AI-Powered Code Development Architecture Guide

📋 Questions

1. **Migration from Anthropic to ChatGPT**: If I am using anthropic and I want to migrate the artifacts to chatgpt, how to do this? Is it easy? Does it need some mappings? Are there some models which are compatible? I want this to happen because later when maintenance is done I will have to export these and feed it to a small learning model as a knowledge base for easy maintenance.
2. **UI vs API Token Usage**: If I use UI and API does it make any difference in number of tokens in models? Do I get more tokens if I do not use UI? I know I have more control but does it make it less expensive to use if I use APIs directly?
3. **Multiple Agents Shared Memory**: What is Multiple agents shared memory? => LangGraph
4. **Claude Project Memory via API**: How to mimic claude project via claude api? Like it still has some memory from old chat but it's a new one.
5. **Best AI Integration for Code Development**: Best AI integrated for code development with code editor like VS Code.

🏗️ Architecture Overview

Core Problem Statement: Multiple devs will come and use the AI integrated code development in VS Code or somewhere on a website. All the calls will go to a Virtual machine and then will call the model. The model maybe a paid model like OpenAI, Anthropic or a opensource self deployed model like LLaMA or DeepSeek. The response of the model will be saved and also given back to the devs. The saved response can be starred and exported later as a knowledge base for maintenance.

System Flow Diagram

```
IDE (VS Code / Web IDE) => Continue.dev
   |
   v
Request sent to VM backend (FastAPI/LangChain server)
   |
   v
Route to selected model (OpenAI, Anthropic, LLaMA, etc.)
   |
   v
Return response to user + log it (LangSmith, DB)
   |
   v
Allow user to:
    - Star it
    - Add tags
    - Export later to KB
   |
   v
Use these as knowledge base for maintenance
```

Enhanced Architecture

```
VS Code or Web IDE (Continue.dev / Theia)
      |
      v
Central AI Gateway (FastAPI / LangChain)
      |
      ├──> Model Router (OpenAI / Claude / LLaMA)
      └──> Memory Store (Redis + Postgres + Vector DB)
      └──> Logging + Metadata Extraction (Lang Graph)
      |
      v
Logging & KB Export (LangSmith / Supabase / Notion Export)
```

Technology Stack

Component	Tool	Description
IDE	Continue.dev	Open-source AI pair programmer. Works in VS Code and has backend flexibility.
Model backend	LangChain + FastAPI	Custom service routing prompts to OpenAI, Anthropic, or self-hosted LLMs like LLaMA.
Response logging + metadata	LangSmith or Supabase/Weaviate	LangSmith can log all interactions and let users inspect/star/export.
Knowledge base	Markdown or Notion export, or vector DB (e.g., Qdrant, Weaviate)	Store starred answers for future RAG.
User management	Basic auth/session tracking	Custom or integrate with an auth provider like Firebase/Auth0.

🔧 Alternative Tools Research

Open-Source Alternatives to Continue.dev

🧩 1. CodeGeeX 2.0

- **GitHub**: https://github.com/THUDM/CodeGeeX2
- **Features**:
  - AI code completion, chat, and instruction following
  - Works with LLaMA, StarCoder, DeepSeek, or OpenAI APIs
  - VS Code extension available
  - Good multi-language support
- **Pros**: Open-source, model-agnostic, chat + inline coding
- **Cons**: Less polished UI/UX than Continue.dev

🧩 2. Tabby

- **Website**: https://tabbyml.github.io
- **GitHub**: https://github.com/TabbyML/tabby
- **Features**:
  - Self-hosted code completion engine
  - Compatible with OpenAI or self-hosted LLMs
  - Can be used in VS Code and JetBrains IDEs
- **Pros**: Optimized for privacy; great for internal enterprise deployment
- **Cons**: Not conversational like Continue; mainly for autocomplete

🧩 3. CodeGPT (by Daniel San)

- **GitHub**: https://github.com/daileb/codegpt
- **Features**:
  - Works inside VS Code
  - Supports OpenAI, Azure, Anthropic, etc.
  - Chat, explain, refactor, test generation
- **Pros**: Lightweight and configurable
- **Cons**: Not multi-user or memory-aware

🧩 4. OpenDevin (experimental, but promising)

- **GitHub**: https://github.com/OpenDevin/OpenDevin
- **Features**:
  - Agent-style dev assistant (think: self-acting AI devs)
  - Can operate via CLI or browser
  - Includes memory, workspace, and multi-step reasoning
- **Pros**: Multi-agent like AutoGPT, integrated code environment
- **Cons**: More for experimentation, less production-ready

🧩 5. Continue.dev (reference)

- **GitHub**: https://github.com/continuedev/continue
- **Features**:
  - Multi-model, works with Claude, GPT, DeepSeek, LLaMA, etc.
  - Built-in memory, sidebar UI, VS Code + JetBrains support
  - Custom prompt templates, routing, and logs

🧩 6. Phind (Partially Open Source)

- **Website**: https://www.phind.com
- **Focus**: Code search + LLM chat + docs
- **Use**: Can be embedded in a dev environment with plugins

Comparison Summary

Tool	Editor Support	Chat-Based	Multi-Model	Memory	Open Source	Suitable For
Continue.dev	VS Code, JetBrains	✅	✅	✅	✅	Best all-rounder
CodeGeeX	VS Code	✅	✅	❌	✅	General-purpose
Tabby	VS Code, JetBrains	❌	❌	✅	✅	Autocomplete use cases
CodeGPT	VS Code	✅	✅	❌	✅	Basic chat + tasks
OpenDevin	Browser / CLI	✅ (agent-style)	✅	✅	✅	Research and dev automation
Phind	VS Code (via web/CLI)	✅	✅	Partial	Partial	Docs + coding focused

💡 Expert Recommendations

Startup-Scalable, Modular Approach

Here's a comprehensive answer to each of the 5 questions, tailored to the core architecture goal:

1. Anthropic to ChatGPT Artifact Migration

Problem: You use Claude (Anthropic) today and want to switch later to ChatGPT/OpenAI — will your saved conversations, prompts, or artifacts transfer cleanly?

Answer: Yes, but you'll need lightweight normalization.

Solution:

- Save all interactions in a standard format:

json

```json
{
  "user": "What's the time complexity of merge sort?",
  "model": "O(n log n)",
  "timestamp": "2025-06-07T10:00:00",
  "model_used": "Claude-3",
  "tags": ["algorithms", "interview"],
  "starred": true
}
```

- Avoid vendor-specific memory structures or metadata
- Use your own JSON schema or logging structure
- Later, just re-route it to GPT-4-Turbo or any other model

Claude and GPT may differ in how system prompts or memory context works, but raw Q&A can be reused easily for:

- RAG
- Fine-tuning
- Retrieval-based memory (like LangChain + vector DB)

Tool Tip: Use a central memory service (Redis or Postgres) to manage stored artifacts — not tied to Claude or GPT internally.

2. UI vs API — Token Control & Cost

Problem: Does using API directly give more control or reduce token usage compared to UI?

Answer: Yes. API gives you full control and is cheaper.

Mode	Token Limit	Cost	Control
Web UI (ChatGPT/Claude)	~128K tokens	Fixed pricing (e.g. $20/mo)	Low
API (GPT/Claude)	Up to 128K (Claude), 128K (GPT-4 Turbo)	Pay-per-token	High

Recommendation:

- Use API only in your app
- Budget: Claude Opus ≈ $15/1M input tokens, GPT-4 Turbo ≈ $10/1M input
- You can:
  - Compress context
  - Choose which parts to send
  - Avoid sending full history every time

Pro Tip: Use token trimming + chunking tools (like LangChain's `TokenTextSplitter`) to optimize payloads.

3. What is Shared Memory with Multiple Agents?

In **LangGraph**, "multiple agents shared memory" means:

- You have multiple LLM-powered agents doing different things (e.g., Refactorer, BugFixer)
- They all read/write to a shared memory (like a knowledge graph, or a dict in Python)

Example:

python

```python
shared_memory = {
    "code_context": "def add(a, b): return a + b",
    "user_goal": "optimize code",
    "last_response": "Try using numpy for speed."
}
```

Each agent gets this memory and updates it.

Use in your case:

- Let each dev session be one LangGraph instance
- Shared memory = project context + starred suggestions
- Helps in chaining actions (e.g., ask, fix, refactor, document)

4. Mimic Claude Project-like Memory Using API

Claude UI feels like it "remembers" chats because it's storing past threads. You can mimic this:

Solution: Maintain a session store:

json

```json
{
  "session_id": "user1-20250607",
  "history": [
    {"role": "user", "content": "How to fix this bug?"},
    {"role": "assistant", "content": "Try changing the index..."}
  ]
}
```

Then pass history in the `messages` payload when calling Claude/GPT:

json

```json
"messages": [
  {"role": "user", "content": "..."},
  {"role": "assistant", "content": "..."},
  ...
]
```

Use **Redis** or **Postgres** to store and query this memory.

5. Best AI for Code in VS Code (Lean & Scalable)

Best Option: Continue.dev

Feature	Why it's Good
VS Code integration	✅ Native support
Supports multiple backends	✅ OpenAI, Claude, LLaMA
Extendable & open-source	✅ You can plug your own API server
Central routing possible	✅ Add middleware in your VM

Implementation:

- Dev installs Continue extension in VS Code
- You configure it to send API calls to your central VM
- VM uses a FastAPI + router:
  - `/api/ask`: routes to GPT or Claude
  - `/api/save`: logs Q&A
  - `/api/star`: marks favorites
- Store responses in Postgres/JSON + vector store for RAG/export

🚀 Overall System Summary (Lean Startup Phase)

```
[VS Code / Web UI]
        ↓
  [Continue.dev Plugin]
        ↓
   [Your Central VM Backend - FastAPI]
        ↓
    ┌──────────────┬──────────────┬─────────────┐
    │  OpenAI API  │  Claude API  │ LLaMA Local │
    └──────────────┴──────────────┴─────────────┘
        ↓
  [Save Responses in DB]
        ↓
[Star/Tag/Export to KB or VectorDB]
```

✅ Final Recommendations

- Use **Continue.dev + FastAPI** for lean MVP
- Start with **Claude or OpenAI**, but **log and normalize everything** so migration is painless
- Build a **modular logging + export pipeline** early — it will serve both **maintenance** and **RAG/chatbot use**
- Keep **APIs over UIs** for cost control and flexibility

📚 Learning Path

Key Technologies to Master

1. **LangSmith**: For logging + metadata extraction which will be used by SLMs in maintenance. LangSmith can log all interactions and let users inspect/star/export.
2. **LangGraph**: For Shared Memory Across Multiple Agents

Next Steps

Would you like a sample `FastAPI` backend for routing, saving, and starring responses as a starting template?

Recommendation for Your Use Case: If you're building a multi-user, centralized AI code assistant with logging and knowledge base:

- Start with **Continue.dev**
- For autocomplete-focused usage: add **Tabby**
- For future automation (e.g., multi-agent workflows): experiment with **OpenDevin**