Interacting with LLMs

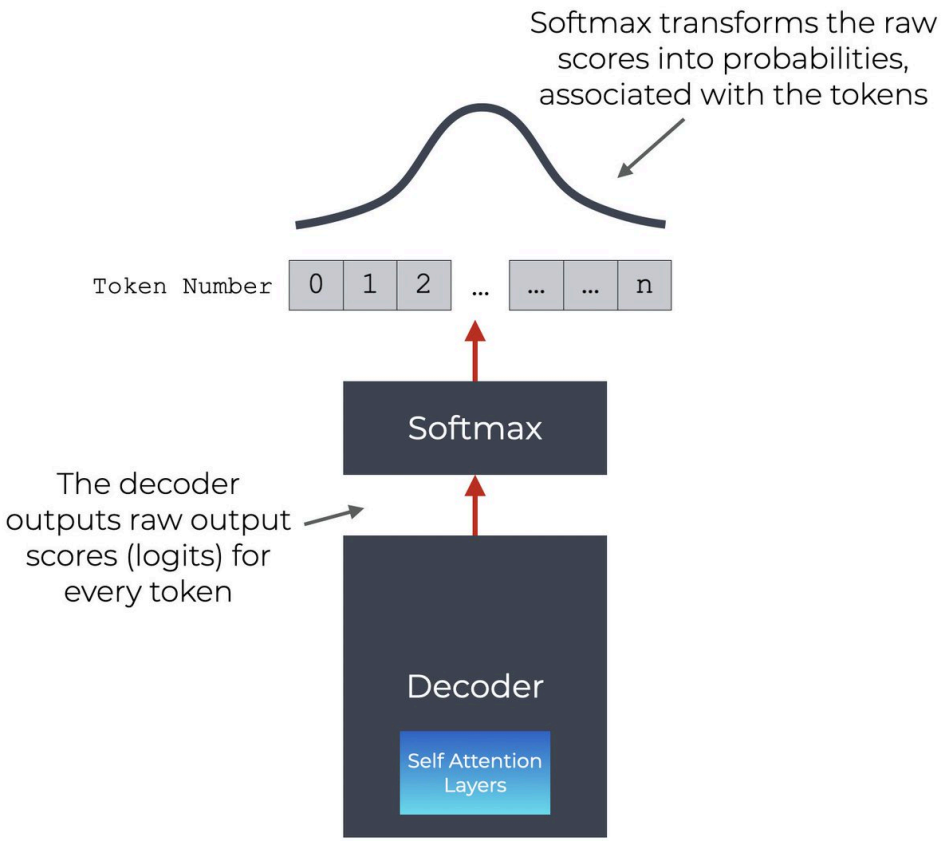
Iván Moreno (ivan@nieveconsulting.com)

Table of Contents

Text Generation Process

- **Step 3: Decoding**
  - The output is a probability distribution over the vocabulary for the next token.
  - **Example**: After the input `"Hello,"`, the model predicts the next word with probabilities:
    - `"world"`: 0.7
    - `"there"`: 0.2
    - `"everyone"`: 0.1
- **Step 4: Sampling**
  - A decoding strategy is applied to sample or select the next token (e.g., greedy, beam search, top-k, top-p).
  - The selected token is added to the input sequence, and the process repeats for the desired length.

Decoding the Output



- **Logits**: The model outputs logits, which are raw, unnormalized scores for each token in the vocabulary.
- **Softmax Function**: Converts logits into a probability distribution over the vocabulary

- **Example**:
  - Input: `"Hello,"`
  - Logits: `[5.1, 2.3, 3.8, ...]` (for the entire vocabulary)
  - Probabilities: After applying softmax:
    - `"world"`: 0.7
    - `"there"`: 0.2
    - `"everyone"`: 0.1

Next Token Prediction

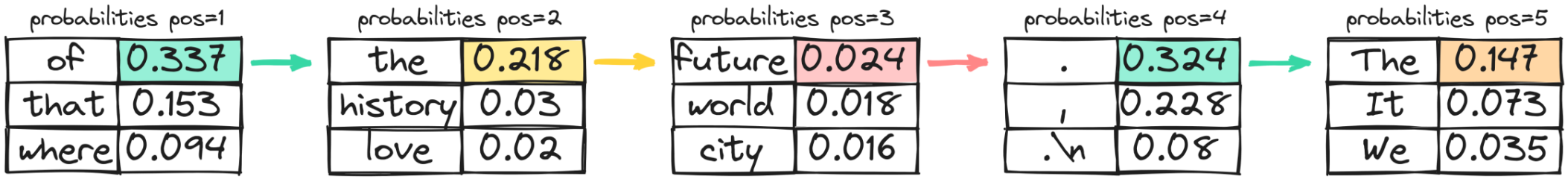
- **Prediction**: The model predicts the next token by selecting from the probability distribution.
  - **Deterministic Approach**: Choosing the token with the highest probability (e.g., `"world"` with a probability of 0.7).
  - **Example**:
    - Given the input `"Hello,"`, the most likely next word is predicted to be `"world"`.

Sampling Strategies

- **Greedy Sampling**: Selects the token with the highest probability.
- **Beam Search**: Expands multiple sequences in parallel and chooses the one with the highest cumulative probability.
- **Top-k Sampling**: Limits sampling to the top k highest probability tokens.
- **Top-p (Nucleus) Sampling**: Selects tokens until their cumulative probability exceeds a threshold p.

Sampling Methods

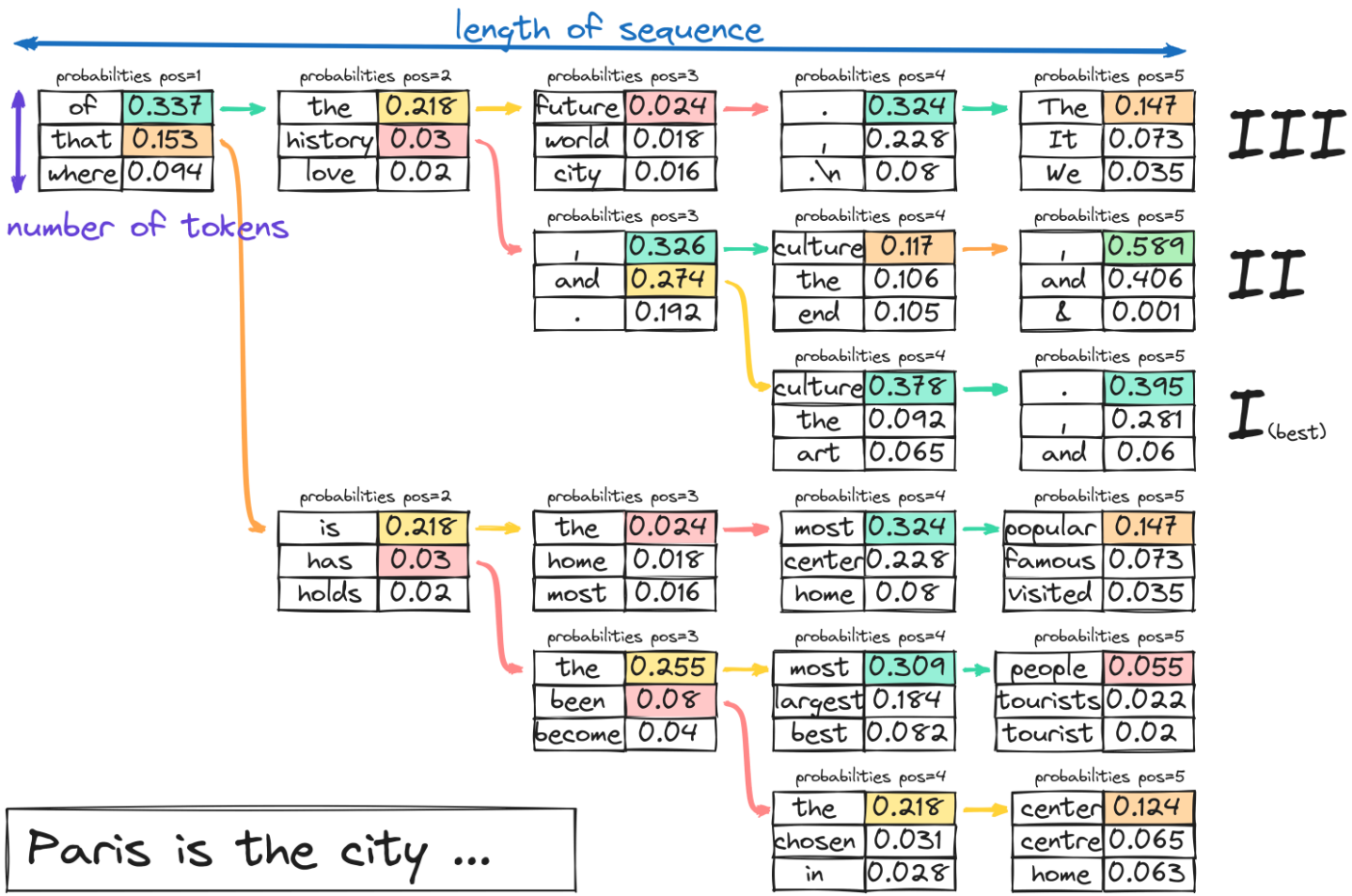
Greedy Sampling



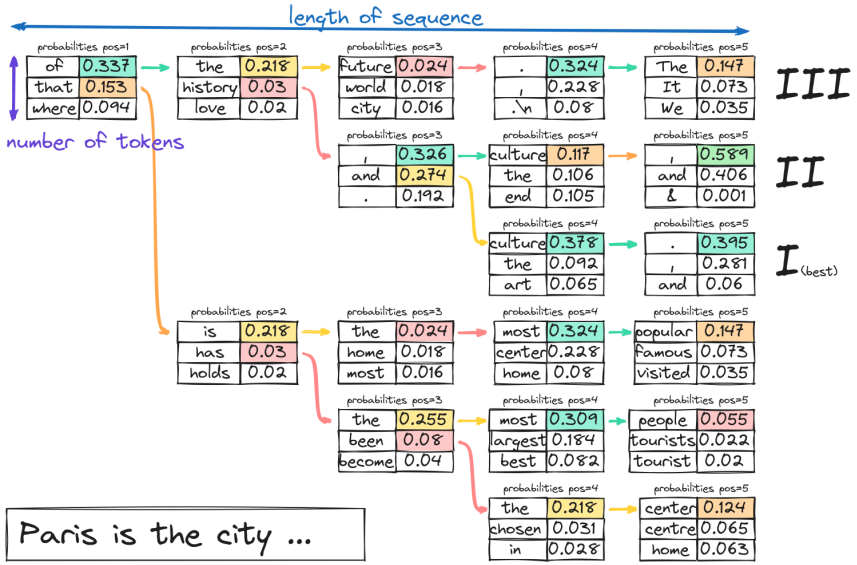
- **Description**: Always selects the token with the highest probability at each step.
- **Advantages**:
  - Simple and fast.
  - Produces deterministic outputs.
- **Disadvantages**:
  - Tends to generate repetitive or boring text.
  - Can get stuck in local optima, missing out on more creative or contextually appropriate continuations.
- **Example**:

```
output = model.generate(inputs, max_length=50)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Beam Search



- **Description**: Expands multiple hypotheses (beams) at each step and selects the top sequences based on cumulative probability.
- **Beam Width**: The number of beams (possible sequences) considered at each step.

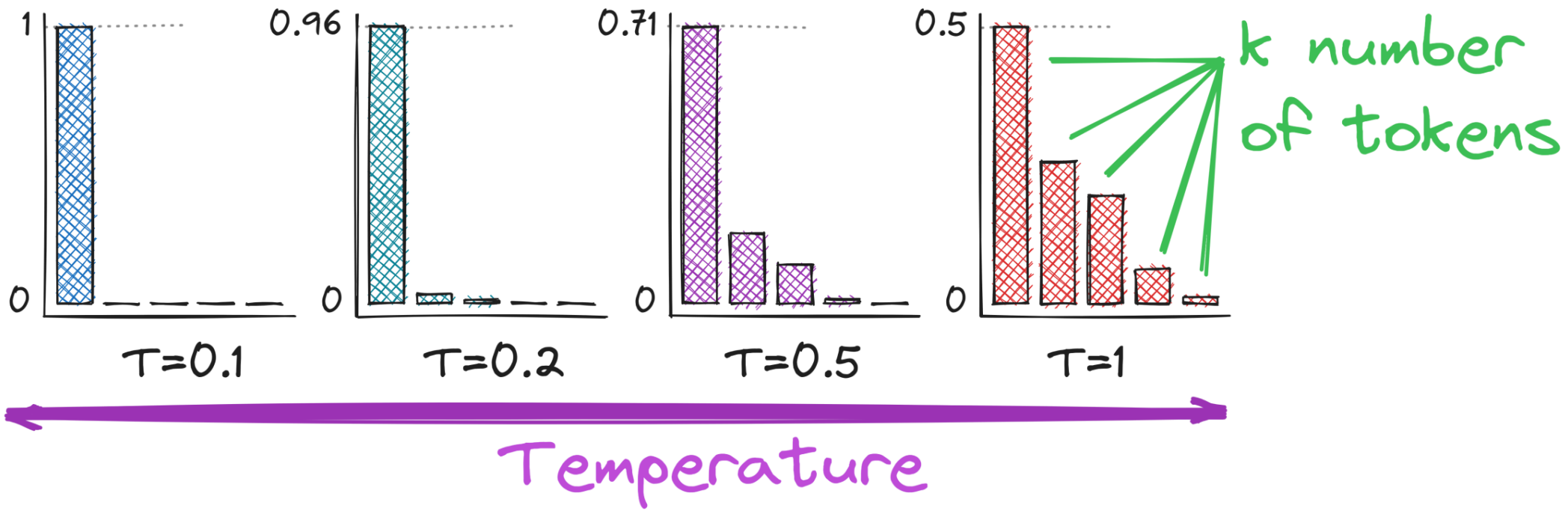


- **Advantages**:
  - More diverse and higher-quality outputs than greedy sampling.
  - Better at finding globally optimal sequences.
- **Disadvantages**:
  - Computationally more expensive.
  - Can still produce repetitive sequences if all beams converge on similar paths.
- **Example**:

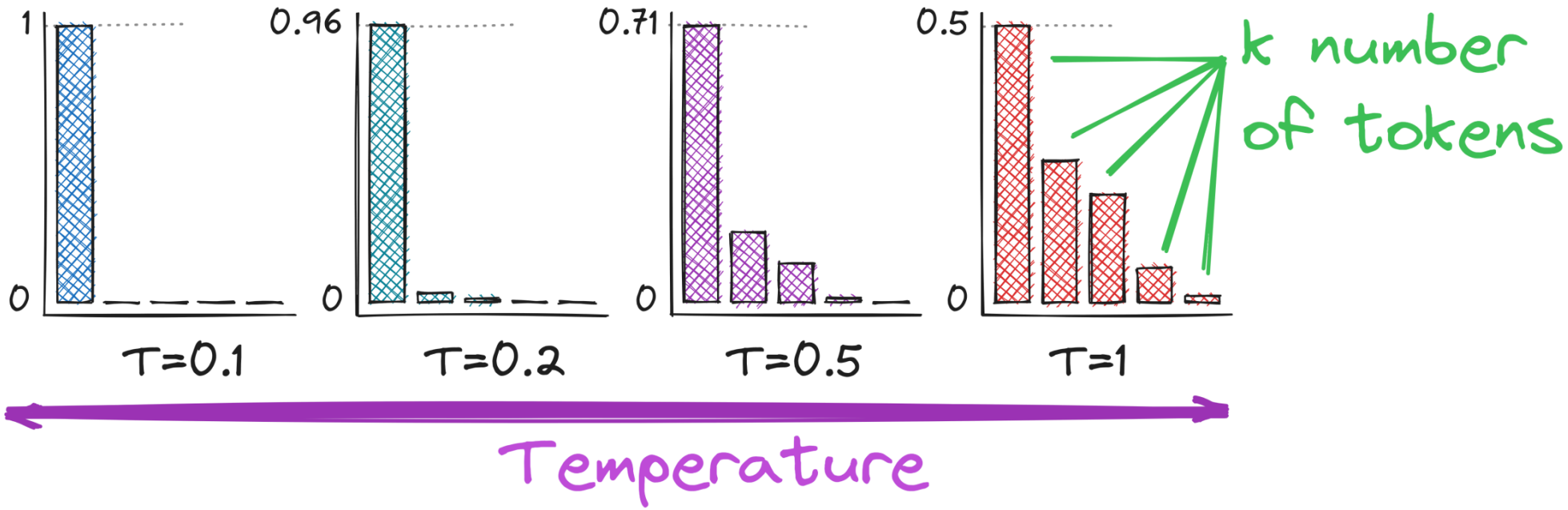
```
output = model.generate(inputs, max_length=50, num_beams=5)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Decoding Strategies

Top-k Sampling



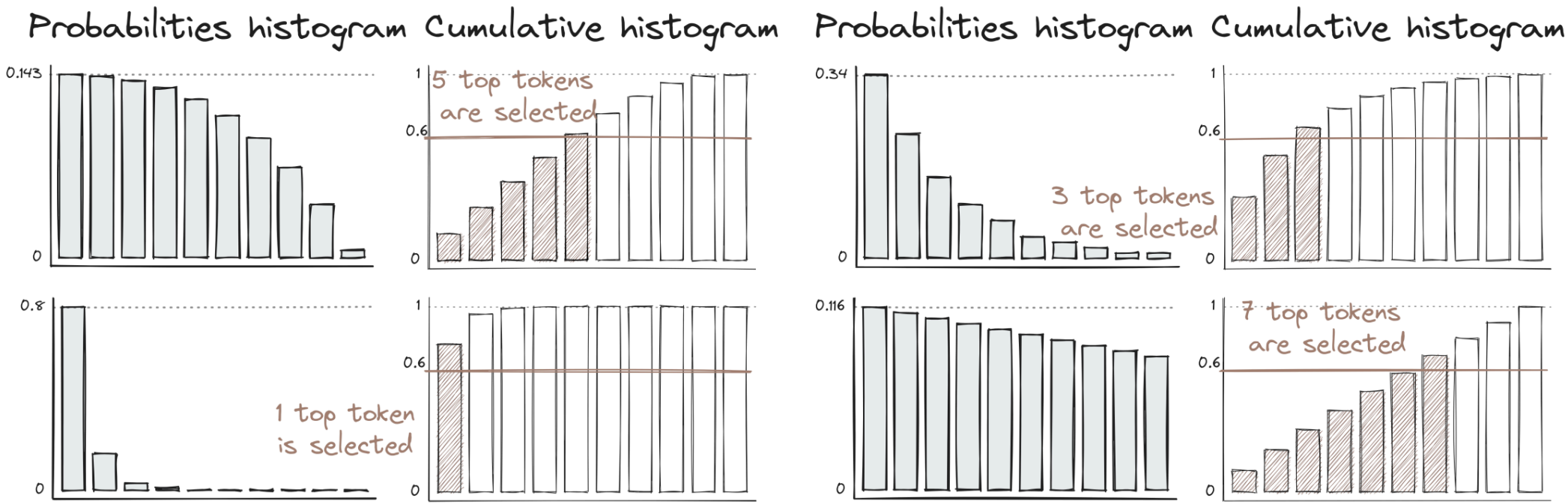
- **Description**: Limits the sampling pool to the top k highest probability tokens at each step.
- **Control**: The value of k controls the randomness; lower k means more focused and deterministic, higher k allows more diversity.



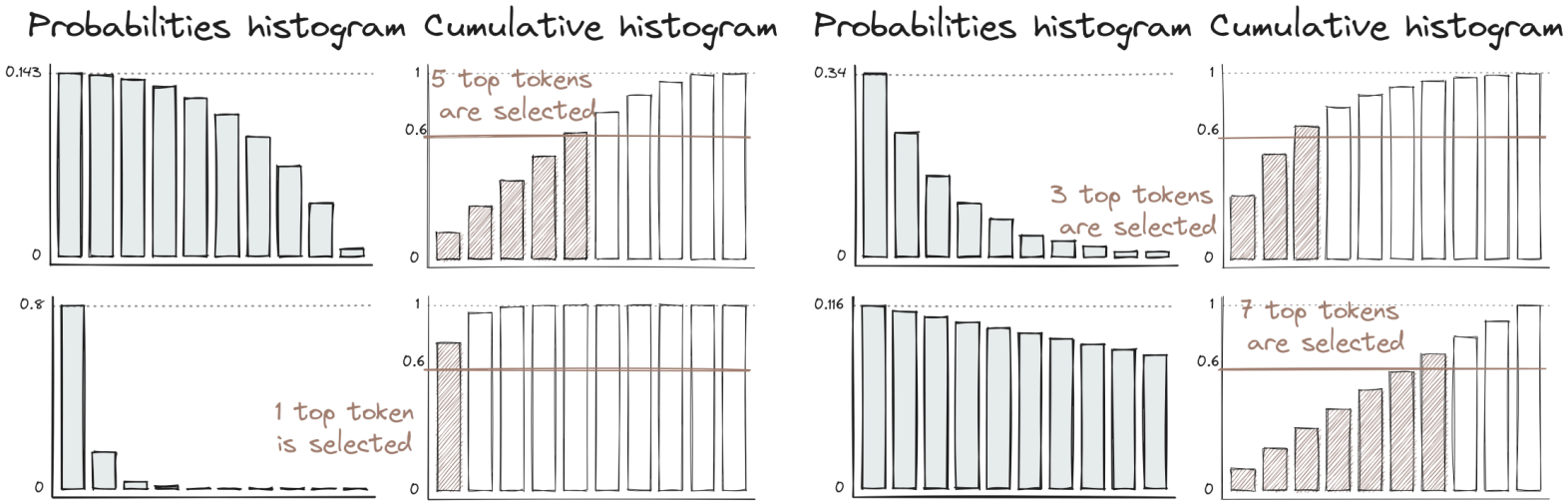
- **Advantages**:
  - Introduces diversity while avoiding extremely low-probability tokens.
  - Useful for balancing creativity and coherence.
- **Disadvantages**:
  - May still produce suboptimal sequences if the top k tokens are not ideal.
- **Example**:

```
output = model.generate(inputs, max_length=50, do_sample=True, top_k=50)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Top-p (Nucleus) Sampling



- **Description**: Selects tokens from the smallest set of tokens whose cumulative probability exceeds a threshold p.
- **Control**: The value of p determines the diversity; lower p results in more focused outputs, higher p allows more randomness.



- **Advantages**:
  - Dynamically adjusts the sampling pool based on the distribution's shape.
  - Balances creativity and reliability, particularly effective for generating coherent and diverse text.
- **Disadvantages**:
  - Can still lead to less optimal sequences if the cumulative probability includes less appropriate tokens.
- **Example**:

```
output = model.generate(inputs, max_length=50, do_sample=True, top_p=0.9)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

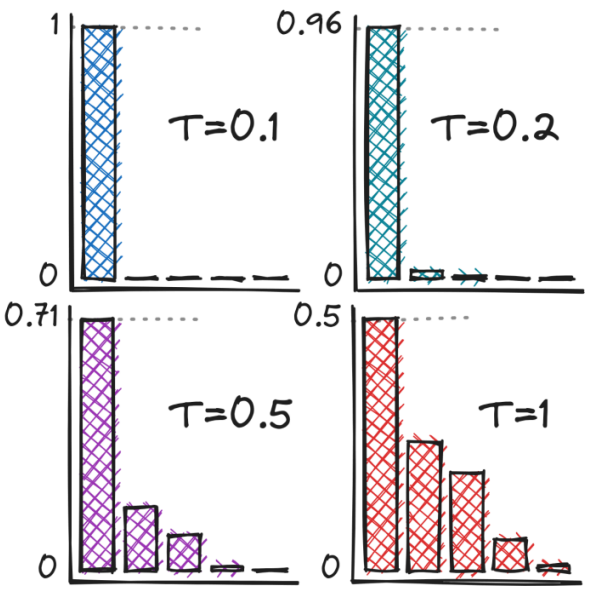
Output Control

Temperature Scaling

	logits
token A	-1.806
token B	-2.499
token C	-2.786
token D	-3.885
token E	-5.494

softmax

$$softmax(x_i) = \frac{e^{x_i/T}}{\sum_i e^{x_i/T}}$$

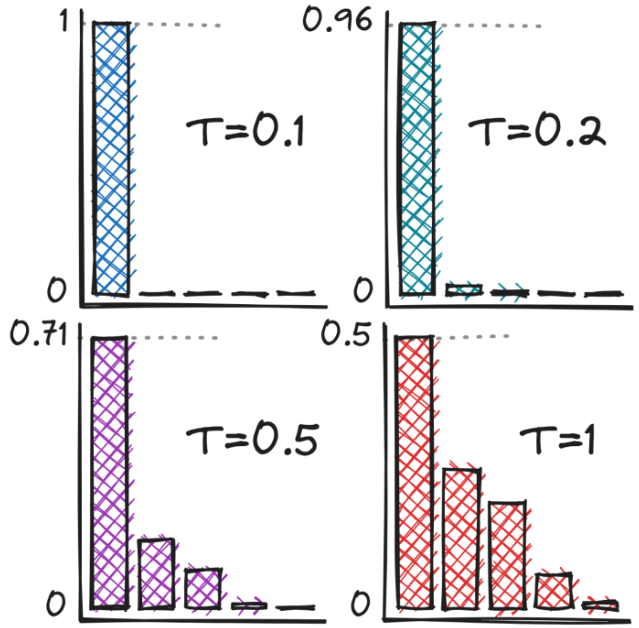


- **Description**: Modifies the logits (predicted probabilities) by dividing them by a temperature value before applying softmax.
- **Control**:
  - **Low Temperature (<1)**: Reduces randomness, making the model more confident in its predictions.
  - **High Temperature (>1)**: Increases randomness, allowing for more creative and diverse outputs.

	logits
token A	-1.806
token B	-2.499
token C	-2.786
token D	-3.885
token E	-5.494

softmax

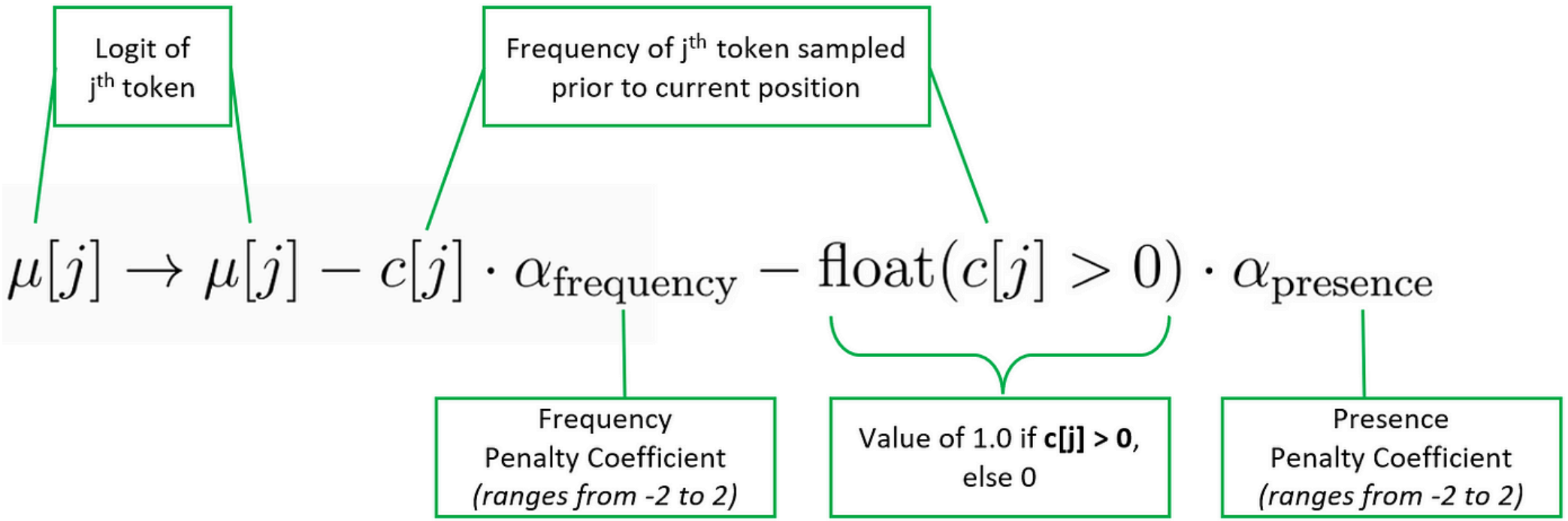
$$softmax(x_i) = \frac{e^{x_i/T}}{\sum_i e^{x_i/T}}$$



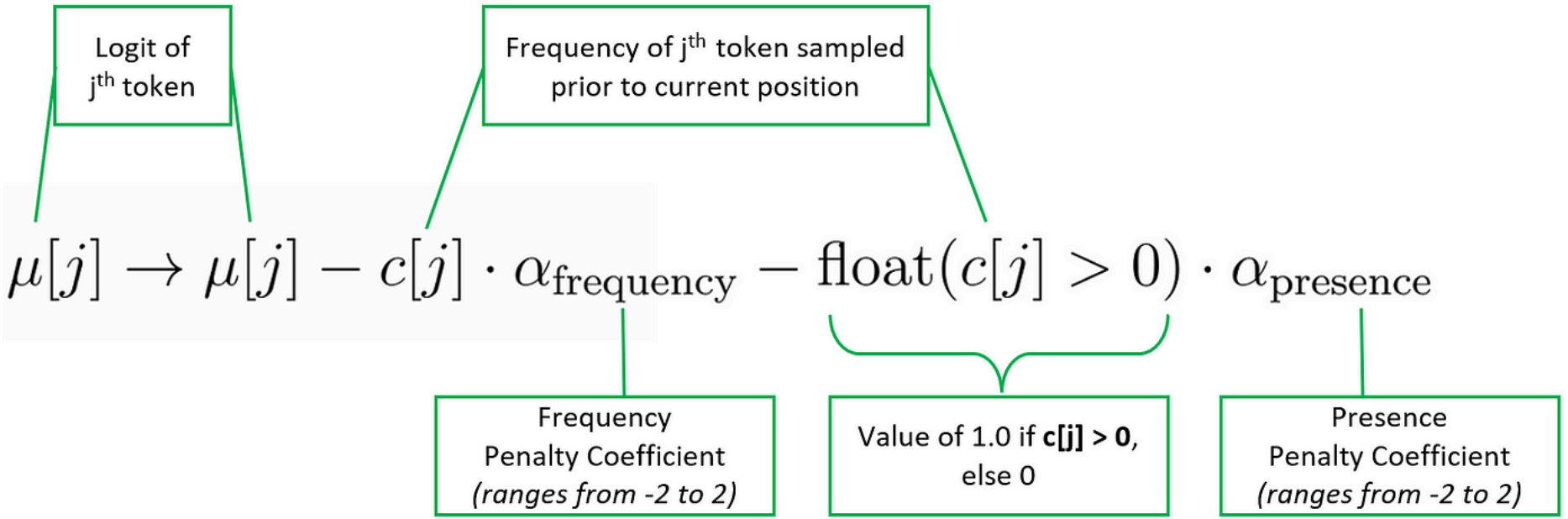
- **Advantages**:
  - Fine-tunes the balance between predictability and creativity.
- **Disadvantages**:
  - Too low can make outputs deterministic and repetitive.
  - Too high can lead to incoherent or irrelevant outputs.
- **Example**:

```
output = model.generate(inputs, max_length=50, do_sample=True, temperature=0.7)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Repetition Penalty



- **Description**: Penalizes repeated tokens by reducing their probability if they have already been generated.
- **Purpose**: Prevents the model from generating loops or repetitive sequences.



- **Advantages**:
  - Enhances diversity by avoiding repetition.
  - Useful for longer text generation tasks where repetition is a common issue.
- **Disadvantages**:
  - Can overly penalize valid repetitions (e.g., necessary repeated phrases).
- **Example**:

```
output = model.generate(inputs, max_length=50, repetition_penalty=1.2)
print(tokenizer.decode(output[0], skip_special_tokens=True))
```

Prompt Engineering

- **Definition**: The process of designing and refining prompts to optimize LLM outputs for specific tasks.
- **Objective**: Guide the model to generate desired outputs by providing informative, structured, and contextually relevant prompts.
- **Strategies**: Zero-shot, one-shot, few-shot prompting, chain-of-thought prompting, etc.

Zero-shot Prompting

- **Definition**: Providing the model with a task without any examples.
- **Example**:

```
prompt = "Summarize the following text: 'Artificial intelligence is transforming the world.'"
```

One-shot Prompting

- **Definition**: Providing one example of the task before asking the model to perform the task.
- **Example**:

```
prompt = "Example: Translate 'Hello' to French: 'Bonjour'. Now translate 'Goodbye' to French."
```

Few-shot Prompting

- **Definition**: Providing a few examples to improve model performance on specific tasks.
- **Example**:

```
prompt = """Translate the following sentences to French:
1. 'Hello' -> 'Bonjour'
2. 'Goodbye' -> 'Au revoir'
3. 'Thank you' ->"""
```

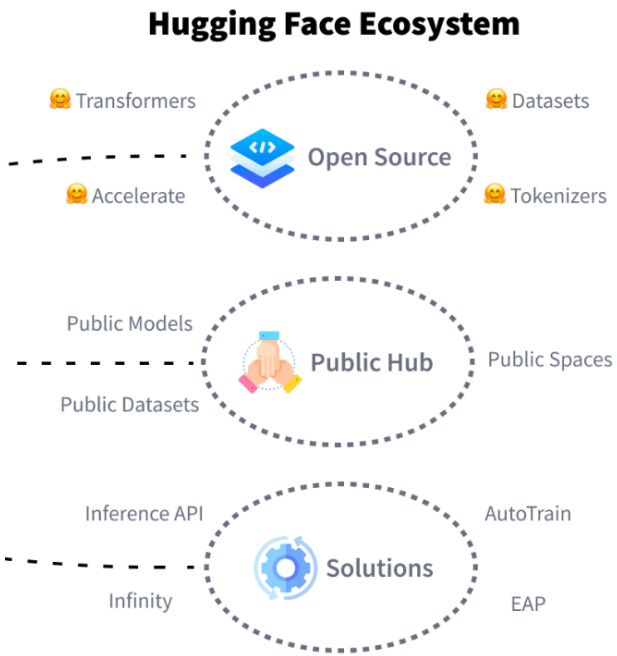
Chain-of-Thought Prompting

- **Definition**: Encouraging the model to reason step-by-step through a problem to improve logical coherence.
- **Example**:

```
prompt = """Question: If you have three apples and give away one, how many do you have left?
Let's think step by step. First, you have three apples..."""
```

Libraries for Interacting with LLMs

Hugging Face Transformers



- **Overview**: Hugging Face Transformers is a library that provides pre-trained models and tools for natural language processing tasks.
- **Models**: Access to a wide range of pre-trained language models, including GPT-2, BERT, T5, and more.
- **Scope**: Supports various NLP tasks like text generation, translation, summarization, question answering, and more.

Architecture

Why different tokenizers for different models?

- Different models may use different tokenization strategies (e.g., word-based, subword-based).
- Tokenizers are model-specific and designed to work with the corresponding model's architecture.

Demo: Text Generation

```
from transformers import pipeline

generator = pipeline('text-generation', model='gpt2')
result = generator("The future of AI is", max_length=30, num_return_sequences=1)
print(result[0]['generated_text'])
```

Demo: Sentiment Analysis

```
sentiment_analyzer = pipeline('sentiment-analysis')
result = sentiment_analyzer("I love using LLMs!")
print(result)
```

LangChain

Components

Demo: Simple Prompt Chain

```python
from langchain import PromptTemplate, LLMChain
from langchain.llms import OpenAI

llm = OpenAI(model_name="text-davinci-003")

template = PromptTemplate(input_variables=["product"], template="Write a creative ad for {product}")
chain = LLMChain(llm=llm, prompt=template)

ad_text = chain.run(product="eco-friendly water bottle")
print(ad_text)
```

Other Libraries

- **Microsoft Guidance**: library revolving around the concepts of contrained generation (e.g, selects, regular expressions, context-free grammars) and logical control (conditionals, loops, tool usage).
- **Guardrails.ai**: focused on validating model outputs against constraints, with custom retry mechanisms.
- **LLMQL**: a query language for interacting with LLMs, with support for typed prompting, control flow, constraints and tools.
- **LlamaIndex**: a library revolving around augmenting LLM applications with external data sources, such as databases, documents, or APIs.
- and many more…