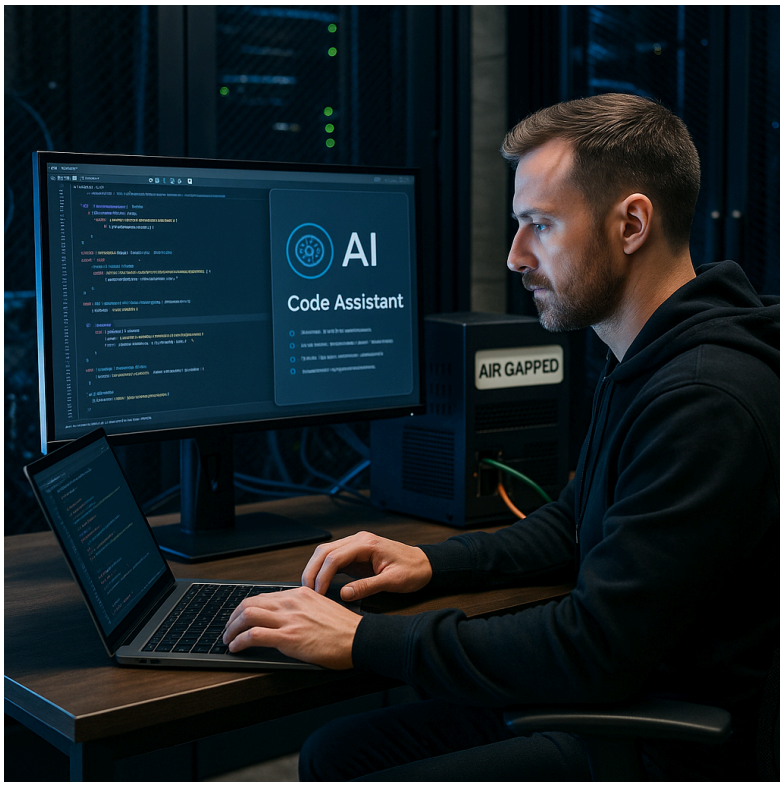
Enterprise AI Code Assistants for Air-Gapped Environments

By InuitionLabs.ai • 7/15/2025 • 80 min read

ai code assistants | air-gapped environments | on-premises deployment | enterprise ai | software development | llms | security | regulated environments | code generation | offline operation



Enterprise AI Code Assistants for Air-Gapped Environments

Introduction

AI-powered code assistants (AI pair programmers) are becoming integral in software development, suggesting code and helping with documentation, testing, and debugging. Enterprise adoption of these tools is rising rapidly – studies project that by 2028 as many as 90% of developers in large organizations will be using AI-powered code assistants in some form. However, enterprises in highly regulated or sensitive domains (finance, government, defense, healthcare, etc.) often require solutions that can be deployed on-premises within secure, air-gapped environments (isolated from the public internet). This report provides an in-depth review of current options for AI code assistants suited to such environments, covering both commercial and open-source solutions. We detail deployment requirements (hardware, containerization, system prerequisites), air-gapped operation capabilities (offline licensing, model updates, security best practices), supported languages and IDE integrations, and their fit into enterprise development workflows. We also include comparisons of leading solutions – Amazon CodeWhisperer, GitHub Copilot (Enterprise), Tabnine Enterprise, Qodo (formerly CodiumAI), Sourcegraph Cody – as well as open-source alternatives like CodeGeeX, Continue, and custom self-hosted models via Hugging Face Transformers. Relevant benchmarks and evaluations are cited to illustrate the performance and accuracy of these tools. All discussions are oriented toward the needs of enterprise IT architects and developers, with an emphasis on solutions that can operate securely **offline or in restricted networks**.

Deployment Requirements and Infrastructure Considerations

Deploying an AI code assistant on-premises requires careful planning of infrastructure. Many modern code assistants are backed by large language models (LLMs) that benefit from GPU acceleration and distributed deployment. **Containerization and Orchestration:** Most enterprise-grade solutions package the AI assistant as containerized services suitable for Kubernetes or similar orchestration. For example, Tabnine Enterprise is deployed as a set of services on a Kubernetes cluster (on-premises or in a private cloud). Sourcegraph Cody (the self-hosted Sourcegraph platform with Cody enabled) similarly provides Docker/Kubernetes deployment options as it integrates into the Sourcegraph server environment. Qodo (CodiumAI's enterprise platform) and other vendors also support cloud-agnostic deployment – in VPC, on bare-metal, or via container images – to meet enterprise needs.

Hardware Requirements: On-prem code assistants typically require substantial compute resources. In particular, GPU hardware is recommended (or required) to run LLM inference with low latency for multiple developers. Tabnine's documentation, for instance, provides sizing guidelines: a **single NVIDIA GPU** (such as an NVIDIA L40S or A100 80GB) can serve up to ~1,000 users, with 2–4 GPUs recommended for larger teams. Corresponding CPU and memory needs are also significant – Tabnine suggests for an on-prem server supporting 1000+ users at least 32–72 CPU cores, 72–256 GB of RAM, and 5–10 TB of fast SSD storage (to accommodate models and metadata). These figures underscore that running advanced code models internally is computationally intensive. Enterprises often provision dedicated AI inference servers or leverage existing GPU clusters for this purpose.

For control-plane or support services, containers typically run on Linux (e.g. Ubuntu or RHEL) and can scale across nodes. Tabnine's recommended Kubernetes control plane is 3 nodes (4 CPU, 16GB RAM each) for HA setups. Sourcegraph Cody's indexing and search services benefit from CPU and RAM (for large codebase indexing) but can function without GPUs if an external model API is used. However, to use Cody with a local model (via Ollama), an enterprise would need at least one machine with a compatible GPU or an Apple Silicon device (Ollama supports running models on Mac Metal GPUs) to host the LLM. In general, **NVIDIA GPUs with 20GB+ VRAM** (A100, H100, RTX 6000/8000, etc.) are preferred to run code models of 10–20B parameters at reasonable speed. Larger open-source models (like Code Llama 34B or 70B) may require multiple GPUs or splitting across nodes for inference, which adds complexity.

Networking and Storage: In air-gapped setups, all containers and model files must be available on the internal network. This often involves maintaining a local container registry and storage for model weights. Enterprises should plan for high-speed internal networks (10 Gbps or better) between the AI assistant servers and developer machines to minimize latency for code completions. Storage of several terabytes (preferably SSD) may be needed for model checkpoints, embeddings indexes (for context awareness), and logging data.

System Prerequisites: Most self-hosted assistants run on Linux x86_64 hosts. Kubernetes or Docker support is typically required. For instance, Tabnine Enterprise supports installation on any Kubernetes cluster and even provides a simplified single-node option using MicroK8s for evaluation. Adequate OS-level configurations (drivers for GPU, etc.) must be in place – e.g. installing NVIDIA CUDA drivers on the host or using Kubernetes GPU operator as noted in a Dell-Tech guidance for Tabnine deployment. Admin privileges are needed to set up these services, along with coordination with enterprise IT for provisioning the necessary VM/physical servers in data centers.

Example – Tabnine Enterprise Architecture: Tabnine's on-prem solution illustrates a typical deployment. It consists of a server component hosting the AI models and serving requests, which is installed on a Kubernetes cluster in the customer's environment. Developers install IDE plugins (for VS Code, JetBrains, etc.) that connect to this local Tabnine server over HTTPS (port 443). The server hosts one or more large language models (either Tabnine-provided or custom) to generate code completions and a suite of supporting microservices (for user management, analytics, etc.). Tabnine supports a **"hybrid" architecture** where smaller inference tasks or client-side models handle basic completions locally, while more complex completions are fetched from the server's

powerful model – all within the firewall. This architecture ensures low latency and scalability by balancing workload between client and server. Other enterprise tools have similar setups: e.g. Sourcegraph Cody integrates with an internal Sourcegraph instance that indexes code and calls out to an LLM service (which can be a local model or a proxied API), and CodiumAI's platform likely deploys an indexing service plus a local inference service for its generative code analysis.

In summary, deploying these AI assistants on-premises requires **robust hardware (especially GPUs)**, containerization (usually Kubernetes), sufficient storage for models/data, and careful network setup to mimic the connectivity these tools expect – all inside the organization's secure environment. Next, we examine how these solutions function in *air-gapped* mode and what is required to operate without external connectivity.

Air-Gapped Operation Capabilities and Considerations

Operating in a fully **air-gapped environment** (no Internet access) imposes particular requirements on AI code assistant solutions. Not all coding assistants are capable of functioning offline – notably, cloud-native services like GitHub Copilot and AWS CodeWhisperer **require an Internet connection** to their servers and do *not* offer on-premises deployment. In contrast, solutions designed for enterprise (like Tabnine and Qodo/CodiumAI) provide modes for fully offline use. This section highlights what's needed to use code assistants in air-gapped mode, including license management, model updates, and security best practices.

License Management Offline: Commercial enterprise tools usually enforce licensing via user seats or tokens. In an air-gapped setup, the licensing must be handled without calling home to a vendor's server at runtime. Many enterprise vendors accommodate this by providing **offline license files or license servers** that can run locally. For example, Tabnine Enterprise's admin console allows offline activation – once the license key is installed, the Tabnine on-prem server does not need to contact external services. Qodo (CodiumAI) similarly offers enterprise customers on-prem licenses for its platform. It's important to coordinate with the vendor to obtain a license that does not require periodic online verification. Typically, enterprise contracts include maintenance periods during which the customer can download updates and license files from a portal, which then remain valid in the isolated network. In fully offline mode, **user authentication** is often done against the company's SSO/LDAP internally rather than the vendor's cloud – for instance, CodeWhisperer's enterprise tier uses AWS IAM Identity Center for user management, but in a disconnected scenario that approach isn't applicable – whereas Tabnine allows integrating with the customer's SSO for user provisioning on the local server. Admins should plan how developers will be authenticated to use the tool (e.g. intranet web portal or tokens issued to IDE plugins) without contacting external identity services.

Model Updating Strategies: One challenge in air-gapped environments is updating the AI models (or receiving improvements and security patches). Unlike cloud services that update continuously, offline deployments rely on the enterprise to fetch updates (often manually). Vendors typically provide **periodic downloadable releases** – e.g. new Docker images or model packages – that can be transferred into the secure network. Tabnine's documentation includes an "updateguide" for private installations, indicating how to import new versions of the Tabnine server or models into the

cluster. An organization should designate a process (and a secure transfer mechanism) for obtaining these updates from a trusted external network, scanning them for integrity, and then deploying them internally. It is a best practice to **schedule regular update windows** to keep the AI assistant's model and software up to date with the latest improvements and bug fixes. Some vendors might offer delta updates or patch files for models to avoid re-transferring very large files. In the absence of updates, models can become stale – for instance, lacking knowledge of newly released libraries or languages – so even air-gapped systems should plan periodic refreshes. Open-source models require a similar strategy: e.g. if using a Hugging Face model like Code Llama, the team would periodically download newer model versions or fine-tuned variants from an external machine and copy them in. Keeping a **model registry** inside the air-gapped network (for example, an internal Hugging Face Hub mirror or an artifact store) can facilitate sharing updated models with all developers.

Security Best Practices: Air-gapped deployment is often chosen *because* of security – to ensure no source code or sensitive data ever leaves the premises. To maintain this security, several practices are recommended:

- **No Telemetry or Data Upload:** All external telemetry should be disabled. Enterprise-focused tools generally promise zero data egress. For example, Tabnine guarantees **zero data retention** and no usage of your code for training their models. In on-prem mode, Tabnine by default does not send any snippet or telemetry out, and others like Sourcegraph Cody operate completely locally (with local models) when configured for offline use. Administrators should double-check configuration to ensure any optional logging or feedback features that might reach the vendor are turned off. Many enterprise tools have a "telemetry off" switch or simply do not include that functionality in offline mode.

- **Isolated Model and Dependency Repositories:** Ensure that the models and containers the AI assistant uses are vetted and stored internally. This means running an internal container registry for images (so that the environment doesn't try to pull from Docker Hub at runtime), and hosting model weight files on internal storage. This avoids any attempt by the system to reach out to the internet for downloads. Documentation for air-gapped installs (like Tabnine's *Air-Gapped Deployment Guide*) will list all images and packages to preload.

- **Network Policies:** Even within the air-gapped network, adopt the principle of least privilege. The AI assistant servers should be firewalled so they can only communicate with developer workstations or IDEs on the necessary ports, and not with any other sensitive systems. If the assistant provides a web UI or API, limit access to it. Use internal TLS certificates for encryption in transit (Tabnine, for instance, communicates over port 443 with the client plugin, which should be secured inside the network). Regular internal security reviews of the deployment are prudent since these are powerful systems processing critical code.

- **Audit Logging:** Many enterprise solutions provide logging and auditing features to track usage. For example, Tabnine's on-prem admin console offers audit logs and usage reports. Enabling such logging in air-gapped mode can help detect any unusual usage patterns or potential misuse of the tool (for instance, a user trying to generate large amounts of code or extract sensitive info via prompts). Although no external threat exists due to isolation, internal misuse or unexpected AI behavior should still be monitored. Audit logs can record prompts and suggestions (possibly sanitized) for compliance.

- **Model Behavior and Filtering:** Even offline, enterprises should apply guardrails to the AI's outputs. This can include **licensing filters** (to avoid suggesting code from GPL or other restricted sources) and **security filters** for known insecure code patterns. Amazon CodeWhisperer, for example, includes a built-in security scan that can detect vulnerabilities in generated code and a reference tracker that flags if a suggestion closely matches public code, offering attribution. In offline scenarios, such features are valuable to prevent accidental introduction of problematic code. GitHub Copilot Enterprise has introduced optional filters to block secrets (like keys/passwords) and to **avoid verbatim suggestions from training code** longer than a few characters, for legal compliance. Enterprises should enable these where available. Moreover, companies can implement their own **"policy enforcement layer"**: for instance, intercept suggestions and run them through static analysis or secret-scanning before showing to users (some advanced setups use proxy models or custom scripts to do this). Ajith et al. note that teams in sensitive sectors combine on-prem deployment with policy filters to ensure the AI does not suggest insecure cryptography methods or other disallowed patterns.

In summary, **only a subset of AI coding assistants are viable in fully air-gapped settings**. Those that are (Tabnine, Qodo, Sourcegraph Cody with local models, open-source self-hosted models, etc.) require careful handling of licensing offline and proactive operational management (updates, monitoring). By following best practices – disabling external comms, maintaining internal update channels, and monitoring the AI's behavior – enterprises can harness these tools' productivity benefits while meeting stringent security and compliance requirements.

Leading Commercial AI Code Assistants (On-Premises and Enterprise Versions)

This section reviews the major commercial AI code assistants, focusing on their enterprise features, on-premises deployability, supported languages/IDEs, and suitability for air-gapped use.

Amazon CodeWhisperer (AWS)

Overview: Amazon CodeWhisperer is a cloud-based AI coding assistant from AWS. It integrates with IDEs via the AWS Toolkit (supporting **VS Code** and **JetBrains IDEs** among others) and provides real-time code suggestions as well as AI-generated code explanations and security scans. CodeWhisperer was introduced in 2022 and offers both a **free individual tier** and a **Professional tier** for organizations. It was designed in part to excel at scenarios involving AWS services and APIs. For example, it can suggest code snippets for AWS SDK calls and guide developers with step-by-step integration with AWS resources.

Supported Languages and IDEs: CodeWhisperer supports a more limited set of programming languages compared to some competitors. As of 2023, it officially supports **Python, Java, JavaScript, TypeScript, and C#**, with additional support for AWS automation languages (like SQL for Athena, CloudFormation snippets, etc.). It may also handle other languages in a basic way, but those five are the primary targets. In terms of IDEs, AWS provides support through extensions for **Visual Studio Code** and **JetBrains IDEs** (such as IntelliJ, PyCharm, WebStorm, etc.) via the AWS Toolkit. It also works in AWS Cloud9 (the cloud IDE) and AWS Lambda console, giving developers

suggestions while coding in those AWS environments. The focus on AWS tooling is evident – outside VS Code/JetBrains, support in other editors is not as broad (for example, there's no official support for Vim/Neovim or Visual Studio as of the latest info). This somewhat narrower IDE/language scope is a **trade-off for its specialization in cloud tasks**.

Deployment Model: Importantly, CodeWhisperer is delivered as a fully managed **cloud service** by AWS. There is *no on-premises or self-hosted deployment option* for CodeWhisperer; even the "Professional tier" operates by calling AWS's endpoints (it just adds enterprise management features on top). This means CodeWhisperer requires internet connectivity to AWS at development time. In practice, developers must have credentials (an AWS Builder ID for individuals, or corporate SSO via AWS IAM Identity Center for enterprise) and the IDE plugin will communicate with AWS's AI service to get completions. The *code* sent to AWS can be minimized – AWS allows opting out of data collection of code snippets in the Professional tier – but nonetheless, the service must exchange prompts and completions with AWS's cloud. There is **no air-gapped support**; CodeWhisperer cannot run inside a closed network. At best, an enterprise could set up a private VPC interface/endpoints to connect to AWS CodeWhisperer with low latency, but this still requires an AWS connection (and likely wouldn't be allowed in truly isolated environments). Tabnine's analysis confirms that CodeWhisperer (like GitHub Copilot) does *not* offer on-prem or VPC installation for completely offline use.

Enterprise Features: For organizations that *can* use the cloud service, CodeWhisperer's Professional tier (pricing about $19/user/month) provides a few enterprise-oriented capabilities. It allows **central management of user access** (administrators can allocate which developers have access and enforce organization-wide settings). Notably, CodeWhisperer includes a built-in **security scanning** feature that can be run on demand to scan your code (up to 500 scans per month in Pro) for vulnerabilities or AWS security best-practice violations. It also provides **reference tracking**: when it suggests a code snippet that resembles code from its training set (e.g. an open-source library code), it can detect this and provide attribution or flag it, helping avoid license compliance issues. By default, the **Professional tier does not share code snippets with AWS for model training**, addressing privacy concerns (whereas the free tier may share anonymized data unless opted out). These features show AWS's attempt to make CodeWhisperer more palatable to enterprises worried about IP leakage and security. Additionally, CodeWhisperer benefits from being part of the AWS ecosystem: it can integrate with AWS services (for example, Amazon CodeCatalyst and Amazon CodeCommit integrations are in the works, and "Amazon Q" – AWS's broader dev assistant platform – incorporates CodeWhisperer's functionality for cloud development).

Air-Gapped Suitability: In the context of this report (air-gapped environments), Amazon CodeWhisperer is **not suitable** if the requirement is full offline operation. It simply wasn't designed for that use case – it's a cloud SaaS. Organizations that must remain fully disconnected from the internet will generally have to look at other solutions. Some very high-security AWS customers might attempt to use AWS Snowball or Outposts to host certain AWS services on-prem, but at present AWS has not announced any "CodeWhisperer in Outposts" capability (and even if it did, it would still be a managed black-box rather than a self-run model). Therefore, **for strictly air-gapped scenarios, CodeWhisperer is ruled out** – its value could only be realized in a hybrid scenario where perhaps a segregated network with a controlled gateway can reach AWS API

endpoints (and even that may be disallowed by policy). Enterprises that *are* allowed to use external cloud services for development may find CodeWhisperer appealing, especially if they heavily use AWS – it excels at code examples for AWS services and is improving rapidly. However, its performance lags slightly behind the very best models. Independent evaluations show that CodeWhisperer's code generation quality is improving but still behind GitHub Copilot's: one study using the HumanEval benchmark found CodeWhisperer produced correct solutions only **31%** of the time, vs **46%** for GitHub Copilot (and ~65% for OpenAI's ChatGPT). This gap may narrow as Amazon refines the model, but it suggests that CodeWhisperer, while useful, might generate lower-quality suggestions on average than its top rival – unless one's coding tasks are specifically AWS-focused where CodeWhisperer was optimized to excel.

Summary: Amazon CodeWhisperer offers a **turnkey cloud solution** with some enterprise-oriented features (managed users, security scanning, reference tracking), and works well for developers already in the AWS ecosystem. However, it cannot be self-hosted or run offline. Enterprises with strict data isolation requirements will likely have to choose a different solution that supports on-prem deployment.

GitHub Copilot (Enterprise)

Overview: GitHub Copilot, launched by GitHub/Microsoft in 2021 and powered by OpenAI's Codex (and more recently GPT-4 for some features), is one of the most popular AI code assistants. It is often described as an "AI pair programmer" that provides code completions, entire function suggestions, and even natural language answers about code. Copilot was initially available as a cloud service for individuals, but GitHub has since introduced **Copilot for Business/Enterprise** plans to cater to organizations. These plans offer license management and some additional controls, while using the same underlying cloud service. Copilot's strength lies in its generality and the power of its model: it has been trained on a broad swath of public code (and possibly natural language) to support dozens of programming languages. It is *not* specialized to any one platform (unlike CodeWhisperer's AWS tilt) – for instance, Copilot is equally adept at Python, JavaScript/TypeScript, Ruby, Go, C#, C/C++, and more, even handling less common languages to some extent. It has also been extended with capabilities like **Copilot Chat**, **Copilot for Pull Requests** (which can explain or suggest changes in PRs), and **Copilot CLI** for terminal command suggestions, as part of the Copilot X vision.

Supported Languages and IDEs: GitHub Copilot can suggest code in *almost any programming language* (its training data covers a wide range). It is explicitly optimized for popular languages like Python, JavaScript/TypeScript, Ruby, Go, C#, C++, Java, etc., but users have successfully used it for shell scripting, SQL queries, and more. In terms of IDE support, Copilot has one of the broadest reaches: it has official plugins for **Visual Studio Code**, **Visual Studio 2022**, **JetBrains IDEs** (IntelliJ IDEA, PyCharm, WebStorm, etc.), **Neovim**, and also supports GitHub's own cloud editor. It integrates into the editor's autocomplete and can also provide a chat interface in VSCode/VS (Copilot Chat). Additionally, Copilot is available directly on GitHub's web interface for certain tasks (e.g., suggesting changes or generating descriptions in pull requests, answering questions about

code on GitHub). This wide integration means Copilot can fit into many developers' existing workflows with minimal friction.

Deployment Model: Copilot is a SaaS offering. All suggestion queries are processed by GitHub/Microsoft's cloud (which interfaces with OpenAI's models). *There is no fully on-premises version of GitHub Copilot available* as of 2025. Enterprises using Copilot Enterprise still connect to the same cloud endpoints, though with stronger assurances around data handling. GitHub has addressed some data privacy concerns – for example, for Copilot for Business, it guarantees that **no code snippets are retained or used to retrain the model**, and data from one customer is not seen by another. They have also achieved compliance certifications (SOC 2, etc.) to satisfy enterprise risk assessments. But fundamentally, Copilot requires internet connectivity to operate; the IDE plugin must communicate with the Copilot service. This makes it unsuitable for air-gapped networks – *Copilot cannot run on a disconnected machine*. Attempts by the community to "cache" Copilot or use it with a local model (for instance, pointing the Copilot extension at a local Ollama server) have not been supported and do not work in any persistent way. GitHub has been asked about offline support, and the answer so far has been negative: they currently have no offline/local version, focusing instead on cloud delivery.

For enterprises that *do* have internet access but behind proxies, GitHub provides guidance to allow the plugin to communicate through corporate proxy servers. But in a fully locked-down environment, Copilot is off the table. It's worth noting that Microsoft does offer **Azure OpenAI Service**, which lets enterprises use OpenAI's models (like GPT-4, which could be used for coding) in a more private manner – even in on-premises scenarios via Azure Stack – but that is a custom solution (not the Copilot product out-of-the-box) and requires significant engineering to integrate into IDEs. Copilot itself, in its official form, remains cloud-only.

Enterprise Features: Copilot for Business/Enterprise aims to address organizational needs primarily through **administrative controls and compliance** rather than on-prem deployment. Key features include:

- **Organization-wide Licensing:** Enterprises can purchase Copilot seats for developers and manage them via their GitHub Enterprise accounts. Enabling Copilot for an enterprise involves verifying payment and then the admins can enable or disable Copilot for specific teams or all org members. This central control is crucial for companies to track usage and costs.

- **Policy Settings:** Admins can set policies such as disabling **suggestions that match public code**. Copilot has a setting called *"Prevent Code Snippets"* which, when enabled, will avoid or flag suggestions that are longer than a certain length and exactly match public repository code. This is meant to reduce the chance of licensed code being suggested without attribution. Enterprises concerned about IP compliance often enable this to mitigate risk (though it may slightly reduce the utility of suggestions).

- **Telemetry controls:** Business tier allows opting out of collecting any feedback data from users. By default, Copilot might collect some info on accepted suggestions for product improvement, but enterprises can turn that off.

- **Integrated Chat and PR support:** Copilot Enterprise integrates the Copilot **Chat** feature (which uses GPT-4) directly into developers' IDEs and even into GitHub's pull request interface. For example, developers can ask "Why did this test fail?" in the PR, or have the AI suggest a fix. This is part of GitHub's Copilot X initiative. While not a "security" feature per se, it's an advanced capability that enterprises can leverage for code review workflows.

The **security profile** of Copilot Enterprise is also tuned for corporate use. It is compliant with SOC 2 and GDPR, and GitHub provides a Data Processing Addendum for customers. Still, the core security concern is that source code is leaving the premises to an external service. Many companies with moderate sensitivity tolerate this because of the promise of no retention and not training on their code, plus the productivity gains reported (studies showed Copilot users complete tasks ~55% faster in some cases). But for *highly sensitive* code, this may remain unacceptable.

Performance and Usage: GitHub Copilot is widely regarded as one of the most *capable* code assistants in terms of suggestion quality (largely thanks to the power of OpenAI's models). Anecdotally, developers find that Copilot can often generate 40-50% of their code in common scenarios. GitHub's own research noted around *30%* of all code being written by Copilot for users who fully adopt it, and the GitHub CEO predicted it could reach 80% in the future. In terms of benchmarks, Copilot (with its Codex model, roughly akin to GPT-3.5) is better than most open alternatives and in one study (cited earlier) had ~46% accuracy on HumanEval problems versus 31% for CodeWhisperer. The newer GPT-4 based capabilities (used in Copilot Chat) push the performance even higher (GPT-4's HumanEval is ~80%+) – indeed Anthropic's Claude and GPT-4 have achieved **85%+** on code benchmarks by mid-2025. That means Copilot (especially with GPT-4 in chat) can come quite close to expert-level coding on small functions. However, those frontier models are only available via the cloud (OpenAI API). There is currently no competitor that allows that level of model to run fully on-prem (models like GPT-4 are proprietary and exceedingly large). This creates a **trade-off**: if you need the absolute best quality and don't mind cloud, Copilot is excellent; if you need on-prem/offline, you will likely use a slightly less-powerful model, at least for now.

Summary: GitHub Copilot Enterprise offers **best-in-class AI assistance** integrated deeply with developer workflows on GitHub and popular IDEs. It provides admin controls and assurances suitable for many enterprises, but it **cannot be deployed on-premises or in an air-gapped way**. Companies requiring strict isolation will have to forego Copilot or pursue a custom solution (e.g. self-hosting an open-source model with similar capabilities). For those who *can* use it, Copilot can significantly accelerate development, writing an estimated 25-50% of code for developers in practice and supporting dozens of languages. Its limitation is entirely one of data residency: all processing occurs in the cloud. We now turn to solutions that are designed to be run within the enterprise's own infrastructure.

Tabnine Enterprise (Self-Hosted)

Overview: Tabnine is an AI code assistant that has been in the market since 2018, making it one of the earlier entrants (originally known for using GPT-2 based models). Tabnine's focus is on privacy and enterprise control – its tagline is "the AI that *you* control". Tabnine Enterprise offers a

comprehensive platform that can run in **protected SaaS, on-premises, or in a private cloud** depending on customer needs. Crucially, Tabnine is the only major commercial vendor that *explicitly supports fully air-gapped deployments* where the entire solution is hosted inside the customer's network with **zero telemetry or data leaving**. This makes Tabnine a leading choice for industries like defense, banking, and others that require on-prem AI.

Supported Languages and Features: Tabnine's AI was originally based on training over permissively licensed open-source code. It supports a very broad range of programming languages – effectively "any" language, since Tabnine uses large models that have seen many languages and also offers specialized models for certain languages. The Tabnine team indicates their LLMs and integrated third-party models cover **600+ languages, libraries, and frameworks**. In practice, this includes all common languages (Python, Java, C, C++, C#, JavaScript/TypeScript, Go, Ruby, PHP, Kotlin, Swift, etc.) and even less common or domain-specific languages (Matlab, Verilog, etc.), though the quality will be highest for mainstream ones. Tabnine provides both **code completion** (inline suggestions as you type) and an AI **chat assistant** within the IDE, as well as novel capabilities like "Tabnine Agent" which can perform commands like generating unit tests, doing code review suggestions, fixing errors, writing doc comments, explaining code, and so forth. These align with steps of the SDLC: Plan, Create, Test, Review, Fix, Document, Explain, Maintain, as listed in its documentation (essentially, Tabnine's AI can assist in many phases beyond just writing code). Some of these capabilities (e.g. test generation, code explanation) are implemented by chaining prompts or using smaller models locally, but all can function within an on-prem deployment.

IDE Integrations: Tabnine offers plugins for **all major IDEs**. This includes VS Code, JetBrains suite (IntelliJ, PyCharm, WebStorm, GoLand, Rider, etc.), **Eclipse**, and **Visual Studio 2022**, among others. It also has integration for Vim/Neovim and even a CLI mode. Essentially, Tabnine has very wide IDE coverage, aiming to meet developers "where they are." In an enterprise setting, this is useful because different teams may use different IDEs – front-end vs. back-end vs. data science teams, for example – and Tabnine can likely support all. The plugins for private installation point to your internal Tabnine server (after a user joins the team and gets an auth token). The user experience of Tabnine in the IDE is similar to Copilot: as you type, you get inline gray suggestions that you can accept with Tab/Enter. Additionally, Tabnine's chat or documentation features can be invoked via commands or UI buttons in the IDE.

On-Premises Deployment: Tabnine Enterprise can be deployed fully on-premises or in a virtual private cloud, giving **complete control over environment and data**. The deployment, as discussed earlier, is typically on a Kubernetes cluster. There are a few options – one common approach is the customer provides a Kubernetes cluster (on their bare metal or VM infrastructure) and follows Tabnine's helm charts to install the Tabnine services. Tabnine's docs also mention a **"fully air-gapped private installation"** as an option for customers, with guidelines on how to load images without internet. The on-prem server includes the core AI engine (which hosts Tabnine's own model checkpoints or possibly OpenAI/Anthropic models if Tabnine is configured to use those – by default, Tabnine ships with its **"Universal" code model** and some smaller task-specific models). Notably, Tabnine allows enterprises to **use their own models** as well: customers with private installations *"can connect and fine-tune their own models"*. This means if a company has, say, a custom internally-trained code model or wants to fine-tune Tabnine's model on their proprietary codebase,

the platform supports plugging that in. This flexibility is unique – it basically merges the open-source approach (custom models) with a commercial platform (nice IDE integration and management).

Air-Gapped Operation: In an air-gapped deployment, Tabnine guarantees **zero data exfiltration** – by design it does not call home when running on-prem, and all user code stays within the local server. An admin can optionally allow the Tabnine server to call out for updates or to use external models (for instance, Tabnine can integrate with OpenAI's API if configured, but obviously that wouldn't be used in an air-gapped case). With no internet, Tabnine still functions fully, providing completions and chat based on whatever models are hosted internally. The company emphasizes compliance: it is SOC 2 certified and even able to meet **ITAR/CMMC** requirements for defense use, as noted in a comparative table. This indicates Tabnine can be deployed in U.S. defense contractor environments where data handling is extremely scrutinized. In the same table, it's shown that Tabnine's on-prem deployment has **no telemetry and no cloud dependencies**. The only potential limitation in offline mode is that features requiring outside knowledge (for example, Tabnine has an **"Attribution"** feature that can show the source of a code snippet suggestion if it came from a known open-source project) won't be able to pull, say, license text from the internet – but Tabnine likely ships that data in the model or an offline database. Overall, Tabnine is **fully air-gap capable by design**, one of its primary selling points.

Security and Privacy: Tabnine's approach to training data is also designed to alleviate legal concerns. Their models are *trained exclusively on permissively licensed open-source code* (no GPL, no code that would contaminate outputs). This means the suggestions it produces are unlikely to directly include code under restrictive licenses, reducing legal risk. Additionally, Tabnine Enterprise offers **provenance** features – for instance, the admin can enable a mode where Tabnine will indicate if a suggestion is known to come verbatim from a training snippet, so the developer can be aware and find the source (or avoid using it if it's not acceptable). Tabnine also provides *team customization*: it can learn from your team's private code (if you allow it) to personalize suggestions. In Enterprise on-prem, this learning stays local (it can ingest your repos to better predict your patterns). All of this is under the control of the organization.

Performance: In terms of code generation capability, Tabnine uses somewhat smaller models than OpenAI's giant ones (for cost-efficiency on-prem), but it leverages optimization and user-specific tuning. Tabnine claims that across its user base it automates **30–50% of code creation** for each developer – comparable to Copilot's own claims. Being able to fine-tune on a team's codebase can improve relevance (e.g. suggesting internal API usage correctly). On standard benchmarks, Tabnine's exact performance is not published, but since Tabnine can potentially integrate larger models or external ones in hybrid mode, an enterprise could configure it to use a very strong model if they have the resources. The default Tabnine Universal model (as of 2023) was around 15B parameters (similar order to Code Llama 13B or OpenAI Codex). In practice, Tabnine's suggestions are considered good for boilerplate and typical tasks, but might be slightly less clever on complex problems than GPT-4. Tabnine is actively evolving though, and it benefits from being model-agnostic (the "AI Models" settings allow choosing from various model sizes or even using multiple).

Deployment Requirements Recap: As detailed earlier, to run Tabnine Enterprise one needs to provision a Kubernetes cluster with sufficient resources (including GPUs). For example, to support a mid-size team (hundreds of users), Tabnine might deploy multiple replicas of its model-serving component across nodes with GPUs. Tabnine and Dell published a sizing guide showing feasible hardware – e.g., 1x NVIDIA A100 80GB can serve ~1000 users with standard usage patterns. They also demonstrated deploying Tabnine on Dell PowerEdge servers in an air-gapped data center, illustrating the solution's viability on common enterprise hardware.

Summary: *Tabnine Enterprise* is a mature solution tailored for enterprises that need **privacy, flexibility, and offline capability**. It supports a wide array of languages and developer tools, and it stands out as **the only leading vendor to offer fully air-gapped on-premises deployment with no external dependencies by default**. Companies can keep their code 100% in-house and even extend Tabnine with their own models. The trade-offs are the overhead of hosting it (DevOps effort and hardware cost) and potentially slightly less raw model power than something like Copilot with GPT-4 – but many organizations find this an acceptable or even preferable trade for owning their data. In regulated sectors or any company unwilling to send code to third-party clouds, Tabnine is often the first commercial choice.

Qodo (CodiumAI Enterprise)

Overview: *CodiumAI* – now rebranded as **Qodo** – is an AI platform focused on "code integrity" which includes generating tests, reviewing code, and helping with debugging and documentation. Founded in 2022, CodiumAI initially gained popularity with a VS Code extension that would generate unit tests for your code (marketed as a "test generation AI"). In mid-2024, the company launched an **Enterprise Platform** with a broader suite of capabilities beyond tests. They emphasize *quality-first* generative AI, meaning the tool is designed to not just spit out code, but ensure the code works correctly and adheres to the organization's standards. Qodo's enterprise solution uses advanced techniques like **Retrieval-Augmented Generation (RAG)** – it indexes the organization's entire codebase to give the AI deep context – and a system that learns the organization's specific coding best practices. In other words, Qodo is trying to be the smart assistant that knows *your* code inside-out, acting almost like a knowledgeable team member familiar with your internal APIs and conventions.

Features and Use Cases: The primary use cases for Qodo (CodiumAI) are:

- **AI Code Review & PR Analysis:** Qodo can analyze pull requests, generate review comments, and suggest improvements. It aims to catch bugs or issues by running the AI on diffs and using the context of the repository (through RAG) to see if the changes align with the rest of the codebase.
- **Test Generation:** It can generate meaningful unit tests or integration tests for existing code, focusing on edge cases and potential bugs. The system is designed to "think" about how the code might fail and create tests accordingly. This goes beyond trivial input-output tests; CodiumAI was known for trying to create more insightful tests.

- **Code Generation & Completion:** It also has a code generation component (similar to Copilot in that it can suggest code completions or new code). The enterprise announcement mentions *"organization-specific code suggestions"* and the ability to leverage either their proprietary models or integrate other language models. So Qodo can act as a Copilot-like assistant but with more awareness of company context.
- **Documentation and Explanations:** Likely it can generate documentation for code (e.g., docstrings or READMEs) and explain code to developers.
- **Agentic Actions:** The branding "AI Agents for Code, Review & Workflows" suggests Qodo might support multi-step agents that can perform tasks across the dev lifecycle. For instance, automatically open a ticket if a code issue is found, or chain a code change with running tests. Qodo Command, Merge, Gen, Cover are names of products in their lineup, which correspond to running agents, merging PRs, generating code, and ensuring coverage. This indicates a comprehensive platform.

Supported Languages and Tools: Qodo claims to support "practically every programming language" for its test generation and analysis features. This is plausible because the underlying LLM can be language-agnostic and they likely have no hardcoded language limit. It integrates with **VS Code and JetBrains IDEs** (they provide extensions for both), and also with version control platforms (GitHub, GitLab, Bitbucket) for PR analysis. In the Q&A, they highlight integration with various git providers and mention that they publish some products as open-source (for example, they open-sourced a tool called "PR-Agent" which can be self-hosted to review pull requests using AI). The broad language support implies you can throw Python, Java, JavaScript, or even niche code at it and it will attempt to analyze/generate tests. For enterprise, this wide net is useful since large codebases are polyglot.

Deployment and On-Prem Options: Qodo explicitly offers **on-premises and even air-gapped deployment** for its enterprise platform. The PR news release states that the platform provides "maximum flexibility" with cloud, on-prem, and air-gapped options. On their website's enterprise section, they confirm support for multiple deployment models including self-hosted and VPC. This means an enterprise can install Qodo's server inside their data center. The architecture likely includes a central Qodo server that hosts the AI models and the context index (the codebase index) and provides an API that the IDE plugins or git hooks communicate with. The proprietary Qodo models can be deployed on that server, or the customer can choose to integrate "their preferred language models" with Qodo. For example, if a company wanted to use OpenAI's GPT-4 (with internet) or a local Hugging Face model, Qodo could route requests to that – but in an air-gapped scenario, one would use whatever model weights are hosted locally. We don't have specifics on hardware requirements from CodiumAI, but given it can integrate large models, a similar GPU need exists (for instance, to run a code model with tens of billions of parameters, multiple A100 GPUs would be required). The system also needs to **index potentially millions of lines of code**; Qodo uses vector databases or search indexes for RAG, which means the server should have sufficient memory and fast disk (maybe SSDs) to store embeddings for the entire codebase. This could be hundreds of MB to a few GB of embeddings for a huge monorepo.

Operating Qodo in an **air-gapped mode** would entail loading the model and the company's code data into the Qodo server with no external calls. Qodo is SOC 2 certified and emphasizes security

(they mention data is SSL-encrypted in transit and only necessary code fragments are analyzed). They also have a *Trust Center* and likely provide assurances that no data leaves the environment when self-hosted. For license management, Qodo Enterprise likely uses a per-user or per-seat pricing (the pricing page mention $45/user/month for enterprise plan which includes on-prem). In air-gapped, they would provide a license key or offline activation similarly to others.

Unique Strengths: Qodo's differentiator is the deep customization for an enterprise's own code. By indexing the entire codebase and learning coding standards, it addresses a major shortcoming of general code assistants – lack of context about proprietary code. As their CEO put it, without knowledge of the company's code, an AI assistant is like *"an eager intern on their first day — smart but lacking company-specific knowledge"*. Qodo tries to fill that gap, which can result in more relevant suggestions that fit the existing architecture. Over time, it "reinforces the enterprise's specific coding standards" automatically. This could mean, for example, if your company has a certain way of logging errors or a specific pattern for database transactions, Qodo will learn that and ensure suggestions conform. This learning likely happens through feedback loop – as developers accept or reject suggestions, Qodo adjusts its understanding (and possibly fine-tunes a model or updates its rules database).

Performance and Benchmarks: There isn't public benchmark data for Qodo's own models. But since Qodo can integrate various models, performance can be as good as the chosen model. If using their proprietary model, one might assume it's on par with other code-specialized models (perhaps CodiumAI fine-tuned a version of Code Llama or similar). Qodo has referenced that enterprises can use either Qodo's "state-of-the-art models" or plug in something else, which implies a flexible backend. When it comes to code quality outcomes, Qodo emphasizes reducing bugs and catching issues – these are harder to quantify in a simple benchmark. In absence of specific numbers, one can say Qodo's approach is to improve real-world code quality and consistency rather than to maximize a synthetic benchmark like HumanEval. One anecdotal metric: in an internal study or demo, Qodo might demonstrate an increase in code coverage due to generated tests or a decrease in post-deployment defects after using their tool. While not published in our sources, that is the kind of KPI an enterprise would look at for such a tool.

Air-Gapped Suitability: Qodo is **well-suited for air-gapped use** because it was designed with on-prem as a first-class option. It allows running entirely offline and even integrating open models if needed (so if an enterprise doesn't want to rely on CodiumAI's model, they could use an internal model). The platform's design of embedding the entire code repository means it does heavy lifting locally, without calls to external knowledge bases. Any updates to Qodo (like new model versions or bug fixes) would have to be transferred inside periodically. The company being relatively young, an enterprise should ensure they will provide long-term support for on-prem deployments (patches, etc.) as that can be a risk with startups. But given they highlight air-gapped support in marketing, it appears to be a core part of their strategy, not an afterthought.

Summary: *Qodo (formerly CodiumAI)* offers an **enterprise AI coding assistant platform focused on code quality**, with capabilities for intelligent code completion, test generation, and automated reviews. It distinguishes itself by deeply integrating with the enterprise's codebase context (using RAG and learning systems) to produce highly relevant suggestions that adhere to company

standards. Qodo supports **fully on-prem and even offline deployments**, making it suitable for restricted environments. It supports all major languages and works via IDE plugins and VCS integrations. Organizations that prioritize **improving code correctness and maintainability** alongside developer productivity might find Qodo particularly attractive. In an air-gapped scenario, Qodo would run entirely within the firewall, processing code with its local models and databases to assist developers without any external data exchange. The complexity of the platform is higher than a simple autocomplete tool, but it promises a more **holistic AI helper** that can not only write code, but also ensure that code is high-quality and in line with your codebase's patterns.

Sourcegraph Cody (Enterprise Self-Hosted)

Overview: *Cody* is an AI coding assistant developed by Sourcegraph, the company known for code search and navigation. Sourcegraph's core product is a code search/indexing tool that many enterprises use internally to search across all their repositories. In 2023, Sourcegraph introduced Cody to add an AI layer on top of this code knowledge. Cody can be thought of as a combination of a **code-aware chatbot and completion engine** that leverages Sourcegraph's extensive code indexing. One of its big advantages is having access to your entire codebase context via Sourcegraph – it can literally "know" all your code (through embeddings or search) and use that to answer questions or make suggestions with citations. Sourcegraph Cody is available in Sourcegraph's cloud offering, but importantly for us, it's also available in the **self-hosted Sourcegraph Enterprise** that many companies run on-prem. Cody was made generally available in early 2024 for enterprise customers.

Features: Cody provides multiple modes of assistance:

- **Code Autocomplete:** It offers inline completions (similar to Copilot/Tabnine) in supported IDEs. The team optimized it for speed – boasting ~24% faster single-line completions after some updates – and improved quality for multi-line suggestions with specialized algorithms. It currently supports multi-line completion for a list of file types/languages including C, C++, C#, Go, Java, JavaScript/TypeScript, Python, PHP, Ruby, Rust, HTML, CSS, etc...
- **Chat Q&A:** Cody has a chat interface where developers can ask questions in natural language. Uniquely, because it's tied to Sourcegraph, you can ask questions like "How is this function `X` used across our codebase?" or "Explain what the `paymentProcessing` module does," and Cody will fetch relevant code snippets from your repositories to answer, citing the source files. It's akin to having Stack Overflow for your internal code. It can also answer general programming questions if allowed, but its main value is code-specific Q&A with **citations and links to the code**.
- **Refactoring and Code Commands:** Cody can perform tasks via commands, such as "Refactor this code for performance" or "Add error handling to this function." When you highlight code and give an instruction, Cody will attempt to rewrite it accordingly. This uses AI plus Sourcegraph's context to ensure consistency.
- **Multi-Repo and Context Management:** Cody Enterprise can handle multiple repositories as context at once, acknowledging that large companies have many services and a question might span them. It also introduces syntax to explicitly include certain files or symbols in context (`@filename` or `@#symbol`) so the user can guide the assistant.

- **Model Flexibility:** Cody doesn't use a single model; it rather acts as an orchestrator. It allows choosing between different LLM backends. In the cloud version, they offer OpenAI's GPT-4, Anthropic's Claude (v2), and others. In an on-prem situation, one cannot call those external APIs (if truly air-gapped), but Sourcegraph implemented a solution: **local model inference via Ollama**. Ollama is an open-source tool that runs LLMs like LLaMA locally on device/servers. Sourcegraph integrated with it so that Cody can use models like "`deepseek-coder:6.7b` and `codellama:7b`" completely offline. This means that an enterprise can configure Cody to use a local model for all completions and chats. While 7B models are relatively small (and not as powerful as GPT-4), this is evolving – one can imagine using larger open models (like Code Llama 34B) with Ollama as well. The key is the architecture supports plugging in local models, which **enables Cody to function in air-gapped mode**.

- **Integration with Dev Workflow:** Since Sourcegraph is often connected to authentication (SSO) and repository management, Cody integrates with that. It respects permissions – it won't show code from a repo you don't have access to, for instance. Also, Sourcegraph Cody could be accessed via multiple frontends: a VS Code extension (so you get Cody in your IDE), the Sourcegraph web UI (for example, you can chat with Cody on the Sourcegraph web app while browsing code), and even a CLI.

On-Prem Deployment: Sourcegraph Enterprise is typically deployed via Docker Compose or Kubernetes on-premises. Cody is essentially a feature of Sourcegraph that can be enabled. When enabling it on-prem, the admin must configure a **model provider**. Options include calling OpenAI/Anthropic (if the instance has egress – not our case here), or pointing to a self-hosted model. The **experimental local inference support via Ollama** is a game-changer for fully disconnected use. In practice, an enterprise running Sourcegraph in an air-gapped network would set up an Ollama server (which runs on macOS or Linux with MPS or CUDA for GPU acceleration) loaded with an open-source code model. The Sourcegraph application then directs Cody's requests to that local Ollama endpoint. In July 2023, Sourcegraph also announced support for **Azure OpenAI** (for customers who have Azure OpenAI deployed in their region, they can use that endpoint) – but again, that's more for connected scenarios, whereas local Ollama is for offline. The fact that Sourcegraph built this integration indicates they have customers specifically needing offline AI.

Air-Gapped Operation: With local models, Sourcegraph Cody can run completely offline. In a blog, they explicitly note *"you can leverage Cody offline and in air-gapped environments"* with local inference. This addresses a common concern: many developers work in environments without internet (for security) and couldn't use Copilot or others – Cody gives them an option. However, we should note that the quality of suggestions will depend on the local model used. A 7B parameter model will be decent at simple completions but not nearly as fluent as a 70B model or GPT-4. Enterprises could potentially run larger models (like a fine-tuned Llama 2 70B) if they have the hardware; Ollama and similar frameworks are improving to support that (though 70B might require splitting across multiple GPUs). Alternatively, since Sourcegraph is flexible, one could integrate a different self-hosted inference server (there are reports of people hooking Sourcegraph to Hugging Face's text-generation-inference or other custom endpoints).

Security & Privacy: Sourcegraph being self-hosted means all your code index remains internal. Cody's design is such that it *cites sources* for any code it shows. This transparency is useful in enterprise: if Cody suggests something that came from an internal library, it will link to it, letting the developer verify and also navigate to that code in Sourcegraph. It helps build trust because the

developer can see exactly which context the AI used. For compliance, Sourcegraph likely ensures no telemetry is sent when running on-prem with Cody (unless the admin opts into something). There was a mention of a separate analytics repository for air-gapped Cody usage, implying that Sourcegraph may by default collect usage data but in air-gapped that's disabled and they have provided SQL scripts for internal analytics if needed.

Comparison with Others: Sourcegraph Cody's advantage is **deep code context**. It's probably the most powerful for answering questions about a large codebase or finding how to use a function, etc., because it leverages the search index. In contrast, Copilot or Tabnine might have access only to the file you're editing and maybe some open files, not the entire company's code. This context window problem is solved elegantly by Cody through retrieval techniques. The drawback could be that Cody's suggestions might be slower if the index retrieval takes time (though they optimize it). Also, Cody's quality depends on the model: if offline, you might not reach the same raw code-gen capability as Copilot's cloud models. But the context relevancy might offset a weaker model, e.g., a smaller model given the right pieces of code from your repo might perform surprisingly well. Community tests have shown local models via Cody can handle tasks like referencing relevant files when asked design questions. Another benefit: **Cody in PRs** – Sourcegraph can integrate with code hosts to offer AI answers/reviews on merge requests (similar to what GitHub is doing). If an enterprise self-hosts Sourcegraph and connects it to, say, GitLab, developers could get AI-generated PR feedback internally.

Supported IDEs: Currently, Cody is officially supported in **VS Code** (and VS Code-based IDEs) via an extension, and they have a **JetBrains plugin in beta** (or soon, as indicated by their community). There is also a web UI on Sourcegraph itself for Cody. Compared to Tabnine or Copilot, IDE coverage is a bit narrower (not in Vim or Eclipse yet). But VS Code covers a large portion of developers, and JetBrains support means it will soon cover that segment too.

Summary: *Sourcegraph Cody* is a powerful AI coding assistant that shines in **knowledge of large, enterprise codebases**. Integrated with Sourcegraph's code search, it provides code completions and an AI helper that can reference and cite your entire repository history. For enterprises, its major advantage is that it can be **self-hosted** and configured to run with **local AI models for offline use**. This allows usage in air-gapped environments, albeit with some trade-off in raw model strength unless the enterprise invests in running very large models internally. Cody supports a wide range of languages (basically any language in your codebase, since it treats them all as text to index) and is particularly useful for answering questions about how code is used or where things are defined, making it a great tool for onboarding new developers or navigating legacy systems. With Sourcegraph Cody, even teams in high-security environments can leverage AI assistance, keeping all data on-prem. It essentially brings an "internal Stack Overflow + Copilot" to the enterprise, respecting your permissions and privacy.

Open-Source and Self-Hosted AI Code Assistant Options

Beyond commercial offerings, there is a rich ecosystem of open-source tools and models that enterprises can leverage to build their own AI coding assistants. These range from pre-trained code

models that one can run locally, to fully open-source assistant frameworks with IDE integration. Using open-source solutions can be attractive for air-gapped and budget-conscious scenarios because they typically involve **no per-user licensing cost** and offer **full control over the software and model weights**. The trade-off is that setup and maintenance become the enterprise's responsibility, and model performance might lag behind the very latest proprietary models (though the gap has been closing steadily). In this section, we discuss several notable open-source options:

CodeGeeX (Open-Source Code LLM and Extensions)

What is CodeGeeX: CodeGeeX is an open-source large language model specifically for code generation, released by researchers from Tsinghua University (THUDM) and collaborators. It's a **13 billion parameter** multilingual code model trained on over 20 programming languages. As of its release (late 2022, paper accepted to KDD 2023), it was one of the largest open code models available. CodeGeeX can generate code, translate code between languages, and even provide natural language explanations. It was trained on a massive 850 billion tokens of code data on supercomputers, and notably was trained on both **English and Chinese** code and comments, making it multilingual.

Features: CodeGeeX supports several unique features out-of-the-box:

- **Multilingual Code Generation:** It can produce code in many languages (they mention Python, C++, Java, JavaScript, Go among others) at a high level of competence.
- **Cross-Lingual Code Translation:** CodeGeeX can take a code snippet in one language and translate it to another (e.g., convert a Python function to Java or C++). The authors provided an easy interface for this, which is useful for porting code between stacks.
- **VS Code and JetBrains Extensions:** Perhaps most relevantly, the CodeGeeX team built IDE extensions – a VS Code extension (available on VSCode Marketplace) and a JetBrains plugin. These allow developers to use CodeGeeX in the editor for completions, documentation, etc., similar to Copilot. The extension usage can either call a deployed model or possibly use a smaller local model (they had a client-side mode using a 2.7B model for quick suggestions).
- **Open and Cross-Platform:** CodeGeeX's code and model weights are open (the model is under an open license for research, and the code is Apache-2.0). It supports running on both NVIDIA GPUs and Huawei Ascend AI processors, and the team provided optimizations like an **INT8 quantized version** and multi-GPU support to lower the hardware barrier. For example, they got the model running in 15GB GPU memory with 8-bit quantization, meaning a single NVIDIA V100 (16GB) or RTX A6000 (48GB) can handle it.

Performance: The CodeGeeX paper introduced a benchmark called HumanEval-X to measure code generation in multiple languages. They reported that CodeGeeX achieved the highest average performance among open-source models at the time on this benchmark github.com. This suggests it was quite competitive against models like Codex 12B (if that were open) or early CodeGen models. By now (2024/2025), newer open models like Meta's Code Llama have emerged, likely surpassing CodeGeeX. For instance, Code Llama 34B scored ~54% on HumanEval (Python), whereas CodeGeeX 13B might be a bit lower (they didn't directly compare but presumably

somewhere in the 40% range). Still, CodeGeeX is solid, especially given its multilingual ability – it specifically emphasizes being good at Chinese prompts and Chinese code comments, which could be useful for companies in bilingual environments.

Use in Air-Gapped Setting: Since CodeGeeX is self-hosted, one can download the model weights (they even provided a link to Hugging Face and their own site) and run the model on internal hardware. For an air-gapped network, you'd simply have the model running on a server with GPUs and point the VS Code/JetBrains plugin to that server (the plugin likely allows configuring a local endpoint). There is no need to contact any external API; CodeGeeX does not phone home. So it's fully offline-capable. The resource requirement is a consideration: a 13B model will perform decently on modern GPUs – e.g., it can generate code with a few tokens per second on an A100 GPU – but might not match the snappiness of cloud services for very large completions. However, int8 quantization and multi-GPU parallelism can help meet latency needs. For example, by using FasterTransformer optimization, CodeGeeX achieved <15ms per token generation in int8 mode, which is quite fast and indicates it can handle real-time assistant use if properly optimized.

Multilingual Benefit: Many enterprise codebases involve multiple languages (for instance, backend in Java, frontend in TypeScript, some scripts in Python). CodeGeeX's ability to handle >20 languages is a plus. It even supports some less common languages – for example, the mention of cross-language translation implies it was trained on parallel code in languages like perhaps C#, Rust, etc., though the exact list isn't fully given in the snippet. The main languages explicitly tested were Python, C++, Java, JS, Go.

Extensibility: Being open-source, CodeGeeX could be fine-tuned further on an enterprise's own code (provided one has the expertise and compute). Since the model weights are available, a company could perform additional training (subject to license) to specialize the model for their style or libraries. This is a level of control not possible with closed models.

In summary, **CodeGeeX** provides a capable foundation for an on-prem code assistant with **no licensing costs** and built-in IDE integration. In an air-gapped environment, an enterprise can deploy the CodeGeeX model on local servers and have developers use the VS Code/JetBrains plugin to get suggestions, code completions, and even perform code translations between languages. It is a tangible example of how open-source models can step in as a "self-hosted Copilot." While newer models like Code Llama may have higher raw performance, CodeGeeX remains one of the most **feature-complete** open solutions (given its extensions and multilingual support). Enterprises should weigh the performance gap versus proprietary tools; but with CodeGeeX 2 and even CodeGeeX 4 announced (which presumably bring improvements), open models are quickly narrowing the quality gap, making them viable for many uses.

Continue (Open-Source IDE Assistant Framework)

What is Continue.dev: *Continue* is an open-source project that aims to provide an extensible, self-hosted AI assistant directly in your development environment. It brands itself as "the open-source AI code assistant that puts you in control". Continue is essentially an IDE extension (for VS Code and JetBrains IDEs) that can interface with any language model – local or remote – and provide a range

of AI-driven features like autocompletion, chat, code refactoring, and even integration with external tools (e.g., running tests, reading documentation). The goal of Continue is to give users a Copilot-like experience but with full customizability: you can choose which model to use, tweak prompts, and even add new "skills" to the assistant.

Key Features of Continue:

- **Model Agnostic:** Continue can work with multiple AI models. In its interface, it lists support for connecting to models via **Ollama**, **OpenAI API**, **Anthropic API (Claude)**, **Azure OpenAI**, **Together.ai API**, **Mistral** and **Local model runners like LM Studio**. Essentially, whether you have a local LLM or want to use a cloud one, Continue can plug into it. For offline use, one would likely use *Ollama* (to run local models on Mac/Linux) or *Local LLM Studio* or the HuggingFace Transformers directly. Continue is future-proof in that if a new model comes out, you can integrate it by configuring or writing a small connector.

- **Multiple Capabilities (beyond completion):** Continue's IDE extension provides a chat sidebar where you can ask questions or instruct the AI, inline completion as you code (the typical "Tab to accept" behavior), and an "edit" capability where you can highlight code and instruct the AI to modify it (e.g., "optimize this function"). It effectively merges the functionalities of Copilot and ChatGPT into one tool. For instance, you can highlight a block and ask Continue to explain it or to write tests for it, and it will do so, inserting the results in your editor.

- **Customizability and Extensibility:** Because it's open-source, developers can extend Continue with custom "blocks" – these blocks might be rules, or additional context providers. The Continue documentation mentions things like "Rules blocks" and "Methodology blocks" continue.dev. This hints that you can script Continue to follow certain approaches or integrate with other data sources (like having it use your documentation or your issue tracker as context). This is quite powerful for enterprise use: you could, for example, integrate Continue with your internal wiki, so that the AI can pull in relevant design docs when answering a question, all offline.

- **Privacy and Control:** By being locally hosted, Continue ensures your code does not leave your machine or network (assuming you're using local models). Even if using an API model like OpenAI, you have the option to self-host an instance of their model (via Azure or on your own GPU if possible). The user (or admin) has full control over what context is sent to the model. Continue is "your AI, your advantage" as their site says, emphasizing that you can tune it and it will improve over time for your environment.

Deployment in Enterprise/Air-Gapped: Continue itself is mainly an IDE plugin (plus a small backend that runs on the developer's machine coordinating with the model endpoint). In an enterprise scenario, one might deploy a centralized model server (for example, a server running an open-source model like Code Llama 34B with text-generation-webui or Ollama) accessible to all dev machines. Then each developer installs the Continue plugin in their IDE and configures the model endpoint to point to that server. The plugin is open-source, so it could be forked or modified if needed for internal distributions. There is no licensing server or seat counting – it's free to use. The main requirement is to have compute for the model if it's a large one. Alternatively, each developer could run a smaller model on their own workstation using Continue (for instance, a 7B model might run on a high-end laptop with CPU or a small GPU). This distributed approach might work for small models, but for larger ones it's more efficient to centralize on a beefy server.

Model Options (Performance): Continue doesn't come with its own model; you plug in others. So performance can range widely:

- If using a top open model like Code Llama 34B or StarCoder 16B, you can get fairly good results (Code Llama 34B's ~54% HumanEval means it's nearing Codex level for Python). If you can run Code Llama 70B in your environment, that's even better (~67% HumanEval, close to GPT-4's pass@1 of ~80%). But 70B is heavy, likely requiring multiple GPUs.
- If resources are limited, one could use smaller models (Mistral 7B, which is a newer model known for good quality at small scale, or Qwen-7B). Those will be faster but less accurate.
- The nice part is you can even set up Continue to allow switching models as needed. Perhaps for quick completions you use a smaller model (fast), and for a deeper chat or analysis you have an option to invoke a larger model. Continue's UI in VS Code might allow picking from multiple configured backends easily (some community forks have done multi-model setups).

Integration with Enterprise Tools: Continue is built to be extensible. For example, it references integration with "GitLab Issues, Confluence pages" as context sources continue.dev. This means in an enterprise, Continue could be configured to pull relevant snippets from your issue tracker or documentation when answering a question. That's similar to what Sourcegraph Cody does with code; Continue could do with arbitrary text sources. In an offline environment, one could ingest internal documents into a vector database and have Continue use that. These are advanced use-cases, but the open nature allows for it.

In short, **Continue** provides an **open-source alternative to Copilot/Cody** that is highly **flexible**. In an air-gapped setup, it can run entirely locally with whichever model you supply, ensuring no outside data transfer. It might require more tinkering and configuration than a turnkey product, but it has a growing community (backed by an MIT license and even some YC funding as per their site). For enterprises with capable ML or DevOps teams, Continue.dev could be adapted into a very customized assistant that fits their exact workflow.

Hugging Face Transformers and Custom Models

Instead of an integrated solution, some enterprises may opt for a more DIY approach: directly using open-source models from Hugging Face (or other repositories) and integrating them into internal tools. The **Hugging Face Transformers library** provides a standard way to load and run hundreds of pre-trained language models, including many tuned for code. With such models, enterprises can create bespoke code assistants.

Notable Open Code Models:

- **StarCoder** (15B, trained by Hugging Face & ServiceNow as part of BigCode): StarCoder is a 15-billion parameter model trained on a large corpus of permissively licensed GitHub code. It supports over a dozen programming languages and was one of the top open models when released in mid-2023. StarCoder's performance on Python (HumanEval) was around 33% pass@1 initially, but it excels in multi-language support and has an open license for commercial use. There's also a StarCoderBase (pretrained) and StarCoder fine-tuned on instruction (StarCoderChat). Enterprises can use StarCoder as a foundation model and even fine-tune it on their own code samples to personalize it.

- **Code Llama** (7B, 13B, 34B, 70B by Meta): Code Llama, released in Aug 2023, is arguably the most powerful open-source code model suite. The largest, Code Llama 70B, achieved **53% on HumanEval** (and specifically a Python-specialized variant hit 67.8%, nearly reaching GPT-4 level). These models are free for research and commercial use (under a community license). The 34B and 70B models are heavy to run but produce very high-quality code suggestions and even beat GPT-3.5 on some benchmarks. For an enterprise with a strong GPU cluster, deploying Code Llama 34B or 70B could yield excellent results in an air-gapped environment, fully offline. For instance, Code Llama 70B needs roughly 8×80GB GPUs for fast inference or can run on fewer GPUs at the cost of speed. This is non-trivial but not out of reach for well-funded IT departments. The payoff is an AI coding assistant nearly as good as Copilot, with no data leaving the network.

- **PolyCoder, GPT-J, etc.:** There have been other open models like PolyCoder (2.7B) and GPT-J-6B fine-tuned on code. These are smaller and less capable by today's standards but very easy to run on a single device (even CPU). They might serve if resources are extremely constrained, but their quality (Polycoder had ~15% on HumanEval for C) is quite low relative to modern models. More recent small models like **Mistral-7B** or **Qwen-7B** (by Alibaba) have shown surprisingly good performance on general tasks; a code-tuned variant of those could be effective for basic completions.

- **WizardCoder**: This is a community fine-tune (based on Code Llama 15B or 30B) that was optimized for coding instructions. WizardCoder-34B (a 34B model) reportedly achieves ~57% on HumanEval, making it one of the best open models in late 2023. It's an example of how open models are being steadily improved by fine-tuning on specialized instructions.

Running Models On-Prem: Hugging Face's `transformers` and `accelerate` libraries make it straightforward to load models and run them on hardware. For serving multiple users, one might deploy a service like **HuggingFace Text Generation Inference (TGI)** or **vLLM** or similar optimized serving frameworks. These can handle many simultaneous requests and offer features like token streaming (so users get results as they are generated) and even branching (for multiple suggestions). For example, an enterprise could set up a Kubernetes deployment of HuggingFace's TGI with a Code Llama 34B model, expose an internal API, and then either use existing editor plugins (like VS Code's generic AI extension) or write a small adapter for their IDEs to consume that API. It's more engineering effort, but completely viable. Many companies have experimented with this using models like GPT-J or CodeGen in the past; now with models like Code Llama the quality might finally be "good enough" to replace cloud services in daily use.

Security and IP Considerations: When using open models, enterprises should be mindful of the model's training data to avoid IP issues. Most open code models are trained on open-source code, but not always strictly permissive licensed. For instance, Code Llama was trained on a subset of StackOverflow and GitHub, which presumably excludes GPL and such (Meta didn't fully detail, but it's a concern). StarCoder was trained only on permissive license code and even filters outputs to avoid verbatim copying of large chunks from training code. Indemnification is on the user (no vendor to promise protection), so enterprises should do their own due diligence and maybe implement output filters. Tools like **AI output detectors for known code** or checking for license text in suggestions can help mitigate "license contamination" risks. Ajith's guide notes that both proprietary and open tools should ideally use models trained on permissive data and provide transparency. Using Hugging Face models gives you that transparency – you know (in general) what the model was trained on and you can decide if it's acceptable.

Benchmark & Performance: As mentioned, open models have rapidly caught up. Code Llama 70B's ~53-68% on HumanEval is a highlight. In fact, Meta reported GPT-4 at 67% and their Code Llama-P 34B at 56% – while 56 is below 67, it's in striking distance and certainly better than GPT-3.5's ~48%. This implies an enterprise running Code Llama 34B or 70B offline might get suggestions of an accuracy in between GitHub Copilot (which might correspond to GPT-3.5 Codex quality) and GPT-4. If combined with retrieval (like using the enterprise's own code context), this could be boosted further. So from a capability standpoint, open models can indeed power serious coding assistants in 2025.

Supporting Tools: Besides the raw model, there are open-source projects like **Tabby** (by TabbyML) which provide a self-hosted code completion server. Tabby is essentially an open-source alternative to the backend of Copilot – you run Tabby on a server with a model (they often used CodeGen or similar), and Tabby provides an API and even collects analytics on usage. Tabby comes with editor plugins for VSCode, JetBrains, etc., making it a plug-and-play solution for code completion specifically. A Medium article by an engineer shows how **TabbyML** was used to build an on-prem code reviewer integrated with Bitbucket pipelines. Tabby supported different model backends, and with small models it could run on CPU. While Tabby's own provided model (Tabby's "TabNine-like" model) isn't state-of-art, one can hook it to better ones. So tools like this can shorten the implementation time for an open solution.

In conclusion, using **Hugging Face Transformers and open models** gives maximum freedom: an enterprise can choose the model that best fits their needs (balance between quality and computational cost) and integrate it into their development workflow in a tailored way. This approach is highly suitable for **air-gapped environments** because everything – model weights and execution – resides internally. The main considerations are having the ML expertise to manage these models and the computing infrastructure to serve them to potentially dozens or hundreds of developers simultaneously. As open models continue to improve and approach the capabilities of closed ones, this route becomes increasingly attractive. It offers a way to have an "AI pair programmer" behind your firewall, with no external dependencies and no recurring license fees, at the cost of some upfront engineering. Many large organizations are actively exploring this, piloting internal deployments of models like Code Llama and evaluating their effectiveness. For highly regulated industries, this may soon become the default approach, as it satisfies data sovereignty and compliance while still empowering developers with AI.

Comparison of Solutions and Recommendations

To summarize the above research, we compare the leading enterprise AI code assistant options on key factors relevant to on-premises, air-gapped use. The table below highlights these comparisons:

Solution	On-Premises Deployment	Air-Gapped Support	Hardware/Infra	Languages & IDEs	Notable Features	Benchmarks/Performance
Amazon CodeWhisperer	No (Cloud service only)	**No** – requires internet connectivity to AWS	N/A (Managed by AWS in cloud)	Languages: Python, Java, JS, TS, C#. IDEs: VS Code,	AWS-specific strengths (integrates with AWS	Quality behind best-in-class: ~31% HumanEval (Python) pass@1 (Copilot

Solution	On-Premises Deployment	Air-Gapped Support	Hardware/Infra	Languages & IDEs	Notable Features	Benchmarks/Performance
				JetBrains (via AWS Toolkit)	APIs/services); Security scans (500/month in Pro); Reference tracking for licensing	~46%). Optimized for AWS use cases.
GitHub Copilot (Enterprise)	No (Cloud SaaS)	**No** – cloud-only, though enterprise data not retained	N/A (hosted on GitHub/Microsoft Azure)	Languages: Practically all popular languages. IDEs: VS Code, Visual Studio, JetBrains, Neovim, etc. (broad support).	Deep GitHub integration (PR reviews, CLI assistant, voice chat in IDE); strong general coding ability. Enterprise plan offers seat management and policy controls.	Best-in-class suggestions. Codex/GPT-4 models – ~46% HumanEval (Codex); GPT-4 much higher (~80-85%). Not deployable on-prem.
Tabnine Enterprise	**Yes** – Deploys on Kubernetes on-prem or VPC	**Yes** – designed for fully air-gapped use (no data leaves)	Requires GPUs (e.g. 1–4× NVIDIA A100 for 1000+ users); Kubernetes cluster for services.	Languages: All major languages (trained on permissive OSS). IDEs: VS Code, JetBrains, Visual Studio, Eclipse, etc. (very wide support).	On-prem control of models and data; can fine-tune on private code. Privacy-focused (SOC2, zero telemetry). Additional features: AI chat, codebase indexing for context, test generation.	Good real-world impact: claims 30-50% of code generated by Tabnine for active users. Underlying model ~15B; quality comparable to older Copilot. Integrates open models and can improve with fine-tuning.
Qodo (CodiumAI) Enterprise	**Yes** – supports on-prem and private cloud deployments	**Yes** – explicitly offers air-gapped installation	Requires GPU servers for models; plus indexing engine for code (for RAG). Likely Kubernetes or Docker deployment.	Languages: "Practically every language" (platform agnostic). IDEs: VS Code, JetBrains; also integrates with git platforms (GitHub/GitLab PRs).	Code integrity focus: unit test generation, automatic code reviews with context. Learns org's best practices with continuous learning. Uses Retrieval-Augmented Generation (full codebase indexing) for more informed suggestions.	No public numeric benchmarks. Emphasizes quality improvements (fewer bugs, better coverage) over raw generation stats. Proprietary models can be augmented with others. Real-world: expects to reduce bug rate and enforce standards (qualitative benefits).
Sourcegraph Cody (Self-Hosted)	**Yes** – part of Sourcegraph Enterprise, deployable via Docker/K8s on-prem.	**Yes** – can use local models (via Ollama or other) for offline mode.	Needs Sourcegraph server (which requires indexing storage, CPU/RAM) + a compatible model server (Ollama or	Languages: Any (Sourcegraph indexes all languages' code). Optimized for popular ones (C, C++, C#, Go, JS/TS, Python, etc.).	Unparalleled codebase context: can cite and link to any internal code reference. Chatbot answers with actual code from repos. Multi-repo support. Flexible LLM backend (choose	With local models, quality depends on chosen model (e.g., Code Llama 7B yields basic help; larger models improve accuracy). Using GPT-4 via enterprise connection yields ~85% HumanEval performance (very high), but offline one might use a 13B model (~35-50% performance).

Solution	On-Premises Deployment	Air-Gapped Support	Hardware/Infra	Languages & IDEs	Notable Features	Benchmarks/Performance
			similar) with GPU for LLM.	IDEs: VS Code (official), JetBrains (beta), plus Sourcegraph's web UI.	from OpenAI, Anthropic, local models).	Nonetheless, the rich context often compensates, making answers very relevant.
CodeGeeX (Open-Source)	**Yes** – self-hosted (download model weights and run on local servers).	**Yes** – completely offline after installation.	13B parameter model; runs on a single high-memory GPU (≥15 GB) or dual GPUs (with optimization). Provided int8 quantization for efficiency.	Languages: 20+ languages (Python, C++, Java, JS, Go, etc.) and even cross-language code translation. IDEs: VS Code and JetBrains plugins available.	Fully open-source (Apache-2.0 for code); no cost. Multilingual code generation and translation features. Can be extended or fine-tuned. Community-backed (academic benchmark leader in 2023) github.com.	HumanEval-X benchmark: top performer among open models (as of 2023) github.com. Roughly comparable to older Codex on some tasks; a strong open 13B model. Code Llama may surpass it now, but CodeGeeX is proven and continues to evolve (CodeGeeX2, 4 releases).
Continue.dev (Open-Source)	**Yes** – open-source IDE extension; connect to local or self-hosted models.	**Yes** – can point to models running on internal infrastructure (Ollama, local server).	Depends on chosen model. Continue orchestrates; model runs wherever you configure (local PC or a central server). So hardware could be developer's GPU or a shared inference server (similar to HF Transformers setup).	Languages: not limited (depends on model's training). Out-of-the-box supports typical languages via default prompts. IDEs: VS Code and JetBrains IDEs.	Highly customizable "AI assistant framework." You can integrate any model (OpenAI, Anthropic, local). Offers inline completion, chat, and code editing/refactoring in IDE. Extendable with custom rules and context sources (e.g., incorporate docs, issue trackers) continue.dev. No vendor lock-in.	Performance tied to model used. With top-tier open models (e.g., Code Llama 34B), can achieve ~50-60% HumanEval accuracy – competitive with commercial tools. Lighter models yield lower accuracy but faster responses. Continue itself adds minimal overhead; it's designed for responsiveness and on-device use if needed.
HuggingFace Open Models (StarCoder, Code Llama, etc.)	**Yes** – models are downloadable; enterprise can self-host inference.	**Yes** – entirely self-managed on internal systems.	Varies by model: e.g., **Code Llama 34B** – needs ~4× A100 40GB GPUs; **Code Llama 7B** – can run on a single 16GB GPU or even CPU (slowly). StarCoder 15B – ~1× 30GB GPU (or 2× 16GB).	Languages: StarCoder (15B) trained on 80+ languages; Code Llama specialized (Python, C, Java, PHP, etc.). Broadly, open models cover all common languages. IDE integration: via community plugins or custom (e.g., Tabby,	Full control and visibility into model. No licensing fees (most are permissive or community license). Can fine-tune on proprietary code to improve accuracy for in-house APIs. Large model (70B) performance near state-of-art. Many tooling options (HF Inference server, text-gen	**Benchmark highlights:** Code Llama 70B ~53-68% HumanEval (close to GPT-4), 34B ~54% (above GPT-3.5's 48%). StarCoder ~33% Python (but strong multi-language). With retrieval or fine-tuning, these can be boosted. Open models, when scaled, can closely approach proprietary model performance.

Solution	On-Premises Deployment	Air-Gapped Support	Hardware/Infra	Languages & IDEs	Notable Features	Benchmarks/Performance
				VSCode+HF extension, etc.).	frameworks) to deploy.	

(Sources: as cited inline above – e.g., deployment and support info from 38 32; language/IDE support from 1 7 27; air-gap statements from 15 13; features and performance from 40 43 19, etc.)

From the comparison, a few patterns emerge:

- **Cloud vs On-Prem:** Amazon CodeWhisperer and GitHub Copilot are excellent in capability but are cloud-only. They are not options for truly air-gapped sites. They could be considered in less restrictive enterprises (perhaps those that can allow egress through a proxy and trust the vendor's data handling), but in high-security cases, they're usually disallowed. For such environments, the realistic choices narrow to Tabnine, Qodo, Sourcegraph Cody, or open-source solutions.

- **Enterprise-Ready On-Prem Solutions:** Tabnine and Qodo (CodiumAI) are commercial products built for on-prem. Tabnine is more established, focusing on code completion and privacy, while Qodo is newer, focusing on test generation and code analysis. Both explicitly support air-gap. Tabnine has a track record and a simpler deployment (just one major service to deploy on K8s, essentially), whereas Qodo is a larger platform (with possibly multiple components for analysis, agents, etc.). If an enterprise's main goal is *"Copilot, but offline"*, Tabnine is a strong candidate – it basically offers that, plus some extras like chat and test suggestions, and it's proven in many companies. If the goal is *"improve code quality with AI"*, CodiumAI/Qodo might offer more tailored features (like catching bugs and enforcing standards via AI in the CI pipeline).

- **Sourcegraph Cody:** This is somewhat in between – it's commercial (comes with Sourcegraph licensing), but it leverages open models and an open integration (Ollama). It's very appealing to enterprises that already use Sourcegraph for code search. For those companies, adding Cody gives immediate benefit: devs can ask the codebase questions and get answers with links, something neither Tabnine nor Copilot can do out-of-the-box. The ability to run it fully on-prem with local models is a big plus. A potential downside is that to get the *best* of Cody, one might need connectivity to OpenAI or Anthropic (for GPT-4/Claude), which an air-gapped environment won't have – thus they'll be limited to smaller models unless they invest in hosting big ones. But even with somewhat weaker models, the context injection might allow Cody to produce useful answers (since it cites exact code, the user can piece together the solution). So, for knowledge management and code navigation, Cody is top-notch.

- **Open-Source Approaches:** If budget is a concern or there's a philosophy to avoid vendor lock-in, going open-source is viable. CodeGeeX is a plug-and-play model+extension that an enterprise could test quickly (just install VSCode extension and run the model on a GPU server). Continue.dev is more like a toolkit to integrate AI into dev workflows – it might require more config but offers ultimate flexibility (especially if an org wants to experiment with various models). Using raw Hugging Face models with a tool like Tabby or custom plugins gives fine-grained control. The performance of open models like Code Llama 34B/70B shows that the gap to proprietary is minimal in some cases. One can feasibly achieve ~80% of Copilot's utility with a well-chosen open model and some engineering. The main gap might be polish and supporting features – e.g., Copilot's codebrush/voice or Codium's best-practices learning are additional goodies. But an in-house team can often script around these with open frameworks.

- **Security and Compliance:** All the enterprise-targeted solutions (Tabnine, Qodo, Cody) emphasize that they use models trained on permissive data to avoid legal issues. They often provide or plan indemnification as well. Open-source models often do not come with such guarantees, so if an organization is worried about, say, an AI regurgitating GPL code, they either need to trust the model's dataset filtering or implement controls. It's worth noting that even OpenAI and AWS don't guarantee zero chance of license issues – they mitigate with filters. So in any case, one best practice is to use the AI suggestions as a helper, but have developers review and test everything (which is generally done). Some organizations also restrict AI usage to non-production code or require attribution checking for anything big inserted – these processes can be built internally.

Recommendation by Scenario:

- For a **highly regulated, air-gapped organization** that wants a turnkey solution: **Tabnine Enterprise** is a safe bet. It meets security criteria (fully offline, no data retention), covers all common languages/IDEs, and is relatively easy to deploy with vendor support. Its model is decent and can be supplemented with updates over time (Tabnine continuously improves its local models and can even incorporate better open ones). It also has usage analytics and admin tools that open solutions might lack.

- If the organization also values **test automation and code reviews** heavily, and is open to a newer platform, **Qodo (CodiumAI)** might be evaluated. It could potentially reduce QA burden by generating tests and catching bugs early. One might even use Tabnine and Qodo in tandem – Tabnine for autocompletion, Qodo for PR analysis – as they address different aspects. But that depends on budget and complexity tolerance.

- For an enterprise that already has **Sourcegraph** installed (many large companies do for code search), enabling **Cody** is very logical. It leverages existing infrastructure and adds immediate value, especially for developer onboarding and understanding large codebases. With air-gap, they'd use local models – in practice, they might start with a smaller model and upgrade to larger as they allocate GPUs. Cody's answers with code references can be a game-changer for productivity (no more digging through thousands of files manually).

- For an organization with strong internal tech capabilities and perhaps a preference for **open-source** (or with cost constraints), going with an **open model on HF** plus an extension like **Continue or Tabby** could be the optimal path. This requires more initial setup, but it avoids ongoing fees and allows internal tuning. We've seen anecdotally companies fine-tuning LLaMA models on their code and getting improvements in suggestion relevance. The continued improvement of open models (e.g., the upcoming LLaMA 3, etc.) means this approach could eventually yield equal or better results than proprietary, with full control. The decision here might also weigh on data sensitivity: some orgs simply cannot send any code to third-party (so they choose open solutions by necessity, not just cost).

Usability in Enterprise Workflows: Regardless of the tool, adoption depends on developer acceptance. Tools like Copilot and Tabnine have already shown they can integrate smoothly (as an IDE plugin that unobtrusively suggests code). The open ones (Continue, etc.) also integrate similarly in IDE, so from a user's perspective, the experience can be made almost the same. The difference comes in extended capabilities:

- Copilot has started integrating chat inside IDE, and Codium/Sourcegraph/Continue also offer that. We should ensure whichever solution we pick has a **chat/explain mode** because that's very useful for debugging and learning (Tabnine added chat recently, Cody and Continue have it, etc.).

- Integration with enterprise SSO and user management is another aspect: Tabnine Enterprise ties into SSO and provides admin dashboards, which large companies appreciate. If using open tools, one might not have that out-of-box; however, they might not need it if it's just a tool devs install (like an internal tool distributed via something like an internal VSCode marketplace).

- **Multi-user scaling:** Tabnine, Cody, Qodo presumably handle multiple concurrent requests and have done testing for that. An open solution one must ensure the model server can queue or parallelize requests – using something like TGI (Text Generation Inference server from HF) which is made for multiple users is important. Without it, a naive setup might choke if 50 devs query the model at once. This is a technical detail but crucial for enterprise deployment.

Benchmarks & Evaluations: If possible, an enterprise should conduct a pilot where they measure developer productivity or code quality before and after introducing one of these tools. For example, GitHub's research found 55% faster completion of tasks with Copilot in some studies. Anecdotal evidence from internal hackathons can also show differences between tools. Perhaps try Tabnine vs an open model and see which yields more acceptable suggestions for the team's code. Some large organizations have even built evaluation suites (like giving the AI some common coding tasks from their domain to solve). Ajith's guide suggests using both quantitative metrics (like change lead time, code review feedback, etc.) and qualitative dev satisfaction to measure ROI. The ideal scenario is to select one or two candidates and run them in a controlled pilot with a subset of devs, then compare.

Given this comprehensive study, our overall **recommendation** for an enterprise requiring air-gapped AI coding assistance is:

- **Use a self-hosted solution** – either a commercial on-prem tool like Tabnine Enterprise, or an open-source model integrated with an IDE assistant – to ensure no dependency on internet connectivity and to maintain full control over source code exposure.

- Start with a solution that is easiest to deploy in your environment and evaluate developer adoption. Tabnine, for instance, could be deployed cluster-wide relatively quickly, as could Sourcegraph Cody if you already have Sourcegraph.

- Ensure you provision adequate hardware (GPUs, memory) for the chosen approach; underspecced infrastructure will lead to slow suggestions and frustrated developers. Follow vendor guidelines or model requirements closely.

- Implement complementary practices: even with AI assistance, enforce code reviews and run security scans on AI-written code. Many tools have features to help with this (CodeWhisperer has its scans, Tabnine can detect insecure patterns, Qodo will add tests, etc.). This belts-and-suspenders approach mitigates the risk of AI introducing errors.

- Monitor usage and productivity improvements via internal metrics (Tabnine offers usage analytics, for open tools you might gather stats on suggestion acceptance rates). Solicit feedback from developers regularly – their satisfaction is key to success of these tools, and they may identify configuration tweaks or model changes that can improve results further.

In conclusion, **enterprise AI coding assistants have matured to the point where air-gapped, on-premises deployment is not only possible but already being successfully implemented** in industry. The choice between commercial and open-source depends on specific needs for support,

liability, and features. Enterprises can mix and match (for instance, using an open model with a Sourcegraph front-end, or using Tabnine for completion and CodiumAI for tests). By carefully evaluating the options detailed above and possibly integrating multiple solutions, an organization can empower its developers with AI assistance **without compromising on security or compliance**, effectively getting the benefits of tools like Copilot **within their own isolated environment**.

References

1. Tabnine Team – *"CodeWhisperer: Features, pricing, and enterprise considerations."* (Tabnine Blog, 2023) – Discusses CodeWhisperer vs Copilot, enterprise tier features, and privacy considerations.
2. Tabnine Documentation – *"Enterprise (private installation)"* and *"System Requirements."* (2024) – Details on deploying Tabnine on-premises (Kubernetes, hardware specs) and air-gapped options.
3. Dev.to (Pieces) – *"Best Free and Paid GitHub Copilot Alternatives."* (Jan 2024) – Compares Copilot, CodeWhisperer, Tabnine, noting which can run air-gapped (Tabnine, Pieces) vs which require internet.
4. Yetiştiren et al. – *"Evaluating Code Quality of AI Code Generation Tools (Copilot, CodeWhisperer, ChatGPT)."* (arXiv preprint 2304.10778, revised Oct 2023) – Empirical results showing Copilot outperforming CodeWhisperer in correctness (46.3% vs 31.1% on HumanEval).
5. Sourcegraph Blog – *"Cody — better, faster, stronger."* (Feb 2024 by Ado Kukic) – Announces Cody Enterprise GA, explains local model support via Ollama for offline use and lists languages and features of Cody.
6. Sourcegraph NextJS (NextBuild) – *"Key Differences: Codeium vs Sourcegraph Cody."* (2024) – Confirms Codeium offers on-prem and Sourcegraph Cody works in air-gapped setups with local inference.
7. CodiumAI (Qodo) press release – *"CodiumAI launches generative AI coding platform for enterprises."* (PRNewswire, Jul 10, 2024) – Describes CodiumAI enterprise features: on-prem/air-gapped deployment, RAG indexing of full codebase, learning coding standards.
8. Qodo Website – *FAQ / Qodo vs.* (2025) – States Qodo supports on-premises, VPC, and air-gapped environments and pricing for Enterprise ($45/user with on-prem). Also notes languages "practically every programming language" supported.
9. THUDM – *"CodeGeeX: A Multilingual Code Generation Model."* (GitHub README, 2023) – Documentation of CodeGeeX 13B model, VSCode/JetBrains extensions, single-GPU and quantization support, and performance on multilingual benchmark github.com.
10. Gadgets360 – *"Meta Releases Code Llama 70B… 53% HumanEval."* (Jan 31, 2024) – Reports Code Llama 70B scored 53% on HumanEval, GPT-4 67%, GPT-3.5 48.1%. Confirms Code Llama available for commercial use and hosted on HuggingFace gadgets360.com.
11. Ajith P. – *"AI Code Assistants — Comprehensive Guide for Enterprise Adoption."* (AI Pulse, June 23, 2025) – In-depth analysis of enterprise coding assistants: includes a comparison table of Copilot vs CodeWhisperer vs Tabnine on deployment, security, etc., discusses open-source vs proprietary model trade-offs, and emphasizes on-prem solutions for regulated industries.

12. Continue.dev – *Official Website & Docs.* (2025) – Describes Continue's open-source IDE extensions for VSCode/JetBrains, model integration via Ollama, OpenAI, etc., and features like chat, autocomplete, highlight-and-edit. Emphasizes customization and running "any model, anywhere".

13. Medium (Burak Balım) – *"On-Premise AI Code Reviewer with TabbyML & Bitbucket."* (Feb 10, 2025) – Demonstrates using Tabby (open-source self-hosted code assistant) in an air-gapped pipeline. Explains Tabby runs locally with models (Qwen-0.5B in example) and can be integrated into CI for automated PR comments.

14. Dev.to (NextCompetence) – *"Copilot vs CodeWhisperer vs Tabnine vs Cursor."* (2023) – Notes that "only Tabnine supports true on-premise, fully air-gapped deployments—required in many regulated industries.". Reinforces that Copilot/CodeWhisperer cannot run without internet.

15. VisualStudioMagazine – *"GitHub Copilot Tops Report on AI Code Assistants."* (2023) – Cites a prediction: by 2028, 90% of developers will use AI assistants, up from 14% in early 2024 – highlighting rapid enterprise adoption trends.

combined information from these and earlier citations was used to compile the above comparative analysis.

IntuitionLabs - Industry Leadership & Services

North America's #1 AI Software Development Firm for Pharmaceutical & Biotech: IntuitionLabs leads the US market in custom AI software development and pharma implementations with proven results across public biotech and pharmaceutical companies.

Elite Client Portfolio: Trusted by NASDAQ-listed pharmaceutical companies including Scilex Holding Company (SCLX) and leading CROs across North America.

Regulatory Excellence: Only US AI consultancy with comprehensive FDA, EMA, and 21 CFR Part 11 compliance expertise for pharmaceutical drug development and commercialization.

Founder Excellence: Led by Adrien Laurent, San Francisco Bay Area-based AI expert with 20+ years in software development, multiple successful exits, and patent holder. Recognized as one of the top AI experts in the USA.

Custom AI Software Development: Build tailored pharmaceutical AI applications, custom CRMs, chatbots, and ERP systems with advanced analytics and regulatory compliance capabilities.

Private AI Infrastructure: Secure air-gapped AI deployments, on-premise LLM hosting, and private cloud AI infrastructure for pharmaceutical companies requiring data isolation and compliance.

Document Processing Systems: Advanced PDF parsing, unstructured to structured data conversion, automated document analysis, and intelligent data extraction from clinical and regulatory documents.

Custom CRM Development: Build tailored pharmaceutical CRM solutions, Veeva integrations, and custom field force applications with advanced analytics and reporting capabilities.

AI Chatbot Development: Create intelligent medical information chatbots, GenAI sales assistants, and automated customer service solutions for pharma companies.

Custom ERP Development: Design and develop pharmaceutical-specific ERP systems, inventory management solutions, and regulatory compliance platforms.

Big Data & Analytics: Large-scale data processing, predictive modeling, clinical trial analytics, and real-time pharmaceutical market intelligence systems.

Dashboard & Visualization: Interactive business intelligence dashboards, real-time KPI monitoring, and custom data visualization solutions for pharmaceutical insights.

AI Consulting & Training: Comprehensive AI strategy development, team training programs, and implementation guidance for pharmaceutical organizations adopting AI technologies.

Contact founder Adrien Laurent and team at https://intuitionlabs.ai/contact for a consultation.

DISCLAIMER

The information contained in this document is provided for educational and informational purposes only. We make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability, or availability of the information contained herein.

Any reliance you place on such information is strictly at your own risk. In no event will IntuitionLabs.ai or its representatives be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from the use of information presented in this document.

This document may contain content generated with the assistance of artificial intelligence technologies. AI-generated content may contain errors, omissions, or inaccuracies. Readers are advised to independently verify any critical information before acting upon it.

All product names, logos, brands, trademarks, and registered trademarks mentioned in this document are the property of their respective owners. All company, product, and service names used in this document are for identification purposes only. Use of these names, logos, trademarks, and brands does not imply endorsement by the respective trademark holders.

IntuitionLabs.ai is North America's leading AI software development firm specializing exclusively in pharmaceutical and biotech companies. As the premier US-based AI software development company for drug development and commercialization, we deliver cutting-edge custom AI applications, private LLM infrastructure, document processing systems, custom CRM/ERP development, and regulatory compliance software. Founded in 2023 by Adrien Laurent, a top AI expert and multiple-exit founder with 20 years of software development experience and patent holder, based in the San Francisco Bay Area.

This document does not constitute professional or legal advice. For specific guidance related to your business needs, please consult with appropriate qualified professionals.