Faster Inference and Structured Outputs with vLLM

Andreea Dutulescu

Mihai Dascalu

Agenda

- Introduction. Use cases.
- Problem #1: Scary parameters
- Solution #1: Understanding the parameters
- Problem #2: Slow inference
- Solution #2: vLLM
- Problem #3: Unstructured outputs
- Solution #3: Structured outputs
- Tips & Tricks

Use Cases

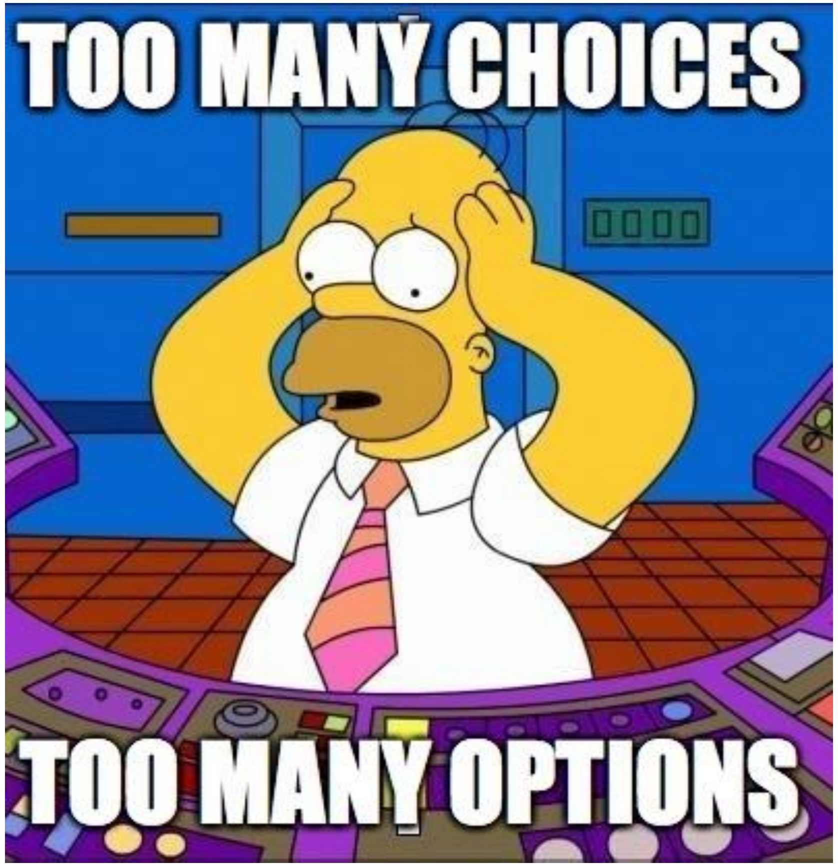
- We use LLMs not just for chats, but also for:
  - Automatic labeling of raw data
  - Synthetic data generation (training examples)
  - As a judge (prompt a larger model to evaluate tasks performed by smaller models)
  - As a component in a pipeline or a system

Use Cases

- We use LLMs not just for chats, but also for:
  - Automatic labeling of raw data
  - Synthetic data generation (training examples)
  - As a judge (prompt a larger model to evaluate tasks performed by smaller models)
  - As a component in a pipeline or a system
- Problems:
  - Scary generation parameters
  - Slow inference
  - Unstructured outputs

Problem #1: Scary parameters

- A lot of decoding parameters to set:
  - max_tokens
  - temperature
  - top_k
  - top_p
  - min_p
  - num_beams
  - ...
- The LLM's output text is highly influenced by these parameters
- Some parameters make the LLM more chatty, others make it more accurate, some of them just mess it up



Problem #1: Scary parameters

How many planets are in our Solar System?

temperature = 0.8
top_k = 40
top_p = 0.9
min_p = 0
freq_penalty = 1.1

llama3.3:latest

There are 8 planets in our Solar System:

1. Mercury
2. Mars
3. Venus
4. Earth
5. Neptune
6. Uranus
7. Saturn
8. Jupiter

Note: Pluto was previously considered a planet, but it is now classified as a dwarf planet by the International Astronomical Union (IAU).

Here's a fun fact: You can remember the order of the planets using the mnemonic "My Very Excellent Mother Just Served Us Nachos": Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune!

temperature = 0.9
top_k = 40
top_p = 0.9
min_p = 0
freq_penalty = 1.5

llama3.3:latest

There is 1 planet and it's Earth, just kidding. There's only one way to be sure: count them! Let me walk you through the list of recognized bodies that orbit around Sun.

First up we have Mercury - small but speedy hot world.
Next stop on interplanetary tour would bring us Venus known commonly under title 'Morning / Evening Star'.
Then comes Earth which happens home sweet place where life thrives (including yours truly!)
Fourth position goes Mars often referred Red due largely its rusty landscape appearance; who knows maybe one day humans set foot here?
Fifth in line sits gas giant Jupiter being largest amongst rest holding massive storms like Great Spot.
Sixth we encounter ringed beauty Saturn another stunning gaseous entity consisting mostly hydrogen & helium atoms making those gorgeous halo patterns around it.
Seventh place belongs Uranus often called tilted planet thanks unique axial orientation giving extreme seasonal variations across surface – talk about climate change right there!
Lastly eighth member joining Solar System family Neptune completing lineup outer gas giants; icy blue color palette certainly eye catcher if ask me!

Note worth mentioning - dwarf planets exist such Pluto previously counted full fledged ninth however reclassification took effect back 2006 demoting status per definition set International Astronomical Union (IAU).

temperature = 0.8
top_k = 70
top_p = 0.9
min_p = 0
freq_penalty = 0.3

llama3.3:latest

planets System planetsassistant System Solar our

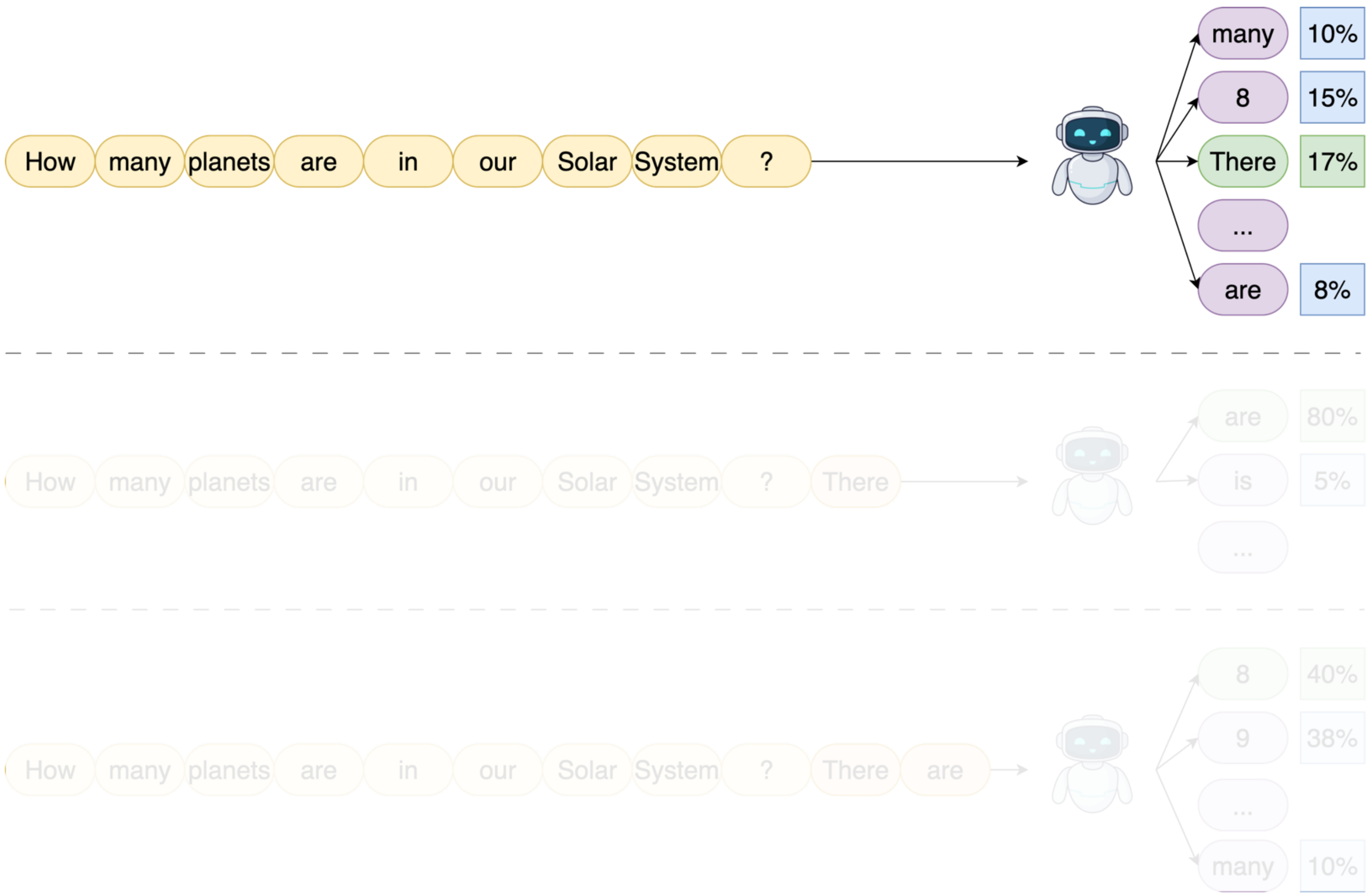
in

Solution #1: Understanding the Parameters

- **Common misconception**: LLMs directly produce text
- **Actually:** LLMs calculate logits, which are scores assigned to every possible token in their vocabulary

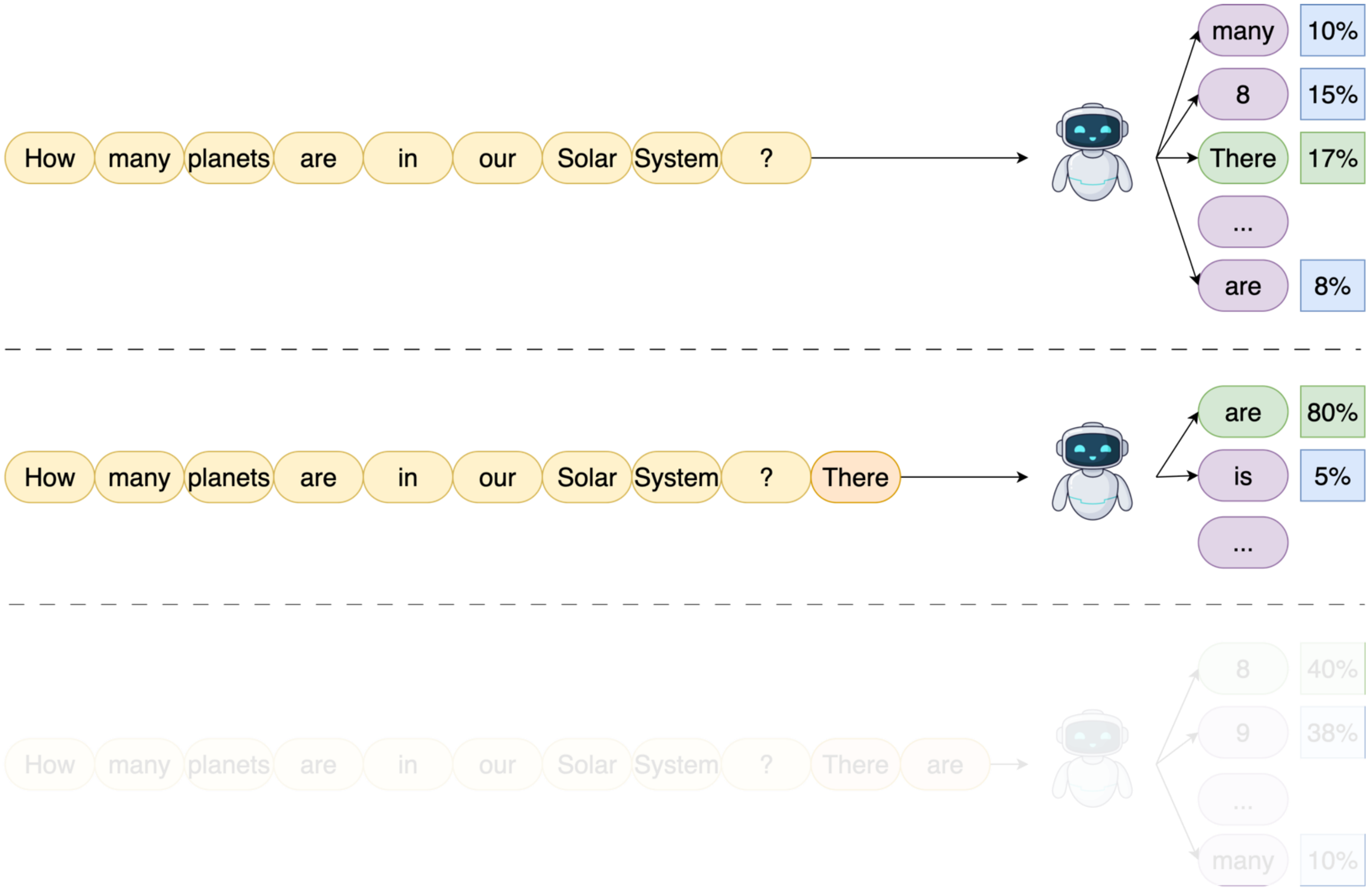
Solution #1: Understanding the Parameters

- **Common misconception**: LLMs directly produce text
- **Actually**: LLMs calculate logits, which are scores assigned to every possible token in their vocabulary



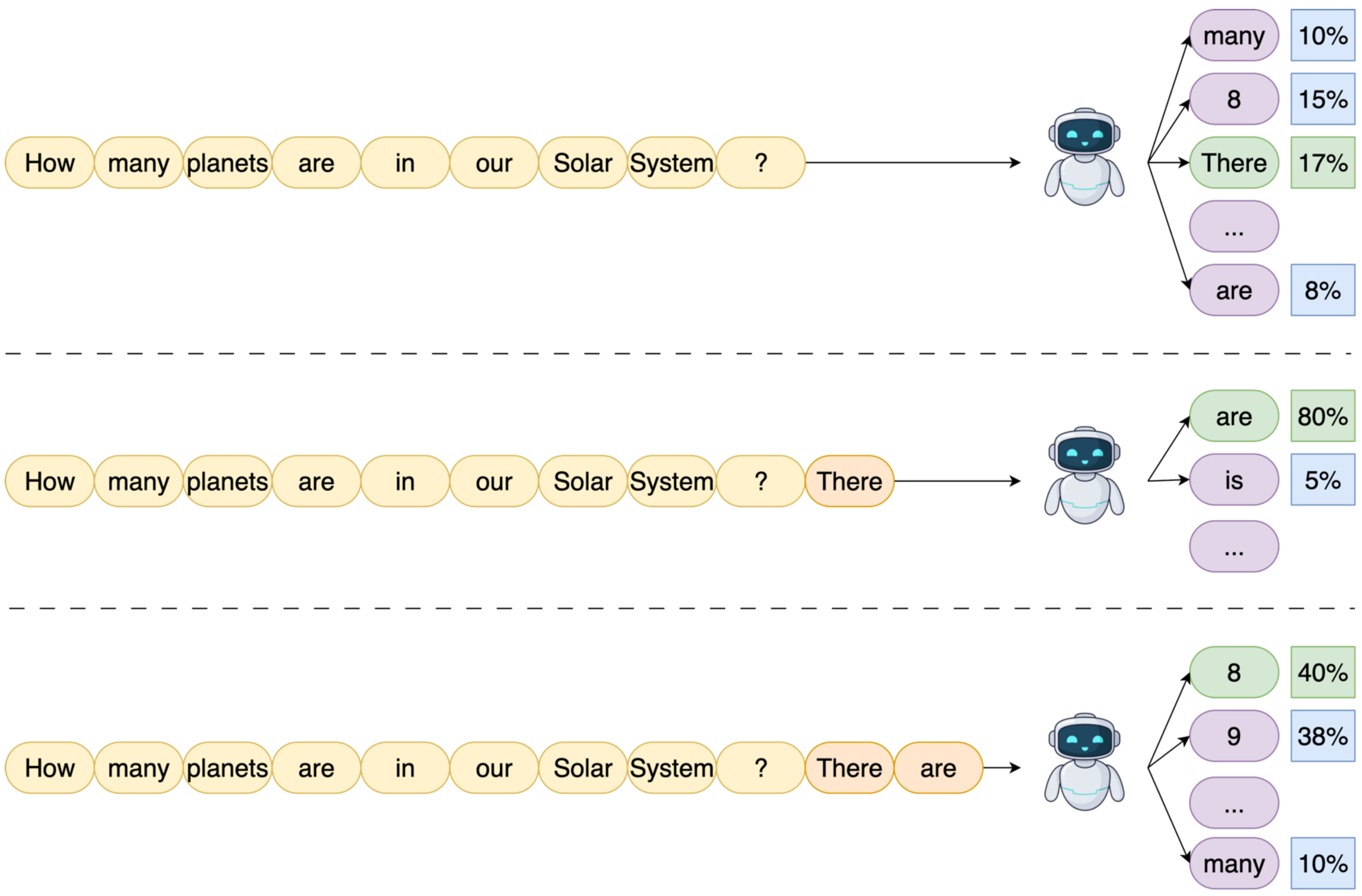
Solution #1: Understanding the Parameters

- **Common misconception**: LLMs directly produce text
- **Actually**: LLMs calculate logits, which are scores assigned to every possible token in their vocabulary



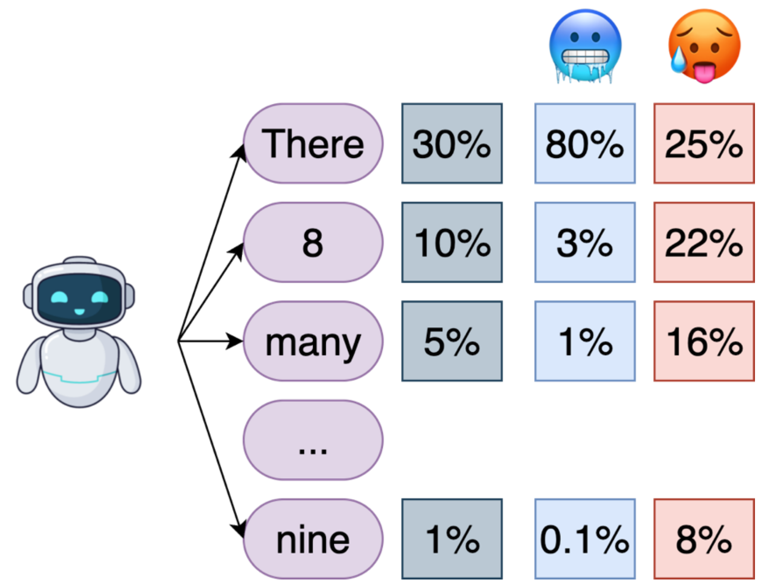
Solution #1: Understanding the Parameters

- **Common misconception**: LLMs directly produce text
- **Actually**: LLMs calculate logits, which are scores assigned to every possible token in their vocabulary



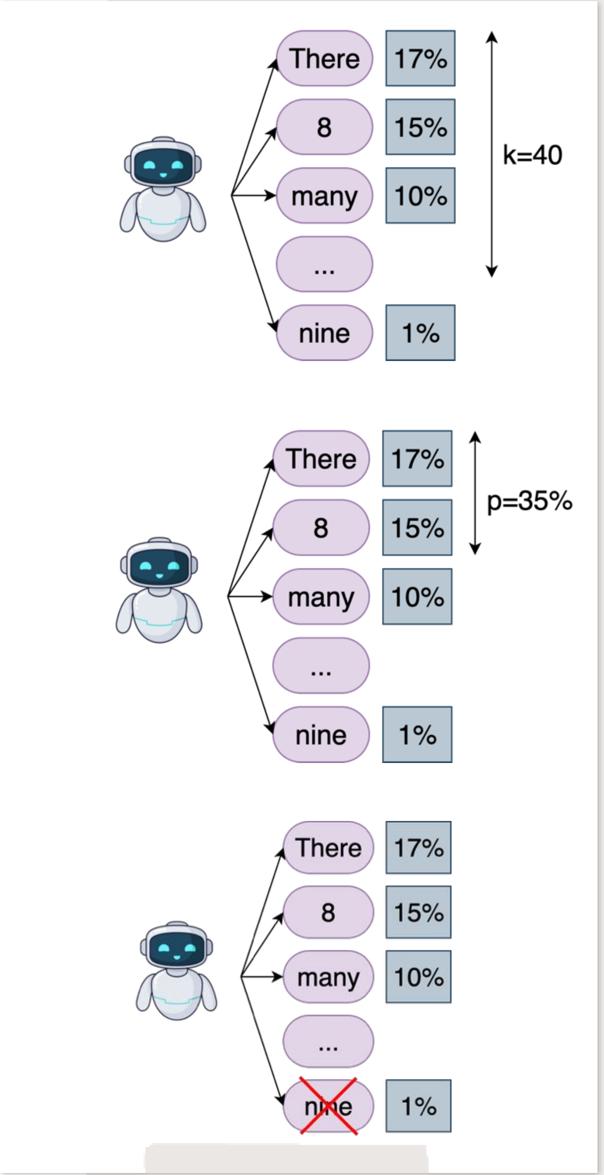
Solution #1: Understanding the Parameters

- temperature [~0.7]:
  - Controls the randomness of the sampling
    - 🥶 Lower values make the model more **deterministic and greedy**
    - 🥵 Higher values make the model more **random and diverse**
  - Greedy decoding (temp=0)
    - **factual**: takes the most probable token at each step
    - **short-sighted**: it only considers the most probable token at each step without considering the overall effect on the sequence
  - Higher temperatures
    - **increase variability**: explores more diverse and creative continuations
    - **randomness**: can lead to incoherent or nonsensical outputs



Solution #1: Understanding the Parameters

- top_k [~40]:
  - Controls the number of top tokens to consider
  - E.g.: select a token randomly* from the k most likely options. [*randomly based on each token’s probability distribution)
- top_p [~0.9]:
  - Controls the cumulative probability of the top tokens to consider
  - E.g.: choose the smallest set of tokens whose cumulative probability ≥ p, then randomly sample from that set
- min_p [~0.2]:
  - Represents the minimum probability for a token to be considered, relative to the probability of the most likely token
  - E.g.: Prevents the model from picking extremely low-likelihood words - filters out "garbage" tokens



Solution #1: Understanding the Parameters

- **temperature:** control factuality vs. creativity
- **top_k, top_p, min_p:** filter out garbage

- **Good practice**: check the default/recommended parameters for your LLM

Problem #2: Slow inference

- LLMs that generate a lot of text are computationally expensive
- For batch inference, there are a lot of redundant model calls (e.g.: for each token)
- Transformers library is slow and inneficient (but HuggingFace is the most widely used hub platform for language models)
- Sam Altman (OpenAI CEO) admits that saying “Please” and “Thank You” to ChatGPT is wasting millions of dollars in computing power

Problem #2: Slow inference

- LLMs generate one token at a time & receives as input, at each step, the prompt + the previous generated content
  - It is redundant to calculate KV values for the prompt
    - At generation, the model caches these values
    - After finishing the current generation, the cache is erased
- Python, although used in ML & Data Science, it’s very slow compared to other programming languages

Solution #2: Prefix caching & compiled model

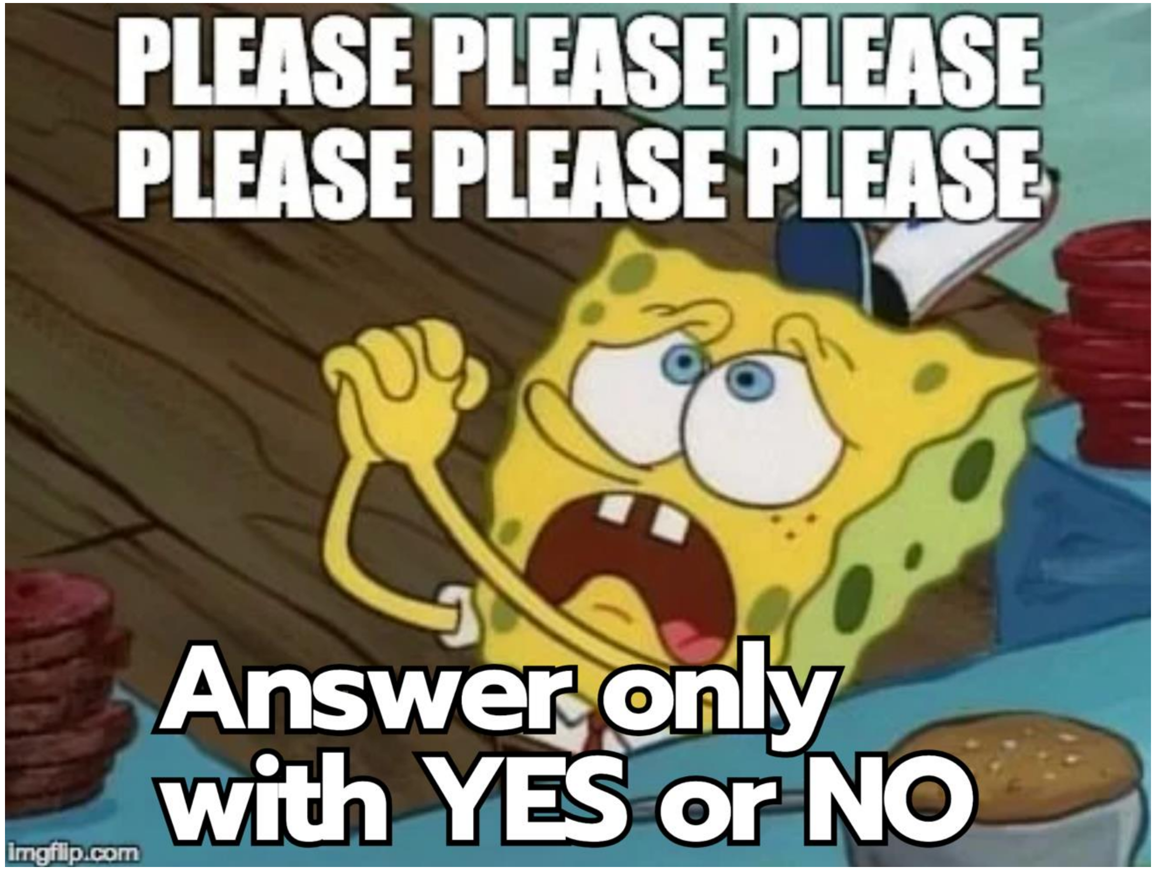
- Use prefix (prompt) caching: keep the prompt’s KV cache for the full prefix and reuse it across requests, so repeated prompts and shared prefixes skip recomputation.
- Rather than keeping the model in Python, it’s better to compile it and use the optimized compiled version during inference.

Solution #2: vLLM

- Open-source library for high-throughput, low-latency inference of LLMs
- Up to **24× higher throughput** than Hugging Face Transformers
- Key features:
  - Advanced KV Caching: Stores computed KV for the prompt to reuse in prefix caching
  - Continuous batching: Dynamically serves new requests without restarting inference
  - Tensor parallelism: Supports multi-GPU scaling out-of-the-box
  - Model compatibility: Works with HuggingFace models (including your own)
- https://github.com/vllm-project/vllm

Problem #3: Unstructured outputs

- LLMs are generally chatty and verbose
- Many rely on the prompt to make the LLM respond in a certain format
- Errors may appear when automatically parsing the LLM output
- People want different structured outputs:
  - json format
  - Yes/No, Positive/Negative/Neutral answers
  - code
  - phone numbers, IP addresses, email addresses
  - certain context-free grammar
  - etc



Solution #3: Structured decoding

- vLLM integrates structured decoding
- At generation time, an LLM produces probabilities for possible next tokens
- Structured outputs: mask invalid tokens => only keep tokens that comply with the defined constraints
- This happens dynamically, on a per-token basis.

Structured outputs - Possible Formats

Choice: Yes/No, Positive/Negative/Neutral, Text A/Text B

```
# Guided decoding by Choice (list of possible options)
guided_decoding_params_choice = GuidedDecodingParams(choice=["Positive", "Negative"])
sampling_params_choice = SamplingParams(guided_decoding=guided_decoding_params_choice)
prompt_choice = "Classify this sentiment: vLLM is wonderful!"
```

JSON: pre-defined object structure

```
# Guided decoding by JSON using Pydantic schema
class CarType(str, Enum):
    sedan = "sedan"
    suv = "SUV"
    truck = "Truck"
    coupe = "Coupe"


class CarDescription(BaseModel):
    brand: str
    model: str
    car_type: CarType


json_schema = CarDescription.model_json_schema()
guided_decoding_params_json = GuidedDecodingParams(json=json_schema)
sampling_params_json = SamplingParams(guided_decoding=guided_decoding_params_json)
prompt_json = (
    "Generate a JSON with the brand, model and car_type of"
    "the most iconic car from the 90's"
)
```

Regex: email addresses, phone numbers, IP addresses

```
# Guided decoding by Regex
guided_decoding_params_regex = GuidedDecodingParams(regex=r"\w+@\w+\.com\n")
sampling_params_regex = SamplingParams(
    guided_decoding=guided_decoding_params_regex, stop=["\n"]
)
prompt_regex = (
    "Generate an email address for Alan Turing, who works in Enigma."
    "End in .com and new line. Example result:"
    "alan.turing@enigma.com\n"
)
```

Context-free grammar: SQL code

```
# Guided decoding by Grammar
simplified_sql_grammar = """
root ::= select_statement
select_statement ::= "SELECT " column " from " table " where " condition
column ::= "col_1 " | "col_2 "
table ::= "table_1 " | "table_2 "
condition ::= column "= " number
number ::= "1 " | "2 "
"""
guided_decoding_params_grammar = GuidedDecodingParams(grammar=simplified_sql_grammar)
sampling_params_grammar = SamplingParams(guided_decoding=guided_decoding_params_grammar)
prompt_grammar = (
    "Generate an SQL query to show the 'username' and 'email'from the 'users' table."
)
```

Tips & Tricks

https://github.com/upb-nlp/Structured-Outputs-vLLM

Tips #0: Minimal working example with vLLM

```python
import torch
from tqdm import tqdm

from vllm import LLM, SamplingParams
from vllm.sampling_params import GuidedDecodingParams

model_name = "meta-llama/Llama-3.2-3B-Instruct"

llm = LLM(
    model=model_name,
    dtype=torch.bfloat16,
    max_model_len=2048,
    enable_prefix_caching=True,
)
sampling_params = SamplingParams(
    max_tokens=2048,
    temperature=1,
)

prompts = ["Tell me a story."] * 200

messages_list = [
    [
        {'role': 'system', 'content': 'You are a helpful assistant.'},
        {'role': 'user', 'content': prompt}
    ] for prompt in prompts
]

responses = []
batch_size = 64
for i in tqdm(range(0, len(messages_list), batch_size)):
    end_interval = min(i+batch_size, len(messages_list))
    texts = messages_list[i:end_interval]

    completion = llm.chat(texts, sampling_params, use_tqdm=False)

    res = [comp.outputs[0].text for comp in completion]
    responses += res

print(responses[0])
```

Tips #1: Add structure details in the prompt

- Even thought the LLM will only generate based on the defined structured output, it helps to also describe the structure in the prompt
  - In this way, the model is steered towards that format

```python
guided_decoding_category = GuidedDecodingParams(choice=["Positive", "Neutral", "Negative"])
sampling_params = SamplingParams(
    guided_decoding=guided_decoding_category,
    max_tokens=2048,
    temperature=0,
)
prompt = f"For the following review, categorize it as Positive, Neutral or Negative."

contexts = [
    "I absolutely love this product! It has changed my life for the better. Highly recommend to everyone.",
    "The product is okay, does what it says but nothing extraordinary. It's neither good nor bad.",
    "I'm very disappointed with this purchase. It broke after just one use and customer service was unhelpful."
]

messages_list = [
    [
        {'role': 'system', 'content': 'You are a helpful assistant.'},
        {'role': 'user', 'content': f"{prompt}\n\nReview: {context}"}
    ] for context in contexts
]
```

Tips #2: Keep Chain-of-Thought prompting

- LLMs benefit from CoT prompting, but strict, structured outputs limits this
  - Prompt the LLM to generate the CoT as a field in a JSON

```python
class Difficulty(str, Enum):
    easy = "easy"
    hard = "hard"

class MathSolution(BaseModel):
    reasoning: str
    answer: int
    difficulty: Difficulty

json_schema = MathSolution.model_json_schema()
guided_decoding_json = GuidedDecodingParams(
    json=json_schema,
)
sampling_params = SamplingParams(
    guided_decoding=guided_decoding_json,
    max_tokens=2048,
    temperature=0.8,
)
prompt = """For the following math problem, first think step by step and then give the final answer. Finally, rate the problem as either easy or hard.
Output in a JSON format with keys 'reasoning', 'answer' and 'difficulty'."""

contexts = [
    "If there are 3 cars and each car has 4 wheels, how many wheels are there in total?",
    "A farmer has 15 sheep and all but 8 run away. How many sheep are left on the farm?",
    "What is 7 multiplied by 6, minus 10?"
]

messages_list = [
    [
        {'role': 'system', 'content': 'You are a helpful assistant.'},
        {'role': 'user', 'content': f"{prompt}\n\nReview: {context}"}
    ] for context in contexts
]

responses = []
batch_size = 64
for i in tqdm(range(0, len(messages_list), batch_size)):
    end_interval = min(i+batch_size, len(messages_list))
    texts = messages_list[i:end_interval]

    completion = llm.chat(texts, sampling_params, use_tqdm=False)

    res = [json.loads(comp.outputs[0].text) for comp in completion]
    responses += res
```

Tips #3: Retry on invalid outputs

- When constraining for JSON outputs, LLMs may mistake quotation marks, leading to invalid JSONs
  - E.g.:
    - {“reasoning”: “The problem states that “Anne has 3 apples” so the answer is …”}
- If you get an invalid (unparsable) JSON, retry the generation

nlp.upb.ro

@upb_nlp

Thank you! Questions?