Radium Integrations

App integrations — v1 draft

Radium serves OpenAI-compatible Chat Completions and an Anthropic-compatible Messages API. Most agents, coding tools, and chat apps need a base URL, an API key, and a model string, and nothing else.

> **New page** Nothing like this exists on radium.cloud. The homepage names Claude Code, Codex, VS Code, and Vertex, and the Agents page adds LangGraph, LlamaIndex, Vercel AI SDK, Pydantic AI, CrewAI, and the OpenAI Agents SDK, but no page tells anyone how to configure any of them. This is the destination for those logo strips, which are currently decorative.

> **Inconsistency to fix** The homepage body copy says Radium works with "Claude Code, Codex, VS Code, and Vertex" while the logo row directly beneath it shows Claude Code, VS Code, Codex, and Vercel. Vertex and Vercel are different products. Decide which one is true, then fix the page that is wrong. This page assumes Vercel AI SDK, which is the one Radium plausibly supports through an OpenAI-compatible client.

> **Keep this page open while you get your key** — Open the dashboard in a second tab, copy the one-time secret, then come back and finish setup. *(Open API keys)*

Before you start

1. Create an API key in the dashboard. The secret is shown once, so copy it before you leave the page.
2. Pick a model string from the table below.
3. Store the key in an environment variable wherever the app supports it. Never put a live key in a repository or a shared settings file.

Model	Model string	Use for
Hal 1.0	`hal-1.0`	Agents, tool calling, coding
Clarke 1.0	`clarke-1.0`	Chat, RAG, retrieval
Tycho 1.0	`tycho-1.0`	Classification, extraction, routing

> **Hard gate** Model string unresolved. Use the switcher in the build bar to preview this page under either convention. Every config file, environment variable, and code block below updates together. Owner: Vijay.

> **Confirm** API key prefix. Cheaper Inference uses `ir_live_` and shows it in every example, which makes a key visually identifiable in a log or a screenshot. Radium's prefix is not published. Examples below use `YOUR_RADIUM_API_KEY` until it is. Owner: Brendan.

Base URLs

OpenAI-compatible *(Chat Completions)*

```
https://api.radium.cloud/v1
```

Anthropic-compatible *(Messages)*

```
https://api.radium.cloud
```

Use the OpenAI-compatible base URL for everything on this page except Claude Code, which appends `/v1/messages` itself and therefore takes the base URL without the version segment.

Pick your app

Coding agents: Claude Code, Codex, Cursor, Cline, Continue, OpenCode, Aider

Frameworks: Vercel AI SDK, LangGraph, LlamaIndex, CrewAI, Pydantic AI, OpenAI Agents SDK

Chat interfaces: Open WebUI, LibreChat

Claude Code

Coding agent

Run local Claude Code tasks against Radium through the Anthropic-compatible Messages API.

Reference: Anthropic LLM gateway documentation

> **Scope** This changes local Claude Code API traffic. It does not change claude.ai. Local tools, tool results, multi-turn tasks, and streamed text are supported.

1. Install or verify Claude Code

```
npm install --global @anthropic-ai/claude-code
claude --version
```

2. Export the connection settings

Claude Code appends `/v1/messages` itself, so this base URL intentionally does not end in `/v1`.

macOS or Linux:

```
export ANTHROPIC_BASE_URL="https://api.radium.cloud"
export ANTHROPIC_AUTH_TOKEN="YOUR_RADIUM_API_KEY"
export ANTHROPIC_MODEL="hal-1.0"
export ANTHROPIC_SMALL_FAST_MODEL="tycho-1.0"

claude
```

Windows PowerShell:

```
$env:ANTHROPIC_BASE_URL = "https://api.radium.cloud"
$env:ANTHROPIC_AUTH_TOKEN = "YOUR_RADIUM_API_KEY"
$env:ANTHROPIC_MODEL = "hal-1.0"
$env:ANTHROPIC_SMALL_FAST_MODEL = "tycho-1.0"

claude
```

Setting the small and fast model explicitly matters. Claude Code makes background helper calls, and left unset it will reach for an Anthropic model string that Radium does not serve. Pointing it at

Tycho keeps those calls cheap and keeps them working.

3. Verify the endpoint before you trust the agent

```
curl https://api.radium.cloud/v1/messages \
  -H "X-Api-Key: YOUR_RADIUM_API_KEY" \
  -H "Anthropic-Version: 2023-06-01" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "hal-1.0",
    "max_tokens": 64,
    "messages": [{"role": "user", "content": "Reply with: connected"}]
  }'
```

Then run one non-interactive task from the configured terminal.

```
claude -p "Reply with: Claude Code connected"
```

> **Expected result** Claude Code prints the response, and the dashboard shows a settled request against the Messages endpoint with the model and token usage you expect.

Troubleshooting

Issue	Fix
Requests return 401	The process cannot read the token. Start Claude Code from the terminal where you exported it rather than from a launcher.
Background calls fail while the main task works	`ANTHROPIC_SMALL_FAST_MODEL` is unset or points at a model Radium does not serve.
404 on every request	The base URL ends in `/v1`. Remove it. Claude Code adds the path itself.

> **Confirm** Whether extended thinking blocks replay across tool-use turns, and whether `/v1/messages/count_tokens` is implemented. Both matter for Claude Code specifically. Owner: Vijay.

Codex *(Blocked)*

Coding agent

Send new local Codex tasks through Radium.

Reference: Codex configuration reference

> **Blocks this section — decide first** Codex's `wire_api` setting has two values. `"responses"` requires an OpenAI Responses API implementation, and `"chat"` uses Chat Completions. Cheaper Inference built a stateless Responses layer specifically so Codex would work properly. Radium publishes no Responses endpoint anywhere, so this section is drafted against `wire_api = "chat"`, which is the fallback and may degrade Codex's local tool handling including `apply_patch`. If Radium has a Responses endpoint, this section changes. If it does not, we should say so plainly rather than publishing a config that half works. Owner: Vijay.

1. Install or verify Codex

```
npm install --global @openai/codex
codex --version
```

2. Make your key available

macOS or Linux:

```
export RADIUM_API_KEY="YOUR_RADIUM_API_KEY"
```

Windows PowerShell:

```
$env:RADIUM_API_KEY = "YOUR_RADIUM_API_KEY"
```

This sets the key for that terminal session only. Run Codex from the same window.

3. Add the provider

`~/.codex/config.toml`:

```
model = "hal-1.0"
model_provider = "radium"
```

```
[model_providers.radium]
name = "Radium"
base_url = "https://api.radium.cloud/v1"
env_key = "RADIUM_API_KEY"
wire_api = "chat"
```

In the desktop app, open Settings, then Configuration, then Open config.toml.

4. Restart and test

```
codex exec --model hal-1.0 --sandbox read-only \
  "Run pwd without changing files, then tell me the directory."
```

In the desktop app, quit completely and reopen it, then start a new local task. An existing task keeps its original model.

Troubleshooting

Issue	Fix
`codex: command not found`	Open a new terminal after installation and check `codex --version` again.
Requests return 401 from the desktop app	Apps opened from the Dock do not inherit a key exported in Terminal. Use Codex's command-backed authentication with a key stored in the system keychain.
The model is missing from the picker	Custom models may not appear. Set the exact string in `config.toml`, restart, and start a new local task.

Cursor

AI code editor

Use Cursor's OpenAI base URL override for Ask and Agent.

Reference: Cursor API key documentation

> **Supported with limits** Cursor applies one OpenAI base URL and key across its OpenAI-family models. It cannot hold separate provider settings per model, and tab completion and other features that depend on Cursor-hosted models

> continue to use Cursor's own infrastructure.

1. Update Cursor to the latest stable version.
2. Create a dedicated Radium key. Restrict it to the models Cursor should use and set a monthly spend cap.
3. Open Cursor Settings, then Models, then find OpenAI API Key.
4. Paste the key and enable Override OpenAI Base URL.
5. Set the base URL to `https://api.radium.cloud/v1` and click Verify.
6. Select a model whose exact string matches the table above.
7. Test a short Ask prompt, then ask Agent to inspect a file and make one harmless edit.

> **Do not restrict the key to your workstation IP** Cursor assembles these requests through its own service, so Radium sees Cursor's outbound address rather than your machine. An IP restriction scoped to your laptop will reject every request. Use model, rate, spend, and expiry controls instead.

Troubleshooting

Issue	Fix
Key verification fails	Check that the key is not expired, has access to the selected model, and is not IP-restricted to your workstation.
Ask works but Agent cannot edit files	Agent sends tool traffic in a shape the gateway has to normalize. Capture the request ID from the dashboard before contacting support.
A Cursor built-in model stops working	The override is global. Turn it off to return OpenAI-family models to Cursor's normal routing.

> **Confirm** Whether the gateway normalizes Cursor Agent's streamed `ApplyPatch` tool shape. Cheaper Inference does this explicitly and documents it. If Radium does not, Agent mode will not work and this section should say so rather than listing it as supported. Owner: Vijay.

Cline

VS Code agent

Use the OpenAI Compatible provider inside VS Code.

Reference: Cline provider configuration

1. Open Cline settings and choose **OpenAI Compatible** as the API provider.
2. Set Base URL to `https://api.radium.cloud/v1`.
3. Paste your API key and enter the model string.
4. Save, run a small task, and confirm the request appears in the dashboard.

> **Advanced model settings** Enable image support, tool use, and context or output limits only where the selected model supports them. These settings are per model even when the same connection is reused.

Continue

IDE assistant

Reference: Continue OpenAI provider documentation

`config.yaml`:

```
name: Radium
version: 0.0.1
schema: v1

models:
  - name: Hal 1.0 through Radium
    provider: openai
    model: hal-1.0
    apiBase: https://api.radium.cloud/v1
    apiKey: YOUR_RADIUM_API_KEY
    useResponsesApi: false
```

Keep `useResponsesApi` set to false so Continue uses the Chat Completions path.

OpenCode

Coding agent

Reference: OpenCode provider documentation

1. Open `/connect`, choose **Other**, enter `radium`, and add your key.
2. Add the provider below to `opencode.json`, then use `/models` to select it.

`opencode.json`:

```
{
  "$schema": "https://opencode.ai/config.json",
  "provider": {
    "radium": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Radium",
      "env": ["RADIUM_API_KEY"],
      "options": {
        "baseURL": "https://api.radium.cloud/v1"
      },
      "models": {
        "hal-1.0": { "name": "Hal 1.0" },
        "clarke-1.0": { "name": "Clarke 1.0" },
        "tycho-1.0": { "name": "Tycho 1.0" }
      }
    }
  }
}
```

Use `@ai-sdk/openai-compatible` so OpenCode sends Chat Completions requests. The `env` declaration names the variable, it does not contain the secret.

Aider

Terminal pair programmer

Reference: Aider OpenAI-compatible documentation

macOS or Linux:

```
export OPENAI_API_BASE=https://api.radium.cloud/v1
export OPENAI_API_KEY=YOUR_RADIUM_API_KEY

cd /path/to/your/project
aider --model openai/hal-1.0
```

Windows PowerShell:

```
$env:OPENAI_API_BASE = "https://api.radium.cloud/v1"
$env:OPENAI_API_KEY = "YOUR_RADIUM_API_KEY"

Set-Location C:\path\to\your\project
aider --model openai/hal-1.0
```

Keep Aider's `openai/` prefix in front of the Radium model string. Aider uses the prefix to pick the client, not the provider.

Vercel AI SDK

Framework

```
import { createOpenAI } from "@ai-sdk/openai";
import { generateText } from "ai";

const radium = createOpenAI({
  apiKey: process.env.RADIUM_API_KEY,
  baseURL: "https://api.radium.cloud/v1",
});

const { text } = await generateText({
  model: radium("hal-1.0"),
  prompt: "Hello, Radium!",
});
```

Streaming, tool calling, and structured output work through the same provider instance.

LangGraph

Agent framework

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="hal-1.0",
    api_key=os.environ["RADIUM_API_KEY"],
    base_url="https://api.radium.cloud/v1",
)

# Bind tools and build the graph exactly as you do today.
llm_with_tools = llm.bind_tools(tools)
```

Graph structure, checkpointing, and tool nodes are untouched. Only the client construction changes.

LlamaIndex

RAG framework

```
from llama_index.llms.openai_like import OpenAILike

llm = OpenAILike(
    model="clarke-1.0",
    api_key=os.environ["RADIUM_API_KEY"],
    api_base="https://api.radium.cloud/v1",
    is_chat_model=True,
    is_function_calling_model=True,
)
```

> **Embeddings** Radium serves chat models. Keep your existing embedding provider configured separately and point only the LLM at Radium.

> **Confirm** Whether Radium serves an embeddings endpoint. If it does, this callout is wrong and the page should show the config instead. If it does not, the callout stays, because a RAG team will hit this within five minutes. Owner: Vijay.

CrewAI

Agent framework

```
from crewai import LLM

llm = LLM(
    model="openai/hal-1.0",
    base_url="https://api.radium.cloud/v1",
    api_key=os.environ["RADIUM_API_KEY"],
)
```

CrewAI routes through LiteLLM, so keep the `openai/` prefix in front of the Radium model string.

Pydantic AI

Agent framework

```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel
from pydantic_ai.providers.openai import OpenAIProvider

model = OpenAIModel(
    "hal-1.0",
    provider=OpenAIProvider(
        base_url="https://api.radium.cloud/v1",
        api_key=os.environ["RADIUM_API_KEY"],
    ),
)

agent = Agent(model, output_type=YourSchema)
```

Structured output validation is unchanged, because the response shape is unchanged.

OpenAI Agents SDK

Agent framework

```
from agents import Agent, OpenAIChatCompletionsModel, set_tracing_disabled
from openai import AsyncOpenAI

client = AsyncOpenAI(
    api_key=os.environ["RADIUM_API_KEY"],
    base_url="https://api.radium.cloud/v1",
)

agent = Agent(
    name="Assistant",
    model=OpenAIChatCompletionsModel(
        model="hal-1.0",
        openai_client=client,
    ),
)
```

> **Use the Chat Completions model class** The Agents SDK defaults to the Responses API. Construct the agent with `OpenAIChatCompletionsModel` so it uses the Chat Completions path.

> **Tracing** The SDK uploads traces to OpenAI by default using an OpenAI key. Call `set_tracing_disabled(True)`, or configure your own tracing processor, so trace data does not leave for a provider you are no longer using.

Open WebUI

Chat interface

Reference: Open WebUI connection documentation

1. Open Admin Settings, then Connections, then OpenAI.
2. Click Add Connection.
3. Enter `https://api.radium.cloud/v1` and your API key, then save.
4. Select a discovered model, or add exact model strings under Model IDs.

> **Feature scope** Chat and model discovery work through this connection. Configure a separate provider for embeddings, speech to text, and text to speech.

LibreChat

Multi-user chat

Reference: LibreChat custom endpoint documentation

`.env`:

```
RADIUM_API_KEY=YOUR_RADIUM_API_KEY
```

`librechat.yaml`:

```
version: 1.3.13
endpoints:
  custom:
    - name: Radium
      apiKey: '${RADIUM_API_KEY}'
      baseURL: 'https://api.radium.cloud/v1'
      models:
        default:
          - hal-1.0
          - clarke-1.0
          - tycho-1.0
        fetch: true
      titleConvo: true
      titleModel: tycho-1.0
      modelDisplayLabel: Radium
```

Setting `titleModel` to Tycho keeps conversation-title generation on the cheapest tier, which is most of what that call is for.

Verify the connection

1. Send a short prompt from the app.
2. Confirm the response completes.
3. Open the dashboard and check that the request shows the model, token usage, and charge you expect.

If the app works but nothing appears in the dashboard, the app is still talking to its original provider. Check that the base URL override is actually enabled and that the app was fully restarted.

> **Moving real work across** Point one workload at Radium first, compare cost per completed task against your current provider, and expand from there. The switch is two values, and so is the way back.

Not supported

Being explicit here saves a support ticket.

- Embeddings, speech to text, and text to speech. Keep your existing provider configured for these. *(Confirm)*
- Image and video generation. *(Confirm)*
- Provider-hosted tools such as server-side web search or file search.
- Stored conversations and server-side conversation state.
- Batch APIs. *(Confirm)*

> **Confirm the whole list** Every line above is an assumption drawn from what Radium does not advertise. A wrong entry here is worse than no page, because it tells a developer that something is unsupported when it works. Owner: Vijay.

Sign-off ledger

Item	Owner	Status
Model string resolved, page-wide	Vijay	Hard gate
Codex wire_api, Responses endpoint exists or does not	Vijay	Hard gate
"Not supported" list confirmed line by line	Vijay	Hard gate
Cursor Agent ApplyPatch normalization	Vijay	Open
Anthropic Messages scope: thinking blocks, count_tokens	Vijay	Open
Embeddings endpoint exists or does not	Vijay	Open
API key prefix	Brendan	Open
Key restriction controls shipped in dashboard	Brendan	Open
Vertex versus Vercel on the homepage	Leo	Open
Each app verified end to end before publication	Leo / Vijay	Open
Homepage and Agents page logo strips link here	Brandon	Open

> **One process note** Cheaper Inference's guides read as though someone actually ran each one, because they document the failure modes rather than the happy path. The Codex Dock keychain workaround, the Cursor IP warning, and the Claude Code small-model note are all things you only learn by breaking it. Every section here should be run end to end by someone who has not set it up before, and the troubleshooting rows rewritten from what actually went wrong. Publishing this page untested would be worse than not having it.

Created 2026-08-19 17:02:17 UTC by Admin
Updated 2026-08-19 17:03:06 UTC by Admin