Unified AI Dev Setup (macOS + WSL2) — VS Code + Roo Code, Gemini Code, Claude Code, OpenCode (Sept 2025)

This is a single, cross-platform guide that merges three setup documents into one pragmatic workflow for **remote API–first AI development** with an **upgrade path** to local models and GPU acceleration. It targets:

- **Novices** (simple, safe, remote-only installs)
- **Advanced users** (Docker, GPU, Ollama / LM Studio / vLLM, model routing via LiteLLM)

Focus: Visual IDE (**VS Code with Roo Code**) + Terminal assistants (**Gemini Code, Claude Code, OpenCode/Crush**), with **direct APIs** (Gemini, OpenAI, Claude) and **aggregators** (OpenRouter, Requesty). Demos included.

---

0) Target Architecture (remote-first, local-optional)

One interface (OpenAI-compatible) for everything.

- **Editor layer**: VS Code (+ Roo Code) for planning + edits; Terminal assistants (Gemini Code, Claude Code, OpenCode/Crush) for repo-wide tasks.
- **API layer** (choose one):
- **Direct vendors**: Gemini, OpenAI, Anthropic (Claude)
- **Aggregators**: OpenRouter, Requesty (single key → many models)
- **Local router**: LiteLLM Proxy (optional). Gives you one URL with **fallbacks, budgets, logs**; points to direct vendors/aggregators/local servers.
- **Local models (optional)**: LM Studio (port `1234`), Ollama (port `11434`), vLLM (OpenAI-compatible server).

Start **remote-only**. Add **router** later. Add **local models** when you need offline/privacy control.

---

1) Prerequisites (macOS & WSL2 Ubuntu 22.04)

macOS (Apple Silicon)

1. **Homebrew** → install from brew.sh.
2. **Node 20+**: `brew install fnm && fnm install --lts && fnm use --lts`
3. **Python tools**: `curl -LsSf https://astral.sh/uv/install.sh | sh`
4. **direnv**: `brew install direnv` → add to `~/.zshrc`: `eval "$(direnv hook zsh)"`
5. **Docker Desktop** (for containers). *(Optional now, required for vLLM containers later)*

Windows 11 (WSL2 + Ubuntu 22.04)

1. **WSL 2**: in PowerShell (Admin) → `wsl --install` → choose **Ubuntu 22.04**.
2. In Ubuntu: `sudo apt update && sudo apt install -y direnv git curl`.
3. **uv** (Python): `curl -LsSf https://astral.sh/uv/install.sh | sh`
4. **fnm** (Node): `curl -fsSL https://fnm.vercel.app/install | bash` → `exec $SHELL` → `fnm install --lts && fnm use --lts`
5. **Docker Desktop** on Windows → Settings ▸ Resources ▸ **Enable WSL integration** for Ubuntu. *(For GPU: install NVIDIA drivers on Windows; verify `nvidia-smi` inside WSL.)*

   Keep repos under `~/code` **inside WSL** for speed (avoid `/mnt/c`).

---

2) Secrets & per-project environment

Use `direnv` for per-repo secrets and to keep keys **out of git**.

```
cd ~/code/my-ai-project
printf '%s\n'
  'export GEMINI_API_KEY="..."'
  'export OPENAI_API_KEY="..."'
  'export ANTHROPIC_API_KEY="..."'
  'export OPENROUTER_API_KEY="..."'
  'export REQUESTY_API_KEY="..."'
  > .envrc

direnv allow
```

Add `.envrc` / `.env*` to `.gitignore`. Prefer 1Password/Bitwarden CLIs for injecting secrets when possible.

---

3) Install the IDE & extensions (Visual)

VS Code (host OS; use Remote – WSL on Windows)

- **Must-have extensions**:
- **Roo Code** (agentic coding; OpenAI-compatible providers)
- **Remote – WSL** (Windows only)
- **GitLens**, **Docker**, **Dev Containers**, **Prettier/Black**, **Markdown All in One**

Configure Roo Code to use different backends

You can point Roo Code at: - **Direct**: OpenAI / Gemini / Anthropic (if the extension supports) - **OpenAI-compatible**: **OpenRouter**, **Requesty**, **LiteLLM**, **LM Studio** (`http://localhost:1234/v1`), **Ollama** (`http://localhost:11434/v1`)

In Roo Code settings: - **Provider**: *OpenAI-compatible / Custom* - **Base URL**: e.g., `https://openrouter.ai/api/v1` or `https://api.requesty.ai/v1` or `http://localhost:4000/v1` (LiteLLM) - **API key**: leave blank if the extension reads env vars; otherwise paste a **throwaway workspace key**.

> Create **VS Code Profiles** (e.g., *Roo@OpenRouter*, *Roo@LiteLLM*, *No-AI*) to avoid conflicts.

---

4) Install terminal assistants (CLI)

> Use **one** terminal assistant at a time per repo to avoid stepping on each other's edits.

4.1 Gemini Code (Gemini CLI)

- **Install** (macOS/WSL):

```
# Node 20+ required
npm install -g @google/gemini-cli   # or: brew install gemini-cli (macOS/Linux)
```

- **Auth**: either OAuth flow or API key

```
# OAuth mode (browser login)
gemini
# API key mode
export GEMINI_API_KEY=... && gemini
```

- **Common uses**:
- `gemini -m gemini-2.5-flash` (pick model)
- `gemini -p "Explain this repo's architecture"`
- Project context: run `gemini` in repo root, include dirs via `--include-directories`.

4.2 Claude Code (CLI)

- **Install** (requires Node):

```
npm i -g @anthropic-ai/claude-code
```

- **Auth**:

```
export ANTHROPIC_API_KEY=...
claude code  # opens interactive assistant in current repo
```

- **Handy flags**: `--diff` to review changes; `--apply` to write patches after preview.

4.3 OpenCode (status & alternative)

- **OpenCode** is **archived** upstream and superseded by **Crush** (Charmbracelet). If you still need OpenCode, follow its archived README to install from the last release.
- **Recommended**: use **Crush** (active successor) for a similar terminal coding experience.
- **Install** (macOS/WSL): see Crush README for Homebrew/Linux install; then:

```
crush
```

- **Usage**: open in repo root; ask for multi-file refactors, tests, or docs. Keep `git` clean to review diffs.

  You can run these CLIs **inside VS Code's Terminal** to pair with Roo Code without fighting the editor.

---

5) Configure providers — direct & aggregators

Direct vendors

- **Gemini**: set `GEMINI_API_KEY` (personal or Vertex).
- **OpenAI**: set `OPENAI_API_KEY`.
- **Anthropic (Claude)**: set `ANTHROPIC_API_KEY`.

When the tool asks for an "OpenAI-compatible" endpoint but you're using a direct vendor SDK, point the tool at the vendor's native setting (if supported). Otherwise, route via **LiteLLM** (below) or use an aggregator.

Aggregators

- **OpenRouter**
- Base URL: `https://openrouter.ai/api/v1`
- Key: `OPENROUTER_API_KEY`
- Notes: wide model catalog (GLM/Qwen/DeepSeek/etc.), simple model swapping.
- **Requesty**
- Base URL: `https://api.requesty.ai/v1`
- Key: `REQUESTY_API_KEY`
- Notes: open-source router + platform; per-key routing & controls; OpenAI-compatible.

  Best practice: Start with an aggregator in early prototyping for fast model A/Bs; switch to direct vendor for production if needed for price/SLA.

---

6) Optional: Local router (LiteLLM) to unify everything

Run a local **LiteLLM Proxy** so IDEs/CLIs talk to **one** URL while you route to direct vendors, aggregators, or local servers (LM Studio/Ollama/vLLM). You also get **fallbacks**, **budgets**, and **metrics**.

```
uv tool install litellm
cat > ~/litellm.yaml <<'YAML'
model_list:
  - model_name: chat-default
    litellm_params:
      model: google/gemini-2.5-pro
      api_key: ${GEMINI_API_KEY}
  - model_name: code-fast
    litellm_params:
      model: openrouter/deepseek/deepseek-coder
      api_key: ${OPENROUTER_API_KEY}
      api_base: https://openrouter.ai/api/v1
  - model_name: local-llama
    litellm_params:
      model: openai/llama-3.1-8b-instruct
      api_base: http://127.0.0.1:11434/v1
      api_key: dummy
router_settings:
  fallback_strategy:
    - primary: chat-default
      fallbacks: [code-fast]
  budget:
    per_user_usd_monthly: 10
YAML

litellm --port 4000 --config ~/litellm.yaml --num_workers 2
```

- Point Roo Code / CLIs at `http://localhost:4000/v1` with `OPENAI_API_KEY=local`.
- Swap routes centrally without changing your tools.

---

7) Optional: Local models

LM Studio (GUI; OpenAI-compatible on port `1234`)

1. Install the app; download a model.
2. Enable **Developer/Local Server**; note the base URL `http://localhost:1234/v1`.
3. In Roo Code/CLIs, set **Base URL** to that address; any string works as a dummy key.

Ollama (daemon; OpenAI-compatible on `11434`)

```
# macOS: brew install ollama ; Windows: winget install Ollama.Ollama
ollama serve
ollama pull llama3.1:8b
# Use via OpenAI-compatible endpoint: http://localhost:11434/v1
```

vLLM (OpenAI-compatible server; best on Linux+NVIDIA)

- **WSL2 GPU**: install NVIDIA drivers on Windows; verify `nvidia-smi` in WSL; run with Docker:

  ```
  docker run --gpus all -p 8000:8000 vllm/vllm-openai:latest
    --model meta-llama/Llama-3.1-8B-Instruct
  # Base URL: http://localhost:8000/v1
  ```

- macOS: CPU-only; prefer LM Studio/Ollama for Metal acceleration.

  You can also route LM Studio/Ollama/vLLM through **LiteLLM** for unified budgets, logs, and fallbacks.

---

8) Demos (requested flows)

Demo A — OpenCode at the CLI (with stable fallback)

> Because **OpenCode is archived**, we show the **Crush** successor flow; adapt to OpenCode if you must (commands may differ per archived README).

```
# In your repo root
crush
# Examples:
# 1) Generate tests for a module
#   "Write unit tests for src/utils/date.ts with vitest; follow existing
patterns."
# 2) Multi-file refactor with preview
#   "Refactor config loading into /src/config/, add zod schema, keep API back-
compat."
# 3) Docs
#   "Create a CONTRIBUTING.md with setup, run, test, and release steps."
# Review git diffs; commit selectively.
```

Demo B — VS Code with Roo Code (OpenAI-compatible backend)

1. Install Roo Code; open project.

2. Settings ▸ Provider: **OpenAI-compatible**; Base URL = **OpenRouter** `https://openrouter.ai/api/v1` (or your **LiteLLM** `http://localhost:4000/v1`).
3. Set key in env (`OPENROUTER_API_KEY=...`) so Roo reads it.
4. Use Roo's *Plan* → *Edit* → *Review* workflow for repo-wide changes.
5. For long tasks, run one agent (Roo) + one terminal assistant (e.g., Gemini Code) to avoid conflicts.

Demo C — Gemini Code as a complement inside VS Code

Open **VS Code Terminal** in project root:

```
# Start Gemini Code with OAuth (browser pops up)
GEMINI_API_KEY= gemini  # or just `gemini` and choose OAuth
# Analyze repo, then ask for a patch plan
#   "Summarize architecture and propose a safe migration from requests to
httpx."
# Apply edits stepwise; review diffs in VS Code Source Control.
```

> Tip: Keep Gemini Code for planning/explanations and Roo Code for surgical file edits, or vice-versa. Avoid two agents editing the same files concurrently.

---

9) Direct vs Aggregator — Practical Tradeoffs

Aspect	**Direct (Gemini / OpenAI / Claude)**	**Aggregator (OpenRouter / Requesty)**
Setup	Multiple keys, vendor-specific settings	One key / endpoint for many models
Cost	Often best at scale with committed spend	Small markup common; cheap alt models available
Features	Latest vendor features fastest	Feature parity varies by provider
Latency/ SLA	Fewer hops, clearer support	Extra hop; routing/quotas can vary
Portability	Medium (SDK ties)	High (swap models with config)
Governance	Vendor dashboards	Centralized budgets/rate-limits (or use **LiteLLM**)

Recommendation: Prototype with an **aggregator**; for production, either keep aggregator with **LiteLLM** controls, or switch critical paths to **direct** vendors for stability/SLA.

---

10) Security & isolation

- **Workspace Trust**: Keep unknown repos in **Restricted Mode** until you review.

- **Secrets**: `.envrc` + password-manager CLI; never commit secrets; enable **GitHub secret scanning / push protection**.
- **Process isolation**: use **Dev Containers** for risky repos; mount read-only if needed.
- **Key hygiene**: per-project keys with minimal scopes; rotate regularly; disable keys in CI logs.
- **Agent permissions**: grant terminal/file write access **explicitly** and review proposed diffs before applying.

---

11) Troubleshooting

- **Roo Code can't auth** → ensure base URL & key match provider; some extensions need app restart after env changes.
- **Aggregator 401/429** → check key; try another model; add **LiteLLM** fallbacks.
- **WSL2 GPU missing in Docker** → update Windows NVIDIA driver; `wsl --update`; verify `nvidia-smi` in WSL; run test image with `--gpus all`.
- **Slow Windows I/O** → keep repos under WSL's ext4, not `/mnt/c`.
- **Port collisions** → LM Studio (1234), Ollama (11434), LiteLLM (4000), vLLM (8000) are defaults; change if needed.
- **Two agents editing same files** → serialize: use terminal assistant for planning, Roo for edits (or vice-versa). Commit often.

---

12) Scaling & maintenance

- **Router first**: put **LiteLLM** in front early → one URL, budgets, logs, retries, fallbacks, rate limits.
- **Reproducible envs**: `uv` (Python) + `fnm` (Node) + **Dev Containers** in repo.
- **Templates**: a course/research skeleton with `.devcontainer/`, `pyproject.toml`, `.node-version`, `Makefile`, `.envrc.example`.
- **Local model path**: LM Studio → Ollama → vLLM (remote Linux GPU) without changing client code (OpenAI-compat endpoints).
- **Data & models**: use **Hugging Face Hub** for artifacts; pin versions/refs.

---

13) Real-world workflow patterns

- **Teaching lab**: Instructor hosts a small **LiteLLM** router with per-student budgets & model allow-lists; students point Roo Code + CLIs at the router URL; heavy tasks routed to a remote vLLM; light tasks hit Gemini/GLM via aggregator.
- **Research notebook → code**: Prototype prompts locally with **LM Studio** (OpenAI-compat @ `1234`) to avoid cloud costs; once ready, flip base URL to **OpenRouter** or direct vendor; keep tests deterministic by pinning model + temperature; record prompts in repo.

Quick Reference (copy/paste)

Env vars

```
export GEMINI_API_KEY=...
export OPENAI_API_KEY=...
export ANTHROPIC_API_KEY=...
export OPENROUTER_API_KEY=...
export REQUESTY_API_KEY=...
```

Local endpoints

```
LM Studio:  http://127.0.0.1:1234/v1
Ollama:     http://127.0.0.1:11434/v1
LiteLLM:    http://127.0.0.1:4000/v1
vLLM:       http://127.0.0.1:8000/v1
```

CLI installs

```
# Gemini Code (Gemini CLI)
npm i -g @google/gemini-cli   # or: brew install gemini-cli
# Claude Code (CLI)
npm i -g @anthropic-ai/claude-code
# Crush (successor to OpenCode)
# See Crush README for brew/apt install, then: `crush`
```

VS Code Roo Code → OpenAI-compatible

```
Base URL: (choose one)
- OpenRouter: https://openrouter.ai/api/v1
- Requesty:  https://api.requesty.ai/v1
- LM Studio: http://localhost:1234/v1
- Ollama:    http://localhost:11434/v1
- LiteLLM:   http://localhost:4000/v1
```

You now have a single, upgradable workflow: remote APIs for novices, with clean on-ramps to routing (LiteLLM), local models (LM Studio/Ollama/vLLM), and reproducible MLOps. Pair **Roo Code** in VS Code with one **terminal assistant** (Gemini Code / Claude Code / Crush) for fast, auditable development.