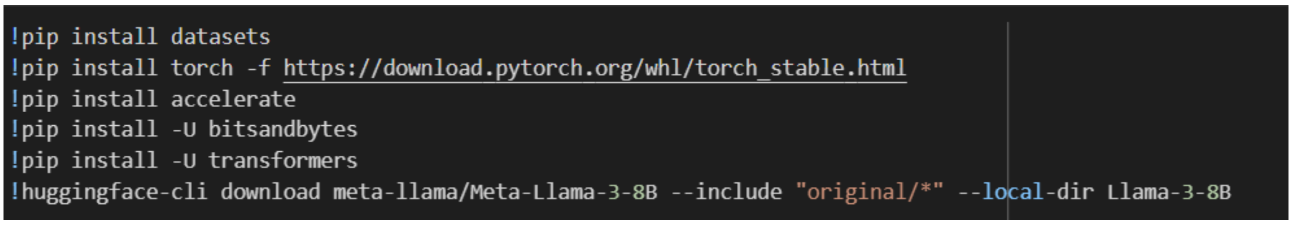
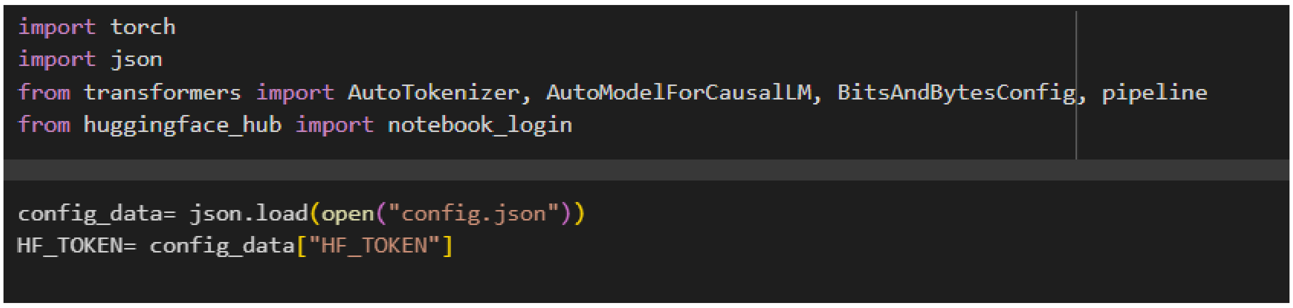
Report on HW3 (Student: Fattori Giorgia 2023138087)

Model choice: LLAMA 3-8B

I chose to use LLAMA 3 because it is a stable, quite recent, model for language generation powered by Meta(ex.Facebook). Specifically, I chose the 8B version because it is big enough to have resources for more precise and context relevant text generation but is not big enough to make its execution too lengthy (or crash during execution). LLAMA3.8B features pretrained and instruction-fine-tuned models. Following the documentation LLAMA 3 has also improved reasoning abilities compared to its predecessors LLAMA and LLAMA2. Furthermore, LLAMA 3 is an open-source model obtainable for free with a token key on the HUGGINGFACE website.

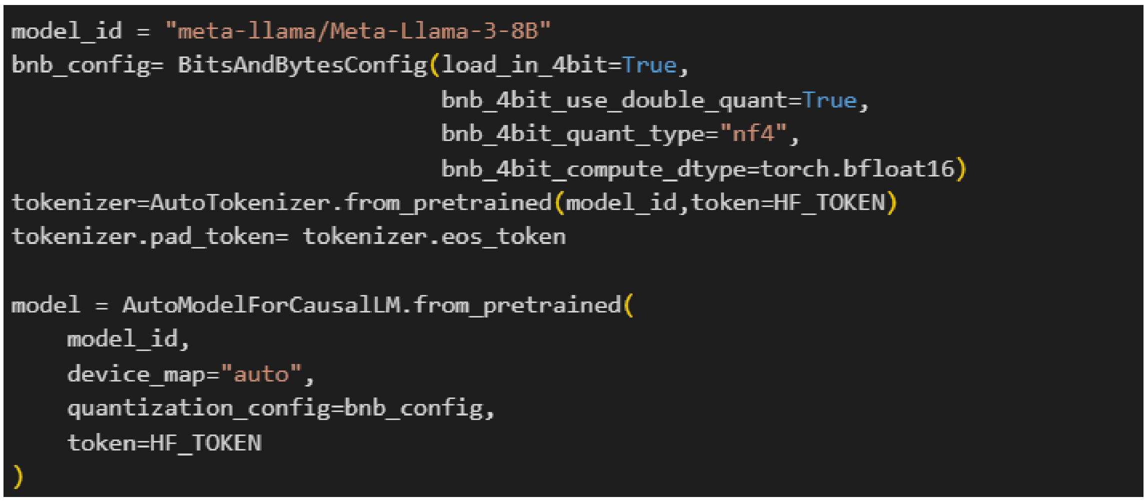


```
!pip install datasets
!pip install torch -f https://download.pytorch.org/whl/torch_stable.html
!pip install accelerate
!pip install -U bitsandbytes
!pip install -U transformers
!huggingface-cli download meta-llama/Meta-Llama-3-8B --include "original/*" --local-dir Llama-3-8B
```



```
import torch
import json
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig, pipeline
from huggingface_hub import notebook_login
```

```
config_data= json.load(open("config.json"))
HF_TOKEN= config_data["HF_TOKEN"]
```

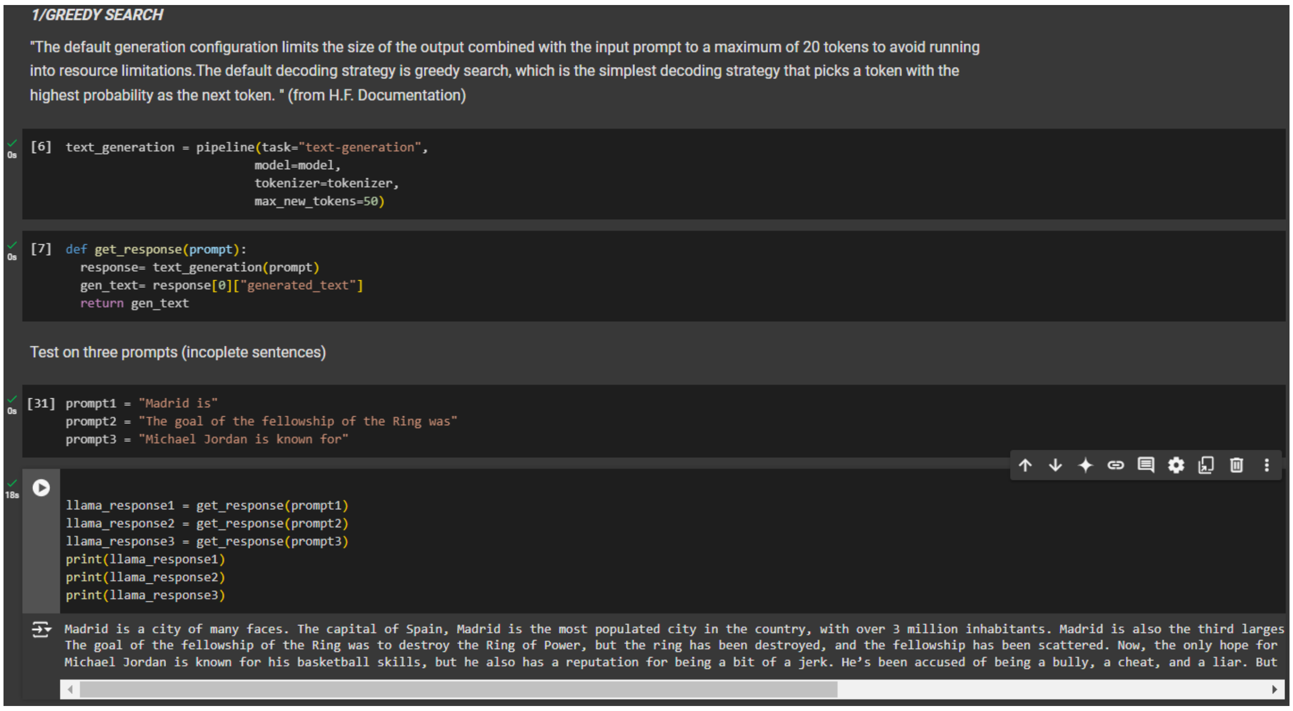


```
model_id = "meta-llama/Meta-Llama-3-8B"
bnb_config= BitsAndBytesConfig(load_in_4bit=True,
                               bnb_4bit_use_double_quant=True,
                               bnb_4bit_quant_type="nf4",
                               bnb_4bit_compute_dtype=torch.bfloat16)
tokenizer=AutoTokenizer.from_pretrained(model_id,token=HF_TOKEN)
tokenizer.pad_token= tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    quantization_config=bnb_config,
    token=HF_TOKEN
)
```

Decoding strategies:

1/Greedy search

Greedy search is the default method used for text generation by LLAMA3, even though greedy search uses a greedy algorithm (optimal search) it' not always true that its result is optimal, since sometimes it hides probabilities with high scores that can be found in posterior tokens and were "discarded" due to a low probability score of a previous token.



1/GREEDY SEARCH

"The default generation configuration limits the size of the output combined with the input prompt to a maximum of 20 tokens to avoid running into resource limitations.The default decoding strategy is greedy search, which is the simplest decoding strategy that picks a token with the highest probability as the next token. " (from H.F. Documentation)

```
[6] text_generation = pipeline(task="text-generation",
                               model=model,
                               tokenizer=tokenizer,
                               max_new_tokens=50)
```

```
[7] def get_response(prompt):
      response= text_generation(prompt)
      gen_text= response[0]["generated_text"]
      return gen_text
```

Test on three prompts (incoplete sentences)

```
[31] prompt1 = "Madrid is"
     prompt2 = "The goal of the fellowship of the Ring was"
     prompt3 = "Michael Jordan is known for"
```

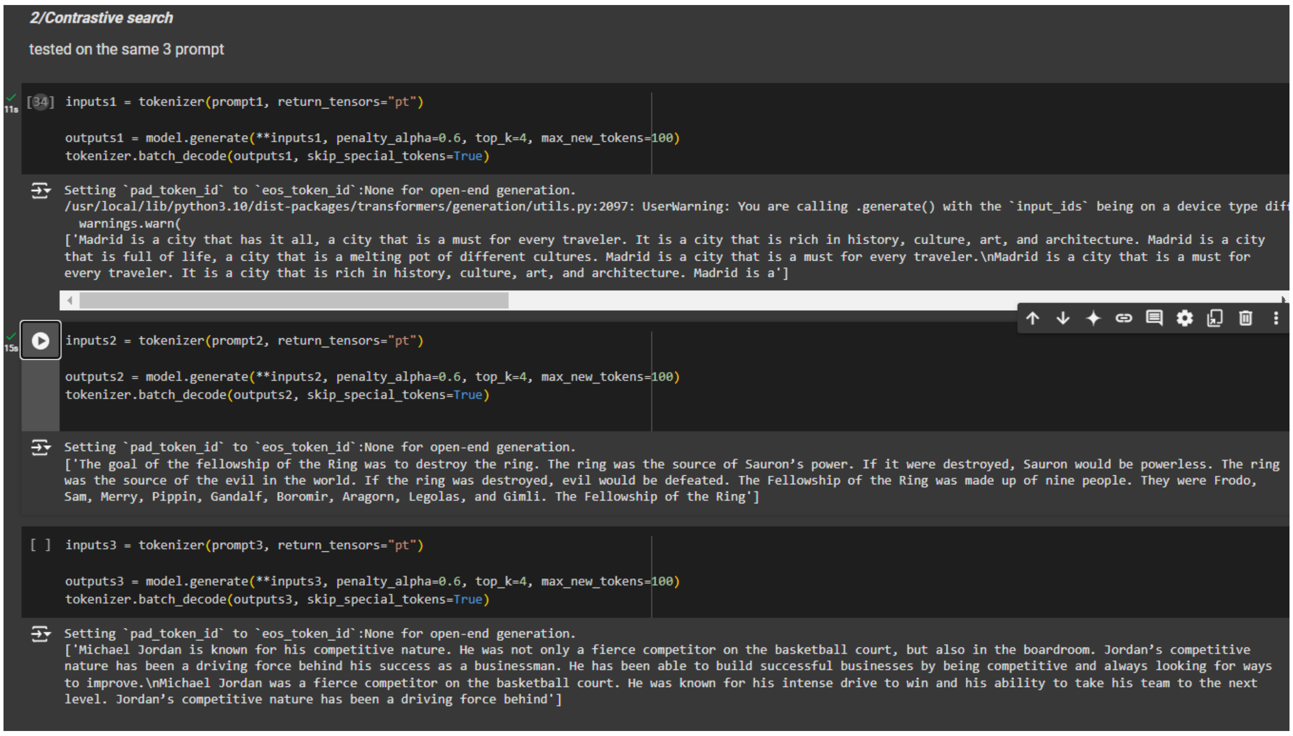
```
llama_response1 = get_response(prompt1)
llama_response2 = get_response(prompt2)
llama_response3 = get_response(prompt3)
print(llama_response1)
print(llama_response2)
print(llama_response3)
```

```
Madrid is a city of many faces. The capital of Spain, Madrid is the most populated city in the country, with over 3 million inhabitants. Madrid is also the third larges
The goal of the fellowship of the Ring was to destroy the Ring of Power, but the ring has been destroyed, and the fellowship has been scattered. Now, the only hope for
Michael Jordan is known for his basketball skills, but he also has a reputation for being a bit of a jerk. He's been accused of being a bully, a cheat, and a liar. But
```

2/Contrastive search

Contrastive search demonstrates superior results for generating non-repetitive yet coherent long outputs.

The two main parameters that enable and control the behavior of contrastive search
are penalty_alpha and top_k



2/Contrastive search

tested on the same 3 prompt

```
[34] inputs1 = tokenizer(prompt1, return_tensors="pt")

     outputs1 = model.generate(**inputs1, penalty_alpha=0.6, top_k=4, max_new_tokens=100)
     tokenizer.batch_decode(outputs1, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:2097: UserWarning: You are calling .generate() with the `input_ids` being on a device type dif
  warnings.warn(
['Madrid is a city that has it all, a city that is a must for every traveler. It is a city that is rich in history, culture, art, and architecture. Madrid is a city
that is full of life, a city that is a melting pot of different cultures. Madrid is a city that is a must for every traveler.\nMadrid is a city that is a must for
every traveler. It is a city that is rich in history, culture, art, and architecture. Madrid is a']
```

```
inputs2 = tokenizer(prompt2, return_tensors="pt")

outputs2 = model.generate(**inputs2, penalty_alpha=0.6, top_k=4, max_new_tokens=100)
tokenizer.batch_decode(outputs2, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
['The goal of the fellowship of the Ring was to destroy the ring. The ring was the source of Sauron's power. If it were destroyed, Sauron would be powerless. The ring
was the source of the evil in the world. If the ring was destroyed, evil would be defeated. The Fellowship of the Ring was made up of nine people. They were Frodo,
Sam, Merry, Pippin, Gandalf, Boromir, Aragorn, Legolas, and Gimli. The Fellowship of the Ring']
```

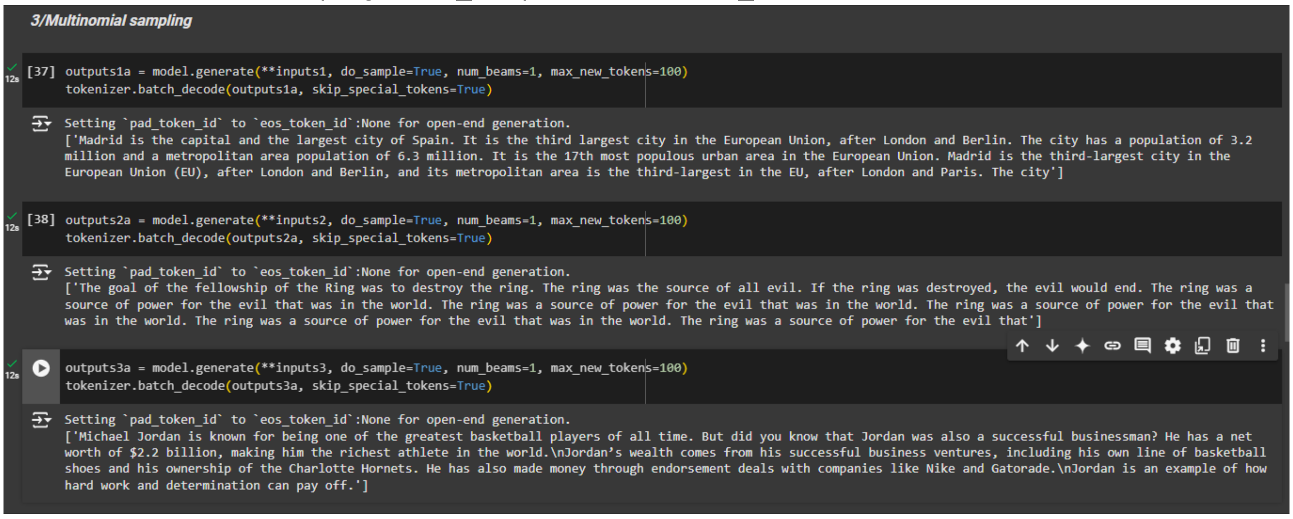
```
[ ] inputs3 = tokenizer(prompt3, return_tensors="pt")

    outputs3 = model.generate(**inputs3, penalty_alpha=0.6, top_k=4, max_new_tokens=100)
    tokenizer.batch_decode(outputs3, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
['Michael Jordan is known for his competitive nature. He was not only a fierce competitor on the basketball court, but also in the boardroom. Jordan's competitive
nature has been a driving force behind his success as a businessman. He has been able to build successful businesses by being competitive and always looking for ways
to improve.\nMichael Jordan was a fierce competitor on the basketball court. He was known for his intense drive to win and his ability to take his team to the next
level. Jordan's competitive nature has been a driving force behind']
```

3/Multinomial sampling

Unlike greedy search, which picks the token with the highest probability, multinomial sampling chooses the next token randomly from the probability distribution across the entire vocabulary provided by the model. Each token that has a probability greater than zero can be chosen, which lowers the likelihood of repeating.

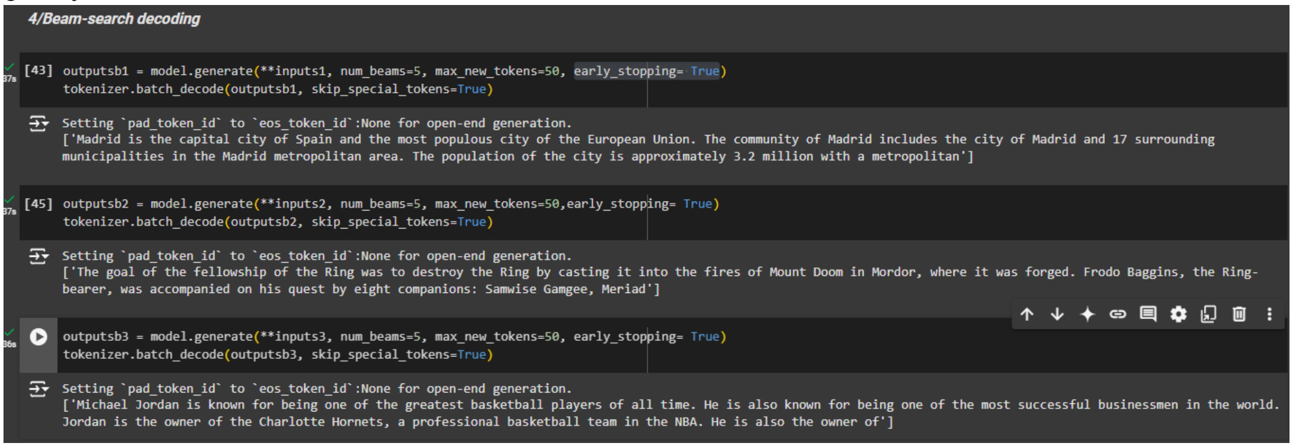
To enable multinomial sampling set do_sample=True and num_beams=1.


3/Multinomial sampling

```
[37] outputs1a = model.generate(**inputs1, do_sample=True, num_beams=1, max_new_tokens=100)
     tokenizer.batch_decode(outputs1a, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['Madrid is the capital and the largest city of Spain. It is the third largest city in the European Union, after London and Berlin. The city has a population of 3.2
     million and a metropolitan area population of 6.3 million. It is the 17th most populous urban area in the European Union. Madrid is the third-largest city in the
     European Union (EU), after London and Berlin, and its metropolitan area is the third-largest in the EU, after London and Paris. The city']

[38] outputs2a = model.generate(**inputs2, do_sample=True, num_beams=1, max_new_tokens=100)
     tokenizer.batch_decode(outputs2a, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['The goal of the fellowship of the Ring was to destroy the ring. The ring was the source of all evil. If the ring was destroyed, the evil would end. The ring was a
     source of power for the evil that was in the world. The ring was a source of power for the evil that was in the world. The ring was a source of power for the evil that
     was in the world. The ring was a source of power for the evil that was in the world. The ring was a source of power for the evil that']

     outputs3a = model.generate(**inputs3, do_sample=True, num_beams=1, max_new_tokens=100)
     tokenizer.batch_decode(outputs3a, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['Michael Jordan is known for being one of the greatest basketball players of all time. But did you know that Jordan was also a successful businessman? He has a net
     worth of $2.2 billion, making him the richest athlete in the world.\nJordan's wealth comes from his successful business ventures, including his own line of basketball
     shoes and his ownership of the Charlotte Hornets. He has also made money through endorsement deals with companies like Nike and Gatorade.\nJordan is an example of how
     hard work and determination can pay off.']
```

4/Beam-search decoding

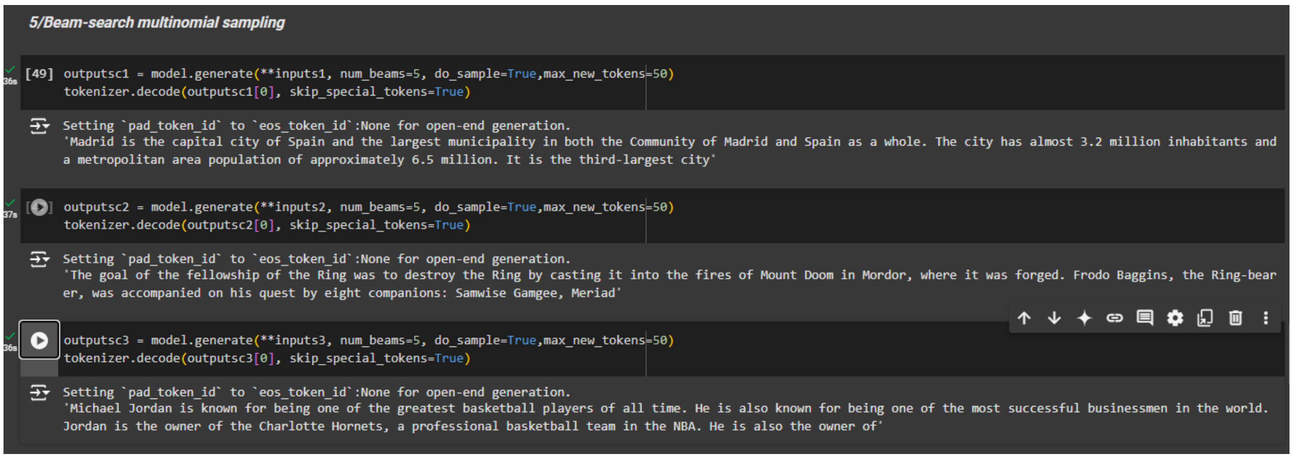
Beam-search decoding, in contrast to greedy search, maintains multiple hypotheses at every time step and ultimately selects the hypothesis with the highest overall probability for the complete sequence. This benefits from recognizing sequences with high probability that begin with lower probability starting tokens, which the greedy search method would overlook.


4/Beam-search decoding

```
[43] outputsb1 = model.generate(**inputs1, num_beams=5, max_new_tokens=50, early_stopping= True)
     tokenizer.batch_decode(outputsb1, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['Madrid is the capital city of Spain and the most populous city of the European Union. The community of Madrid includes the city of Madrid and 17 surrounding
     municipalities in the Madrid metropolitan area. The population of the city is approximately 3.2 million with a metropolitan']

[45] outputsb2 = model.generate(**inputs2, num_beams=5, max_new_tokens=50,early_stopping= True)
     tokenizer.batch_decode(outputsb2, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['The goal of the fellowship of the Ring was to destroy the Ring by casting it into the fires of Mount Doom in Mordor, where it was forged. Frodo Baggins, the Ring-
     bearer, was accompanied on his quest by eight companions: Samwise Gamgee, Meriad']

     outputsb3 = model.generate(**inputs3, num_beams=5, max_new_tokens=50, early_stopping= True)
     tokenizer.batch_decode(outputsb3, skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     ['Michael Jordan is known for being one of the greatest basketball players of all time. He is also known for being one of the most successful businessmen in the world.
     Jordan is the owner of the Charlotte Hornets, a professional basketball team in the NBA. He is also the owner of']
```

5/Beam-search multinomial sampling

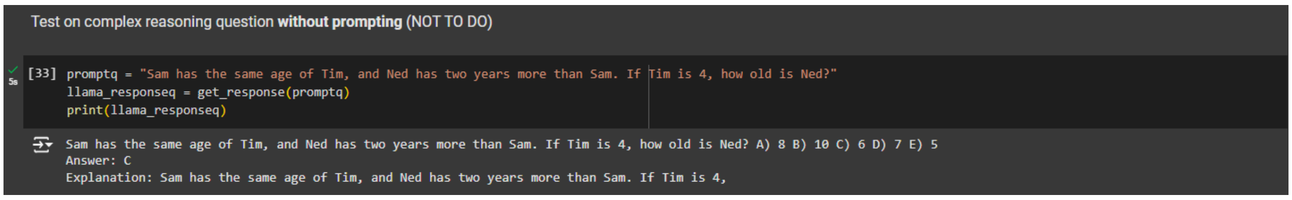
This strategy, as its name suggests, merges beam search with multinomial sampling. To utilize this decoding strategy, make sure to set num_beams greater than 1 and enable do_sample=True.{I setted a max_new_token of 50 for practical fast generation}


5/Beam-search multinomial sampling

```
[49] outputsc1 = model.generate(**inputs1, num_beams=5, do_sample=True,max_new_tokens=50)
     tokenizer.decode(outputsc1[0], skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     'Madrid is the capital city of Spain and the largest municipality in both the Community of Madrid and Spain as a whole. The city has almost 3.2 million inhabitants and
     a metropolitan area population of approximately 6.5 million. It is the third-largest city'

     outputsc2 = model.generate(**inputs2, num_beams=5, do_sample=True,max_new_tokens=50)
     tokenizer.decode(outputsc2[0], skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     'The goal of the fellowship of the Ring was to destroy the Ring by casting it into the fires of Mount Doom in Mordor, where it was forged. Frodo Baggins, the Ring-bear
     er, was accompanied on his quest by eight companions: Samwise Gamgee, Meriad'

     outputsc3 = model.generate(**inputs3, num_beams=5, do_sample=True,max_new_tokens=50)
     tokenizer.decode(outputsc3[0], skip_special_tokens=True)

     Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
     'Michael Jordan is known for being one of the greatest basketball players of all time. He is also known for being one of the most successful businessmen in the world.
     Jordan is the owner of the Charlotte Hornets, a professional basketball team in the NBA. He is also the owner of'
```

Simple question reasoning with the 5 decoding strategies:

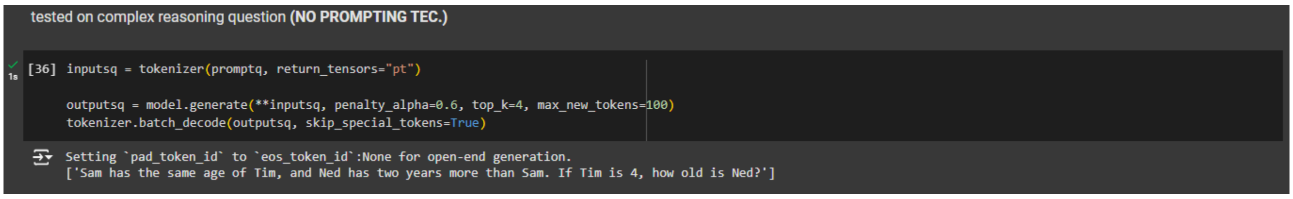
1.


Test on complex reasoning question **without prompting** (NOT TO DO)

```
[33] promptq = "Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned?"
     llama_responseq = get_response(promptq)
     print(llama_responseq)
```

```
Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned? A) 8 B) 10 C) 6 D) 7 E) 5
Answer: C
Explanation: Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4,
```

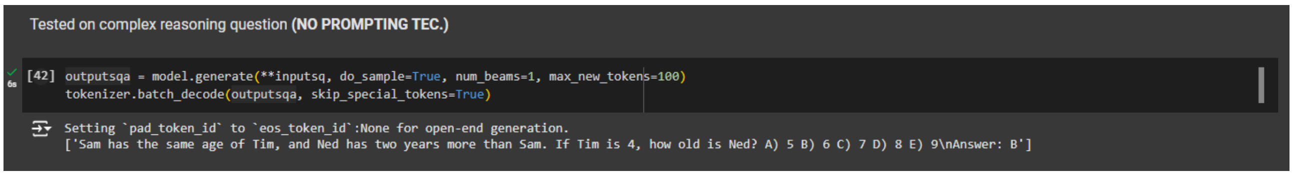
2.


tested on complex reasoning question **(NO PROMPTING TEC.)**

```
[36] inputsq = tokenizer(promptq, return_tensors="pt")

     outputsq = model.generate(**inputsq, penalty_alpha=0.6, top_k=4, max_new_tokens=100)
     tokenizer.batch_decode(outputsq, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
['Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned?']
```

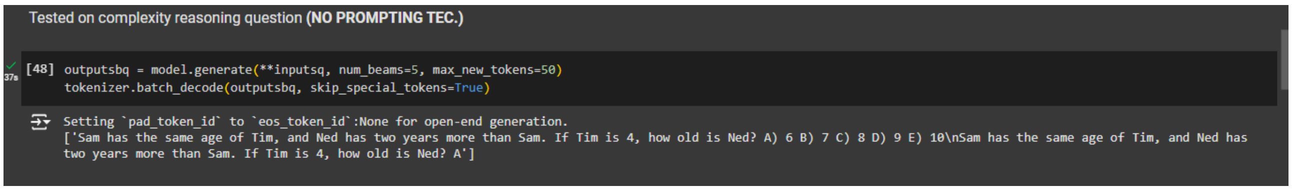
3.


Tested on complex reasoning question **(NO PROMPTING TEC.)**

```
[42] outputsqa = model.generate(**inputsq, do_sample=True, num_beams=1, max_new_tokens=100)
     tokenizer.batch_decode(outputsqa, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
['Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned? A) 5 B) 6 C) 7 D) 8 E) 9\nAnswer: B']
```

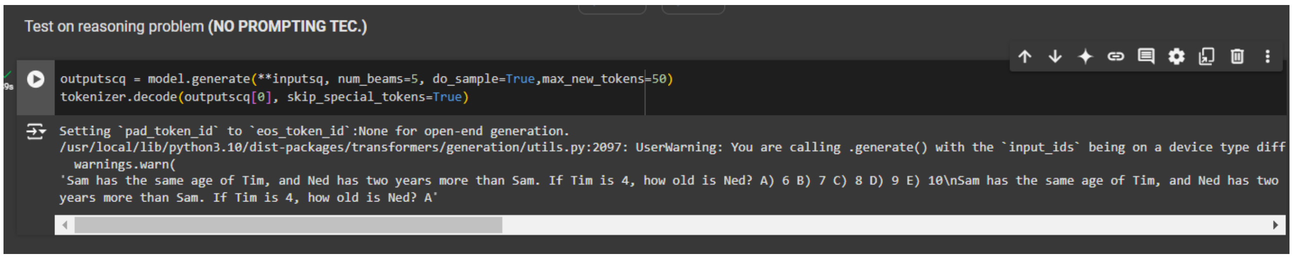
4.


Tested on complexity reasoning question **(NO PROMPTING TEC.)**

```
[48] outputsbq = model.generate(**inputsq, num_beams=5, max_new_tokens=50)
     tokenizer.batch_decode(outputsbq, skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
['Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned? A) 6 B) 7 C) 8 D) 9 E) 10\nSam has the same age of Tim, and Ned has
two years more than Sam. If Tim is 4, how old is Ned? A']
```

5.


Test on reasoning problem **(NO PROMPTING TEC.)**

```
outputscq = model.generate(**inputsq, num_beams=5, do_sample=True,max_new_tokens=50)
tokenizer.decode(outputscq[0], skip_special_tokens=True)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:2097: UserWarning: You are calling .generate() with the `input_ids` being on a device type diff
  warnings.warn(
'Sam has the same age of Tim, and Ned has two years more than Sam. If Tim is 4, how old is Ned? A) 6 B) 7 C) 8 D) 9 E) 10\nSam has the same age of Tim, and Ned has two
years more than Sam. If Tim is 4, how old is Ned? A'
```

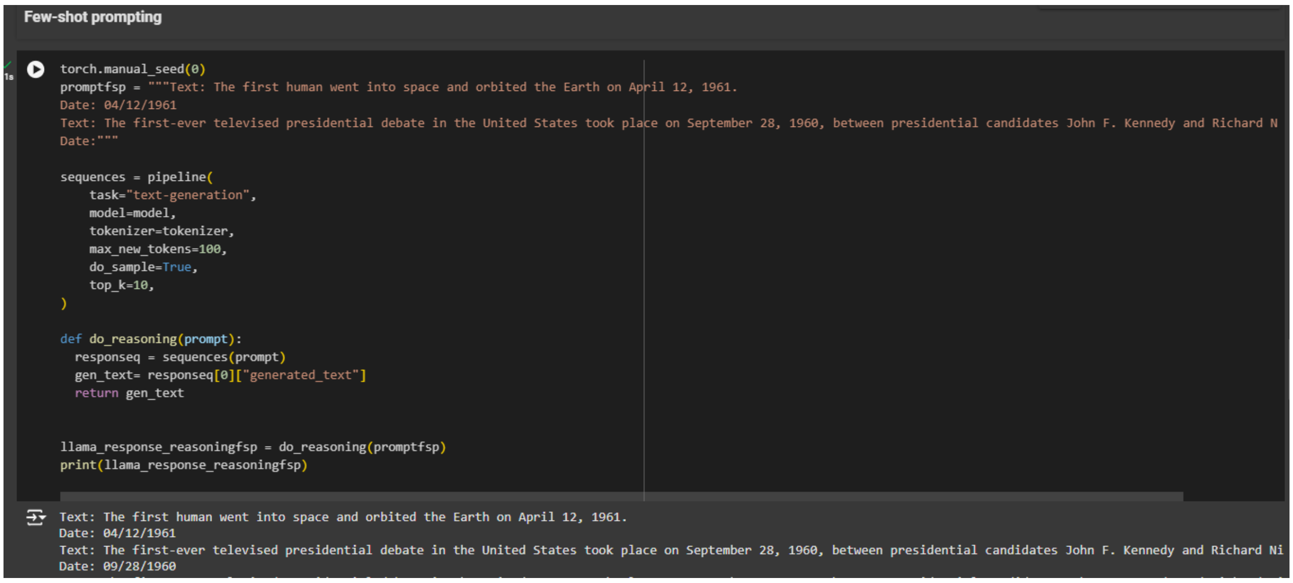
The best performing ones surprisingly seem to be 1.Greedy search.

As for reasoning-based problems, I think the non-trained non-prompted base model performs quite poorly.

We can see great improvement in reasoning when it comes to extensive pretraining or detailed prompting like Chain-of-thoughts.

Prompting:

Few shots prompting

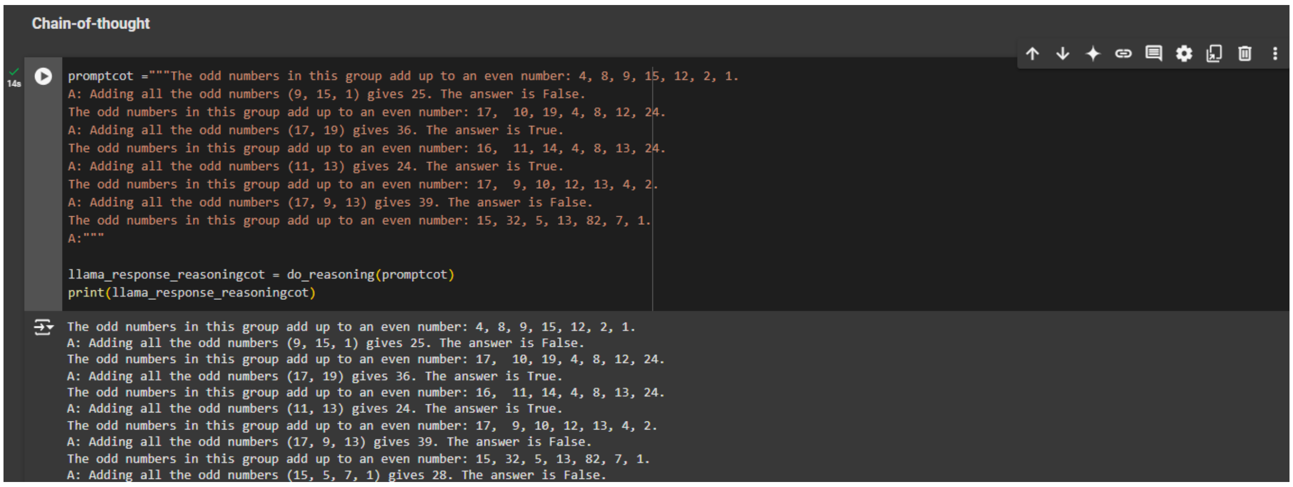


Few-shot prompting

```
torch.manual_seed(0)
promptfsp = """Text: The first human went into space and orbited the Earth on April 12, 1961.
Date: 04/12/1961
Text: The first-ever televised presidential debate in the United States took place on September 28, 1960, between presidential candidates John F. Kennedy and Richard N
Date:"""

sequences = pipeline(
    task="text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=100,
    do_sample=True,
    top_k=10,
)

def do_reasoning(prompt):
  responseq = sequences(prompt)
  gen_text= responseq[0]["generated_text"]
  return gen_text


llama_response_reasoningfsp = do_reasoning(promptfsp)
print(llama_response_reasoningfsp)
```

```
Text: The first human went into space and orbited the Earth on April 12, 1961.
Date: 04/12/1961
Text: The first-ever televised presidential debate in the United States took place on September 28, 1960, between presidential candidates John F. Kennedy and Richard Ni
Date: 09/28/1960
```

Chain of thoughts



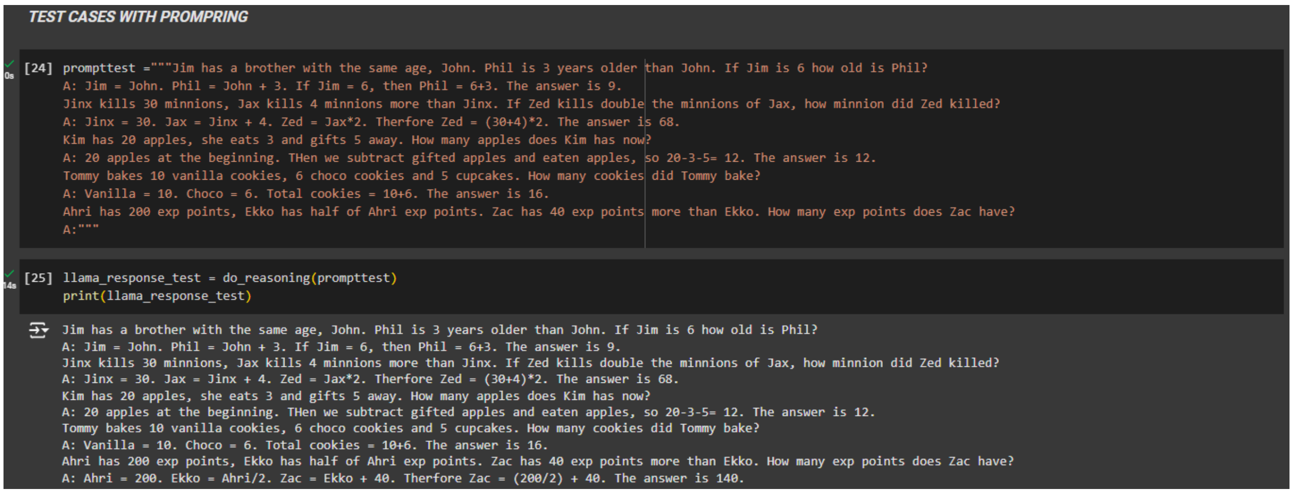
Chain-of-thought

```
promptcot ="""The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A:"""

llama_response_reasoningcot = do_reasoning(promptcot)
print(llama_response_reasoningcot)
```

```
The odd numbers in this group add up to an even number: 4, 8, 9, 15, 12, 2, 1.
A: Adding all the odd numbers (9, 15, 1) gives 25. The answer is False.
The odd numbers in this group add up to an even number: 17,  10, 19, 4, 8, 12, 24.
A: Adding all the odd numbers (17, 19) gives 36. The answer is True.
The odd numbers in this group add up to an even number: 16,  11, 14, 4, 8, 13, 24.
A: Adding all the odd numbers (11, 13) gives 24. The answer is True.
The odd numbers in this group add up to an even number: 17,  9, 10, 12, 13, 4, 2.
A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.
The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1.
A: Adding all the odd numbers (15, 5, 7, 1) gives 28. The answer is False.
```

Chain of thoughts good and bad prompting cases:

Nice propting/training example:



TEST CASES WITH PROMPRING

```
[24] prompttest ="""Jim has a brother with the same age, John. Phil is 3 years older than John. If Jim is 6 how old is Phil?
     A: Jim = John. Phil = John + 3. If Jim = 6, then Phil = 6+3. The answer is 9.
     Jinx kills 30 minnions, Jax kills 4 minnions more than Jinx. If Zed kills double the minnions of Jax, how minnion did Zed killed?
     A: Jinx = 30. Jax = Jinx + 4. Zed = Jax*2. Therfore Zed = (30+4)*2. The answer is 68.
     Kim has 20 apples, she eats 3 and gifts 5 away. How many apples does Kim has now?
     A: 20 apples at the beginning. THen we subtract gifted apples and eaten apples, so 20-3-5= 12. The answer is 12.
     Tommy bakes 10 vanilla cookies, 6 choco cookies and 5 cupcakes. How many cookies did Tommy bake?
     A: Vanilla = 10. Choco = 6. Total cookies = 10+6. The answer is 16.
     Ahri has 200 exp points, Ekko has half of Ahri exp points. Zac has 40 exp points more than Ekko. How many exp points does Zac have?
     A:"""
```

```
[25] llama_response_test = do_reasoning(prompttest)
     print(llama_response_test)
```

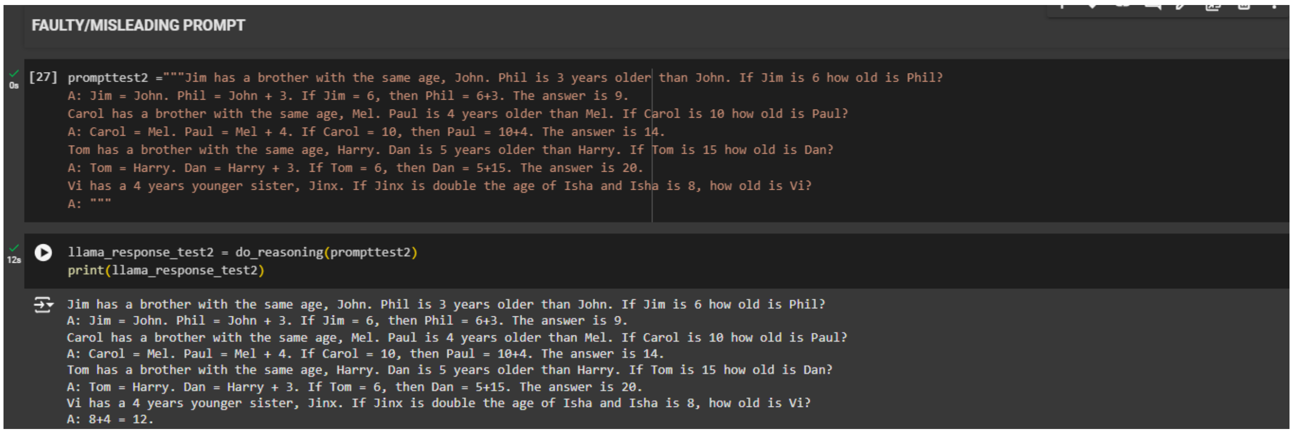
```
Jim has a brother with the same age, John. Phil is 3 years older than John. If Jim is 6 how old is Phil?
A: Jim = John. Phil = John + 3. If Jim = 6, then Phil = 6+3. The answer is 9.
Jinx kills 30 minnions, Jax kills 4 minnions more than Jinx. If Zed kills double the minnions of Jax, how minnion did Zed killed?
A: Jinx = 30. Jax = Jinx + 4. Zed = Jax*2. Therfore Zed = (30+4)*2. The answer is 68.
Kim has 20 apples, she eats 3 and gifts 5 away. How many apples does Kim has now?
A: 20 apples at the beginning. THen we subtract gifted apples and eaten apples, so 20-3-5= 12. The answer is 12.
Tommy bakes 10 vanilla cookies, 6 choco cookies and 5 cupcakes. How many cookies did Tommy bake?
A: Vanilla = 10. Choco = 6. Total cookies = 10+6. The answer is 16.
Ahri has 200 exp points, Ekko has half of Ahri exp points. Zac has 40 exp points more than Ekko. How many exp points does Zac have?
A: Ahri = 200. Ekko = Ahri/2. Zac = Ekko + 40. Therfore Zac = (200/2) + 40. The answer is 140.
```

In the above attempt we included different math problems that combined can lead to the solution to the last problem. That would not be the case if we instruct the prompt with only "one-type" of resolution. We could intentionally or non-intentionally mislead the model into believing that that the problem must be solved like the above ones.

In most cases, math/logic related to the use of numerical expressions linked with explanations have shown to be much more efficient for training the model.

Model misleading is caused mainly because of lack of variety in out prompt/dataset. Models do not really “reason” when solving problems but rather combine pattern that have been memorized from previous training.

I will show an example of misleading prompts below:



FAULTY/MISLEADING PROMPT

```
[27] prompttest2 ="""Jim has a brother with the same age, John. Phil is 3 years older than John. If Jim is 6 how old is Phil?
     A: Jim = John. Phil = John + 3. If Jim = 6, then Phil = 6+3. The answer is 9.
     Carol has a brother with the same age, Mel. Paul is 4 years older than Mel. If Carol is 10 how old is Paul?
     A: Carol = Mel. Paul = Mel + 4. If Carol = 10, then Paul = 10+4. The answer is 14.
     Tom has a brother with the same age, Harry. Dan is 5 years older than Harry. If Tom is 15 how old is Dan?
     A: Tom = Harry. Dan = Harry + 3. If Tom = 6, then Dan = 5+15. The answer is 20.
     Vi has a 4 years younger sister, Jinx. If Jinx is double the age of Isha and Isha is 8, how old is Vi?
     A: """
```

```
llama_response_test2 = do_reasoning(prompttest2)
print(llama_response_test2)
```

```
Jim has a brother with the same age, John. Phil is 3 years older than John. If Jim is 6 how old is Phil?
A: Jim = John. Phil = John + 3. If Jim = 6, then Phil = 6+3. The answer is 9.
Carol has a brother with the same age, Mel. Paul is 4 years older than Mel. If Carol is 10 how old is Paul?
A: Carol = Mel. Paul = Mel + 4. If Carol = 10, then Paul = 10+4. The answer is 14.
Tom has a brother with the same age, Harry. Dan is 5 years older than Harry. If Tom is 15 how old is Dan?
A: Tom = Harry. Dan = Harry + 3. If Tom = 6, then Dan = 5+15. The answer is 20.
Vi has a 4 years younger sister, Jinx. If Jinx is double the age of Isha and Isha is 8, how old is Vi?
A: 8+4 = 12.
```

Regarding **commercial applications** the best combination would be a general pretrained model, perhaps trainable in a second moment on a custom dataset based on the type of use. Ex. Hospital homepage chatbot (Need training on customer service content, medical content, scientific worlds, glocalization, payments method…)

Rather than specific decoding strategies the dataset pretraining/prompting shows far better improvements in performance and context relevance.