

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

Mohammad Faiz and Vaibhav Bajaj

Department of Computer Science and Engineering

Indian Institute of Technology Delhi

{cs5210128, cs5210126}@iitd.ac.in

Abstract

Model Repo: <https://huggingface.co/reaper-113/Llama-3>.

2-1B-Instruct-MOE-Finetuned

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 $\sim 19.9M$ 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}.\nRead the text below and generate 5\nunique Question-Answer pairs. Focus\non clinical/therapeutic advice found\nin the text. Format strictly as a\nJSON list: [{ "instruction": "User\nQuestion", "output": "Expert Answer"\n}]\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 $\sim 3,750$ 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 (~ 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 hyper-ventilation 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 ~ 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;

335	6. it performs batched inference on the CPU backend;	7 Device Specifications	378
336	337	338	379
	7. finally, it sends responses back to clients and logs timing information.	• Memory - 16.0 GB; Speed: 7467 MT/s	380
		• CPU - Intel(R) Core(TM) Ultra 7 155H; Base speed: 3.80 GHz;	381
		• Cores: 16; Logical processors: 22;	382
		• L1 cache: 1.6 MB; L2 cache: 18.0 MB; L3 cache: 24.0 MB	383
			384
339	We run separate logging threads on both server and clients to collect trace-aligned metrics: se- lected experts, queue times, load times, inference times, and end-to-end latencies.		
340			
341			
342			
343	6.3 Optimizations	8 Experimental Results	385
344	We implemented several optimization passes:	8.1 End-to-End Latency	386
345	Naive version. Each expert is a full HF model (base + LoRA weights) loaded and unloaded in- dependently. Loading a 1B model on CPU can	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):	387
346	take several seconds, and repeated loads dominate		388
347	end-to-end latency.		389
348			390
349			
350	Base Optimisations. Due to memory limits of		
351	our device, we needed to create a strict check for		
352	the number of models loaded at a time in the mem- ory to not cause BSOD errors in Windows, along		
353	with LRU eviction policy. Apart from this, we also		
354	implemented batching support in the server.		
355			
356	Adapter-switch version. Instead of unloading		
357	the entire model, we keep the <i>base</i> model resident		
358	in memory and only switch PEFT adapters when		
359	changing domains. This greatly reduces load time,		
360	since adapter weights are small. We further extend		
361	this to support multiple bases (e.g., one base for		
362	some disorders and another for others), each with		
363	its own set of adapters.		
364	GGUF + 11ama.cpp version. For maximum		
365	CPU efficiency, we:		
366	1. merge each LoRA adapter into the base		
367	Llama-3.2-1B-Instruct weights to obtain a full		
368	FP16 expert;		
369	2. convert the merged HF model to GGUF using		
370	convert_hf_to_gguf.py;		
371	3. quantize to 8-bit (q8_0) GGUF format;		
372	4. serve these experts via 11ama-cpp-python,		
373	which provides a fast CPU backend with		
374	batched inference.		
375	This pipeline preserves the semantics of the		
376	trained experts while reducing memory footprint		
377	and improving inference throughput.		
	The major bottleneck here was the Memory		415
	Limit of 16 Gb imposed by our device. Memory		416
	was quickly filled, as shown in Figure 4 in the base		417

418 server, where only 2 models could be loaded and
 419 inferred at a time. CPU usage was always 100%
 420 when requests came in bursts as shown in Figure 5.
 421 The best part with using quantized GGUF models
 422 was that we were able to load all the five models
 423 at once on the CPU, which reduced the queue time
 424 and repeated load-unload times by a lot, as shown
 425 in Figure 6

426 8.1.1 Latency over Time and Load

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

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

440 8.1.2 Latency Component Breakdown

441 We further decompose server time into queue, load,
 442 and inference components. Figure 9 shows a
 443 stacked area view over time; queue time domi-
 444 nates the tail, followed by inference. GGUF makes
 445 load time almost negligible, while the Baseline HF
 446 server frequently spends several seconds loading
 447 models.

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

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

458 8.1.3 Per-component Comparison Across 459 Techniques

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

462 From Table 2 we observe:

- 463 • **Queue time dominates tail latency.** All three
 464 systems are ultimately bottlenecked by queueing,
 465 but GGUF reduces p95 queue time by

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.

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.

486 8.2 Qualitative Case Studies

487 To complement the aggregate metrics, we inspect
 488 individual routed conversations. Below we show
 489 representative examples where the router and ex-
 490 perts behave as intended.

491 Bipolar: mood swings between extremes.

492 **User:** “My mood keeps swinging be-
 493 tween feeling like a genius and feeling
 494 like a complete failure.”

495 **Bipolar expert:** “It’s not uncommon for
 496 people experiencing bipolar disorder to
 497 have pronounced mood swings. These
 498 shifts can be intense and affect many
 499 aspects of life, including relationships,
 500 work, and daily activities. The feelings
 501 of being a ‘genius’ and a ‘failure’ are of-
 502 ten linked to the manic and depressive
 503 phases, respectively. It’s important to

504	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.”	552
505		553
506		554
507		555
508		556
509		557
510	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.	558
511		559
512		560
513		561
514		562
515	Anxiety: social/performance anxiety.	
516		563
517	User: “My heart races, my hands shake, and I can’t focus whenever I have to talk to someone new.”	564
518		565
519	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.”	566
520		567
521		568
522		569
523		570
524		571
525		572
526		573
527		574
528		575
529		576
530		577
531		578
532		579
533		580
534	Here the system correctly interprets the prompt as social anxiety, explains the fear-of-judgement mechanism, and provides standard psychoeducational guidance.	581
535		582
536		583
537		584
538	OCD: obsessions, rituals, and ERP.	585
539		586
540	User: “My mind gets stuck on certain thoughts and I can’t move on until I repeat a ritual.”	587
541		588
542	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.	589
543		590
544		591
545		592
546		593
547		594
548		595
549		596
550		597
551	The router correctly assigns this prototypical psychotic symptom to the schizophrenia expert, which responds with appropriate framing and a recommendation for specialist care.	598
		599
		600

601 **Optional examples.** For completeness, we also
602 observe more generic but still reasonable re-
603 sponses:

- 604 • *Bipolar fatigue*: prompts such as “I feel
605 like I’m constantly tired, even when I sleep
606 enough.” are routed to the bipolar expert,
607 which provides a generic discussion of fatigue
608 and encourages medical evaluation. This is
609 sensible but less clearly disorder-specific.
- 610 • *Worthlessness thoughts*: prompts like “I keep
611 thinking that everyone would be better off
612 without me, even if I don’t want to do any-
613 thing extreme.” are often routed to the anxiety
614 expert, which responds with reassurance,
615 self-worth framing, and suggestions for seek-
616 ing support. These cases illustrate grey areas
617 where content overlaps with depression, anxiety,
618 and suicidality.

619 9 Discussion

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

622 **Data generation vs. quality.** Using a strong ex-
623 ternal LLM (Gemini-2.5-Flash-Lite) to generate
624 QA pairs from textbooks yields high-quality super-
625 vision with relatively little engineering effort, but
626 inherits both the biases and stylistic preferences of
627 the external model and the source books.

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

636 **Router training data.** Training the router on syn-
637 thetic instructions yields excellent held-out accu-
638 racy but does not fully capture the diversity of real
639 user queries. Training on raw textbook spans low-
640 ers macro-F1 slightly but produces more intuitive
641 routing in practice. This highlights a common pat-
642 tern: router performance on synthetic benchmarks
643 is not always predictive of MoE behaviour in the
644 wild.

645 **Latency bottlenecks.** Across all techniques,
646 queue time is the dominant contributor to tail lat-
647 ency, with inference as the second-largest factor.

648 Baseline HF deployment suffers from catastrophic
649 load times ($p95 \approx 119$ s), which amplify queueing
650 and lead to $p95$ total latency near 200 s. The Dual-
651 LM variant removes load time but still collapses un-
652 der load because queue time becomes extreme ($p95$
653 ≈ 194 s). GGUF, by contrast, reduces both load and
654 inference times and keeps queueing under control,
655 achieving $p95$ total latency of only 57,959 ms, a
656 3–4× improvement over the alternatives.

657 **Deployment choices.** LoRA experts are conve-
658 nient during training and experimentation, but HF
659 CPU inference is slow and tuning eviction/loading
660 logic is painful. Merging and quantizing experts
661 to GGUF allows us to leverage `llama.cpp`’s opti-
662 mized CPU kernels, yielding both lower average
663 latency and vastly better tail behaviour under re-
664 alistic Azure-style traces. In short, **quantization**
665 and **runtime choice are as important as model**
666 **architecture** for practical MoE systems.

667 Limitations

668 This work has several limitations:

- 669 • **Synthetic data dependence.** All training data
670 for experts is generated by Gemini from text-
671 books, which may limit diversity and propa-
672 gate biases from both sources.
- 673 • **No real user evaluation.** We do not conduct
674 studies with actual users or clinicians; all qual-
675 itative assessments are based on the authors’
676 judgments.
- 677 • **Safety and scope.** The system is not intended
678 for deployment as a real mental-health assis-
679 tant. It does not implement safety filters, crisis
680 detection, or referral mechanisms.
- 681 • **Limited model size and techniques.** We re-
682 strict ourselves to 1B-parameter experts and
683 simple linear routing. Larger experts or more
684 sophisticated neural routers (or multi-expert
685 selection) might yield better performance but
686 were out of scope for this assignment.

687 9.1 Failure Cases and Limitations

688 While the system behaves plausibly on many
689 prompts, qualitative inspection also reveals impor-
690 tant failure modes.

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

694	User: “I get the sense that messages on TV or online are secretly directed at me.”	743
695		744
696		745
697		746
698		747
699		748
700		749
701		750
702		751
703		752
704		753
705		754
706		755
707		756
708		757
709		758
710		759
711		760
712		761
713		762
714		763
715		764
716		765
717		766
718		767
719		768
720		769
721		770
722		771
723		772
724		773
725		774
726		775
727		776
728		777
729		778
730		779
731		780
732		781
733		782
734		783
735		784
736		785
737	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	786
738		787
739		788
740		789
741		790
742		791
743		
744		
745		
746		
747		
748		
749		
750		
751		
752		
753		
754		
755		
756		
757		
758		
759		
760		
761		
762		
763		
764		
765		
766		
767		
768		
769		
770		
771		
772		
773		
774		
775		
776		
777		
778		
779		
780		
781		
782		
783		
784		
785		
786		
787		
788		
789		
790		
791		

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× 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

743		744
744		745
745		746
746		747
747		748
748		749
749		750
750		751
751		752
752		753
753		754
754		755
755		756
756		757
757		758
758		759
759		760
760		761
761		762
762		763
763		764
764		765
765		766
766		767
767		768
768		769
769		770
770		771
771		772
772		773
773		774
774		775
775		776
776		777
777		778
778		779
779		780
780		781
781		782
782		783
783		784
784		785
785		786
786		787
787		788
788		789
789		790
790		791

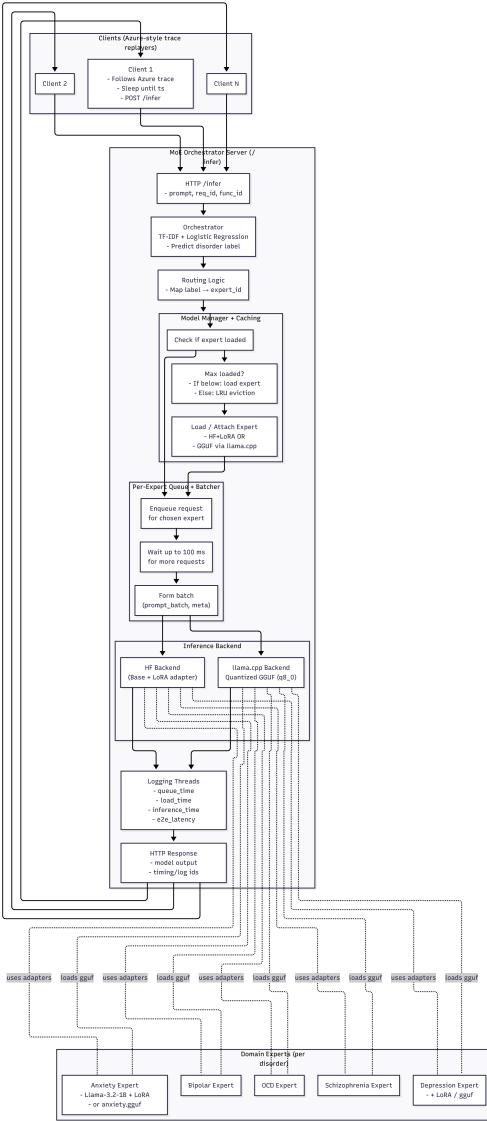


Figure 1: Overall architecture of the MoE system. A TF-IDF + Logistic Regression orchestrator routes each query to one domain expert. Experts are LoRA- or GGUF-based variants of Llama-3.2-1B-Instruct. A custom server manages model loading, batching, and logging under Azure-style traces.

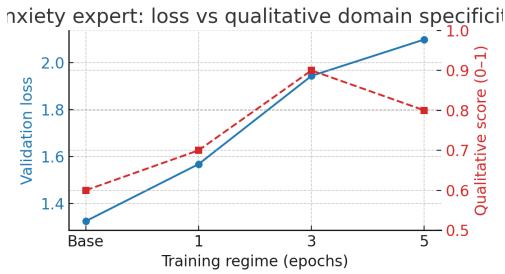


Figure 2: Illustrative trade-off between epochs and behaviour of a single expert (anxiety). While cross-entropy is best for the base model, 3 epochs produce the most clinically grounded yet conversational responses.

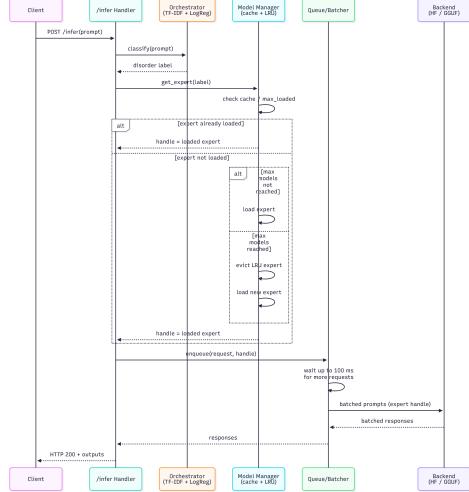


Figure 3: server timeline / state diagram, showing routing, model loading/eviction, batching and inference.



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

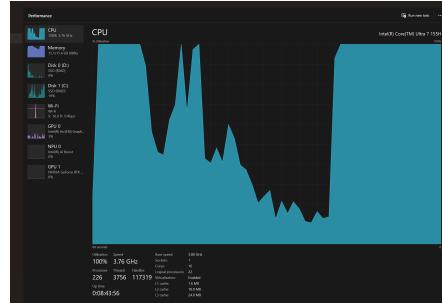


Figure 5: CPU Usage reaches 100% when requests come in bursts

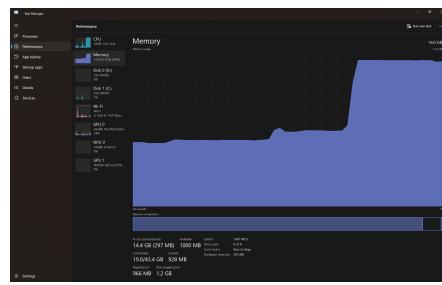


Figure 6: Memory Usage in Llama-CPP and Quantized GGUF models where all the 5 models were loaded at once



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.

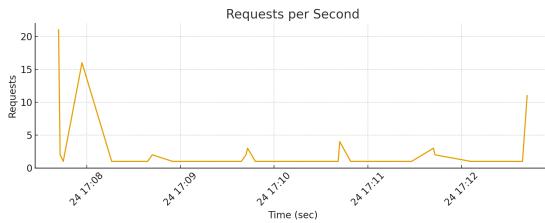


Figure 8: Requests per second (RPS) over time. Burst regions correlate with queue-time spikes in Baseline and Dual-LM, whereas GGUF absorbs load more gracefully.

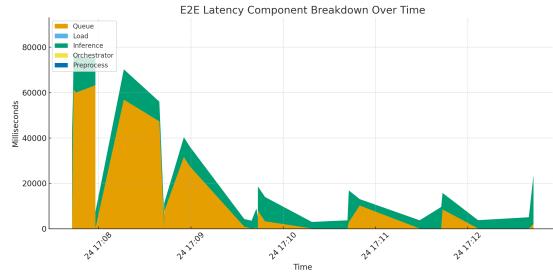


Figure 9: Stacked area breakdown of server time into queue, load, and inference. Queue time drives most latency spikes; load time is substantial only in the Baseline HF server.

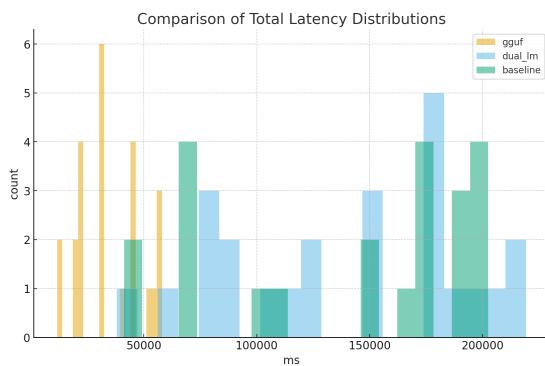


Figure 10: Distribution of end-to-end latency across all requests. GGUF has the tightest distribution; Baseline and especially Dual-LM show heavy long tails.

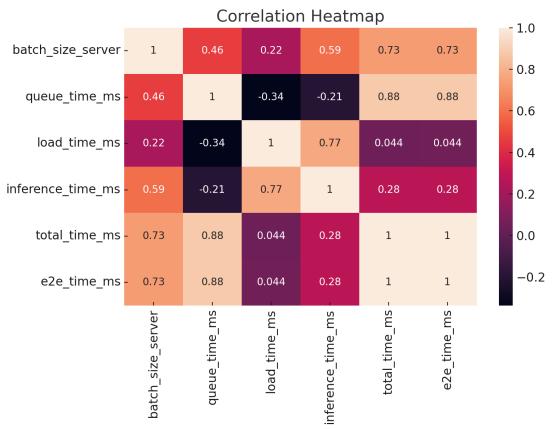


Figure 11: Correlation between latency components and total end-to-end latency. Queue time is the strongest predictor, followed by inference time.