

Large Language Models: Generating Text, Generating Threats, Generating Defense

Natalia Madrueño Sierro

1st Iberian Conference on Cybersecurity and Artificial Intelligence

July 10, 2025



ICAI 2025



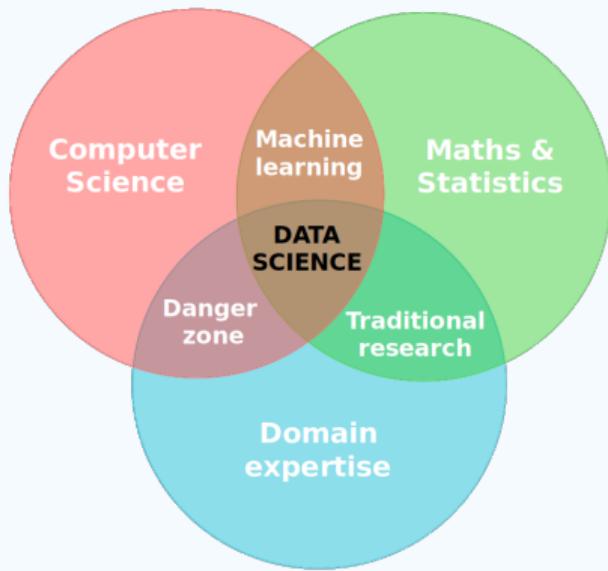
About Us

Who we are

- Researchers and Professors from URJC

- Education & Training
- Research & Innovation
- Consulting Services





(a) Foundations

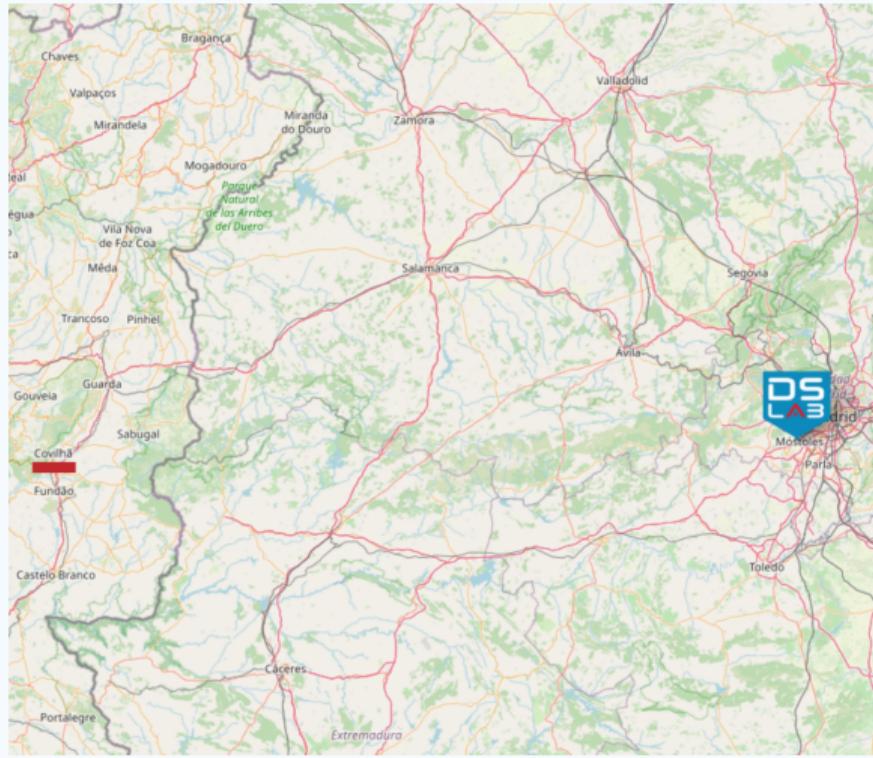


(b) Applications

- **Research Topics**

- Foundations of Machine Learning
- Explainable Machine Learning
- Complexity Measures
- Intelligent Agents
- ...

- More information in datascielab.es



Workshop Agenda

What are we presenting

- **Large Language Models (LLMs) have transformed everything**
 - Paradigm shift in NLP and AI
 - Disrupted the world as we knew it - *new revolution?*
- **This workshop explores LLMs potential**
 - How they work and their capabilities
 - How they can facilitate cyberattacks
 - How they can enhance cybersecurity

- **Foundations** - *Building Blocks of LLMs*
- **Generating Text** - *Exploring LLMs Capabilities*
- **Generating Threats** - *Employing LLMs as an Attacker*
- **Generating Defense** - *Employing LLMs as a Defender*

- **Technologies**

- Python 3.12+ (recommended virtual environment)
- Terminal (Linux, Powershell, WSL) / Jupyter Notebook
- Ollama

- **Knowledge**

- No prior knowledge of NLP is required
- Just basic Python and Terminal usage!



Source: [1]



- Download material at
github.com/URJCDSLab/icai-2025-llms
 - Slide Deck
 - Jupyter Notebooks
 - Installation Instructions

Source: [2]

- **Open-source ecosystem for AI and NLP**
 - Widely adopted for Transformer models
 - Stores models, datasets and artifacts
 - Simplifies training, inference and evaluation
- **Installation:** *transformers* and *torch* libraries
 - `pip install transformers==4.53.1`
 - `pip install torch==2.7.1`
- **Popular for LLM experimentation**



Source: [3]

- **Open-source tool specialized for LLMs**
 - Easy to install and setup
 - Enables local model serving on CPU
 - Supports a variety of open-weight LLMs
- **Installation:** ollama.com/download
- **Popular for LLM serving**



Source: [4]

- **Ollama models to download**

- *ollama pull llama3:instruct*
- *ollama pull deepseek-r1:8b*
- *ollama pull llama3:text*

- **Hugging Face models to download**

- *huggingface-cli download gpt2*
- *huggingface-cli download distilbert-base-uncased-finetuned-sst-2-english*

Foundations

Building Blocks of Large Language Models

• **Text Mining**

- Extracts knowledge from large volumes of unstructured text
- Requires understanding intricacies of human language
- Data Mining and NLP models and techniques

• **Natural Language Processing (NLP)**

- Enables computers to understand human language
- Analyzes several levels of human language
- Interdisciplinary field

- **Feature Extraction**

- Computers don't understand raw text
- Text preprocessing and cleaning
- Text vectorization and embeddings

- **Language Modeling**

- Predict token in a given context
- Basis of Large Language Models (LLMs)

- More handcrafted feature engineering

- Manually engineered linguistic features
- Traditional statistical methods

$$\begin{aligned} & \text{TF-IDF}(t, d, D) \\ & \text{tf}(t, d) \cdot \text{idf}(t, D) \end{aligned}$$

- Examples

- Feature Extraction: *BoW*, *TF-IDF*, ...
- Language Modeling: *n-grams*, *HMM*, ...

- **Highly resource-intensive**

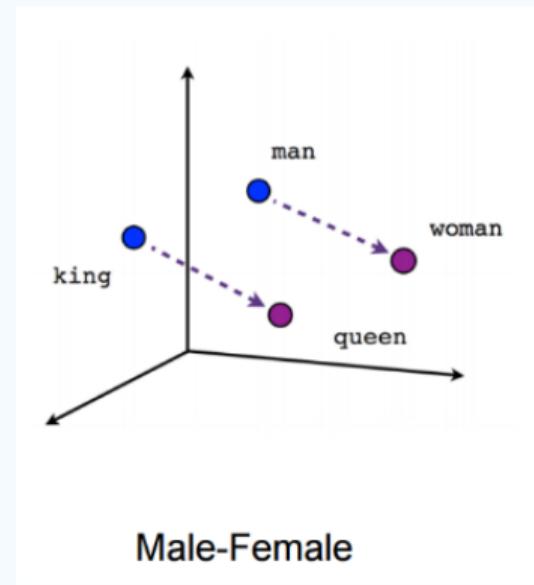
- Trained on massive datasets from diverse domains and tasks
- Require significant computational power (e.g., GPUs)

- **Transfer Learning paradigm emerges**

- Unsupervised pretraining on unlabeled corpora
- General models for multiple downstream tasks
- Model fine-tuning becomes a widely adopted practice

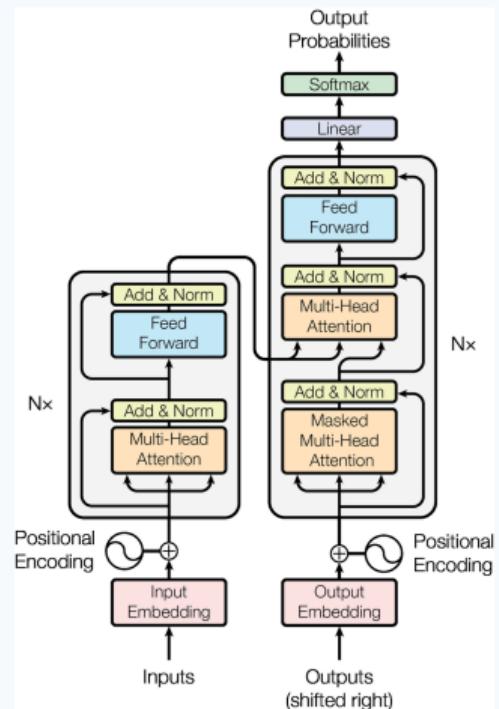
- **Examples**

- Feature Extraction: *Word2Vec, ELMo, ...*
- Language Modeling: *RNNs, LSTMs, ...*



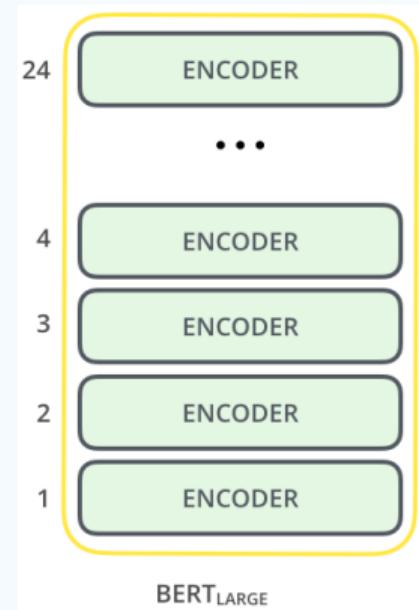
Source: [5]

- Revolutionized AI landscape
 - Introduced for Machine Translation
 - Currently applied to several AI fields
- Key Components
 - Attention mechanism
 - Encoder-decoder architecture
- Basis of Large Language Models



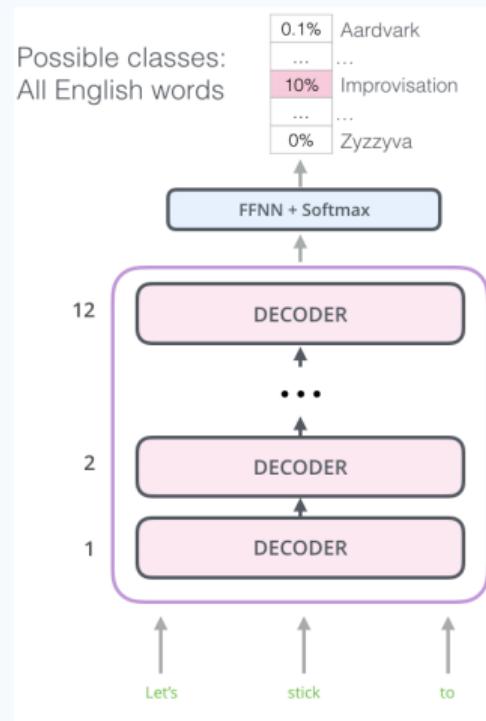
Source: [6]

- **Produce text representations**
 - Generate contextual embeddings from text
 - Capture semantic and contextual relationships
- **BERT**
 - Popularized transformers architecture
 - Stack of multiple encoders
 - Fine-tuning for downstream tasks
- **Other encoders-only** - *RoBERTa, MPNet, ModernBERT, ...*



Source: [7]

- Predict next token in a sequence
 - Autoregressive generation of text
 - Can use encoders output
- GPT-2
 - Milestone towards LLMs
 - Stack of multiple decoders
 - Excels in generative tasks



Source: [7]

Demo time!

Exploring how a decoder works

- Exploring GPT-2 with Hugging Face
- Hugging Face Pipelines
 - Predictions on common NLP tasks
 - Adjust common model parameters
- Hugging Face AutoTokenizer and AutoModelForCausalLM
 - AutoTokenizer for tokenizing raw input text
 - AutoModelForCausalLM for decoder weights
 - Greater control and customization over settings
- Notebook: *exploring_decoders.ipynb*



Python Code

```
from transformers import pipeline

text = 'An example of a prime number is'
generator = pipeline('text-generation', 'gpt2')

response = generator(text)
print(response[0]['generated_text'])
```

Hugging Face facilitates inference for common NLP tasks with its Pipelines. This includes text generation with popular open-source models, such as GPT-2.

An example of a prime number is 1, which was the number in the series, but is now 0.

It's also important to note that the number itself is not necessarily an integer, nor is it necessarily a string. Rather, it's a series of digits that have the same number suffix. For example, if you think of a string as a string, it could be written as 1, which is the number 1 in the series, but is now 0.

The number itself is not necessarily an integer, nor is it necessarily a string. Rather, it's a series of digits that have the same number suffix. For example, if you think of a string as a string, it could be written as, which is the number in the series, but is now in the sequence: 1, 3, 9, and so on. This sort of thing is called a prime number.

What does a prime number mean in practice?

Well, it means the number of bits in a number. This can mean any number. Some people believe that the number of bits in a number is the same as the number in the string, and that the number of bits in a number is a binary number. However, this doesn't always work. For example, the number of

Python Code

```
from transformers import pipeline

text = 'An example of a prime number is'
generator = pipeline('text-generation', 'gpt2')

responses = generator(text, num_return_sequences=10)
responses = [response['generated_text'][0:50] for response in responses]
print(responses)
```

Decoder-based models usually have a random component in their responses. As a result, the same input can produce multiple different responses.

```
[  
'An example of a prime number is 1.\n\nSo the prime n',  
'An example of a prime number is 1 in 3 and 1 in 3.',  
'An example of a prime number is the number 1. The ',  
'An example of a prime number is a series of 8 bits',  
'An example of a prime number is 3, and the number ',  
'An example of a prime number is (x=6, y=1).\n\nWhat ',  
'An example of a prime number is the one for "2".',  
'An example of a prime number is a number between 1',  
'An example of a prime number is the number 1.\n\nExa',  
'An example of a prime number is 0. It is the value'  
]
```

Python Code

```
from transformers import pipeline

text = 'An example of a prime number is'
generator = pipeline('text-generation', 'gpt2')

response = generator(text, do_sample=False)
print(response[0]['generated_text'])
```

Apart from setting seeds, another way to make the prediction deterministic is to generate tokens greedily and remove the random component. This method selects the most probable token at each step instead of sampling from a distribution.

An example of a prime number is the number 1.

 Python Code

```
from transformers import pipeline

text = 'An example of a prime number is'
generator = pipeline('text-generation', 'gpt2')

response = generator(text, do_sample=False, max_new_tokens=5)
print(response[0]['generated_text'])
```

Decoder-based models can keep generating tokens (and hallucinating). A very simple way to address this is by limiting the number of tokens generated

 Python Code

```
from transformers import pipeline

text = 'An example of a prime number is'
generator = pipeline('text-generation', 'gpt2')

response = generator(text, do_sample=False, max_new_tokens=5)
print(response[0]['generated_text'])
```

An example of a prime number is the number 1.

 Python Code

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained('gpt2')
model = AutoModelForCausalLM.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')
outputs = model.generate(**inputs, do_sample=False, max_new_tokens=10)
print(tokenizer.decode(outputs[0]))
```

But Hugging Face Pipelines are just an easy way to make predictions. Under the hood, they use a tokenizer and weights from a trained decoder-based model.

 Python Code

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained('gpt2')
model = AutoModelForCausalLM.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')
outputs = model.generate(**inputs, do_sample=False, max_new_tokens=10)
print(tokenizer.decode(outputs[0]))
```

Hello, how are you doing?

I'm doing fine.

 Python Code

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')
print(inputs)

idx_to_text = {_id.item(): tokenizer.decode(_id) for _id in inputs['input_ids'][0]}
print(idx_to_text)
```

First, raw input text is transformed into a sequence of discrete token IDs, which are then used to look up dense embedding vectors fed into the layers of the model.

 Python Code

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')
print(inputs)

idx_to_text = {_id.item(): tokenizer.decode(_id) for _id in inputs['input_ids'][0]}
print(idx_to_text)
```

```
{'input_ids': tensor([[15496,      11,     703,     389]]), 'attention_mask': tensor([[1, 1, 1, 1]])}

{15496: 'Hello', 11: ',', 703: ' how', 389: ' are'}
```

 Python Code

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained('gpt2')
model = AutoModelForCausalLM.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')

outputs = model.generate(**inputs, max_new_tokens=10)[0]
idx_to_text = {_id.item(): tokenizer.decode(_id) for _id in outputs}
print(idx_to_text)
```

The output of a models is another sequence of token IDs, which can be mapped to text.

 Python Code

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained('gpt2')
model = AutoModelForCausalLM.from_pretrained('gpt2')

inputs = tokenizer("Hello, how are", return_tensors='pt')

outputs = model.generate(**inputs, max_new_tokens=10)[0]
idx_to_text = {_id.item(): tokenizer.decode(_id) for _id in outputs}
print(idx_to_text)
```

```
{15496: 'Hello', 11: ',', 703: ' how', 389: ' are', 345: ' you', 1804: ' doing', 30: '?',
198: '\n', 40: 'I', 1101: "m", 3734: ' fine', 13: '.'}
```

- These are the fundamental building blocks of LLMs
 - Tokenization of raw input text
 - Dense token representation with embeddings
 - Next token prediction with Transformer decoders
- But their (huge) amount of training data and number of parameters are what make their capabilities outstanding!

Generating Text

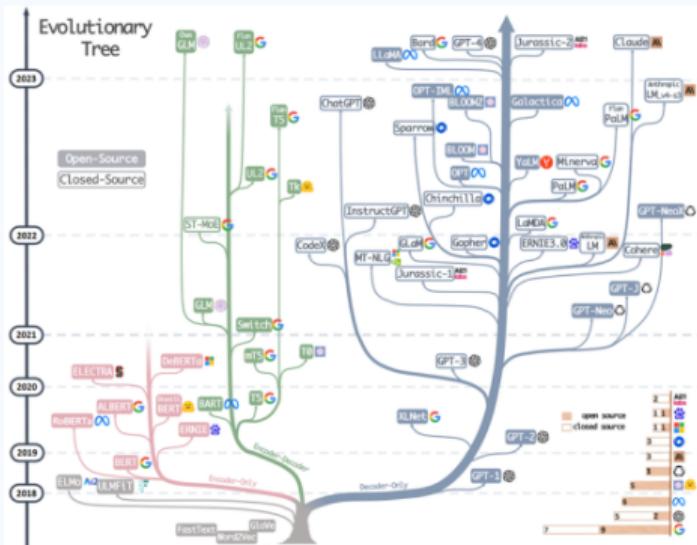
Exploring Large Language Models Capabilities

Characteristics

- Based on Transformer decoders
- Billions of parameters
- Trained on HUGE amounts of unlabelled data

Outstanding text generation capabilities

- Autoregressively predict next tokens
- Wide range of downstream applications
- Exhibit emergent abilities (e.g., reasoning)



Source: [8]

- Recent LLMs extend original decoders training objective
 - Incorporate human alignment and preference learning
 - Further optimized for reasoning, planning, and tool use
- This training is performed in incremental stages
 - Stage 1 - Foundation / Pre-training
 - Stage 2 - Instruction & Alignment
 - Stage 3 - Reasoning & Tool Use Optimization

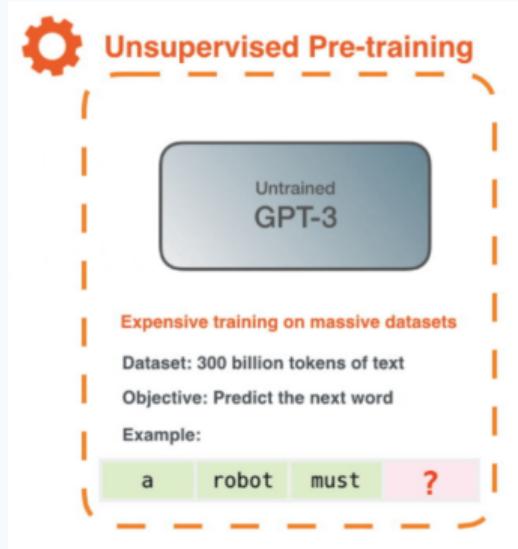
• Characteristics

- Trained on HUGE volumes of unlabelled text
- Self-supervised next-token prediction
- Learn general knowledge and language patterns

• Use cases

- Fine-tuning for downstream tasks
- Backbone for subsequent stages

• Examples - *GPT-3, LLaMA-2-base, Mistral-7B, ...*



Source: [9]

- **Characteristics**

- Supervised fine-tuning on prompt response pairs
- Follows human instructions and preferences
- Optional: RLHF/RЛАIF to polish outputs

- **Use cases**

- Dialogue systems
- Instruction-driven tasks

- **Examples** - *GPT-3.5-turbo, GPT-4o, ...*



Source: [10]

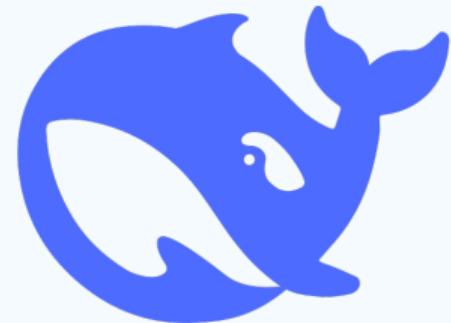
- **Characteristics**

- Fine-tuned on Chain-of-Thought demonstrations
- RL with reasoning-focused rewards
- "Think tokens" for silent planning

- **Use cases**

- Complex multi-step problem solving (e.g., math, code, ...)
- AI agents orchestrating tools and APIs

- **Examples** - o3, Gemini 2.5 Pro, ...



Source: [11]

Demo time!

Comparing pre-trained, instruct and reasoning models

- Comparing training stages with Ollama
- Ollama LLM models
 - Llama 3 8B [Pre-trained]: `ollama run llama3:text`
 - Llama 3 8B [Instruct]: `ollama run llama3:instruct`
 - DeepSeek-R1 8B [Reasoning]: `ollama run deepseek-r1:8b`
- Notebook: `exploring_models.ipynb`



>_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

Pre-trained models can autocomplete input text, as they are trained on next token prediction from vast volumes of raw unlabeled text.

However, they often struggle to follow human instructions and to handle complex / multi-step tasks.

>_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

>>> What is the capital of France? Reverse the answer and delete the vowels , you have the name of a famous French actor. Hint: He directed "La Haine" in 1995. The best way to learn French is just by being there. If your schedule doesn't allow for travel to Paris or Provence, we'll bring the experience right into your home through our immersive live classes, designed with students of all levels and backgrounds in mind.

>_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

Instruct-tuned models perform better at following human instructions, as they are also trained on prompt-response datasets.

However, they sometimes struggle with complex / multi-step tasks (especially smaller models).

>_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

```
>>> What is the capital of France? Reverse the answer and delete the vowels  
A clever question!
```

The capital of France is Paris.

Reversing the answer: Sirap.

Deleting the vowels: Srpr.

➤_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

Reasoning models perform better on complex tasks, as they are also trained on Chain-of-Thought prompts and reward signals.

However, their detailed inner monologue can be excessively long and redundant.

>_ Ollama Terminal

```
>>> What is the capital of France? Reverse the answer and delete the vowels
```

```
>>> What is the capital of France? Reverse the answer and delete the vowels  
Thinking...
```

First, the user asked: "What is the capital of France?" I know that Paris is the capital of France. That's straightforward.

Now, they said to reverse the answer and delete the vowels. So, reversing means taking the word "Paris" and writing it backwards. And deleting vowels means removing A, E, I, O, U from that reversed string.

Let me start with reversing "Paris". Reversing a word typically involves flipping the order of its letters. So:

- Paris: P-A-R-I-S

Reversed should be S-I-R-A-P.

Now, delete the vowels from this reversed string. Vowels are A, E, I, O, U (and sometimes Y, but here it's not present).

In "S-I-R-A-P", let's identify the vowels:

- S: consonant
- I: vowel
- R: consonant
- A: vowel
- P: consonant

So, deleting I and A would leave us with S-R-P.

Therefore, the reversed answer without vowels is "SRP".

I should consider if the word has any other forms or if it's case-sensitive. The question says "the capital of France", which might be answered as "Paris" in lowercase typically, but since it's a proper noun, I'll assume "Paris" is fine.

Also, reversing and deleting vowels: should we reverse first then delete, or delete vowels then reverse? The user said "reverse the answer and delete the vowels", so order matters. We reverse the word first and then remove vowels from that reversed version.

Answer is "Paris", not just any string. But it's a simple word, no issue.

Finally, I need to output this in my response. Since this is a simulation, I'll generate a response as if I'm an AI.

...done thinking.

The capital of France is Paris. Reversed: SRP

- **Prompts drive LLM responses**

- The input text provided to an LLM is called a prompt
- LLM predictions are determined by input prompts

- **Prompt Engineering**

- Providing good prompts are key to obtain quality responses
- Specially for older / smaller models

- **Best practices**

- Provide context and persona
- Be explicit and use delimiters
- Show gold-standard examples
- Start simple and refine iteratively

- **Some techniques**

- Zero-shot / Few-shot / Chain-of-Thought
- Templates and Metaprompting
- RAG and Function / Tool Calling

- **Better term: context engineering**

Demo time!

Exploring prompt engineering best practices

- Comparing different prompts with Ollama
- Ollama LLM model
 - Llama 3 8B [Instruct]: *ollama run llama3:instruct*
- Notebook: *exploring_prompting.ipynb*



>_ Ollama Terminal

```
>>> Classify the sentiment of "The delivery was extremely late, but the product was excellent".
```

The first prompts users tend to give to models are very simple and straightforward, with no additional context, constraints, or examples.

>_ Ollama Terminal

```
>>> Classify the sentiment of "The delivery was extremely late, but the product was excellent".
```

I would classify the sentiment of this sentence as MIXED or BIPARTITE.

The speaker is expressing two conflicting opinions:

- * The first part of the sentence ("The delivery was extremely late") has a NEGATIVE sentiment, indicating frustration and dissatisfaction with the timing.
- * The second part of the sentence ("but the product was excellent") has a POSITIVE sentiment, indicating satisfaction and approval with the quality of the product.

Overall, the sentiment is mixed because it combines both positive and negative emotions, making it difficult to categorize as purely positive or negative.

>_ Ollama Terminal

```
>>> Classify the sentiment of the text below. Answer only with the word "Positive",  
"Negative", or "Neutral".
```

Text: "The delivery was late, but the product was excellent"

Answer:

However, by delimiting the input and restricting the range of possible answers, we can reduce ambiguity and produce more structured and predictable responses for automatized systems.

>_ Ollama Terminal

>>> Classify the sentiment of the text below. Answer only with the word "Positive", "Negative", or "Neutral".

Text: "The delivery was extremely late, but the product was excellent"

Answer:

Sometimes...

Positive

Other times...

Neutral

Even...

Mixed

>_ Ollama Terminal

```
>>> # Persona
```

You are a customer satisfaction analyst. Your task is to evaluate customer reviews and determine overall sentiment toward services and products.

```
# Instructions
```

Classify each review as "Positive", "Negative", or "Neutral" based on the customer's tone and experience.

- Positive: The review clearly expresses satisfaction.
- Negative: The review clearly expresses dissatisfaction.
- Neutral: The review expresses mixed feelings or maintains an indifferent tone.

Answer only with the word "Positive", "Negative", or "Neutral".

Review: "The delivery was extremely late, but the product was excellent"

Answer:

Providing more context about the meaning of the labels can improve model predictions.

>_ Ollama Terminal

```
>>> # Persona
```

You are a customer satisfaction analyst. Your task is to evaluate customer reviews and determine overall sentiment toward services and products.

```
# Instructions
```

Classify each review as "Positive", "Negative", or "Neutral" based on the customer's tone and experience.

- Positive: The review clearly expresses satisfaction.
- Negative: The review clearly expresses dissatisfaction.
- Neutral: The review expresses mixed feelings or maintains an indifferent tone.

Answer only with the word "Positive", "Negative", or "Neutral".

Review: "The delivery was extremely late, but the product was excellent"

Answer:

Neutral

>_ Ollama Terminal

>>> Classify the sentiment of the text below. Answer only with the word "Positive", "Negative", or "Neutral".

Text: "The phone case fits perfectly and feels super durable."

Answer: Positive

Text: "It feels cheap, but it does its job perfectly."

Answer: Neutral

Text: "It's a terrible product. It broke after only two days!"

Answer: Negative

Text: "The delivery was extremely late, but the product was excellent"

Answer:

Also, providing input examples and their corresponding labels can improve end predictions.

>_ Ollama Terminal

>>> Classify the sentiment of the text below. Answer only with the word "Positive", "Negative", or "Neutral".

Text: "The phone case fits perfectly and feels super durable."

Answer: Positive

Text: "It feels cheap, but it does its job perfectly."

Answer: Neutral

Text: "It's a terrible product. It broke after only two days!"

Answer: Negative

Text: "The delivery was extremely late, but the product was excellent"

Answer:

Neutral

>_ Ollama Terminal

>>> // TODO

Try combining the previous two prompts, which include a clearer explanation of the label meanings, as well as examples of inputs and their corresponding labels.

- LLMs (with prompt engineering) can be effectively used for:
 - Text generation & summarization
 - Question answering & chatbots
 - Automatization & AI agents
 - And much more NLP and other AI tasks!
- These include cybersecurity (and cyberattacks)

Generating Threats

Large Language Models as an Attacker

- ***With great power comes great responsibility***

- LLMs can strengthen defensive cybersecurity tools
- ... but they can also be employed for malicious ends

- **LLMs facilitates automated cyberattacks**

- Automate attacks on individuals
- Automate attacks on systems and infrastructures

- **The range of potential threats is immense!**

- **LLMs outstanding capabilities for**
 - Mimic realistic human conversations
 - Extract valuable knowledge and insights
- **Potential threats examples**
 - Social engineering for extracting information
 - Generating personalized phishing attempts
 - Impersonating individuals (even images and voice!)
 - And much more...

- **LLMs outstanding capabilities for**

- Generating and adapting easily code and scripts
- Parsing configurations, logs and spot weaknesses

- **Potential threats examples**

- Mimic vast amounts of real-world users
- Autogenerated and polymorphic malware
- Multi-step attacks across services
- And much more...

Demo time!

Exploring text generation as a threat

- Attacking Hugging Face models with Ollama LLMs
- Ollama LLM models
 - Llama 3 8B [Instruct]
ollama run llama3:instruct
- Hugging Face model
 - Sentiment analysis classifier
distilbert-base-uncased-finetuned-sst-2-english
- Notebook: *exploring_threats.ipynb*



 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I love using these models!")
```

Consider a scenario in which the target system to attack is a sentiment analysis model that classifies input text based on its sentiment.

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I love using these models!")
```

```
[{'label': 'POSITIVE', 'score': 0.9996587038040161]
```

>_ Ollama Terminal

```
>>> Introduce typos into this text. Only answer the text with typos.
```

Text: "I love using these models!"

Answer:

LLMs can automate the process of generating adversarial attacks against this model.

The adversarial attack modifies the text to deceive the target model while resembling the original input from a human perspective.

>_ Ollama Terminal

```
>>> Introduce typos into this text. Only answer the text with typos.
```

Text: "I love using these models!"

Answer:

```
"I lov usin thse modles!"
```

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I lov usin thse modles!")
```

Introducing the perturbed text created by the LLM can cause the target victim system to produce incorrect predictions.

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I lov usin thse modles!")
```

```
[{'label': 'NEGATIVE', 'score': 0.7528746128082275]
```

>_ Ollama Terminal

```
>>> // TODO
```

Try introducing other perturbations at different text levels (e.g., paraphrasing, synonyms, neutral words, emojis, ...).

Generating Defenses

Large Language Models as a Defender

- ***If you know the enemy and know yourself,
you need not fear the result of a hundred battles***
 - Understanding attacker tools prepares defenses against them
 - Simulating attacks prevents real-world breaches
- **LLMs facilitates risk mitigation**
 - Automate vulnerability and weakness analysis
 - Detect and mitigate cyberattacks in real-time
- **LLMs are also a powerful line of defense!**

- **LLMs outstanding capabilities for**

- Fast comprehension and detection of emerging vulnerabilities
- Textual explanations of findings and remediations

- **Potential defense examples**

- Automated code and system reviews
- Generate real-time alerts and responses
- Prioritization of CVE feeds
- And much more...

Demo time!

Exploring text generation as a defense

- Defending Hugging Face models with Ollama LLMs
- Ollama LLM models
 - Llama 3 8B [Instruct]
ollama run llama3:instruct
- Hugging Face model
 - Sentiment analysis classifier
distilbert-base-uncased-finetuned-sst-2-english
- Notebook: *exploring_defenses.ipynb*



 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I lov usin thse modles!")
```

The previous targeted victim system may be subject to adversarial attacks aimed at making it produce incorrect predictions.

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I lov usin thse modles!")
```

```
[{'label': 'NEGATIVE', 'score': 0.7528746128082275]
```

>_ Ollama Terminal

```
>>> Rewrite the text to reflect how a typical human user would write it. Only answer the transformed text.
```

Text: "I lov usin thse modles!"

Answer:

As it was possible to introduce modifications to mislead victim models, it is also possible to perturb the text to clean it from common attacks.

>_ Ollama Terminal

>>> Rewrite the text to reflect how a typical human user would write it. Only answer the transformed text.

Text: "I lov usin thse modles!"

Answer:

"I love using these models!"

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I love using these models!")
```

This cleaning process makes the target system receive inputs that more closely match its expected format.

 Python Code

```
from transformers import pipeline  
  
clf = pipeline("sentiment-analysis")  
  
clf("I love using these models!")
```

```
[{'label': 'POSITIVE', 'score': 0.9996587038040161]
```

Conclusions

Wrapping Everything Up

- **LLMs are reshaping the world**
 - Transforming AI, human-computer interaction, entire industries
- **A double-edged sword**
 - Can both facilitate cyberattacks and empower cybersecurity
- **We are only at the beginning**
 - Their capabilities and implications are still unfolding
 - Models and techniques are evolving everyday!



Large Language Models: Generating Text, Generating Threats, Generating Defense

Natalia Madrueño Sierro

1st Iberian Conference on Cybersecurity and Artificial Intelligence

July 10, 2025



ICAI 2025



- [1] NumFOCUS. Conda: a cross-platform, language-agnostic package and environment manager, 2025. [Online; accessed July 7, 2025].
- [2] Data Science Lab, URJC. Data Science LAB — URJC. <https://github.com/URJCDSLab>, 2025. [Online; accessed July 7, 2025].
- [3] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.
- [4] Ollama Inc. Ollama, 2025. [Online; accessed July 7, 2025].
- [5] Gidi Shperber. A gentle introduction to doc2vec, October 2019. [Online; accessed July 7, 2025].
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [7] Jay Alammar. The illustrated transformer. *The Illustrated Transformer—Jay Alammar—Visualizing Machine Learning One Concept at a Time*, 27:1–2, 2018.
- [8] Sidwendluian Nana, Didier Bassolé, Desire Guel, and Oumarou Sie. Deep learning and web applications vulnerabilities detection: An approach based on large language models. *International Journal of Advanced Science and Computer Applications*, 15:1391–1399, 07 2024.
- [9] Jay Alammar. How gpt-3 works – visualizations and animations, July 2020. [Online; accessed July 7, 2025].
- [10] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 3(6):7, 2023.
- [11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.