Radium Developer Docs (old)

- Radium Docs
- Radium Integrations

Radium Docs

Documentation — v1 draft

Radium serves Hal, Clarke, and Tycho over OpenAI-compatible and Anthropic-compatible endpoints. Point your existing client at Radium, name a model, and the SDK, the loop, the tools, and the evals carry on as they are.

> **Hard gate — blocks publication** Model string unresolved. Use the switcher in the build bar to preview this page under either convention. Every code sample, table, and endpoint reference on this page updates together. The live Agents page currently ships `radium-1` next to Hal 1.0, Clarke 1.0, and Tycho 1.0 cards, so whichever way this goes, that page needs correcting in the same pass. Owner: Vijay.

> **What this replaces** radium.cloud/docs is currently a resource index with two PDF downloads and four cross-links. There is no API reference on the site. A developer evaluating Radium today has to open a PDF to find the base URL. This page closes that gap and is the destination for the "Full documentation" link in the homepage Start Here block.

> **Unverified content** Sections marked with a blue tag carry values or behaviour that need Vijay's confirmation before publication. Everything in those sections is drafted from the Agents page, the switching guide, and the public model cards, and should be treated as a proposal rather than fact.

Quick start

1. Create an account at deploy.radium.cloud.
2. Create an API key from the dashboard and copy it. The secret is shown once.
3. Set the base URL on your existing client to the compatible endpoint below.
4. Put a Radium model string in the `model` field.
5. Send your first request and review usage in the dashboard.

Nothing else in your integration changes. If your application already speaks to OpenAI or Anthropic, it already speaks to Radium.

Base URLs

OpenAI-compatible

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

Anthropic-compatible

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

Anthropic SDKs and Claude Code append `/v1/messages` themselves, so the Anthropic-compatible base URL is given without the version segment.

> **Confirm** Anthropic-compatible base URL path shape. The homepage claims OpenAI and Claude compatibility, and the Agents page shows an Anthropic SDK tab, but the exact base URL for the Anthropic path is not published anywhere on the current site. Owner: Vijay.

Authentication

Pass your key to the client and the SDK builds the header. You do not need to construct it yourself unless you are calling the API over raw HTTP.

Three things about keys that are not obvious from the header.

Detail	Why it matters
The Messages endpoint also accepts `X-Api-Key`	Anthropic SDKs and Claude Code send this header rather than a bearer token, so both work without a shim.
Keys are shown once	Copy the secret at creation. There is no way to reveal it later, so a lost key is replaced rather than recovered.
Keys carry a `YOUR_RADIUM_API_KEY` prefix *(Confirm)*	The prefix makes a Radium key identifiable in a log, an environment dump, or a repository scan, which is what makes automated secret detection work.

For direct HTTP clients, the header is `Authorization: Bearer YOUR_API_KEY`.

> **Rewritten** This section previously led with a code block containing nothing but the bearer header, which every OpenAI SDK user already assumes and the SDK writes for them. Cheaper Inference runs the same block and gets away with it because `ir_live_` is visible in it, so the row carries a fact. Radium has no published prefix, so the block carried none. Leading with the three non-obvious facts instead, and the raw header drops to one line at the end for the people who actually need it.

> **Confirm** The key prefix. Getting one published is worth more than it looks, because it is what lets a customer's own secret scanner catch a leaked Radium key before it is used. Owner: Brendan.

Your first request

(Interactive code sample on the live page offers curl, Python, TypeScript, Anthropic Python, and Go tabs.)

curl

```
curl https://api.radium.cloud/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "hal-1.0",
    "messages": [{"role": "user", "content": "Hello, Radium!"}]
  }'
```

Python

```
from openai import OpenAI

client = OpenAI(
    api_key="YOUR_API_KEY",
    base_url="https://api.radium.cloud/v1",
)

response = client.chat.completions.create(
```

```
    model="hal-1.0",
    messages=[{"role": "user", "content": "Hello, Radium!"}],
)

print(response.choices[0].message.content)
```

TypeScript

```
import OpenAI from "openai";

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

const response = await client.chat.completions.create({
  model: "hal-1.0",
  messages: [{ role: "user", content: "Hello, Radium!" }],
});

console.log(response.choices[0].message.content);
```

Anthropic Python

```
from anthropic import Anthropic

client = Anthropic(
    api_key="YOUR_API_KEY",
    base_url="https://api.radium.cloud",
)

message = client.messages.create(
    model="hal-1.0",
    max_tokens=1024,
    messages=[{"role": "user", "content": "Hello, Radium!"}],
)

print(message.content[0].text)
```

Go

```
client := openai.NewClient(
    option.WithAPIKey(os.Getenv("RADIUM_API_KEY")),
    option.WithBaseURL("https://api.radium.cloud/v1"),
)

resp, err := client.Chat.Completions.New(ctx, openai.ChatCompletionNewParams{
    Model: "hal-1.0",
    Messages: []openai.ChatCompletionMessageParamUnion{
        openai.UserMessage("Hello, Radium!"),
    },
})
```

A successful response

```
{
  "id": "chatcmpl_...",
  "object": "chat.completion",
  "model": "hal-1.0",
  "choices": [{
    "index": 0,
    "message": {"role": "assistant", "content": "Hello."},
    "finish_reason": "stop"
  }],
  "usage": {
    "prompt_tokens": 12,
    "completion_tokens": 3,
    "total_tokens": 15
  }
}
```

Using an AI app

Radium works with the tools your team already runs. Each of these needs a base URL override and a Radium key, and nothing else.

App	Endpoint used	Where to set it

Claude Code	Anthropic Messages	`ANTHROPIC_BASE_URL` and `ANTHROPIC_API_KEY` environment variables
Codex	OpenAI Chat Completions	Base URL override in config
Cursor	OpenAI Chat Completions	Settings, Models, Override OpenAI Base URL
VS Code	OpenAI Chat Completions	Extension provider settings
Vercel AI SDK	OpenAI Chat Completions	`createOpenAI({ baseURL })`
LangGraph, LlamaIndex, CrewAI, Pydantic AI	OpenAI Chat Completions	Standard OpenAI client construction

> **Cursor** The base URL override is global for OpenAI-family models. Do not restrict a Cursor key to your workstation IP, because Cursor sends these requests from its own service rather than your machine.

> **To build** Each row should link to a per-app setup page with the exact environment variables and a verification step. Cheaper Inference runs this as a separate integration directory at /docs/integrations and it is the right pattern. Scope as a follow-on once this page ships.

Models and rates *(Confirm rates)*

Rates are published, fixed, and the same for every account. There is no volume tier, no committed-spend discount, and no negotiated rate, so the number below is the number on the invoice.

Model	Model string	Input / MTok	Output / MTok	Tier
Hal 1.0	`hal-1.0`	$2.25	$11.50	Maximum capability
Clarke 1.0	`clarke-1.0`	$1.50	$7.00	Balanced performance
Tycho 1.0	`tycho-1.0`	$0.50	$2.25	High-efficiency scale

Rates are billed per token on actual usage. Reasoning tokens, where a model produces them, are billed at the output rate.

> **Positioning note** The fixed rate is the commercial handle and this table is where a developer meets it, so the framing sentence above the table is doing real work. Cheaper Inference publishes a volatile market rate and an hourly savings heatmap. The inverse artifact, a rate history page showing these three rows unchanged since launch against published Anthropic and OpenAI list price changes, would link from here. Proposed to Adam.

Choosing a model

Workload	Start with	Notes
Agent loops, tool calling, software tasks	Hal 1.0	Loop length holds against frontier models in the production benchmark, so the rate difference carries through to cost per completed task.
RAG, retrieval, support chat	Clarke 1.0	The default for most production traffic.
Classification, extraction, routing	Tycho 1.0	High-throughput work where the task is narrow and the volume is large.

The cheapest model is not always the cheapest task. Weaker reasoning shows up as more steps, and more steps cost more than a higher per-token rate. Measure cost per completed task rather than cost per million tokens.

Model catalog API

```
curl https://api.radium.cloud/v1/models \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Returns the available models with current rates, modality, and capability flags. Use it to confirm a model is available and supports the modality a workload needs before routing traffic to it.

> **Confirm** Response shape, capability flag names, and whether filtering parameters are supported. Owner: Vijay.

Parameters *(Confirm support)*

OpenAI-compatible

Chat requests use the OpenAI Chat Completions shape. The following fields are forwarded to the model.

Field	Notes
`temperature`, `top_p`	Standard sampling controls.
`max_tokens`, `max_completion_tokens`	Caps generated output.
`stop`	Stop sequences.
`tools`, `tool_choice`	See tools and structured output.
`response_format`	JSON object and JSON schema.
`stream`	Server-sent events.
`reasoning`	Where the model exposes an effort control.

Anthropic-compatible

Messages requests accept `model`, `messages`, `max_tokens`, `system`, `stream`, `temperature`, `top_p`, `stop_sequences`, `tools`, `tool_choice`, and `thinking`. Use Radium model strings rather than Anthropic model aliases.

> **Confirm** Which of the above are supported, which are accepted and ignored, and which are rejected. This table is the most-read part of any compatibility doc and being wrong here costs more trust than leaving a row out. Owner: Vijay.

Tools and structured output

Tool calling uses the standard `tools` and `tool_choice` fields. Parallel tool calls are supported, and tool deltas are delivered in the stream.

```
"response_format": {
  "type": "json_object"
```

```
}
```

Tool schemas do not need changing. Tool results replay across turns in the shape your SDK already sends.

Streaming

Set `"stream": true`. Streams use server-sent events and follow the event names of whichever compatibility layer you are on, so OpenAI clients receive OpenAI-shaped chunks and Anthropic clients receive `message_start`, `content_block_delta`, `message_delta`, and `message_stop`.

> **Interrupted streams** Treat an interrupted stream as an incomplete response and retry the whole request. A partial stream cannot be resumed from the point of failure.

Vision input *(Confirm limits)*

Vision-capable models accept OpenAI `image_url` content parts and Anthropic-style base64 image blocks.

```
curl https://api.radium.cloud/v1/chat/completions \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "hal-1.0",
    "messages": [{
      "role": "user",
      "content": [
        {"type": "text", "text": "Describe this image."},
        {"type": "image_url",
         "image_url": {"url": "https://example.com/image.png"}}
      ]
    }]
  }'
```

> **Missing** Per-image size cap, image count cap, total decoded payload cap, and maximum serialized body size. Every one of these is a support ticket if it is not documented. Owner: Vijay.

Prompt caching *(Confirm behaviour)*

`cache_control` is passed through where the model supports it. Cache reads are billed at a reduced input rate and cache writes at the standard input rate.

> **Confirm** Whether caching is supported at all in v1, the cache read rate, the minimum cacheable prefix, and the cache lifetime. If it is not supported, say so plainly here rather than omitting the section, because a team migrating from Anthropic will look for it. Owner: Vijay.

Context and limits *(Missing)*

Model	Context window	Max output tokens
Hal 1.0	—	—
Clarke 1.0	—	—
Tycho 1.0	—	—

> **Blocks publication** Context windows and output caps are not published anywhere on radium.cloud, including the individual model pages. This is the first thing a technical evaluator checks and its absence reads as an unfinished product. Owner: Vijay.

Usage metadata

Successful responses include token usage. Streaming responses include it in the final event before the stream closes.

```
"usage": {
  "prompt_tokens": 1234,
  "completion_tokens": 321,
  "total_tokens": 1555
}
```

> **Opportunity** Cheaper Inference returns a namespaced object with a request ledger ID and the exact settled charge on every response, plus the request ID in a response header. Returning the settled cost inline is a strong idea for Radium, because it makes the fixed rate visible on every single request rather than only in a monthly invoice. Worth scoping with Brendan alongside the HubSpot usage sync.

Rate limits *(Missing)*

Limits apply per key and per account, covering requests per minute, concurrent requests, and tokens per minute.

> **Blocks publication** No published limits. A team cannot plan a migration without knowing the concurrency ceiling, and this is the specific question that comes up when an agent workload fans out. The Agents page FAQ already lists "What are the concurrency limits when an agent fans out?" as a question, so it is a known gap. Owner: Vijay.

Errors and retries

OpenAI-compatible endpoints return the OpenAI error envelope. Anthropic-compatible endpoints return the Anthropic error shape. Use the error type for program logic and keep the message for logs.

```
{
  "error": {
```

```
    "message": "Invalid API key.",
    "type": "authentication_error",
    "param": null,
    "code": "invalid_api_key"
  }
}
```

Status	Meaning	Action
`400`	Unsupported model or invalid request body	Correct the request. Do not retry unchanged.
`401`	Invalid or missing API key	Check the key and the auth header.
`403`	The key does not allow this model or client IP	Check key restrictions.
`413`	Request body too large	Split the payload.
`429`	Rate, concurrency, or quota limit reached	Back off. Respect `Retry-After`.
`500`, `502`, `503`	Service or transport failure	Retry with exponential backoff.
`504`	Generation timed out	Retry, and consider a lower output cap.

> **Confirm** The exact error type strings and codes Radium returns, and whether a `Retry-After` header is sent on 429. Owner: Vijay.

Data handling and residency *(Pending Alex)*

> **Blocks publication — Alex owns** One approved sentence on data handling and residency goes here, and the same sentence goes verbatim on the security page, the technology page, the enterprise page, and the homepage evidence band. Cheaper Inference restates their data boundary in identical words in four places and it is the single most disciplined thing on their site. Four paraphrases are worth less than one sentence used four times. Nothing is drafted here on purpose, because this is compliance copy and it is not marketing's to write.

[Approved data handling and residency statement]

Radium owns and operates the infrastructure that serves these models, in British Columbia and Montreal. Requests are not forwarded to a third-party model provider.

> **Note** The sentence above is the real differentiator against every gateway and reseller in this category, including Cheaper Inference, whose entire model routes customer prompts through OpenAI, Anthropic, and Google. It should survive Alex's review in some form because it is a factual statement about the stack rather than a compliance claim.

Production checklist

- Load the API key from a server-side environment variable, never from frontend code.
- Use separate keys for production, staging, and local testing.
- Confirm the model and modality with `GET /v1/models` before routing traffic.
- Set an explicit client timeout appropriate for long generations.
- Retry transport errors, `429`, and `5xx` with exponential backoff.
- Do not retry `400`, `401`, or `413` without correcting the request.
- Treat an interrupted stream as incomplete and retry the whole request.
- Move a controlled share of traffic first, compare cost per completed task, then expand.

Migrating from OpenAI

Before

```
client = OpenAI(
    api_key=OPENAI_API_KEY,
    base_url="https://api.openai.com/v1",
)
response = client.chat.completions.create(
    model="gpt-5.4",
    messages=messages,
)
```

After

```
client = OpenAI(
    api_key=RADIUM_API_KEY,
    base_url="https://api.radium.cloud/v1",
)
response = client.chat.completions.create(
    model="hal-1.0",
    messages=messages,
)
```

Two values change. The SDK, the message list, the tools, the streaming setting, and the response handling are untouched.

Migrating from Anthropic

After

```
client = Anthropic(
    api_key=RADIUM_API_KEY,
    base_url="https://api.radium.cloud",
)
message = client.messages.create(
    model="hal-1.0",
    max_tokens=1024,
    messages=messages,
)
```

For Claude Code, set `ANTHROPIC_BASE_URL` and `ANTHROPIC_API_KEY` and leave everything else in place.

Compatibility matrix

What changes	What stays the same
The base URL in your client	The agent loop
The model string	The OpenAI or Anthropic SDK
The invoice	Tool and function calls
	Parallel tool calls
	Streaming, including tool deltas

What changes	What stays the same
	Structured output
	Retry and error handling
	Evals, tracing and observability
	Your prompts and system messages

The switch is two values, and so is the way back.

Security and keys

- Store keys in environment variables.
- Issue one key per environment so a revocation has a bounded blast radius.
- Restrict a key by model, IP, expiry, rate, and spend where the dashboard supports it.
- Revoke an exposed key immediately and issue a replacement.

> **Confirm** Which key restriction controls actually exist in the dashboard today. Do not document a control that is not shipped. Owner: Brendan.

Billing

Usage is billed per token at the published rate for the model that served the request. Usage, token counts, and spend are available in the dashboard.

> **Open** Self-serve pricing publication and the rate lock mechanic are Adam's decision and are not settled. This section stays thin until they are. If a rate lock ships, it belongs here and on the pricing page in the same words.

Support and status

For API, billing, or account questions, contact support. Check status before escalating an availability issue.

>

> **Missing** Radium has no public status page. Cheaper Inference links one from the footer of every page including the docs, and an enterprise evaluator will look for it. Worth raising with Vijay separately from this page.

Sign-off ledger

Item	Owner	Status
Model string resolved, page-wide	Vijay	Hard gate
Context windows and output caps	Vijay	Hard gate
Rate limits and concurrency ceilings	Vijay	Hard gate
Data handling and residency sentence	Alex	Hard gate
Anthropic-compatible base URL shape	Vijay	Open
Parameter support table, all three columns	Vijay	Open
Vision limits	Vijay	Open
Prompt caching support and rates	Vijay	Open
Error type strings and Retry-After	Vijay	Open
Model catalog response shape	Vijay	Open
Key restriction controls actually shipped	Brendan	Open
Published rates confirmed current	Adam	Open
Rate lock mechanic and self-serve pricing	Adam	Open
Settled cost returned inline on every response	Brendan	Proposed
Per-app integration directory	Leo	Proposed
Public status page	Vijay	Proposed

© 2026 Radium

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.

© 2026 Radium