Optimize response times by adding caching or background processing

**✅ 1. Add Caching to Avoid Repeated LLM Work**

If a user asks a question that was **already asked before**, don’t hit the LLM again — just return the cached response.

**📦 Option: In-Memory Caching (Good for local/dev)**

Add simple Map-based caching to LlmService:

ts

1. // app/Services/LlmService.ts

2. const memoryCache = new Map<string, string>()

3.

4. export default class LlmService {

5. static async generateResponse(prompt: string): Promise<string> {

6. if (memoryCache.has(prompt)) {

7. return memoryCache.get(prompt)!

8. }

9.

10. const res = await axios.post(

11. 'http://127.0.0.1:11434/api/generate',

12. {

13. model: 'llama3',

14. prompt,

15. stream: false,

16. },

17. {

18. httpAgent: agent,

19. }

20. )

21.

22. const output = res.data.response

23. memoryCache.set(prompt, output)

24. return output

25. }

26. }

27.

📦 For production, swap Map with Redis, Memcached, or even DB lookup with .firstOrCreate.

**✅ 2. Use Background Processing for Heavy Tasks**

Let store() or stream() return immediately, and **save to DB or call LLaMA in the background**.

**Option A: Using AdonisJS Queue (e.g., BullMQ)**

Install a queue runner like BullMQ:

bash

1. npm install bullmq

Then, offload prompt handling to a job:

ts

1. import { Queue } from 'bullmq'

2.

3. const llamaQueue = new Queue('llama')

4.

5. await llamaQueue.add('generate', {

6. userId,

7. prompt,

8. })

In your worker:

ts

1. llamaQueue.process('generate', async (job) => {

2. const { userId, prompt } = job.data

3. const response = await LlmService.generateResponse(prompt)

4. await Chat.create({ userId, prompt, response })

5. })

6.

🧠 Your API returns immediately, and LLM + DB write happen off-thread

**Option B: Do DB write after the response**

This is what you're already doing in stream() — but you could extend it by:

* Deferring DB writes slightly
* Batching common requests
* Debouncing user input

**⚡ Summary: Choose Your Strategy**

| **Strategy** | **When to Use** | **Benefits** |
| --- | --- | --- |
| 🔁 Caching | Repeated prompts, low variance | Instant response, save CPU |
| ⏳ Background Job | Heavy load, high user concurrency | Fast API, async processing |
| ✍️ Save-after-stream | Real-time UI | Instant feedback + reliable DB |

Let me know if you'd like to integrate Redis for persistent caching or set up BullMQ workers — I can walk you through that step-by-step!