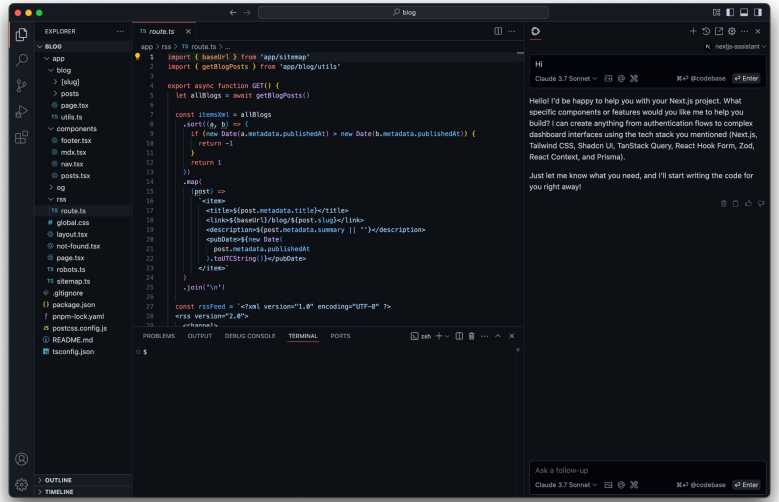
An open architecture for AI-native development

Ty Dunn, Co-founder of Continue

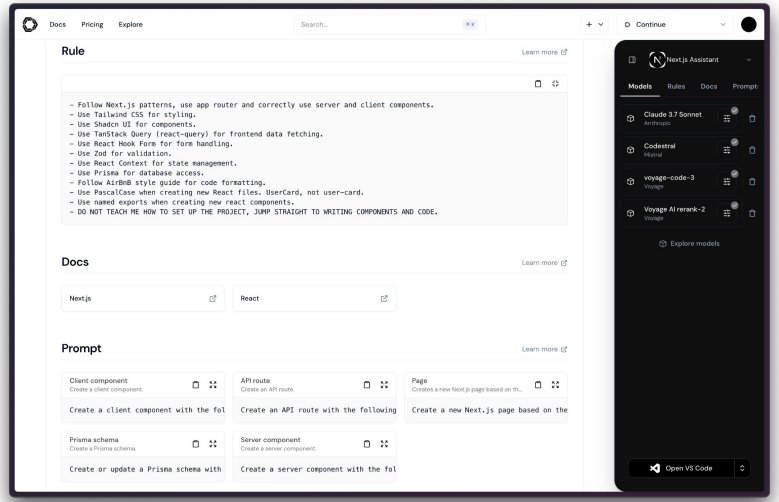
Goals for this talk

1. Describe one of many possible open architectures for AI-native development
2. Inspire you to think more deeply about the architecture underlying the future of software development

What is Continue?

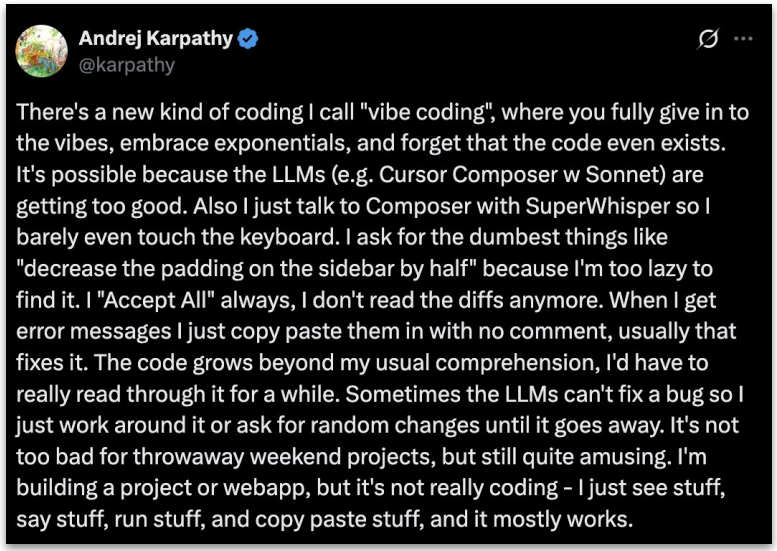


Open-source IDE extensions



Hub of rules, tools, and models to create, share, and use custom AI code assistants

What is AI-native development?

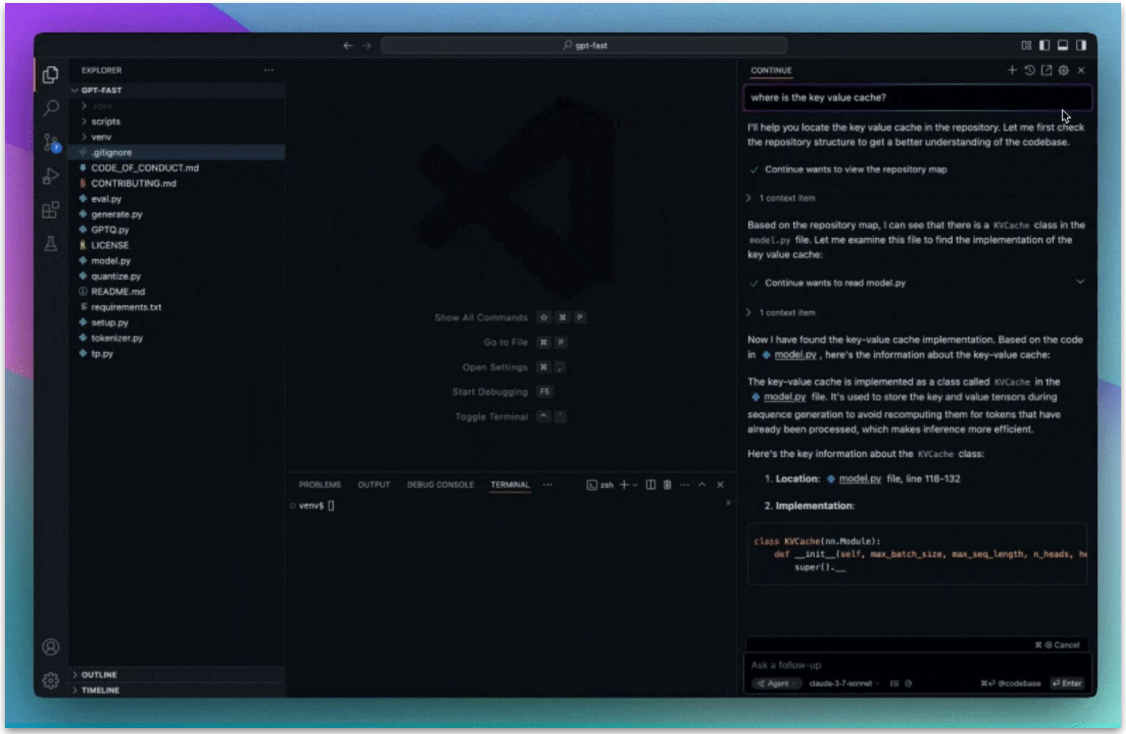


Vibe coding

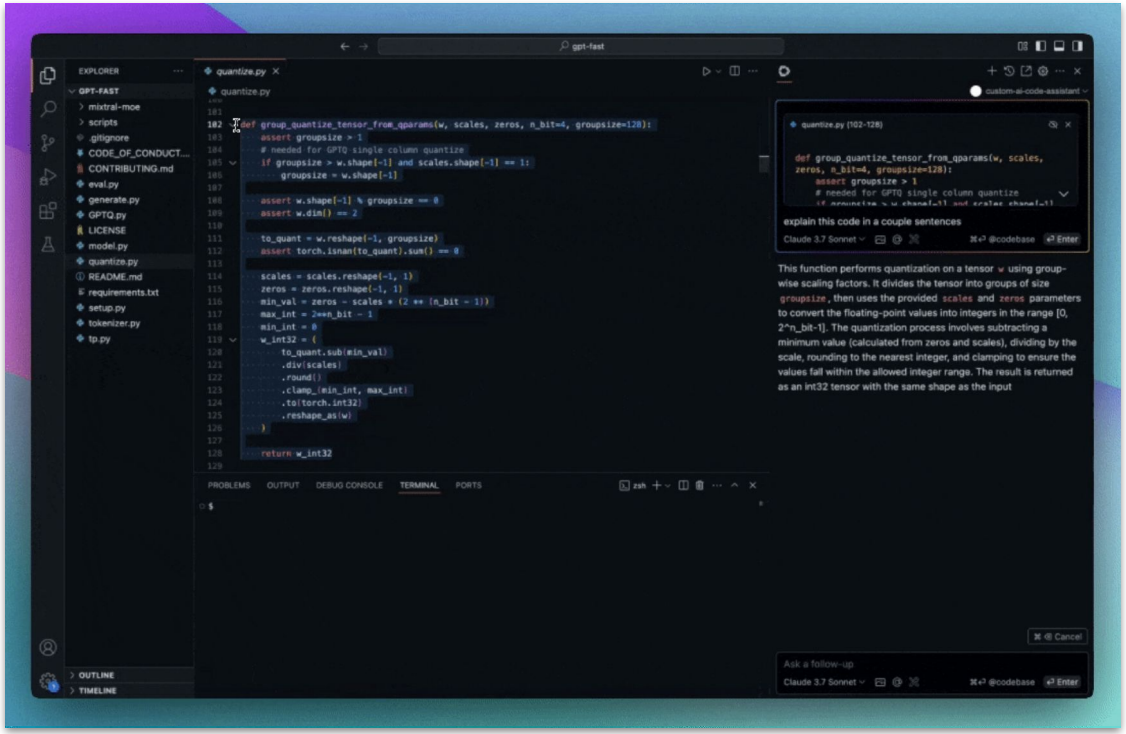
AI-native development

An example AI-native development workflow

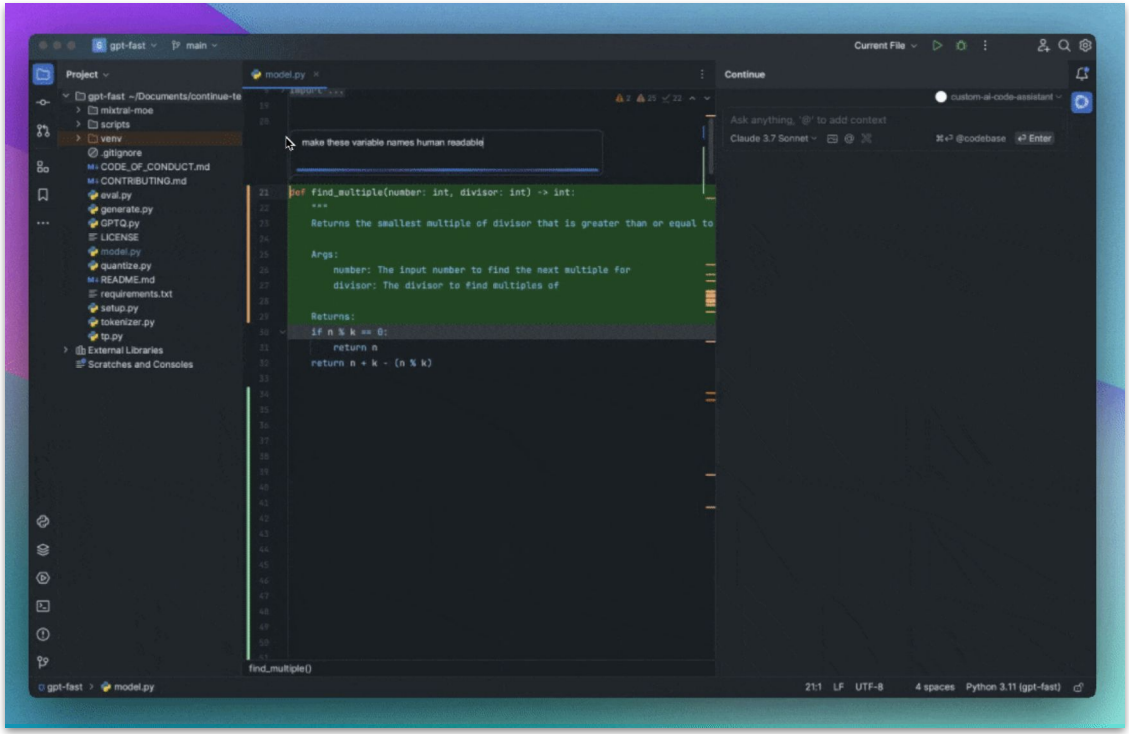
Agent to ask a model to do a task for you



Chat to understand and iterate on code with a model

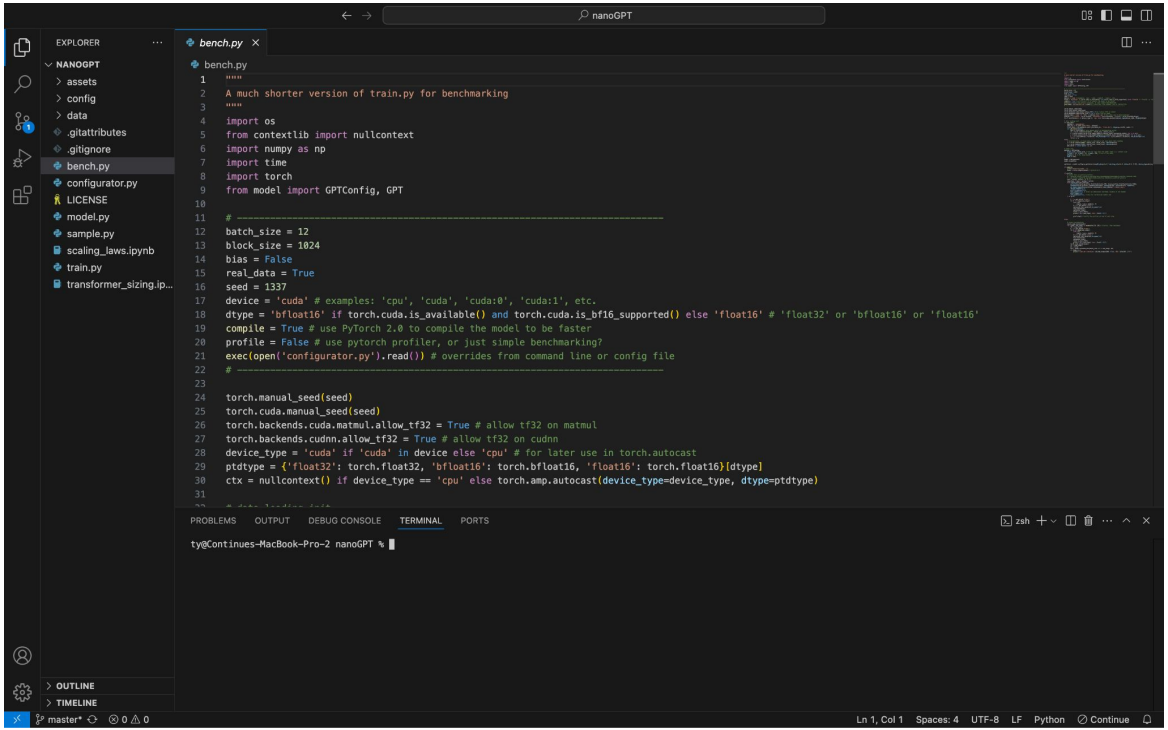


Edit to transform a section of code using a model



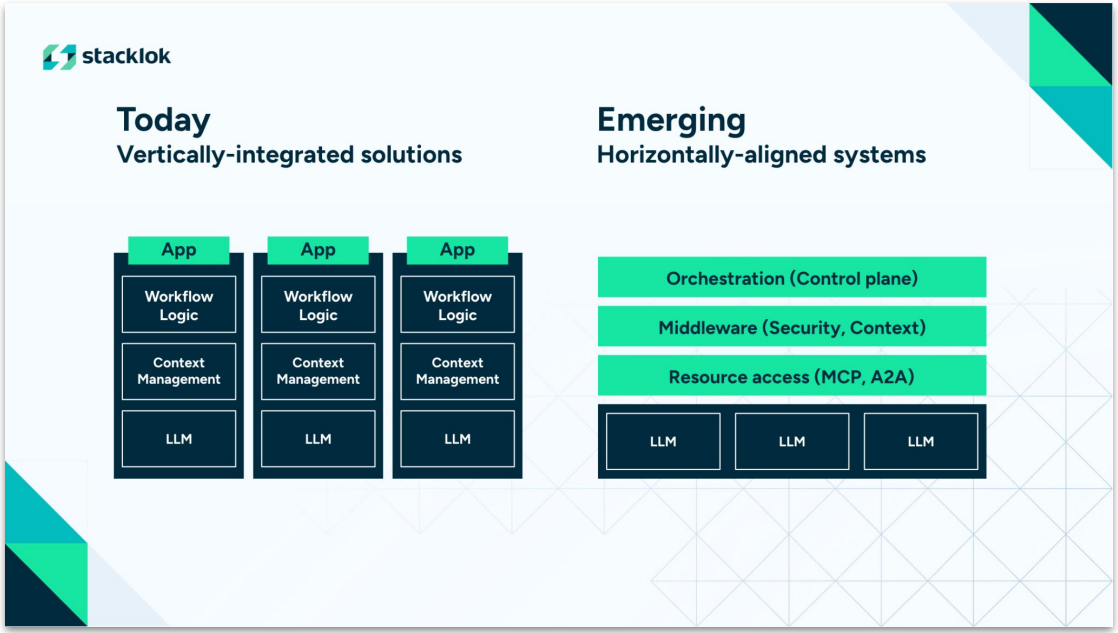
Autocomplete to have a model finish lines of code

Write the code manually without using a model



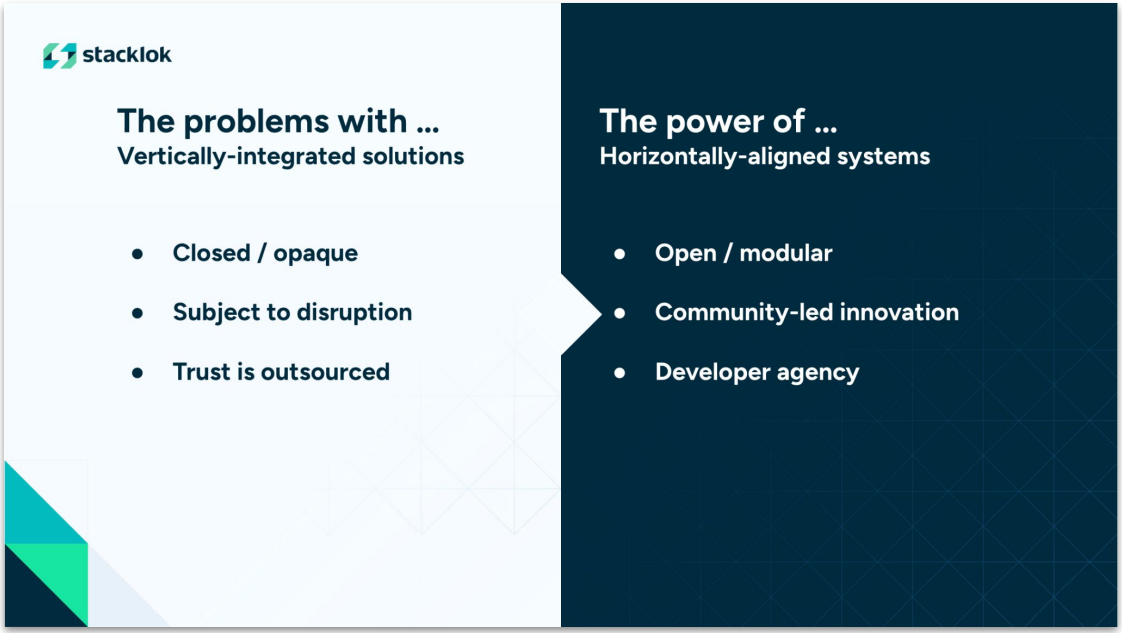
What is an AI-native open architecture?

An open architecture is a horizontally-aligned system



Slide from "Tipping AI Code Generation on its Side" at the O'Reilly AI Codecon, presented by Craig McLuckie, co-founder of Stacklok, Kubernetes, and CNCF

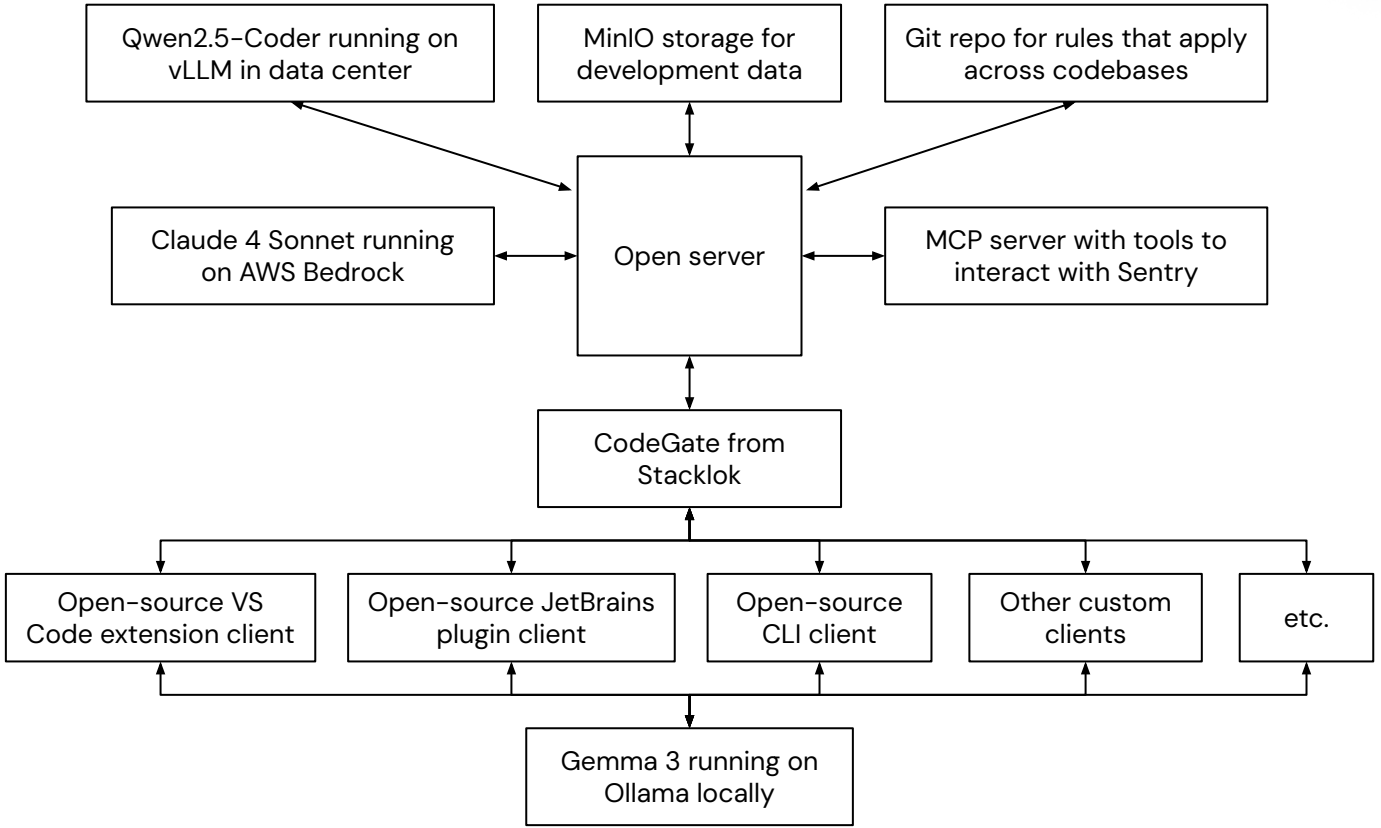
An open architecture is a horizontally-aligned system



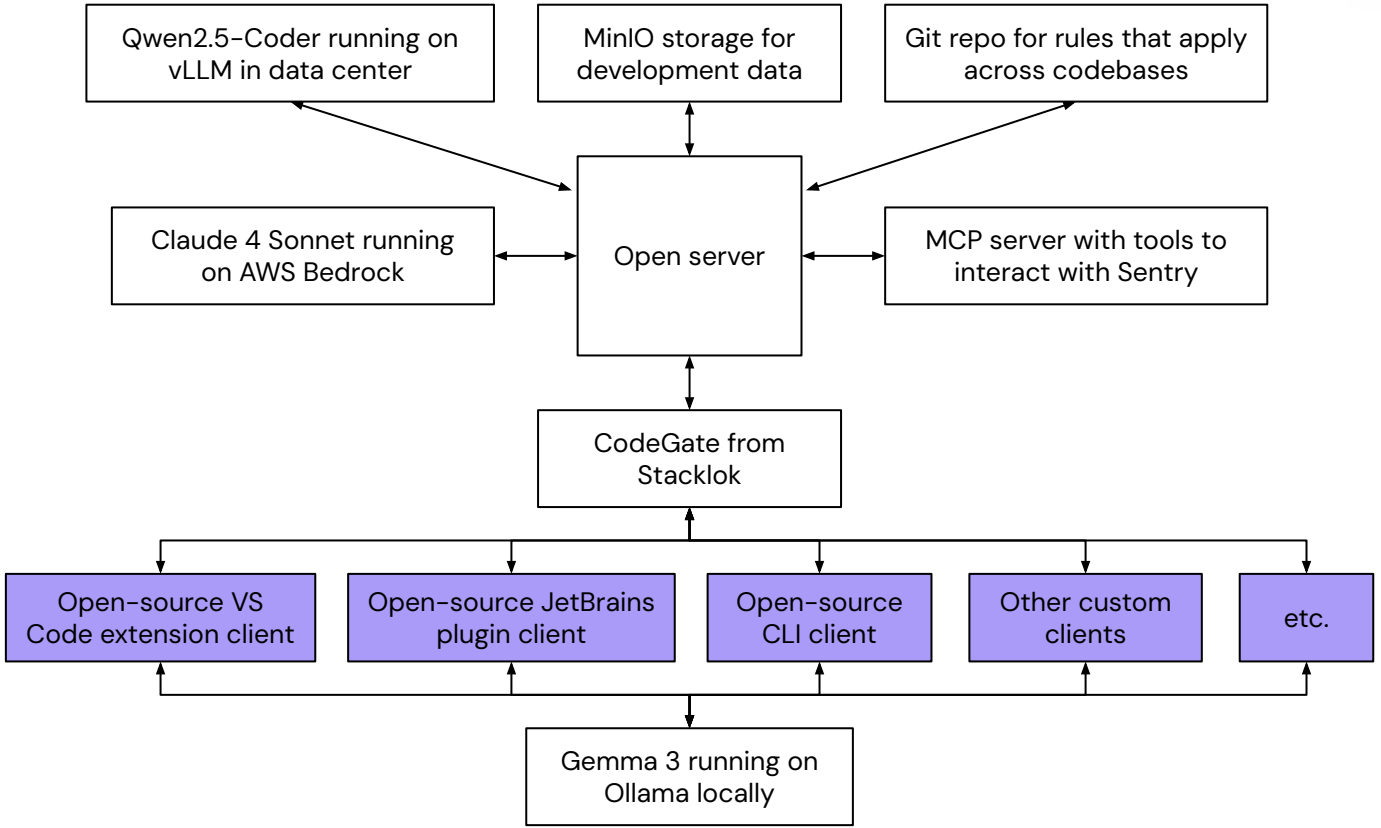
Slide from "Tipping AI Code Generation on its Side" at the O'Reilly AI Codecon, presented by Craig McLuckie, co-founder of Stacklok, Kubernetes, and CNCF

Overview of an AI-native open architecture

An AI-native open architecture



An AI-native open architecture: Clients



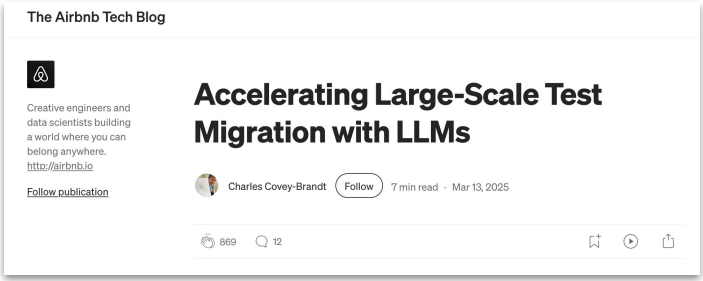
Open-source clients

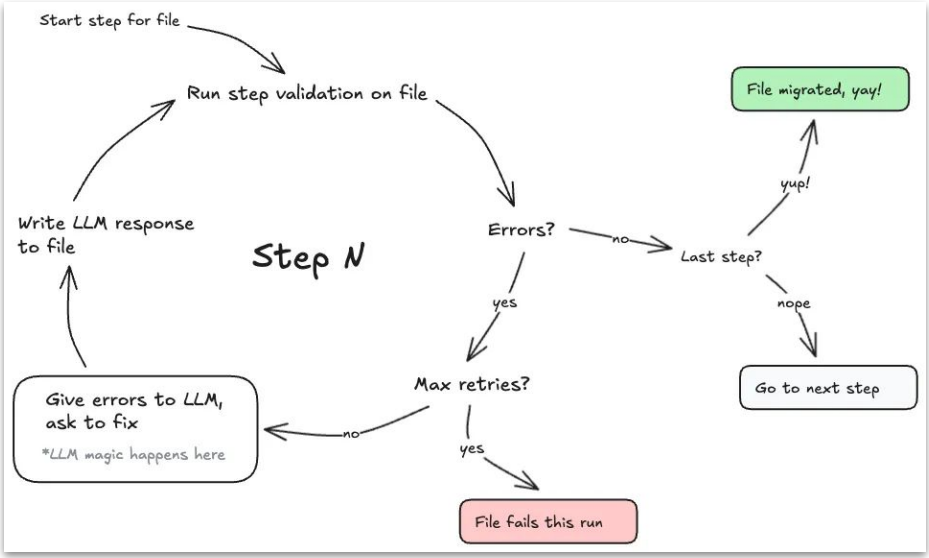
- Examples
  - VS Code extension, JetBrains plugin, CLI tool
- Benefits
  - You understand how it works and can learn to use it better
  - Empower an ecosystem to build on top of and around them
  - Enables developer to keep and leverage their development data

Custom clients

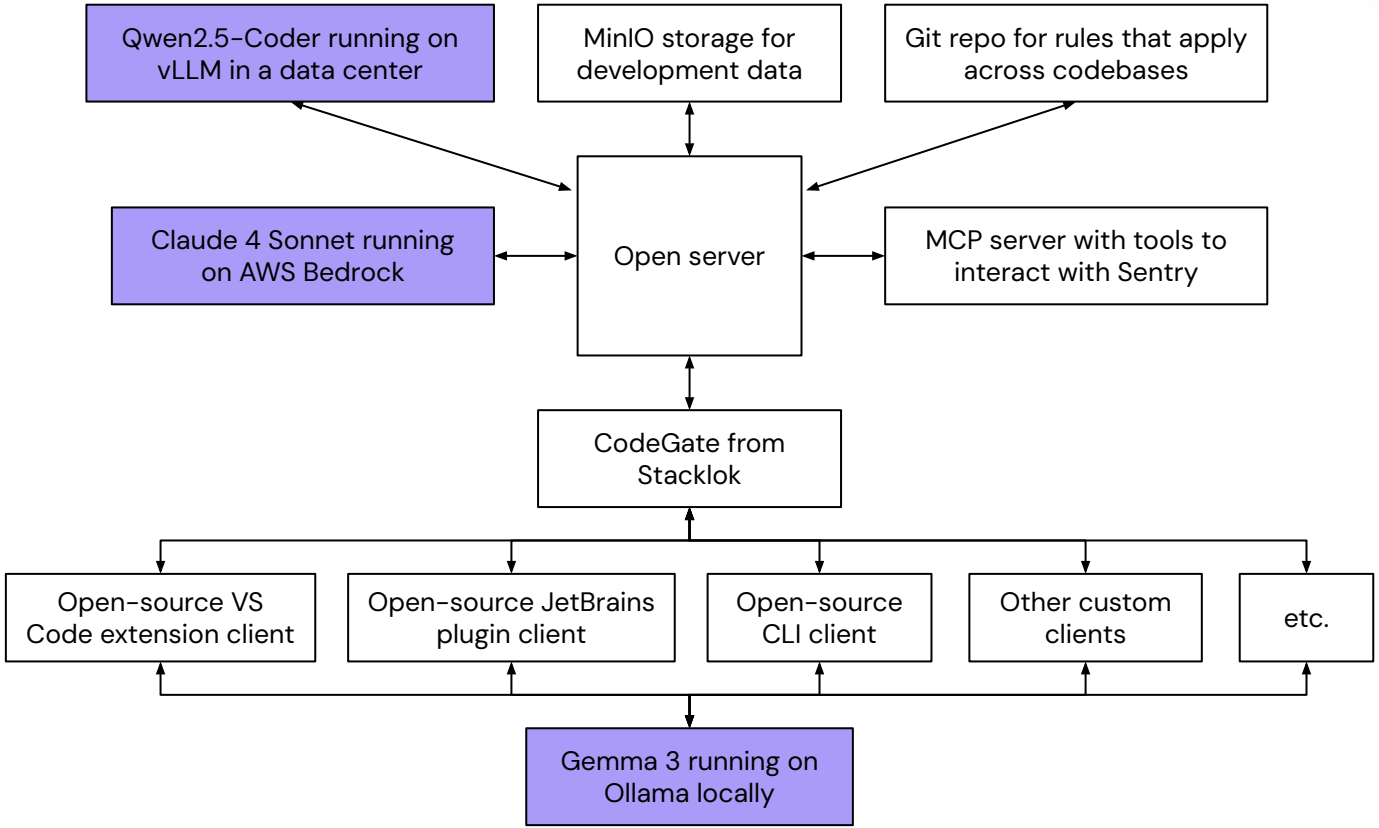
SDK / API that lets you pull in rules, models, tools into automation anywhere

For example, automation known as “agents” (rules, tools, and models in a loop)





An AI-native open architecture: Models



Models

Models are the intelligence that follows the rules using the tools to automate tasks. They provide the reasoning and tool use capabilities needed to implement automation.

Key Model Trends

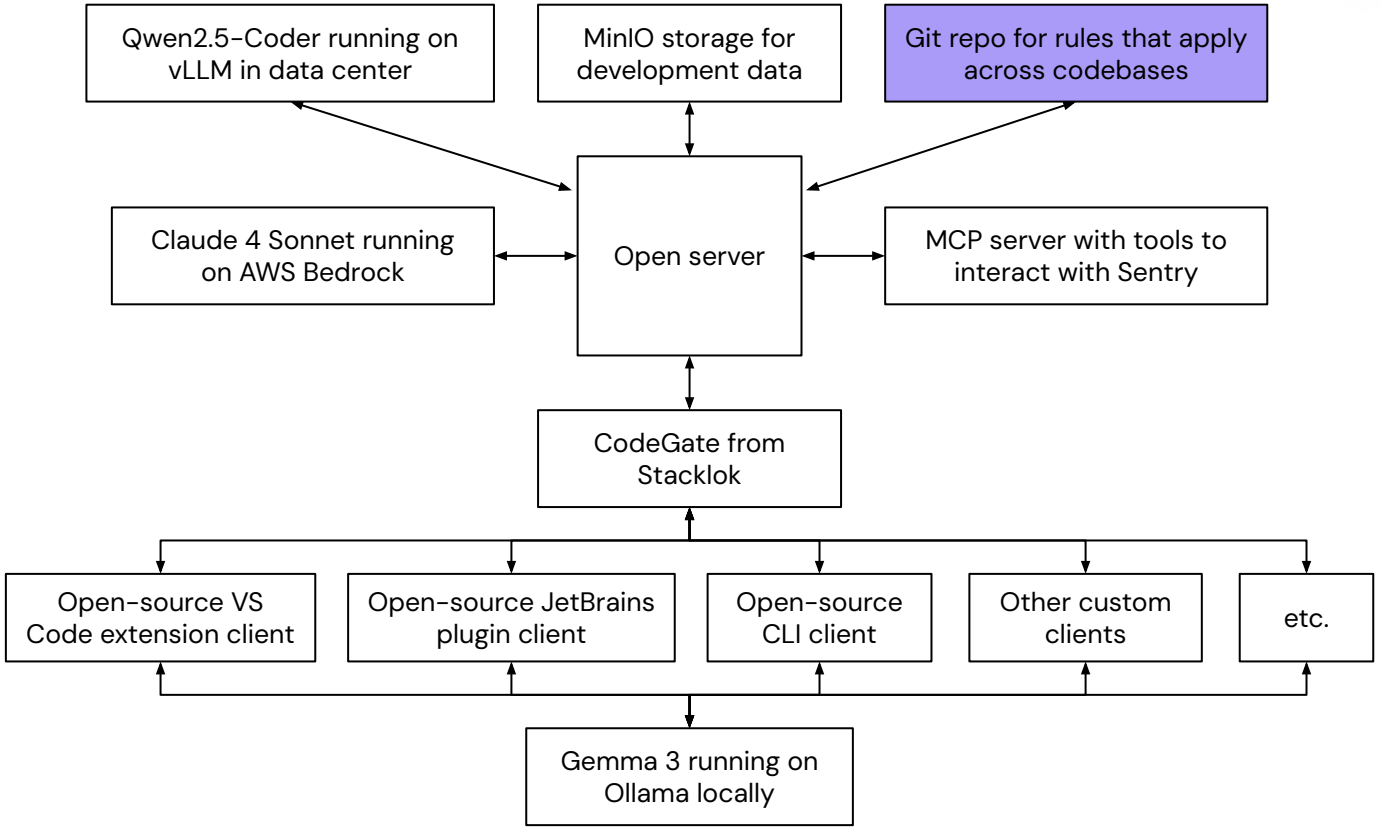
1. Increasing Capabilities
2. More capable open models
3. Improved economics
4. Diversifying task requirements
5. Growing domain specialization

Models distributed across compute environments

Specialized model deployments based on security, cost, and performance needs:

- **Local Development**: Efficient models running directly on developer machines for low-latency interactions
- **Enterprise Self-Hosting**: Support for organizations to fine-tune and deploy models behind their firewalls
- **VPC Deployments**: Secure cloud deployments within customer VPCs to maintain data sovereignty

An AI-native open architecture: Rules



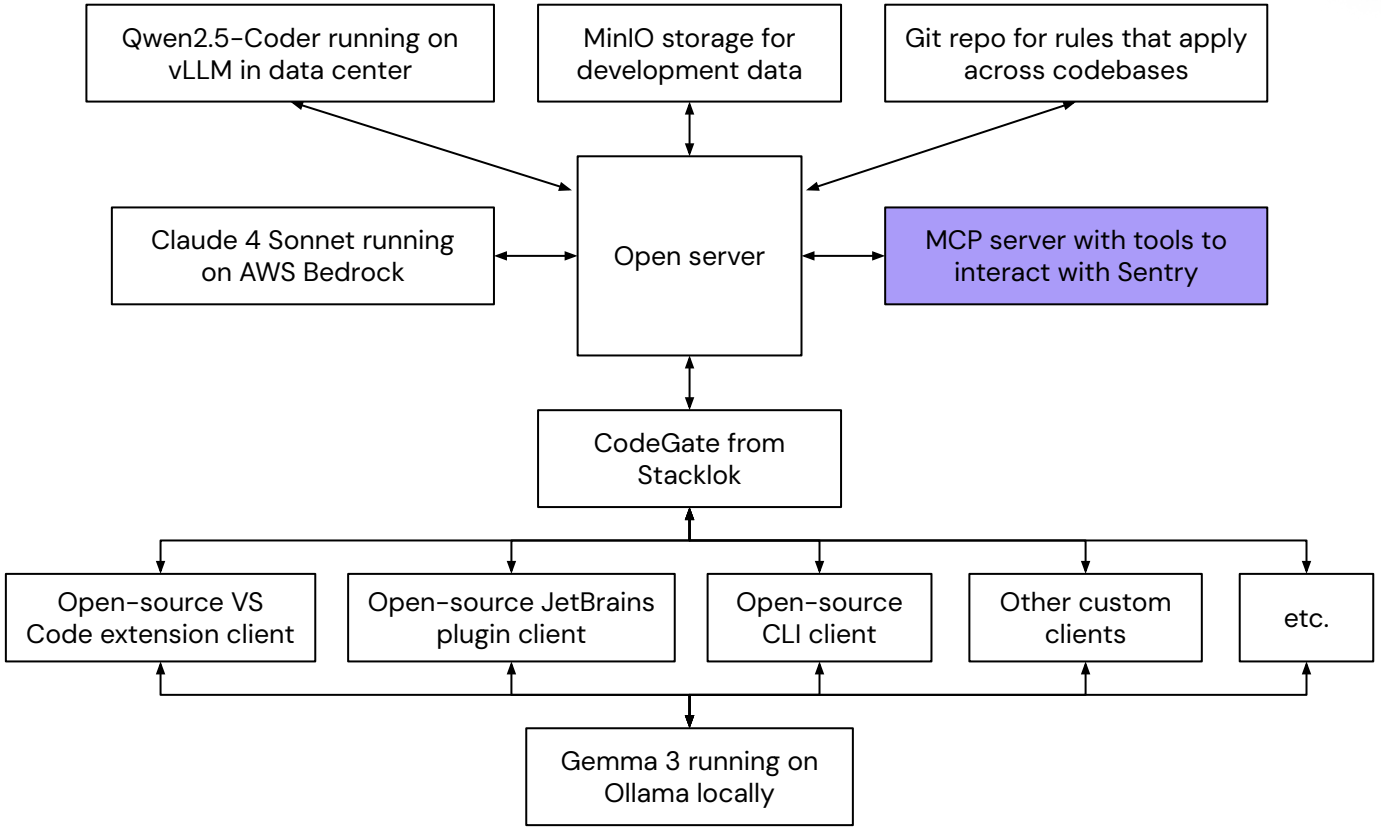
Rules

Rules are used to describe the **intents, constraints, policies, specifications, etc.** within an organization

Collectively, they create comprehensive natural language descriptions that **make how you build software legible (i.e. rules) and accessible (i.e. tools) to models**

Some rules stored with codebases in Git, others that cut across codebases are stored in another centralized place (e.g. a Git repository)

An AI-native open architecture: Tools



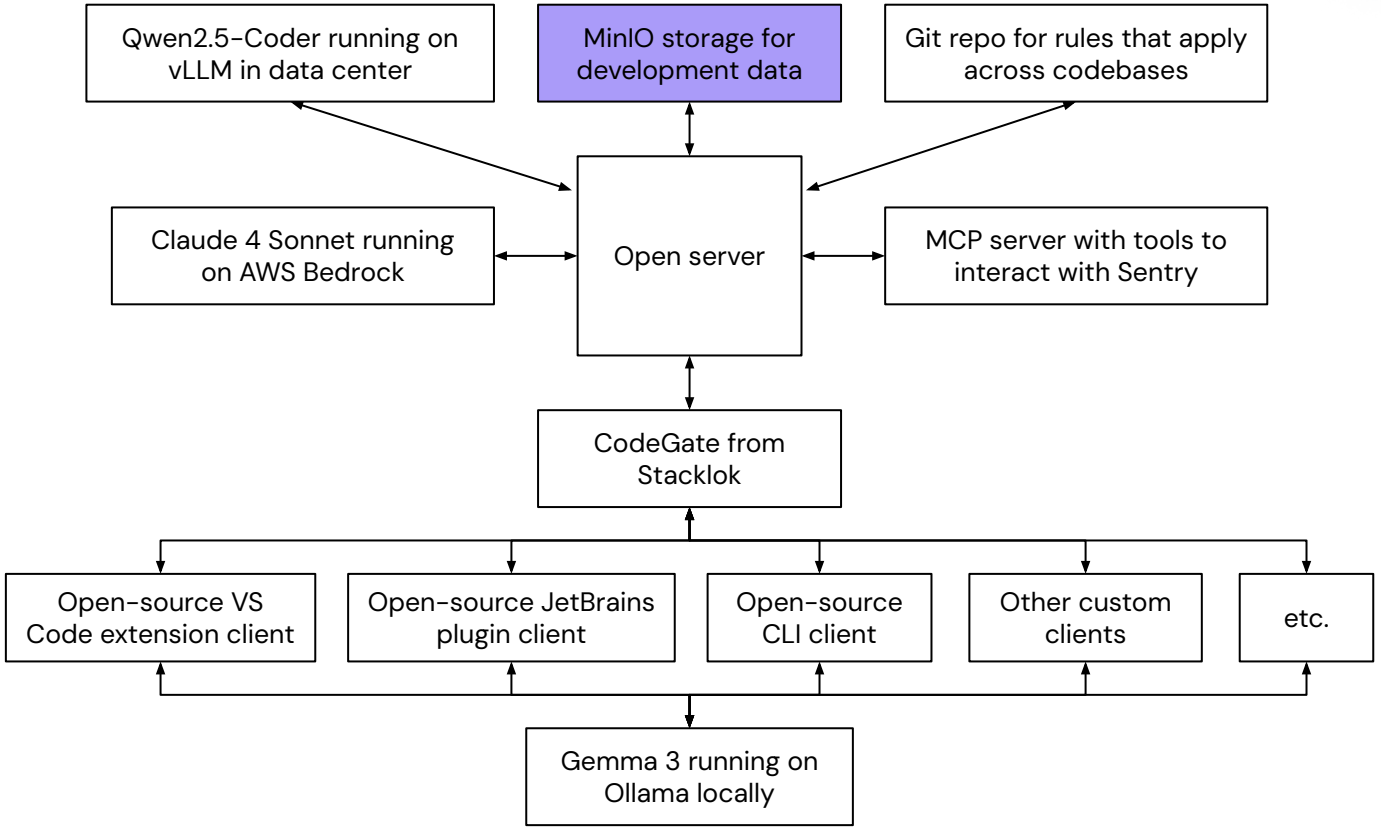
Tools

Tools are used to retrieve information in real-time and take action, enabling models to interact with systems and data

1. “Runtime” Context: retrieving tickets from Linear/Jira/GitHub Issues, retrieving documentation from Confluence/Notion/wikis, etc.

2. Actions: running tests to validate code changes, updating database schemas based on code modifications, editing files in development environments, etc.

An AI-native open architecture: Development data



Development data

Dev data captures **the "how" of development that exists between Git commits** and is automatically generated when you use models while coding

The **step-by-step process of software creation** that shows the

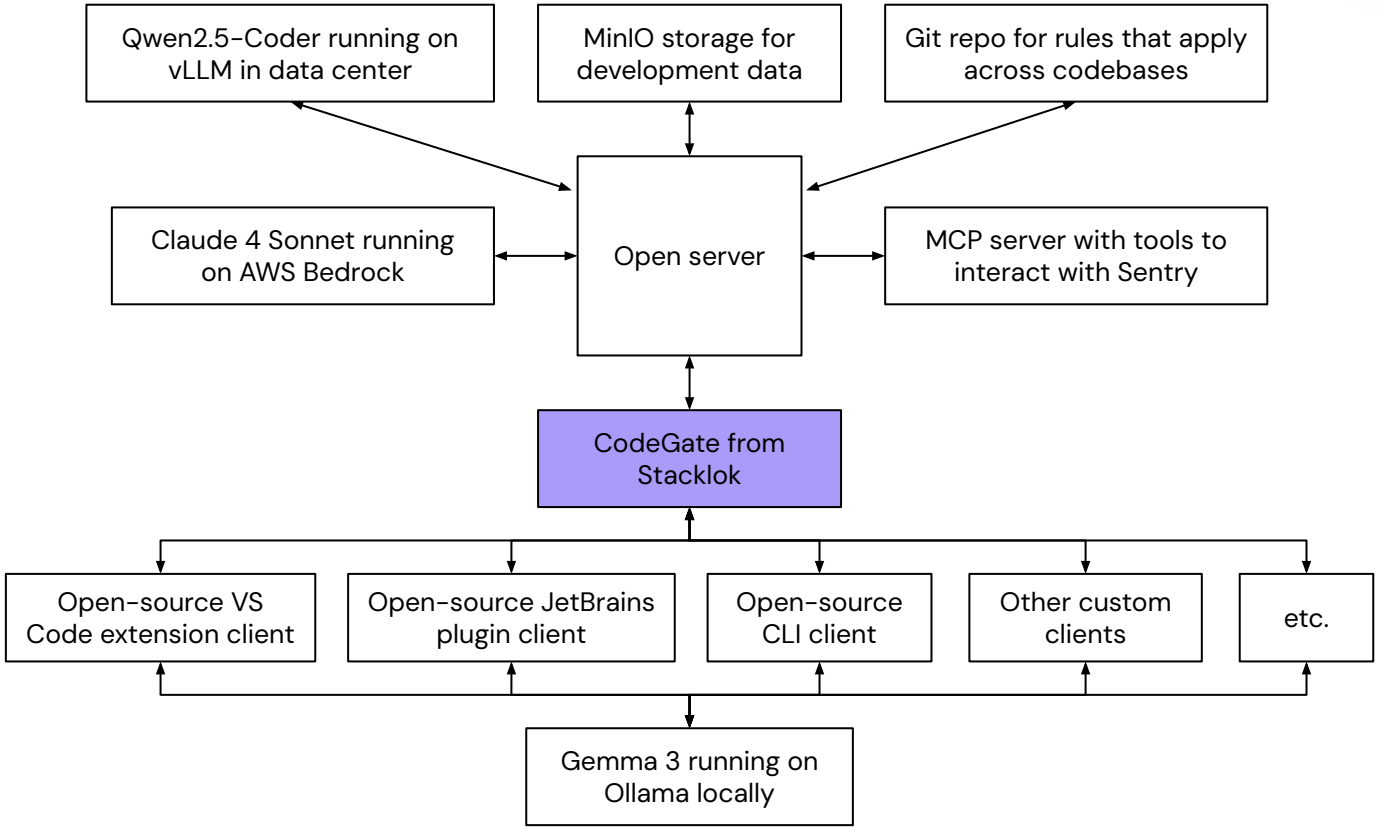
- patterns that make your organization unique
- context used to make development decisions
- reasoning behind each implementation choice

Development data

You can use dev data to **transform implicit, tribal knowledge into explicit, collective assets** that can be used to further automate development with AI

That is, development data can be used to improve rules, tools, and models

An AI-native open architecture: Ecosystem



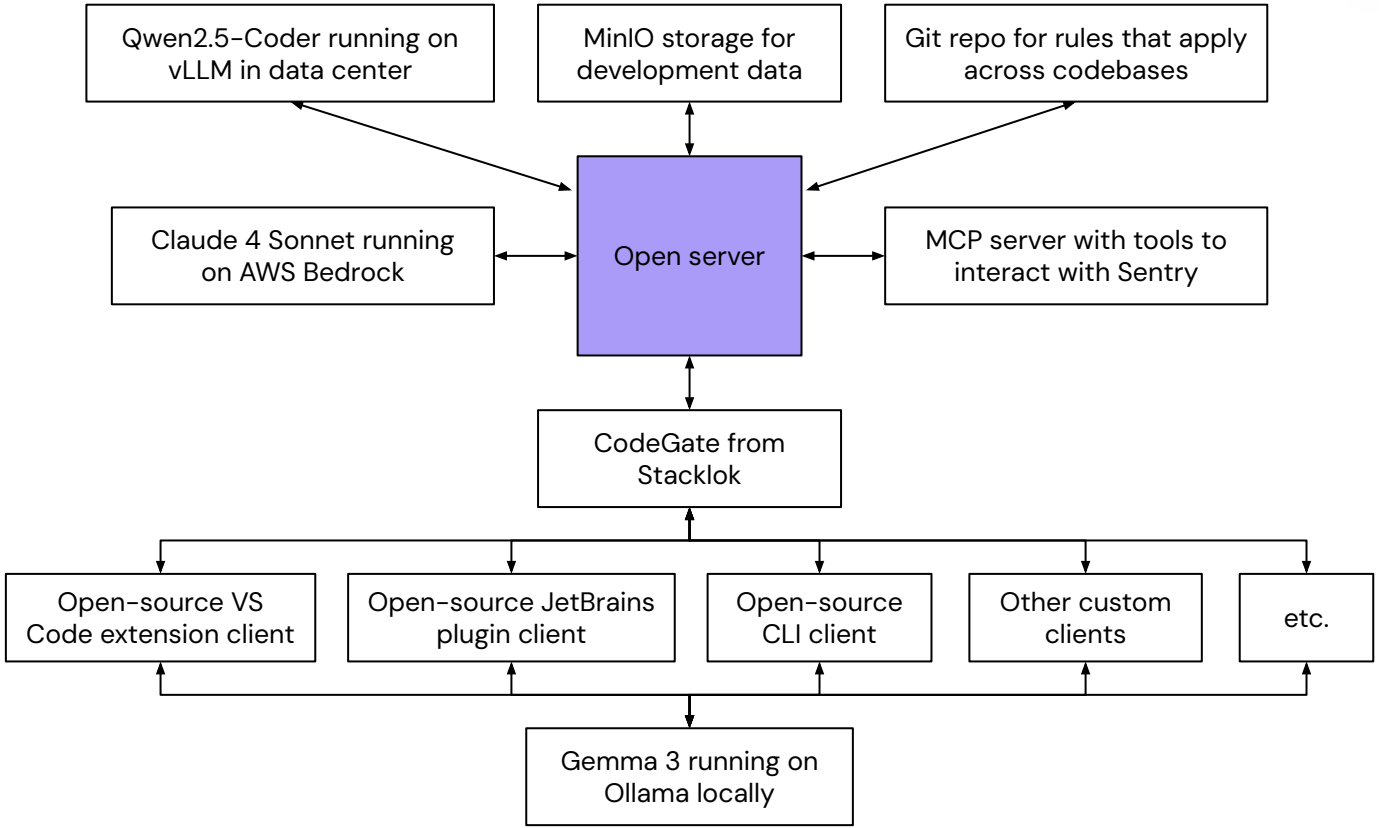
Ecosystem

Anyone can extend open servers; an open architecture empowers an ecosystem to form and grow around it. Example:

CodeGate from Stacklok

- Secrets PII redaction to protect your sensitive credentials and anonymize personally identifiable information
- Dependency risk awareness to update the LLM's knowledge of malicious or deprecated open source packages
- One of many examples of to come; win together by working together

An AI-native open architecture: Server



What is the AI-native open server now?

Examples:

- Authenticates users
- Proxies requests
- Manages configuration
- Enables observability
- Integrates with context sources
- Transforms development data before storage

What is the AI-native open server becoming?

- **Single source of truth** for how development works in an organization
- A natural language specification that **makes how you build software legible (i.e. rules) and accessible (i.e. tools) to models**
- This living description needs to be comprehensive, evolve with you, and require minimal human support / supervision

For this to happen, we need to design the required **1) workflows, 2) collaboration, and 3) governance**

1. Customization via the open server

Custom assistants to personalize the rules, tools, and models necessary for specific workflows. They make it easier for models to complete a task as desired by constraining and focusing the rules and tools. Examples:

- **Audit assistant** to audit for security vulnerabilities and create a report that helps you refactor and fix them
- **Plan assistant** to create a comprehensive product requirements document that can be used to guide implementation
- **Refactor assistant** to focus on improving existing code without adding new features

2. Collaboration via the open server

Collaborating with the organization and models to make **improvements of rules, tools, and models**

“Runtime” improvements (i.e. context windows)

- Enhancing rules utilization
- Improving tool integration
- Using dev data to improve context
- Better in-context learning

“Compile Time” Improvements (i.e. model training and fine-tuning)

- Enhancing via pre-training
- Optimizing via post-training
- Specializing for an organization
- Better domain-specific models

3. Governance via the open server

Enterprise requirements to provide the necessary guardrails. Examples:

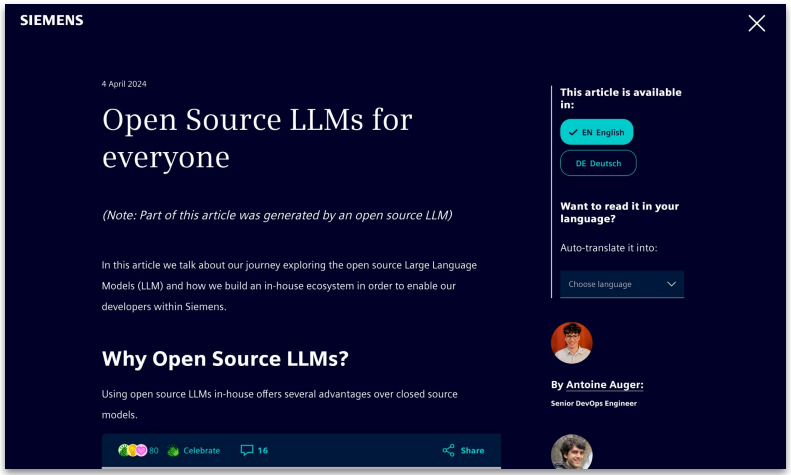
- Security-first design
- Data sovereignty options
- Granular permission controls
- Policy enforcement
- Comprehensive audit trails
- Credential management
- Cost, performance monitoring

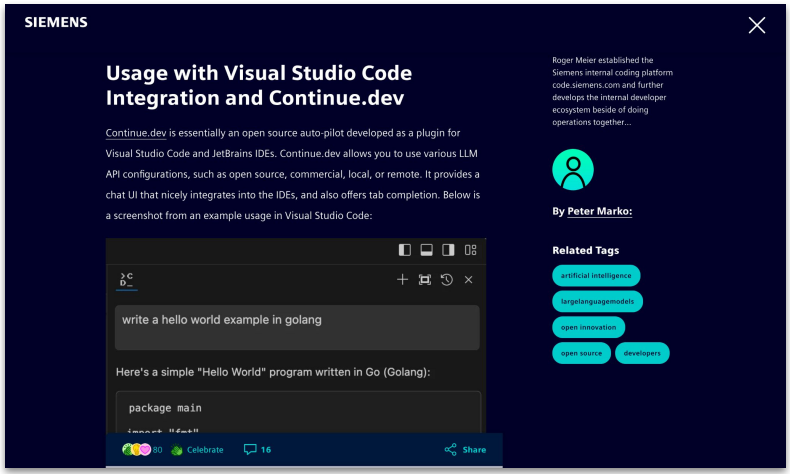
Why does an AI-native open architecture matter?

Why does an AI-native open architecture matter?

Creates significant advantages closed systems cannot match through

modularity, transparency, and interoperability





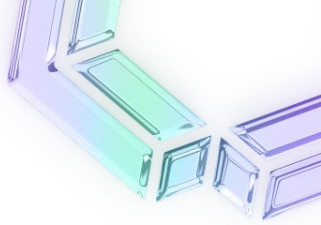
Why does an AI-native open architecture matter?

AI-native open architectures aim to amplify developers in the long run

- You keep your data to further automate your development with models

AI-native closed solutions aim to automate developers in the long run

- They collect your data to further automate you with models



Thanks!

Ty Dunn

Co-founder of Continue

ty@continue.dev