IDE and tool integrations

- Prerequisites
- Cline (VS Code)
  - Configuration
  - Troubleshooting Cline
- Continue (VS Code / JetBrains)
  - Configuration
  - Using Ollama provider
- Cursor
  - Configuration
- Zed
  - Configuration
- Open WebUI
- Aider
  - Configuration
- LangChain
  - Python
  - JavaScript/TypeScript
- LlamaIndex
- OpenCode
  - Configuration
- Claude Code
  - Configuration
- Common issues
  - "Connection refused" errors
  - "Model not found" errors
  - Slow responses or timeouts
  - CORS errors (web-based tools)
- Recommended models by use case

- What's next

Docker Model Runner can serve as a local backend for popular AI coding assistants and development tools. This guide shows how to configure common tools to use models running in DMR.

Prerequisites

Before configuring any tool:

1. Enable Docker Model Runner in Docker Desktop or Docker Engine.
2. Enable TCP host access:
   - Docker Desktop: Enable **host-side TCP support** in Settings > AI, or run:

     ```
     $ docker desktop enable model-runner --tcp 12434
     ```

   - Docker Engine: TCP is enabled by default on port 12434.
3. Pull a model:

   ```
   $ docker model pull ai/qwen2.5-coder
   ```

> [!TIP]
>
> The default context size for many models (such as `gpt-oss`) is 4,096 tokens, which is limiting for coding tasks. You can repackage it with a larger context window:
>
> ```
> $ docker model pull gpt-oss
> $ docker model package --from ai/gpt-oss --context-size 32000 gpt-oss:32k
> ```
>
> Alternatively, models like ai/glm-4.7-flash, ai/qwen2.5-coder, and ai/devstral-small-2 come with 128K context by default and work without repackaging.

Cline (VS Code)

Cline is an AI coding assistant for VS Code.

Configuration

1. Open VS Code and go to the Cline extension settings.
2. Select **OpenAI Compatible** as the API provider.
3. Configure the following settings:

Setting	Value
Base URL	`http://localhost:12434/engines/v1`
API Key	`not-needed` (or any placeholder value)
Model ID	`ai/qwen2.5-coder` (or your preferred model)

> [!IMPORTANT] The base URL must include `/engines/v1` at the end. Do not include a trailing slash.

Troubleshooting Cline

If Cline fails to connect:

1. Verify DMR is running:

   ```
   $ docker model status
   ```

2. Test the endpoint directly:

   ```
   $ curl http://localhost:12434/engines/v1/models
   ```

3. Check that CORS is configured if running a web-based version:
   - In Docker Desktop Settings > AI, add your origin to **CORS Allowed Origins**

Continue (VS Code / JetBrains)

Continue is an open-source AI code assistant that works with VS Code and JetBrains IDEs.

Configuration

Edit your Continue configuration file (`~/.continue/config.json`):

```
{
  "models": [
    {
      "title": "Docker Model Runner",
      "provider": "openai",
      "model": "ai/qwen2.5-coder",
      "apiBase": "http://localhost:12434/engines/v1",
      "apiKey": "not-needed"
    }
  ]
}
```

Using Ollama provider

Continue also supports the Ollama provider, which works with DMR:

```
{
  "models": [
    {
      "title": "Docker Model Runner (Ollama)",
      "provider": "ollama",
      "model": "ai/qwen2.5-coder",
      "apiBase": "http://localhost:12434"
    }
  ]
}
```

Cursor

Cursor is an AI-powered code editor.

Configuration

1. Open Cursor Settings (Cmd/Ctrl + ,).
2. Navigate to **Models** > **OpenAI API Key**.
3. Configure:

Setting	Value
OpenAI API Key	`not-needed`
Override OpenAI Base URL	`http://localhost:12434/engines/v1`

4. In the model drop-down, enter your model name: `ai/qwen2.5-coder`

> [!NOTE] Some Cursor features may require models with specific capabilities (e.g., function calling). Use capable models like `ai/qwen2.5-coder` or `ai/llama3.2` for best results.

Zed

Zed is a high-performance code editor with AI features.

Configuration

Edit your Zed settings (`~/.config/zed/settings.json`):

```
{
  "language_models": {
    "openai": {
      "api_url": "http://localhost:12434/engines/v1",
      "available_models": [
        {
          "name": "ai/qwen2.5-coder",
          "display_name": "Qwen 2.5 Coder (DMR)",
          "max_tokens": 8192
        }
      ]
    }
  }
}
```

Open WebUI

Open WebUI provides a ChatGPT-like interface for local models.

See Open WebUI integration for detailed setup instructions.

Aider

Aider is an AI pair programming tool for the terminal.

Configuration

Set environment variables or use command-line flags:

```
export OPENAI_API_BASE=http://localhost:12434/engines/v1
export OPENAI_API_KEY=not-needed

aider --model openai/ai/qwen2.5-coder
```

Or in a single command:

```
$ aider --openai-api-base http://localhost:12434/engines/v1 \
        --openai-api-key not-needed \
        --model openai/ai/qwen2.5-coder
```

LangChain

Python

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    base_url="http://localhost:12434/engines/v1",
    api_key="not-needed",
    model="ai/qwen2.5-coder"
)
```

```
response = llm.invoke("Write a hello world function in Python")
print(response.content)
```

JavaScript/TypeScript

```
import { ChatOpenAI } from "@langchain/openai";

const model = new ChatOpenAI({
  configuration: {
    baseURL: "http://localhost:12434/engines/v1",
  },
  apiKey: "not-needed",
  modelName: "ai/qwen2.5-coder",
});

const response = await model.invoke("Write a hello world function");
console.log(response.content);
```

LlamaIndex

```
from llama_index.llms.openai_like import OpenAILike

llm = OpenAILike(
    api_base="http://localhost:12434/engines/v1",
    api_key="not-needed",
    model="ai/qwen2.5-coder"
)

response = llm.complete("Write a hello world function")
print(response.text)
```

OpenCode

OpenCode is an open-source coding assistant designed to integrate directly into developer workflows. It supports multiple model providers and exposes a flexible configuration system that makes it easy to switch between them.

See Use OpenCode with Docker Model Runner for a task-focused guide that walks through model setup, configuration, and troubleshooting.

Configuration

1. Install OpenCode (see docs)
2. Reference DMR in your OpenCode configuration, either globally at `~/.config/opencode/opencode.json` or project specific with a `opencode.json` file in the root of your project

```
{
  "$schema": "https://opencode.ai/config.json",
  "provider": {
    "dmr": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Docker Model Runner",
      "options": {
        "baseURL": "http://localhost:12434/v1"
      },
      "models": {
        "ai/qwen2.5-coder": {
          "name": "ai/qwen2.5-coder"
        },
        "ai/llama3.2": {
          "name": "ai/llama3.2"
        }
      }
    }
  }
}
```

3. Select the model you want in OpenCode

You can find more details in this Docker Blog post

Claude Code

Claude Code is Anthropic's command-line tool for agentic coding. It lives in your terminal, understands your codebase, and executes routine tasks, explains complex code, and handles Git workflows through natural language commands.

See Use Claude Code with Docker Model Runner for a task-focused guide that walks through model setup, configuration, and inspecting requests. To run Claude Code in an isolated Docker Sandbox against a local model, see Run Claude Code in a Docker Sandbox with Docker Model Runner.

Configuration

1. Install Claude Code (see docs)
2. Use the `ANTHROPIC_BASE_URL` environment variable to point Claude Code at DMR. On Mac or Linux, you can do this, for example if you want to use the `gpt-oss:32k` model:

   ```
   ANTHROPIC_BASE_URL=http://localhost:12434 claude --model qwen2.5-coder
   ```

   On Windows (PowerShell) you can do it like this:

   ```
   $env:ANTHROPIC_BASE_URL="http://localhost:12434"
   claude --model gpt-oss:32k
   ```

> [!TIP]
>
> To avoid setting the variable each time, add it to your shell profile (`~/.bashrc`, `~/.zshrc`, or equivalent):
>
> ```
> export ANTHROPIC_BASE_URL=http://localhost:12434
> ```

You can find more details in this Docker Blog post

> [!NOTE]
>
> While the other integrations on this page use the OpenAI-compatible API, DMR also exposes a Anthropic-compatible API used here.

Common issues

"Connection refused" errors

1. Ensure Docker Model Runner is enabled and running:

   ```
   $ docker model status
   ```

2. Verify TCP access is enabled:

   ```
   $ curl http://localhost:12434/engines/v1/models
   ```

3. Check if another service is using port 12434.
4. If you run your tool in WSL and want to connect to DMR on the host via `localhost`, this might not directly work. Configuring WSL to use mirrored networking can solve this.

"Model not found" errors

1. Verify the model is pulled:

   ```
   $ docker model list
   ```

2. Use the full model name including namespace (e.g., `ai/qwen2.5-coder`, not just `qwen2.5-coder`).

Slow responses or timeouts

1. For first requests, models need to load into memory. Subsequent requests are faster.
2. Consider using a smaller model or adjusting the context size:

   ```
   $ docker model configure --context-size 4096 ai/qwen2.5-coder
   ```

3. Check available system resources (RAM, GPU memory).

CORS errors (web-based tools)

If using browser-based tools, add the origin to CORS allowed origins:

1. Docker Desktop: Settings > AI > CORS Allowed Origins
2. Add your tool's URL (e.g., `http://localhost:3000`)

Recommended models by use case

Use case	Recommended model	Notes
Code completion	`ai/qwen3-coder`	Optimized for coding tasks with a large context window
Agentic coding	`ai/devstral-small-2`	Good fit for tools such as Claude Code and OpenCode
General assistant	`ai/llama3.2`	Good balance of capabilities
Small/fast	`ai/smollm2`	Low resource usage
Embeddings	`ai/all-minilm`	For RAG and semantic search

What's next

- API reference - Full API documentation
- Configuration options - Tune model behavior
- Open WebUI integration - Set up a web interface

□□□ #2
□ □□□ □□□ 2026-10-31 18:54:18 UTC
□ □□□ □□□ 2026-10-31 18:54:19 UTC