LLMOps Database / Vanishing Gradients

Overview

This episode of the *Vanishing Gradients* podcast is hosted by Hugo, featuring Alex (from ZenML) as the guest. The primary focus is on a comprehensive database Alex and his team created, cataloging over 400 real-world LLM-based deployments and the key lessons learned from them. They discuss how engineers can leverage these insights—covering prompt engineering, architecture patterns, tool selection, evaluation strategies, and production pitfalls—to build more robust applications. Major themes include the importance of solid software engineering fundamentals, structured workflows, limiting blast radius in production, and thoughtfully transitioning prototypes into real-world systems. Key terms and concepts explained include "prompt engineering," "human in the loop," "RAG" (Retrieval-Augmented Generation), "agentic systems," and "observability."

1. The Power of Diverse Use Cases

The database reveals a remarkable variety of LLM applications—from simple text generation to complex multimodal and enterprise-scale solutions—underscoring just how flexible large language models can be. Engineers can learn from myriad examples of what worked (and what didn't) across healthcare, banking, real estate, and more. By exploring these diverse scenarios, developers gain fresh ideas and see proven patterns for everything from small internal pilots to high-stakes, customer-facing deployments. Ultimately, a broad view of use cases helps teams identify pitfalls and refine their own approach.

Quote (alex):

"It's clear a lot of people are doing a lot of different things. So think of the diversity of approaches and use cases—and this only continues to expand."

2. Fundamentals Matter More Than Ever

LLM-driven products still rely on classic software engineering best practices. These include DevOps essentials like stable deployments, robust error handling, scaling strategies, and version control for iterative improvements. Even though LLMs enable new kinds of capabilities, teams must not neglect the fundamentals that keep software functional and reliable at scale. When you mix complex machine intelligence with real user data, disciplined engineering is indispensable.

Quote (alex):

"All of the classic software engineering, like bread and butter DevOps stuff—it still matters. It's not a demo app anymore once you have real users and production data."

3. Start With the Problem, Not the Technology

The most successful LLM-powered products do not start with "Let's add AI!" but rather with a clear business or user need. This problem-first mindset guides the selection of model, design of the solution, and success metrics from the start. Companies that skip problem definition and go straight to adding AI often waste resources and end up with subpar products. Engineers should anchor all design and development in a real, validated problem to solve.

Quote (alex):

"The lasting products or experiences are the ones driven by the problem rather than 'Let's just dump an LLM in.' It's all about starting from a specific use case and user need."

4. Prompt Engineering is Surprisingly Hard

Despite popular belief, prompt engineering isn't trivial—especially in production, where small changes can drastically alter results. Engineers must manage brittle prompts, create robust versioning systems, and handle the interplay between model updates and prompt shifts. It requires a blend of linguistic and technical skills, and a mindset that goes beyond coding. Proper prompt management is essential for controlling output quality over time.

Quote (alex):

"It seems really super clear that the fundamentals still matter, and part of that is realizing prompt engineering and prompt management is actually hard."

5. RAG and Structured Workflows Outperform Pure Autonomy

While fully autonomous "agents" get lots of attention, most companies find that structured workflows augmented by retrieval (RAG) and microservices offer better reliability and control. Setting up a reactive flow, with well-defined steps and conditional logic, prevents LLMs from going off track or triggering unintended actions. Restrictive or guided architectures also simplify debugging and maintain a consistent user experience.

Quote (hugo):

"It isn't like we've got these giant autonomous LLM brains as operating systems doing all types of stuff. The people who've been most successful embed LLMs in structured workflows with well-defined business logic."

6. Observability and Logging are Non-Negotiable

Teams repeatedly emphasize the critical nature of logging, tracing, and monitoring—especially for multi-step or agentic systems. Observability enables root-cause analysis when an LLM makes an incorrect call or "goes rogue." Solutions range from capturing every single prompt and model response to building specialized dashboards for real-time error detection. Having robust observability in place is essential for preventing small glitches from turning into major incidents.

Quote (alex):

"One thing they really suffered from was agents veering off course and getting lost, and backtracking isn't easy... Observability, logging, and tracing are super important to keep issues from spiraling."

7. Incremental Rollouts Limit Blast Radius

Large enterprises such as Thomson Reuters, BNY Mellon, and Alaska Airlines show that rolling out LLM features gradually is the safest path. Controlled testing with internal staff, then limited user groups, ensures problems are caught before they affect the entire user base. This systematic approach also gathers real feedback to refine both the model and the user experience. Feature flags and canary deployments from standard software practices translate well to LLM systems.

Quote (hugo):

"Never put a chatbot out there immediately... you do risk assessments, cost evaluations, control testing with internal users first, and then proceed to limited deployment."

8. Human-in-the-Loop Remains Critical

Even as LLMs become more advanced, human oversight is still a core strategy for ensuring correctness and handling edge cases. Teams in healthcare, banking, and other regulated industries find that partial automation plus a human review step strikes the right balance between efficiency and safety. Having experts validate, confirm, or correct outputs is a proven way to boost reliability—especially when mistakes have real legal or financial consequences.

Quote (alex):

“Anytime you have agents dealing with customer service or support bots, there’s almost always a human in the loop somewhere to handle edge cases and expert support.”

9. Simple Guardrails Can Go a Long Way

Structured output formats (like JSON schemas) and conservative fallback mechanisms radically improve LLM reliability. By forcing the model to adhere to specified formats, you reduce unpredictable behavior and streamline downstream integrations. When in doubt, it is often safer for an LLM to do nothing than to act incorrectly. These straightforward guardrails have minimal overhead yet boost overall stability.

Quote (alex):

“Having structured outputs and validation is pretty easy to do, minimal overhead, and it makes your system nicer.”

10. The Future of Multi-Agent Systems is Promising, but Not Immediate

Many hope that multi-agent setups will unlock advanced autonomous workflows, but real production use remains limited. Single-agent systems already pose reliability challenges; multiple agents introduce additional layers of complexity and error potential. Though research on multi-agent coordination and planning is ongoing, most organizations still benefit more from simpler, structured LLM integrations. This “crawl-walk-run” approach ensures safety and maintainability.

Quote (alex):

“Even single-agent systems are still a bit of a struggle... We’re only just starting to figure out how it all fits together.”

These insights, drawn from Alex and Hugo’s conversation, offer practical guidance for engineers building LLM-based applications—emphasizing tested software engineering practices, deliberate rollout strategies, and guarded adoption of emerging agentic capabilities.