"""

[**openai-cookbook**](https://github.com/openai/openai-cookbook/tree/main)//[examples](https://github.com/openai/openai-cookbook/tree/main/examples)//[fine-tuned\_qa](https://github.com/openai/openai-cookbook/tree/main/examples/fine-tuned_qa)

# answers\_with\_ft.py

Note: To answer questions based on text documents, we recommend the procedure in

[Question Answering using Embeddings](https://github.com/openai/openai-cookbook/blob/main/examples/Question\_answering\_using\_embeddings.ipynb).

Some of the code below may rely on [deprecated API endpoints]

(<https://github.com/openai/openai-Cookbook/tree/main/transition_guides_for_deprecated_API_endpoints>).

"""

import argparse

import openai

def create\_context(

question, search\_file\_id, max\_len=1800, search\_model="ada", max\_rerank=10

):

"""

Create a context for a question by finding the most similar context from the search file.

:param question: The question

:param search\_file\_id: The file id of the search file

:param max\_len: The maximum length of the returned context (in tokens)

:param search\_model: The search model to use

:param max\_rerank: The maximum number of reranking

:return: The context

"""

results = openai.Engine(search\_model).search(

search\_model=search\_model,

query=question,

max\_rerank=max\_rerank,

file=search\_file\_id,

return\_metadata=True,

)

returns = []

cur\_len = 0

for result in results["data"]:

cur\_len += int(result["metadata"]) + 4

if cur\_len > max\_len:

break

returns.append(result["text"])

return "\n\n###\n\n".join(returns)

def answer\_question(

search\_file\_id="<SEARCH\_FILE\_ID>",

fine\_tuned\_qa\_model="<FT\_QA\_MODEL\_ID>",

question="Which country won the European Football championship in 2021?",

max\_len=1800,

search\_model="ada",

max\_rerank=10,

debug=False,

stop\_sequence=["\n", "."],

max\_tokens=100,

):

"""

Answer a question based on the most similar context from the search file, using your fine-tuned model.

:param question: The question

:param fine\_tuned\_qa\_model: The fine tuned QA model

:param search\_file\_id: The file id of the search file

:param max\_len: The maximum length of the returned context (in tokens)

:param search\_model: The search model to use

:param max\_rerank: The maximum number of reranking

:param debug: Whether to output debug information

:param stop\_sequence: The stop sequence for Q&A model

:param max\_tokens: The maximum number of tokens to return

:return: The answer

"""

context = create\_context(

question,

search\_file\_id,

max\_len=max\_len,

search\_model=search\_model,

max\_rerank=max\_rerank,

)

if debug:

print("Context:\n" + context)

print("\n\n")

try:

# fine-tuned models requires model parameter, whereas other models require engine parameter

model\_param = (

{"model": fine\_tuned\_qa\_model}

if ":" in fine\_tuned\_qa\_model

and fine\_tuned\_qa\_model.split(":")[1].startswith("ft")

else {"engine": fine\_tuned\_qa\_model}

)

response = openai.Completion.create(

prompt=f"Answer the question based on the context below\n\nText: {context}\n\n---\n\nQuestion: {question}\nAnswer:",

temperature=0,

max\_tokens=max\_tokens,

top\_p=1,

frequency\_penalty=0,

presence\_penalty=0,

stop=stop\_sequence,

\*\*model\_param,

)

return response["choices"][0]["text"]

except Exception as e:

print(e)

return ""

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser(

description="Rudimentary functionality of the answers endpoint with a fine-tuned Q&A model.",

formatter\_class=argparse.ArgumentDefaultsHelpFormatter,

)

parser.add\_argument(

"--search\_file\_id", help="Search file id", required=True, type=str

)

parser.add\_argument(

"--fine\_tuned\_qa\_model", help="Fine-tuned QA model id", required=True, type=str

)

parser.add\_argument(

"--question", help="Question to answer", required=True, type=str

)

parser.add\_argument(

"--max\_len",

help="Maximum length of the returned context (in tokens)",

default=1800,

type=int,

)

parser.add\_argument(

"--search\_model", help="Search model to use", default="ada", type=str

)

parser.add\_argument(

"--max\_rerank",

help="Maximum number of reranking for the search",

default=10,

type=int,

)

parser.add\_argument(

"--debug", help="Print debug information (context used)", action="store\_true"

)

parser.add\_argument(

"--stop\_sequence",

help="Stop sequences for the Q&A model",

default=["\n", "."],

nargs="+",

type=str,

)

parser.add\_argument(

"--max\_tokens",

help="Maximum number of tokens to return",

default=100,

type=int,

)

args = parser.parse\_args()

response = answer\_question(

search\_file\_id=args.search\_file\_id,

fine\_tuned\_qa\_model=args.fine\_tuned\_qa\_model,

question=args.question,

max\_len=args.max\_len,

search\_model=args.search\_model,

max\_rerank=args.max\_rerank,

debug=args.debug,

stop\_sequence=args.stop\_sequence,

max\_tokens=args.max\_tokens,

)

print(f"Answer:{response}")