2 min read · Jul 16, 2025

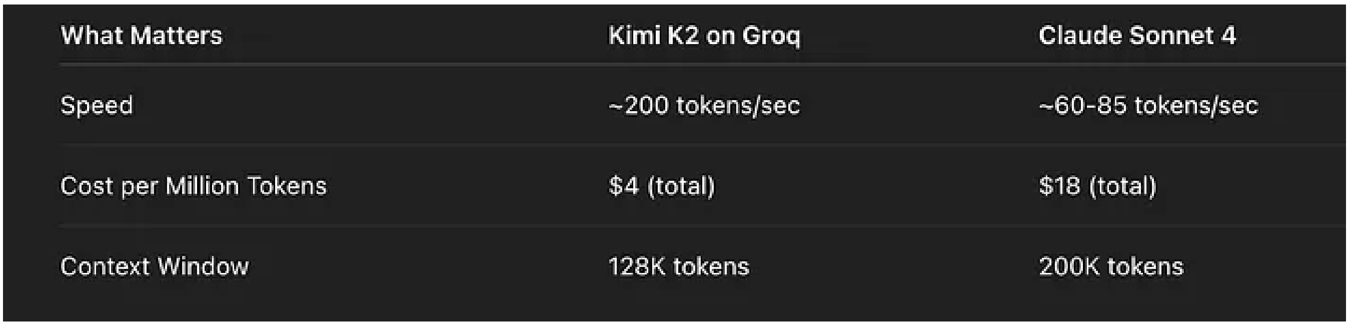
Hungrysoul Follow

Listen Share

Recently, I started using Claude Code with Groq's Kimi K2-Instruct model, and honestly, the combo feels too good to be true (open source FTW). If you've been relying heavily on Anthropic's Claude Sonnet model, this alternative might be exactly what you're looking for — it's significantly faster and way cheaper. Let's dive in!

Why Groq + Kimi K2 Makes Sense

Just to give you an idea why I'm excited, here's a quick comparison between Kimi K2 on Groq and Claude Sonnet 4 from Anthropic:



What Matters	Kimi K2 on Groq	Claude Sonnet 4
Speed	~200 tokens/sec	~60-85 tokens/sec
Cost per Million Tokens	$4 (total)	$18 (total)
Context Window	128K tokens	200K tokens

Yup, that's right — it's nearly 3x faster and about 5x cheaper per token. Pretty hard to beat.

Setting It Up (Super Easy!)

The best part? The setup takes about five minutes. Here's how you do it:

First, install Claude Code and Claude Code Router:

```
npm install -g @anthropic-ai/claude-code
npm install -g @musistudio/claude-code-router
```

Once installed, all you have to do is launch Claude Code through the router:

```
ccr code
```

Configuring the Router (Copy & Paste Job)

Now, let’s quickly create a config file for your router. Just copy the following snippet into `~/.claude-code-router/config.json`:

```
{
  "LOG": false,
  "Providers": [
    {
      "name": "groq",
      "api_base_url": "https://api.groq.com/openai/v1/chat/completions",
      "api_key": "<YOUR_GROQ_API_KEY>",
      "models": ["moonshotai/kimi-k2-instruct"],
      "transformer": {
        "use": [
          ["maxtoken", { "max_tokens": 16384 }],
          "openrouter"
        ]
      }
    }
    ],
    "Rout
      "de
    }
  }
```

Don’t forget to replace `<YOUR_GROQ_API_KEY>` with your actual Groq API key, and keep file permissions tight (`chmod 600`) to secure your key.

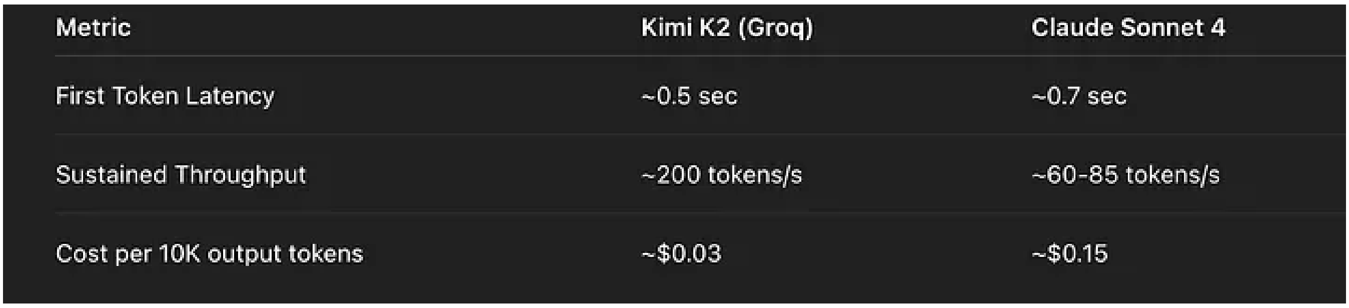
What Can You Actually Do?

A lot, actually. Here are some common tasks I’ve been using it for:

- **Generating Unit Tests Recursively:**
- **Massive Refactoring (like updating APIs)**

Performance in the Real World

Here’s what performance looks like based on my experience and some community benchmarks:



Metric	Kimi K2 (Groq)	Claude Sonnet 4
First Token Latency	~0.5 sec	~0.7 sec
Sustained Throughput	~200 tokens/s	~60-85 tokens/s
Cost per 10K output tokens	~$0.03	~$0.15

Simply put, you’re getting faster results at a fraction of the cost.

When Should You Stick to Sonnet?

This combo isn’t perfect for every scenario, though. You might want to stick with Claude Sonnet if you need:

- Complex UI tasks, I believe kimi k2 is still not that great at UI gen
- Planning and reasoning — combine kimi k2 along with claude and o3 for complex tasks is the best approach

If you code regularly — whether writing tests, refactoring large codebases, or just prototyping quickly — this setup can seriously streamline your workflow. It gives you instant speed at a much lower price.

Give it a shot, and let me know how it works out! Happy coding and hacking.

Follow

Written by Hungrysoul

258 followers · 3 following

Python Dev, Part time Bug Bounty Hunter & a Full time entrepreneur.

Responses (4)

Write a response

What are your thoughts?

Kofi Adom
Jul 21, 2025

How about combining gemini2.5 pro with kimi on claude code? Use gemini for plan and kimi for execution? What is your thought?

9 1 reply Reply

Henry Zhang
Jul 19, 2025

Very good article, very detailed.

1 Reply

Henry

Jul 19, 2

```
{
"name": "groq",
"api_base_url": "https://api.groq.com/openai/v1/chat/completions",
"api_key": "<YOUR_GROQ_API_KEY>",
"models": ["moonshotai/kimi-k2-instruct"],...
```

```
{

"name": "groq",

"api_base_url": "https://api.groq.com/openai/v1/chat/completions",

"api_key": "",

"models": [... more
```

1 Reply

See all responses

More from Hungrysoul