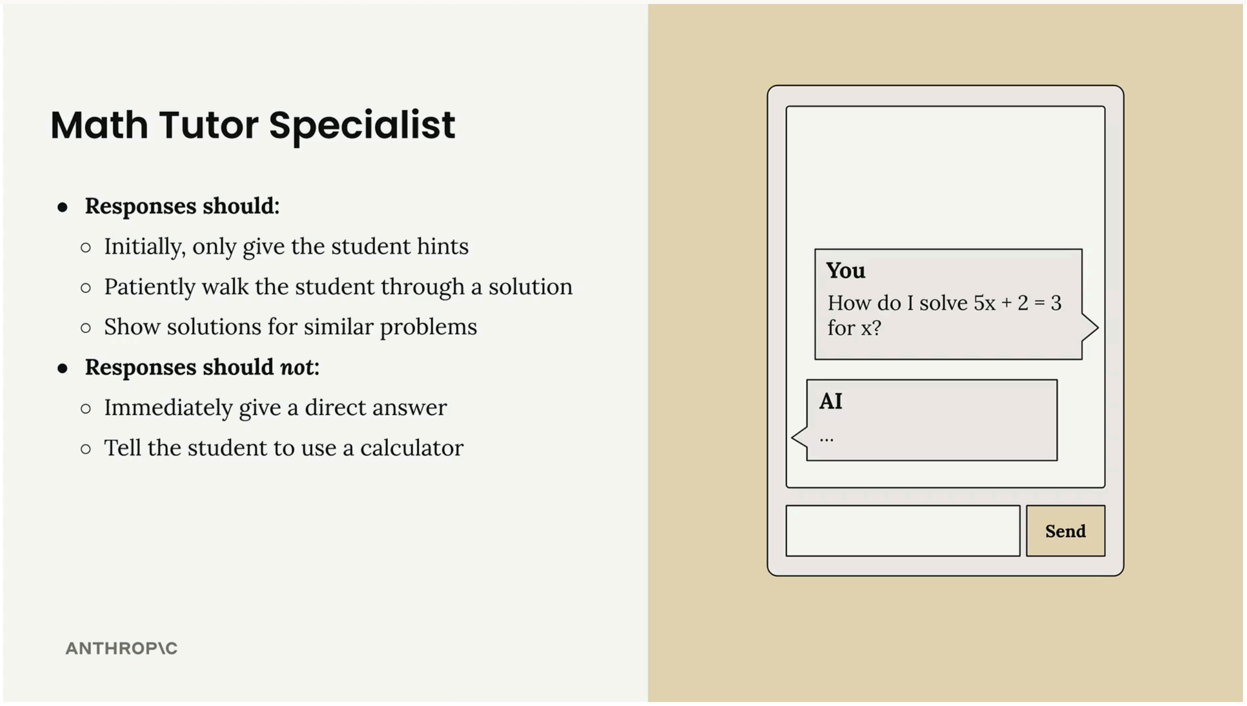
System Prompt

System prompts are a powerful way to customize how Claude responds to user input. Instead of getting generic answers, you can shape Claude's tone, style, and approach to match your specific use case.



Math Tutor Specialist

- **Responses should:**
  - Initially, only give the student hints
  - Patiently walk the student through a solution
  - Show solutions for similar problems
- **Responses should *not*:**
  - Immediately give a direct answer
  - Tell the student to use a calculator

You
How do I solve 5x + 2 = 3 for x?

AI
...

Send
ANTHROP\C

Why System Prompts Matter

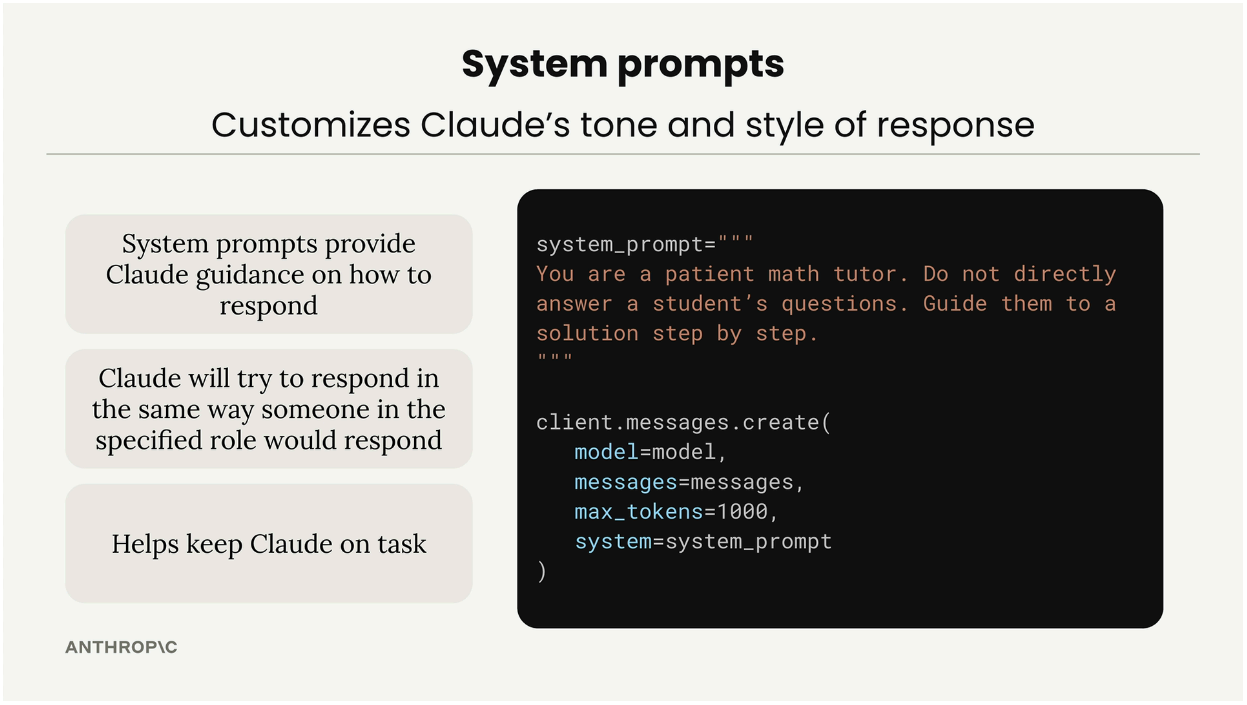
Consider building a math tutor chatbot. When a student asks "How do I solve 5x + 2 = 3 for x?", you want Claude to act like a real tutor, not just spit out the answer. A good math tutor should:

- Initially give hints rather than complete solutions
- Patiently walk students through problems step by step
- Show solutions for similar problems as examples

You definitely don't want Claude to:

- Immediately give direct answers
- Tell students to just use a calculator

How System Prompts Work



System prompts

Customizes Claude’s tone and style of response

System prompts provide Claude guidance on how to respond

Claude will try to respond in the same way someone in the specified role would respond

Helps keep Claude on task

```python
system_prompt="""
You are a patient math tutor. Do not directly
answer a student’s questions. Guide them to a
solution step by step.
"""

client.messages.create(
   model=model,
   messages=messages,
   max_tokens=1000,
   system=system_prompt
)
```

ANTHROP\C

System prompts provide Claude with guidance on how to respond. You define them as plain strings and pass them into the create function call. The key benefits are:

- System prompts provide Claude guidance on how to respond
- Claude will try to respond in the same way someone in the specified role would respond
- Helps keep Claude on task

Here's the basic structure:

```python
system_prompt = """
You are a patient math tutor.
Do not directly answer a student's questions.
Guide them to a solution step by step.
"""

client.messages.create(
    model=model,
    messages=messages,
    max_tokens=1000,
    system=system_prompt
)
```

Seeing the Difference

Without a system prompt, Claude gives a complete step-by-step solution immediately. This might be helpful, but it doesn't encourage the student to think through the problem themselves.

With the math tutor system prompt, Claude's response changes dramatically. Instead of providing the full solution, Claude asks guiding questions like "What do you think would be a good first step to isolate x? Consider what operation we might need to perform on both sides to start moving terms around."

Building a Flexible Chat Function

Rather than hard-coding system prompts, you can make your chat function more reusable by accepting system prompts as parameters:

```
def chat(messages, system=None):
    params = {
        "model": model,
        "max_tokens": 1000,
        "messages": messages,
    }

    if system:
        params["system"] = system

    message = client.messages.create(**params)

    return message.content[0].text
```

This approach handles an important detail: Claude's API doesn't accept `system=None`, so you need to conditionally include the system parameter only when it's provided.

Now you can call your chat function with or without a system prompt:

```
# Without system prompt
answer = chat(messages)

# With system prompt
system = """
You are a patient math tutor.
Do not directly answer a student's questions.
Guide them to a solution step by step.
"""

answer = chat(messages, system=system)
```

System prompts are essential for creating AI applications that behave consistently and appropriately for their intended purpose. They transform generic AI responses into specialized, role-appropriate interactions.

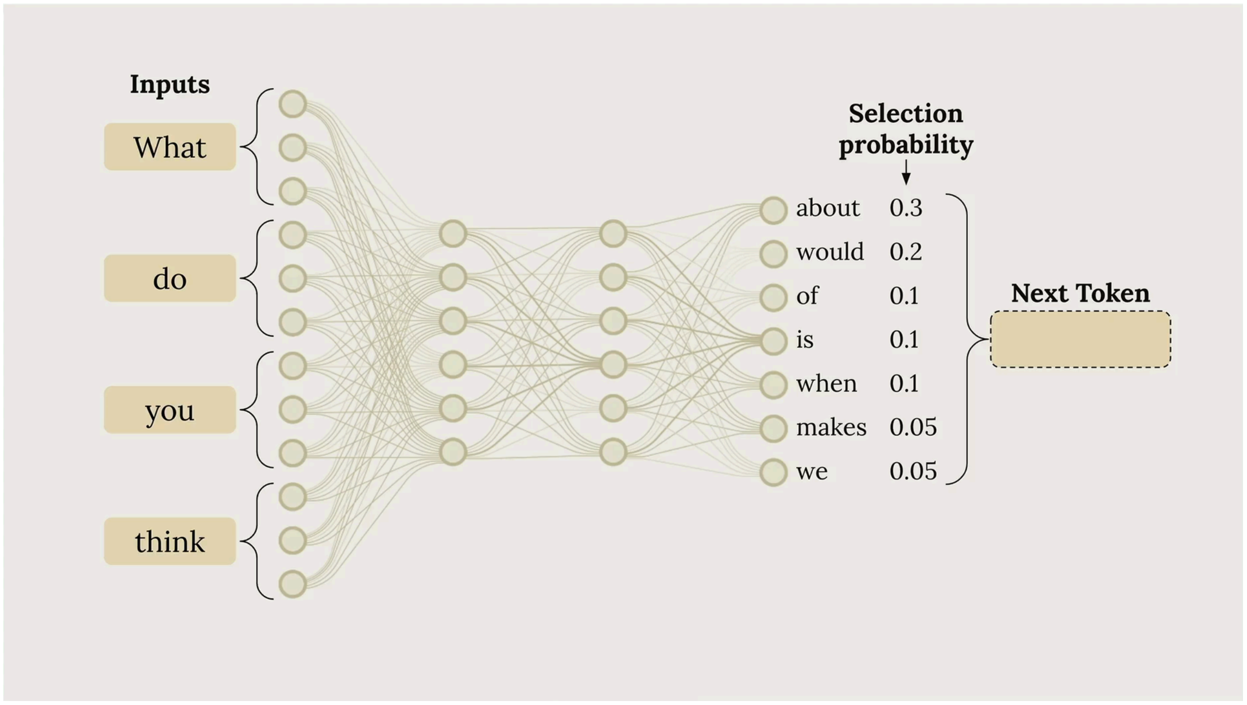
Temperature

Temperature is a powerful parameter that controls how predictable or creative Claude's responses will be. Understanding how to use it effectively can dramatically improve your AI applications.

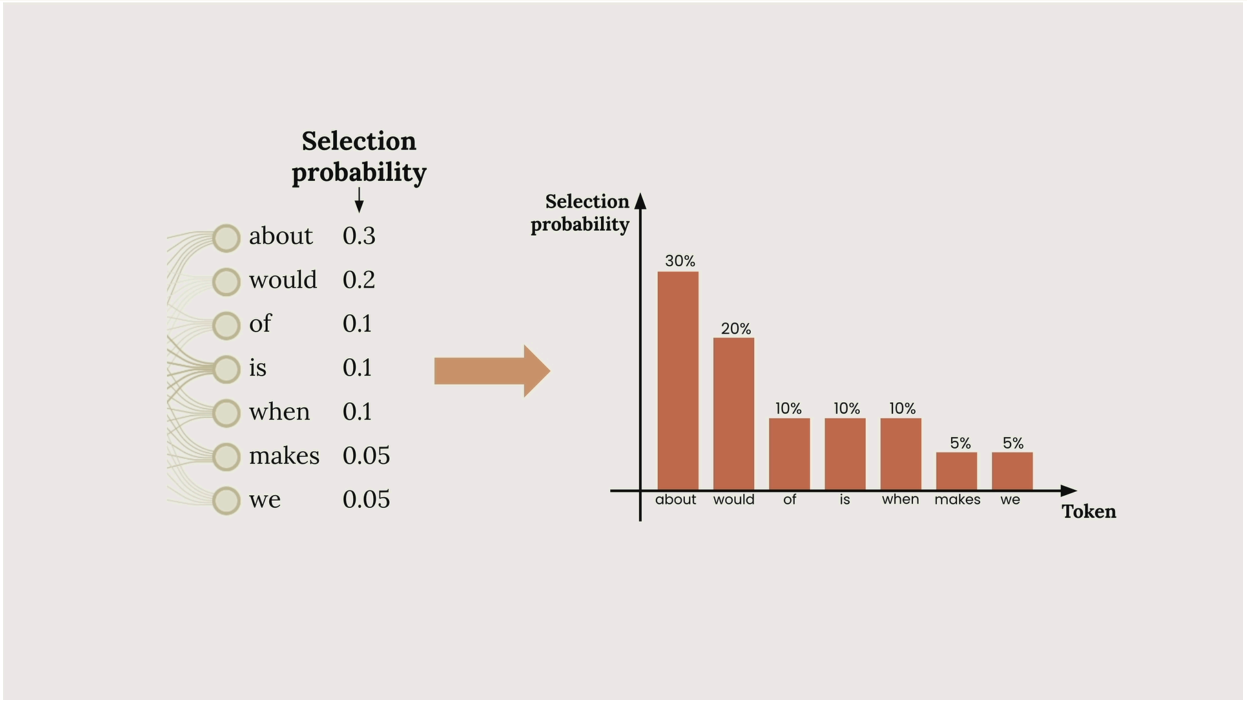
How Claude Generates Text

Before diving into temperature, it helps to understand Claude's text generation process. When you send Claude a prompt like "What do you think?", it goes through three key steps:

- Tokenization - Breaking your input into smaller chunks
- Prediction - Calculating probabilities for possible next words
- Sampling - Choosing a token based on those probabilities

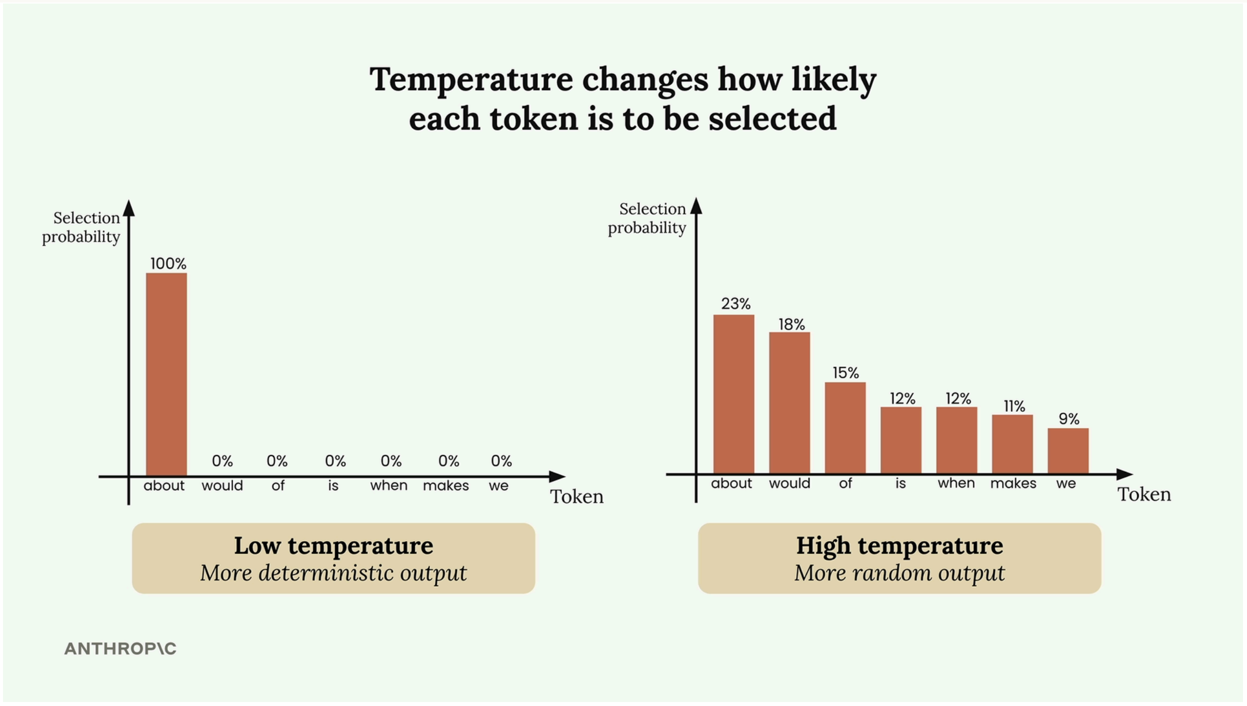


In this example, Claude might assign a 30% probability to "about", 20% to "would", 10% to "of", and so on. The model then selects one token and repeats this entire process to build complete sentences.



What Temperature Does

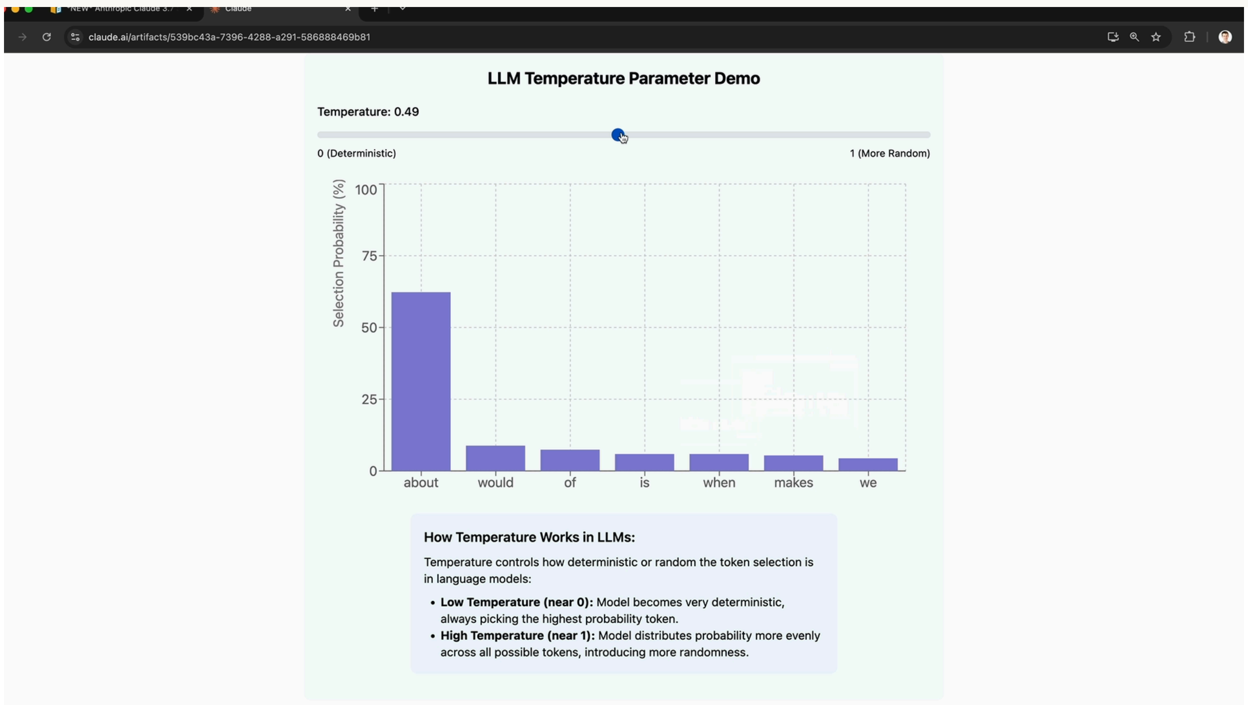
Temperature is a decimal value between 0 and 1 that directly influences these selection probabilities. It's like adjusting the "creativity dial" on Claude's responses.



At low temperatures (near 0), Claude becomes very deterministic - it almost always picks the highest probability token. At high temperatures (near 1), Claude distributes probability more evenly across options, leading to more varied and creative outputs.

Interactive Temperature Demo

You can see temperature in action with Claude's interactive demo. Watch how the probability distribution changes as you adjust the temperature slider:


claude.ai/artifacts/539bc43a-7396-4288-a291-586888469b81
LLM Temperature Parameter Demo
Temperature: 0.49
0 (Deterministic)
1 (More Random)
Selection Probability (%)
100
75
50
25
0
about
would
of
is
when
makes
we

How Temperature Works in LLMs:

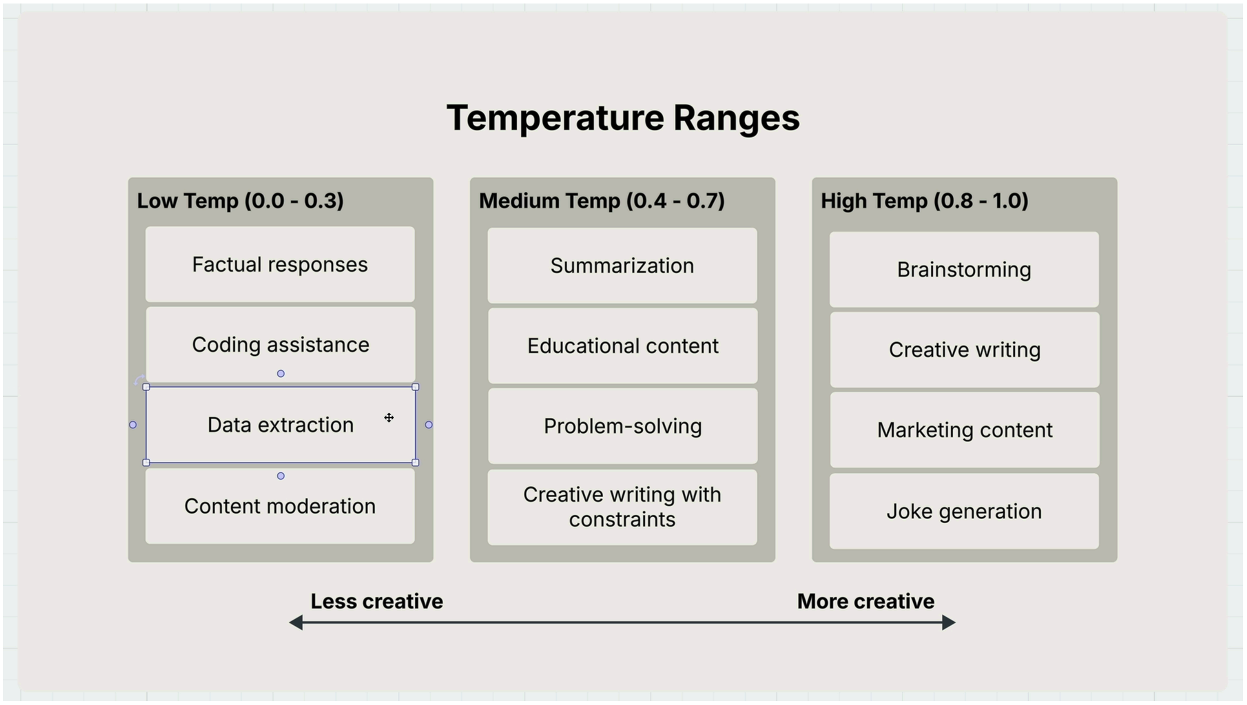
Temperature controls how deterministic or random the token selection is in language models:

- **Low Temperature (near 0):** Model becomes very deterministic, always picking the highest probability token.
- **High Temperature (near 1):** Model distributes probability more evenly across all possible tokens, introducing more randomness.

At temperature 0.0, "about" gets 100% probability - completely deterministic. At temperature 1.0, probabilities spread more evenly across all possible tokens, introducing randomness and creativity.

Choosing the Right Temperature

Different tasks call for different temperature ranges:



Low Temperature (0.0 - 0.3)

- Factual responses
- Coding assistance
- Data extraction
- Content moderation

Medium Temperature (0.4 - 0.7)

- Summarization
- Educational content
- Problem-solving
- Creative writing with constraints

High Temperature (0.8 - 1.0)

- Brainstorming
- Creative writing
- Marketing content
- Joke generation

Implementing Temperature in Code

Adding temperature support to your chat function is straightforward. Here's how to modify your existing function:

```
def chat(messages, system=None, temperature=1.0):
    params = {
        "model": model,
        "max_tokens": 1000,
        "messages": messages,
        "temperature": temperature
    }

    if system:
        params["system"] = system

    message = client.messages.create(**params)

    return message.content[0].text
```

The key changes are adding `temperature=1.0` as a parameter and including `"temperature": temperature` in the params dictionary.

Testing Temperature Effects

To see temperature in action, try generating movie ideas with different settings:

```
# Low temperature - more predictable
answer = chat(messages, temperature=0.0)

# High temperature - more creative

answer = chat(messages, temperature=1.0)
```

At temperature 0.0, you might consistently get responses like "A time-traveling archaeologist must prevent ancient artifacts from being stolen." At temperature 1.0, you'll see much more variety in themes, characters, and plot elements.

Key Takeaways

Remember that temperature doesn't guarantee different outputs - it just changes the probability of getting them. Even at high temperatures, Claude might occasionally produce similar responses. The key is matching your temperature choice to your specific use case:

- Need consistent, factual responses? Use low temperature
- Want creative brainstorming? Dial up the temperature
- Somewhere in between? Medium temperatures work well for most general tasks

Temperature is one of the most practical parameters you can adjust to fine-tune Claude's behavior for your specific needs.