system design for AI AGENTS

Learn everything from scalability to speed

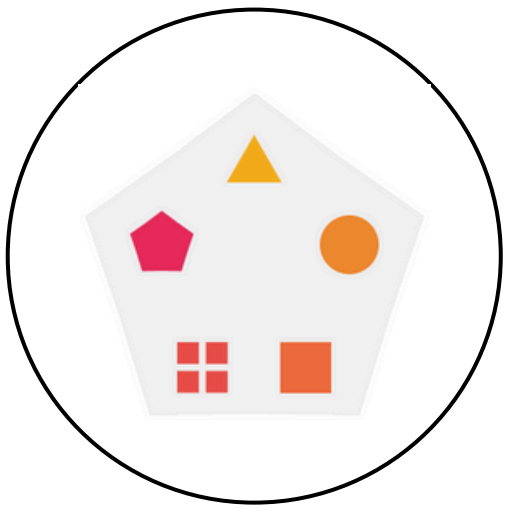
Bhavishya Pandit

Introduction

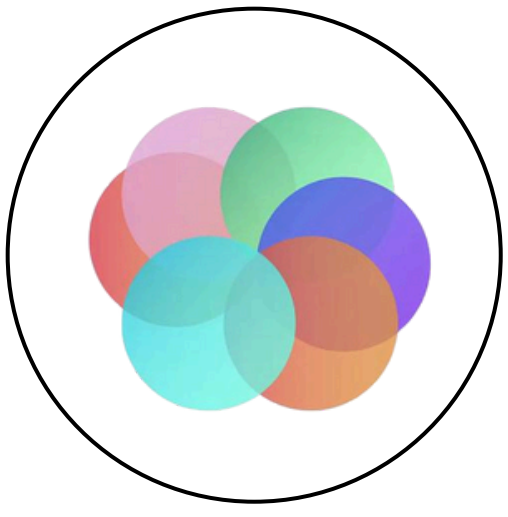
Did you know that **70% of AI agent projects fail in production.**

With so many Agentic systems shipped and launched people have forgot the basic principle of software engineering i.e System Design.

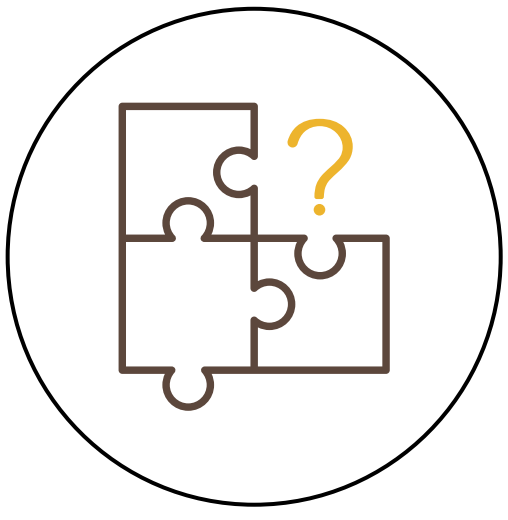
🚫 Why Most AI Agents Fail:



Monolithic Approach



Tightly coupled services



Missing Foundations

In this post we will unfold all the aspects of building a scalable agentic system that can handle traffic without breaking.

✅ What You'll Learn: This carousel breaks down the exact architectural patterns used by top engineering teams to achieve:

- **5,000+ tasks/min throughput**
- **90%+ accuracy**
- **True production reliability**

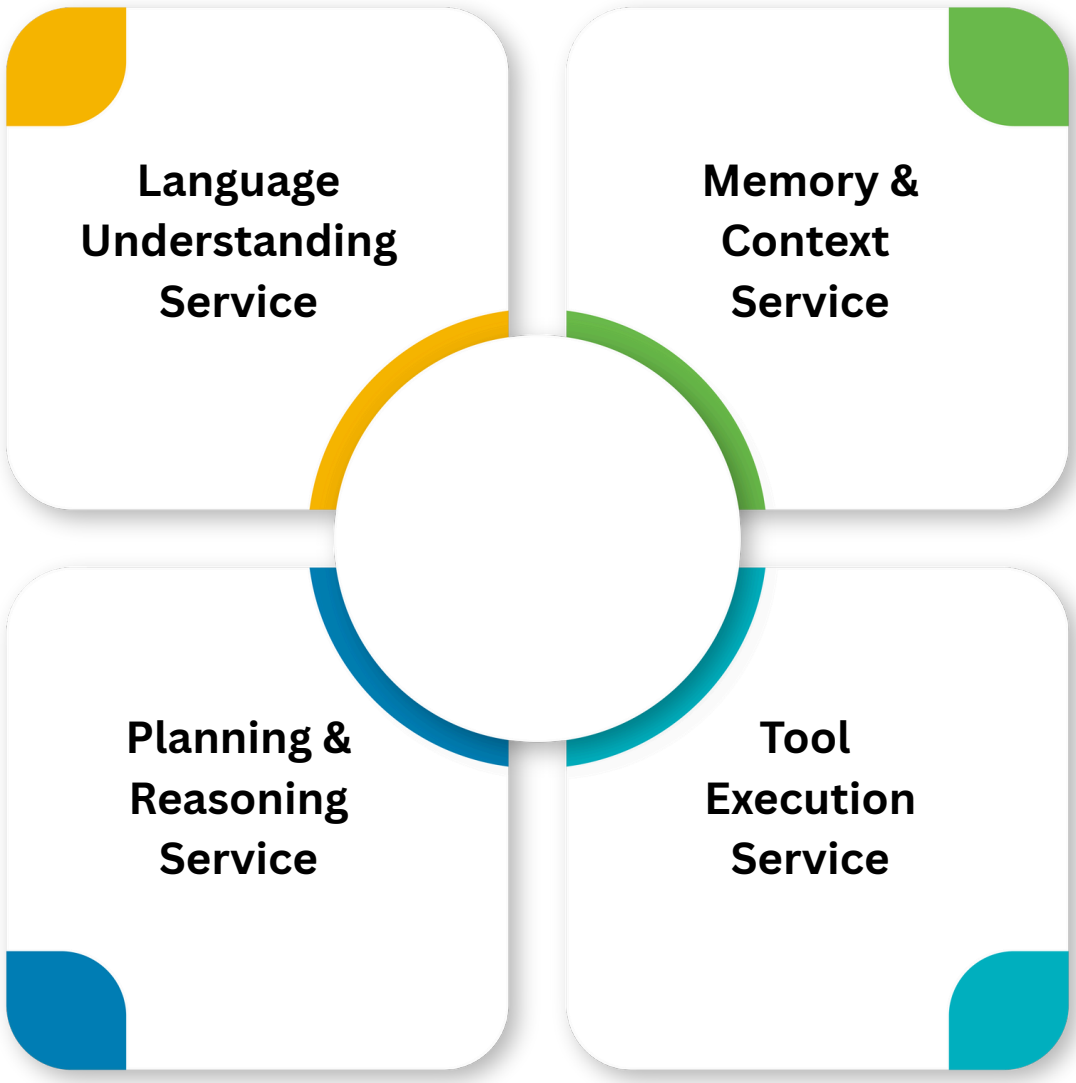
Let’s start -

The Foundation

Most teams build AI agents as a single, massive application. This creates a scaling, debugging, and update nightmare. A single bug in one capability can take your entire agent offline.

The Solution:

Decompose your agent into a suite of specialized microservices. Each core capability operates as an independent, loosely-coupled service communicating via clean APIs.



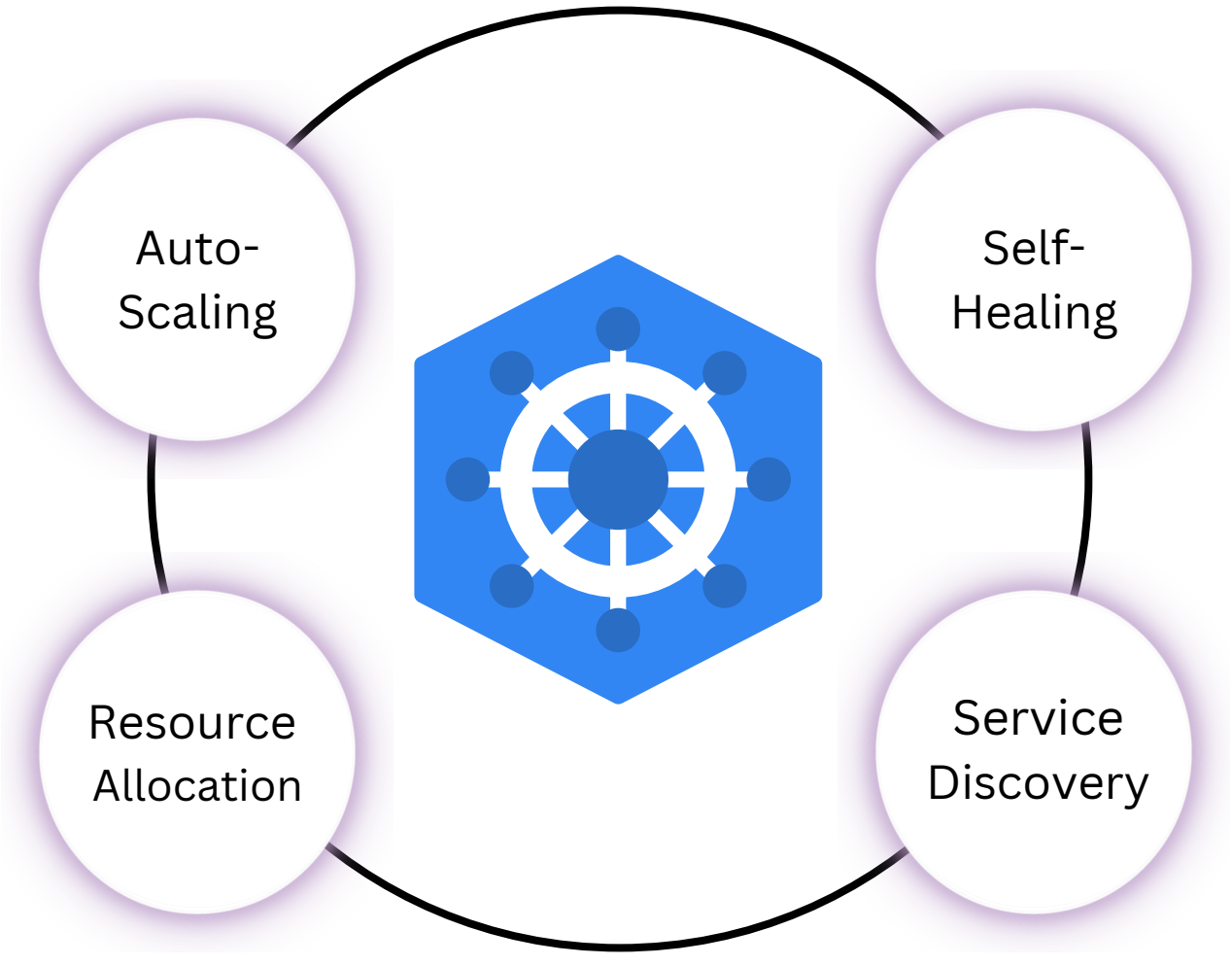
Netflix's recommendation engine works on the same principle, It's a symphony of microservices for content analysis, user modeling, and A/B testing.

Now that we have a suite of independent services, how do we manage and deploy them at scale?

Container Orchestration

AI agents have complex dependencies - specific Python versions, ML libraries, GPU drivers. Containers solve the "it works on my machine" problem by packaging everything together.

For microservices to operate at scale, you need an orchestrator. Kubernetes automates the entire deployment and management lifecycle:



The Three-Layer Evolution:

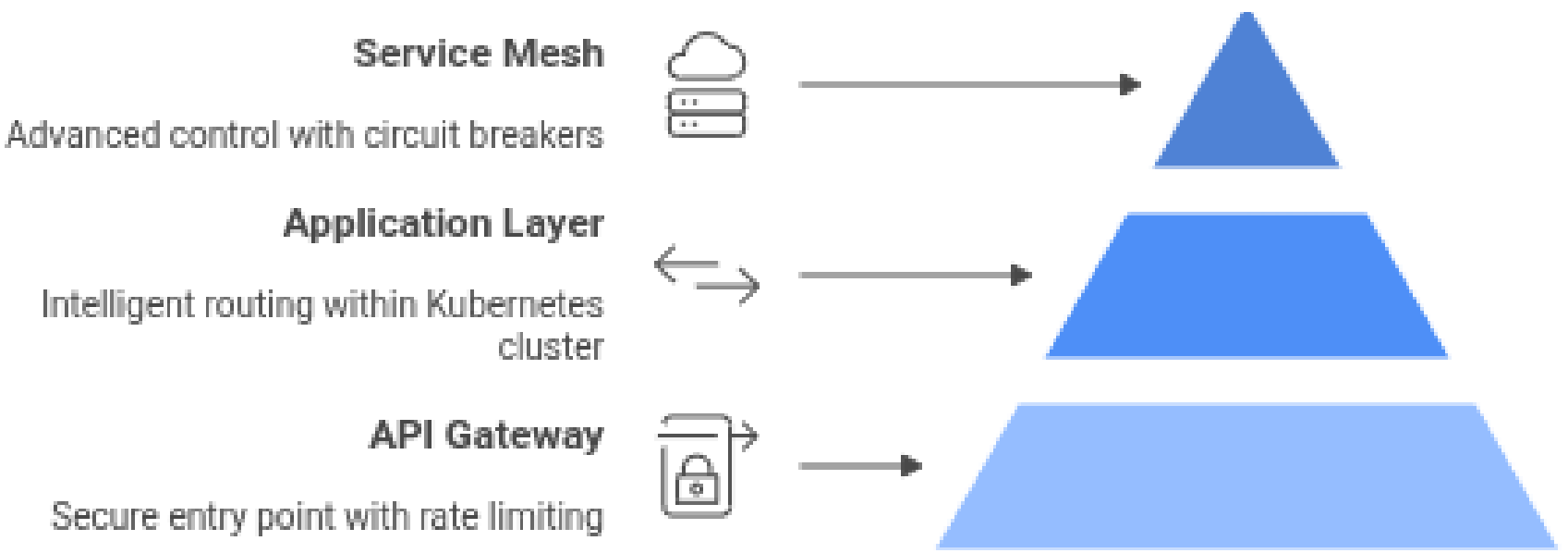
- Prototype: A single containerized agent.
- Staging: Multi-agent coordination on a single machine.
- Production: Full orchestration across a cluster of machines.

Our services are now running. But how do we handle real-world, spiky traffic without them collapsing?

Load Balancing

Production traffic is never steady. Your system must seamlessly handle **50 requests per minute** and scale instantly to **5,000+** without breaking a sweat. This requires intelligent traffic distribution.

A Multi-Layer Strategy for Resilience:



1. API Gateway -

- Global Rate Limiting
- Authentication & SSL Termination
- Initial Routing

2. Application Layer - Within your Kubernetes cluster, the load balancer intelligently routes requests:

- Dynamically sends traffic to the least busy agent instance.
- Directs traffic based on the compute capacity of each instance (e.g., GPU vs. CPU).

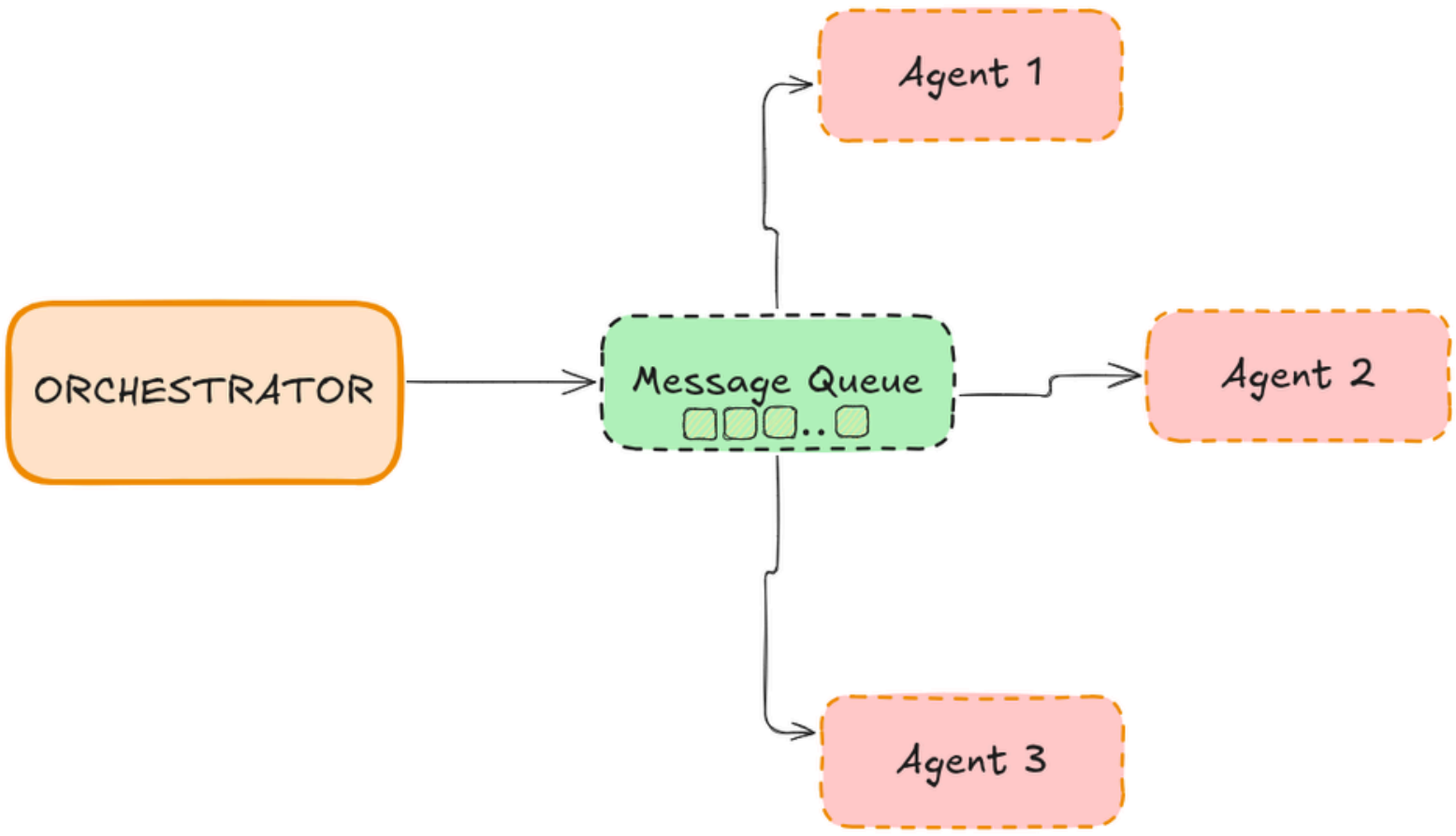
3. Service Mesh - Automatically isolate failing services to prevent cascade failures.

We can handle incoming requests, but what about tasks that take a long time? We can't keep the user waiting.

Message Queues

AI agents manage workflows that take milliseconds to minutes. Blocking requests for these tasks creates bottlenecks and system-wide failures. The solution is asynchronous, non-blocking communication.

Message Queues decouple your microservices, allowing them to communicate via messages instead of direct API calls. This ensures a failure in one service doesn't cascade and that no task is ever lost.



Key Patterns in Action:

- Task Queues: Distribute long-running tasks to specialized worker agents.
- Dead Letter Queues: Automatically capture and retry failed messages for debugging.
- Event Sourcing: Maintain a full, auditable log of every agent decision and action.

Our agent is resilient and scalable. But to be useful, it needs memory and context. How do we manage that data at scale?

Memory Layer that Scales

An AI agent without memory is a stateless, expensive API call. Production agents must maintain context, learn from history, and instantly access vast knowledge to be truly useful.

Smart systems use a layered approach, selecting the right database for each type of memory:

Short-Term Context (Redis)

Long-Term Knowledge (Vector DB)

Structured Data (SQL/NoSQL)

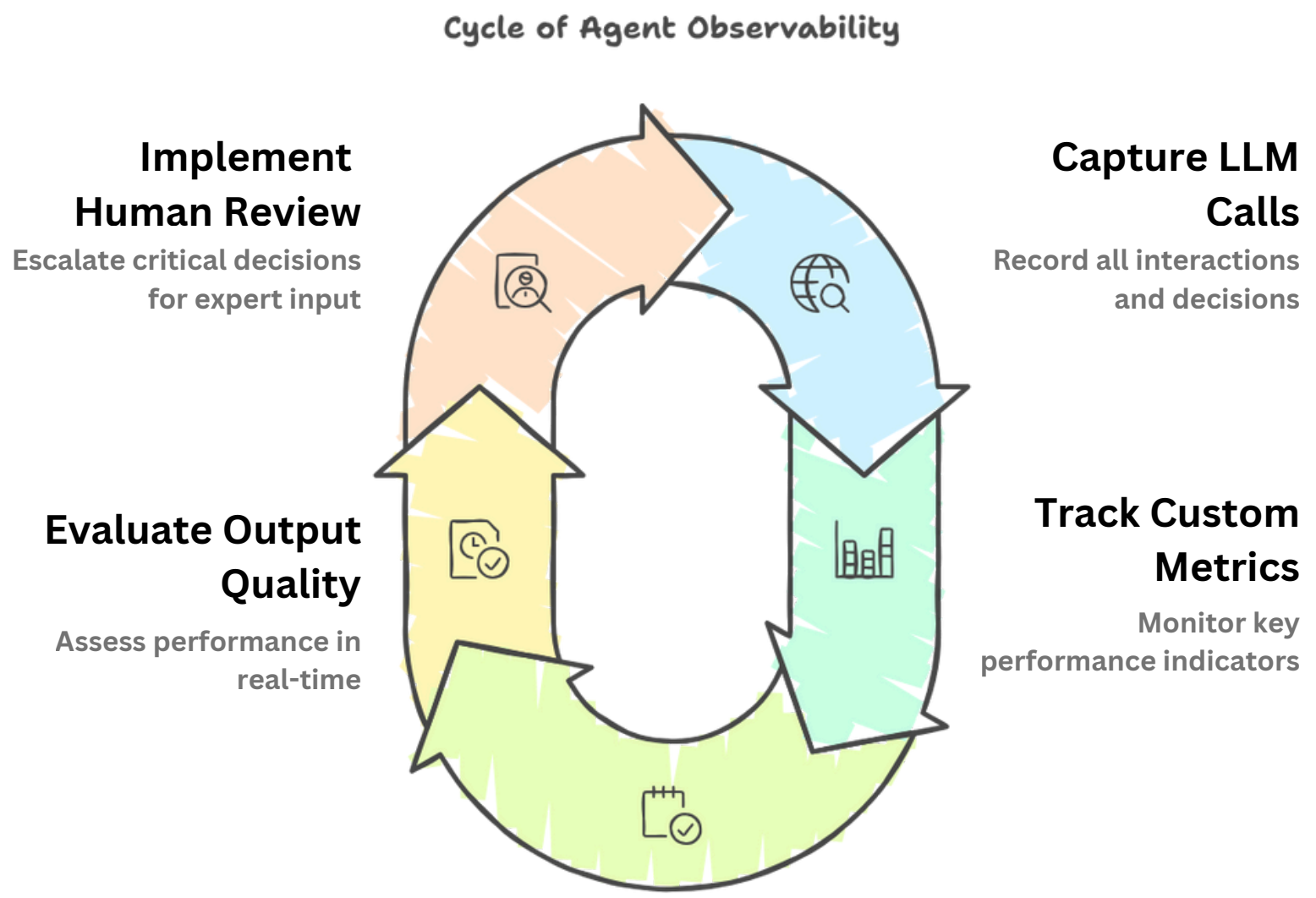
The Vector Database Landscape: **Performance & Fit**

- Pinecone (Managed Solution): **~50,000 QPS.**
    - Ideal for teams wanting a fully-managed service
- Weaviate (Multi-Modal Expert):**~10-15,000 QPS.**
    - The best choice for multi-modal agents.
- Qdrant (Performance Oriented):
    - Highest Raw Speed. **Delivers 4x RPS** in benchmarks for cost-conscious teams with deep technical expertise.

Our system is now built and running. But complex systems fail in complex ways. How can we see what's happening inside?

Observability

AI agents are inherently complex - they make decisions, call multiple services, and process unstructured data. Without proper observability, debugging production issues becomes nearly impossible.



TOOLS THAT WORK IN PRODUCTION:

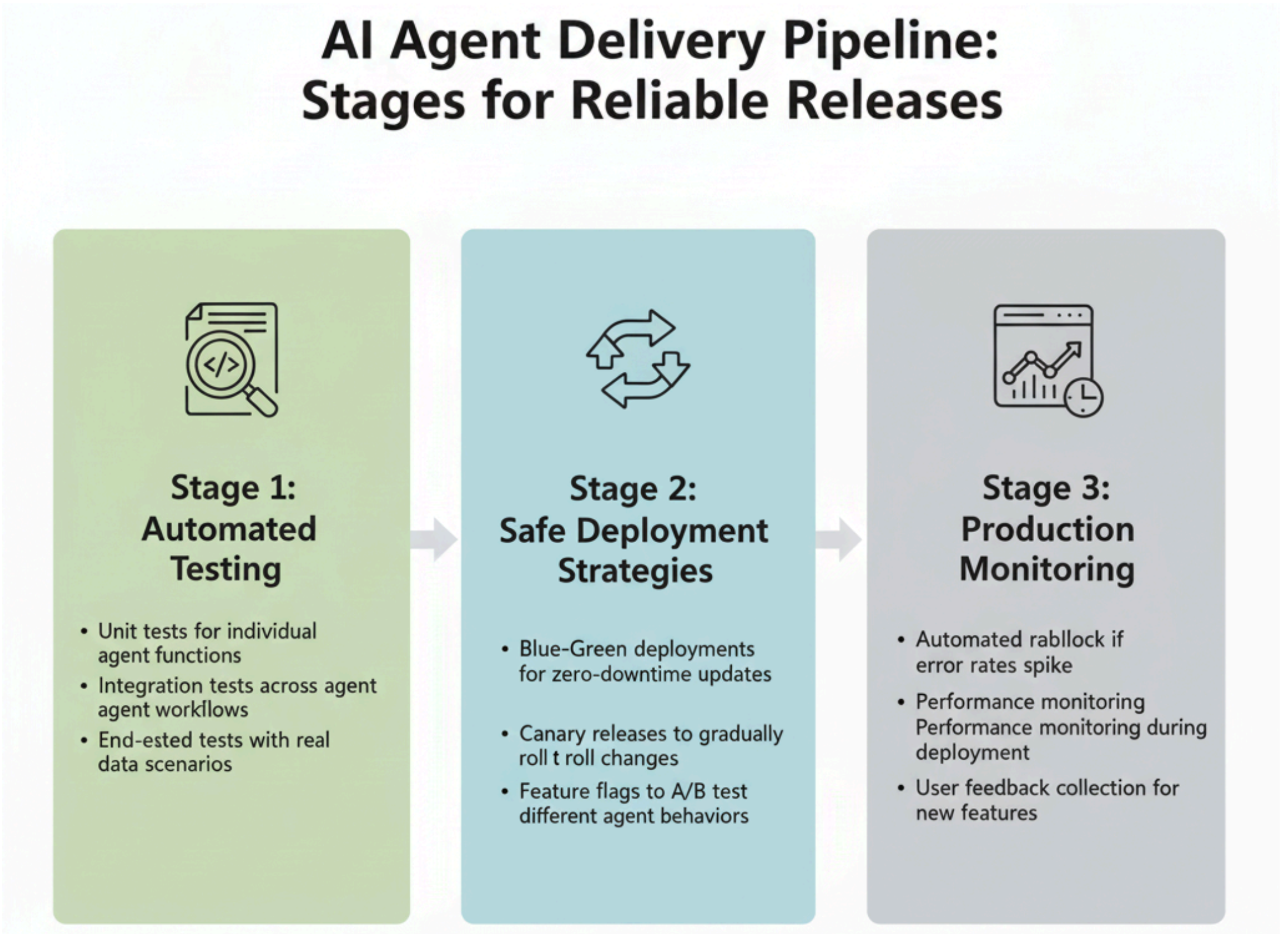
Langsmith **Azure AI Foundry** **AgentOPs**

Finally, our agent isn't static. It will evolve. How do we update it safely without breaking our production environment?

CI/CD

This is the closing piece of the puzzle. It's how you continuously deliver value and improvements to your now-stable, scalable platform. It ensures the system remains reliable as it changes.

The Three-Stage Pipeline:



This is the blueprint: from a fragile monolith to a resilient, scalable, and self-improving AI system. This is how you cross the chasm from demo to production.

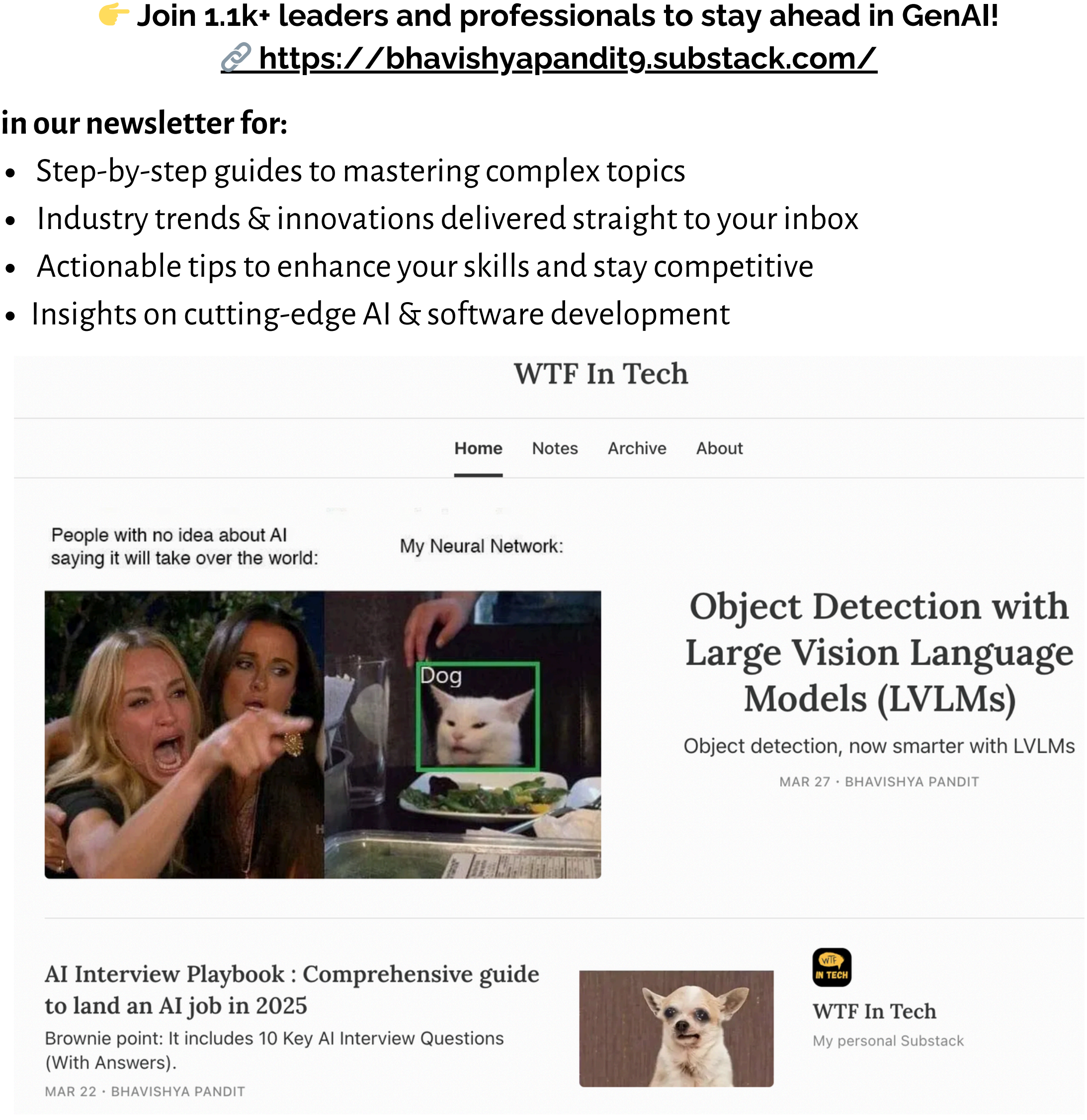
Stay Ahead with Our Tech Newsletter! 🚀

👉 **Join 1.1k+ leaders and professionals to stay ahead in GenAI!**

🔗 **https://bhavishyapandit9.substack.com/**

Join our newsletter for:

- Step-by-step guides to mastering complex topics
- Industry trends & innovations delivered straight to your inbox
- Actionable tips to enhance your skills and stay competitive
- Insights on cutting-edge AI & software development



💡 Whether you're a developer, researcher, or tech enthusiast, this newsletter is your shortcut to staying informed and ahead of the curve.

Bhavishya Pandit

Follow to stay updated on Generative AI

LIKE

COMMENT

REPOST