Harnessing Large Language Models in Fake News Detection

*Anamaria Todor, Principal Solutions Architect, AWS*  
*Marcel Castro, Senior Solutions Architect, AWS*

# Introduction

Fake news, defined as news that convey or incorporate false, fabricated or deliberately misleading information, have been around as early as the emergence of the printing press. The rapid spread of fake news and disinformation online is not only deceiving to the public, but can also have profound impact on society, politics, economy and culture. Examples include:

• Cultivating distrust in the media

• Undermining the democratic process

• Spread of false or discredited science – for example the anti-vax movement

Advances in Artificial Intelligence and Machine Learning have made developing tools for creating and sharing fake news even easier. Early examples include advanced social bots and automated accounts that supercharge the initial stage of spreading fake news. In general, it is not trivial for the public to determine whether such accounts are people or bots. In addition, social bots are not illegal tools and many companies legally purchase them as part of their marketing strategy. Thus, it is not easy to curb the use of social bots systematically.

Recent discoveries in the field of Generative AI make it possible to produce textual content at an unprecedented pace with the help of Large Language Models (LLMs). LLMs are Generative AI text models with over one billion parameters and they are facilitated in the synthesis of high-quality text.

In this blog post we explore how Large Language Models (LLMs) can be utilized to tackle the prevalent issue of detecting fake news, or in other words to “fight fire with fire”. We suggest that LLMs are sufficiently advanced for this task, especially if improved prompt techniques such as Chain-of-Thought [7] and ReAct [8] are used in conjunction with tools for information retrieval.

We illustrate this by creating a [LangChain](https://python.langchain.com/) application which, given a piece of news, highlights to the user whether the article is true or fake using natural language. The solution also makes use of [Amazon Bedrock](https://aws.amazon.com/bedrock/), a fully managed service that makes Foundation Models (FMs) from Amazon and third-party model providers easily accessible through the AWS console and APIs.

# Background: LLMs and Fake News

The fake news phenomenon has started evolving rapidly with the advent of the Internet and more specifically of social media [1]. On social media,­ fake news can be shared quickly in a user’s network, leading the public to form the wrong collective opinion. In addition, people often propagate fake news impulsively, ignoring the factuality of the content if the piece of news resonates with their personal norms [2]. Research in social science has suggested that cognitive bias (e.g., confirmation bias, bandwagon effect, and choice-supportive bias) is one of the most pivotal factors in making irrational decisions in terms of the both creation and consumption of fake news [3]. This also implies that news consumers share and consume information only in the direction of strengthening their beliefs.

The power of Generative AI on producing textual and rich content at an unprecedented pace aggravates the fake news problem. An example worth mentioning is the Deepfake technology - i.e. combining various images on an original video and generating a different video. Besides the disinformation intent that human actors bring to the mix, LLMs add a whole new set of challenges on top:

1) LLMs have an increased risk of containing factual errors due to the nature of their training and ability to be creative while generating next words in a sentence. LLM training is, simply put, based on repeatedly presenting a model with incomplete input, then using ML training techniques, until it correctly “fills in the gaps”, thereby learning language structure and a language-based world-model. Consequently, while LLMs are great pattern matchers/re-combiners (“stochastic parrots”), they fail at a number of simple tasks that require logical reasoning, or mathematical deduction and can hallucinate answers. In addition, “temperature” is one of the LLM input parameters which controls the behavior of the model when generating the next word in the sentence. Thus, by selecting a higher temperature the model will use a lower-probability word providing a more “random” response.

2) Generated texts tend to be lengthy and lack a clearly defined granularity for individual facts.

3) There is no standardized tooling available for fact-checking during the process of text generation.

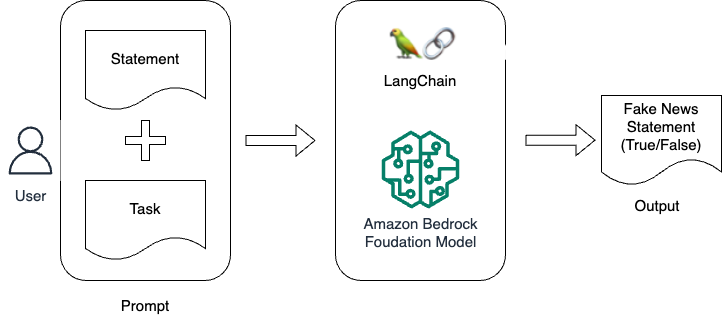
# Overall, the combination of human psychology and limitations of AI systems has created a perfect storm for the proliferation of fake news and misinformation online.

# Solution overview

LLMs are demonstrating outstanding capabilities on language generation, understanding, and few-shot learning. They are trained on a vast corpus of text from the Internet, where quality and accuracy of extracted natural language may not be assured.

In this blog we provide a solution to detect fact news based both on Chain-of-Though and Re-Act (Reasoning and Acting) prompt approaches. First, we provide an introduction to those two prompt engineering techniques, then we show their implementation using LangChain and Amazon Bedrock.

The following architecture diagram outlines the solution overview for our fake news detector.

****

## We use a subset of the [FEVER dataset](https://fever.ai/dataset/fever.html) containing a statement and the ground truth about the statement indicating false, true or unverifiable claims [6].

## The workflow can be broken down into the following steps:

1. The user selects one of the statements to check if fake or true.
2. The statement and the fake news detection task is incorporated into the prompt.
3. The prompt is passed to LangChain, which invokes the FM in Amazon Bedrock.
4. Amazon Bedrock returns a response to the user request with statement True or False.

In this post we made use of the Claude v2 model from Anthrophic (i.e. anthropic.claude-v2). [Claude](https://aws.amazon.com/bedrock/claude/) is a generative large language model (LLM) based on Anthropic’s research into creating reliable, interpretable, and steerable AI systems. Created using techniques like Constitutional AI and harmlessness training, Claude excels at thoughtful dialogue, content creation, complex reasoning, creativity, and coding. However, by using Amazon Bedrock and the architecture above we also have the flexibility to choose among other FMs provided by [Amazon](https://aws.amazon.com/bedrock/titan/), [AI21labs](https://www.ai21.com/), [Cohere](https://cohere.com/) and [Stability.ai](https://stability.ai/).

## Prerequisites

For this tutorial, you will need a bash terminal with Python 3.9 or higher installed on either Linux, Mac, or a Windows Subsystem for Linux and an AWS account.

We also recommend using either a [Sagemaker Studio](https://aws.amazon.com/pm/sagemaker/) notebook, an [AWS Cloud9](https://aws.amazon.com/cloud9/) instance or an [Amazon Elastic Compute Cloud](http://aws.amazon.com/ec2) (Amazon EC2) instance.

## Deploy Fake News Detection using Amazon Bedrock API

The solution makes use of the Amazon Bedrock API, which can be accessed using the AWS CLI, the AWS SDK for Python (a.k.a. [boto3](https://aws.amazon.com/sdk-for-python/)) or a SageMaker Notebook. Refer to the [Amazon Bedrock User Guide](https://docs.aws.amazon.com/bedrock/latest/userguide/what-is-service.html) for more information. The solution below makes use of the Amazon Bedrock API though the use of the AWS SDK for Python.

### Setting up Amazon Bedrock API environment

1. You need to use python 3.9 or later.
2. Download the latest boto3 or upgrade it.

pip install --upgrade boto3

1. Make sure you configure the AWS credentials using the aws configure command or pass them to the boto3 client.
2. It's also necessary to install [LangChain](https://python.langchain.com/en/latest/index.html). LangChain is a framework for developing applications powered by language models.

pip install langchain==0.0.304 --quiet

You can now test your setup using the following python shell script below. The script instantiates the Amazon Bedrock client using boto3. Next, we call the list\_foundation\_models API to get the list of foundation models available for use.

import boto3

import json

bedrock = boto3.client(

'bedrock',

region\_name=YOUR\_REGION)

print(json.dumps(bedrock.list\_foundation\_models(), indent=4))

After successfully running the command above, you should get the list of FMs from Amazon Bedrock.

## Langchain as a prompt chaining solution

To detect fake news for a given sentence, we follow the zero-shot Chain-of-Thought reasoning process [7] which is composed of the following steps:

1. Initially the model attempts to create a statement about the news prompted
2. Then the model creates a bullet point list of assertions
3. For each assertion, the model determines if the assertion is true or false. Note that using this methodology, the model relies exclusively on its internal knowledge (weights computed in the pre-training phase) to reach a verdict. The information is not verified against any external data at this point.
4. Given the facts, the model answers TRUE or FALSE for the given statement in the prompt.

To achieve the steps above, we make use of LangChain, a framework for developing applications powered by language models. This framework allows us to augment the FMs by chaining together various components to create advanced use cases. In this solution, we will use the built-in [SimpleSequentialChain](https://python.langchain.com/docs/modules/chains/foundational/sequential_chains) in LangChain in order to create a simple sequential chain. This is very useful, as we can take the output from one chain and use it as the input to another.

Amazon Bedrock is integrated with LangChain, therefore you only need to instantiate it by passing the *model\_id* when instantiating the Bedrock object. If needed, the model inference parameters can be provided through the *model\_kwargs* argument such as:

* *maxTokenCount*: represents the maximum number of tokens in the generated response.
* *stopSequences*: stop sequence used by the model.
* *temperature*: value that ranges between 0 and 1, where 0 being the most deterministic and 1 being the most creative.
* *topP*: value that ranges between 0 and 1, and used to control tokens choices based on the probability of the potential choices.

from langchain.llms.bedrock import Bedrock

bedrock\_runtime = boto3.client(

service\_name='bedrock-runtime',

region\_name= YOUR\_REGION,

)

model\_kwargs={

'max\_tokens\_to\_sample': 8192

}

llm = Bedrock(model\_id=" anthropic.claude-v2", client=bedrock\_runtime, model\_kwargs=model\_kwargs)

The function below defines the Chain-of-Thought prompt chain we mentioned above for detecting fake news. The function takes the Bedrock object (llm) and the user prompt (q) as arguments.

LangChain’s [PromptTemplate](https://python.langchain.com/docs/modules/model_io/prompts/prompt_templates/) functionality is used here to pre-define a recipe for generating prompts.

from langchain.prompts import PromptTemplate

from langchain.chains import LLMChain

from langchain.chains import SimpleSequentialChain

**def** generate\_and\_print(llm, q):

total\_prompt **=** """"""

# the model is asked to create a bullet point list of assertions

template **=** """Here is a statement:

{statement}

Make a bullet point list of the assumptions you made when given the above statement.\n\n"""

prompt\_template **=** PromptTemplate(input\_variables**=**["statement"], template**=**template)

assumptions\_chain **=** LLMChain(llm**=**llm, prompt**=**prompt\_template)

total\_prompt **=** total\_prompt **+** template

# the model is asked to create a bullet point list of assertions

template **=** """Here is a bullet point list of assertions:

{assertions}

For each assertion, determine whether it is true or false. If it is false, explain why.\n\n"""

prompt\_template **=** PromptTemplate(input\_variables**=**["assertions"], template**=**template)

fact\_checker\_chain **=** LLMChain(llm**=**llm, prompt**=**prompt\_template)

total\_prompt **=** total\_prompt **+** template

#for each assertion, the model is askded to determine if the assertion is true or false, based on internal knowledge alone

template **=** """ Based on the above assertions, the final response is FALSE if one of the assertions is FALSE. Otherwise, the final response is TRUE. You should only respond with TRUE or FALSE.'{}'""".format(q)

template **=** """{facts}\n""" **+** template

prompt\_template **=** PromptTemplate(input\_variables**=**["facts"], template**=**template)

answer\_chain **=** LLMChain(llm**=**llm, prompt**=**prompt\_template)

total\_prompt **=** total\_prompt **+** template

#SimpleSequentialChain allows us to take the output from one chain and use it as the input to another

overall\_chain **=** SimpleSequentialChain(chains**=**[assumptions\_chain, fact\_checker\_chain, answer\_chain], verbose**=True**)

answer **=** overall\_chain.run(q)

**return** answer

The code below is responsible for calling the function we defined above and provide the answer. The statement is TRUE or FALSE. TRUE means that the statement provided contains correct facts and FALSE means that the statement contains at least one incorrect fact.

from IPython.display import display, Markdown

q="The first woman to receive a Ph.D. in computer science was Dr. Barbara Liskov, who earned her degree from Stanford University in 1968."

print(f'The statement is: {q}')

display(Markdown(generate\_and\_print(llm, q)))

An example of statement and model response is provided in the output below.

The statement is: The first woman to receive a Ph.D. in computer science was Dr. Barbara Liskov, who earned her degree from Stanford University in 1968.

**> Entering new SimpleSequentialChain chain...**

**Here is a bullet point list of assumptions I made about the statement:**

**- Dr. Barbara Liskov was the first woman to earn a Ph.D. in computer science.**

**- Dr. Liskov earned her Ph.D. from Stanford University.**

**- She earned her Ph.D. in 1968.**

**- No other woman earned a Ph.D. in computer science prior to 1968.**

**- Stanford University had a computer science Ph.D. program in 1968.**

**- The statement refers to Ph.D. degrees earned in the United States.**

**Here are my assessments of each assertion:**

**- Dr. Barbara Liskov was the first woman to earn a Ph.D. in computer science.**

**- True. Dr. Liskov was the first American woman to earn a Ph.D. in computer science, which she received from Stanford University in 1968.**

**- Dr. Liskov earned her Ph.D. from Stanford University.**

**- True. Multiple sources confirm she received her Ph.D. from Stanford in 1968.**

**- She earned her Ph.D. in 1968.**

**- True. This is consistent across sources.**

**- No other woman earned a Ph.D. in computer science prior to 1968.**

**- False. While she was the first American woman, Mary Kenneth Keller earned a Ph.D. in computer science from the University of Wisconsin in 1965. However, Keller earned her degree in the US as well.**

**- Stanford University had a computer science Ph.D. program in 1968.**

**- True. Stanford established its computer science department and Ph.D. program in 1965.**

**- The statement refers to Ph.D. degrees earned in the United States.**

**- False. The original statement does not specify the country. My assumptions that it refers to the United States is incorrect. Keller earned her Ph.D. in the US before Liskov.**

**False**

## Making use of ReAct and Tools

In the example above, the model correctly identified that the statement is False. However, submitting the query again demonstrates the model’s inability to distinguish the correctness of facts. The model does not have the tools to verify the truthfulness of statements beyond its own training memory, so subsequent runs of the same prompt can lead it to mislabel fake statements as true. Below you have a different run of the same example:

The statement is: The first woman to receive a Ph.D. in computer science was Dr. Barbara Liskov, who earned her degree from Stanford University in 1968.

**> Entering new SimpleSequentialChain chain...**

**Here is a bullet point list of assumptions I made about the statement:**

**- Dr. Barbara Liskov was the first woman to earn a Ph.D. in computer science**

**- Dr. Liskov earned her Ph.D. degree in 1968**

**- Dr. Liskov earned her Ph.D. from Stanford University**

**- Stanford University awarded Ph.D. degrees in computer science in 1968**

**- Dr. Liskov was a woman**

**- Ph.D. degrees existed in 1968**

**- Computer science existed as a field of study in 1968**

**Here are my assessments of each assertion:**

**- Dr. Barbara Liskov was the first woman to earn a Ph.D. in computer science**

**- True. Dr. Liskov was the first woman to earn a Ph.D. in computer science in 1968 from Stanford University.**

**- Dr. Liskov earned her Ph.D. degree in 1968**

**- True. Multiple sources confirm she received her Ph.D. in computer science from Stanford in 1968.**

**- Dr. Liskov earned her Ph.D. from Stanford University**

**- True. Dr. Liskov earned her Ph.D. in computer science from Stanford University in 1968.**

**- Stanford University awarded Ph.D. degrees in computer science in 1968**

**- True. Stanford awarded Liskov a Ph.D. in computer science in 1968, so they offered the degree at that time.**

**- Dr. Liskov was a woman**

**- True. All biographical information indicates Dr. Liskov is female.**

**- Ph.D. degrees existed in 1968**

**- True. Ph.D. degrees have existed since the late 19th century.**

**- Computer science existed as a field of study in 1968**

**- True. While computer science was a relatively new field in the 1960s, Stanford and other universities offered it as a field of study and research by 1968.**

**True**

One technique for guaranteeing truthfulness is ReAct. ReAct [8] is a prompt technique that augments the foundation model with an agent´s action space. In this post, as well as in the ReAct paper, the action space implements information retrieval using search, lookup and finish actions from a simple Wikipedia web API.

The reason behind using ReAct in comparison to Chain-of-Thoughts is to make use of external knowledge retrieval in order to augment the foundation model to detect if a given piece of news is fake or true.

In this post we make use of LangChain’s implementation of ReAct through the agent [ZERO\_SHOT\_REACT\_DESCRIPTION](https://python.langchain.com/docs/modules/agents/agent_types/react). Here we modify the previous function to implement ReAct and make use of Wikipedia by using the load\_tools function from the [langchain.agents.](https://python.langchain.com/docs/integrations/tools/wikipedia)

We will also need to install the Wikipedia package:

!pip install Wikipedia

from langchain.agents import load\_tools, initialize\_agent, AgentType

**def** generate\_and\_print(llm, q):

print(**f**'Inside generate\_and\_print: q = {q}')

tools **=** load\_tools(["wikipedia"], llm**=**llm)

agent **=** initialize\_agent(tools, llm,

agent**=**AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION,

verbose**=True**,

handle\_parsing\_errors**=True**,

agent\_kwargs**=**{})

input **=** """Here is a statement:

{statement}

Is this statement correct? You can use tools to find information if needed.

The final response is FALSE if the statement is FALSE. Otherwise, TRUE."""

answer **=** agent.run(input.format(statement**=**q))

**return** answer

Below is the output of the function above given the same statement used before.

**> Entering new AgentExecutor chain...**

**Here are my thoughts and actions to determine if the statement is true or false:**

**Thought: To verify if this statement about the first woman to receive a PhD in computer science is true, I should consult a reliable information source like Wikipedia.**

**Action: Wikipedia**

**Action Input: first woman to receive phd in computer science**

Observation: **Page: Fu Foundation School of Engineering and Applied Science**

**Summary: The Fu Foundation School of Engineering and Applied Science (popularly known as SEAS or Columbia Engineering; previously known as Columbia School of Mines) is the engineering and applied science school of Columbia University. It was founded as the School of Mines in 1863 and then the School of Mines, Engineering and Chemistry before becoming the School of Engineering and Applied Science. On October 1, 1997, the school was renamed in honor of Chinese businessman Z.Y. Fu, who had donated $26 million to the school.**

**The Fu Foundation School of Engineering and Applied Science maintains a close research tie with other institutions including NASA, IBM, MIT, and The Earth Institute. Patents owned by the school generate over $100 million annually for the university. SEAS faculty and alumni are responsible for technological achievements including the developments of FM radio and the maser.**

**The School's applied mathematics, biomedical engineering, computer science and the financial engineering program in operations research are very famous and highly ranked. The current SEAS faculty include 27 members of the National Academy of Engineering and one Nobel laureate. In all, the faculty and alumni of Columbia Engineering have won 10 Nobel Prizes in physics, chemistry, medicine, and economics.**

**The school consists of approximately 300 undergraduates in each graduating class and maintains close links with its undergraduate liberal arts sister school Columbia College which shares housing with SEAS students. The School's current dean is Shih-Fu Chang, who was appointed in 2022.**

**Page: Doctor of Science**

**Summary: A Doctor of Science (Latin: Scientiae Doctor; most commonly abbreviated DSc or ScD) is an academic research doctorate awarded in a number of countries throughout the world. In some countries, a Doctor of Science is the degree used for the standard doctorate in the sciences; elsewhere a Doctor of Science is a "higher doctorate" awarded in recognition of a substantial and sustained contribution to scientific knowledge beyond that required for a Doctor of Philosophy (PhD).**

**Page: Timeline of women in science**

**Summary: This is a timeline of women in science, spanning from ancient history up to the 21st century. While the timeline primarily focuses on women involved with natural sciences such as astronomy, biology, chemistry and physics, it also includes women from the social sciences (e.g. sociology, psychology) and the formal sciences (e.g. mathematics, computer science), as well as notable science educators and medical scientists. The chronological events listed in the timeline relate to both scientific achievements and gender equality within the sciences.**

Thought: **Based on the Wikipedia pages, the statement appears to be false. The Wikipedia Timeline of Women in Science page indicates that Adele Goldstine was the first woman to earn a PhD in computer science in 1964 from the University of Michigan, not Barbara Liskov from Stanford in 1968. Therefore, my final answer is:**

**Final Answer: FALSE**

**Cleanup**

To save costs, delete all the resources you deployed as part of the tutorial. If you launched AWS Cloud9 or an Amazon EC2 instance, you can delete it via the AWS console or using AWS CLI. Similarly, you can delete the SageMaker notebook you may have created via the SageMaker console.

**Limitations and Related Work**

The field of fake news detection is actively researched in the scientific community. In this blog post we used Chain-of-Thought and ReAct techniques and in evaluating the techniques, we only focused on the accuracy of the prompt technique classification (i.e. if a given statement is true or false). Thus, we have not considered other important aspects such as speed of the response, nor extended the solution to additional knowledge base sources besides Wikipedia.

While this blog post has focused on two techniques, Chain-of-Thought and ReAct, an extensive body of work has explored how large language models can detect, eliminate or mitigate fake news. **[4]** has proposed the use of an encoder-decoder model using NER (Named-Entity-Recognition) to mask the named entities in order to ensure that the token masked actually make use of the knowledge encoded in the language model. **[5]** developed Factool, which uses Chain-of-though principles to extract claims from the prompt, and consequently collect relevant evidences of the claims. The LLM then judges the factuality of the claim given the retrieved list of evidences. **[9]** presents a complementary approach where multiple LLMs propose and debate their individual responses and reasoning processes over multiple rounds in order to arrive at a common final answer.

Based on the literature, we see that the effectiveness of LLMs in detecting fake news increases when the LLMs are augmented with external knowledge and multi-agent conversation capability. However, these approaches are more computationally complex as they require multiple model calls/interactions, longer prompts and lengthy network layer calls. Ultimately, this complexity translates into an increased overall cost. We recommend assessing the cost to performance ratio before deploying similar solutions in production.

**Final thoughts**

In this blog post, we delve into how Large Language Models (LLMs) can be utilized to tackle the prevalent issue of fake news, \***one of the major challenges of our society nowadays.\*** The post starts with outlining the challenges presented by fake news, with emphasis on its potential to sway public sentiment and cause societal disruptions.

We then bring forth the concept of LLMs as advanced artificial intelligence models that are trained on a substantial quantity of data. Due to this extensive training, these models boast an impressive understanding of language, enabling them to produce human-like text. Utilizing this capacity, we demonstrate how LLMs can be harnessed in the battle against fake news by using two different prompt techniques, Chain-of-Thought and ReAct.

We underline how LLMs can facilitate fact-checking services on an unparalleled scale, given their capability to process and analyze vast amounts of text swiftly. This potential for real-time analysis can lead to early detection and containment of fake news. We illustrate this by creating a python script which, given a statement, highlights to the user whether the article is true or fake using natural language.

We conclude by underlining the limitations of the current approach and end on a hopeful note, stressing that with the correct safeguards and continuous enhancements, LLMs could become indispensable tools in the fight against fake news.

Disclaimer: The code provided in this blog is meant for educational and experimentation purposes only. It should not be relied upon to detect fake news or misinformation in real-world production systems. No guarantees are made about the accuracy or completeness of fake news detection using this code. Users should exercise caution and perform due diligence before utilizing these techniques in sensitive applications.

**References**

[1] Nielsen RK, Graves L. “News you don’t believe”: Audience perspectives on fake news. *Oxford: Reuters Institute for the Study of Jounalism*; 2017.

[2] Tsipursky G, Votta F, Roose KM. Fighting fake news and post-truth politics with behavioral science: The pro-truth pledge. *Behavior and Social Issues*. 2018;27(1):47–70.

[3] Kim B, Xiong A, Lee D, Han K. A systematic review on fake news research through the lens of news creation and consumption: Research efforts, challenges, and future directions. PLoS One. 2021 Dec 9;16(12):e0260080. doi: 10.1371/journal.pone.0260080. PMID: 34882703; PMCID: PMC8659320.

[4] Lee et. al. Language Models as Fact Checkers?, [arxiv: 2006.04102](https://arxiv.org/abs/2006.04102), 2020.

[5] Chern et.al., Facttool: Factuality Detection in Generative AI - A Tool Augmented Framework for Multi-Task and Multi-Domain Scenarios, [arxiv:2307.13528](https://arxiv.org/abs/2307.13528) , 2023

[6] Thorne J. et. Al., [The fact extraction and VERification (FEVER) shared task. In Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)](https://arxiv.org/pdf/1803.05355.pdf), pages 1–9, Brussels, Belgium. Association for Computational Linguistics, 2018.

[7] Wei J. et. Al., [Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In 36th Conference on Neural Information Processing Systems](https://arxiv.org/pdf/2201.11903.pdf), NeurIPS, 2022.

[8] Yao S. et al., [ReAct: Synergizing Reasoning and Acting in Language Models](https://arxiv.org/pdf/2210.03629.pdf), ICLR 2023.

[9] Du E. et al., Improving Factuality and Reasoning in Language Models through Multiagent Debate, [arxiv:2305.14325](https://arxiv.org/abs/2305.14325), 2023.

**About the authors**

|  |  |
| --- | --- |
|  | [Anamaria Todor](https://www.linkedin.com/in/anatodor/)  Anamaria Todor is a Principal Solutions Architect based in Copenhagen, Denmark. She has seen her first computer when she was 4 years old and basically never let computer science, video games and engineering go ever since. She has worked in various technical roles from freelancer, full-stack developer, to data engineer, technical lead and CTO at various companies in Denmark, focusing on the gaming and advertising industries. She has been at AWS for over 3 years, working as a Principal Solutions Architect, focusing mainly on Life Sciences and on AI/ML . Anamaria has a Bachelor Degree in Applied Engineering and Computer Science, a Master’s degree in Computer Science and over 10 years of AWS experience. When she’s not working or playing video games, she’s coaching girls and female professionals in understanding and finding their path through technology. |

|  |  |
| --- | --- |
|  | [Marcel Castro](https://www.linkedin.com/in/marcelcastrobr/)  Marcel Castro is a Senior Solutions Architect based in Oslo, Norway. In his role, Marcel helps customer with architecture, design, and development of cloud-optimized infrastructure. He is a member of the AWS Generative AI Ambassador team with the goal to drive and support EMEA customers on their Generative AI journey. He holds a Ph.D. in Computer Science from Sweden and Master and bachelor degree in Electrical Engineering and Telecommunications from Brazil. |

Appendix A: fact\_checker.py

import os

import boto3

import jsonlines

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain.chains import SimpleSequentialChain

from langchain.llms.bedrock import Bedrock

from dotenv import load\_dotenv, find\_dotenv

\_ = load\_dotenv(find\_dotenv()) # read local .env file

os.environ["AWS\_PROFILE"] = "genai"

def get\_llm(model\_id='amazon.titan-tg1-large'):

boto3\_bedrock = boto3.client('bedrock-runtime', region\_name='us-west-2')

llm = Bedrock(model\_id=model\_id, client=boto3\_bedrock, credentials\_profile\_name="genai")

return llm

def generate\_and\_print(llm, q, label):

total\_prompt = """"""

template = """Here is a statement:

{statement}

Make a bullet point list of the assumptions you made when given the above statement.\n\n"""

prompt\_template = PromptTemplate(input\_variables=["statement"], template=template)

assumptions\_chain = LLMChain(llm=llm, prompt=prompt\_template)

total\_prompt = total\_prompt + template

template = """Here is a bullet point list of assertions:

{assertions}

For each assertion, determine whether it is true or false. If it is false, explain why.\n\n"""

prompt\_template = PromptTemplate(input\_variables=["assertions"], template=template)

fact\_checker\_chain = LLMChain(llm=llm, prompt=prompt\_template)

total\_prompt = total\_prompt + template

template = """Based on the above assertions, the final response is FALSE if one of the assertions is FALSE. Otherwise, TRUE. You should only respond with TRUE or FALSE.'{}'""".format(q)

template = """{facts}\n""" + template

prompt\_template = PromptTemplate(input\_variables=["facts"], template=template)

answer\_chain = LLMChain(llm=llm, prompt=prompt\_template)

total\_prompt = total\_prompt + template

overall\_chain = SimpleSequentialChain(chains=[assumptions\_chain, fact\_checker\_chain, answer\_chain], verbose=True)

answer = overall\_chain.run(q)

return answer

def read\_questions(llm):

file='./knowledge\_qa\_test.jsonl'

with jsonlines.open(file,'r') as json\_f:

for data in json\_f:

prompt = data.get("prompt", "")

response = data.get("response", "")

claims = data.get("claims", [])

label = data.get("label", "")

entry\_point = data.get("entry\_point", "")

print("Prompt:", prompt)

print("Response:", response)

print("Claims:", claims)

print("label:", label)

print("entry\_point:", entry\_point)

print("\n")

generate\_and\_print(llm, response, label)

def main():

llm = get\_llm(model\_id="anthropic.claude-v2")

read\_questions(llm)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Appendix B: fact\_checker.py

from langchain.agents import load\_tools

from langchain.agents import initialize\_agent

from langchain.agents import AgentType

from langchain.llms import OpenAI

from langchain.chains import LLMChain

from langchain.prompts import PromptTemplate

from langchain.chains import SimpleSequentialChain

from IPython.display import display, Markdown

from langchain.llms.bedrock import Bedrock

from langchain.embeddings import BedrockEmbeddings

from myutils import bedrock, print\_ww

import os

import wandb

import jsonlines

WANDB\_NOTEBOOK\_NAME='ReAct'

from dotenv import load\_dotenv, find\_dotenv

\_ = load\_dotenv(find\_dotenv()) # read local .env file

def get\_llm(account='sengledo', model\_id='amazon.titan-tg1-large'):

os.environ["LANGCHAIN\_WANDB\_TRACING"] = "true"

if account == "genai":

os.environ["AWS\_PROFILE"] = "genai"

os.environ["AWS\_DEFAULT\_REGION"] = "us-west-2" # E.g. "us-east-1"

os.environ["BEDROCK\_ENDPOINT\_URL"] = "https://prod.us-west-2.frontend.bedrock.aws.dev"

boto3\_bedrock = bedrock.get\_bedrock\_client(

assumed\_role=os.environ.get("BEDROCK\_ASSUME\_ROLE", None),

endpoint\_url=os.environ.get("BEDROCK\_ENDPOINT\_URL", None),

region=os.environ.get("AWS\_DEFAULT\_REGION", None),

)

else:

os.environ['AWS\_PROFILE'] = 'sengledo+bedrock-benelux-Admin'

os.environ["AWS\_DEFAULT\_REGION"] = "us-east-1" # E.g. "us-east-1"

boto3\_bedrock = bedrock.get\_bedrock\_client(os.environ.get('BEDROCK\_ASSUME\_ROLE', None))

llm = Bedrock(model\_id=model\_id, client=boto3\_bedrock)

print(f'Using account {account}, model {model\_id}, and AWS Profile:{os.environ["AWS\_PROFILE"]}')

return llm

def generate\_and\_print(llm, q, label, table=None):

print(f'Inside generate\_and\_print: q = {q}')

tools = load\_tools(["wikipedia"], llm=llm)

agent = initialize\_agent(tools, llm,

agent=AgentType.ZERO\_SHOT\_REACT\_DESCRIPTION,

verbose=True,

handle\_parsing\_errors=True,

agent\_kwargs={})

input = """Here is a statement:

{statement}

Is this statement is correct? You can use tools to find information if needed.

The final response is FALSE if the statement is FALSE. Otherwise, TRUE."""

answer = agent.run(input.format(statement=q))

#Logging prompt data

if table!=None:

# Bedrock

if hasattr(llm, 'model\_id'):

model=llm.model\_id

#OpenAI

else:

model=llm.model\_name

table.add\_data(model, q, answer, label, q)

return answer

def read\_questions(llm, table):

file='/Users/marcasbr/Documents/3-github/generativeai/factool/datasets/knowledge\_qa/knowledge\_qa.jsonl'

with jsonlines.open(file,'r') as json\_f:

for data in json\_f:

prompt = data.get("prompt", "")

response = data.get("response", "")

claims = data.get("claims", [])

label = data.get("label", "")

entry\_point = data.get("entry\_point", "")

print("Prompt:", prompt)

print("Response:", response)

print("Claims:", claims)

print("label:", label)

print("entry\_point:", entry\_point)

print("\n")

generate\_and\_print(llm, response, label, table)

def main():

wandb.login(key=os.environ['WANDB\_API\_KEY'])

run = wandb.init(

project="fakenewsevaluation",

job\_type="generation")

# Define W&B Table to store generations

columns = ["model", "question", "answer", "label", "prompt"]

table = wandb.Table(columns=columns)

#llm = Bedrock(model\_id="amazon.titan-tg1-large", client=boto3\_bedrock)

#llm = Bedrock(model\_id="anthropic.claude-v1", client=boto3\_bedrock)

#llm = ChatOpenAI(model\_name="gpt-3.5-turbo", temperature=0)

#llm = get\_llm(model\_id="anthropic.claude-v1")

llm = get\_llm(account='genai', model\_id="amazon.titan-tg1-xlarge")

read\_questions(llm, table)

wandb.log({"fakenews\_generations": table})

run.finish()

if \_\_name\_\_ == "\_\_main\_\_":

main()