Generative Language Models With Self-Consistency Prompting On Amazon Bedrock

Introduction

Amazon Bedrock offers a very exciting way to explore the landscape of generative language models with its fully managed service. It provides access to high-performing foundation models and a variety of capabilities to build innovative AI applications.
This blog is a guide to optimizing generative language models by using the power of self-consistency prompting through Amazon Bedrock.
Whether you have experience with language models or are someone new to the field, let's embark on this journey together and understand the potential of generative AI.

How To Enhance Generative Language Models With Self-Consistency Prompting?

What is Amazon Bedrock?

Amazon Bedrock is a fully managed service that allows you to use multiple high-performing foundation models from leading AI companies and Amazon. This is done via a single API.

This comes with a large set of capabilities to build extraordinary generative AI applications with security, privacy, and responsible AI.
You can use Amazon Bedrock to run inference with foundation models using the batch inference API. This is done in batches and gets responses more efficiently.

In contrast to the more popular single-generation approaches like CoT (chain-of-thought), the self-consistency procedure produces a huge range of model completions that lead to even more consistent solutions. The generation of these diversified responses for a given prompt/task occurs due to the use of stochastic rather than greedy decoding strategies.

What are Generative Language Models?

Generative language models can be described as a type of artificial intelligence (AI) model that is designed to generate human-like text. They are able to imitate human beings in a manner related to how we respond, think, etc.
These models are trained on large datasets of text and learn to predict the likelihood of a sequence of words when given a starting prompt.
However, in order to use these Generative language models to the best of their ability, we need the ability to engineer effective prompts.

Different Prompting Techniques

Series Prompting Technique

- First, break the prompt into multiple sequential prompts.
- We see that this technique allows outputting more structured and informative results by avoiding irrelevant information in the output.

Example - Say you want to write a blog on music therapy and its benefits.
So, the way you achieve this is simple - you prompt the AI in steps to write the introduction, then the body, and finally, the conclusion. The output from the first prompt is used as input to the second prompt. This cycle just goes on.

Parallel Prompting Technique

- This involves breaking your prompt into chunks and then combining them.
- This technique outputs very diverse and interesting results
- You can use it to get different tones and styles in one combined output.

Example - You want to write a blog on art and its benefits. So you'd ask the language model these:

Chunk 1: "Write a brief history of drawing."
Chunk 2: Explain the different methodologies in art.
Chunk 3: Discuss the psychological benefits of art.

Chunk 4: Share fun anecdotes about the impact of art.

And then combine them with another prompt.

Looping Prompting Technique

This technique is used by repeatedly requesting the same prompt multiple times until you get the desired result, each time asking AI to do/add an extra bit.
It can be used in combination with Series and Parallel Prompts for better results.
Example -

- You give the AI a prompt to write an introduction to music and its benefits.
- The output provided is an introduction to music and highlights some of its benefits, such as stress reduction and improved mood. You can again give a prompt to add more benefits till you get the desired output.

Chain-of-Thought (CoT)

Greedy CoT is a traditional method that was previously used by Amazon Bedrock until very recently (2023).
This prompting enables even the most complicated reasoning capabilities through intermediate reasoning steps.

- You can combine it with a couple of few-shot prompts.
- This then gives better results on more complex tasks that require reasoning before responding.

Example -
Prompt: The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
AI Output: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.

Self-Consistency Prompting

Self-consistency prompting is the method currently used by Amazon Bedrock. It is used to improve the performance of generative language models.
The techniques use a huge variety of stochastic decoding to achieve this goal in three steps:

- The technique is used to prompt the language model with CoT examples to elicit reasoning.
- This can completely replace greedy decoding with sampling strategies to generate a diverse set of reasoning paths.
- Aggregate the outputs to find the most consistent answer in the response set.

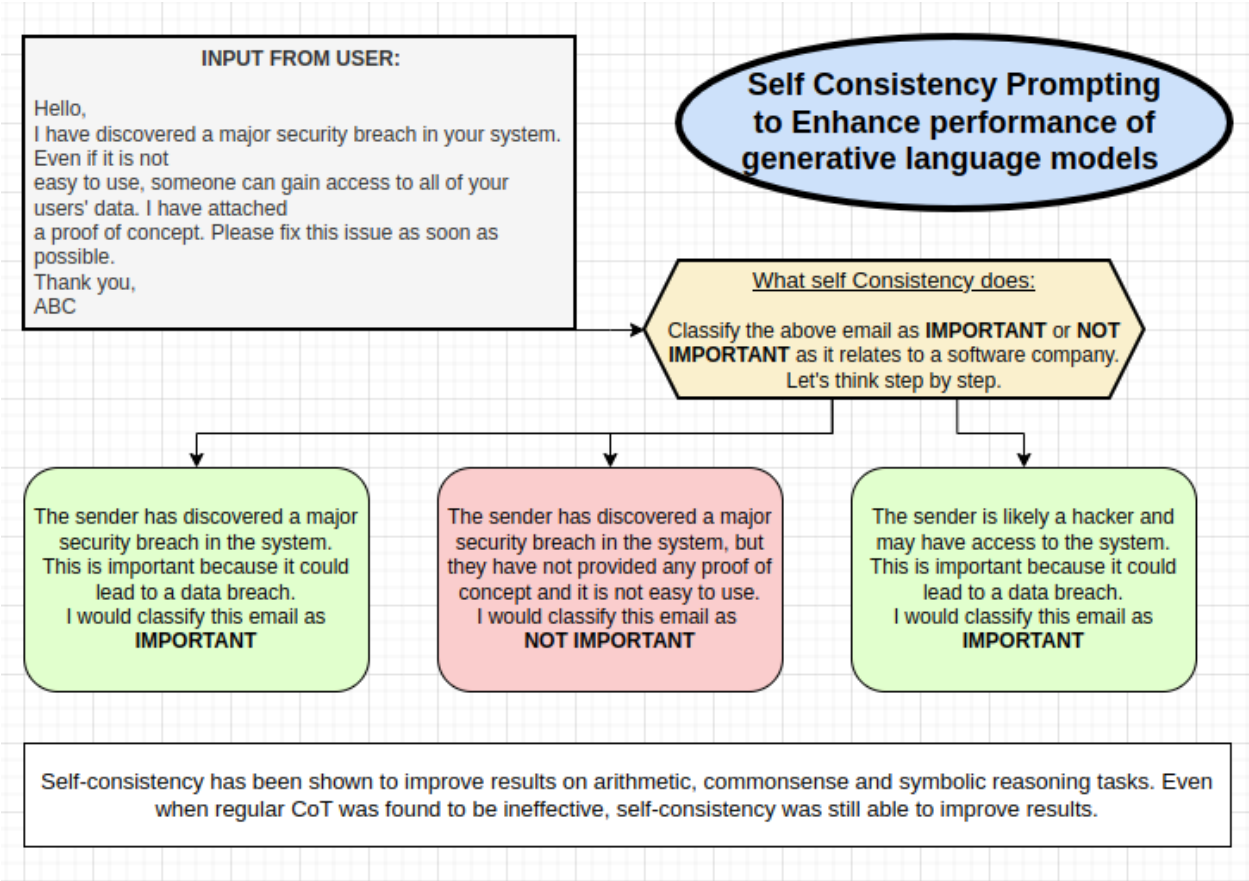

Fig. 1

Steps To Implement Self-Consistency Prompting On Amazon Bedrock

To implement self-consistency prompting on Amazon Bedrock:

- Choose the foundation model that best suits your needs and download an AWS account with a sagemaker-hosted notebook instance.
- Access the batch inference API provided by Amazon Bedrock to run inference efficiently.
- Incorporate self-consistency prompting into your workflow by generating multiple responses for a given prompt using stochastic decoding strategies.
- Upload import data to Amazon S3.
- Aggregate responses using the sample-and-marginalize procedure for enhanced consistency and reliability.
- Finally, try to evaluate the performance of your enhanced generative language model very effectively using self-consistency prompting. You can use Sagemaker for this.

```
ananya@sourav-H610M-H-V2-DDR4:~$ pip3 install -q $(ls ./bedrock-python-sdk-reinv
ent/boto3-*.whl | head -1)
```

Fig. 2

```
data = [
    {
        "recordId": "1",
        "modelInput": {
            "prompt": Complete the sentence "Once upon a time...",
            "temperature": 0.7,
            "max_tokens": 100,
            "num_generations": 2,
        },
    },
]
```

Fig. 3

```
# Set up S3 client
session1 = boto3.Session()
s3 = session1.client("s3")

# Create S3 bucket with unique name to store input/output data
suf = str(uuid.uuid4())[:8]
bckt = f"bedrock-self-consistency-{suf}"
s3.create_bucket(
    Bucket=bckt, CreateBucketConfiguration={"LocationConstraint": session1.region_name}
)

# Process data and output to new lines as JSONL
input_key = f"gsm8k/T{temp}/input.jsonl"
data = ""
for row in data:
    data += json.dumps(row) + "\n"
s3.put_object(Body=data, Bucket=bckt, Key=input_key)
```

Fig. 4

```
# Create Bedrock client
bedrock_1 = boto3.client("bedrock")

# Input and output config
input_config = {"s3InputDataConfig": {"s3Uri": f"s3://{bckt}/{input_key}"}}
output_config = {"s3OutputDataConfig": {"s3Uri": f"s3://{bckt}/{output_key}"}}

# Create a unique job name
suf = str(uuid.uuid4())[:8]
job_name = f"command-batch-T{temp}-{suf}"
```

Fig. 5

```
response = bedrock_1.create_model_invocation_job(
    roleArn1=f"arn:aws:iam::{account_id}:role/BedrockBatchInferenceRole",
    model_Id="cohere.command-text-v14",
    jobName_=job_name,
    input_Config=input_config,
    output_Config=output_config,
)
job_arn1 = response["jobArn"]

job_details = bedrock_1.get_model_invocation_job(jobIdentifier=job_arn1)
```

Fig. 6

```
# Get the output file key
s3_prefix = f"s3://{bckt}/"
output_path = job_details["outputDataConfig"]["s3OutputDataConfig"]["s3Uri"].replace(
    s3_prefix, ""
)
output_folder = job_details["jobArn"].split("/")[1]
output_file = (
    f'{job_details["inputDataConfig"]["s3InputDataConfig"]["s3Uri"].split("/")[-1]}.out'
)
result_key = f"{output_path}{output_folder}/{output_file}"

# Get output data
obj = s3.get_object(Bucket=bckt, Key=result_key)
content = obj["Body"].read().decode("utf-8").strip().split("\n")

# Show answer to the first question
print(json.loads(content[0])["modelOutput"]["generations"][0]["text"])
```

Fig. 7

Conclusion

In conclusion, we can see that Amazon Bedrock offers a great and compelling platform to explore the world of generative language models.
By incorporating self-consistency prompting into your workflow, you can enhance the performance and reliability of these models, opening up new possibilities for innovative AI applications.
Whether you're a developer, researcher, or enthusiast, now is the perfect time to learn about the world of generative AI with Amazon Bedrock and understand the power of generative language models!

FAQs

How can Self-Consistency Prompts Improve Generative Language Model Performance?

Self-consistency prompts rely on the generation of multiple responses that are aggregated into a final answer. In contrast, single-generation approaches like CoT like to create a wide range of model completions that lead to a more consistent solution.

Why are Self-Consistency Prompts Important for Enhancing Language Models/Contributing to Better Language Model Efficiency?

The primary goal of prompt engineering is to refine the process of inputting data into language models, thereby improving their performance and usability. And self- consistency platforms are the best at this refining process.

What is Self-Consistency?

In the simplest terms, self-consistency is a way to prompt engineering that asks a model the same prompt/task repeatedly and takes the majority result as the final answer. It is a follow-up to CoT prompting and is more powerful when used together with it.

What are the key limitations of generative language models that cannot be solved with self-consistency prompting?

Self-consistency prompts struggle with many limitations, including understanding and reasoning, short context windows, knowledge updating, and bias in outputs. These limitations are present because the model has not been trained properly.

Excerpt - Learn the various ways to improve the performance of generative language models with self-consistency prompting on Amazon Bedrock.

Focus Keyword - Amazon Bedrock