ChatGPT

Tool Survey: AI Coding CLIs and Their Capabilities

Claude Code CLI (Anthropic) – An agentic command-line coding assistant [1]. The official repository (**anthropics/claude-code** [1]) and docs describe Claude Code as a natural-language interface that understands a codebase and can execute routine coding tasks (e.g. generate functions, refactor code, run tests) [1]. It handles Git workflows (e.g. diff, commit, merging) via built-in tools and plugins [1]. Claude Code itself is cross-platform (supports macOS, Linux, Windows) and language-agnostic; it can read/edit code in Python and PowerShell just as any other file (the repo even contains Python and PowerShell code) [2]. **Orchestration:** Claude Code is single-threaded and interactive – it only runs one agent/task at a time in a given session. (Parallel usage requires running multiple independent instances, e.g. in separate worktrees or containers.) **Git Integration:** Native – Claude Code can run Git commands (e.g. `git log`, `git diff`, branching) as tools [1]. **Limitations:** Requires internet access to Anthropic's API (no offline mode); context length limits; and, as noted by users, "you can only run one task at a time" without external orchestration [3].

Gemini CLI (Google) – The open-source **google-gemini/gemini-cli** brings Google's Gemini LLM to your terminal [4]. It is a Node/TypeScript REPL that provides "lightweight access" to Gemini, with built-in tools for file I/O, shell commands, web search, etc. [5] [6]. In practice it can query or edit multi-file codebases (including Python or PowerShell scripts) by issuing file-system read/write commands to the LLM. **Orchestration:** Like Claude, it runs one interactive session at a time (no inherent parallelism) but supports "non-interactive" script mode and checkpointing. **Git Integration:** Indirect – Gemini CLI itself does not embed Git operations, but Google provides a [Gemini CLI GitHub Action] for CI workflows [7]. The CLI can execute shell/git commands if prompted (via its shell tool), but it has no special merge/worktree features. **Limitations:** Uses Google's cloud API (free tier limits apply) and thus requires internet; model-specific context window (up to 1M tokens on Gemini 2.5 Pro) [5]; no offline mode without a local Gemini server.

Ollama CLI – The **ollama/ollama** tool is a local LLM host and manager [8]. It is *not* itself a coding agent but a CLI to download, run, and manage open-source models (Llama, Phi, DeepSeek, Gemma, etc.) locally [8]. Ollama supports offline use of code-capable LLMs (e.g. DeepSeek, CodeLlama). It provides commands to create/pull models and run them via a local REST API [9]. **Supports Python/PowerShell:** Ollama itself doesn't "edit" code, but you can run any model that can. For example, it can load DeepSeek-R1 which excels at code editing [10]. **Orchestration:** Ollama can serve multiple models concurrently (via `ollama run` sessions), but it has no built-in multi-agent orchestration – it's essentially a model runtime. **Git Integration:** None – it treats files as generic data. **Limitations:** Requires substantial local resources (models use many GB of RAM/disk) [8]; initial model downloads can be large; local-only (no cloud fallback).

OpenAI Codex CLI – The **openai/codex** tool (Docs: codex/cli [11]) is an open-source Rust-based coding agent similar to Claude Code [12]. It runs locally from the terminal and "can read, modify, and run code on your machine" [12]. Codex CLI supports macOS/Linux (Windows via WSL) [13]. It provides an interactive UI to ask natural-language prompts; it will parse code files, apply edits, and even execute commands on the host. This means it can handle Python or PowerShell scripts as well as other languages (it is language-agnostic). **Orchestration:** Codex is interactive by default. It also offers a non-interactive "`codex exec`" mode for scripted tasks [14]. It does not natively schedule parallel tasks, though one can invoke multiple CLI processes separately. **Git Integration:** No special Git features – it can of course run Git commands if the underlying LLM is prompted to do so, but there is no built-in merge/workflow logic. **Limitations:** Requires internet/ChatGPT account (OpenAI API) with appropriate plan (Pro/Team for Codex) [15]; cannot run offline; context limited by model used.

Aider (with DeepSeek via Ollama) – Aider is an open-source CLI "AI pair programmer" written in Python [16]. It integrates with many LLMs (OpenAI, Anthropic, Gemini, etc.) and can also use local models via Ollama [17]. Aider is explicitly designed to **edit code in a local Git repo** – it maps your codebase, suggests edits, and auto-commits changes with sensible messages [18]. It supports "100+ code languages: python, javascript, rust, ..." [19] (Python is 80% of its codebase [20]). It does not specifically target PowerShell, but since it uses LLM edits it can generate any text-based code. **Orchestration:** Aider runs one conversation/ session per invocation; no built-in parallel tasking. **Git Integration:** Strong – Aider automatically creates commits, and you can use standard Git commands to review or revert AI changes [18]. **Offline:** Yes, if you use Aider with a local model (e.g. DeepSeek via Ollama) it can function entirely offline. **Limitations:** Depends on model quality; weaker models (e.g. free GPT-3.5) may fail to produce proper code edits [21]. Its Git commits use the running HEAD; it has no special merge strategy.

GitHub Copilot CLI – The **github/copilot-cli** brings GitHub Copilot's coding agent to the terminal [22]. It is Node-based, supports Linux/macOS/Windows [23], and authenticates to a Copilot subscription. Copilot CLI can build, edit, debug, and refactor code with natural-language commands [24]. It "knows your repositories, issues, and pull requests" for context [25], since it integrates with GitHub's ecosystem. **Supports** all code languages (undoubtedly Python and PowerShell included), as it functions like an AI collaborator; it can even use PowerShell prompts on Windows. **Orchestration:** The agent runs interactively; it is described as agentic and can plan/execute complex tasks [24]. It has no built-in job queue or parallelism beyond manual spawn of sessions. **Git Integration:** Deep – Copilot CLI is "GitHub-native": it auto-authenticates with your GitHub account, can fetch issues/PRs, and uses the Git repo context. It runs entirely in your local folder and can create branches, make commits, and open pull requests (with explicit user approval) [26]. It supports a Model Context Protocol (MCP) server for custom tools. **Limitations:** Requires a paid Copilot subscription; no offline mode; still in public preview (features may change).

Pipeline Architecture (Modular, Headless, Deterministic)

The proposed pipeline is **manifest-driven and Git-centric**. A high-level architecture is: a **Pipeline Orchestrator** parses a structured input file (e.g. a YAML change plan) and spawns **isolated workstreams** for each task via Git worktrees. Each workstream is a separate branch in the same repo (checked out into its own directory) [27]. The orchestrator dispatches subtasks to the most suitable tool agents:

- **Planning Tasks:** Use Claude Code or Copilot CLI for high-level planning, outline generation, or branching strategies (both can discuss or enumerate tasks with natural prompts). Gemini CLI could serve similar query/analysis roles. These tools refine the change plan, splitting it into sub-steps.
- **Editing Tasks:** Use Aider (with a chosen LLM) to implement code changes (it auto-commits), or Codex CLI/Copilot CLI to write/debug code. For Python/PowerShell scripts, Aider is ideal (it's proven for code edits) [28]; Copilot CLI or Claude Code can be fallback editors. Ollama (with DeepSeek) can be used in offline mode or as a fast local model via Aider.

- **Validation Tasks:** Use shell tools (even Claude or Gemini) to run linters/tests as steps. These can be part of each workstream's flow.
- **Merge/QA Tasks:** Once a workstream completes, automated rules (per the deterministic merge policy) merge it back to the main branch. The orchestrator triggers the deterministic merge train to combine branches in a reproducible way [29] .

Key aspects: - **Parallel Workstreams via Git Worktrees:** Each subtask is executed in a separate worktree/branch, providing full isolation of file states [27] . As shown in [57], "Git worktrees solve [context switching] by letting you run multiple … sessions in parallel, each with its own isolated context" [30] . This allows true simultaneous work (e.g. feature-X and feature-Y branches can be worked by different agents concurrently, each with a full code context). - **Deterministic Merges:** We use a policy-based merge-train (per .merge-policy.yaml) as in the Deterministic Git Workflow [31] . Merges and rebases apply pre-configured strategies by file type (JSON, YAML, lockfiles, etc.), run tests/lints, and only push when all gates pass [32] [33] . All merge decisions are logged in structured JSONL for audit [34] . This ensures the pipeline is fully reproducible ("free of arbitrary human variation" [29]) and transparent.

Example pipeline architecture showing orchestrator, parallel worktrees, AI agents (Claude/Copilot/Aider), and Git merge train (audit trail).

Tool-to-Task Mapping

The following matrix summarizes each tool's strengths for common pipeline tasks. A checkmark (✓) indicates a strong fit for that task, and citation references are given for relevant capabilities.

Task	Claude Code	Gemini CLI	Aider (Ollama)	Codex CLI	Copilot CLI	Ollama (DeepSeek)
High-level planning	✓ [1]	✓ (chat queries)	– (primarily editing)	✓ (text prompts)	✓ [24]	– (model host)
Code generation	✓ [1]	✓ (via chat)	✓ (edits code)	✓ (edits code)	✓ (edits code)	Model (edits via Aider)
Code editing (Python)	✓ (any code)	✓ (any code)	✓ [28]	✓ (any code)	✓ (any code)	✓ (via local model)
Code editing (PS)	✓ (any code)	✓ (shell tool)	○ (not specialized)	✓ (any code)	✓ (any code)	✓ (via local model)
File operations	✓ (file I/O)	✓ [35]	–	✓ (shell tool)	–	✓ (local file access)
Task orchestration	–	–	–	–	– (interactive)	–
Git operations	✓ (built-in)	–	✓ [18]	–	✓ (integrated)	–

Task	Claude Code	Gemini CLI	Aider (Ollama)	Codex CLI	Copilot CLI	Ollama (DeepSeek)
Offline capability	–	–	✓ (with local model) [17]	–	–	✓ [8]

Table: Tools vs. tasks (✓=strong fit; model=through local LLM). For example, Aider excels at making commits of code edits [18], while Copilot CLI is strong at leveraging GitHub context and planning [24]. Ollama itself is a model host (so used indirectly via Aider or CLI models).

Execution Flow

The pipeline runs in **phases**:

1. **Input & Planning:** The orchestrator reads a structured change plan (e.g. YAML or JSON manifest) specifying tasks and target files. It may prompt Claude Code or Copilot CLI to *expand or verify* this plan (e.g. break a large change into smaller subtasks) [24] [1]. The result is a set of parallel sub-tasks (e.g. "add feature X to `module.py`", "update script for config Y").

2. **Spawn Workstreams:** For each sub-task, create a new Git branch and check it out into a separate worktree directory. This isolates the environment: each worktree sees only its branch's files, while sharing the common `.git` history [27]. (For example, `worktree/feature-X` on branch `feature/X`, `worktree/feature-Y` on `feature/Y`.)

3. **Parallel Task Execution:** In each worktree, the orchestrator invokes tools as needed:

4. **Editing agents:** e.g. run `aider --model deepseek --api-key ...` (Aider) or `copilot chat` in that directory to implement code changes. The agent reads context and returns edits which are applied to files. Each tool may prompt for approval before committing.
5. **Language support:** Ensure that any Python or PowerShell tasks (scripts or modules) are handled by the agent. For example:

```
cd worktree/feature-X
# If the task is to write a PowerShell script, use Copilot CLI:
copilot ask "Write a PowerShell script to ... " --allow-scripts
# Or use Aider if using a local model:
aider --model deepseek "Refactor this Python function..."
```

6. **Testing/linting:** After edits, run linters or tests (via shell or CLI tool) to verify correctness. Agents may self-test code if configured (Copilot CLI can run tests as part of a conversation).
7. **Commit Changes:** Each agent commits its changes with a natural-language message (Aider does this automatically [18]). This produces one or more commits on the worktree branch.

Example: Suppose the plan says *"Update* `compute.py` *to use new API."* The orchestrator might launch Aider in the `feature/update-compute` worktree. Aider produces a patch updating `compute.py` and commits it with a message “Refactor `compute.py` to call new API endpoint.” The branch now contains the updated code.

1. **Merge Train:** Once all workstreams finish, the pipeline performs an automated merge train (using the deterministic merge rules [29]):
2. Each feature branch is rebased/merged onto `main` in a defined order (e.g. by priority or queue order).
3. Custom merge drivers (for JSON/YAML) and rerere are used to resolve trivial conflicts according to policy [32] [36] .
4. After merging each branch, post-merge checks run (schema validation, tests) [33] .
5. All actions (decisions, conflict counts, policies applied) are logged as JSONL for audit [34] .
6. If a merge fails policy or tests, the branch is quarantined for manual review (and the JSONL logs record this).
7. **Completion:** Successful merges update the main branch. The pipeline reports results (e.g. via console or writing to a log file). The process is fully headless and repeatable: the same manifest yields identical commits and merges.

Git Integration Strategy

We leverage Git *worktrees* for isolation [27] . Each parallel workstream gets its own worktree directory with its own checked-out branch [27] . This preserves each AI agent’s context and prevents “context loss” when switching branches [30] . The merged `.git` repository remains the single source of truth. Our merge policy (in `.merge-policy.yaml` and `.gitattributes`) enforces deterministic strategies by file type [32] . For example, JSON/YAML changes can use structure-aware merges, lockfiles may use “theirs” strategy, etc. The pipeline’s PowerShell scripts (or Python) configure Git (git config rerere, custom drivers) exactly as specified in the repository [37] , ensuring local and CI merges behave identically. All merges are done via the CI “merge-train” workflow [38] so that the integration is fully automated. Audit logs (JSONL) track every merge step [34] .

Execution Flow Example

Input (manifest):

```
tasks:
  - id: TID-001
    branch: feature/add-logging
    description: "Add logging to data_processor.py"
    files: ["src/data_processor.py"]
  - id: TID-002
    branch: feature/fix-config
    description: "Update config.ps1 defaults"
    files: ["scripts/setup/config.ps1"]
```

The orchestrator reads this, then spawns two branches/worktrees: `feature/add-logging` and `feature/fix-config`.

Task TID-001 (Python): The orchestrator runs `aider` in the `feature/add-logging` worktree. Aider uses DeepSeek to edit `data_processor.py`, adding logging statements where needed. Aider applies and commits the patch.
Output (commit in branch): "Enable detailed logging in data_processor.py (added logger.info calls)."

Task TID-002 (PowerShell): The orchestrator runs Copilot CLI in the `feature/fix-config` worktree with a prompt to update default values. Copilot CLI edits `config.ps1` accordingly and commits "Change default config values for stability.".

After these, the merge-train rebases each branch, merges with `main`, and runs tests. Since no conflicts arose, both merges succeed and `main` is updated with the new logging code and config script.

Fallback Logic

To ensure resilience, the orchestrator checks tool availability and degrades gracefully: - If **Copilot CLI** is unreachable or no license, fall back to **Claude Code** (Anthropic) or **Gemini CLI** for task planning and edits. - If **Claude**/cloud models have latency issues, fall back to **Aider** with a local model (DeepSeek via Ollama) for critical edits (since Aider can run offline) [17]. - If **Aider** (local model) fails to produce coherent edits, the pipeline can switch to **Codex CLI** or **Copilot CLI** if available. - If network/API is down, the pipeline runs only local components (Ollama/Aider) and flags tasks that require cloud. For example, try `ollama run deepseek` to generate text if cloud is off. - All fallback decisions are logged. The pipeline ensures that at least one agent (cloud or local) is attempted for each subtask before marking it failed. - Errors in one branch do not block others: each workstream is isolated, so one agent's unavailability only affects that branch.

Throughout, the pipeline remains **deterministic and auditable**: every action (agent used, model choice, fallback invoked) is recorded. Using Git worktrees and scripted CI merges ensures that the process can be replayed exactly as configured.

Sources: Official tool docs and repos provide these details (e.g. Claude Code CLI on GitHub [1], Gemini CLI docs [4], Ollama README [8], Aider GitHub [16], OpenAI Codex docs [12], Copilot CLI README [22]), as well as best-practice analysis of Git workflows [29] [27]. Each citation above links to the authoritative source.

---

[1] [2] GitHub - anthropics/claude-code: Claude Code is an agentic coding tool that lives in your terminal, understands your codebase, and helps you code faster by executing routine tasks, explaining complex code, and handling git workflows - all through natural language commands.
https://github.com/anthropics/claude-code

[3] How to run Claude Code in parallel | Ona (formerly Gitpod)
https://ona.com/stories/parallelize-claude-code

[4] [5] [7] [35] GitHub - google-gemini/gemini-cli: An open-source AI agent that brings the power of Gemini directly into your terminal.
https://github.com/google-gemini/gemini-cli

[6] Welcome to Gemini CLI documentation | Gemini CLI
https://geminicli.com/docs/

[8] [9] GitHub - ollama/ollama: Get up and running with OpenAI gpt-oss, DeepSeek-R1, Gemma 3 and other models.
https://github.com/ollama/ollama

[10] [17] [21] Connecting to LLMs | aider
https://aider.chat/docs/llms.html

[11] [12] [13] [15] Codex CLI
https://developers.openai.com/codex/cli

[14] GitHub - openai/codex: Lightweight coding agent that runs in your terminal
https://github.com/openai/codex

[16] [18] [19] [20] [28] GitHub - Aider-AI/aider: aider is AI pair programming in your terminal
https://github.com/Aider-AI/aider

[22] [23] [24] [26] GitHub - github/copilot-cli: GitHub Copilot CLI brings the power of Copilot coding agent directly to your terminal.
https://github.com/github/copilot-cli

[25] GitHub Copilot CLI · GitHub
https://github.com/features/copilot/cli

[27] [30] Mastering Git Worktrees with Claude Code for Parallel Development Workflow | by Dogukan Uraz Tuna | Medium | Medium
https://medium.com/@dtunai/mastering-git-worktrees-with-claude-code-for-parallel-development-workflow-41dc91e645fe

[29] [31] [32] [33] [34] [36] [37] [38] Deterministic_Git_Workflow_System.md
file://file-4smWtsQKR9yucTemjoGVrr