AI-Accelerated VS Code Development

A practical workflow for combining **Claude** and **ChatGPT** inside Visual Studio Code to accelerate development while maintaining security, reliability, and code quality. This blueprint covers everything from VS Code configuration and extension setup to multi-agent conflict prevention and security best practices.

While the examples in this guide focus on Claude Pro and ChatGPT Plus, the same workflow can be applied to other AI subscriptions and models as well. The specific VS Code configuration, extensions, or authentication methods may vary depending on the provider, but the underlying principle remains the same: running multiple AI systems in parallel to divide tasks, cross-verify outputs, and accelerate development while preserving code quality and safety.

Hybrid AI Model

Specialised tools for specific tasks

VS Code Config

Optimised settings for AI workflows

Dev Workflow

Plan, implement, test, verify

Security

Safe practices for AI-assisted code

SENTIDO

SOFTWARE MADE SIMPLE

1. Hybrid AI Development Model

Instead of relying on a single AI tool, this setup uses **specialised models for specific tasks**. The core principle is simple: use the best model for the job. Claude Opus excels at deep reasoning across large codebases, whilst ChatGPT Codex delivers fast, structured output for repetitive or generative tasks.

When using **ChatGPT Plus**, developers can maintain continuous development conversations without worrying about a strict cap on the number of chat interactions during normal use, which makes it practical to keep a persistent development thread open while iterating on tasks inside VS Code. This allows ChatGPT to act as a fast execution assistant alongside Claude's deeper reasoning capabilities.

Claude

- Architecture & refactoring large repositories
- Debugging complex multi-file logic
- Tracing issues across dependencies

ChatGPT

- Boilerplate generation
- Unit test coverage generation
- Clean structured documentation

Both Models

- Code review via cross-validation
- Reduces hallucination risk
- Mutual verification of complex logic

By assigning each model to its area of strength, the workflow reduces errors, speeds up delivery, and produces more reliable output than any single-model approach.

2. VS Code Configuration

These settings remove friction and optimise the environment for AI-assisted development. The configuration below is designed to keep AI models focused on meaningful project code by excluding noise, and to provide a comfortable terminal workspace for agent command execution.

```
{
  "claudeCode.preferredLocation": "panel", // Keeps Claude UI docked: quick access
  "claudeCode.selectedModel": "opus", // Uses strongest reasoning model
  "security.workspace.trust.enabled": true, // Allows extensions to run safely
  "terminal.integrated.cwd": "${workspaceFolder}", // Terminal in project root
  "terminal.integrated.defaultLocation": "editor", // terminals as editor tabs
  "terminal.integrated.enablePersistentSessions": false, // Prevent stale terminal
  "search.smartCase": true, // Improves code search accuracy
  "search.useIgnoreFiles": true, // Respects .gitignore to avoid noise
  "extensions.autoUpdate": true, // Keeps AI extensions current
  "files.watcherExclude": {
    "**/.git/objects/**": true, // Prevents watcher overhead
    "**/node_modules/**": true, // Ignores large dependency folders
    "**/bin/**": true, // Skips compiled binaries
    "**/.terraform/**": true // Ignores Terraform state cache
  },
  "files.exclude": {
    "**/bin": true, // Hides build artifacts
    "**/obj": true, // Hides .NET intermediate files
    "**/.terraform": true // Hides Terraform working files
  }
}
```

Directory Exclusion

Removes build artefacts and dependencies from indexing so AI models focus only on meaningful project code, not compiled output or third-party packages. Extend the directories based on the tools you tend to use.

Terminal Inside Editor

`terminal.integrated.defaultLocation` to `"editor"` provides a large terminal workspace where AI agents can run commands and display execution output clearly alongside the code.

SENTIDO

SOFTWARE MADE SIMPLE

3. Extensions Used

Two primary extensions power this workflow, each mapped to its respective AI model. Both are available on the VS Code Marketplace and integrate directly into the editor panel for seamless side-by-side use.

Claude Code — Anthropic

Available at the VS Code Marketplace (anthropic.claude-code).

Used for repository reasoning, multi-file edits, architectural changes, and debugging complex logic across large codebases.

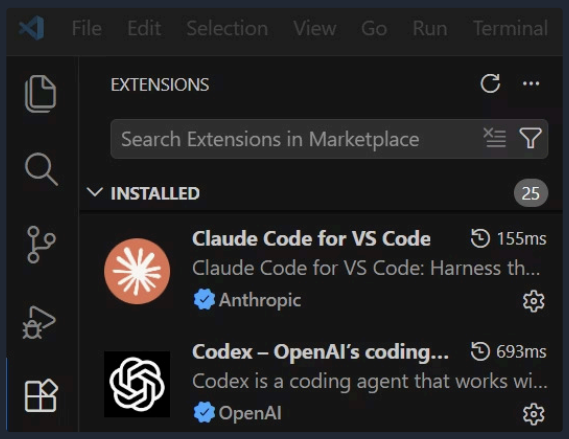
Codex — OpenAI Extension

Available at the VS Code Marketplace (openai.chatgpt).

Used for boilerplate generation, documentation, test generation, and quick questions that don't require deep repository context.

> Both extensions should be listed under `extensions.confirmedUriHandlerExtensionIds` in your settings to prevent repeated permission prompts during AI-assisted sessions. **usually automatic**, but not guaranteed.

```
"extensions.confirmedUriHandlerExtensionIds": [
  "anthropic.claude-code",
  "openai.chatgpt"
]
```



SENTIDO
SOFTWARE MADE SIMPLE

4. Typical Development Workflow

The four-step workflow below structures how Claude and ChatGPT are engaged at each phase of a feature's lifecycle — from initial planning through to cross-verification. Each step assigns the right model to the right task.

Step 1 — Feature Planning

Claude analyses the repository and identifies impacted modules, dependencies, and potential breaking changes. Example prompt: *"Analyse this repository and propose the safest implementation strategy for feature X."*

Step 2 — Implementation

Claude handles architectural edits and multi-file refactors. ChatGPT handles helper functions, configuration files, documentation, and small utilities.

Step 3 — Unit Tests

ChatGPT generates tests covering edge cases and error conditions. Typical output includes xUnit tests, mocks, and integration scaffolding.

Step 4 — Cross Verification

Claude generates complex logic; ChatGPT reviews and augments it. Example prompt: *"Review this code and identify potential bugs or edge cases."* This cross-validation step significantly reduces hallucination risk.

When using multiple AI tools within the same repository, avoid assigning tasks that modify the same files at the same time. Overlapping edits can cause conflicting changes, overwritten work, or unstable code states. Instead, separate responsibilities so each model works on distinct files, modules, or clearly defined areas of the project.

---

5. Separating Claude Configurations

If you maintain separate Claude subscriptions — for example, personal and work — it is useful to isolate configuration, history, and prompts, and guarantee your maximising your tokens in both spaces.

This can be achieved using the `CLAUDE_CONFIG_DIR` environment variable, which tells Claude where to store its configuration for each context.

Personal Environment

```
powershell.exe -WindowStyle Hidden
-Command "$env:CLAUDE_CONFIG_DIR=
'C:\Users\{USERNAME}\.claude-personal';
code"
```

Work Environment

```
powershell.exe -WindowStyle Hidden
-Command "$env:CLAUDE_CONFIG_DIR=
'C:\Users\{USERNAME}\.claude-work'; code"
```

Isolated Memory

Prevents sensitive work code from appearing in personal conversations and vice versa.

Improved Relevance

Each environment's prompts and history are tuned to the specific repository, improving response quality.

Context Separation

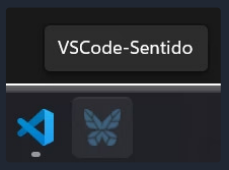
Professional and personal contexts remain cleanly separated, reducing the risk of cross-contamination.

Custom ICO Icons for VS Code Claude Shortcuts

To make environments easier to distinguish, custom Windows shortcuts can be created using the PowerShell commands above. Assign a custom `.ico` icon to each shortcut and pin them to the Windows taskbar. This allows you to immediately see which VS Code environment you are launching without reading the label.

Work brand icon → Work Claude environment

Personal brand icon → Personal Claude environment



SENTIDO

SOFTWARE MADE SIMPLE

6. AI Personalisation

Adding clear response preferences to each model improves consistency across sessions.

ChatGPT Preferences

Recommended preferences:

- Respond concisely, prefer bullet points
- Avoid repeating input back
- Output final code blocks for modifications

Where to configure

ChatGPT allows persistent instructions through **Custom Instructions**.

Location:

- **ChatGPT → Profile Menu → Settings → Custom Instructions**

Add the preferences under:

- **“How would you like ChatGPT to respond?”**

Example configuration:

```
Respond concisely.
Prefer bullet points.
Do not repeat the input prompt.
For code changes, output final unified code blocks.
Avoid unnecessary explanations unless requested.
```

These instructions are automatically applied to **all conversations**, including those used during development workflows.

Claude Preferences

Recommended preferences:

- Be concise
- Output only modified files
- Use minimal diff
- Avoid unnecessary explanation

Where to configure

For Claude Code or the Claude VS Code extension, persistent behaviour is typically defined through a **local configuration prompt file** stored in the Claude configuration directory.

Location example:

```
C:\Users\{USERNAME}\.claude\system-prompt.md
```

Or inside the isolated configuration directories if you are using environment separation:

```
C:\Users\{USERNAME}\.claude-work\
C:\Users\{USERNAME}\.claude-personal\
```

Example system prompt:

```
Be concise.
Output only modified files.
Prefer minimal diffs.
Avoid explanations unless explicitly requested.
Prioritise correctness and maintain project style conventions.
```

Because Claude reads configuration from the **CLAUDE_CONFIG_DIR**, these prompts remain isolated between work and personal environments.

---

7. Context Management & Beating the Claude Limit

Offloading Generic Questions

AI context is valuable — don't waste it on generic questions. Use free browser tools for non-repository queries.

Use Google Gemini for:

- Research and conceptual explanations
- General knowledge queries

Reserve Claude / ChatGPT for:

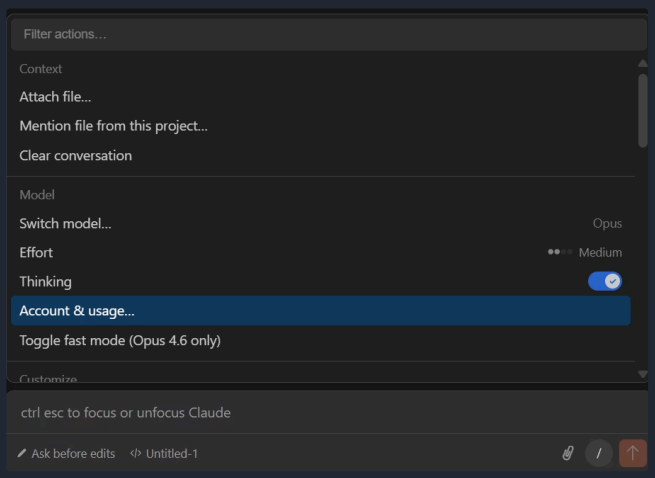
- Repository debugging and architecture
- Code generation and review

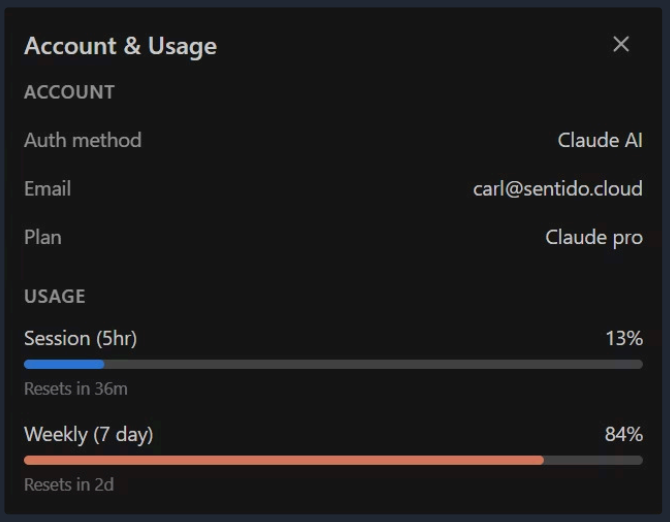
Beating the 5-Hour Claude Wall (Optional)

Claude usage can hit temporary limits. Some developers queue tasks to resume after the reset window using CLI tooling.

```
sleep_until "15:00" && claude -c -p
"Finish the unit tests we started"
```

Community tools are also sometimes used, though this workflow **optional and experimental** — use with caution and verify any third-party tooling before adoption.





8. Security, Sensitive Data & Automation Risks

Never expose sensitive information to AI systems. This is a non-negotiable principle regardless of how trusted the tool appears. The developer must remain the **final approval gate** for all AI-generated changes.

Never Share With AI

- API keys and production credentials
- Database passwords and SSH private keys
- Customer data of any kind

Best Practices

- Redact secrets before pasting code
- Use environment variables for all credentials
- Avoid copying production configuration into prompts

Always Review AI Actions

- Shell commands run by agents
- Dependency and package changes
- Infrastructure scripts and edits

AI tools may run terminal commands, modify files, and generate infrastructure scripts. These capabilities are powerful but require human oversight at every step. **Never allow AI changes to bypass Continuous Integration (CI) checks.**

9. Preventing Conflicts When Running Multiple AI Agents

When running Claude and ChatGPT in parallel or serial workflows, simple project hygiene prevents agents from overwriting each other's work or generating conflicting changes.

File Ownership Strategy

Claude: `/src/core/` — architecture, multi-file refactors, dependency restructuring

ChatGPT: `/src/utils/`, `/tests/`, `/docs/` — helpers, tests, documentation

Branch Isolation

Run AI-generated work inside feature branches:

- feature/ai-tests
- feature/ai-refactor-auth
- feature/ai-docs-update

Lock Files

If two models may touch the same area, use a coordination file at `/ai-work/active-files.md` listing which agent is editing which file.

Atomic Tasks

Prefer small, sequential prompts (generate interface → implement logic → write tests) over large end-to-end requests. Atomic tasks keep AI output predictable.

Diff-Based Edits

Prompt for minimal diffs rather than full file rewrites: *"Return only the changed lines required to fix this bug."* Easier to review and reduces overwrite risk.

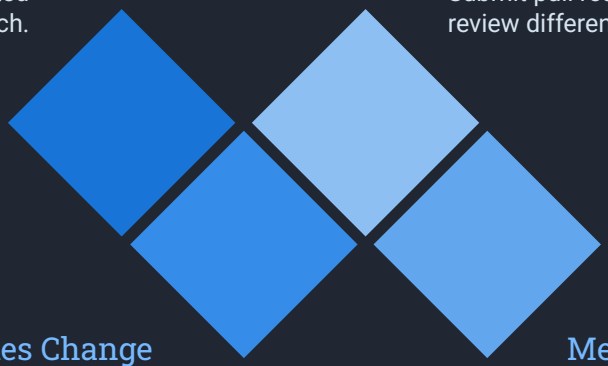
Directory Boundaries

Constrain agents to specific directories in prompts.

"Analyse only /src/auth." "Generate tests only for /src/auth/tokenService.ts."

Pull Requests as Safety Gate

Even for solo developers: AI generates change → commit to branch → open PR → review diff → merge. Pull requests provide the final safety checkpoint before any AI change reaches main.



Commit to Branch

Save changes on an isolated feature branch.

Open PR and Review Diff

Submit pull request and review differences.

AI Generates Change

Agent proposes code or content edits.

Merge to Main

Resolve conflicts and integrate into main.

Legal Disclaimer

> **The information in this document represents general advice and personal workflow practices. Security and architectural decisions should always be validated within your own environment.**

By implementing any techniques described in this document, you acknowledge that:

Your Responsibility

You are responsible for verifying the security implications of any configuration or workflow adopted from this document.

Code Validation

You are responsible for validating all AI-generated code before it is deployed to any environment, production or otherwise.

Consequences

You are responsible for the consequences of implementation. Proceed with caution.

Sentido and its author accept no liability for damages or issues arising from the use of the information contained in this document.