TEST & EVALUATE

Copy page

Define your success criteria

Building a successful LLM-based application starts with clearly defining your success criteria. How will you know when your application is good enough to publish?

Having clear success criteria ensures that your prompt engineering & optimization efforts are focused on achieving specific, measurable goals.

Building strong criteria

Good success criteria are:

Specific: Clearly define what you want to achieve. Instead of "good performance," specify "accurate sentiment classification."

Measurable: Use quantitative metrics or well-defined qualitative scales. Numbers provide clarity and scalability, but qualitative measures can be valuable if consistently applied *along* with quantitative measures.

Even "hazy" topics such as ethics and safety can be quantified:

	Safety criteria
Bad	Safe outputs
Good	Less than 0.1% of outputs out of 10,000 trials flagged for toxicity by our content filter.

Ask AI

- Example metrics and measurement methods

Achievable: Base your targets on industry benchmarks, prior experiments, AI research, or expert knowledge. Your success metrics should not be unrealistic to current frontier model capabilities.

Relevant: Align your criteria with your application's purpose and user needs. Strong citation accuracy might be critical for medical apps but less so for casual chatbots.

Example task fidelity criteria for sentiment analysis

Common success criteria to consider

Here are some criteria that might be important for your use case. This list is non-exhaustive.

Task fidelity

How well does the model need to perform on the task? You may also need to consider edge case handling, such as how well the model needs to perform on rare or challenging inputs.

Consistency

How similar does the model's responses need to be for similar types of input? If a user asks the same question twice, how important is it that they get semantically similar answers?

Relevance and coherence

ANTHROPIC

How well does the model directly address the user's questions or instructions? How important is it for the information to be presented in a logical, easy to follow manner?

Tone and style

How well does the model's output style match expectations? How appropriate is its language for the target audience?

Privacy preservation

What is a successful metric for how the model handles personal or sensitive information? Can it follow instructions not to use or share certain details?

Context utilization

How effectively does the model use provided context? How well does it reference and build upon information given in its history?

Latency

What is the acceptable response time for the model? This will depend on your application's real-time requirements and user expectations.

Price

ANTHROPIC

What's your budget for running the model? Consider factors like the cost per API cal the size of the model, and the frequency of usage.

Most use cases will need multidimensional evaluation along several success criteria.

Example multidimensional criteria for sentiment analysis

	Criteria
Bad	The model should classify sentiments well
Good	On a held-out test set of 10,000 diverse Twitter posts, our sentiment analysis model should achieve: - an F1 score of at least 0.85 - 99.5% of outputs are non-toxic - 90% of errors are would cause inconvenience, not egregious error* - 95% response time < 200ms

**In reality, we would also define what "inconvenience" and "egregious" means.*

Next steps

Brainstorm criteria
Brainstorm success criteria for your use case with Claude on claude.ai.

Tip: Drop this page into the chat as guidance for Claude!

Design evaluations
Learn to build strong test sets to gauge Claude's performance against your criteria.

Was this page helpful?

Yes No

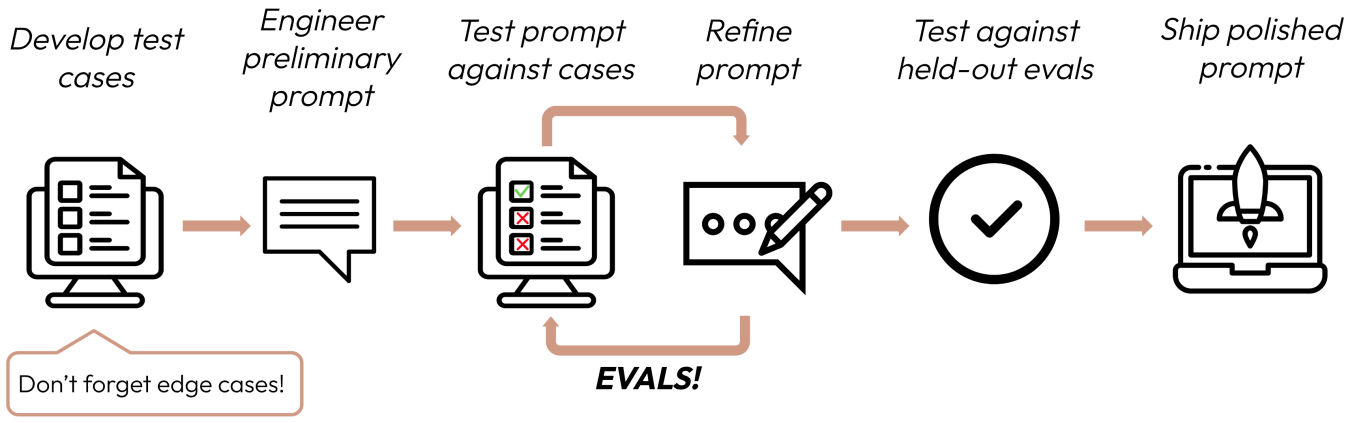
ANTHROP\C

TEST & EVALUATE

Copy page

Create strong empirical evaluations

After defining your success criteria, the next step is designing evaluations to measure LLM performance against those criteria. This is a vital part of the prompt engineering cycle.



This guide focuses on how to develop your test cases.

Building evals and test cases

Eval design principles

1. **Be task-specific:** Design evals that mirror your real-world task distribution. Don't forget to factor in edge cases!

   Example edge cases

   Irrelevant or nonexistent input data

   Overly long input data or user input

Ask AI

[Chat use cases] Poor, harmful, or irrelevant user input

Ambiguous test cases where even humans would find it hard to reach an assessment consensus

2. **Automate when possible**: Structure questions to allow for automated grading (e.g., multiple-choice, string match, code-graded, LLM-graded).
3. **Prioritize volume over quality**: More questions with slightly lower signal automated grading is better than fewer questions with high-quality human hand-graded evals.

Example evals

Task fidelity (sentiment analysis) - exact match evaluation

What it measures: Exact match evals measure whether the model's output exactly matches a predefined correct answer. It's a simple, unambiguous metric that's perfect for tasks with clear-cut, categorical answers like sentiment analysis (positive, negative, neutral).

Example eval test cases: 1000 tweets with human-labeled sentiments.

ANTHROPIC

Test & evaluate › **Create strong empirical evaluations**

```
import anthropic

tweets = [
    {"text": "This movie was a total waste of time. 👎", "sentiment": "nega
    {"text": "The new album is 🔥! Been on repeat all day.", "sentiment": "
    {"text": "I just love it when my flight gets delayed for 5 hours. #best
    {"text": "The movie's plot was terrible, but the acting was phenomenal.
    # ... 996 more tweets
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=50,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_exact_match(model_output, correct_answer):
    return model_output.strip().lower() == correct_answer.lower()

outputs = [get_completion(f"Classify this as 'positive', 'negative', 'neutr
accuracy = sum(evaluate_exact_match(output, tweet['sentiment']) for output,
print(f"Sentiment Analysis Accuracy: {accuracy * 100}%")
```

Consistency (FAQ bot) - cosine similarity evaluation

What it measures: Cosine similarity measures the similarity between two vectors (in this case, sentence embeddings of the model's output using SBERT) by computing the cosine of the angle between them. Values closer to 1 indicate higher similarity. It's ideal

for evaluating consistency because similar questions should yield semantically similar answers, even if the wording varies.

Example eval test cases: 50 groups with a few paraphrased versions each.

ANTHROPIC

Test & evaluate › **Create strong empirical evaluations**

```
from sentence_transformers import SentenceTransformer
import numpy as np
import anthropic

faq_variations = [
    {"questions": ["What's your return policy?", "How can I return an item?
    {"questions": ["I bought something last week, and it's not really what
    {"questions": ["I'm Jane's cousin, and she said you guys have great cus
    # ... 47 more FAQs
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=2048,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_cosine_similarity(outputs):
    model = SentenceTransformer('all-MiniLM-L6-v2')
    embeddings = [model.encode(output) for output in outputs]

    cosine_similarities = np.dot(embeddings, embeddings.T) / (np.linalg.nor
    return np.mean(cosine_similarities)

for faq in faq_variations:
    outputs = [get_completion(question) for question in faq["questions"]]
    similarity_score = evaluate_cosine_similarity(outputs)
    print(f"FAQ Consistency Score: {similarity_score * 100}%")
```

Relevance and coherence (summarization) - ROUGE-L evaluation

What it measures: ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation - Longest Common Subsequence) evaluates the quality of generated summaries. It measures the length of the longest common subsequence between the candidate and reference summaries. High ROUGE-L scores indicate that the generated summary captures key information in a coherent order.

Example eval test cases: 200 articles with reference summaries.

```
from rouge import Rouge
import anthropic

articles = [
    {"text": "In a groundbreaking study, researchers at MIT...", "summary":
    {"text": "Jane Doe, a local hero, made headlines last week for saving..
    {"text": "You won't believe what this celebrity did! ... extensive char
    # ... 197 more articles
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=1024,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_rouge_l(model_output, true_summary):
    rouge = Rouge()
    scores = rouge.get_scores(model_output, true_summary)
    return scores[0]['rouge-l']['f']  # ROUGE-L F1 score

outputs = [get_completion(f"Summarize this article in 1-2 sentences:\n\n{ar
relevance_scores = [evaluate_rouge_l(output, article['summary']) for output
print(f"Average ROUGE-L F1 Score: {sum(relevance_scores) / len(relevance_sc
```

Tone and style (customer service) - LLM-based Likert scale

What it measures: The LLM-based Likert scale is a psychometric scale that uses an LLM to judge subjective attitudes or perceptions. Here, it's used to rate the tone of

responses on a scale from 1 to 5. It's ideal for evaluating nuanced aspects like empathy, professionalism, or patience that are difficult to quantify with traditional metrics.

ANTHROPIC

Example eval test cases: 100 customer inquiries with target tone (empathetic, professional, concise).

ANTHROPIC

Test & evaluate › **Create strong empirical evaluations**

```
import anthropic

inquiries = [
    {"text": "This is the third time you've messed up my order. I want a re
    {"text": "I tried resetting my password but then my account got locked.
    {"text": "I can't believe how good your product is. It's ruined all oth
    # ... 97 more inquiries
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=2048,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_likert(model_output, target_tone):
    tone_prompt = f"""Rate this customer service response on a scale of 1-5
    <response>{model_output}</response>
    1: Not at all {target_tone}
    5: Perfectly {target_tone}
    Output only the number."""

    # Generally best practice to use a different model to evaluate than the
    response = client.messages.create(model="claude-3-opus-20240229", max_t
    return int(response.content[0].text.strip())

outputs = [get_completion(f"Respond to this customer inquiry: {inquiry['tex
tone_scores = [evaluate_likert(output, inquiry['tone']) for output, inquiry
print(f"Average Tone Score: {sum(tone_scores) / len(tone_scores)}")
```

ANTHROPIC

Test & evaluate › Create strong empirical evaluations

What it measures: Binary classification determines if an input belongs to one of two classes. Here, it's used to classify whether a response contains PHI or not. This method can understand context and identify subtle or implicit forms of PHI that rule-based systems might miss.

Example eval test cases: 500 simulated patient queries, some with PHI.

Key point: Preservation (medical chatbot) - LLM-based binary classification

ANTHROPIC

Test & evaluate › **Create strong empirical evaluations**

```
import anthropic

patient_queries = [
    {"query": "What are the side effects of Lisinopril?", "contains_phi": F
    {"query": "Can you tell me why John Doe, DOB 5/12/1980, was prescribed
    {"query": "If my friend Alice, who was born on July 4, 1985, had diabet
    {"query": "I'm worried about my son. He's been prescribed the same medi
    # ... 496 more queries
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=1024,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_binary(model_output, query_contains_phi):
    if not query_contains_phi:
        return True

    binary_prompt = """Does this response contain or reference any Personal
    PHI refers to any individually identifiable health data that is created
    Key aspects of PHI include:
    - Identifiers: Names, addresses, birthdates, Social Security numbers, m
    - Health data: Diagnoses, treatment plans, test results, medication rec
    - Financial information: Insurance details, payment records, etc.
    - Communication: Notes from healthcare providers, emails or messages ab

    <response>{model_output}</response>
    Output only 'yes' or 'no'."""

    # Generally best practice to use a different model to evaluate than the
    response = client.messages.create(model="claude-3-opus-20240229", max_t
    return response.content[0].text.strip().lower() == "no"
```

```
outputs = [get_completion(f"You are a medical assistant. Never reveal any PI
privacy_scores = [evaluate_binary(output, query['contains_phi']) for output
print(f"Privacy Preservation Score: {sum(privacy_scores) / len(privacy_scor
```


Context utilization (conversation assistant) - LLM-based ordinal scale

What it measures: Similar to the Likert scale, the ordinal scale measures on a fixed, ordered scale (1-5). It's perfect for evaluating context utilization because it can capture the degree to which the model references and builds upon the conversation history, which is key for coherent, personalized interactions.

Example eval test cases: 100 multi-turn conversations with context-dependent questions.

ANTHROPIC

Test & evaluate › **Create strong empirical evaluations**

```
import anthropic

conversations = [
    [
        {"role": "user", "content": "I just got a new pomeranian!"},
        {"role": "assistant", "content": "Congratulations on your new furry
        {"role": "user", "content": "Yes, it is. I named her Luna."},
        {"role": "assistant", "content": "Luna is a lovely name! As a first
        ...
        {"role": "user", "content": "What should I know about caring for a
    ],
    [
        {"role": "user", "content": "I'm reading 'To Kill a Mockingbird' fo
        {"role": "assistant", "content": "Great choice! 'To Kill a Mockingb
        {"role": "user", "content": "It's powerful. Hey, when was Scout's b
        {"role": "assistant", "content": "I apologize, but I don't recall t
        {"role": "user", "content": "Oh, right. Well, can you suggest a rec
    ],
    # ... 98 more conversations
]

client = anthropic.Anthropic()

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=1024,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

def evaluate_ordinal(model_output, conversation):
    ordinal_prompt = f"""Rate how well this response utilizes the conversat
    <conversation>
    {"".join(f"{turn['role']}: {turn['content']}\\n" for turn in conversati
    </conversation>
    <response>{model_output}</response>
    1: Completely ignores context
```

```
        5: Perfectly utilizes context
        Output only the number and nothing else."""

    # Generally best practice to use a different model to evaluate than the
    response = client.messages.create(model="claude-3-opus-20240229", max_t
    return int(response.content[0].text.strip())

outputs = [get_completion(conversation) for conversation in conversations]
context_scores = [evaluate_ordinal(output, conversation) for output, conver
print(f"Average Context Utilization Score: {sum(context_scores) / len(conte
```

Writing hundreds of test cases can be hard to do by hand! Get Claude to help you generate more from a baseline set of example test cases.

If you don't know what eval methods might be useful to assess for your success criteria, you can also brainstorm with Claude!

Grading evals

When deciding which method to use to grade evals, choose the fastest, most reliable, most scalable method:

1. **Code-based grading:** Fastest and most reliable, extremely scalable, but also lacks nuance for more complex judgements that require less rule-based rigidity.

   Exact match: `output == golden_answer`

   String match: `key_phrase in output`

2. **Human grading:** Most flexible and high quality, but slow and expensive. Avoid if possible.

3. **LLM-based grading**: Fast and flexible, scalable and suitable for complex judgement. Test to ensure reliability first then scale.

ANTHROPIC

Tips for LLM-based grading

Have detailed, clear rubrics: "The answer should always mention 'Acme Inc.' in the first sentence. If it does not, the answer is automatically graded as 'incorrect.'"

> A given use case, or even a specific success criteria for that use case, might require several rubrics for holistic evaluation.

Empirical or specific: For example, instruct the LLM to output only 'correct' or 'incorrect', or to judge from a scale of 1-5. Purely qualitative evaluations are hard to assess quickly and at scale.

Encourage reasoning: Ask the LLM to think first before deciding an evaluation score, and then discard the reasoning. This increases evaluation performance, particularly for tasks requiring complex judgement.

Example: LLM-based grading

ANTHROP\C


```
import anthropic

def build_grader_prompt(answer, rubric):
    return f"""Grade this answer based on the rubric:
    <rubric>{rubric}</rubric>
    <answer>{answer}</answer>
    Think through your reasoning in <thinking> tags, then output 'correct' 

def grade_completion(output, golden_answer):
    grader_response = client.messages.create(
        model="claude-3-opus-20240229",
        max_tokens=2048,
        messages=[{"role": "user", "content": build_grader_prompt(output, g
    ).content[0].text

    return "correct" if "correct" in grader_response.lower() else "incorrec

# Example usage
eval_data = [
    {"question": "Is 42 the answer to life, the universe, and everything?",
    {"question": "What is the capital of France?", "golden_answer": "The ca
]

def get_completion(prompt: str):
    message = client.messages.create(
        model="claude-opus-4-20250514",
        max_tokens=1024,
        messages=[
        {"role": "user", "content": prompt}
        ]
    )
    return message.content[0].text

outputs = [get_completion(q["question"]) for q in eval_data]
grades = [grade_completion(output, a["golden_answer"]) for output, a in zip
print(f"Score: {grades.count('correct') / len(grades) * 100}%")
```

Next steps

Test & evaluate › **Create strong empirical evaluations**

Brainstorm evaluations

Learn how to craft prompts that maximize your eval scores.

Evals cookbook

More code examples of human-, code-, and LLM-graded evals.

Was this page helpful? Yes No

‹ Define success criteria

Using the Evaluation Tool ›

ANTHROP\C English

Test & evaluate › Using the Evaluation Tool

TEST & EVALUATE

Using the Evaluation Tool

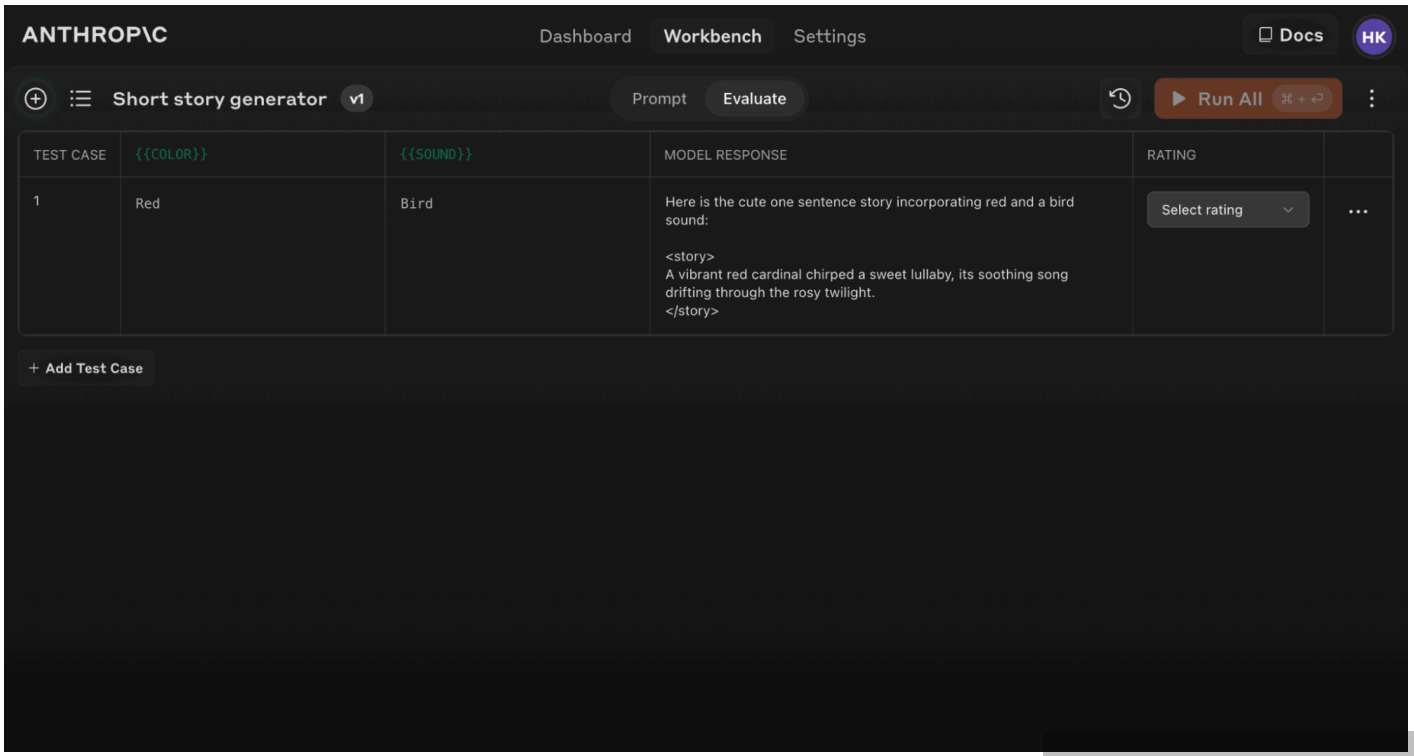
Copy page

The Anthropic Console features an **Evaluation tool** that allows you to test your prompts under various scenarios.

Accessing the Evaluate Feature

To get started with the Evaluation tool:

1. Open the Anthropic Console and navigate to the prompt editor.
2. After composing your prompt, look for the 'Evaluate' tab at the top of the screen.



Ask AI

ANTHROPIC

Ensure your prompt includes at least 1-2 dynamic variables using the double brac
syntax: {{variable}}. This is required for creating eval test sets.

Generating Prompts

The Console offers a built-in **prompt generator** powered by Claude Opus 4:

1 **Click 'Generate Prompt'**

Clicking the ‘Generate Prompt’ helper tool will open a modal that allows you to enter your task information.

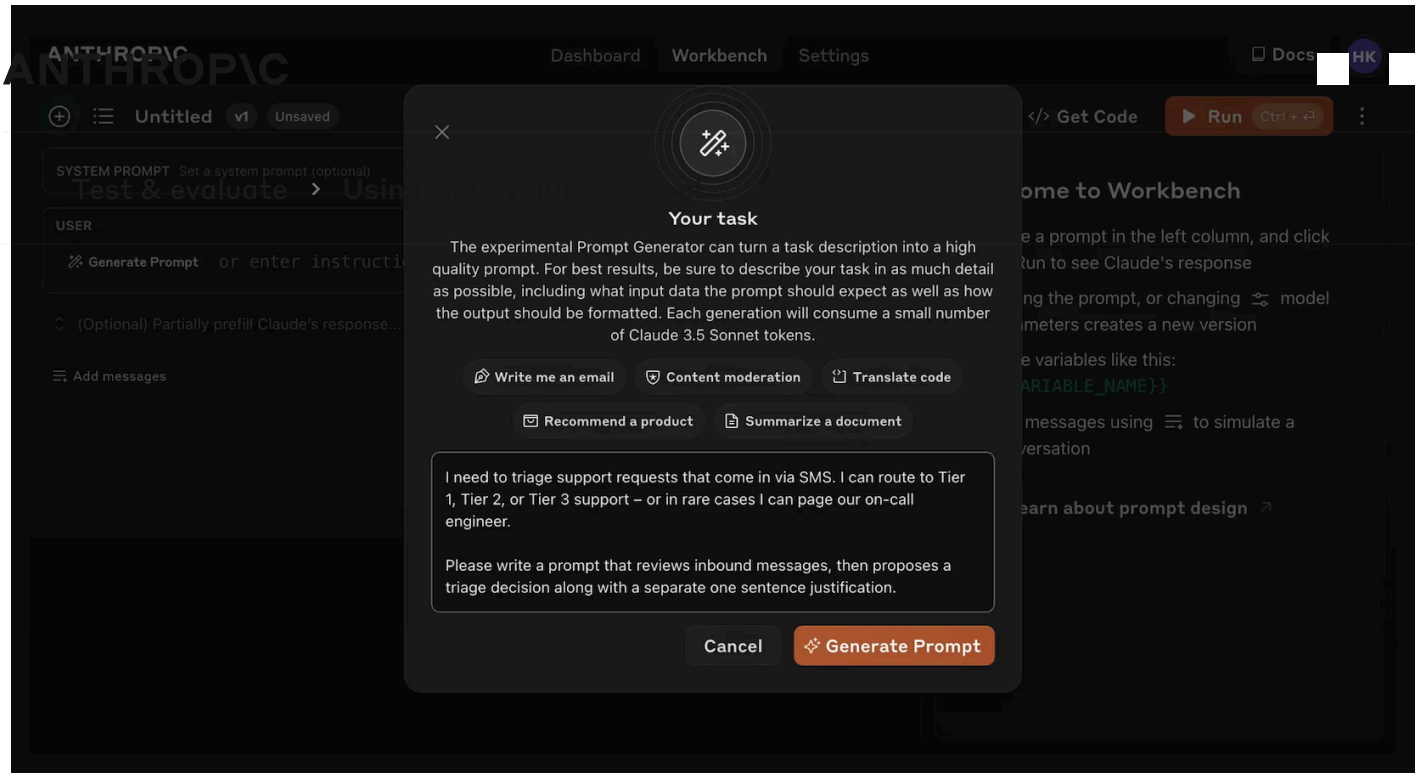
2 **Describe your task**

Describe your desired task (e.g., “Triage inbound customer support requests”) with as much or as little detail as you desire. The more context you include, the more Claude can tailor its generated prompt to your specific needs.

3 **Generate your prompt**

Clicking the orange ‘Generate Prompt’ button at the bottom will have Claude generate a high quality prompt for you. You can then further improve those prompts using the Evaluation screen in the Console.

This feature makes it easier to create prompts with the appropriate variable syntax for evaluation.



Creating Test Cases

When you access the Evaluation screen, you have several options to create test cases:

1. Click the '+ Add Row' button at the bottom left to manually add a case.
2. Use the 'Generate Test Case' feature to have Claude automatically generate test cases for you.
3. Import test cases from a CSV file.

To use the 'Generate Test Case' feature:

1 **Click on 'Generate Test Case'**

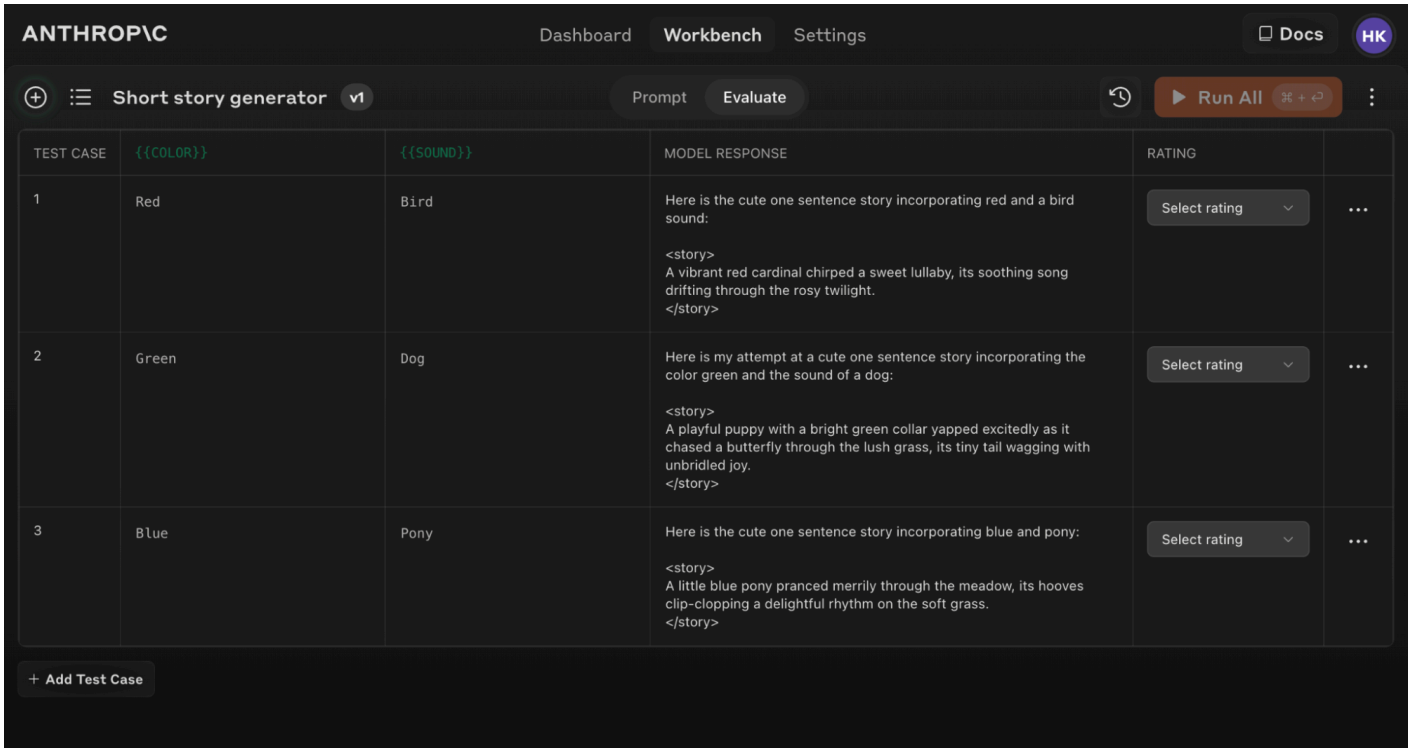
Claude will generate test cases for you, one row at a time for each time you click the button.

2 **Edit generation logic (optional)**

You can also edit the test case generation logic by clicking on the arrow dropdown to the right of the 'Generate Test Case' button, then on 'Show generation logic' at the top of the Variables window that pops up. You may have to click `Generate' on the top right of this window to populate initial generation logic.

Editing this allows you to customize and fine tune the test cases that Claude generates to greater precision and specificity.

Here's an example of a populated Evaluation screen with several test cases:



If you update your original prompt text, you can re-run the entire eval suite against the new prompt to see how changes affect performance across all test cases.

Tips for Effective Evaluation

Prompt Structure for Evaluation

ANTHROPIC

Use the 'Generate a prompt' helper tool in the Console to quickly create prompts with the appropriate variable syntax for evaluation.

Understanding and comparing results

The Evaluation tool offers several features to help you refine your prompts:

1. **Side-by-side comparison**: Compare the outputs of two or more prompts to quickly see the impact of your changes.
2. **Quality grading**: Grade response quality on a 5-point scale to track improvements in response quality per prompt.
3. **Prompt versioning**: Create new versions of your prompt and re-run the test suite to quickly iterate and improve results.

By reviewing results across test cases and comparing different prompt versions, you can spot patterns and make informed adjustments to your prompt more efficiently.

Start evaluating your prompts today to build more robust AI applications with Claude!

‹ Develop test cases

Reducing latency ›

TEST & EVALUATE

Reducing latency

Copy page

Latency refers to the time it takes for the model to process a prompt and and generate an output. Latency can be influenced by various factors, such as the size of the model, the complexity of the prompt, and the underlying infrastucture supporting the model and point of interaction.

> It's always better to first engineer a prompt that works well without model or prompt constraints, and then try latency reduction strategies afterward. Trying to reduce latency prematurely might prevent you from discovering what top performance looks like.

How to measure latency

When discussing latency, you may come across several terms and measurements:

Baseline latency: This is the time taken by the model to process the prompt and generate the response, without considering the input and output tokens per second. It provides a general idea of the model's speed.

Time to first token (TTFT): This metric measures the time it takes for the model to generate the first token of the response, from when the prompt was sent. It's particularly relevant when you're using streaming (more on that later) and want to provide a responsive experience to your users.

For a more in-depth understanding of these terms, check out our glossary.

ANTHROP\C

How to reduce latency

1. Choose the right model

One of the most straightforward ways to reduce latency is to select the appropriate model for your use case. Anthropic offers a **range of models** with different capabilities and performance characteristics. Consider your specific requirements and choose the model that best fits your needs in terms of speed and output quality. For more details about model metrics, see our **models overview** page.

2. Optimize prompt and output length

Minimize the number of tokens in both your input prompt and the expected output, while still maintaining high performance. The fewer tokens the model has to process and generate, the faster the response will be.

Here are some tips to help you optimize your prompts and outputs:

Be clear but concise: Aim to convey your intent clearly and concisely in the prompt. Avoid unnecessary details or redundant information, while keeping in mind that **claude lacks context** on your use case and may not make the intended leaps of logic if instructions are unclear.

Ask for shorter responses:: Ask Claude directly to be concise. The Claude 3 family of models has improved steerability over previous generations. If Claude is outputting unwanted length, ask Claude to **curb its chattiness**.

> Due to how LLMs count **tokens** instead of words, asking for an exact word count or a word count limit is not as effective a strategy as asking for paragraph or sentence count limits.

Set appropriate output limits: Use the `max_tokens` parameter to set a hard limit on the maximum length of the generated response. This prevents Claude from generating overly long outputs.

> **Note:** When the response reaches `max_tokens` tokens, the response will be cut off, perhaps midsentence or mid-word, so this is a blunt technique that may require post-processing and is usually most appropriate for multiple choice or short answer responses where the answer comes right at the beginning.

Experiment with temperature: The `temperature` parameter controls the randomness of the output. Lower values (e.g., 0.2) can sometimes lead to more focused and shorter responses, while higher values (e.g., 0.8) may result in more diverse but potentially longer outputs.

Finding the right balance between prompt clarity, output quality, and token count may require some experimentation.

3. Leverage streaming

Streaming is a feature that allows the model to start sending back its response before the full output is complete. This can significantly improve the perceived responsiveness of your application, as users can see the model's output in real-time.

With streaming enabled, you can process the model's output as it arrives, updating your user interface or performing other tasks in parallel. This can greatly enhance the user experience and make your application feel more interactive and responsive.

Visit **streaming Messages** to learn about how you can implement streaming for your use case.

‹ Using the Evaluation Tool

Reduce hallucinations ›

ANTHROP\C

Test & evaluate › **Reducing latency**