 \
  > vllm_fair.log 2>&1 </dev/null &
```

```
^C
^C
```

```
!tail -n 40 vllm_fair.log

```

Show hidden output

```
df_fair_50_50, summ_fair_50_50 = run_load(
    prompt_sets["mix_50_50.jsonl"],
    concurrency=32,
    total_requests=200,
    desired_max_tokens=16,
    desc="FAIR_protected | mix_50_50 | c=16"
)

print("FAIR summary:", summ_fair_50_50)
df_fair_50_50.head()

```

```
FAIR_protected | mix_50_50 | c=16: 100%                                   200/200 [00:41<00:00, 11.12it/s, fail=0, ok=200]
FAIR summary: {'concurrency': 32, 'total_requests': 200, 'ok': 200, 'fail': 0, 'wall_time_s': 41.268839522000235,
```

	latency_s	prompt_tokens	completion_tokens	total_tokens	status_code	prompt_tokens_est	max_tokens_used	trim
0	3.036116	37	16	53	200	37	16	
1	3.129861	37	16	53	200	37	16	
2	3.074753	45	16	61	200	45	16	
3	3.092510	37	16	53	200	37	16	
4	3.141762	38	16	54	200	38	16	

```

def latency_stats(df, label, mask, subset_name):
    x = df.loc[mask & (df["ok"] == True), "latency_s"].dropna().values
    if len(x) == 0:
        return {"label": label, "subset": subset_name, "n": 0, "p50": None, "p90": None, "p99": None, "mean"
    return {
        "label": label,
        "subset": subset_name,
        "n": int(len(x)),
        "p50": float(np.quantile(x, 0.50)),
        "p90": float(np.quantile(x, 0.90)),
        "p99": float(np.quantile(x, 0.99)),
        "mean": float(np.mean(x)),
    }

def compare_short_long(df, label, short_cut=64, long_cut=2048):
    return [
        latency_stats(df, label, mask=df["prompt_tokens"] >= 0, subset_name="overall"),
        latency_stats(df, label, mask=df["prompt_tokens"] <= short_cut, subset_name=f"short<= {short_cut}"),
        latency_stats(df, label, mask=df["prompt_tokens"] >= long_cut, subset_name=f"long>= {long_cut}"),
    ]

bad_stats  = compare_short_long(df_bad,  "BAD_unfair", short_cut=64, long_cut=2048)
fair_stats = compare_short_long(df_fair_50_50, "FAIR_768", short_cut=64, long_cut=2048)

bad_stats, fair_stats

```

```
([{'label': 'BAD_unfair',
   'subset': 'overall',
   'n': 200,
   'p50': 5.534205468999517,
   'p90': 10.461404818999926,
   'p99': 11.045583048669595,
   'mean': 6.076663721505029},
  {'label': 'BAD_unfair',
   'subset': 'short<= 64',
   'n': 108,
   'p50': 4.6954104774999905,
   'p90': 8.427761869799998,
   'p99': 10.15342957642024,
   'mean': 5.170086654425977},
  {'label': 'BAD_unfair',
   'subset': 'long>= 2048',
   'n': 92,
   'p50': 7.038852010499795,
   'p90': 10.96015420079957,
   'p99': 11.058848744559482,
   'mean': 7.140906365467394}],
 [{'label': 'FAIR_768',
   'subset': 'overall',
   'n': 200,
   'p50': 6.04282219300012,
   'p90': 9.18175588419981,
   'p99': 11.05694531869025,
   'mean': 6.266493499729964},
  {'label': 'FAIR_768',
   'subset': 'short<= 64',
   'n': 108,
   'p50': 5.629765399999542,
   'p90': 8.433177871999579,
   'p99': 10.081671103819572,
   'mean': 5.578912106185119},
  {'label': 'FAIR_768',
   'subset': 'long>= 2048',
   'n': 92,
   'p50': 7.1285672229996635,
   'p90': 9.995110139300506,
   'p99': 11.105451201459774,
   'mean': 7.073654266065215}])
```

Conclusion: btok = 768 and c = 32

- **Overall:** FAIR_768 slightly **hurts median** (p50: 6.04s vs 5.53s) but **improves p90** (9.18s vs 10.46s). **p99 is basically unchanged** (~11.06s in both).
- **Short requests (≤64 tokens):** FAIR_768 **slows p50** (5.63s vs 4.70s) but gives a **small p99 improvement** (10.08s vs 10.15s). Net: better tail for short, worse typical latency.
- **Long requests (≥2048 tokens):** FAIR_768 **improves p90** (10.00s vs 10.96s) while **p99 stays ~the same** (11.11s vs 11.06s). This suggests btok=768 reduces "most long requests" queueing, but the worst-case is still dominated by contention.
- **Implication:** Lowering `max-num-batched-tokens` to 768 shifts the distribution (especially **p90**) but doesn't meaningfully reduce the **worst-case p99** at this concurrency—so a sweep is needed to find the btok "sweet spot" that improves tails without sacrificing too much throughput.

`max-num-batched-token` Sweep @ c = 16

```
BASE = "http://127.0.0.1:8000"
MODEL_ID = "Qwen/Qwen2.5-3B-Instruct"

def kill_server():
    os.system('pkill -f "vllm.entrypoints.openai.api_server" || true')
    os.system('pkill -f "VLLM::EngineCore" || true')
    time.sleep(1.5)

def start_fair_server(max_btok, log_path="vllm_fair.log"):
    # clean + restart
    kill_server()
    try:
        os.remove(log_path)
    except FileNotFoundError:
        pass

    cmd = f"""
nohup python3 -m vllm.entrypoints.openai.api_server \
  --model {MODEL_ID} \
  --host 127.0.0.1 --port 8000 \
  --dtype half \
  --max-model-len 4096 \
  --gpu-memory-utilization 0.70 \
  --disable-log-stats \
  --enforce-eager \
  --max-num-batched-tokens {int(max_btok)} \
  --long-prefill-token-threshold 1024 \
  > {log_path} 2>&1 </dev/null &
"""
    os.system(cmd)
    time.sleep(1.0)

def tail_log(log_path, n=60):
    if os.path.exists(log_path):
        print(f"\n--- tail {log_path} (last {n}) ---")
        os.system(f"tail -n {n} {log_path}")
    else:
        print(f"(log missing: {log_path})")

def wait_for_ready(base=BASE, log_path="vllm_fair.log", timeout_s=300):
    t0 = time.time()
    tries = 0
    while True:
        tries += 1
        try:
            r = requests.get(f"{base}/v1/models", timeout=2)
            if r.status_code == 200:
                data = r.json().get("data", [])
                ids = [m.get("id") for m in data]
                print(f"✅ Server ready. Models: {ids}")
                return True
            else:
                print(f"[{tries}] /v1/models status={r.status_code}")
        except Exception as e:
            print(f"[{tries}] not ready yet: {type(e).__name__}")

        # if process died, fail fast
        alive = os.system('ps -ef | grep -E "vllm.entrypoints.openai.api_server" | grep -v grep > /dev/null
        if not alive:
            print("❌ vLLM process is NOT running anymore.")
            tail_log(log_path, n=120)
            return False

        if time.time() - t0 > timeout_s:
            print("❌ Timed out waiting for server.")
            tail_log(log_path, n=120)
            return False

        if tries % 5 == 0:
            tail_log(log_path, n=30)

        time.sleep(2)

def warmup_request():
    # forces one small request to populate caches
    payload = {
        "model": MODEL_ID,
        "messages": [{"role": "user", "content": "hi"}],
        "max_tokens": 8,
        "temperature": 0.0
    }
    try:
        _ = requests.post(f"{BASE}/v1/chat/completions", json=payload, timeout=30)
    except Exception:
        pass



#  Sweep runner

WL_FILE = "mix_50_50.jsonl"
CONCURRENCY = 16
TOTAL_REQ = 400
```

```
DESIRED_MAX_TOKENS = 16

btok_list = [256, 384, 512, 768, 1024, 1536, 2048]  # edit as you like
rows = []
fair_runs = {}

for btok in btok_list:
    print(f"\n==============================")
    print(f"Starting FAIR server: max-num-batched-tokens={btok}")
    print(f"==============================")

    start_fair_server(btok, log_path="vllm_fair.log")

    ok = wait_for_ready(BASE, log_path="vllm_fair.log", timeout_s=420)
    if not ok:
        rows.append({
            "label": "FAIR",
            "workload": WL_FILE,
            "concurrency": CONCURRENCY,
            "max_num_batched_tokens": btok,
            "status": "server_failed"
        })
        continue

    warmup_request()

    df_fair, summ_fair = run_load(
        prompt_sets[WL_FILE],
        concurrency=CONCURRENCY,
        total_requests=TOTAL_REQ,
        desired_max_tokens=DESIRED_MAX_TOKENS,
        desc=f"FAIR | {WL_FILE} | c={CONCURRENCY} | btok={btok}"
    )

    fair_runs[btok] = (df_fair, summ_fair)

    m = subset_metrics(df_fair)

    rows.append({
        "label": "FAIR",
        "workload": WL_FILE,
        "concurrency": CONCURRENCY,
        "max_num_batched_tokens": btok,
        "status": "ok",
        "ok_count": int((df_fair["ok"] == True).sum()),
        "fail_count": int((df_fair["ok"] != True).sum()),
        "p50_s": float(df_fair[df_fair["ok"]==True]["latency_s"].quantile(0.50)),
        "p90_s": float(df_fair[df_fair["ok"]==True]["latency_s"].quantile(0.90)),
        "p99_s": float(df_fair[df_fair["ok"]==True]["latency_s"].quantile(0.99)),
        "mean_s": float(df_fair[df_fair["ok"]==True]["latency_s"].mean()),
        **m
    })


kill_server()

fair_summary_df = pd.DataFrame(rows)
print("\n✅ FAIR btok sweep done.")
fair_summary_df

```