A Low-Latency Mixture-of-Experts Orchestrator for Domain-Specific Mental Health LLMs

Anonymous ACL submission

Abstract

We present a practical Mixture-of-Experts (MoE) system for domain-specialized mental-health dialogue on resource-constrained hardware. Our architecture combines (i) five small expert language models, each fine-tuned on a distinct mental-health disorder, with (ii) a lightweight orchestrator that routes user queries to the most appropriate expert. Because the provided data consisted of full textbooks rather than ready-made question–answer (QA) pairs, we first construct a high-quality synthetic supervision set by prompting a large external model (Gemini-2.5-Flash-Lite) to generate QA pairs from book chunks. Each expert is then trained as a parameter-efficient LoRA adapter on top of Llama-3.2-1B-Instruct. We additionally design a TF–IDF + Logistic Regression router trained first on instruction texts and later on raw book chunks, achieving a macro-F1 of 0.83 on millions of labeled spans. To evaluate end-to-end latency under realistic multi-client load, we replay Azure-server traces against a custom inference server that supports batching, LRU-based model management, PEFT adapter switching, and quantized GGUF backends via `llama.cpp`. Our experiments show that (1) domain experts provide more clinically grounded answers than the base model despite slightly worse cross-entropy, and (2) replacing HF CPU inference with 8-bit GGUF models reduces end-to-end latencies by roughly 2–3× while keeping routing accuracy high.

1 Introduction

Mixture-of-Experts (MoE) architectures offer a principled way to scale language models by activating only a small subset of parameters per input (**?**). In practice, however, most MoE systems assume GPU-rich environments and do not directly address low-latency inference on commodity CPUs.

This work targets a concrete and constrained setting: a mental-health assistant that must route user queries to one of several small, domain-specific experts (e.g., anxiety, depression, bipolar disorder) and run end-to-end on CPU with tight latency constraints. Our goal is not to push SOTA benchmarks, but to design a *full pipeline*—from data creation to training, routing, and deployment—that is technically sound, latency-aware, and easily reproducible.

The assignment constraints created several challenges: (i) training data was provided as five long PDF textbooks (one per disorder), not as curated question–answer pairs; (ii) the base model was fixed to Llama-3.2-1B-Instruct; and (iii) the system needed to be evaluated under realistic, asynchronous client load emulating Azure traces.

We address these by: (1) generating high-quality QA pairs from each textbook via Gemini-2.5-Flash-Lite; (2) training LoRA experts on the synthetic instruction–answer pairs; (3) designing a text-based orchestrator trained first on instructions and later on raw book chunks; and (4) implementing a custom inference server with model caching, batching, and quantized GGUF backends using `llama.cpp`.

Contributions.

- A data-generation pipeline that converts domain textbooks into thousands of clinically-focused QA pairs using a strong external LLM.
- Five domain experts trained as LoRA adapters on Llama-3.2-1B-Instruct, with qualitative analysis of specialization vs. overfitting.
- A TF–IDF + Logistic Regression orchestrator trained on millions of book-derived spans, achieving 0.83 macro-F1 across five disorders.
- A realistic evaluation harness using Azure-style traces, including an inference server with batching, LRU model management, adapter switching, and GGUF-based quantized experts.

2 System Overview

Our system is a sparse MoE architecture with explicit routing. Figure 1 illustrates the main components.

At a high level, the pipeline is:

1. **Data construction:** Each disorder-specific PDF is chunked into 800–1000 word spans. Using Gemini-2.5-Flash-Lite, we generate multiple QA pairs per span, yielding ∼750 QA pairs per disorder.

2. **Expert training:** For each of the five domains (anxiety, bipolar, depression, OCD, schizophrenia), we fine-tune a LoRA adapter on top of Llama-3.2-1B-Instruct using next-token prediction over instruction + output text.

3. **Orchestrator training:** We initially train a TF–IDF + Logistic Regression classifier on the synthetic instruction texts. Observing some qualitative mismatch, we retrain the orchestrator directly on book chunks (sub-spans), yielding ∼ 19.9M labeled training examples across domains.

4. **Deployment and evaluation:** We implement a multi-client setup replaying Azure-like traces. A central server hosts the orchestrator and a pool of experts, manages LRU-based model loading, batching, and inference, and logs end-to-end latencies. We support both HF-based LoRA experts and quantized 8-bit GGUF experts via `llama.cpp`.

3 Data Construction

3.1 Source Textbooks

We received five textbooks in PDF form, one for each disorder: anxiety, bipolar disorder, depression, obsessive–compulsive disorder (OCD), and schizophrenia. These are long-form clinical and therapeutic texts targeted at practitioners and advanced students, not directly usable as supervised dialogue data.

3.2 Chunking and Prompting

Each PDF is first read via ‘pymupdf‘, and then is split into contiguous chunks of 800–1000 words using simple heuristics based on paragraph boundaries. For each chunk, we call Gemini-2.5-Flash-Lite with the following prompt template:

```
You are an expert in {domain_label}.
Read the text below and generate 5
unique Question-Answer pairs. Focus
on clinical/therapeutic advice found
in the text. Format strictly as a
JSON list: [{ "instruction": "User
Question", "output": "Expert Answer"
}]\n\nTEXT:{text_chunk}
```

We found this task to be well within the capabilities of a small, fast model like Gemini-2.5-Flash-Lite; the generated QA pairs were generally well-formed, clinically sensible, and closely tied to the underlying text.

After filtering malformed JSON and very short answers, we obtained roughly 750 QA pairs per disorder, for a total of ∼3,750 high-quality instruction–answer examples.

3.3 Text Cleaning and Tokenization

Before training, we apply light preprocessing and concatenate `instruction` and `output` with a simple chat-like format for language model training.

We use the tokenizer associated with Llama-3.2-1B-Instruct and cap sequence length at 512 tokens, truncating long outputs as needed.

4 Expert Model Training

4.1 Base Model and Parameter-Efficient Fine-Tuning

All experts share a common base model: `meta-llama/Llama-3.2-1B-Instruct`. To keep memory and disk footprint small, we train domain-specific *LoRA* adapters on top of the base model, rather than fully fine-tuning all parameters.

We use PEFT-style LoRA with rank $r = 16$, $\alpha = 32$, and dropout 0.05, targeting attention and MLP projection layers: `{q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj}`.

4.2 Training Objective and Setup

We use a simple next-token prediction objective (causal LM loss) over the concatenated instruction + output texts, without any special RL or preference optimization. Hyperparameters are chosen conservatively:

- sequence length: 512 tokens;
- batch size: 8 per device (with gradient accumulation where needed);
- learning rate: $2 \cdot 10^{-4}$ for LoRA parameters;
- epochs: 1, 3, and 5 (for ablations);

- optimizer: AdamW with weight decay 0.01.

We train on a single A100 40GB GPU. Each domain-specific expert trains in under a minute for a few epochs, thanks to the small model size and LoRA parameterization.

4.3 Base vs. Expert Validation Loss

Somewhat counterintuitively, we observe that the base model sometimes achieves slightly lower validation cross-entropy on our small domain datasets than the LoRA experts. This is expected:

- the base model is trained on massive, diverse corpora and is a strong general language model;
- our domain-specific datasets are small ($\sim$750 examples per disorder) and narrow, so additional finetuning can degrade global perplexity;
- our validation loss is computed over both user and assistant tokens, which may penalize domain specialization.

However, qualitative inspection shows that the experts produce *more domain-specific, clinically relevant* responses. For example, given *"What are common physical sensations someone with anxiety might experience?"*, we observe:

- **1 epoch:** answers remain relatively generic, describing "feeling drained", "overwhelmed", and generic stress responses.
- **3 epochs:** responses reference domain-specific constructs such as interoceptive cues, differentiation from cardiovascular disease, and cognitive–behavioural strategies (e.g., CBT, relaxation training).
- **5 epochs:** answers become more textbook-like, summarising research findings and experimental paradigms (e.g., voluntary hyperventilation studies) in a way that is less conversational.

Based on this, we treat perplexity as a secondary metric and consider 3 epochs to be a "sweet spot" that balances specialization, readability, and overfitting. Figure 2 (placeholder) will visualize the trade-off.

5 Orchestrator (Router) Design

5.1 Initial Instruction-Level Router

We first trained an orchestrator using only the `instruction` fields from our synthetic QA dataset. We model routing as a 5-way text classification problem over disorders, using a lightweight scikit-learn pipeline:

- **Features:** TF–IDF vectors with unigrams and bigrams, `max_features = 20000`, `min_df = 2`, English stopwords.
- **Classifier:** Logistic Regression with L2 penalty, `max_iter = 1000`, and class-balanced weights.

On a stratified 70/15/15 train/validation/test split (over $\sim$750 instructions per class), we compared Logistic Regression and Multinomial Naive Bayes. Logistic Regression outperformed Naive Bayes:

- val accuracy: 89.6%;
- macro-F1: 0.897 vs. 0.861 for Naive Bayes.

Per-class F1 scores for Logistic Regression were:

- anxiety: 0.93,
- bipolar: 0.85,
- depression: 0.81,
- OCD: 0.91,
- schizophrenia: 0.97.

Most misclassifications occurred between bipolar and depression, and between anxiety and OCD, which is clinically plausible given overlapping symptom vocabulary (mood swings, rumination, worry, compulsive checking). Schizophrenia queries were almost perfectly separated due to distinctive language about hallucinations, delusions, and thought disorder.

We then retrained Logistic Regression on the combined train+val set and serialized the full pipeline (TF–IDF + classifier) via `joblib` for use in the inference server.

Class	P	R	F1
anxiety	0.91	0.79	0.85
bipolar	0.89	0.76	0.82
depression	0.87	0.88	0.87
OCD	0.67	0.91	0.77
schizophrenia	0.91	0.76	0.83
overall acc		0.82	
macro-F1		0.83	

Table 1: Router performance on book-chunk test set (3.99M examples). Training on raw textbook spans improves qualitative routing on free-form queries, at the cost of slightly lower but still strong macro-F1.

5.2 Book-Chunk Router for Better Generalization

Although the instruction-level router performed well on held-out instructions, we observed that it sometimes misrouted *free-form user queries* that resembled the textbook text more than our synthetic questions. To address this, we re-trained the orchestrator directly on book chunks.

For each PDF, we:

- chunked the text into spans (as in QA generation);
- further subdivided each chunk into smaller spans (e.g., sentences or overlapping windows);
- labeled each span with its disorder domain (anxiety, bipolar, depression, OCD, schizophrenia).

We sampled 50 base chunks per book for Gemini QA, but we used many more spans for router training. In total, we constructed 19,976,983 labeled training examples across the five domains, and held out 3,995,397 examples for testing:

Split: 15,981,586 train / 3,995,397 test.

We trained the same TF–IDF + Logistic Regression pipeline on this much larger dataset. Table 1 reports test performance.

Compared to instruction-only training, overall accuracy and macro-F1 are slightly lower (0.82 vs. 0.90), but this is expected: book spans contain more ambiguous, mixed-topic content than synthetic questions. In qualitative testing on user-style prompts, the book-chunk router produced more robust and intuitive route decisions, so we use it as our final orchestrator.

6 Inference Server and Load Generation

6.1 Azure-Inspired Client Traces

To evaluate latency under realistic multi-client load, we follow Azure-server traces commonly used for LLM service benchmarking. We take a week-long trace of function invocation times and restrict to the first 100 seconds for convenience.

Each trace entry includes:

- a function ID (used to map to an expert domain in round-robin fashion),
- a relative start time,
- a unique request ID.

We implement a trace-driven client that:

1. sleeps until the relative start time,
2. issues an asynchronous HTTP request to `/infer` on the server,
3. records send timestamp and end-to-end latency.

Prompts for these experiments are sampled from our QA instructions (and additional free-form queries), with ground-truth labels taken from the underlying dataset for routing evaluation.

6.2 Server Architecture

The server exposes a single `/infer` endpoint. Upon receiving a request, it:

1. runs the orchestrator to obtain a domain label and expert identifier;
2. checks whether the corresponding expert model is already loaded;
3. if loaded, it enqueues the request into that expert's queue;
4. if not loaded:
    - if the number of loaded models is below a configured maximum, it loads the expert;
    - otherwise, it identifies the least-recently-used (LRU) expert, evicts it, and loads the new expert.
5. after loading, the server waits up to 100 ms for additional requests targeting the same expert to arrive, forming a batch;

6. it performs batched inference on the CPU backend;

7. finally, it sends responses back to clients and logs timing information.

We run separate logging threads on both server and clients to collect trace-aligned metrics: selected experts, queue times, load times, inference times, and end-to-end latencies.

6.3 Optimizations

We implemented several optimization passes:

Naive version. Each expert is a full HF model (base + LoRA weights) loaded and unloaded independently. Loading a 1B model on CPU can take several seconds, and repeated loads dominate end-to-end latency.

Base Optimisations. Due to memory limits of our device, we needed to create a strict check for the number of models loaded at a time in the memory to not cause BSOD errors in Windows, along with LRU eviction policy. Apart from this, we also implemented batching support in the server.

Adapter-switch version. Instead of unloading the entire model, we keep the *base* model resident in memory and only switch PEFT adapters when changing domains. This greatly reduces load time, since adapter weights are small. We further extend this to support multiple bases (e.g., one base for some disorders and another for others), each with its own set of adapters.

GGUF + `llama.cpp` version. For maximum CPU efficiency, we:

1. merge each LoRA adapter into the base Llama-3.2-1B-Instruct weights to obtain a full FP16 expert;

2. convert the merged HF model to GGUF using `convert_hf_to_gguf.py`;

3. quantize to 8-bit (`q8_0`) GGUF format;

4. serve these experts via `llama-cpp-python`, which provides a fast CPU backend with batched inference.

This pipeline preserves the semantics of the trained experts while reducing memory footprint and improving inference throughput.

7 Device Specifications

- Memory - 16.0 GB; Speed: 7467 MT/s
- CPU - Intel(R) Core(TM) Ultra 7 155H; Base speed: 3.80 GHz;
- Cores: 16; Logical processors: 22;
- L1 cache: 1.6 MB; L2 cache: 18.0 MB; L3 cache: 24.0 MB

8 Experimental Results

8.1 End-to-End Latency

We evaluate three server variants under the same Azure-style client trace (replaying timestamps and function IDs, mapping functions to experts in round-robin fashion):

1. **Baseline HF server:** a straightforward HF deployment that loads and unloads full FP16 experts on demand.

2. **Dual-LM server (adapter-switch):** keeps one or more base models resident and switches LoRA adapters instead of unloading whole models.

3. **GGUF server:** uses 8-bit GGUF experts via `llama-cpp-python`, with quantized experts loaded on demand.

Each client issues asynchronous `POST /infer` requests after sleeping to match the original trace timestamps. The server logs for every request:

- *e2e_time_ms:* client send to client receive;
- *server_total_time_ms:* server receive to response completion;
- *queue_time_ms:* time spent in the expert-specific queue before batching;
- *load_time_ms:* time spent loading or attaching a model / adapter;
- *inference_time_ms:* pure generation time inside the backend;
- *orchestrator_time_ms, preprocess_time_ms:* router and tokenization overheads.

The major bottleneck here was the Memory Limit of 16 Gb imposed by our device. Memory was quickly filled, as shown in Figure 4 in the base server, where only 2 models could be loaded and inferred at a time. CPU usage was always 100% when requests came in bursts as shown in Figure 5. The best part with using quantized GGUF models was that we were able to load all the five models at once on the CPU, which reduced the queue time and repeated load-unload times by a lot, as shown in Figure 6

8.1.1 Latency over Time and Load

Figure 7 shows end-to-end latency over wall-clock time for all three techniques, aligned by client timestamps. GGUF exhibits the tightest latency band with relatively low variance. The Baseline HF server shows frequent spikes under bursty load, and the Dual-LM server experiences prolonged periods of extremely high latency when queues back up.

To contextualize these spikes, Figure 8 plots requests-per-second (RPS) over time. Peaks in RPS align closely with queue build-up and e2e latency spikes, especially for Baseline and Dual-LM, while GGUF remains substantially more stable.

8.1.2 Latency Component Breakdown

We further decompose server time into queue, load, and inference components. Figure 9 shows a stacked area view over time; queue time dominates the tail, followed by inference. GGUF makes load time almost negligible, while the Baseline HF server frequently spends several seconds loading models.

Figure 10 shows the e2e latency histogram. GGUF has a relatively tight unimodal distribution with a much shorter tail. Baseline and Dual-LM exhibit heavy right tails, with Dual-LM particularly pathological under sustained load.

A correlation heatmap (Figure 11) confirms that *queue_time_ms* is the primary driver of e2e latency across techniques, followed by *inference_time_ms*. Load time correlates strongly with latency for the Baseline HF server but is negligible for GGUF.

8.1.3 Per-component Comparison Across Techniques

Table 2 summarizes the p95 latency for each major component across the three server variants.

From Table 2 we observe:

- **Queue time dominates tail latency.** All three systems are ultimately bottlenecked by queueing, but GGUF reduces p95 queue time by roughly 3–4× compared to Baseline and Dual-LM (44.8s vs. 170.7s / 193.6s).

- **Baseline HF suffers catastrophic load time.** Its p95 load time (118.9s) is two orders of magnitude larger than GGUF (0.8s), and directly amplifies both queue and total latency.

- **GGUF is the fastest inference backend.** GGUF achieves the lowest p95 inference time (31.2s), Dual-LM is moderate (41.0s), and Baseline HF is slowest (62.4s).

- **Dual-LM collapses under load despite zero load time.** Removing model-load cost is not enough: p95 total time for Dual-LM (214.1s) is actually worse than Baseline HF (198.1s) because queues explode under bursty traffic.

- **Overall ranking.** Combining all components, GGUF is clearly the most robust deployment strategy (p95 total 57.9s), while Baseline HF and especially Dual-LM are not viable under sustained load.

Component	GGUF	Baseline HF	Dual-LM
Queue time (ms)	44,820	170,717	193,563
Load time (ms)	800	118,972	—
Inference time (ms)	31,172	62,417	40,996
Total time (ms)	57,959	198,099	214,102

Table 2: p95 latency (in milliseconds) for key components across techniques. Dual-LM has no explicit model load (only adapter switches), hence "—" for load time.

8.2 Qualitative Case Studies

To complement the aggregate metrics, we inspect individual routed conversations. Below we show representative examples where the router and experts behave as intended.

Bipolar: mood swings between extremes.

> **User:** "My mood keeps swinging between feeling like a genius and feeling like a complete failure."
>
> **Bipolar expert:** "It's not uncommon for people experiencing bipolar disorder to have pronounced mood swings. These shifts can be intense and affect many aspects of life, including relationships, work, and daily activities. The feelings of being a 'genius' and a 'failure' are often linked to the manic and depressive phases, respectively. It's important to

> seek professional help to manage these fluctuations and develop effective coping strategies. A therapist can help you identify patterns in your mood, work through these feelings, and create a personalised plan to improve daily functioning."

In this case, the router assigns the query to the bipolar expert, which in turn correctly frames the symptom pattern in terms of manic and depressive episodes and recommends appropriate clinical follow-up.

Anxiety: social/performance anxiety.

> **User:** "My heart races, my hands shake, and I can't focus whenever I have to talk to someone new."

> **Anxiety expert:** "These symptoms suggest a form of social anxiety, sometimes called social phobia. It involves an intense fear of social or performance situations where you might feel judged, evaluated, or embarrassed. Physical sensations such as a racing heart, shaking hands, and difficulty focusing are common manifestations of this anxiety. If these experiences are affecting your daily life, it can be helpful to speak with a mental health professional. They can work with you to understand your fears, develop coping strategies, and gradually build confidence in social situations."

Here the system correctly interprets the prompt as social anxiety, explains the fear-of-judgement mechanism, and provides standard psychoeducational guidance.

OCD: obsessions, rituals, and ERP.

> **User:** "My mind gets stuck on certain thoughts and I can't move on until I repeat a ritual."

> **OCD expert:** "That pattern is common in Obsessive–Compulsive Disorder (OCD). Repeating rituals can temporarily reduce the distress caused by unwanted thoughts, but over time they can also intensify anxiety and interfere with daily life. Treatments such as cognitive–behavioural therapy (CBT) and exposure and response prevention (ERP) are specifically designed to help with this. A therapist can work with you to understand your thought patterns, gradually reduce reliance on rituals, and develop more adaptive coping strategies."

The OCD expert explicitly names OCD, correctly links rituals to distress-reduction, and references CBT/ERP as gold-standard interventions.

OCD: checking compulsion.

> **User:** "I have to check the door multiple times before I can leave, or I feel something bad will happen."

> **OCD expert:** "Needing to check the door repeatedly, combined with a fear that something bad will happen if you do not, is a common presentation of Obsessive–Compulsive Disorder (OCD). Intrusive thoughts about danger can trigger compulsive checking as a way to temporarily feel safer. Cognitive–behavioural therapy and exposure-based approaches can help you gradually reduce the checking and tolerate the anxiety that comes up when you resist the urge to check."

This example shows that the system also handles canonical checking-type compulsions in a clinically plausible way.

Schizophrenia: auditory hallucinations.

> **User:** "I hear voices commenting on what I'm doing, but nobody else seems to hear them."

> **Schizophrenia expert:** "What you describe sounds like auditory hallucinations, which can be a challenging part of conditions such as schizophrenia. These voices can be intrusive and may make comments or criticisms about you or your actions, even though other people cannot hear them. It is important to talk with a mental health professional, such as a psychiatrist or psychologist, who can help you understand what is happening and discuss treatment options to reduce the impact of the voices on your daily life."

The router correctly assigns this prototypical psychotic symptom to the schizophrenia expert, which responds with appropriate framing and a recommendation for specialist care.

Optional examples. For completeness, we also observe more generic but still reasonable responses:

- *Bipolar fatigue:* prompts such as "I feel like I'm constantly tired, even when I sleep enough." are routed to the bipolar expert, which provides a generic discussion of fatigue and encourages medical evaluation. This is sensible but less clearly disorder-specific.

- *Worthlessness thoughts:* prompts like "I keep thinking that everyone would be better off without me, even if I don't want to do anything extreme." are often routed to the anxiety expert, which responds with reassurance, self-worth framing, and suggestions for seeking support. These cases illustrate grey areas where content overlaps with depression, anxiety, and suicidality.

9 Discussion

Our results illustrate several trade-offs in building a practical MoE system under resource constraints.

Data generation vs. quality. Using a strong external LLM (Gemini-2.5-Flash-Lite) to generate QA pairs from textbooks yields high-quality supervision with relatively little engineering effort, but inherits both the biases and stylistic preferences of the external model and the source books.

Perplexity vs. specialization. Domain-specific fine-tuning on small datasets slightly worsens global cross-entropy but improves task-specific quality. For mental-health dialogue, we explicitly prioritize clinically relevant content and routing behaviour over raw perplexity, choosing 3-epoch experts as a compromise between generalization and specialization.

Router training data. Training the router on synthetic instructions yields excellent held-out accuracy but does not fully capture the diversity of real user queries. Training on raw textbook spans lowers macro-F1 slightly but produces more intuitive routing in practice. This highlights a common pattern: router performance on synthetic benchmarks is not always predictive of MoE behaviour in the wild.

Latency bottlenecks. Across all techniques, queue time is the dominant contributor to tail latency, with inference as the second-largest factor. Baseline HF deployment suffers from catastrophic load times (p95 $\approx$119 s), which amplify queueing and lead to p95 total latency near 200 s. The Dual-LM variant removes load time but still collapses under load because queue time becomes extreme (p95 $\approx$194 s). GGUF, by contrast, reduces both load and inference times and keeps queueing under control, achieving p95 total latency of only 57,959 ms, a 3–4$\times$ improvement over the alternatives.

Deployment choices. LoRA experts are convenient during training and experimentation, but HF CPU inference is slow and tuning eviction/loading logic is painful. Merging and quantizing experts to GGUF allows us to leverage `llama.cpp`'s optimized CPU kernels, yielding both lower average latency and vastly better tail behaviour under realistic Azure-style traces. In short, **quantization and runtime choice are as important as model architecture** for practical MoE systems.

Limitations

This work has several limitations:

- **Synthetic data dependence.** All training data for experts is generated by Gemini from textbooks, which may limit diversity and propagate biases from both sources.

- **No real user evaluation.** We do not conduct studies with actual users or clinicians; all qualitative assessments are based on the authors' judgments.

- **Safety and scope.** The system is not intended for deployment as a real mental-health assistant. It does not implement safety filters, crisis detection, or referral mechanisms.

- **Limited model size and techniques.** We restrict ourselves to 1B-parameter experts and simple linear routing. Larger experts or more sophisticated neural routers (or multi-expert selection) might yield better performance but were out of scope for this assignment.

9.1 Failure Cases and Limitations

While the system behaves plausibly on many prompts, qualitative inspection also reveals important failure modes.

Misrouting delusional content to anxiety. Some prompts are linguistically mild but clinically closer to psychosis. For example:

> **User:** “I get the sense that messages on TV or online are secretly directed at me.”
>
> **Routed expert (anxiety):** “The idea that messages on TV or online are aimed directly at you can feel unsettling. In most cases, the information we encounter is not personal but created for a broad audience to inform or entertain. Sometimes this feeling comes from misinterpreting the context or intention behind what you see. It can help to pause, consider the bigger picture, and check whether there is any concrete evidence that these messages are specifically about you before drawing conclusions.”

Clinically, this kind of “messages are secretly directed at me” content is much closer to delusions of reference and would typically raise concern about psychotic processes. The router, however, assigns it to the anxiety expert, which treats it as a media-literacy or misinterpretation issue and does not surface the potential for psychosis or recommend urgent evaluation. This highlights a safety-relevant limitation of a purely text-based, TF–IDF router without explicit risk-awareness.

Anhedonia routed to OCD expert. We also observe confusion between affective symptoms and OCD. For instance:

> **User:** “Lately nothing feels enjoyable, even the things I used to love.”
>
> **Routed expert (OCD):** “A loss of pleasure in activities that used to feel rewarding is often associated with depression. People sometimes describe it as a sense of detachment or not getting anything out of things they once enjoyed. If this persists or starts to interfere with daily life, it can be important to talk with a mental health professional. They can help you explore possible causes, including depression, and work with you on steps to gradually re-engage with activities and rebuild a sense of enjoyment.”

Here the router sends a classic anhedonia prompt to the OCD expert. The expert’s answer itself frames the issue in terms of depression and recommends appropriate next steps, but the misrouting indicates that the router struggles to reliably distinguish between depressive and OCD-like language when overlap in vocabulary is high. This aligns with our quantitative confusion patterns and underscores that routing errors can occur even when individual experts behave reasonably.

10 Conclusion

We presented an end-to-end MoE-style system for mental-health dialogue on resource-constrained hardware. By combining synthetic QA generation from textbooks, LoRA-based domain experts, a TF–IDF + Logistic Regression router trained on book spans, and a quantized GGUF deployment via `llama.cpp`, we obtain:

- strong routing performance (macro-F1 0.83 on 3.99M test spans, with clinically meaningful confusions),
- domain-specialized experts that produce more clinically grounded responses than the base model, and
- a GGUF-based server whose p95 latency is 3–4$\times$ lower than Baseline HF and Dual-LM variants under Azure-style traces.

Although designed as a course project, the architecture and experimental setup highlight several generally useful patterns: using external LLMs for targeted data creation, preferring simple but robust routers, treating quantization and runtime as first-class design decisions, and evaluating MoE systems under realistic load patterns rather than only static benchmarks. Future work could explore more principled preference optimization for expert responses, richer safety mechanisms, and learned neural gating networks, while preserving the lightweight, latency-aware deployment philosophy demonstrated here.

References

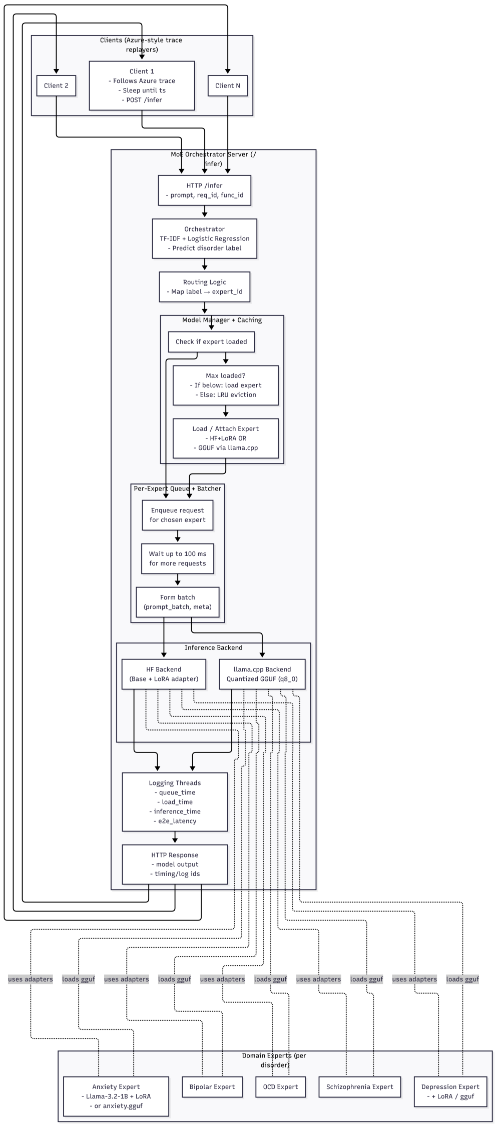

Figure 1: Overall architecture of the MoE system. A TF–IDF + Logistic Regression orchestrator routes each query to one domain expert. Experts are LoRA- or GGUF-based variants of Llama-3.2-1B-Instruct. A custom server manages model loading, batching, and logging under Azure-style traces.

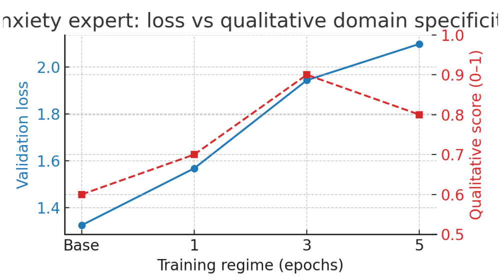

Figure 2: Illustrative trade-off between epochs and behaviour of a single expert (anxiety). While cross-entropy is best for the base model, 3 epochs produce the most clinically grounded yet conversational responses.

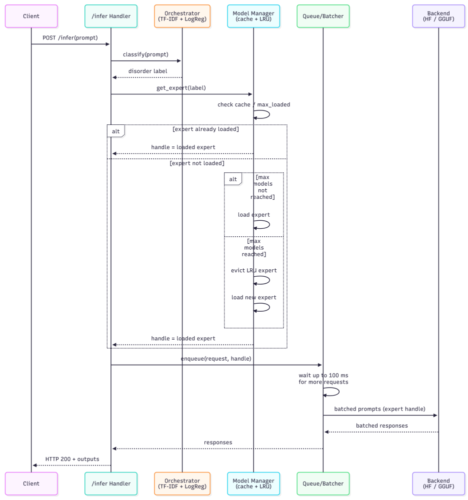

Figure 3: server timeline / state diagram, showing routing, model loading/eviction, batching and inference.

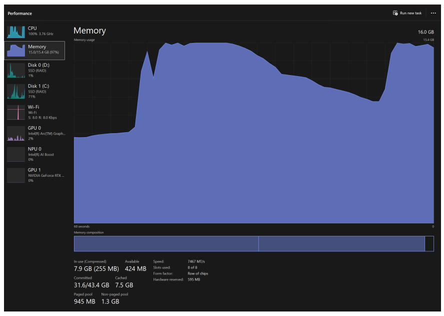

Figure 4: Base Server: Memory usage lowers after the model is loaded, but still we can only load 2 models.

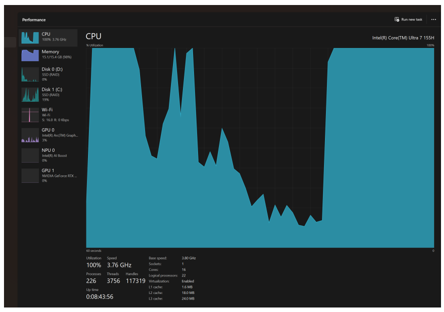

Figure 5: CPU Usage reaches 100% when requests come in bursts

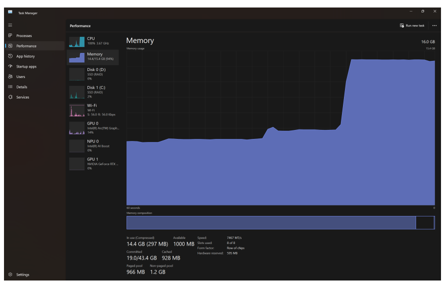

Figure 6: Memory Usage in Llama-CPP and Quantized GGUF models where all the 5 models were loaded at once

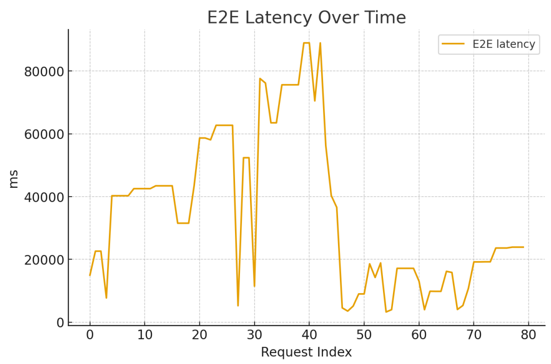

Figure 7: End-to-end latency over time under Azure-style traces for three server variants. GGUF maintains the tightest band; Baseline and Dual-LM exhibit large spikes.

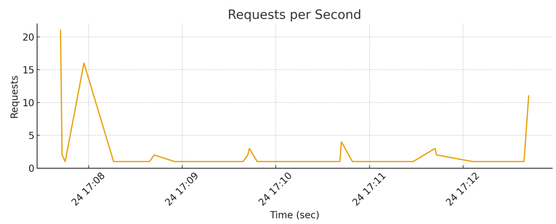

Figure 8: Requests per second (RPS) over time. Burst regions correlate with queue-time spikes in Baseline and Dual-LM, whereas GGUF absorbs load more gracefully.

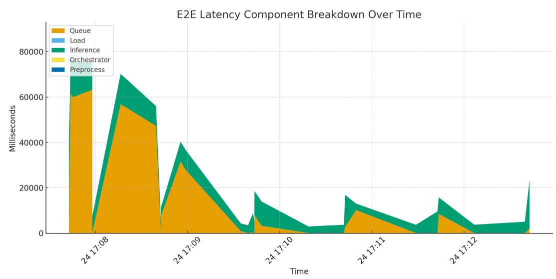

Figure 9: Stacked area breakdown of server time into queue, load, and inference. Queue time drives most latency spikes; load time is substantial only in the Baseline HF server.

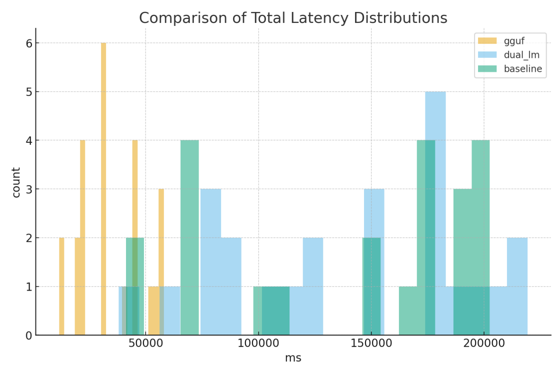

Figure 10: Distribution of end-to-end latency across all requests. GGUF has the tightest distribution; Baseline and especially Dual-LM show heavy long tails.

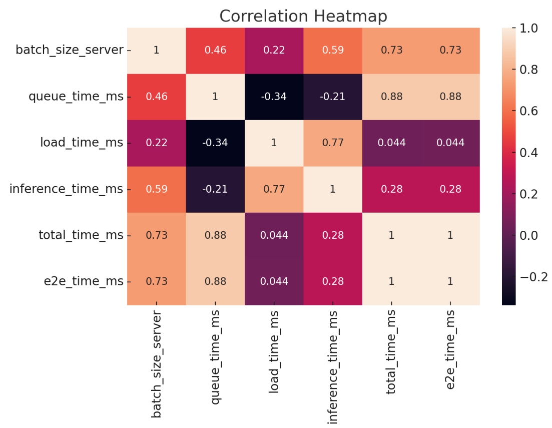

Figure 11: Correlation between latency components and total end-to-end latency. Queue time is the strongest predictor, followed by inference time.