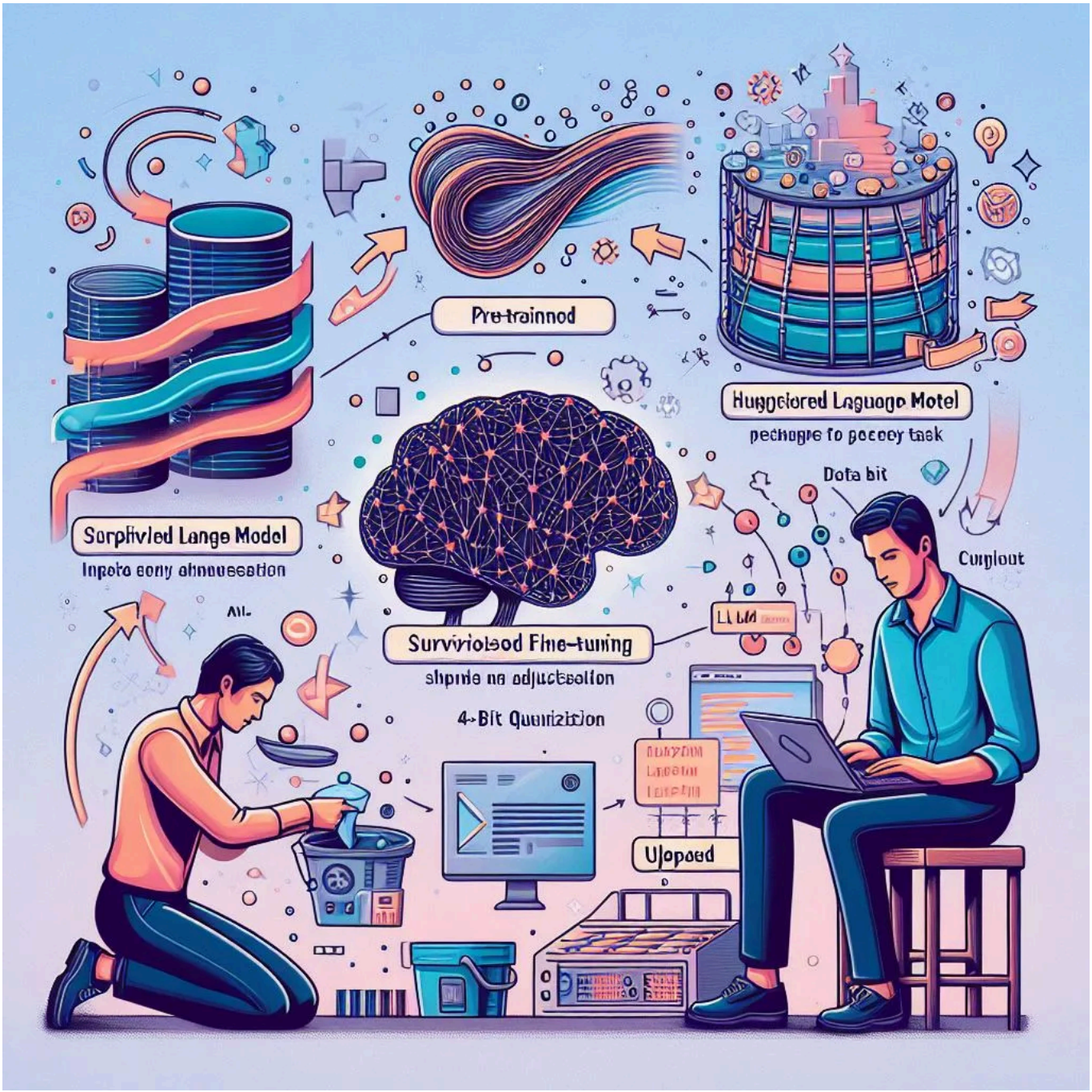
A Guide to Supervised Fine-Tuning and 4-Bit Quantization for Language Models, Pushed to the Hugging Face Model Hub

Mohammed Ashraf · Follow

7 min read · Mar 3, 2024



(**Generated with AI Prompt:** Create an illustration depicting a pathway of transformation, starting with a pre-trained large language model (LLM) and progressing through supervised fine-tuning and 4-bit quantization. Show the model's journey as it becomes optimized for a specific task, represented by various data inputs and adjustments. Finally, visualize the model being uploaded and shared on the Hugging Face Model Hub, surrounded by a community of developers and researchers. Emphasize the efficiency and collaborative nature of this process in advancing natural language processing technology.)

In my previous blog post, I provided a comprehensive, step-by-step guide on the process of fine-tuning a Large Language Model (LLM) using the Supervised Fine-tuning method. This method allows you to tailor the model's parameters to suit your specific data and task requirements, thereby

enhancing its performance and effectiveness. If you're interested in delving deeper into the intricacies of fine-tuning models on your data, I highly recommend checking out that blog post. You can find it

Your Ultimate Guide to Instinct Fine-Tuning and Optimizing Google's Gemma 2B Using LoRA

On February 21, 2024, Google's Keras team unveiled Gemma, a new set of lightweight open-source models. Gemma models...

medium.com

In this blog, our primary focus is on the crucial step of saving your newly fine-tuned model and seamlessly pushing it to the Hugging Face Model Hub. Many individuals encounter challenges when attempting to save their quantized models and integrate them with the Hugging Face ecosystem. Therefore, we'll provide detailed guidance and practical tips to streamline this process, ensuring that your meticulously crafted model can be readily accessed and utilized by others in the community.

For demonstration purposes, we'll undertake the fine-tuning of the Gemma-2B-IT model specifically for the task of generating Python code. Leveraging the SFTTrainer, bitsandbytes, and peft frameworks, we'll delve into the intricacies of refining this pre-trained model to excel in generating Python scripts.

Let's begin our journey into the fascinating world of fine-tuning and model deployment.

Part 1: Fine-Tuning an LLM Model in 4-Bit Quantization Using Supervised Fine-Tuning Method

Step 1: Install all necessary libraries

```
!pip3 install -q -U bitsandbytes==0.42.0
!pip3 install -q -U peft==0.8.2
!pip3 install -q -U trl==0.7.10
!pip3 install -q -U accelerate==0.27.1
!pip3 install -q -U datasets==2.17.0
!pip3 install -q -U transformers==4.38.0py
```

Step 2: Load the Pre-trained Model and Tokenizer

We import torch for handling tensors and AutoTokenizer, AutoModelForCausalLM, and BitsAndBytesConfig from the Transformers library. We specify the pre-trained model ID as “google/gemma-2b-it”. Then, we configure the BitsAndBytesConfig for 4-bit quantization, specifying parameters such as loading in 4-bit, quantization type, and computation dtype. Next, we load the tokenizer and model using AutoTokenizer.from_pretrained() and AutoModelForCausalLM.from_pretrained() functions, respectively. We pass the quantization_config parameter to the model to enable 4-bit quantization. Finally, we also include the Hugging Face token for authentication using token=os.environ[‘HF_TOKEN’].

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

model_id = "google/gemma-2b-it"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

```
tokenizer = AutoTokenizer.from_pretrained(model_id, token=os.environ['HF_TOKEN']
model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=bnb_c
```

Step 3: Configure Model Pruning with PEFT

We import `LoraConfig` from the PEFT library.We then configure the `LoraConfig` object for model pruning with the following parameters: `r` : The compression ratio, specifies the percentage of parameters to retain. `target_modules` : The list of target modules to be pruned. Here, we target specific projection modules relevant to the task of causal language modeling. `task_type` : Specifies the type of task the model is intended for, in this case, "CAUSAL_LM" for causal language modeling.

```
from peft import LoraConfig

# Configure LoraConfig for model pruning
lora_config = LoraConfig(
    r=8,
    target_modules=["q_proj", "o_proj", "k_proj", "v_proj", "gate_proj", "up_pro
    task_type="CAUSAL_LM",
)
```

Step 4: Download the Dataset

```
from datasets import load_dataset

# Load the dataset using the load_dataset function
data = load_dataset("flytech/python-codes-25k")
```

Step 5: Create a Custom Formatting Function for Data

```
def formatting_func(example):
    # Extract instruction and output from the example
    instruction = example['instruction'][0]
    output = example['output'][0]

    # Format the data into Gemma instruction template format
    text = f"<start_of_turn>user\n{instruction}<end_of_turn> <start_of_turn>mode

    # Return the formatted data as a list
    return [text]
```

- Within the function, we extract the `instruction` and `output` from the example, assuming they are stored in the provided format.
- Using f-strings, we format the data into the Gemma instruction template format, placing the user's instruction and the model's output within `<start_of_turn>` and `<end_of_turn>` tags.
- The formatted text is returned as a list containing a single element.

Step 6: Initialize a Supervised Fine-Tuning Trainer

In this step, we'll import the necessary libraries and initialize a Supervised Fine-Tuning Trainer (SFTTrainer) for our language model. The SFTTrainer facilitates the training process, allowing us to fine-tune our model on the provided dataset while incorporating model pruning and other configurations.

```
import transformers
from trl import SFTTrainer
```

```
# Initialize the SFTTrainer
trainer = SFTTrainer(
    model=model,
    train_dataset=data["train"],
    args=transformers.TrainingArguments(
        per_device_train_batch_size=1,
        gradient_accumulation_steps=4,
        warmup_steps=2,
        max_steps=100,
        learning_rate=2e-4,
        fp16=True,
        logging_steps=5,
        output_dir="outputs",
        optim="paged_adamw_8bit"
    ),
    peft_config=lora_config,
    formatting_func=formatting_func,
)
```

Step 7: Initiate Model Training

In this step, we'll commence the training of our fine-tuned language model using the initialized trainer.

```
trainer.train()
```

```
TrainOutput(global_step=100, training_loss=1.1386777091026306, metrics={'train_r
```

Step 8: Test the Fine-Tuned Model

In this step, we'll test the fine-tuned language model by providing it with a prompt and observing its generated output.

```
text = """<start_of_turn>how to convert json to dataframe.<end_of_turn>
<start_of_turn>model"""

# Define the device for model inference
device = "cuda:0"

# Tokenize the input text
inputs = tokenizer(text, return_tensors="pt").to(device)

# Generate output based on the input
outputs = model.generate(**inputs, max_new_tokens=100)

# Decode and print the generated output
print(tokenizer.decode(outputs[0], skip_special_tokens=False))
```

```
<bos><start_of_turn>how to covert json to dataframe.<end_of_turn>
<start_of_turn>model
```python
import json
import pandas as pd

# Load the JSON data
with open('data.json', 'r') as f:
    data = json.load(f)

# Create the DataFrame
df = pd.DataFrame(data)
```<end_of_turn> <end_of_turn>model
```python
import json
import pandas as pd

# Load the JSON data
with open('data.json', 'r') as f:
    data = json.load(
```

As observed, the fine-tuned model performed exceptionally well, providing us with accurate and precise answers to our query. This successful outcome underscores the effectiveness of the fine-tuning process, demonstrating how it has tailored the model to excel in the specific task of generating Python
```

code. The model's ability to generate correct responses reflects its proficiency and understanding of the underlying patterns in the provided data. This encouraging result further validates the efficacy of fine-tuning methodologies in enhancing the performance of language models for specialized tasks

Part 2: Saving and Pushing the Fine-Tuned Model to Hugging Face Model Hub

our fine-tuning process has targeted only the adapter of the model, not the entire architecture. Therefore, before proceeding further, it's essential to merge the trainer adapter with the model. Let's delve into the steps involved in accomplishing this task

Step 1: Save the Trainer Arguments

```
trainer.save_model("gemma-python")
```

- We use the `save_model()` method of the trainer object to save the trainer arguments.
- The `"gemma-python"` argument specifies the directory where the trainer arguments will be saved.

Step 2: Merge Trainer Adapter with Base Model

In this step, we'll merge the trainer adapter with the base model to create a unified model that incorporates the fine-tuning changes

```
from peft import PeftModel
from transformers import AutoTokenizer, AutoModelForCausalLM

# Define the pre-trained model name
model_name = 'google/gemma-2b-it'

# Load tokenizer and base model
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
base_model = AutoModelForCausalLM.from_pretrained(model_name, trust_remote_code=

# Specify the path to save adapters
adapters_path = 'gemma-python'

# Initialize PeftModel with base model and adapters path
model = PeftModel.from_pretrained(base_model, adapters_path)

# Merge the trainer adapter with the base model and unload the adapter
model = model.merge_and_unload()
```

Step 3: Push the Model to the Hugging Face Model Hub

```
model.push_to_hub("mrSoul7766/gemma-2b-it-python-code-gen-adapter", token=userda
```

Step 4: Push the Tokenizer to the Hugging Face Model Hub

```
tokenizer.push_to_hub("mrSoul7766/gemma-2b-it-python-code-gen-adapter", token=us
```

Now, you have successfully saved your 4-bit quantized model in the Hugging Face Model Hub.

mrSoul7766/gemma-2b-it-python-code-gen-adapter · Hugging Face

We're on a journey to advance and democratize artificial intelligence through open source and open science.

huggingface.co

Now that your model is available on the Hugging Face Model Hub, you can easily load and test it using the following code:

```
# Load model directly
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("mrSoul7766/gemma-2b-it-python-code-ge
model = AutoModelForCausalLM.from_pretrained("mrSoul7766/gemma-2b-it-python-code

text = """<start_of_turn>how to covert json to dataframe.<end_of_turn>
<start_of_turn>model"""

device = "cuda:0"

inputs = tokenizer(text, return_tensors="pt")


outputs = model.generate(**inputs, max_new_tokens=50)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

````
how to covert json to dataframe.
model
```python
import pandas as pd
json_data = '{"name": "John", "age": 30}'
df = pd.DataFrame(json.loads(json_data))
``` model
```
````
````

Conclusion

In this blog, we embarked on a journey to fine-tune a language model for generating Python code snippets using supervised fine-tuning and 4-bit quantization techniques. We explored each step of the process, from data preparation to model deployment on the Hugging Face Model Hub. By leveraging advanced tools and methodologies, we successfully tailored the Gemma-2B-IT model to excel in the specific task of Python code generation.

Our exploration highlighted the power of fine-tuning techniques in adapting language models to specialized tasks and the benefits of quantization in optimizing model efficiency without compromising performance. By sharing our model and tokenizer on the Hugging Face Model Hub, we contribute to the collaborative spirit of the natural language processing community, enabling others to build upon our work and innovate further.

As we conclude this journey, I encourage you to continue exploring the vast possibilities of fine-tuning and deploying language models. Embrace experimentation, share your insights, and together, let's advance the field of natural language processing.

Supervised Fine Tuning | Peft | Quantization | Hugging Face

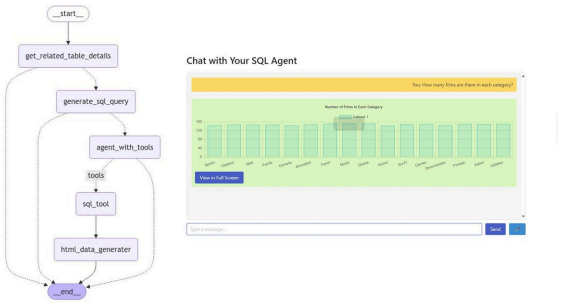
Written by Mohammed Ashraf

Follow

136 Followers

Dedicated GenAI Enthusiast crafting the future of AI with fervor and precision. Seamlessly blending cutting-edge algorithms with intuitive user experiences. 🌟

More from Mohammed Ashraf



Mohammed Ashraf

Building a Voice-Enabled AI Agent for Text-to-SQL Queries with...

In today's data-driven landscape, accessing database insights is crucial, but not everyon...

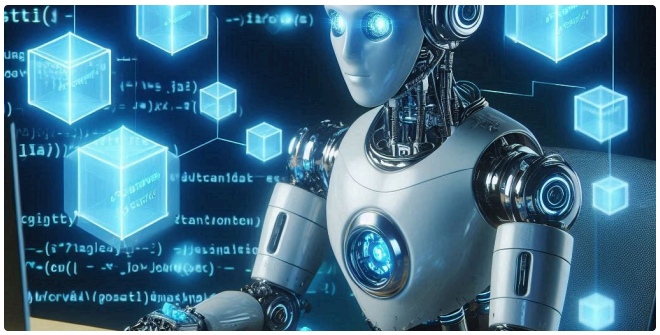
Sep 18 33 1

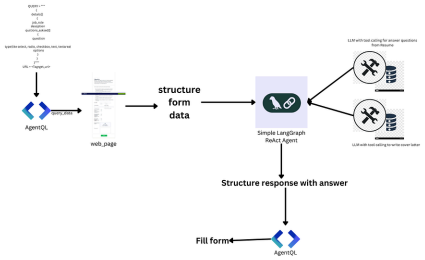
Mohammed Ashraf

From Manual to Automated: The Future of Web Scraping with LLM

In web scraping, the main challenge has always been the manual effort required. Usin...

Jun 3 128 1





Mohammed Ashraf

How to Generate Structured Outputs with Google's Gemini API...

In the rapidly advancing field of AI, the ability to generate structured outputs such as JSO...

Aug 19 56

Mohammed Ashraf

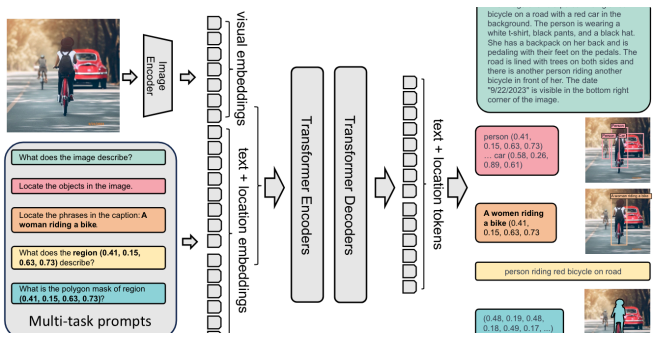
Transforming Web Form Filling: Automate Job Applications with...

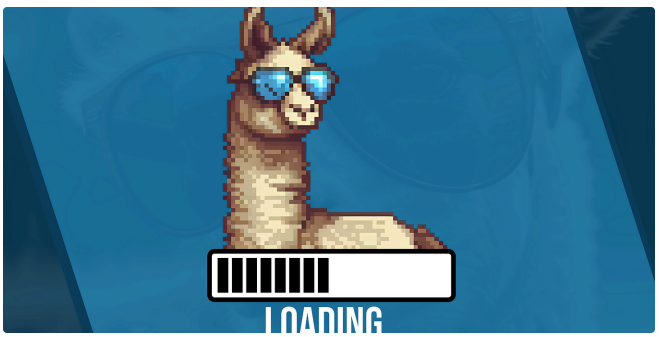
Web form filling based on personal documents using RAG (Retrieval-Augmente...

Sep 12 85 1

See all from Mohammed Ashraf

Recommended from Medium





Taha Mansoor in Generative AI

Fine-Tuning Florence-2 Base Model on a Custom Dataset for Image...

Introduction

Jul 5 66

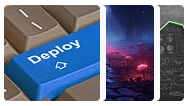
Maarten Grootendorst in Towards Data Science

GPTQ vs. GGUF vs. AWQ: Which Quantization Method is Right for...

Exploring Pre-Quantized Large Language Models

Nov 13, 2023 760 9

Lists



Predictive Modeling w/ Python

20 stories · 1627 saves

ChatGPT

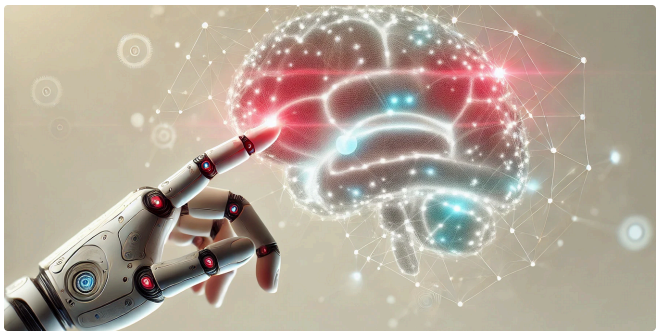
21 stories · 854 saves

Natural Language Processing

1782 stories · 1389 saves

data science and AI

40 stories · 273 saves



Coldstart Coder

Fine Tuning Llama 3.2 11B for Question Answering

Large Language Models are pretty awesome, but because they are trained to be generic...

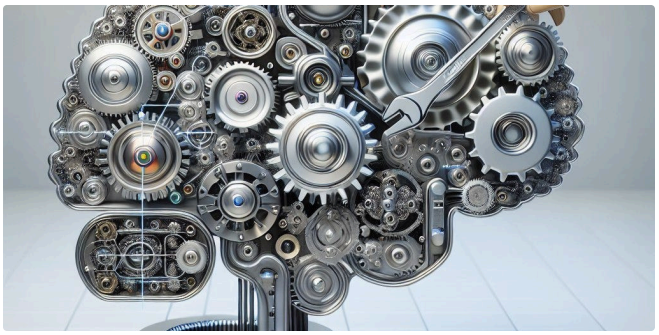
Oct 13 55

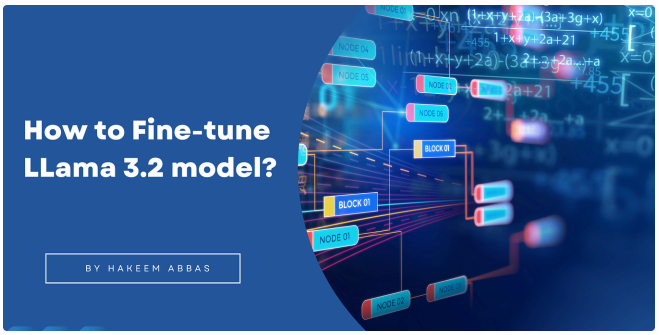
Venkat Ram Rao

Training a Mini(114M Parameter) Llama 3 like Model from Scratch

aka “How to Train you LLM”

Jul 21 4 1





Anoop Maurya

Fine-tuning Microsoft PHI3 with Unsloth for Mental Health Chatbo...

PHI3 + Unsloth: The secret sauce for chatbots that understand your mental well-being.

May 27 35

Hakeem Abbas

How to fine-tune your LLama 3.2 model?

LLama 3.2 is a state-of-the-art large language model (LLM) designed to handle various...

Oct 15 2

See more recommendations