22.04.24

OPENCAMPUS.sh

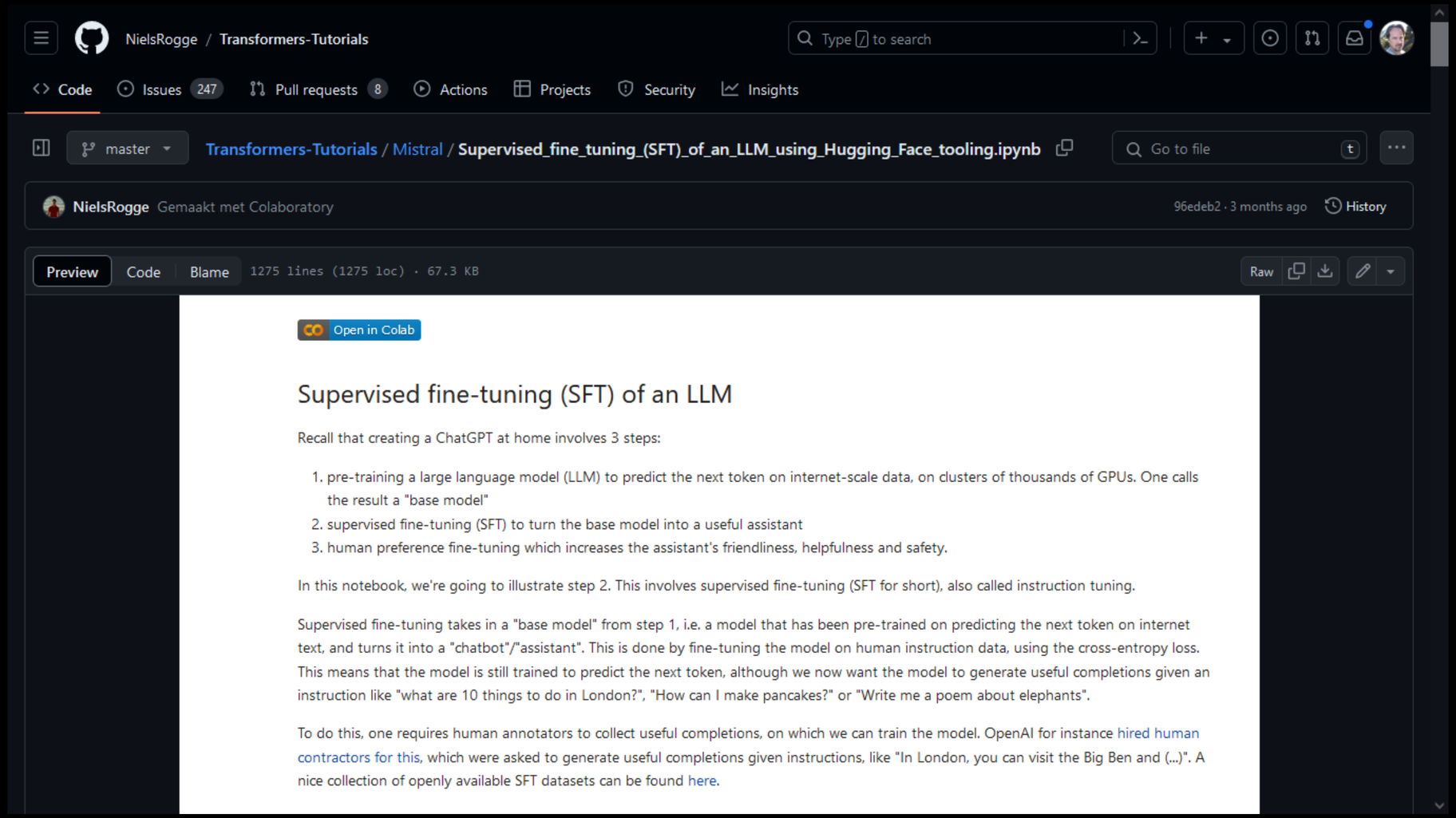
Fine-Tuning and Deployment of Large Language Models

CHARACTERISTICS OF LLM FINE-TUNING

- News
- Creating your own ChatGPT: Supervised fine-tuning (SFT)
- Pydantic and Instructor
- Project Discussions
- Tasks until next week

NEWS

CREATING YOUR OWN CHATGPT: SUPERVISED FINE-TUNING (SFT)



PYDANTIC

```python
from datetime import datetime

from pydantic import BaseModel, PositiveInt


class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]


external_data = {
    'id': 123,
    'signup_ts': '2019-06-01 12:22',
    'tastes': {
        'wine': 9,
        b'cheese': 7,
        'cabbage': '1',
    },
}
```

```python
user = User(**external_data)

print(user.id)
#> 123
print(user.model_dump())
"""
{
    'id': 123,
    'name': 'John Doe',
    'signup_ts': datetime.datetime(2019, 6, 1, 12, 22),
    'tastes': {'wine': 9, 'cheese': 7, 'cabbage': 1},
}
"""

```

PYDANTIC‘S BASEMODEL CLASS

- **Data Validation**
  When you create an instance of a BaseModel, Pydantic validates the input data based on the field types declared in the model. This process ensures that the data conforms to specified formats and constraints.

- **Data Parsing**
  Automatically converts or parses incoming data to the declared types. For example, if a field is declared as int, but the input is a string that can be parsed into an integer, Pydantic will handle this conversion.

- **Data Serialization**
  Models can easily be converted to dictionaries, JSON, and other formats, supporting clean serialization of complex structures.

- **Rich Type Annotations**
  Supports advanced type hints, including List, Dict, Optional, and even custom types, making it powerful for use in both simple and complex data structures.

- **Immutability Option**
  You can make models immutable so that their attributes can't be changed once they are set.

- **Default Values & Validation**
  You can define default values for fields and add additional validation through the use of custom validator methods.

PYDANTIC VALIDATION FAIL

```python
# continuing the above example...

from pydantic import ValidationError


class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]


external_data = {'id': 'not an int', 'tastes': {}}

```

```python
try:
    User(**external_data)
except ValidationError as e:
    print(e.errors())
    """
    [
        {
            'type': 'int_parsing',
            'loc': ('id',),
            'msg': 'Input should be a valid integer, unable to pars
            'input': 'not an int',
            'url': 'https://errors.pydantic.dev/2/v/int_parsing',
        },
        {
            'type': 'missing',
            'loc': ('signup_ts',),
            'msg': 'Field required',
            'input': {'id': 'not an int', 'tastes': {}},
            'url': 'https://errors.pydantic.dev/2/v/missing',
        },
    ]
    """
```

Instructor: Structured LLM Outputs

Instructor is a Python library that makes it a breeze to work with structured outputs from large language models (LLMs). Built on top of Pydantic, it provides a simple, transparent, and user-friendly API to manage validation, retries, and streaming responses. Get ready to supercharge your LLM workflows!

Key Features

- **Response Models:** Specify Pydantic models to define the structure of your LLM outputs
- **Retry Management:** Easily configure the number of retry attempts for your requests
- **Validation:** Ensure LLM responses conform to your expectations with Pydantic validation
- **Streaming Support:** Work with Lists and Partial responses effortlessly
- **Flexible Backends:** Seamlessly integrate with various LLM providers beyond OpenAI

```python
import instructor
from pydantic import BaseModel
from openai import OpenAI


# Define your desired output structure
class UserInfo(BaseModel):
    name: str
    age: int


# Patch the OpenAI client
client = instructor.from_openai(OpenAI())

# Extract structured data from natural language
user_info = client.chat.completions.create(
    model="gpt-3.5-turbo",
    response_model=UserInfo,
    messages=[{"role": "user", "content": "John Doe is 30 years old."}],
)

print(user_info.name)
#> John Doe
print(user_info.age)
#> 30
```

PROJECT DISCUSSIONS

TASKS UNTIL NEXT WEEK

- Who is doing the news section?

TASKS UNTIL NEXT WEEK

- Do a “literature review” for your project.
- Watch the first two chapters of LLM Engineering: Structured Outputs (“Asking LLMs for Structured Data” and “Prompting LLMs”)
- Note a least one question on the videos above.