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

Created 2026-08-19 16:56:48 UTC by Admin
Updated 2026-08-19 17:01:28 UTC by Admin