



BOOTCAMP

**Python Aplicado à Análise de Dados e IA
para o Setor Elétrico Brasileiro**



Universidade Federal do
Rio Grande do Norte
IEEE Student Branch



Introdução a Agentes com LLMs

MATHEUS GOMES DINIZ ANDRADE

matheus.diniz.122@ufrn.edu.br



WHO I AM



Matheus Andrade

Bachelor of Science and Technology - **UFRN**

Computer Engineer - **UFRN**

Master's Candidate - **PPgEEC/UFRN**

Member of the **Conect2ai** Research Group | Projects: **CNPq**, **Rota2030** and
PlaforEDU

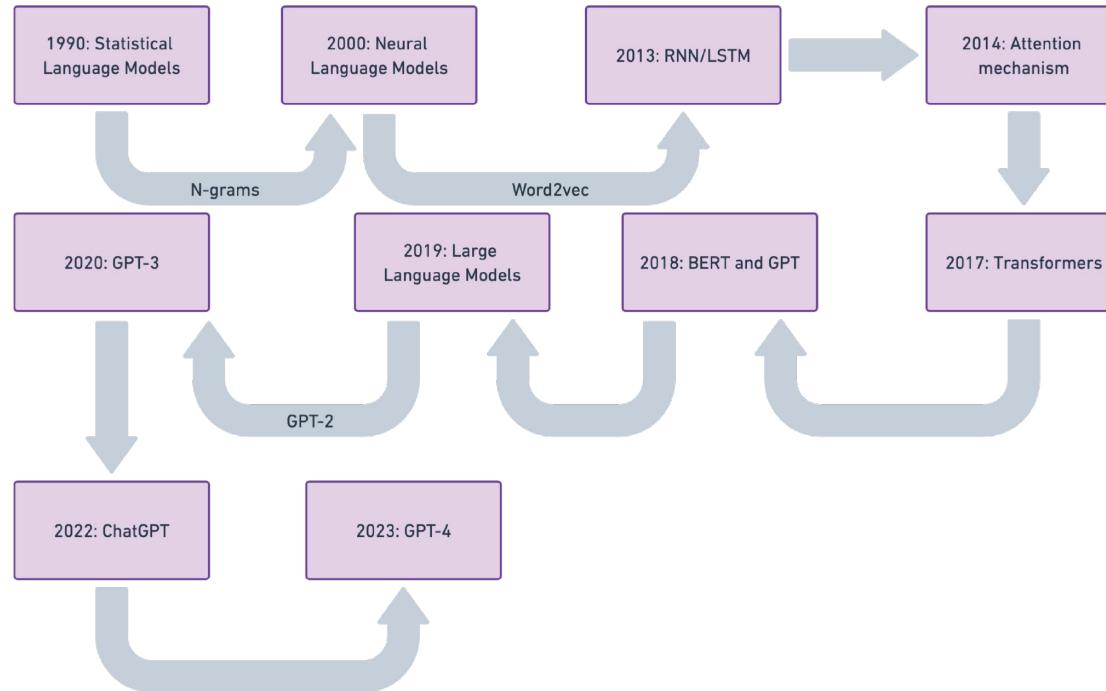
INTRODUCTION

WHAT ARE LLMS?



INTRODUCTION

A BRIEF HISTORY



INTRODUCTION

WHAT ARE LLMS?

Provided proper attribution is provided, Google hereby grants permission to reproduce the tables and figures in this paper solely for use in journalistic or scholarly works.

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

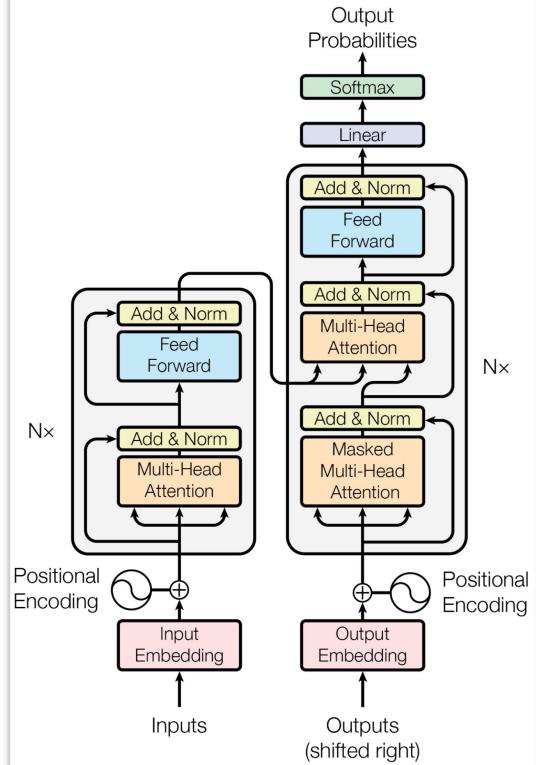
Lukasz Kaiser*
Google Brain
lukasz.kaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

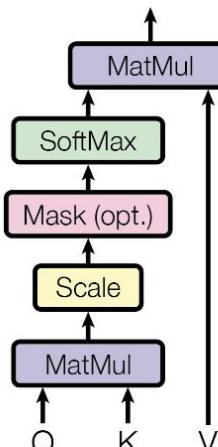
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been involved in every aspect of the work. Noam proposed scaled dot-product attention multi-head attention and the parameter-free position representation and designed the other parts involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and

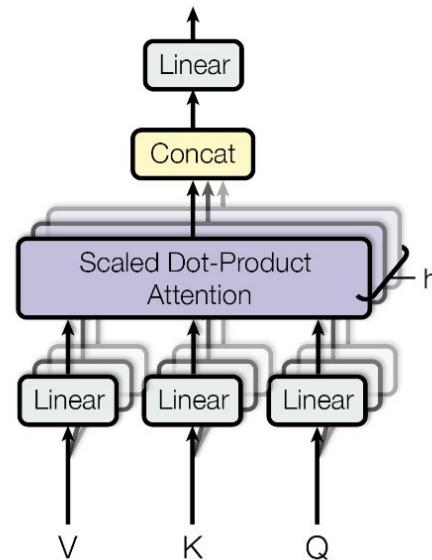


WHAT ARE LLMS?

Scaled Dot-Product Attention



Multi-Head Attention

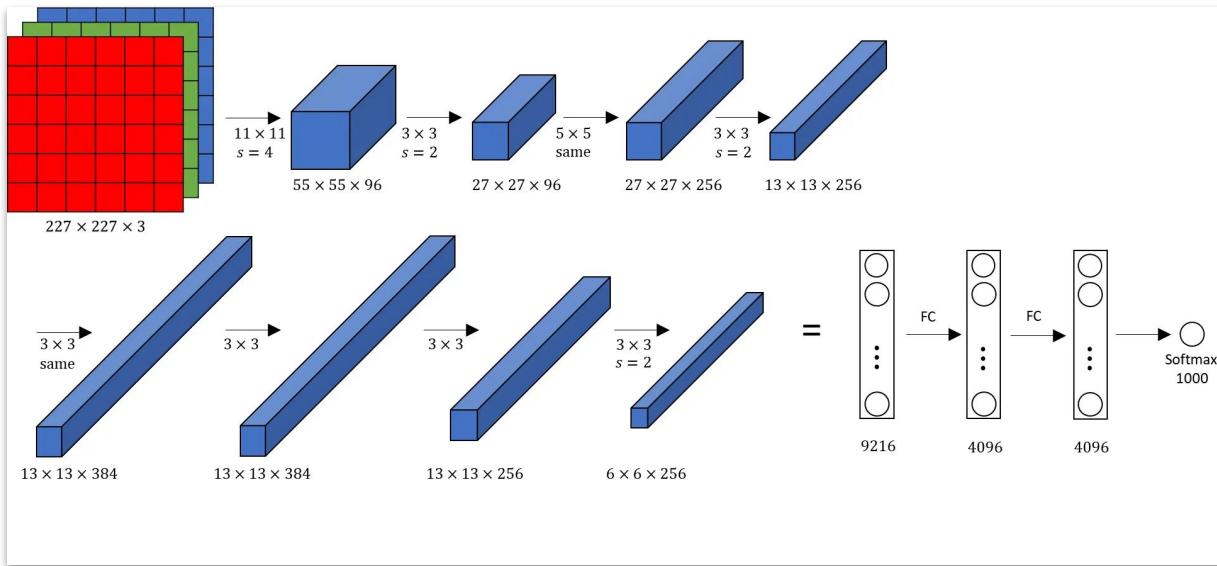


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \\ \text{where } \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

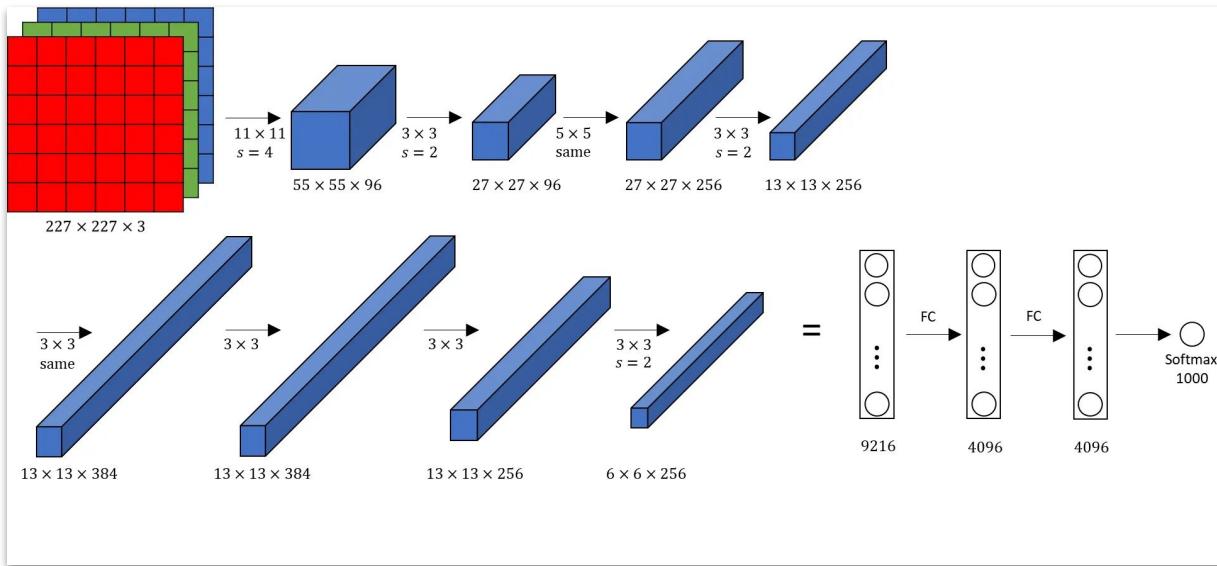
INTRODUCTION

WHAT ARE LLMS?



INTRODUCTION

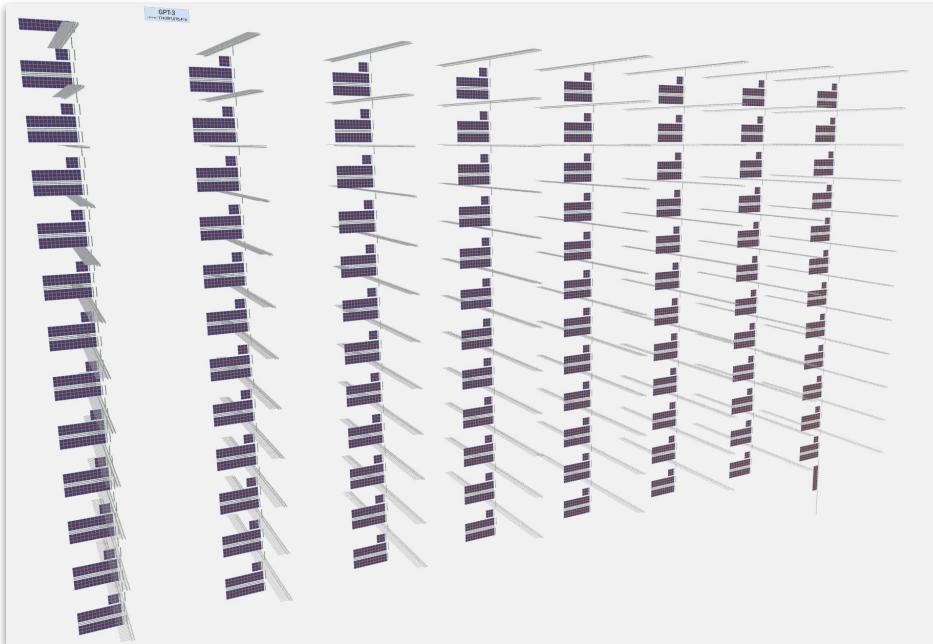
WHAT ARE LLMS?



60 Milhões

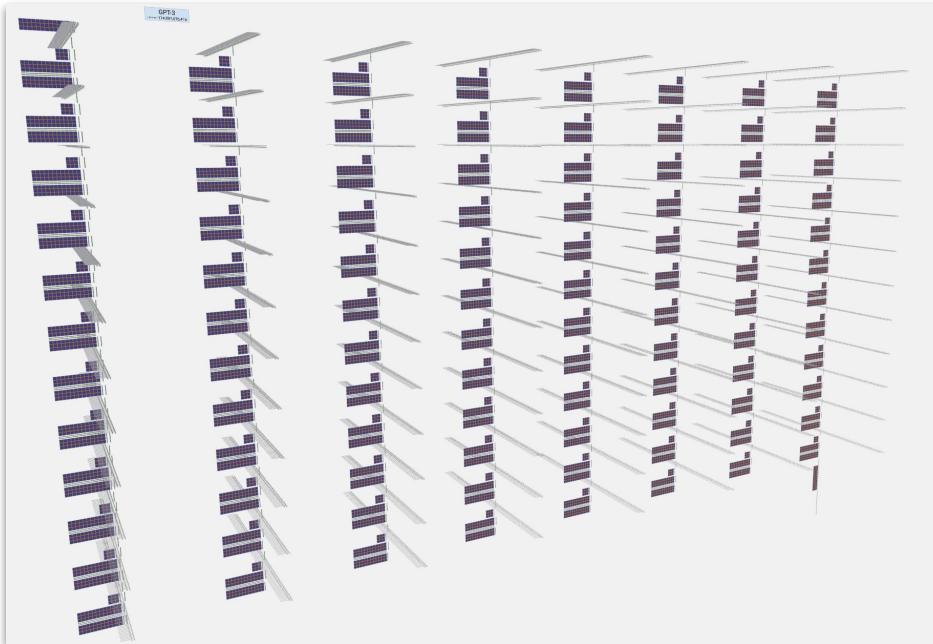
INTRODUCTION

WHAT ARE LLMS?

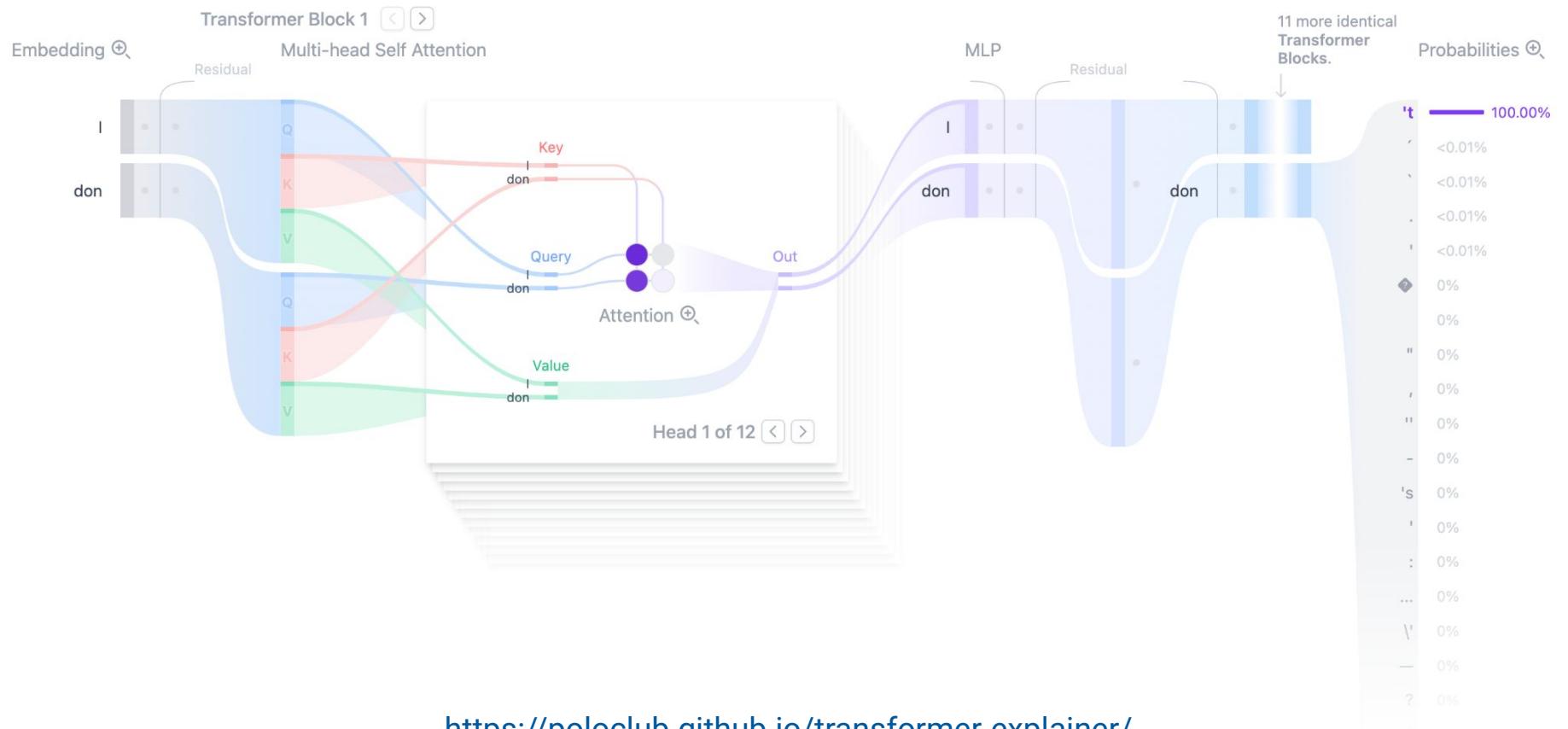


INTRODUCTION

WHAT ARE LLMS?



175 Bilhões



<https://poloclub.github.io/transformer-explainer/>

Visualize LLM Tokens

GPT2

GPT3.5 / GPT4

LLaMA

I don't know how to make LLM Agents

Tokens: 10 Characters: 35

Clear

Show example

I don 't know how to make LL M Agents

<https://tokenvisualizer.netlify.app/>

SPECIAL TOKENS

```
✖ ▶ Uncaught Error: The text contains a special token that is  
not allowed: <|endoftext|>  
    at App.tsx:35:5  
    at App.tsx:75:18  
Show ignore-listed frames
```

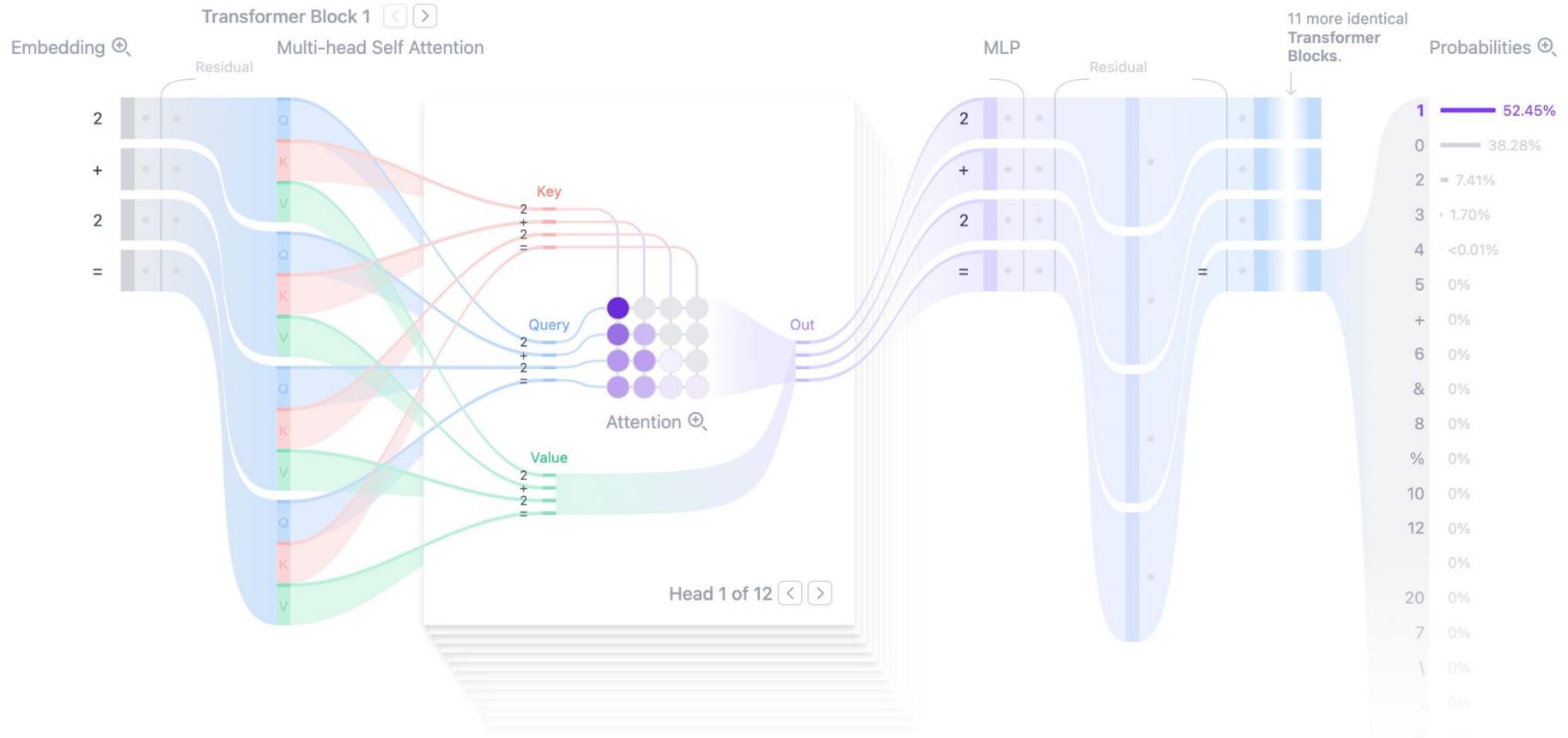


Hello how are you?

```
<|begin_of_text|>  
<|start_header_id|>user<|end_header_id|>  
  
Hello how are you?<|eot_id|>
```

- Hello! I'm just a language model, so I don't have feelings or emotions like humans do, but I'm functioning properly and ready to assist you with any questions or tasks you may have. How about you? How's your day going so far?

```
<|start_header_id|>assistant<|end_header_id|>  
  
Hello! I'm just a language model, so I don't have feelings or emotions like humans do, but I'm functioning properly and ready to assist you with any questions or tasks you may have. How about you? How's your day going so far?<|eot_id|>
```





LLM
AGENT



WHAT IS AN AGENT?

What is $2 + 2$?

sum_tool

subtract_tool

division_tool

multiplication_tool



WHAT IS AN AGENT?

What is $2 + 2$?

sum_tool

subtract_tool

division_tool

multiplication_tool



WHAT IS AN AGENT?

What is $2 + 2$?

sum_tool(2, 2)



WHAT IS AN AGENT?

What is $2 + 2$?



INTRODUCTION

PILLARS

INTRODUCTION

PILLARS

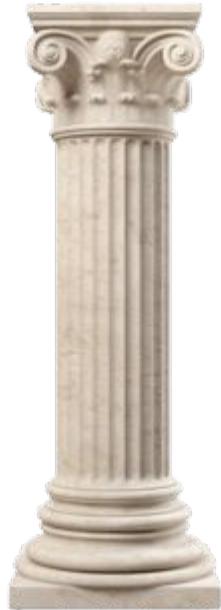
THOUGHT



INTRODUCTION

PILLARS

THOUGHT



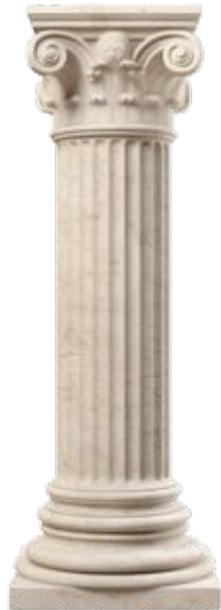
ACTION



INTRODUCTION

PILLARS

THOUGHT



ACTION



OBSERVATION



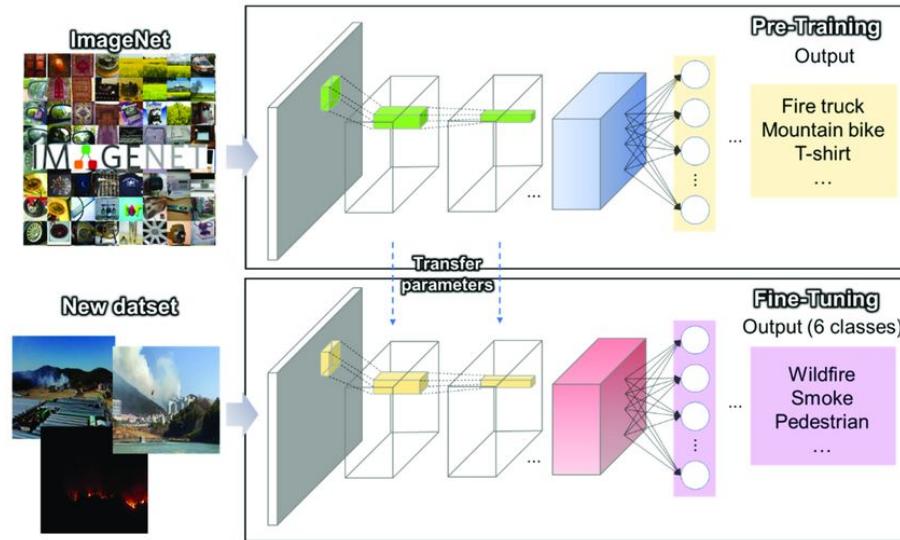
INTRODUCTION

HOW AGENTS ARE TRAINED?



INTRODUCTION

FINE-TUNING



HOW AGENTS ARE TRAINED?

Function Calling

Four yellow starburst icons of varying sizes are positioned around the word "Function Calling". One is at the top right, one is at the bottom left, one is at the top left, and one is at the middle right.

HOW AGENTS ARE TRAINED?

```
{  
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type": "function", "function": {"name": "get_stock_price", "description": "Get the current stock price of a company", "parameters": {"type": "object", "properties": {"company": {"type": "string", "description": "The name of the company"}}, "required": ["company"]}}}, {"type": "function", "function": {"name": "get_movie_details", "description": "Get details about a movie", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "The title of the movie"}}, "required": ["title"]}}}]</tools> Use the following pydantic model json schema for each tool call you will make: {"title": "FunctionCall", "type": "object", "properties": {"arguments": {"title": "Arguments", "type": "object"}, "name": {"title": "Name", "type": "string"}}, "required": ["arguments", "name"]} For each function call return a json object with function name and arguments within <tool_call></tool_call> XML tags as follows:\n<tool_call>\n{tool_call}\n</tool_call> ",  
  "role": "system"  
},
```

INTRODUCTION

HOW AGENTS ARE TRAINED?

```
{  
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type": "function", "function": {"name": "get_stock_price", "description": "Get the current stock price of a company", "parameters": {"type": "object", "properties": {"company": {"type": "string", "description": "The name of the company"}}, "required": ["company"]}}}, {"type": "function", "function": {"name": "get_movie_details", "description": "Get details about a movie", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "The title of the movie"}}, "required": ["title"]}}}]</tools> Use the following pydantic model json schema for each tool call you will make: {"title": "FunctionCall", "type": "object", "properties": {"arguments": {"title": "Arguments", "type": "object", "name": {"title": "Name", "type": "string"}, "required": ["arguments", "name"]}}} For each function call return a json object with function name and arguments within <tool_call></tool_call> XML tags as follows:\n<tool_call>\n{tool_call}\n</tool_call> ",  
  "role": "system"  
},  
{  
  "content": "Hi, can you tell me the current stock price of Apple?",  
  "role": "human"  
},
```

HOW AGENTS ARE TRAINED?

```
{  
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more  
  functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type':  
    'function', 'function': {'name': 'get_stock_price', 'description': 'Get the current stock price of a company', 'parameters': {'type': 'object',  
      'properties': {'company': {'type': 'string', 'description': 'The name of the company'}}}, 'required': ['company']}]}}, {'type': 'function', 'function':  
    {'name': 'get_movie_details', 'description': 'Get details about a movie', 'parameters': {'type': 'object', 'properties': {'title': {'type': 'string',  
      'description': 'The title of the movie'}}, 'required': ['title']}]}] </tools> Use the following pydantic model json schema for each tool call you will  
  make: {'title': 'FunctionCall', 'type': 'object', 'properties': {'arguments': {'title': 'Arguments', 'type': 'object'}, 'name': {'title': 'Name', 'type':  
      'string'}}, 'required': ['arguments', 'name']} For each function call return a json object with function name and arguments within <tool_call></tool_call>  
  XML tags as follows:<\n<tool_call>\n{tool_call}\n</tool_call>\n>\",  
  "role": "system"  
},  
{  
  "content": "Hi, can you tell me the current stock price of Apple?",  
  "role": "human"  
},  
{  
  "content": "<think>Alright, the user asked for the current stock price of Apple. Looking at the tools available, there's a function called  
  get_stock_price which exactly fits this request. I'll need to call this function with the parameter company set to 'Apple' to retrieve the information  
  the user is asking for. This seems like the most straightforward and appropriate action to take.\n</think><tool_call>\n{'name': 'get_stock_price',  
  'arguments': {'company': 'Apple'}}\n</tool_call>",  
  "role": "model"  
},
```

HOW AGENTS ARE TRAINED?

```
{
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type": "function", "function": {"name": "get_stock_price", "description": "Get the current stock price of a company", "parameters": {"type": "object", "properties": {"company": {"type": "string", "description": "The name of the company"}}, "required": ["company"]}}}, {"type": "function", "function": {"name": "get_movie_details", "description": "Get details about a movie", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "The title of the movie"}}, "required": ["title"]}}}]</tools> Use the following pydantic model json schema for each tool call you will make: {"title": "FunctionCall", "type": "object", "properties": {"arguments": {"title": "Arguments", "type": "object", "name": {"title": "Name", "type": "string"}, "required": ["arguments", "name"]}}} For each function call return a json object with function name and arguments within <tool_call></tool_call> XML tags as follows:<\n<tool_call>\n<{tool_call}>\n</tool_call>>",
  "role": "system"
},
{
  "content": "Hi, can you tell me the current stock price of Apple?",
  "role": "human"
},
{
  "content": "<think>Alright, the user asked for the current stock price of Apple. Looking at the tools available, there's a function called get_stock_price which exactly fits this request. I'll need to call this function with the parameter company set to 'Apple' to retrieve the information the user is asking for. This seems like the most straightforward and appropriate action to take.</think><tool_call>\n{"name": "get_stock_price", "arguments": {"company": "Apple"} }\n</tool_call>",
  "role": "model"
},
{
  "content": "<tool_response>\n{"stock_price": "$150.75"}\n</tool_response>",
  "role": "tool"
}
}
```

HOW AGENTS ARE TRAINED?

```
{
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type": "function", "function": {"name": "get_stock_price", "description": "Get the current stock price of a company", "parameters": {"type": "object", "properties": {"company": {"type": "string", "description": "The name of the company"}}, "required": ["company"]}}}, {"type": "function", "function": {"name": "get_movie_details", "description": "Get details about a movie", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "The title of the movie"}}, "required": ["title"]}}}]</tools> Use the following pydantic model json schema for each tool call you will make: {"title": "FunctionCall", "type": "object", "properties": {"arguments": {"title": "Arguments", "type": "object", "name": {"title": "Name", "type": "string"}, "required": ["arguments", "name"]}}} For each function call return a json object with function name and arguments within <tool_call></tool_call> XML tags as follows:<\n<tool_call>\n<{tool_call}>\n</tool_call>>",
  "role": "system"
},
{
  "content": "Hi, can you tell me the current stock price of Apple?",
  "role": "human"
},
{
  "content": "<think>Alright, the user asked for the current stock price of Apple. Looking at the tools available, there's a function called get_stock_price which exactly fits this request. I'll need to call this function with the parameter company set to 'Apple' to retrieve the information the user is asking for. This seems like the most straightforward and appropriate action to take.</think><tool_call>\n{"name": "get_stock_price", "arguments": {"company": "Apple"} }\n</tool_call>",
  "role": "model"
},
{
  "content": "<tool_response>\n{"stock_price": "$150.75"}\n</tool_response>",
  "role": "tool"
},
{
  "content": "The current stock price of Apple is $150.75.",
  "role": "model"
}
}
```

INTRODUCTION

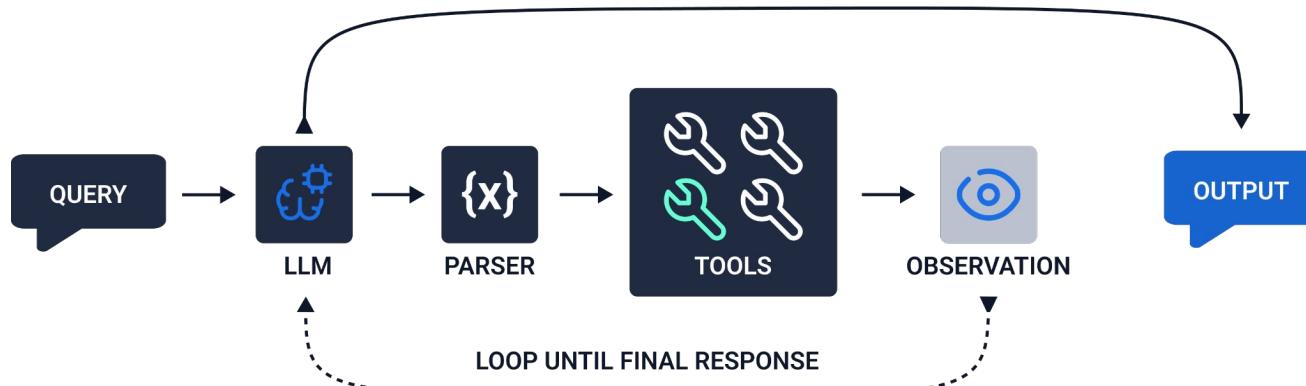
HOW AGENTS ARE TRAINED?

```
{  
  "content": "You are a function calling AI model. You are provided with function signatures within <tools></tools> XML tags. You may call one or more functions to assist with the user query. Don't make assumptions about what values to plug into functions. Here are the available tools:<tools> [{"type": "function", "function": {"name": "get_stock_price", "description": "Get the current stock price of a company", "parameters": {"type": "object", "properties": {"company": {"type": "string", "description": "The name of the company"}}, "required": ["company"]}}}, {"type": "function", "function": {"name": "get_movie_details", "description": "Get details about a movie", "parameters": {"type": "object", "properties": {"title": {"type": "string", "description": "The title of the movie"}}, "required": ["title"]}}}]</tools> Use the following pydantic model json schema for each tool call you will make: {"title": "FunctionCall", "type": "object", "properties": {"arguments": {"title": "Arguments", "type": "object", "name": {"title": "Name", "type": "string"}, "required": ["arguments", "name"]}}} For each function call return a json object with function name and arguments within <tool_call></tool_call> XML tags as follows:<\n<tool_call>\n<{tool_call}>\n</tool_call>>\",  
  "role": "system"  
},  
{  
  "content": "Hi, can you tell me the current stock price of Apple?",  
  "role": "human"  
},  
{  
  "content": "<think>Alright, the user asked for the current stock price of Apple. Looking at the tools available, there's a function called get_stock_price which exactly fits this request. I'll need to call this function with the parameter company set to 'Apple' to retrieve the information the user is asking for. This seems like the most straightforward and appropriate action to take.\n</think><tool_call>\n{"name": "get_stock_price", "arguments": {"company": "Apple"} }\n</tool_call>",  
  "role": "model"  
},  
{  
  "content": "<tool_response>\n{"stock_price": \"$150.75"}\n</tool_response>",  
  "role": "tool"  
},  
{  
  "content": "The current stock price of Apple is $150.75.",  
  "role": "model"  
},
```

3570 rows

INTRODUCTION

HOW IT WORKS?



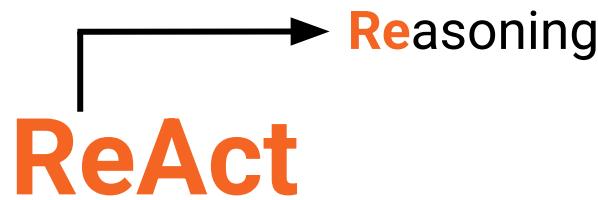
INTRODUCTION

REACT APPROACH

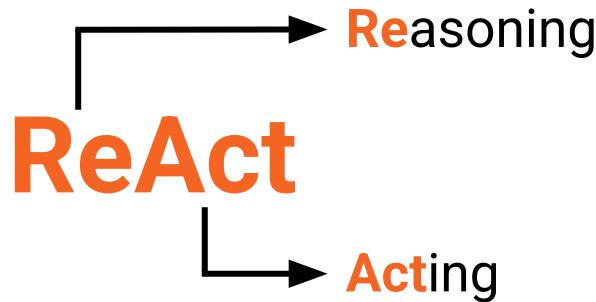
ReAct

INTRODUCTION

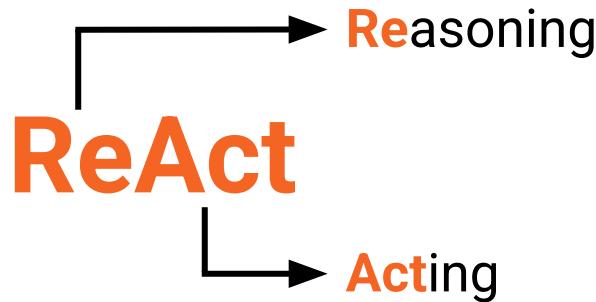
REACT APPROACH



REACT APPROACH



REACT APPROACH



It's a prompting technique that encourages the model to think **step-by-step** and **interleave** actions between reasoning steps.



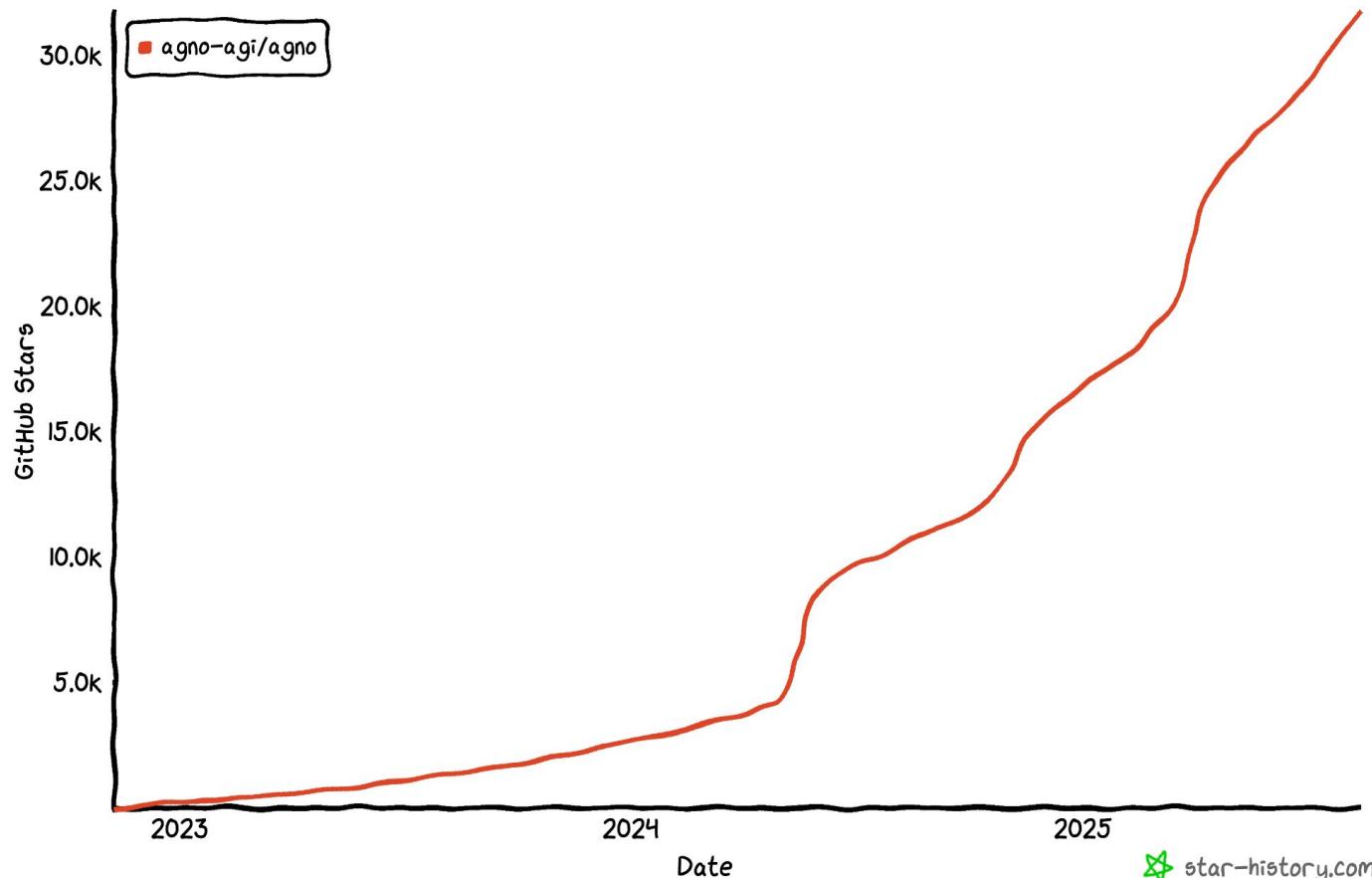
Hugging Face



HANDS-ON

#1

Star History





The logo consists of the word "Gemini" in a bold, sans-serif font. The letters are colored in a gradient: blue for 'G', transitioning through purple for 'e', 'm', and 'i', and ending in red for the final 'i'. A small, four-pointed purple star or sparkles icon is positioned above the letter 'i'.

Gemini

The screenshot shows the Google AI for Developers website. At the top, there's a navigation bar with links for Google AI for Developers, Models, Solutions, Code assistance, Showcase, and Community. On the right side of the header are search, language selection (Português), and other settings icons. A banner at the top left says "NEW Veo 3 is now available in the Gemini API! >". The main feature is a large white text "AI for every developer" on a dark background. Below it is a subtext: "Unlock AI models to build innovative apps and transform development workflows with tools across platforms." A blue button with white text "Explore models in Google AI Studio" has a red arrow pointing to it from the left. To the right, there's a screenshot of the Google AI Studio interface showing an audio recording segment with a play button and a progress bar (0:00 / 1:49). The interface includes sections for "USER" (instructions to split audio into segments) and "MODELS" (listing Gemini 2.5 Pro, Gemini 2.5 Flash, Gemini 2.0 Flash, Veo 2, and Imagen 3). A "Running..." status message and a "Stop" button are also visible. At the bottom right of the main area, the text "Start building" is displayed.

<https://ai.google.dev/>

The screenshot shows the Google AI Studio interface. On the left, there's a sidebar with options like Chat, Stream, Generate media, Build, History, and Enable saving. The main area has a "Chat prompt" section with a text input field containing "Generate Python code for a simple calculator app →". To the right of the input is a "Run" button with a plus sign and a "Run" button with a refresh icon. Below this is a "What's new" section with four cards: "URL context tool" (Fetch information from web links), "Native speech generation" (Generate high quality text to speech with Gemini), "Live audio-to-audio dialog" (Try Gemini's natural, real-time dialog with audio and video inputs), and "Native image generation" (Interleaved text-and-image generation with Gemini 2.0 Flash). A red arrow points to the "Get API key" button at the top right of the interface.

Google AI Studio

Chat prompt

Get API key

Studio Dashboard Documentation

Run settings

Gemini 2.5 Pro

Token count 0 / 1.048.576

Temperature 1

Media resolution Default

Thinking

Thinking mode

Set thinking budget

Tools

Structured output

Code execution

Function calling

Grounding with Google Search
Source: Google Search

URL context

Advanced settings

What's new

- URL context tool Fetch information from web links
- Native speech generation Generate high quality text to speech with Gemini
- Live audio-to-audio dialog Try Gemini's natural, real-time dialog with audio and video inputs
- Native image generation Interleaved text-and-image generation with Gemini 2.0 Flash

Google AI models may make mistakes, so double-check outputs.

Google AI Studio

Get API key Studio Dashboard Documentation ☰

API keys Usage & Billing Changelog

Chaves de API

Testar rapidamente a API Gemini

Guia de início rápido da API

```
<> Code ↴ ↵ ⌂ ⌄ ⌅
```

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent" \
-H 'Content-Type: application/json' \
-H 'X-goog-api-key: GEMINI_API_KEY' \
-X POST \
-d '{
  "contents": [
    {
      "parts": [
        {
          "text": "Explain how AI works in a few words"
        }
      ]
    }
}'
```

Suas chaves de API estão listadas abaixo. Também é possível visualizar e gerenciar seu projeto e as chaves de API no Google Cloud.

Look up API Key for project

Número do projeto	Nome do projeto	Chave de API	Criação	Plano
Crie uma chave de API para ver seus projetos				

View status

Use as chaves de API com segurança. Não as compartilhe nem as incorpore em códigos públicos. O uso da API Gemini em um projeto com faturamento ativado está sujeito aos [preços do modelo de pagamento por uso](#).

← →

Google AI Studio

Get API key Studio Dashboard Documentation ☰

Chaves de API

+ Criar chave de API

Testar rapidamente a API Gemini

Guia de Início rápido da API

Code

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-f16gb:generateContent" \
-H "Content-Type: application/json" \
-H "X-gcp-api-key: GEMINI_API_KEY" \
-X POST \
-d "{\n  \"contents\": {\n    \"parts\": [\n      {\n        \"text\": \"Explain how AI works\"\n      }\n    ]\n  }\n}"
```

Criar chave de API

Selecione um dos seus projetos atuais do Google Cloud

Search Google Cloud projects

Q

↳ Criar uma chave de API em um projeto atual

Look up API Key for project

Número do projeto	Nome do projeto	Chave de API	criação	Promo

Crie uma chave de API para ver seus projetos

[View status](#)

Use as chaves de API com segurança. Não as compartilhe nem as incorpore em códigos públicos. O uso da API Gemini em um projeto com faturamento ativado está sujeito aos [preços do modelo de pagamento por uso](#).

Google AI Studio

Get API key Studio Dashboard Documentation ☰

Chaves de API

+ Criar chave de API

Testar rapidamente a API Gemini

Guia de Início rápido da API

Code

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.8-f1ash:generateContent" \
-H "Content-Type: application/json" \
-H "X-goog-api-key: GEMINI_API_KEY" \
-X POST \
-d "{\n  \"contents\": {\n    \"parts\": [\n      {\n        \"text\": \"Explain how AI works\"\n      }\n    ]\n  }\n}"
```

Criar chave de API

Selecione um dos seus projetos atuais do Google Cloud

Search Google Cloud projects

|

Gemini API gen-lang-client-0581720549

Suas chaves de API estão listadas abaixo. Também é possível visualizar e gerenciar seu projeto e as chaves de API no Google Cloud.

Look up API Key for project

Número do projeto	Nome do projeto	Chave de API	criação	Promo

Crie uma chave de API para ver seus projetos

View status

Use as chaves de API com segurança. Não as compartilhe nem as incorpore em códigos públicos. O uso da API Gemini em um projeto com faturamento ativado está sujeito aos [preços do modelo de pagamento por uso](#).

Google AI Studio

Get API key Studio Dashboard Documentation ☰

Chaves de API

+ Criar chave de API

Testar rapidamente a API Gemini

Guia de Início rápido da API

Code

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.8-Flash:generateContent" \
-H "Content-Type: application/json" \
-H "X-goog-api-key: GEMINI_API_KEY" \
-X POST \
-d "{\n  \"contents\": {\n    \"parts\": [\n      {\n        \"text\": \"Explain how AI works\"\n      }\n    ]\n  }\n}"
```

Criar chave de API

Selecione um dos seus projetos atuais do Google Cloud

Search Google Cloud projects

Q Gemini API (gen-lang-client-0581720549)

↳ Criar uma chave de API em um projeto atual

Suas chaves de API estão listadas abaixo. Também é possível visualizar e gerenciar seu projeto e as chaves de API no Google Cloud.

Look up API Key for project

Número do projeto	Nome do projeto	Chave de API	criação	Promo

Crie uma chave de API para ver seus projetos

View status

Use as chaves de API com segurança. Não as compartilhe nem as incorpore em códigos públicos. O uso da API Gemini em um projeto com faturamento ativado está sujeito aos [preços do modelo de pagamento por uso](#).

Google AI Studio

Get API key Studio Dashboard Documentation ☰

Chaves de API

+ Criar chave de API

Testar rapidamente a API Gemini

Guia de Início rápido da API

Code

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent" \
-H "Content-Type: application/json" \
-H "X-goog-api-key: GEMINI_API_KEY" \
-X POST \
-d '{ \
  "contents": { \
    "parts": [ \
      { \
        "text": "B" \
      } \
    ] \
  } \
}'
```

Chave de API gerada

Use suas chaves de API com segurança. Não compartilhe nem as incorpore em códigos que o público possa ver.

AlzaSyAMaONxG9FojqVGFnj9X3T9_pv-vzErmTE 

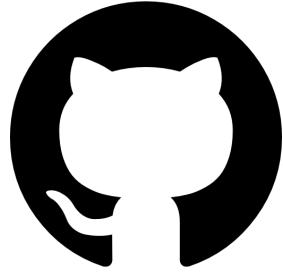
Suas chaves de API estão listadas abaixo. Também é possível visualizar e gerenciar seu projeto e as chaves de API no Google Cloud.

Look up API Key for project

Número do projeto	Nome do projeto	Chave de API	criação	Promoção
1456	Gemini API ☰	...mTE	25 de ago. de 2025	Configurar o faturamento Ver detalhes de uso ☰

[View status](#)

Use as chaves de API com segurança. Não as compartilhe nem as incorpore em códigos públicos. O uso da API Gemini em um projeto com faturamento ativado está sujeito aos [preços do modelo de pagamento por uso](#).



<https://github.com/conect2ai/Agents-Bootcamp>
01_HandsOn.ipynb





HANDS-ON

#2

Inscrições abertas

Participe

ccee

a ccee ▾

comunicação ▾

dados e análises ▾

preços ▾

mercado ▾

documentos ▾

ccee academy ▾



Veja mais



Preço d



Hora vigente 11:00

1º Fórum ANÁLISE SETORIAL ccee

home

bandeira tarifária

O regime de bandeiras tarifárias é um mecanismo que repassa imediatamente ao consumidor eventuais aumentos nos custos da geração de energia elétrica. Esses recursos são administrados pela CCEE, por determinação da Agência Nacional de Energia Elétrica – Aneel.

Existem três tipos de bandeiras tarifárias, que adotam uma simbologia parecida com um semáforo: elas são divididas nas cores verde, amarela e vermelha.

A verde indica que as condições para a geração de energia são favoráveis e que, portanto, não é preciso fazer cobranças adicionais. A amarela sinaliza que as circunstâncias não são tão cômodas e que é necessário cobrar um acréscimo na energia consumida. E a vermelha aponta o quadro mais desfavorável, com um custo de produção muito elevado, sendo inevitável recolher uma tarifa adicional.

Os consumidores podem descobrir qual é a bandeira tarifária em vigor procurando a informação no boleto mensal de cobrança.



bandeirastarifárias

ccee

0800 881 2233 • www.ccee.org.br • agosto/2025

Tabela 11 – Número de horas de cada semana e patamar de carga para o mês de agosto de 2025

Patamar	Nº de Horas por Patamar nas Semanas Operativas					
	Sem 1	Sem 2	Sem 3	Sem 4	Sem 5	Sem 6
Pesado	5	25	25	25	25	0
Médio	11	65	65	65	65	10
Leve	8	78	78	78	78	38
Total	24	168	168	168	168	48

Os valores de PLD para o cálculo do PLD gatilho, considerando a média mensal são obtidos da simulação do DECOMP da primeira semana operativa de agosto de 2025 de cálculo do PLD, o mesmo que estabelece a função de custo futuro para o modelo DESSEM da primeira semana operativa do mês, e uma expectativa de PLD para as próximas semanas⁹. Estes valores de expectativa de PLD para cada semana são apresentados na Tabela 12.

Tabela 12 – Resultado da Função de Custo Futuro do DECOMP para a primeira semana de agosto e expectativa para as demais semanas

Subm.	Patamar	Expectativa de PLD do modelo DECOMP(R\$/MWh)						PLDmédio
		Sem 1	Sem 2	Sem 3	Sem 4	Sem 5	Sem 6	
Sudeste	Pesado	315,19	313,65	317,16	320,43	321,54	322,94	310,30
	Médio	308,68	308,68	309,16	312,37	314,19	316,24	
	Leve	302,67	303,90	305,44	308,04	310,04	312,17	
Sul	Pesado	315,19	313,65	317,16	320,43	321,54	322,94	310,30
	Médio	308,68	308,68	309,16	312,37	314,19	316,24	
	Leve	302,67	303,90	305,44	308,04	310,04	312,17	
Nord.	Pesado	315,19	313,65	317,16	320,43	321,54	322,94	310,30
	Médio	308,68	308,68	309,16	312,37	314,19	316,24	
	Leve	302,67	303,90	305,44	308,04	310,04	312,17	
Norte	Pesado	315,19	313,65	317,16	320,43	321,54	322,94	310,30
	Médio	308,68	308,68	309,16	312,37	314,19	316,24	
	Leve	302,67	303,90	305,44	308,04	310,04	312,17	

Os valores de carga prevista para cada semana⁹, utilizados no cálculo da carga média mensal de cada submercado para ponderar o PLD médio mensal de cada submercado, são apresentados na Tabela 13.

Tabela 13 – Expectativa de carga para cada semana e patamar e a média mensal de agosto de 2025 por submercado

Subm.	Patamar	CARGA (MWmed)						Carga Média
		Sem 1	Sem 2	Sem 3	Sem 4	Sem 5	Sem 6	

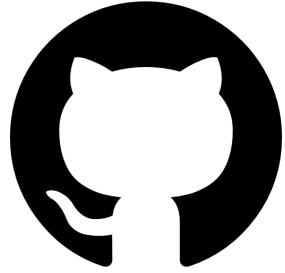
Cor da Bandeira	Gatilho
Verde	R\$ 58,6/MWh <= PLD <= R\$ 74,27/MWh
Amarela	R\$ 74,28/MWh < PLD <= R\$ 186,46/MWh
Vermelha 1	R\$ 186,47/MWh < PLD <= R\$ 256,9/MWh
Vermelha 2	R\$ 256,91/MWh < PLD <= R\$ 751,73/MWh

mostrado na Tabela 12.

Tabela 17 – Cor da Bandeira Tarifária de agosto de 2025

Cor da Bandeira Tarifária	Agosto de 2025	Vermelha 2

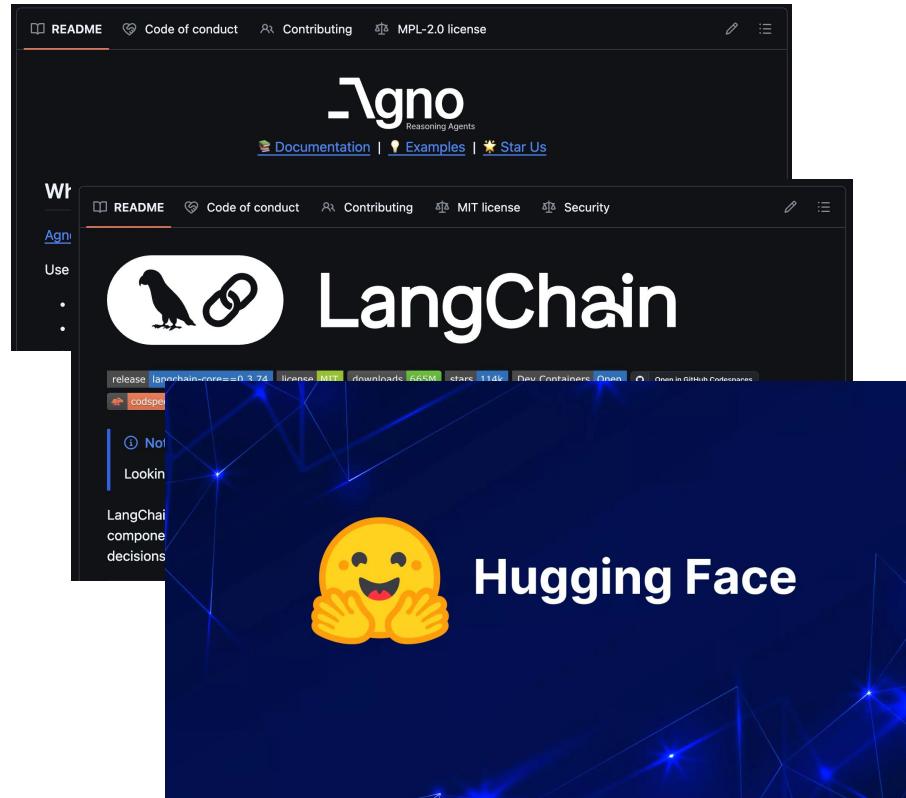
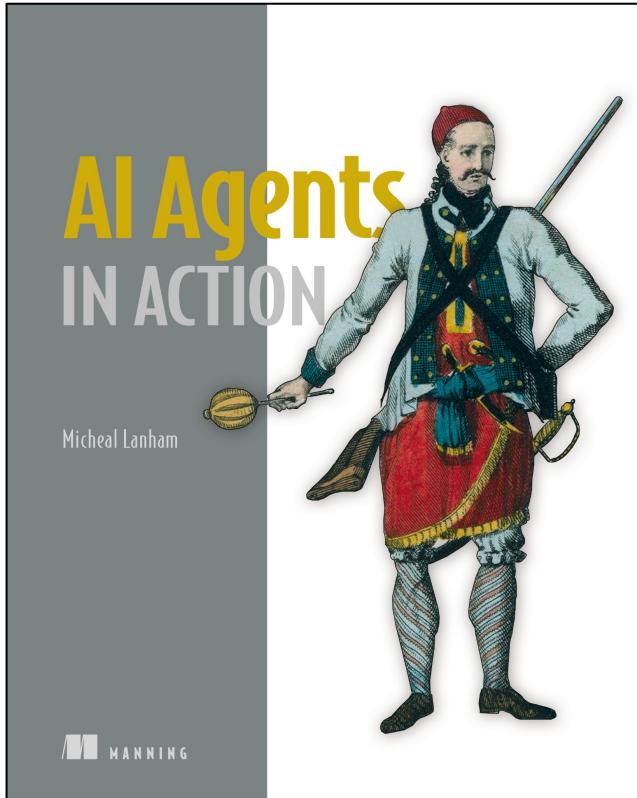
Note 1: Cabe destacar que o cálculo do PLD_{liminf_pat} e PLD_{limsup_pat} consideram os valores apresentados na Figura 5 da Nota Técnica nº 021/2021-SRG-SGT-SRM/ANEEL e o valor de GSF_{band} obtido pela proporção de GH_{band}/GF_{band} é considerado com arredondamento em duas casas decimais, seguindo solicitação da ANEEL.



<https://github.com/conect2ai/Agents-Bootcamp>
02_HandsOn.ipynb

CONCLUSION

REFERENCES





THANKS!



matheus.diniz.122@ufrn.edu.br



@conect2ai / @dinizmaths

