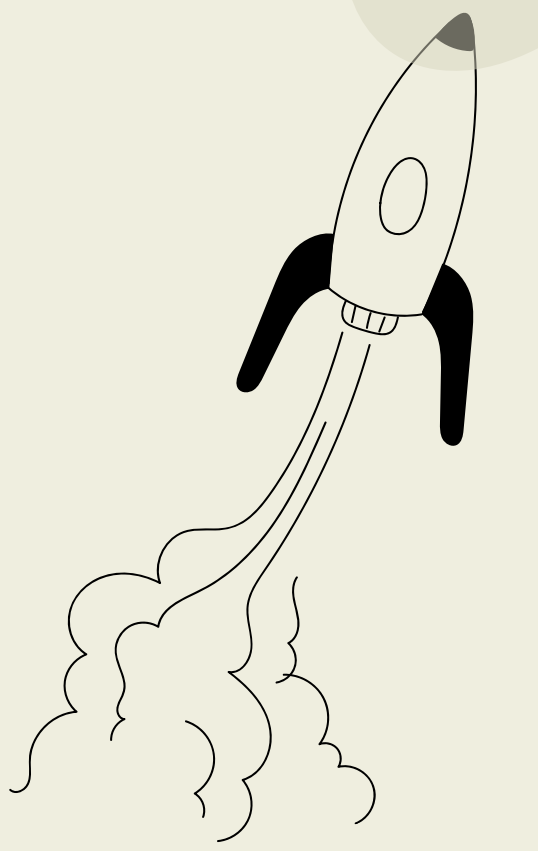
AI Agents Series

Beyond the Demo: Why LLM Agents Break in Production



Anantha Narayanan

Swipe Left

1. Runtime 

When working with agents, we make numerous LLM calls because we're using LLMs as reasoning engines. Every step and tool selection requires an LLM call where the model decides which tools to use.

These calls are sequential—each one waits for the previous call to complete. Depending on the task complexity and the number of reasoning steps required, this creates long-running applications with significant latency.

Potential solutions include implementing semantic caching and LLM response caching to reduce redundant calls.

2. Context Window Limitations

Each reasoning step requires sending large prompts to the LLM. While most modern models support context windows of around 128K tokens, complex applications can easily surpass this limit during multi-step reasoning processes.

This constrains the number of steps we can perform within a single context.

Although models like Jamba and Gemini support 1-2M tokens, working with such large contexts introduces other challenges, such as the 'lost in the middle' problem, where important information in the middle of the context gets overlooked by the model

Anantha Narayanan

Swipe Left

3. Hallucination Compounding

LLMs inherently tend to hallucinate due to their nature of predicting text word by word. Retrieval Augmented Generation (RAG) is an effective technique to reduce hallucinations by grounding the LLM with reliable context.

However, LLMs remain statistical in nature, with each response having a certain probability of being correct.

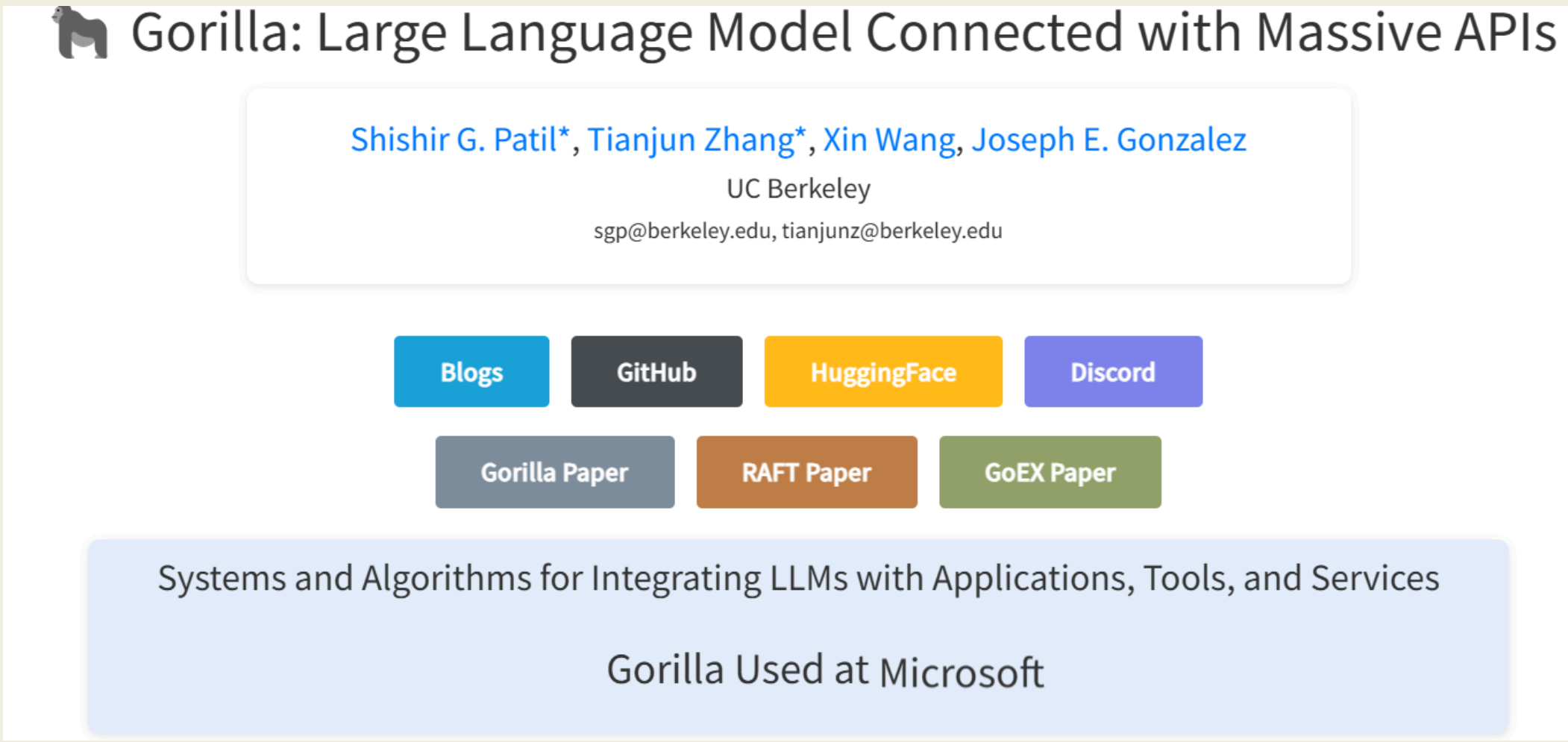
In agent systems, we make sequences of calls where each step depends on previous outputs.

This means probability errors compound—if each call has a 90% chance of being correct, after multiple steps, the overall probability of a fully correct process drops significantly

Anantha Narayanan

Swipe Left

4. Fine-Tuning



To address hallucination problems, we can implement fine-tuning strategies. By fine-tuning a model specifically for tool selection, we can significantly increase the probability of correct choices.

This approach transforms a model with perhaps a 90% chance of selecting the right tool into one with much higher reliability. Fine-tuning helps reduce the compounding errors that occur in multi-step reasoning processes, making the overall agent system more dependable

Anantha Narayanan

Swipe Left

5. Cost Considerations

LLM providers charge for both input and output tokens. When using agents with lengthy system prompts that must be sent repeatedly, costs can quickly escalate—especially at scale with millions of interactions.

Using advanced reasoning models like GPT-4o compounds this issue, as they're both slower and more expensive. This cost structure makes scaling agent systems financially challenging for many applications.

Strategic solutions include implementing robust semantic caching to reduce repeated LLM calls and carefully designing prompts to minimize token usage while maintaining effectiveness.

5. Response Validation Challenges

Each LLM call in our application requires validation before determining next steps—whether that's tool calling or presenting output to users. We need robust mechanisms to validate not just the correctness of responses but also their formatting.

While testing these validation systems can be complex, modern LLMs generally handle formatting requirements well when properly prompted. Nevertheless, implementing comprehensive validation remains an essential safeguard in production agent systems to prevent cascading errors downstream

7. Security Vulnerabilities

In agentic applications, we extend LLM capabilities by integrating with external APIs—running database queries or making calls to third-party services. This creates significant security risks.

If a malicious user hijacks our prompt through injection attacks, they could potentially gain access to these powerful tools and expose sensitive systems like databases.

Best practices to mitigate these risks include:

- Implementing the principle of least privilege—giving agents only the minimum permissions required
- Adding robust guardrails to filter and validate prompts before they reach the agent
- Using specialized tools like LLM Guard to detect and prevent prompt injection attacks

8. Avoiding Overengineering

Agents excel when dealing with undetermined sequences of steps that require dynamic decision-making. However, if you know exactly what needs to be executed in sequence, traditional code is often more efficient and reliable.

Don't fall into the trap of using agents when a straightforward, deterministic solution would work better. Implementing agents for tasks with predictable workflows creates unnecessary complexity, increased latency, and higher costs without adding real value.

Choose the right tool for the job, sometimes a simple Python script is the most robust solution

Follow me to stay updated on LLM Agent × RAG!




Anantha Narayanan