Private AI Development Assistant

and Company Know-How Platform

Commercial Offer

Fixed implementation fee	$10,000
Execution timeline	Maximum 3 months
Primary goal	Private AI layer for knowledge, onboarding, tasks, and development
Delivery type	Working pilot with production recommendation

Prepared on May 29, 2026

Pricing examples are indicative and depend on final GPU availability, model size, concurrency, context length, and workload.

Table of Contents

1. Project Summary

We propose building a private AI development assistant for the software agency. The system will work similarly to Claude Code or Codex, but it will be adapted to the company repositories, documents, onboarding process, development standards, task structure, and internal workflows.

The goal is not to build a simple chatbot. The goal is to build a private AI operating layer for the company that helps developers and managers reuse internal knowledge and turn that knowledge into real software delivery flows.

- Process internal documents and turn them into structured company know-how.
- Help onboard developers faster with role-based project learning paths.
- Understand repositories, search code, and explain legacy systems.
- Convert requirements, tickets, and client requests into technical task flows.
- Create implementation plans, acceptance criteria, and test plans.
- Safely edit code through controlled patches, run tests, and prepare pull requests.
- Keep sensitive code and knowledge inside private infrastructure where possible.

2. Main Deliverable

At the end of the 3-month pilot, the company will have a working private AI development assistant that can process documents, build project knowledge, generate onboarding flows, create task flows, understand repositories, apply controlled code changes, run tests, show diffs, and prepare pull requests for human review.

The pilot will not try to replace all developers or fully replace Claude Code or Codex across 120 people from day one. The realistic goal is a strong internal pilot that proves value, benchmarks models on real company work, and gives a clear production rollout roadmap.

3. Technical Architecture

The platform will be composed of six technical layers. This modular design keeps the system model-agnostic, secure, and easier to scale.

```
Users / Developers / Project Managers
        |
        v
AI Web App or CLI Agent
        |
        v
Agent Orchestrator
        |---- Document Retrieval and Company Know-How
        |---- Repository Tools
        |---- Task Flow Engine
        |---- Sandbox and Permission Manager
        |
        v
Internal Model API
        |
        v
Self-hosted Qwen, DeepSeek, Kimi, or other open model
```

3.1 Self-Hosted Model Layer

Open-weight models will be served behind an internal OpenAI-compatible API. The recommended production runtimes are vLLM or SGLang. KTransformers may be evaluated for lower-cost CPU/GPU offload experiments. Ollama can still be used for local prototypes or smaller fallback models, but it is not the main recommendation for production-grade self-hosted Kimi.

- Recommended production runtimes: vLLM or SGLang.
- Cost experiment runtime: KTransformers.
- Prototype runtime: Ollama or llama.cpp for smaller models.
- Internal endpoint example: http://ai-inference.internal:8000/v1/chat/completions.

3.2 AI Development Agent Layer

This is the Claude Code-style layer. The model does not directly control the server. A secure agent orchestrator exposes approved tools, validates actions, executes them safely, and returns results to the model.

- Read tools: list files, read files, search code, find symbols, inspect git status.
- Write tools: apply patches, create files, update files, show diff.
- Validation tools: run tests, run linters, run static analysis, read test output.
- Delivery tools: create branch, prepare commit, prepare pull request after approval.

3.3 Company Knowledge Layer

This layer processes documents and turns them into structured company know-how. It is not only document upload and search. It builds project overviews, architecture summaries, onboarding paths, task templates, and reusable workflows.

3.4 Task Flow Engine

The task engine converts client requests, product requirements, tickets, bug reports, meeting notes, and internal documents into actionable development flows with affected files, acceptance criteria, risks, and validation steps.

3.5 Safe Development Sandbox

Every coding task should run in an isolated workspace or temporary git branch. The agent will not have access to production secrets, production databases, deployment commands, or unrestricted shell commands.

- No .env access.
- No private key access.
- No destructive database commands.
- No production deployment commands.
- Human approval before commit or pull request.
- Full audit log for all actions.

3.6 Admin and Monitoring Layer

The platform will include basic administration and monitoring for user feedback, model usage, GPU usage, failed tasks, indexing status, permission rules, test results, and cost tracking.

4. Document Processing and Company Know-How

This is a core part of the project. The system will not simply upload documents and ask the AI to search them. It will transform scattered company information into structured knowledge that can power onboarding, planning, task creation, code explanations, and delivery workflows.

4.1 Supported Knowledge Sources for the Pilot

- Internal documentation and onboarding guides.
- Technical specifications, architecture documents, API documentation, and deployment guides.
- README files, repository documentation, database schema notes, and test documentation.
- Client briefs, product requirements, project notes, and process documents.
- Jira, Linear, Trello, ClickUp, GitHub Issues, or GitLab Issues if access is provided.
- Pull request descriptions and code review comments.
- Meeting notes if available and approved.

4.2 Document Processing Pipeline

1. Import documents from approved sources.
2. Extract clean text and preserve source references.
3. Detect document type, project, client, module, owner team, and permissions.
4. Split documents into logical sections instead of random chunks.
5. Remove duplicate content and outdated sections where possible.
6. Scan for secrets, credentials, tokens, and sensitive values.
7. Generate embeddings using a local embedding model.
8. Store content, vectors, and metadata in PostgreSQL plus pgvector or Qdrant.
9. Track document freshness and source links.
10. Allow important generated knowledge to be reviewed by a team lead before becoming official.

4.3 Example Knowledge Metadata

```
{
  "source_type": "technical_document",
  "project": "Billing Platform",
  "client": "Internal",
  "document_type": "technical_specification",
  "section": "Subscription renewal process",
  "owner_team": "Backend",
  "permission_group": "backend-team",
  "last_updated": "2026-05-20",
  "source_url": "internal-docs/billing/subscription-renewal"
}
```

4.4 Company Know-How Builder

For each project, the AI will help generate and maintain a structured knowledge pack.

- Project overview and business purpose.
- System architecture summary and main modules.
- Important repositories and important files.
- Local setup guide and common commands.
- Database overview and API overview.
- Deployment process and testing process.
- Coding standards and pull request rules.
- Known risks, common bugs, and developer FAQ.
- Role-based onboarding path.
- Task creation rules and workflow templates.

```
Project: Billing Platform

Purpose:
Handles subscriptions, invoices, payment provider webhooks, renewals, and trial logic.

Main modules:
- Subscription management
- Invoice generation
- Payment webhook processing
- Trial expiration logic
- Admin billing dashboard

Important files:
- app/Models/Subscription.php
- app/Services/BillingService.php
- app/Jobs/ProcessPaymentWebhook.php
- routes/api.php
- tests/Feature/Billing

Developer onboarding:
1. Read billing overview
2. Set up local environment
3. Run billing tests
```

4. Understand payment webhook flow
5. Complete first small billing task

4.5 Human Review and Quality Control

Important generated knowledge should not automatically become official. The recommended workflow is AI generation, team lead review, approval, and then use as official project knowledge. Wrong or outdated sections can be flagged and corrected.

5. Task and Workflow Automation

The system will include reusable workflows for software delivery. These workflows connect documents, tasks, code, and company rules into a practical execution process.

Workflow	Input	Output
New Feature Workflow	Client request, product requirement, internal task, GitHub issue, or Jira ticket	Technical summary, affected modules, development tasks, acceptance criteria, test plan, PR checklist
Bug Fix Workflow	Bug report, error message, failed test, stack trace, or user complaint	Suspected cause, related files, debugging steps, fix plan, regression test, PR summary
Onboarding Workflow	Developer role, project name, technology stack	3 to 5 day onboarding plan, required documents, important repositories, first tasks, local setup, common issues
Pull Request Review Workflow	Pull request diff, linked task, project rules	Summary of changes, risk areas, missing tests, security concerns, documentation updates, review comments
Project Documentation Workflow	Repository, existing docs, code structure	Project overview, architecture summary, local setup guide, deployment notes, common commands, developer FAQ

5.1 Example Task Flow

Input:
Add trial expiration reminder emails 3 days before the subscription ends.

Generated flow:
Business goal:
Notify users before their trial ends to improve conversion.

Technical tasks:
1. Review Subscription model
2. Check trial_ends_at field
3. Add scheduled command
4. Query users whose trial ends in 3 days
5. Add email notification
6. Prevent duplicate reminders
7. Add feature tests
8. Update billing documentation

Acceptance criteria:
- Users receive one reminder 3 days before trial expiration
- Canceled subscriptions are skipped

```
- Duplicate reminders are prevented
- Tests pass
```

6. Claude Code-Style Development Agent

The development agent will work like an internal Claude Code-style assistant. A developer can give it a task, and the agent can inspect the repository, create a plan, make safe code changes, run tests, fix failures, show the final diff, and prepare a pull request.

```
Example command:
ai-code "Add trial expiration emails 3 days before subscription ends"

Agent flow:
1. Create a temporary branch
2. Read project memory
3. Inspect repository structure
4. Search relevant code
5. Read related files
6. Generate implementation plan
7. Ask for approval
8. Apply patches
9. Run tests and linters
10. Read errors and fix failed tests
11. Show final diff
12. Prepare commit or pull request after approval
```

6.1 Project Memory File

Each project can include an AI instruction file, similar to CLAUDE.md, but adapted to this platform. This gives the agent project-specific rules before it starts editing.

```
# Project AI Instructions

## Stack
Laravel 11
PHP 8.3
MySQL
Redis
Vue 3

## Rules
Use services for business logic.
Do not put billing logic in controllers.
Use feature tests for user-facing flows.
Run tests before final answer.
Run code formatter before commit.

## Forbidden
Do not edit `.env`.
Do not edit production config.
```

```
Do not run destructive database commands.
Do not change dependencies without approval.

## Common commands
composer install
php artisan test
vendor/bin/pint
npm run build
```

7. Local Open Models vs Claude Code and Codex

Claude Code and Codex are strong and mature tools. They are usually faster to adopt and easier to use immediately. The private open-model approach has more upfront complexity, but it gives the company deeper control over data, workflow, and internal knowledge.

Option	Pros	Cons
Official tools: Claude Code and Codex	Fast setup; strong model quality; mature developer experience; good repository understanding; no GPU server management; strong terminal, GitHub, and IDE workflows.	Vendor lock-in; ongoing user or usage costs; less control over model and workflow; harder to deeply connect with private company know-how; source code and context may leave company-controlled infrastructure depending on setup.
Self-hosted open models	Private infrastructure option; strong data control; customizable company workflows; can deeply integrate docs, tasks, repos, and onboarding; model-agnostic; can optimize cost at agency scale.	Requires GPU infrastructure; more setup and maintenance; requires benchmarking; may not match best closed models on every task; needs strong sandboxing and security.

7.1 Practical Recommendation

We should not try to beat Claude Code and Codex on day one. The private system should focus on the company-specific advantage: internal documents, repositories, coding standards, onboarding, workflows, task structure, and client delivery processes.

8. Open Model Options

Model	Best Use	Pros	Cons
Qwen	Best practical first candidate for daily coding-agent workflows.	Strong coding model family; good for agentic coding; realistic self-hosting options; good for code explanation, task planning, tests, and patches.	Largest variants still need serious infrastructure; must be benchmarked on company repositories; still needs strong agent orchestration.
Kimi	High-end reasoning and complex agentic tasks.	Strong reasoning; good for complex coding workflows, architecture analysis, large refactors, and long-horizon development tasks.	Expensive to self-host properly; needs multi-GPU infrastructure; more complex deployment; not the first choice for cost-effective daily use across 120 people.
DeepSeek	Strong benchmark candidate for coding and reasoning.	Strong coding ability; good MoE options; useful for bug fixing, code explanation, backend development, and test	Best versions can be heavy to host; behavior depends heavily on tools and prompts; model selection must be tested

		generation.	carefully.

8.1 Recommended Model Strategy

The architecture should be model-agnostic. The pilot should test Qwen for practical daily coding, DeepSeek for code reasoning and bug fixing, and Kimi for complex agentic development and architecture tasks. The final production recommendation should be based on quality, cost, latency, and developer adoption.

9. RunPod Server Cost Examples for 120 People

These are infrastructure examples, not final quotes. They are based on publicly listed RunPod on-demand pod prices checked on May 29, 2026. The real cost depends on active concurrency, selected model, context length, tokens per request, repository size, workload type, and whether the system runs 24/7 or only during working hours.

For 120 people, we should not assume all 120 are running heavy AI coding tasks at the exact same second. A more realistic pilot assumption is 12 to 30 active concurrent sessions during busy periods, with queue-based execution for heavier jobs.

Setup	Example Use Case	Hourly Cost	Approx. Monthly Cost
1x L40S	Small pilot, 5 to 15 active users	$0.86/hr	~$628/mo
2x L40S	Better pilot, 10 to 25 active users	$1.72/hr	~$1,256/mo
1x RTX Pro 6000	Stronger single-GPU pilot	$2.09/hr	~$1,526/mo
2x RTX Pro 6000	Practical agency pilot	$4.18/hr	~$3,051/mo
4x RTX Pro 6000	Strong agency setup	$8.36/hr	~$6,103/mo
4x H100 PCIe	High-end coding server	$11.56/hr	~$8,439/mo
8x H200	Serious Kimi-style setup	$35.12/hr	~$25,638/mo
16x H200	Large Kimi-style setup	$70.24/hr	~$51,275/mo
8x B200	Premium high-end setup	$47.12/hr	~$34,398/mo

9.1 Practical Infrastructure Recommendation

- Cost-controlled pilot: 2x L40S, around $1,256/month, good for document processing, smaller coding models, and limited concurrency.
- Stronger pilot: 2x RTX Pro 6000, around $3,051/month, better for coding models, more users, and better context handling.
- High-end benchmark: 4x H100 PCIe, around $8,439/month, useful for larger model benchmarking and more realistic agency workloads.
- Kimi heavy setup: 8x H200 or more, from around $25,638/month, only recommended if the company accepts a high inference budget.

10. Execution Plan

Period	Focus	Deliverables
Month 1	Foundation and Knowledge Processing	Architecture setup; model runtime setup; first benchmark; document ingestion pipeline; first vector database; first company knowledge structure; first repository import; document Q&A with source references; security rules for blocked files and secrets.
Month 2	Task Flows and Development Agent	Company know-how builder; project overview generation; onboarding flow generation; task breakdown workflow; bug-fix workflow; pull request review workflow; repository search; file reading; planning mode; safe workspace; patch generation; git diff output.
Month 3	Coding Pilot and Handover	Controlled file editing; test execution; linter execution; error analysis; test-fix loop; pull request preparation; admin review workflow; model comparison report; RunPod cost recommendation; developer usage guide; handover session.

11. Scope Included and Not Included

11.1 Included in the $10,000 Pilot

- Architecture design and model selection strategy.
- Self-hosted model runtime setup guidance.
- Model benchmarking setup for Qwen, DeepSeek, and Kimi.
- Document processing pipeline and knowledge base structure.
- Project know-how generation and onboarding flow generation.
- Task breakdown workflow, bug-fix workflow, and pull request review workflow.
- Repository search and code understanding.
- Claude Code-style agent prototype.
- Safe sandbox rules, blocked paths, and command restrictions.
- Patch and diff workflow, test runner integration, and PR preparation workflow.
- RunPod infrastructure recommendation, technical documentation, and final handover.

11.2 Not Included

- GPU server monthly costs and cloud storage costs.
- Large production cluster management.
- Fine-tuning custom models or training a foundation model from scratch.
- Full enterprise SSO implementation unless agreed separately.
- Full migration of all company documents and all repositories.
- Production rollout to all 120 users.
- Guaranteed replacement of Claude Code or Codex.
- Unlimited support after delivery.

12. Success Criteria

The pilot will be considered successful if the system can demonstrate the following on real company material.

- Process and search internal documents with source references.
- Build project-specific know-how from documents and repositories.
- Generate onboarding flows for developers.
- Generate structured task flows from requirements and tickets.
- Search and explain code in real repositories.
- Create implementation plans with affected files and acceptance criteria.
- Apply controlled code changes inside a safe workspace.
- Run tests and show final diffs.
- Prepare pull requests for human review.
- Compare Qwen, DeepSeek, and Kimi on real company tasks.
- Provide a clear production infrastructure and cost recommendation.

12.1 Final Positioning

This project is not about installing a local model. It is about building a private AI development and knowledge platform for the agency. The platform combines self-hosted open models, company document processing, structured know-how, onboarding flows, task flow automation, repository understanding, safe code editing, test execution, and pull request preparation.

13. Source Notes

The following sources were used to validate current model and infrastructure facts. Pricing and model capabilities can change, so production decisions should be re-checked before infrastructure commitment.

- **RunPod GPU pricing:** https://www.runpod.io/pricing - Used for current hourly GPU pricing examples.
- **MoonshotAI Kimi K2 GitHub:** https://github.com/MoonshotAI/Kimi-K2 - Used for Kimi K2 model size and agentic capability description.
- **Qwen3-Coder GitHub:** https://github.com/QwenLM/Qwen3-Coder - Used for Qwen3-Coder agentic coding model description.
- **DeepSeek-Coder-V2 GitHub:** https://github.com/deepseek-ai/deepSeek-Coder-V2 - Used for DeepSeek-Coder-V2 model sizes and active parameters.
- **Claude Code product page:** https://claude.com/product/claude-code - Used for Claude Code capabilities and pricing context.
- **OpenAI Codex cloud documentation:** https://developers.openai.com/codex/cloud - Used for Codex repository and pull request workflow context.
- **OpenAI Codex GitHub repository:** https://github.com/openai/codex - Used for Codex CLI context.