# GAL Challenge: LLM LAB

**Step into the Lab.**

Let's build an experimental console where you'll dissect the quirks of large language models.

When working with Large Language Models, parameter choices like temperature and top_p can drastically change response characteristics. Build a tool that helps users understand these effects by generating multiple responses and analyzing their quality.This isn't just about calling an API — it's about *understanding* one. Build a system that turns the invisible (temperature, top_p, coherence) into something you can see, compare, and explain — a visual dashboard of how AI thinks when you turn the dials.

**Important Note on Expectations**: This challenge is designed to evaluate not just basic functionality, but your ability to deliver a polished, professional product that showcases your best work as a full-stack developer. We are looking for thoughtful design decisions, exceptional UI/UX, clean code, and clear articulation of your process. Bare-minimum implementations (e.g., rudimentary UIs, incomplete features, or unpolished demos) will not stand out. Aim to create something you'd confidently present to a client or team—prioritize quality, creativity, and completeness over rushing to finish. This is your opportunity to demonstrate excellence under realistic constraints.

| | |
|---|---|
| **What You Need to Build** | A full-stack web application where users can:<br>1. *Input a prompt and specify parameter ranges*<br>2. *Generate multiple LLM responses using different parameter combinations*<br>3. *View responses alongside programmatically calculated quality metrics*<br>4. *Compare responses to understand which parameters produced better results*<br>5. *Export their experiments for later reference*<br><br>**Important**: This should be a product you'd be proud to demo to a potential customer. Treat the UI/UX as a core deliverable, not an afterthought. Focus on creating an intuitive, visually appealing experience that feels modern and reliable. |
| **Key Technical Challenge** | **You must design and implement your own quality metrics** that evaluate LLM responses programmatically. These metrics should provide meaningful insights into response characteristics without relying on another LLM call for evaluation.<br>Think about what makes a good LLM response and how you might measure that objectively. Consider aspects like completeness, coherence, length appropriateness, structural patterns, or other characteristics you deem important. |

| | |
|---|---|
| **Requirements** | **Must Have:**<br>• A functional user interface for the complete workflow<br>• Backend integration with an LLM API (or mock service if no API access)<br>• At least 1 custom quality metrics with clear rationale<br>• Some form of data persistence for experiments<br>• Ability to compare results across different parameter combinations<br>• Export functionality<br>• **Polished, professional UI/UX** following modern design principles (e.g., responsive design, accessibility considerations, smooth interactions, and a cohesive visual theme)<br>• **Deployed and publicly accessible** (use any hosting platform)<br>• **Demo video** (5-10 minutes) with **audio narration** where you talk through the solution, features, and decisions (see Deliverables for details) |
| **Technical Constraints** | • **Frontend**: Use a full-stack SSR framework such as Next.js, integrated with state management and routing tools like TanStack Query or React Router. Ensure server-side rendering for better performance and SEO where applicable.<br>• **Backend**: Implement using TypeScript (e.g., Node.js with Express/NestJS) or Python-based framework like FastAPI. The backend must handle API calls to the LLM, process parameters, compute metrics, and manage data persistence.<br>• Handle errors and edge cases appropriately (e.g., API rate limits, invalid inputs, network failures)<br>• Code should be well-organized, maintainable, and include type validations where possible<br>• Use modern web technologies overall, but adhere to the specified stacks—no exceptions |
| **Not Required** | • User authentication<br>• Extensive test suite |

| What We're Looking For | We want to understand how you:<br>• Approach ambiguous problems and make design decisions<br>• Balance time constraints with quality while delivering your best work<br>• Structure full-stack applications using the specified tech stack<br>• Think about metrics and measurement<br>• Handle real-world challenges like API errors and edge cases<br>• **Design intuitive, professional user experiences** with strong UI/UX best practices (e.g., Figma-inspired layouts, component reusability, animations/transitions for engagement)<br>• **Apply visual design principles** (e.g., color theory, typography, spacing) to create a premium feel<br>• **Articulate and present your technical decisions** clearly in the video and documentation<br>• Make trade-offs when building under time pressure, and reflect on them<br>Think of this as a product you'd demo to a customer—the experience should feel polished, thoughtful, and production-ready. |
| --- | --- |

**Deliverables**

1. **Live Application**
   - Deployed URL where we can access and test your application
   - Should be stable and accessible for at least 2 weeks

2. **Source Code**
   - Repository link (GitHub, GitLab, etc.)
   - README with setup instructions, architectural documentation, tech stack details, and quick-start guide

3. **Demo Video (5-10 minutes, Mandatory with Audio Narration)**
   - **This is required—submissions without a demo video featuring clear audio narration will not be reviewed.**
   - Walk through the live application and its features (e.g., input prompt, generate responses, view metrics, compare/export)
   - Explain your key technical and design decisions (e.g., why you chose specific tech, how metrics work, UI rationale)
   - Discuss your quality metrics and why you chose them, including rationale and implementation
   - Share challenges you faced (e.g., API integration issues, metric design) and how you solved them
   - Mention what you'd improve with more time and any trade-offs made
   - Use screen recording with your voiceover to narrate—ensure audio is clear, professional, and engaging
   - Upload to YouTube (unlisted), Loom, or similar platform and include the link

4. **Documentation**
   - Architectural approach and key decisions (e.g., data flow, API endpoints, component

| | |
|---|---|
| **Time Estimates** | The first task is essential, and often overlooked as an important and difficult quality of advanced engineering. Spend some preliminary drawing board time to break this project into smaller pieces, and estimate how long each will take you. Download a copy of attached “Estimates.CSV (Challenges) - Sheet1” and update it with your estimated information as your first commit. Time yourself and update this file as time goes on, with a new column showing actual logged time so it can be compared to the initial guess. Good engineers are bad at this sometimes, because of the notorious unpredictability of unknowns in technical projects. We are looking at your process, and promise we will not be phased by poor estimates. |
| **A Few Notes** | • **Ambiguity is intentional**—make reasonable decisions and document them thoroughly to showcase your problem-solving<br>• **LLM assistants are allowed** but you should deeply understand your code and be ready to discuss it in detail (e.g., in a follow-up interview)<br>• **Perfect is the enemy of done**—a working core feature set with polish is better than incomplete ambitious features, but prioritize excellence in execution<br>• **Be prepared to discuss** your implementation, design choices, and video content in detail<br>• **Quality over quantity**—a well-executed, polished core experience with professional presentation beats many half-finished features<br>• **Free hosting is fine**—Vercel (ideal for Next.js), Netlify, Railway, Render, etc., but ensure full-stack compatibility |

| Submission | Provide:<br>1. Link to deployed application<br>2. Link to source code repository<br>3. Link to demo video (with audio narration)<br>4. Link to time estimates<br>5. Any additional context or notes<br><br>**Final Reminder**: Showcase your best work—this is your chance to impress with a high-quality, professional submission. We're excited to see what you build! |
| --- | --- |