FSBM
Faculté des sciences ben m'sick
Université Hassan II de Casalanca

MATHÉMATIQUES & INFORMATIQUE

Data Science & Big Data
Master DSBD

Large Language Models: Falcon

Master Data Science & Big Data / Module : Deep Learning

Realiser Par:

Abdelhakim EL Ghayoubi

Encadre Par:

Pr. Benlahmar Elhabib

Table of contents

01

LLM's

What are LLM ?

- **Definition**:
  - Large Language Models (LLMs) are advanced AI systems capable of processing, understanding, and generating human-like text.
  - Built primarily on **Transformer architecture**, leveraging the **self-attention mechanism** for efficient parallel processing and context understanding.

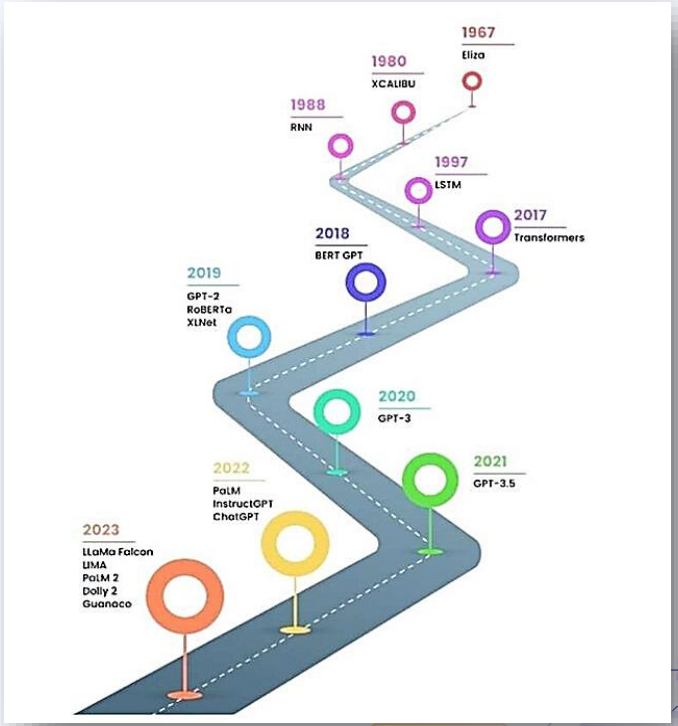
- **Key Characteristics**:
  - Massive number of parameters (e.g., billions to trillions).
  - Trained on diverse and massive datasets (e.g., books, websites, articles).
  - Capable of few-shot and zero-shot learning.

- **Examples**:
  - **OpenAI GPT-3/4**, **Google PaLM**, **LLaMA**, **Falcon, Mistral, BERT, phi, ...**

History of LLM

Milestones in LLM Development:

- **2017**: Introduction of Transformer architecture by Vaswani et al. (attention is all you need).
- **2018**: BERT (Bidirectional Encoder Representations from Transformers) revolutionized NLP tasks.
- **2019**: GPT-2 demonstrated zero-shot capabilities.
- **2020**: GPT-3 showcased massive scaling and general-purpose NLP abilities.
- **2023**: Models like GPT-4, PaLM-2, and Falcon-180B pushed boundaries in performance and efficiency.



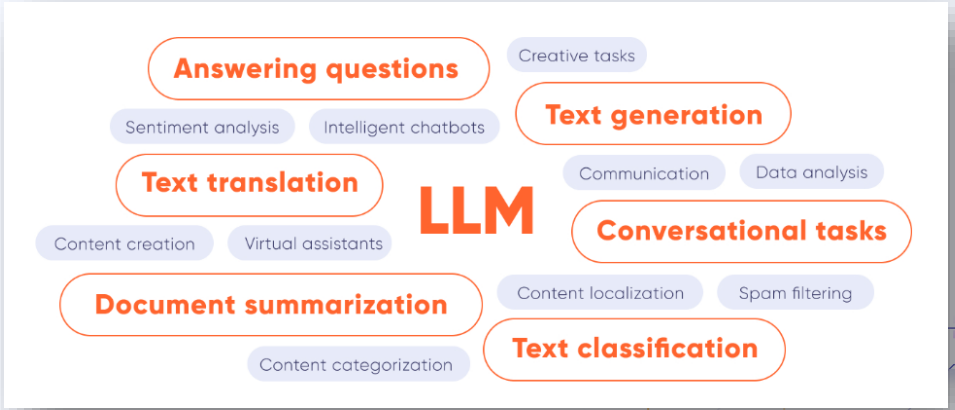
Why LLM

Core Reasons:

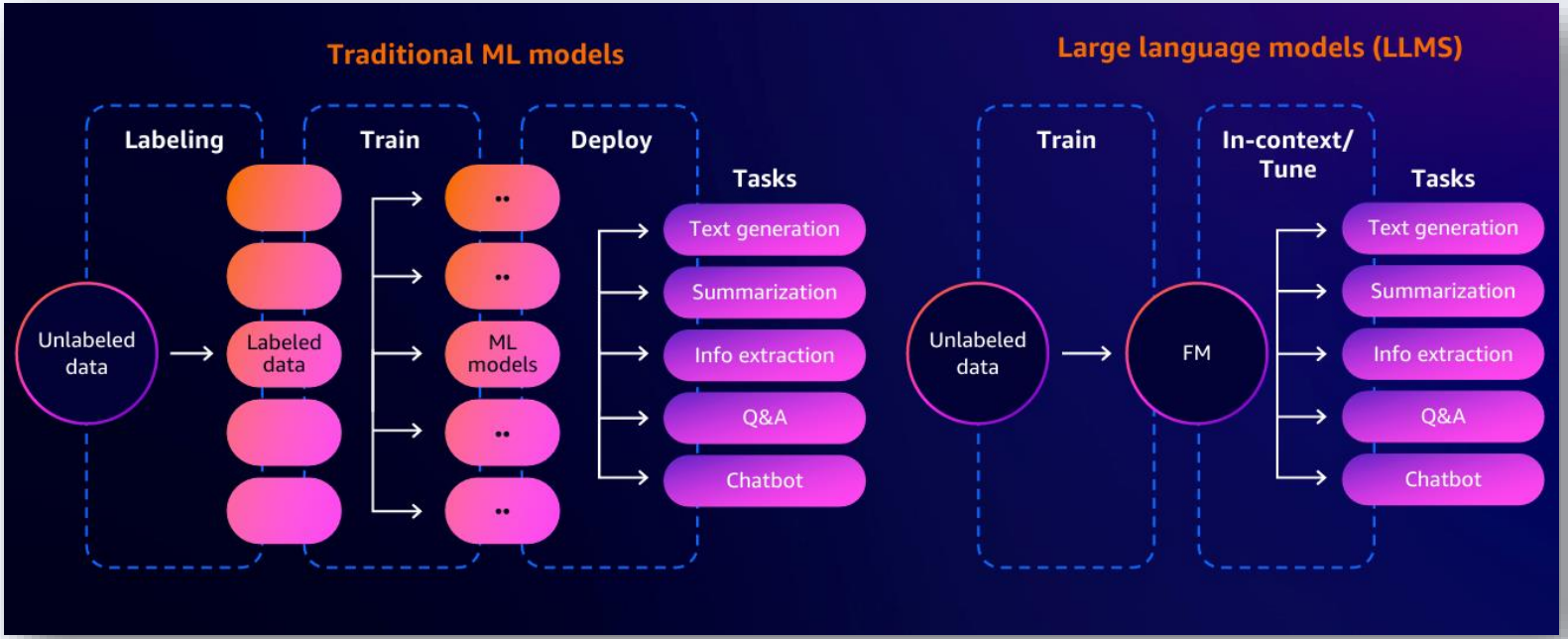
- ❑ **Versatility**: Adaptable to a variety of tasks without task-specific training.
- ❑ **Understanding Context**: Models grasp nuances, tone, and meaning from text.
- ❑ **Automation**: Reduce manual effort in repetitive or cognitive tasks (e.g., summarization, data extraction).
- ❑ **Accessibility**: Democratizes access to complex NLP tasks.

Example Applications:

- ❑ Text & code generation,
- ❑ answering queries,
- ❑ summarizing documents,
- ❑ creating conversational agents
- ❑ ...



How are LLMs different?



Use cases

- **Healthcare**: Summarizing medical records, research synthesis.
- **Education**: Personalized tutoring, summarizing lecture notes.
- **Customer Support**: Chatbots and automated assistants.
- **Software Development**: Generating code, debugging.
- **Content Creation**: Writing articles, creating social media posts

Advantages & inconvenient

Advantages:

- ❑ **Scalability**: Handles multiple languages and domains.
- ❑ **Human-like Interaction**: Offers natural conversation-like responses.
- ❑ **Automation**: Reduces manual workload significantly.
- ❑ **Continuous Improvement**: Models evolve with more data and fine-tuning.

Inconveniences:

- ❑ **Resource Intensive**: High compute and energy requirements.
- ❑ **Bias**: Inherits biases from training data.
- ❑ **Accuracy Issues**: May generate misleading or factually incorrect information.
- ❑ **Complexity**: Difficult to interpret or explain outputs.

02
Overview on Falcon LLM

Falcon LLM

What is Falcon LLM?

- Falcon is an **open-source family of causal decoder-only large language models** designed to compete with proprietary models like GPT-4 and PaLM-2.
- Built to handle diverse natural language processing (NLP) tasks with state-of-the-art efficiency and scalability.

Developer:

- Created by the **Technology Innovation Institute (TII)**, Abu Dhabi, UAE.

Characteristics:

- Fully open-source with a permissive **Apache 2.0 license**, supporting academic and commercial use.
- Models in the Falcon family include **Falcon-1B, Falcon-7B, Falcon-40B, and Falcon-180B**, as well as newer variants in Falcon2, Falcon3, and Falcon-Mamba series.
- Pretrained on **5 trillion tokens** using the refined **RefinedWeb dataset**, ensuring high-quality text inputs.

https://falconllm.tii.ae/index.html

Why Choose Falcon LLM?

- **Open-Source Accessibility**: Fully transparent with open weights and training methodologies.
- **Cost Efficiency**: Optimized for reduced training and inference costs while maintaining performance.
- **High Scalability**: Falcon models range from compact versions suitable for consumer hardware to high-performance versions requiring advanced infrastructure.
- **Cutting-Edge Performance**: Excels in reasoning, contextual understanding, and generalization, comparable to GPT-4 and PaLM-2.
- **Flexibility**: Ideal for fine-tuning on domain-specific applications. Instruction-tuned versions are available for custom NLP use cases.

The Falcon Model Family

Series	Models	Features
Falcon	Falcon-7B, Falcon-40B, Falcon-180B	General-purpose NLP tasks; scales from lightweight models to enterprise-grade setups.
Falcon 2	Falcon-2 11B, Falcon-2 11B VLM	Efficiency-focused; supports long contexts; **VLM** adds multimodal capabilities (e.g., text-to-image).
Falcon 3	1B, 3B, 7B, 10B	Emphasizes multilingual capabilities and faster inference.
Falcon-Mamba	Falcon-Mamba 7B	Uses state-space models (SSMs) for memory-efficient long-context tasks; offers high performance with reduced costs.

Applications: Text Generation, Code Generation, Conversational AI, Summarization, Multimodal AI (Falcon-2 VLM), domain-specific fine-tuning

https://falconllm.tii.ae/falcon-models.html

Strengths and Limitations

Strengths

- ❑ **Open-Source Leadership**: Accessible to researchers and enterprises alike.
- ❑ **High-Quality Training Data**: Trained on RefinedWeb, ensuring accuracy and relevance.
- ❑ **Versatility**: Applicable across domains with instruction and domain-tuned models.
- ❑ **Performance**: Near state-of-the-art accuracy across benchmarks like MMLU and HELM.
- ❑ **Scalability**: From small-scale models to enterprise-grade systems.

Limitations

- ❑ **Language Diversity**: Primarily proficient in English, German, Spanish, and French; weaker in other languages.
- ❑ **Resource Intensive**: Larger models like Falcon-180B require high-end GPUs for training and inference.
- ❑ **Bias in Data**: Web-sourced data may introduce unintended biases.
- ❑ **Limited Fine-Tuning Infrastructure**: Requires expertise and resources for domain-specific adaptations.
- ❑ **Non-Multilingual Emphasis**: Lags behind some competitors in robust multilingual NLP capabilities.

https://github.com/huggingface/blog/blob/main/falcon.md#inference

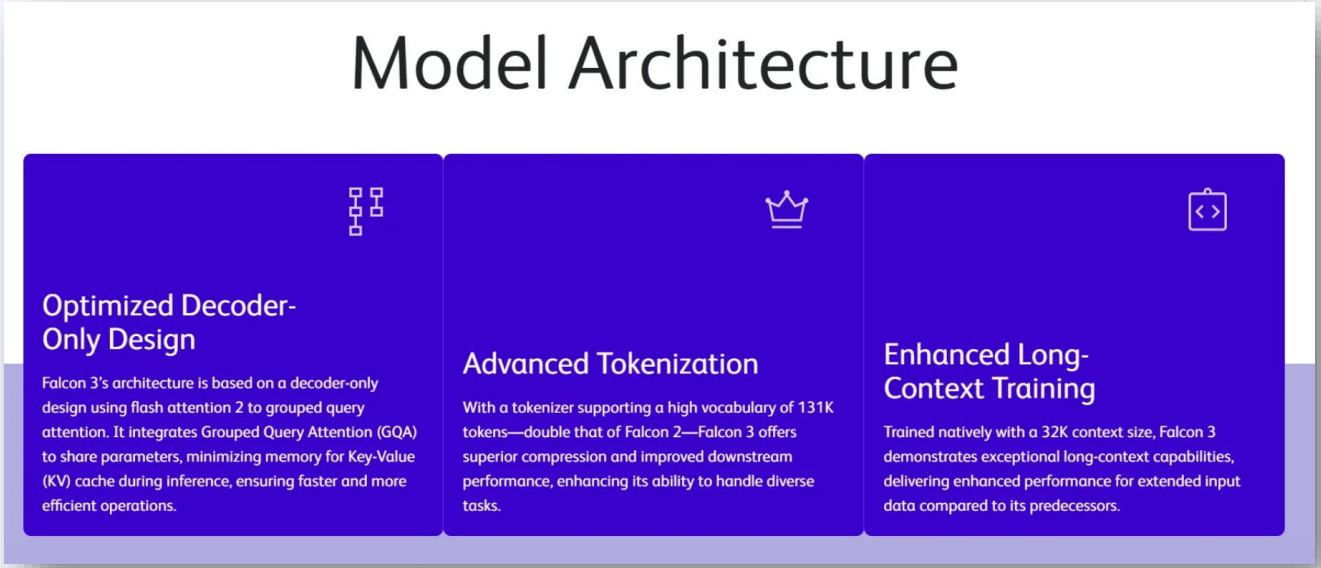
03

Falcon Model

Architecture

The Falcon team tried various combinations and found that the below worked best for them. The criteria for evaluation were the design philosophy that they need to not only improve model performance but also make sure that model design is scalable and cost /memory efficient.

the Falcon models adopt several modifications to their underlying transformer architecture that significantly accelerate inference and can even improve the efficiency of pre-training.

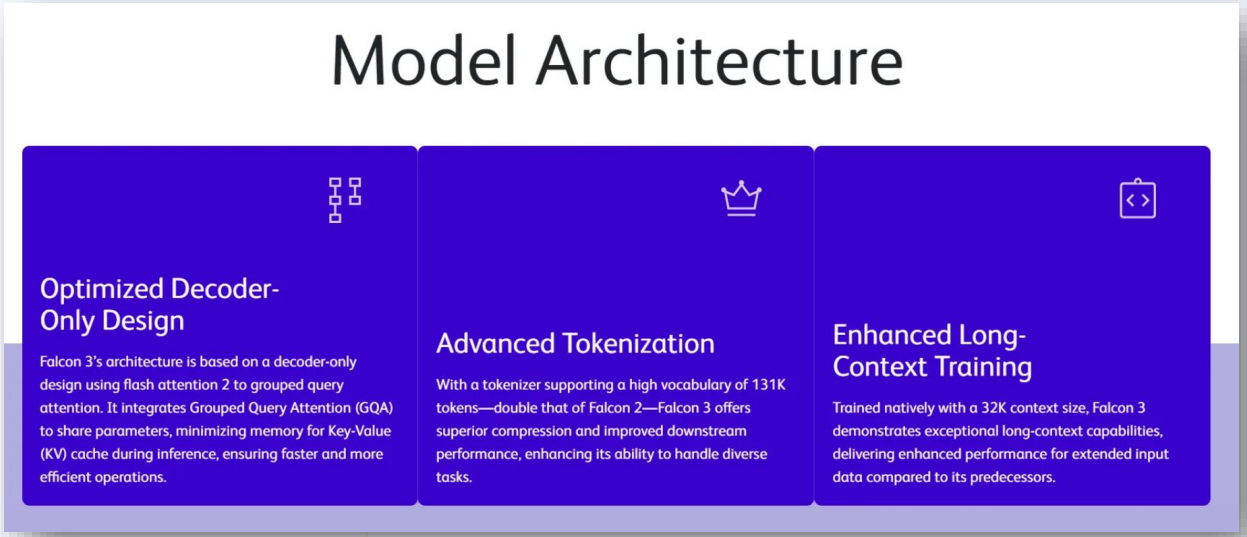


https://falconllm.tii.ae/falcon3/index.html

Architecture

Both Falcon and Falcon3 series use a modified [decoder-only transformer architecture]
Modifications made to this architecture include

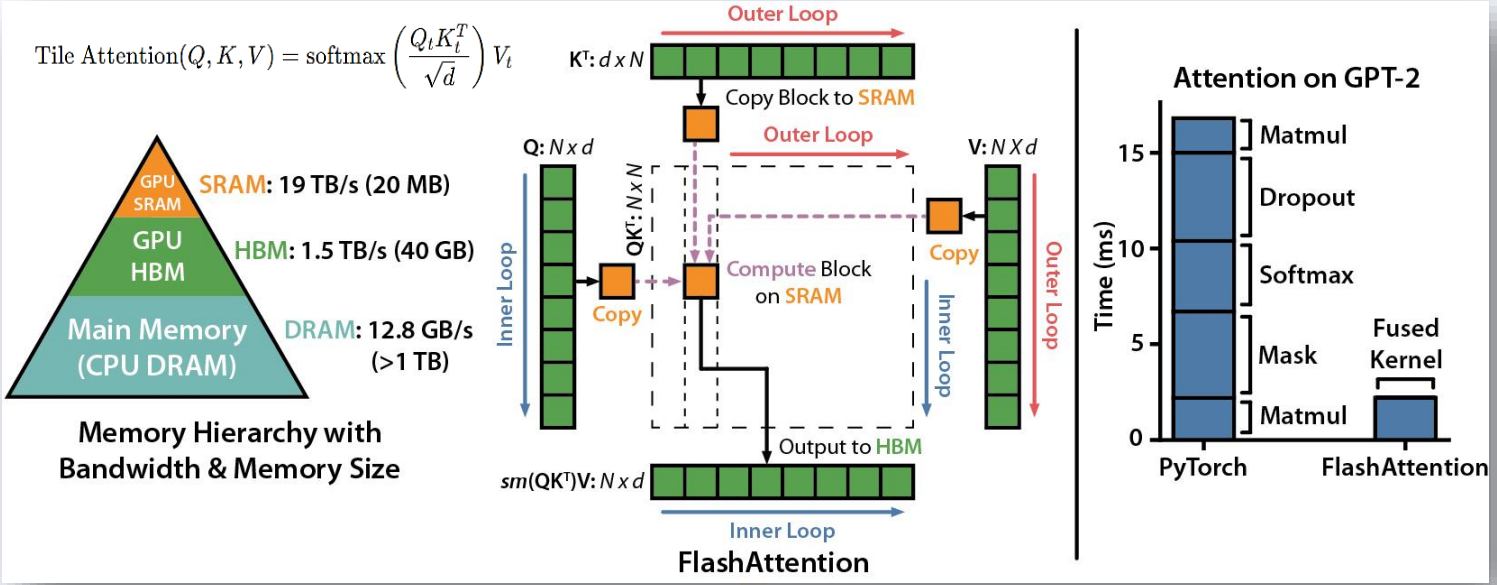
- ❑ Flash Attention
- ❑ Multi-Query Attention
- ❑ RoPE embeddings
- ❑ Parallel Attention and Feed-Forward Layers



https://falconllm.tii.ae/falcon3/index.html

Key Model Components

Flash Attention



https://github.com/Dao-AILab/flash-attention

Key Model Components

Rotary positional embeddings

The traditional transformer model had positional embeddings. RoPE was an improvement over positional embedding that combed both absolute and relative positional encodings. Key advantage was

- ❑ RoPE doesn't require fixed positions and inherently captures relative distances between tokens. This makes it more flexible and generalizable to longer sequences.
- ❑ RoPE is computationally cheaper than relative positional encodings, as it doesn't involve explicit distance calculations.
- ❑ RoPE is memory efficient as it uses a fixed embedding size regardless of sequence length, making it suitable for large models and long sequences.

Relative positional information is supplied to the model on two levels: values and keys. This becomes apparent in the two modified self-attention equations shown below. First, relative positional information is supplied to the model as an additional component to the keys.

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^\top}{\sqrt{d_z}} \quad (1)$$

The softmax operation remains unchanged from vanilla self-attention.

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

Lastly, relative positional information is supplied again as a sub-component of the values matrix.

$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a_{ij}^V) \quad (2)$$

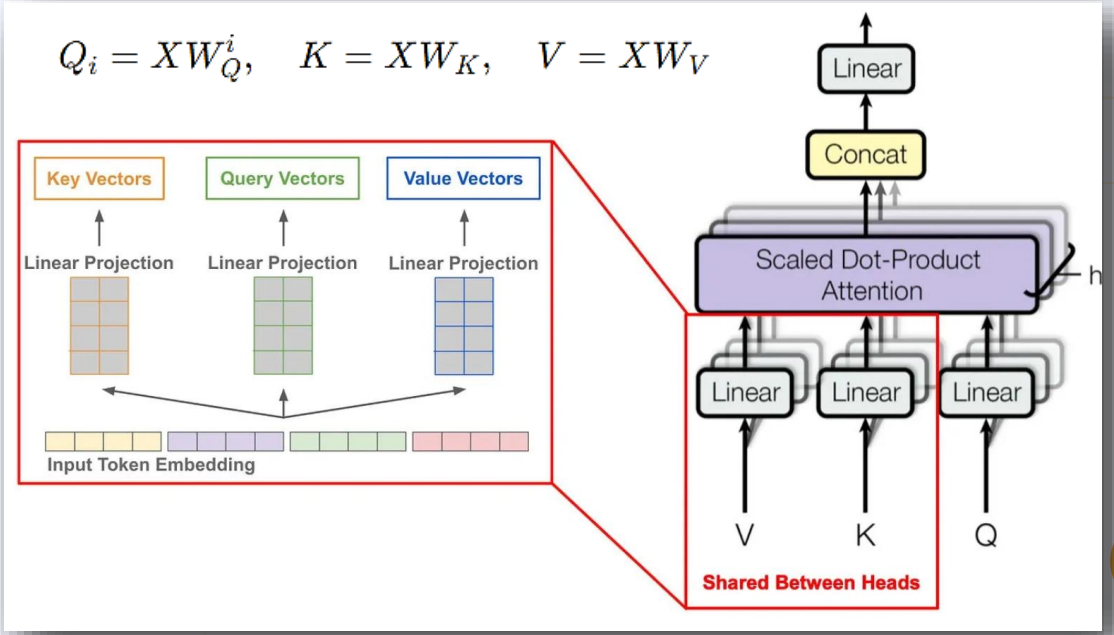
In other words, instead of simply combining semantic embeddings with absolute positional ones, relative positional information is added to keys and values on the fly during attention calculation.

https://jaketae.github.io/study/relative-positional-encoding/

Key Model Components

Multi-query attention

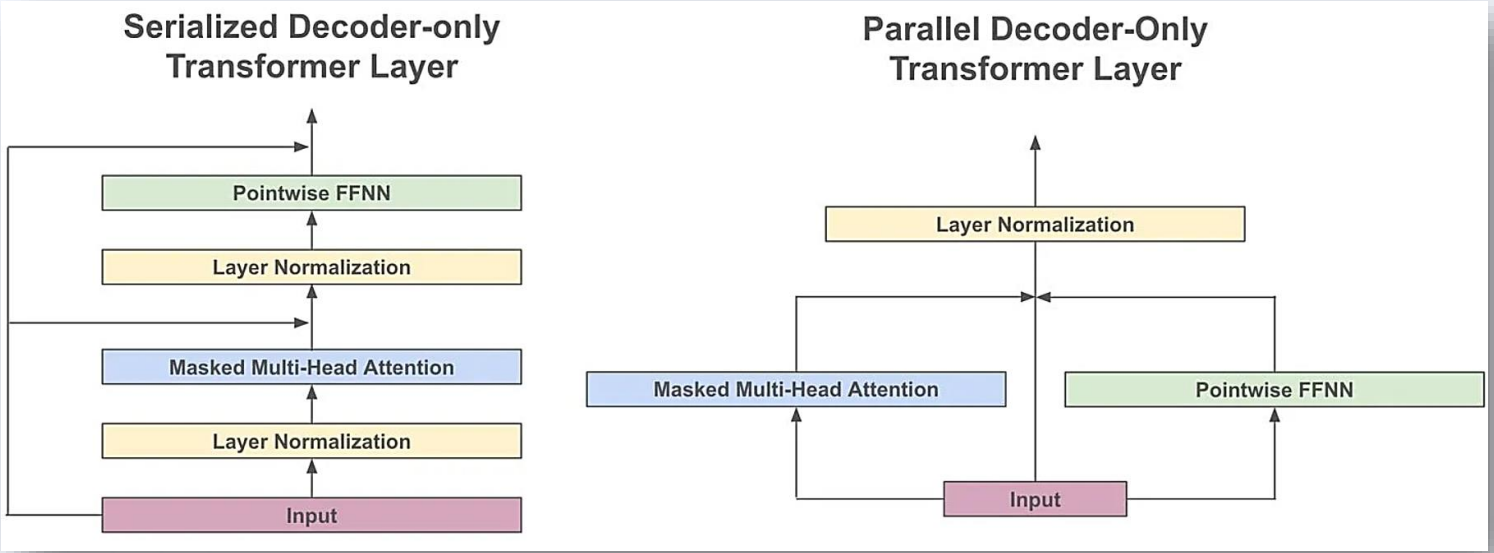
Falcon models replace the typical, multi-headed self-attention operation with an alternative structure called multi-query attention. Multi-query attention just shares key and value vectors (highlighted in red below) between each of a layer's attention heads. Instead of performing a separate projection for each head, all heads share the same projection matrix for keys and the same projection layer for values



https://pub.towardsai.net/multi-query-attention-explained-844dfc4935bf

Key Model Components

Parallel Attention and Feed-Forward Layers



https://deepai.org/publication/parallel-attention-and-feed-forward-net

Falcon Dataset

Dataset used by Falcon AI team

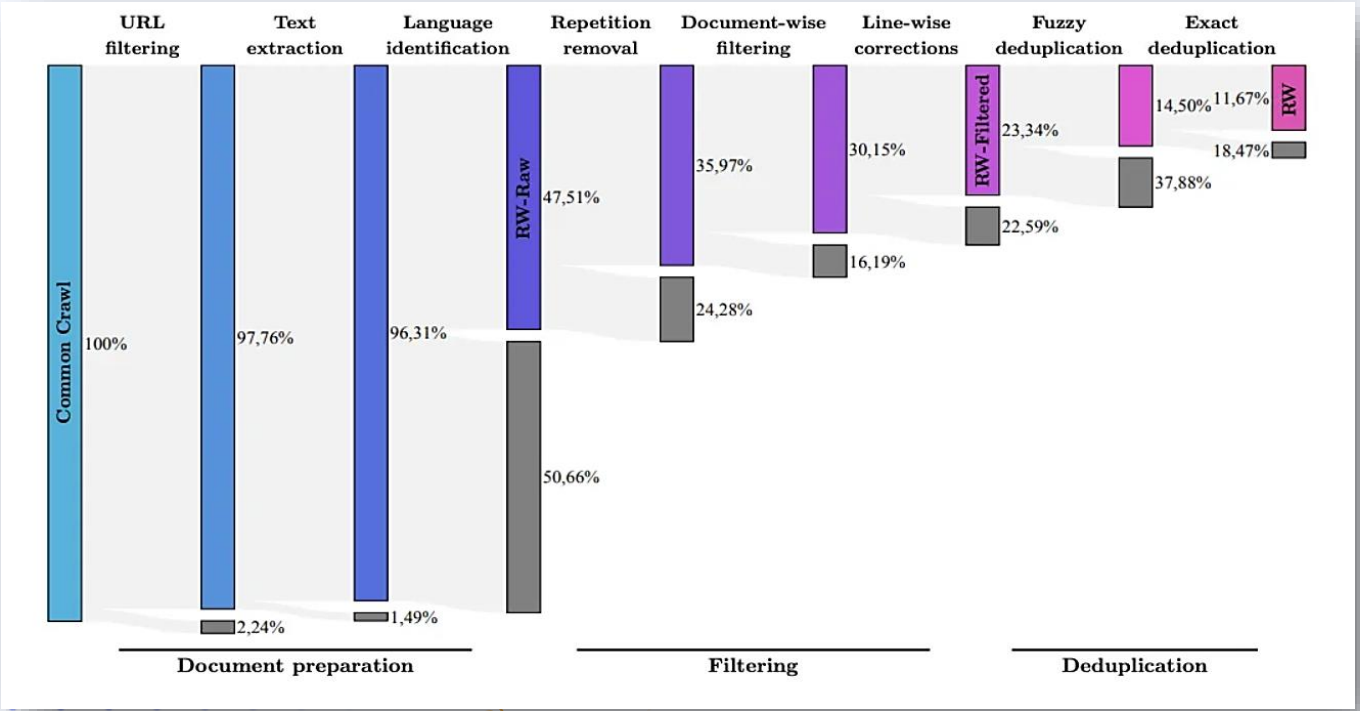
The Falcon team used the following sources of dataset to develop their pertaining dataset known as Falcon mixture.

- **RefinedWeb-English,** sourced from *CommonCrawl* it formed *76%* of the pertaining dataset
- **RefinedWeb-Euro**, sourced from *CommonCrawl* for Europe forced multilingual training it formed *8%* of pretraining dataset
- **Books**, Sourced form project *Gutenberg* it formed *6%* of pretraining dataset
- **Conversations**, It was sourced from *StackOverflow*, *HackerNews*, etc, and formed *5%* of pretraining dataset
- **Code**, It was sourced from *GitHub* and formed *3%* of the pertaining dataset
- **Technical**, It was sourced from arXiv, and Wikipedia ec and formed *2%* of the pertaining dataset.

"Challenging existing beliefs on data quality and LLMs, models trained on adequately filtered and deduplicated web data alone can match the performance of models trained on curated data."

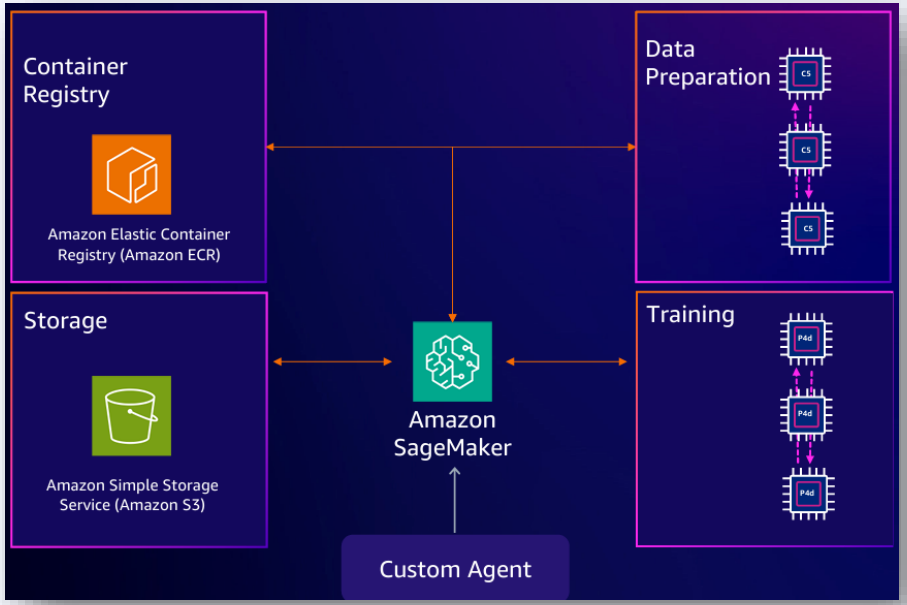
Falcon Dataset

Data Preprocessing



Training Environment

- **Hardware**: Falcon models are trained on high-performance GPU clusters, such as NVIDIA A100 GPUs with 80GB memory, leveraging tensor parallelism for distributed computation.
- **Frameworks**: Built using highly optimized libraries like **PyTorch**, with custom modifications for attention mechanisms and memory efficiency.
- **Precision**: Trained using **mixed precision (FP16)** to balance computation speed and memory efficiency.



Falcon Models Training

	Falcon 7B	Falcon 40B	Falcon 180B
Pretraining tokens	1.5B	1.5B	3.5B
Compute PF-days	730	2,800	43,500
Training [A100 GPUs]	384	384	4,096
Availability	Apache 2.0	Apache 2.0	Open Access
Aggregated Performance	60.8	67.1	70.3

Falcon3 Models Specs

Details	Falcon3-1B	Falcon3-3B	Falcon3-7B	Falcon3-10B	Falcon3-Mamba-7B
Architecture	Decoder-only	Decoder-only	Decoder-only	Decoder-only	Decoder-only
Layers	18	22	28	40	64
Vocab size	131K	131K	131K	131K	65K
Head dimension	256	256	256	256	-
Activation	SwiGLU	SwiGLU	SwiGLU	SwiGLU	SwiGLU
How?	Distillation	Distillation	From scratch	Upscaling	Continue pretrained
How much data?	80 GT	100 GT	14 TT	2 TT	1.5 TT
From which model?	Falcon3 7B	Falcon3 7B	Random Init	Falcon3 7B	Falcon Mamba 7B
Context Length	8K	32K	32K	32K	32K/Infinite

https://huggingface.co/blog/falcon3

Evaluation & Benchmark

With the release of the Falcon-7B and Falcon-40B LLMs, *for the first time* open-source base LLMs that begin to rival the quality of the most popular paid models

Model	Revision	Average	ARC (25-shot)	HellaSwag (10-shot)	MMLU (5-shot)
tiiuae/falcon-40b-instruct	main	63.2	61.6	84.4	54.1
tiiuae/falcon-40b	main	60.4	61.9	85.3	52.7
ausboss/llama-30b-supercot	main	59.8	58.5	82.9	44.3
llama-65b	main	58.3	57.8	84.2	48.8
MetaIX/GPT4-X-Alpasta-30b	main	57.9	56.7	81.4	43.6
Aeala/VicUnlocked-alpaca-30b	main	57.6	55	80.8	44
digitous/Alpacino30b	main	57.4	57.1	82.6	46.1
Aeala/GPT4-x-AlpacaDente2-30b	main	57.2	56.1	79.8	44
TheBloke/dromedary-65b-lora-HF	main	57	57.8	80.8	50.8
TheBloke/Wizard-Vicuna-13B-Uncensored-HF	main	57	53.6	79.6	42.7

The results show that the 40B base and instruct models are very strong, and currently rank 1st and 2nd on the LLM leaderboard 🏆!

Evaluation & Benchmark

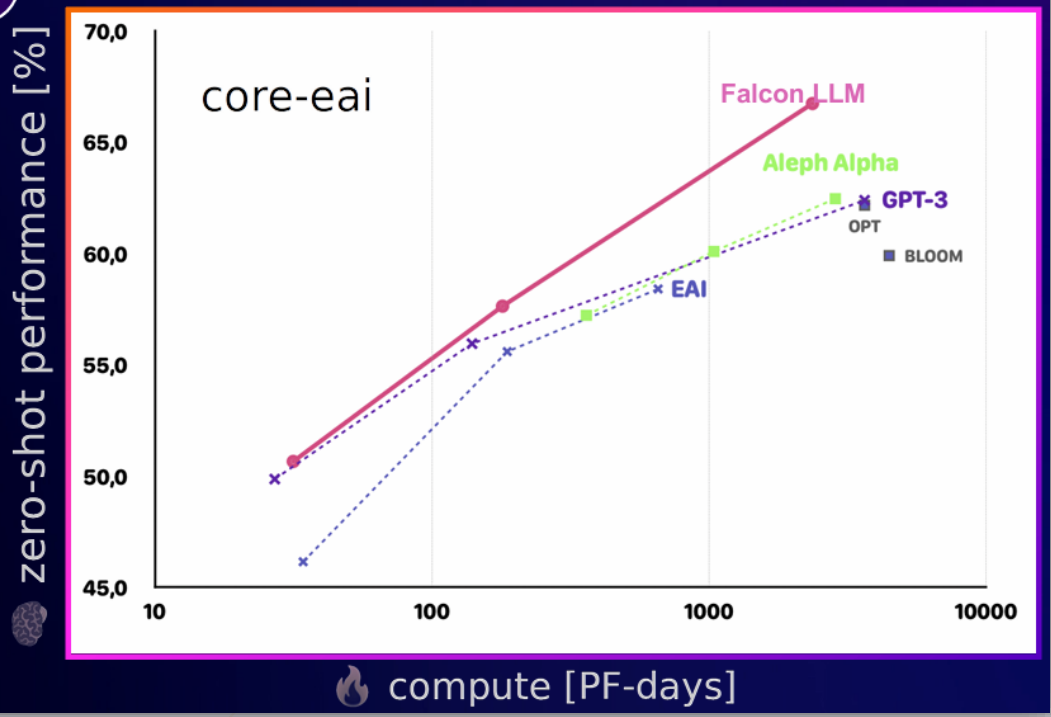
Model	License	Commercial use?	Pretraining length [tokens]	Pretraining compute [PF-days]	Leaderboard score	K,V-cache size for a 2.048 context
StableLM-Alpha-7B	CC-BY-SA-4.0	✅	1,500B	700	34.37	800MB
LLaMA-7B	LLaMA license	❌	1,000B	500	45.65	1,100MB
MPT-7B	Apache 2.0	✅	1,000B	500	44.28	1,100MB
Falcon-7B	Apache 2.0	✅	1,500B	700	44.17	20MB
LLaMA-33B	LLaMA license	❌	1,500B	3200	-	3,300MB
LLaMA-65B	LLaMA license	❌	1,500B	6300	61.19	5,400MB
Falcon-40B	Apache 2.0	✅	1,000B	2800	58.07	240MB

Model	Type	Average leaderboard score
tiiuae/falcon-40b-instruct	instruct	63.2
tiiuae/falcon-40b	base	60.4
llama-65b	base	58.3
TheBloke/dromedary-65b-lora-HF	instruct	57
stable-vicuna-13b	rlhf	52.4
llama-13b	base	51.8
TheBloke/wizardLM-7B-HF	instruct	50.1
tiiuae/falcon-7b	base	48.8
mosaicml/mpt-7b	base	48.6
tiiuae/falcon-7b-instruct	instruct	48.4
llama-7b	base	47.6

https://huggingface.co/blog/falcon

Evaluation & Benchmark

Our first LLM (Falcon, 40B parameters) outperforms GPT-3, Chinchilla, and PaLM-62B, despite being only a fraction of the size.



* Tasks **core-eai**: HellaSwag, LAMBADA, Winogrande, PIQA, ARC, OpenBook QA

Evaluation & Benchmark

Base models	MUSR	BBH	MMLU_PRO	If_eval	GPQA	MATH	Avg
Falcon3 10B	14.17	41.38	36	36.48	12.75	24.77	**27.59**
Falcon3 7B	18.14	31.56	32.34	34.16	12.86	19.26	24.72
Qwen2.5 7B	14.14	35.81	37.39	33.74	9.96	18.88	24.99
Qwen2 7B	14.32	34.71	35.37	31.49	7.27	20.47	23.94
Gemma2 9B	14.3	34.1	34.48	20.4	10.51	13.14	21.15
Mistral-nemo-base-2407 (12B)	6.52	29.37	27.46	16.3	5.82	5.36	15.14
Llama3.1 8B	8.98	25.29	24.95	12.7	6.15	5.14	13.87

Instruct models	MUSR	BBH	MMLU_PRO	If_eval	GPQA	MATH	Avg
Falcon3 10B Instruct	13.61	44.82	38.1	78.17	10.51	25.91	**35.19**
Falcon3 7B Instruct	21.17	37.92	34.3	76.12	8.05	31.87	**34.91**
Llama3.1 8B Instruct	8.41	29.89	30.68	78.56	2.35	19.34	28.2
Qwen2.5 7B Instruct	8.45	34.89	36.52	75.85	5.48	0	26.87
Qwen2 7B Instruct	7.37	37.81	31.64	56.79	6.38	9.44	24.9
Mistral-Nemo-Instruct-2407 (12B)	8.48	29.68	27.97	63.8	5.37	6.5	23.63
Gemma2 9B It	13.77	21.62	20.48	50.1	2.68	6.72	19.23

04

Demo

Falcon 3

- **Falcon 3** can run on *light infrastructures*, **even laptops**, without sacrificing performance.
- The Falcon 3 ecosystem contains four scalable models tailored for diverse applications.
- Falcon 3 supports several languages and is optimized for resource efficiency.
- Download: https://ollama.com/library/falcon3
- Falcon3: https://falconllm.tii.ae/falcon3/index.html

Falcon 3

Different ways of using a Falcon3 LLM

Method	Platform	Best For	Requirements
Local Setup	PC	Offline tasks, development, fine-tuning	Python, HF Transformer
Hugging Face	Online/Local	Quick prototyping, experimentation	Hugging Face account or local GPU
Ollama	Any OS	Easy-to-use, lightweight deployments	Ollama installed

- https://ollama.com/library/falcon3
- https://huggingface.co/collections/tiiuae/falcon3-67605ae03578be86e4e87026

Examples


```
lenovo@Lenovo-Laptop MINGW64 ~
$ ollama list
NAME                            ID              SIZE      MODIFIED
huihui_ai/falcon3-abliterated:1b    6f74181aa7f7    1.1 GB    46 hours ago
phi:latest                      e2fd6321a5fe    1.6 GB    4 days ago
(base)
lenovo@Lenovo-Laptop MINGW64 ~
$ ollama show huihui_ai/falcon3-abliterated:1b
  Model
    architecture        llama
    parameters          1.7B
    context length      8192
    embedding length    2048
    quantization        Q4_K_M

  Parameters
    stop    "<|system|>"
    stop    "<|user|>"
    stop    "<|end|>"
    stop    "<|assistant|>"

  License
    Falcon 3 TII Falcon License
    December 2024
```

```
C:\Users\lenovo>ollama run huihui_ai/falcon3-abliterated:1b
>>> hello model my name is abdelhakim, introduce your self
Hello! My name is AI. I am a language model designed to assist with a wide range of questions and tasks. Feel free to ask me anything, and I'll do my best to provide
helpful information or perform the requested actions.

>>> soo what is falcon llm
Falcon LLM (Language Model) is a specific variant of the Falcon models developed by researchers at the Technology Innovation Institute (TII). Falcon series models are
known for their strong performance in various natural language processing tasks and have been particularly successful in areas like question answering, summarization,
and generating creative text formats. Falcon LLMs, such as Falcon LLM, often achieve state-of-the-art results due to their advanced architecture and pretraining on
large corpora of text data. They are designed to be highly efficient and versatile, making them suitable for a wide range of applications in AI chatbots, virtual
assistants, content generation, and more.

>>> can you code ?
Certainly! To get started with coding, you can use any programming language that supports basic syntax and data structures. Here's a simple example using Python,
which is known for its readability:

```python
# Create a file named "hello_world.py"
with open('hello_world.py', 'w') as f:
    f.write("print('Hello, World!')" )

# Run the script
python hello_world.py
```

This code opens a new file called `hello_world.py`, writes the string "Hello, World!" to it, and then runs the Python interpreter to execute the file. You can save
this file with any name you like.

If you want to create a more interactive program where you can prompt users for input or respond to their inputs, you might also consider using Python's built-in
`input()` function:

```python
# Prompt user for a message
message = input("What is your name? ")
print(f"Hello, {message}! Nice to meet you.")
```
In this case, when the program runs, it will ask the user for their name and then greet them with a personalized message.
```

```
>>> give me a code for linear regression model trained on random numpy values
To train a Linear Regression model using Random NumPy arrays as features and responses, you can follow these steps:

1. **Import Necessary Libraries:** You'll need NumPy for creating the random arrays and SciKit-Learn for defining and training the regression model.
2. **Generate Random Data:** Create two numpy arrays, one for the features (`X`) and one for the responses (`y`). Each array will have a length of 1000 and be
generated randomly using `np.random.rand()`.
3. **Split Data into Training and Testing Sets:** Divide your data into a training set and a testing set. For this example, we'll use 80% for training and 20% for
testing.
4. **Train the Model:** Use `fit()` method from SciKit-Learn to train the Linear Regression model with the provided features and responses.
5. **Evaluate the Model:** Use `score()` to compute the coefficient of determination (`R^2`) which measures how well the model fits the data. The closer it is to 1,
the better the fit.
6. **Predict Values:** Use the trained model to predict new response values for the test set.
7. **Plotting (Optional):** Visualize the results using Matplotlib for a better understanding of the linear relationship between features and responses.
```

Examples

Here's the code:

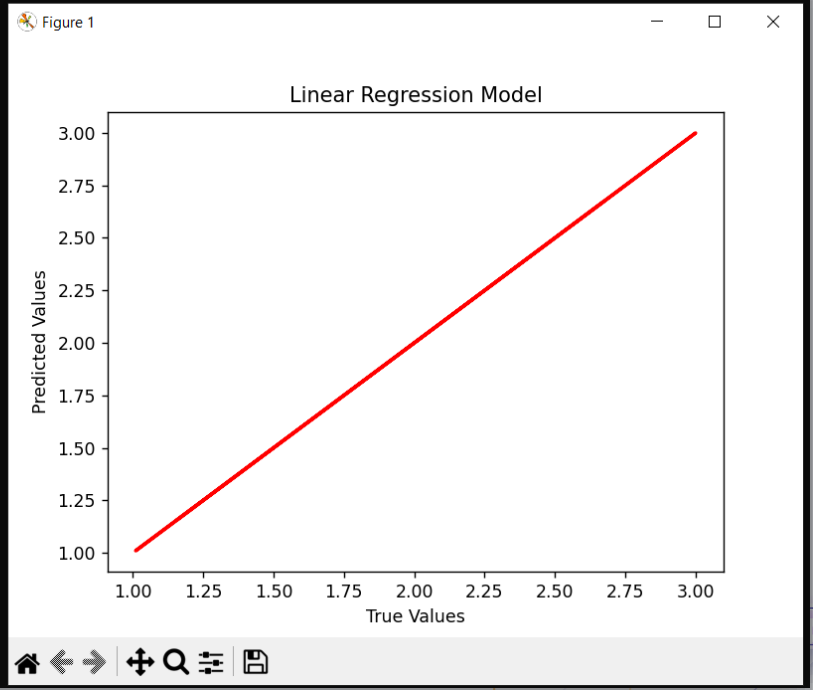
```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

# Step 1: Generate Random Data
np.random.seed(42)  # For reproducibility
X = np.random.rand(1000, 1)  # Features array
y = X * 2 + 1   # Linear relationship between features and responses

# Step 2: Split Data into Training and Testing Sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)  # 80% training, 20% testing

# Step 3: Train the Model
model = LinearRegression()
model.fit(X_train, y_train)

# Step 4: Evaluate the Model
score = model.score(X_test, y_test)  # Compute R^2 score
print(f"R^2 Score: {score}")

# Step 5: Predict Values for Test Set
predictions = model.predict(X_test)

# Optional: Plotting the results (optional)
plt.scatter(y_test, y_train, color='blue')  # Training data points
plt.plot(y_test, predictions, color='red', linewidth=2)  # Predicted values
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.title('Linear Regression Model')
plt.show()
```

This code will train a Linear Regression model on the random data generated with NumPy arrays, evaluate its performance using `r2_score`, and then visualize the results using Matplotlib.

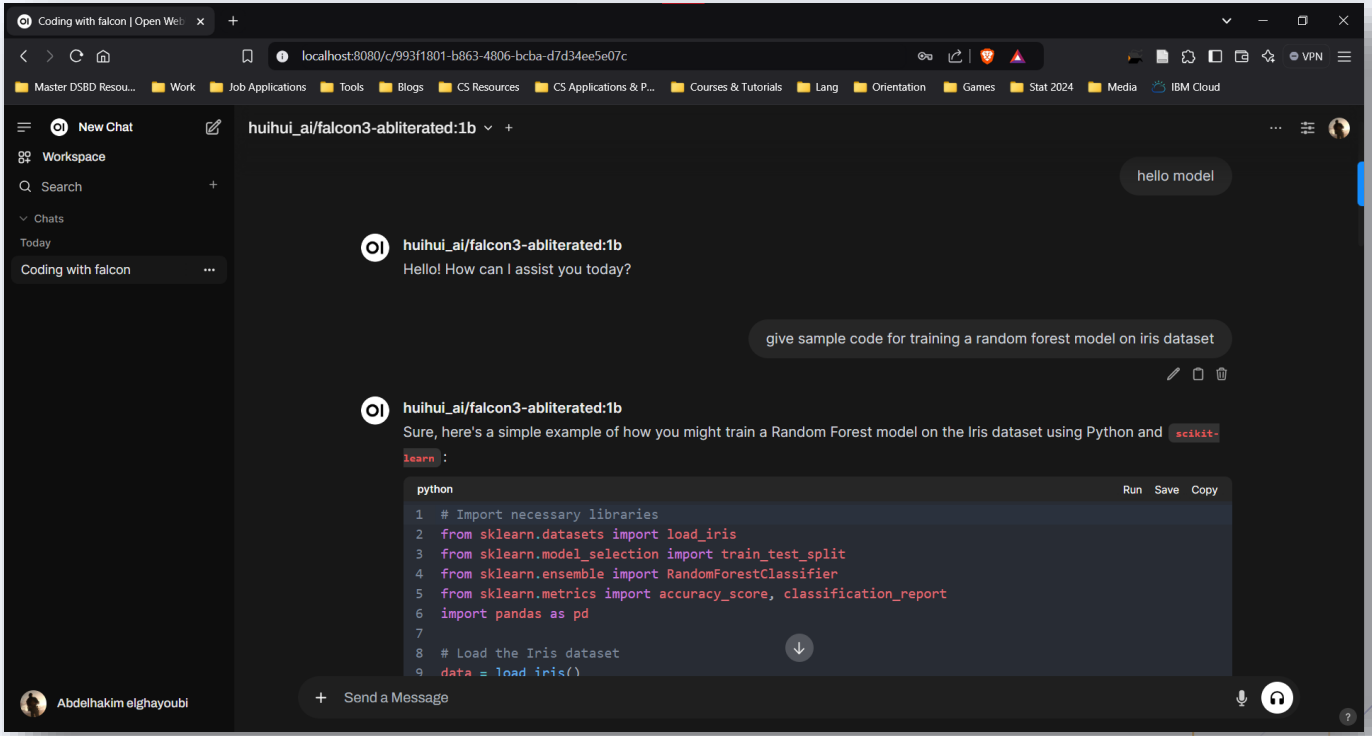
>>> Send a message (/? for help)

Examples


```
Python 3.11 (64-bit)
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> from sklearn.metrics import r2_score
>>> import matplotlib.pyplot as plt
>>>
>>> # Step 1: Generate Random Data
>>> np.random.seed(42)  # For reproducibility
>>> X = np.random.rand(1000, 1)  # Features array
>>> y = X * 2 + 1   # Linear relationship between features and responses
>>>
>>> # Step 2: Split Data into Training and Testing Sets
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
>>>
>>> # Step 3: Train the Model
>>> model = LinearRegression()
>>> model.fit(X_train, y_train)
LinearRegression()
>>>
>>> # Step 4: Evaluate the Model
>>> score = model.score(X_test, y_test)  # Compute R^2 score
>>> print(f"R^2 Score: {score}")
R^2 Score: 1.0
>>>
>>> # Step 5: Predict Values for Test Set
>>> predictions = model.predict(X_test)
>>>
>>> # Optional: Plotting the results (optional)
>>> plt.scatter(y_test, y_train, color='blue')  # Training data points
```



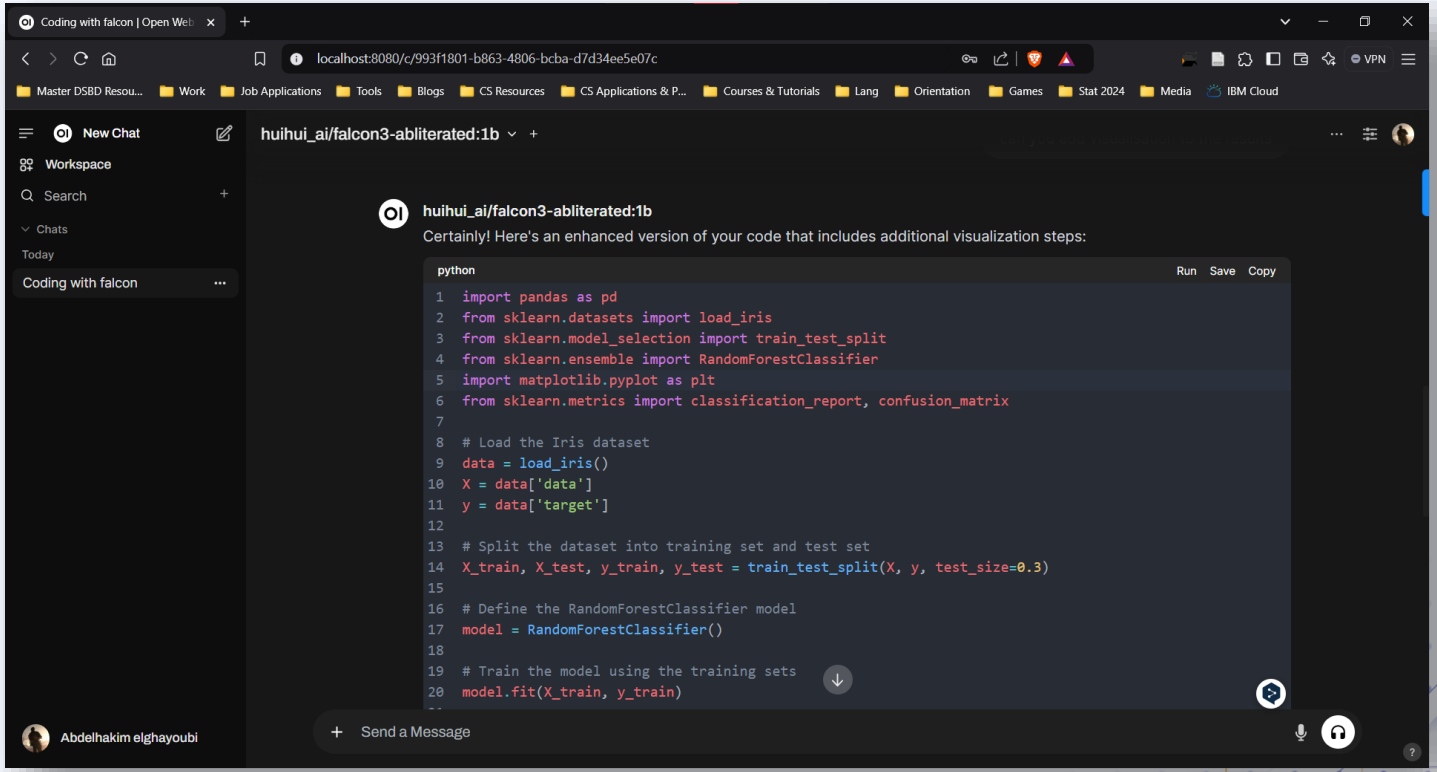
Examples


Coding with falcon | Open Web
localhost:8080/c/993f1801-b863-4806-bcba-d7d34ee5e07c
Master DSBD Resou...
Work
Job Applications
Tools
Blogs
CS Resources
CS Applications & P...
Courses & Tutorials
Lang
Orientation
Games
Stat 2024
Media
IBM Cloud
New Chat
Workspace
Search
Chats
Today
Coding with falcon
Abdelhakim elghayoubi
huihui_ai/falcon3-abliterated:1b
hello model
huihui_ai/falcon3-abliterated:1b
Hello! How can I assist you today?
give sample code for training a random forest model on iris dataset
huihui_ai/falcon3-abliterated:1b
Sure, here's a simple example of how you might train a Random Forest model on the Iris dataset using Python and scikit-learn:
python
Run
Save
Copy
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
# Load the Iris dataset
data = load_iris()
Send a Message

Examples


```
Python 3.11 (64-bit)
Type "help", "copyright", "credits" or "license" for more information.
>>> # Import necessary libraries
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.metrics import accuracy_score, classification_report
>>> import pandas as pd
>>>
>>> # Load the Iris dataset
>>> data = load_iris()
>>> X = data['data']
>>> y = data['target']
>>>
>>> # Split the dataset into training set and test set
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
>>>
>>> # Define the RandomForestClassifier model
>>> model = RandomForestClassifier()
>>>
>>> # Train the model using the training sets
>>> model.fit(X_train, y_train)
RandomForestClassifier()
>>>
>>> # Make predictions on the test set
>>> y_pred = model.predict(X_test)
>>>
>>> # Evaluate the performance of the model
>>> print("Training Accuracy:", accuracy_score(y_test, y_pred))
Training Accuracy: 0.9777777777777777
>>> print("Test Accuracy:", accuracy_score(y_test, y_pred))
Test Accuracy: 0.9777777777777777
>>> print("Classification Report:\n", classification_report(y_test, y_pred))
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.93      0.96        14
           2       0.94      1.00      0.97        16

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

Examples


Coding with falcon | Open Web
localhost:8080/c/993f1801-b863-4806-bcba-d7d34ee5e07c
Master DSBD Resou...
Work
Job Applications
Tools
Blogs
CS Resources
CS Applications & P...
Courses & Tutorials
Lang
Orientation
Games
Stat 2024
Media
IBM Cloud
New Chat
Workspace
Search
Chats
Today
Coding with falcon
Abdelhakim elghayoubi
huihui_ai/falcon3-abliterated:1b
huihui_ai/falcon3-abliterated:1b
Certainly! Here's an enhanced version of your code that includes additional visualization steps:
python
Run
Save
Copy
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
# Load the Iris dataset
data = load_iris()
X = data['data']
y = data['target']
# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Define the RandomForestClassifier model
model = RandomForestClassifier()
# Train the model using the training sets
model.fit(X_train, y_train)
Send a Message

Examples


Python 3.11 (64-bit)
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.ensemble import RandomForestClassifier
>>> import matplotlib.pyplot as plt
>>> from sklearn.metrics import classification_report, confusion_matrix
>>>
>>> # Load the Iris dataset
>>> data = load_iris()
>>> X = data['data']
>>> y = data['target']
>>>
>>> # Split the dataset into training set and test set
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
>>>
>>> # Define the RandomForestClassifier model
>>> model = RandomForestClassifier()
>>>
>>> # Train the model using the training sets
>>> model.fit(X_train, y_train)
RandomForestClassifier()
>>>
>>> # Make predictions on the test set
>>> y_pred = model.predict(X_test)
>>>
>>> # Visualize the confusion matrix
>>> plt.figure(figsize=(8,6))
<Figure size 800x600 with 0 Axes>
>>> confusion_matrix = confusion_matrix(y_test, y_pred)
>>> plt.title('Confusion Matrix')
Text(0.5, 1.0, 'Confusion Matrix')
>>> plt.matshow(confusion_matrix, cmap='Blues', alpha=0.75)
<matplotlib.image.AxesImage object at 0x000001B14FC9E590>
>>> plt.colorbar()
<matplotlib.colorbar.Colorbar object at 0x000001B14D2473D0>
>>> tick_marks = plt.xticks(rotation=45) # Rotate the x-axis labels for better
>>> plt.xlabel('Predicted Label')
Text(0.5, 0, 'Predicted Label')
>>> plt.ylabel('True Label')
Text(0, 0.5, 'True Label')
>>> plt.title('Confusion Matrix for Iris Dataset')
Text(0.5, 1.0, 'Confusion Matrix for Iris Dataset')
>>> plt.show()
Figure 2
Confusion Matrix for Iris Dataset
True Label
Predicted Label
0
1
2
14
12
10
8
6
4
2
0

Thanks for your attention 😊!