Use DSPy with Radium

DSPy connects to OpenAI-compatible APIs through LiteLLM. Prefix the Radium model with `openai/`, then provide Radium's API key and base URL.

Before you start

You need Python 3.10 or newer, a Radium API key, and a terminal or PowerShell window.

1. Create a project and install DSPy

macOS or Linux:

```
mkdir radium-dspy
cd radium-dspy
python3 -m venv .venv
source .venv/bin/activate
python -m pip install dspy
```

Windows PowerShell:

```
mkdir radium-dspy
cd radium-dspy
py -m venv .venv
.venv\Scripts\Activate.ps1
python -m pip install dspy
```

2. Set your Radium credentials

macOS or Linux:

```
export RADIUM_API_KEY="your-radium-api-key"
export RADIUM_MODEL="hal-1.0"
```

Windows PowerShell:

```
$env:RADIUM_API_KEY = "your-radium-api-key"
$env:RADIUM_MODEL = "hal-1.0"
```

3. Create `main.py`

```
import os

import dspy



api_key = os.getenv("RADIUM_API_KEY")
if not api_key:
    raise SystemExit("Set RADIUM_API_KEY before running this program.")

model = os.getenv("RADIUM_MODEL", "hal-1.0")
lm = dspy.LM(
    f"openai/{model}",
    api_key=api_key,
    api_base="https://api.radium.cloud/v1",
    max_tokens=512,
    temperature=0,
    timeout=90,
)
dspy.configure(lm=lm)

responses = lm(
    messages=[
        {"role": "system", "content": "Follow the user's instruction exactly."},
        {"role": "user", "content": "Reply with exactly: Radium connected."},
    ]
)
raw = responses[0] if responses else None
content = raw.get("text") if isinstance(raw, dict) else raw

if not content or not str(content).strip():
    raise RuntimeError("Radium returned no visible text.")
print(f"RADIUM_RESPONSE: {str(content).strip()}")
```

DSPy may return a string or a dictionary containing `text` and `reasoning_content`. The code handles both response shapes.

4. Run it

```
python main.py
```

Expected output:

```
RADIUM_RESPONSE: Radium connected.
```

Choose a model

```
export RADIUM_MODEL="clarke-1.0"  # or hal-1.0 or tycho-1.0
python main.py
```

Migrate an existing DSPy program

Configure the global language model, then keep using your existing signatures and modules:

```
lm = dspy.LM(
    "openai/hal-1.0",
    api_key=os.environ["RADIUM_API_KEY"],
    api_base="https://api.radium.cloud/v1",
    max_tokens=512,
)
dspy.configure(lm=lm)
```

DSPy optimizers can make many model calls. Test latency and usage on a small dataset before starting a large optimization.

Tool-calling note

> **Known issue with hal-1.0** DSPy `ReAct` executed tools and returned grounded answers with all three models. With `hal-1.0`, DSPy logged an output-truncation warning during the agent trace at both 512 and 1024 output tokens, though the

> tool still ran once and the final answer was correct. Test longer ReAct workflows before relying on them in production.

Troubleshooting

- Authentication errors: verify the key in the active terminal.
- Provider errors: include `openai/` before the model ID passed to `dspy.LM`.
- Empty output: retain the response-shape normalization shown above.
- Large optimizer jobs: reduce the dataset first to confirm expected request volume.

Validation

Version note: these instructions were verified with Python 3.12.7 and `dspy==3.3.0` on August 20, 2026. You do not need that exact package version. Text generation passed with `hal-1.0`, `clarke-1.0`, and `tycho-1.0`; ReAct tool calling passed with the caveat above.

Reference: DSPy language model documentation.

Next

- API quickstart, for calling Radium directly
- Tool calling, for the `tool_use` and `tool_result` contract

Created 2026-08-23 21:35:12 UTC by Admin
Updated 2026-08-23 21:53:20 UTC by Admin