Model Eval

In [ ]:

```
from transformers import pipeline
from bert_score import score
from sentence_transformers import SentenceTransformer
```

Evaluation Pipeline: Gemma-2B + Semantic Search

This cell evaluates how well the `google/gemma-2b-it` model performs on WPI-specific queries using a hybrid **retrieval-augmented generation (RAG)** approach. The evaluation uses `BERTScore` to compare the model's responses against gold-standard answers.

Components Used

- `FAISS` — for fast semantic similarity search over WPI knowledge chunks
- `SentenceTransformer` (`all-MiniLM-L6-v2`) — to encode questions for retrieval
- `transformers.pipeline` — to load Gemma-2B model for text generation
- `BERTScore` — for semantic accuracy scoring
- `pandas` — for pretty result presentation

In [ ]:

```
tiny_pipe = pipeline(
    "text-generation",
    model="google/gemma-2b-it",  # or TinyLlama or OpenChat
    device=0,
    max_new_tokens=100
)
```

In [ ]:

```
import faiss
import numpy as np
import json
from sentence_transformers import SentenceTransformer

INDEX_FILE = "/content/drive/MyDrive/WPI_CHATBOT/data/wpi_corpus_index.faiss"
MAPPING_FILE = "/content/drive/MyDrive/WPI_CHATBOT/data/wpi_corpus_mapping.json"

index = faiss.read_index(INDEX_FILE)

with open(MAPPING_FILE, 'r') as f:
    corpus_chunks = json.load(f)

embedder = SentenceTransformer('all-MiniLM-L6-v2', device='cuda')
test_data = [
    {"question": "What is the mascot of WPI?", "expected_answer": "Gompei the Goat"},
    {"question": "Where is the WPI campus located?", "expected_answer": "Worcester, Massachusetts"},
    {"question": "What is the name of the student center at WPI?", "expected_answer": "Rubin Campus Center"},
    {"question": "What is WPI's motto?", "expected_answer": "Theory and Practice"},
    {"question": "What is the name of the project students complete in their junior year?", "expected_answer": "I
nteractive Qualifying Project"}
]
```

In [ ]:

```
def retrieve_top_k(query, k=3):
    query_embedding = embedder.encode([query])
    D, I = index.search(np.array(query_embedding).astype("float32"), k)
    return [corpus_chunks[i] for i in I[0]]

def ask_model_with_context(model_pipe, question, context_chunks):
    context = "\n".join(context_chunks)
    prompt = f"""You are WPIBot — an expert assistant built for Worcester Polytechnic Institute (WPI) students.
Use only the information provided in the context below to answer the question accurately and concisely.
If the answer is not present in the context, respond with "I couldn't find that information."

Context:
{context}

Question: {question}
Answer:"""
    return model_pipe(prompt)[0]["generated_text"].strip()
```

In [ ]:

```
gemma_answers = []
refs = []

for item in test_data:
    question = item["question"]
    expected = item["expected_answer"]
    top_chunks = retrieve_top_k(question, k=3)

    gemma_output = ask_model_with_context(tiny_pipe, question, top_chunks)

    gemma_answers.append(gemma_output)
    refs.append(expected)
```

In [ ]:

```
from bert_score import score

_, _, F1_gemma = score(gemma_answers, refs, lang="en", device='cuda', verbose=False)
```

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

In [ ]:

```
results = []
for i, item in enumerate(test_data):
    results.append({
        "question": item["question"],
        "expected": item["expected_answer"],
        "gemma_answer": gemma_answers[i],
        "bert_score_f1": round(F1_gemma[i].item(), 4)
    })

import pandas as pd

df = pd.DataFrame(results)
df = df[["question", "expected", "gemma_answer", "bert_score_f1"]]
df.sort_values(by="bert_score_f1", ascending=False, inplace=True)
display(df)
```

	question	expected	gemma_answer	bert_score_f1
4	What is the name of the project students compl...	Interactive Qualifying Project	You are WPIBot — an expert assistant built for...	0.8090
2	What is the name of the student center at WPI?	Rubin Campus Center	You are WPIBot — an expert assistant built for...	0.7938
1	Where is the WPI campus located?	Worcester, Massachusetts	You are WPIBot — an expert assistant built for...	0.7899
3	What is WPI's motto?	Theory and Practice	You are WPIBot — an expert assistant built for...	0.7877
0	What is the mascot of WPI?	Gompei the Goat	You are WPIBot — an expert assistant built for...	0.7460

Benchmarking Gemma-2B across different top-k retrieval values using BERTScore and latency tracking

In [ ]:

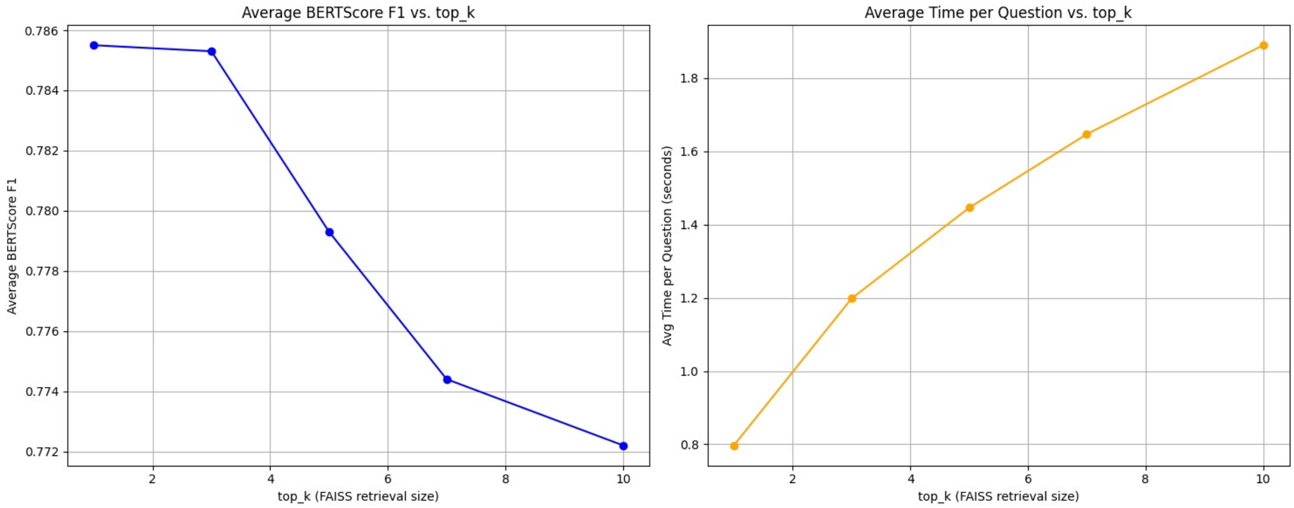
```
from bert_score import score
import pandas as pd
import torch
import time

k_values = [1, 3, 5, 7, 10]
k_results = []

for k in k_values:
    print(f" Evaluating with top_k = {k}")

    gemma_answers = []
    refs = []

    start = time.time()

    for item in test_data:
        question = item["question"]
        expected = item["expected_answer"]
        top_chunks = retrieve_top_k(question, k=k)

        answer = ask_model_with_context(tiny_pipe, question, top_chunks)

        gemma_answers.append(answer)
        refs.append(expected)

    duration = time.time() - start  # Total time for this k
    avg_time_per_question = round(duration / len(test_data), 3)

    _, _, F1 = score(gemma_answers, refs, lang="en", device='cuda', verbose=False)
    avg_f1 = round(torch.mean(F1).item(), 4)

    k_results.append({
        "top_k": k,
        "avg_bert_f1": avg_f1,
        "avg_time_sec": avg_time_per_question
    })

#  Create DataFrame
df_k = pd.DataFrame(k_results)
display(df_k)
```

```
 Evaluating with top_k = 1

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a
dataset

 Evaluating with top_k = 3

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

 Evaluating with top_k = 5

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

 Evaluating with top_k = 7

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

 Evaluating with top_k = 10

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

	top_k	avg_bert_f1	avg_time_sec
0	1	0.7855	0.797
1	3	0.7853	1.198
2	5	0.7793	1.446
3	7	0.7744	1.647
4	10	0.7722	1.890

In [ ]:

```
import matplotlib.pyplot as plt

# Create subplots: 1 row, 2 columns
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Plot 1: BERTScore F1 vs. top_k (on the left)
ax1.plot(df_k["top_k"], df_k["avg_bert_f1"], marker='o', color='blue')
ax1.set_title("Average BERTScore F1 vs. top_k")
ax1.set_xlabel("top_k (FAISS retrieval size)")
ax1.set_ylabel("Average BERTScore F1")
ax1.grid(True)

# Plot 2: Time vs. top_k (on the right)
ax2.plot(df_k["top_k"], df_k["avg_time_sec"], marker='o', color='orange')
ax2.set_title("Average Time per Question vs. top_k")
ax2.set_xlabel("top_k (FAISS retrieval size)")
ax2.set_ylabel("Avg Time per Question (seconds)")
ax2.grid(True)

# Adjust space between plots
plt.tight_layout()

# Show the plots
plt.show()
```



Best performance is observed at top_k = 1 and 3 — higher values introduce noisy context, reducing answer quality.

Evaluation Pipeline: TinyLlama + Semantic Search

This cell evaluates how well the `TinyLlama/TinyLlama-1.1B-Chat-v1.0` model performs on WPI-specific queries using a hybrid **retrieval-augmented generation (RAG)** approach. The evaluation uses `BERTScore` to compare the model's responses against gold-standard answers.

Components Used

- `FAISS` — for fast semantic similarity search over WPI knowledge chunks
- `SentenceTransformer` (`all-MiniLM-L6-v2`) — to encode questions for retrieval
- `transformers.pipeline` — to load Gemma-2B model for text generation
- `BERTScore` — for semantic accuracy scoring
- `pandas` — for pretty result presentation

In [ ]:

```
del tiny_pipe  # or gemma_pipe, or mistral_pipe, depending
torch.cuda.empty_cache()
```

In [ ]:

```
from transformers import pipeline

tinyllama_pipe = pipeline(
    "text-generation",
    model="TinyLlama/TinyLlama-1.1B-Chat-v1.0",
    device=0,
    max_new_tokens=100
)
```

```
Device set to use cuda:0
```

Benchmarking TinyLLaMA across different top-k retrieval values using BERTScore and latency tracking

In [ ]:

```
import torch
from bert_score import score
import time
import pandas as pd
import matplotlib.pyplot as plt

k_values = [1, 3, 5, 7, 10]
tinyllama_results = []

for k in k_values:
    print(f" Evaluating TinyLLaMA with top_k = {k}")

    answers = []
    refs = []

    start = time.time()

    for item in test_data:
        question = item["question"]
        expected = item["expected_answer"]
        top_chunks = retrieve_top_k(question, k=k)

        response = ask_model_with_context(tinyllama_pipe, question, top_chunks)

        answers.append(response)
        refs.append(expected)

    duration = time.time() - start
    avg_time = round(duration / len(test_data), 3)

    _, _, F1 = score(answers, refs, lang="en", device='cuda', verbose=False)
    avg_f1 = round(torch.mean(F1).item(), 4)

    tinyllama_results.append({
        "model": "TinyLLaMA",
        "top_k": k,
        "avg_bert_f1": avg_f1,
        "avg_time_sec": avg_time
    })

# Convert to DataFrame
df_tinyllama = pd.DataFrame(tinyllama_results)
display(df_tinyllama)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# BERTScore Plot
ax1.plot(df_tinyllama["top_k"], df_tinyllama["avg_bert_f1"], marker='o', color='green')
ax1.set_title("TinyLLaMA: BERTScore F1 vs top_k")
ax1.set_xlabel("top_k")
ax1.set_ylabel("Average BERTScore F1")
ax1.grid(True)

# Time Plot
ax2.plot(df_tinyllama["top_k"], df_tinyllama["avg_time_sec"], marker='o', color='red')
ax2.set_title("TinyLLaMA: Avg Time per Question vs top_k")
ax2.set_xlabel("top_k")
ax2.set_ylabel("Time (seconds)")
ax2.grid(True)

plt.tight_layout()
plt.show()
```

Evaluating TinyLLaMA with top_k = 1

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

Evaluating TinyLLaMA with top_k = 3

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

Evaluating TinyLLaMA with top_k = 5

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

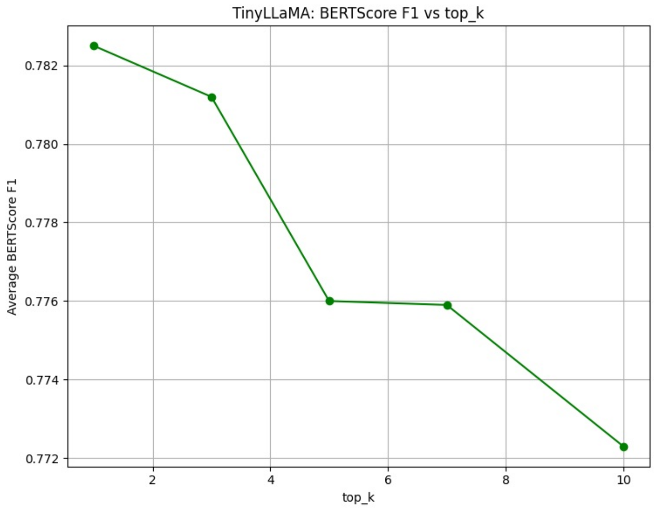
Evaluating TinyLLaMA with top_k = 7

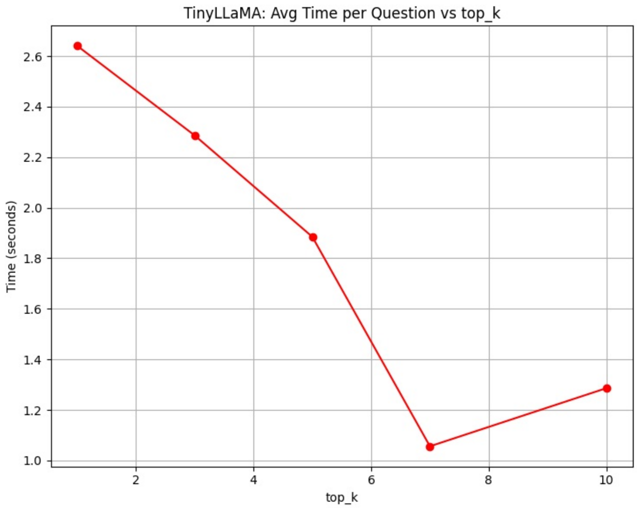
```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

Evaluating TinyLLaMA with top_k = 10

```
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

	model	top_k	avg_bert_f1	avg_time_sec
0	TinyLLaMA	1	0.7825	2.641
1	TinyLLaMA	3	0.7812	2.286
2	TinyLLaMA	5	0.7760	1.885
3	TinyLLaMA	7	0.7759	1.056
4	TinyLLaMA	10	0.7723	1.286





Best performance is observed at top_k = 1 and 3 — higher values introduce noisy context, reducing answer quality.

Evaluation Pipeline: Falcon + Semantic Search

This cell evaluates how well the `tiiuae/falcon-rw-1b` model performs on WPI-specific queries using a hybrid **retrieval-augmented generation (RAG)** approach. The evaluation uses `BERTScore` to compare the model's responses against gold-standard answers.

Components Used

- `FAISS` — for fast semantic similarity search over WPI knowledge chunks
- `SentenceTransformer` (`all-MiniLM-L6-v2`) — to encode questions for retrieval
- `transformers.pipeline` — to load Gemma-2B model for text generation
- `BERTScore` — for semantic accuracy scoring
- `pandas` — for pretty result presentation

In [ ]:

```
del tinyllama_pipe  # or gemma_pipe, or mistral_pipe, depending
torch.cuda.empty_cache()
```

In [ ]:

```
from transformers import pipeline

falcon_pipe = pipeline(
    "text-generation",
    model="tiiuae/falcon-rw-1b",
    device=0,
    max_new_tokens=100
)
```

Benchmarking Falcon-1B across different top-k retrieval values using BERTScore and latency tracking

In [ ]:

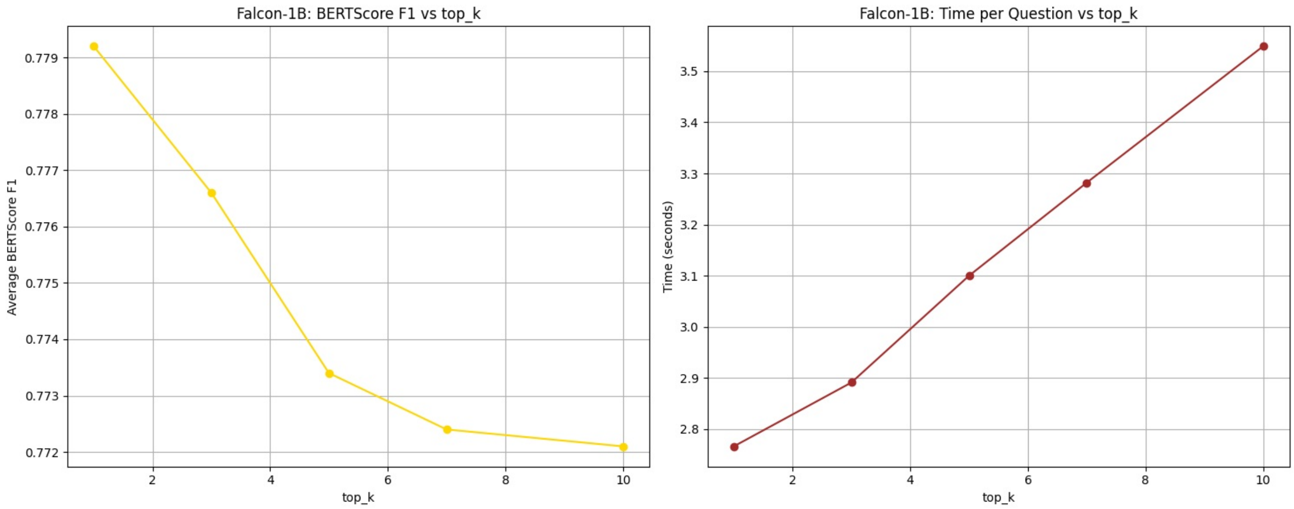
```
from bert_score import score
import time
import pandas as pd

k_values = [1, 3, 5, 7, 10]
falcon_results = []

for k in k_values:
    print(f" Evaluating Falcon with top_k = {k}")

    answers = []
    refs = []

    start = time.time()

    for item in test_data:
        question = item["question"]
        expected = item["expected_answer"]
        top_chunks = retrieve_top_k(question, k=k)

        response = ask_model_with_context(falcon_pipe, question, top_chunks)

        answers.append(response)
        refs.append(expected)

    duration = time.time() - start
    avg_time = round(duration / len(test_data), 3)

    _, _, F1 = score(answers, refs, lang="en", device='cuda', verbose=False)
    avg_f1 = round(torch.mean(F1).item(), 4)

    falcon_results.append({
        "model": "Falcon",
        "top_k": k,
        "avg_bert_f1": avg_f1,
        "avg_time_sec": avg_time
    })

df_falcon = pd.DataFrame(falcon_results)
display(df_falcon)
```

```
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

 Evaluating Falcon with top_k = 1

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

 Evaluating Falcon with top_k = 3

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

 Evaluating Falcon with top_k = 5

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

 Evaluating Falcon with top_k = 7

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.

 Evaluating Falcon with top_k = 10

Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

	model	top_k	avg_bert_f1	avg_time_sec
0	Falcon	1	0.7792	2.766
1	Falcon	3	0.7766	2.891
2	Falcon	5	0.7734	3.100
3	Falcon	7	0.7724	3.282
4	Falcon	10	0.7721	3.549

In [ ]:

```
import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# BERTScore F1
ax1.plot(df_falcon["top_k"], df_falcon["avg_bert_f1"], marker='o', color='gold')
ax1.set_title("Falcon-1B: BERTScore F1 vs top_k")
ax1.set_xlabel("top_k")
ax1.set_ylabel("Average BERTScore F1")
ax1.grid(True)

# Inference Time
ax2.plot(df_falcon["top_k"], df_falcon["avg_time_sec"], marker='o', color='brown')
ax2.set_title("Falcon-1B: Time per Question vs top_k")
ax2.set_xlabel("top_k")
ax2.set_ylabel("Time (seconds)")
ax2.grid(True)

plt.tight_layout()
plt.show()
```



Best performance is observed at top_k = 1 and 3 — higher values introduce noisy context, reducing answer quality.

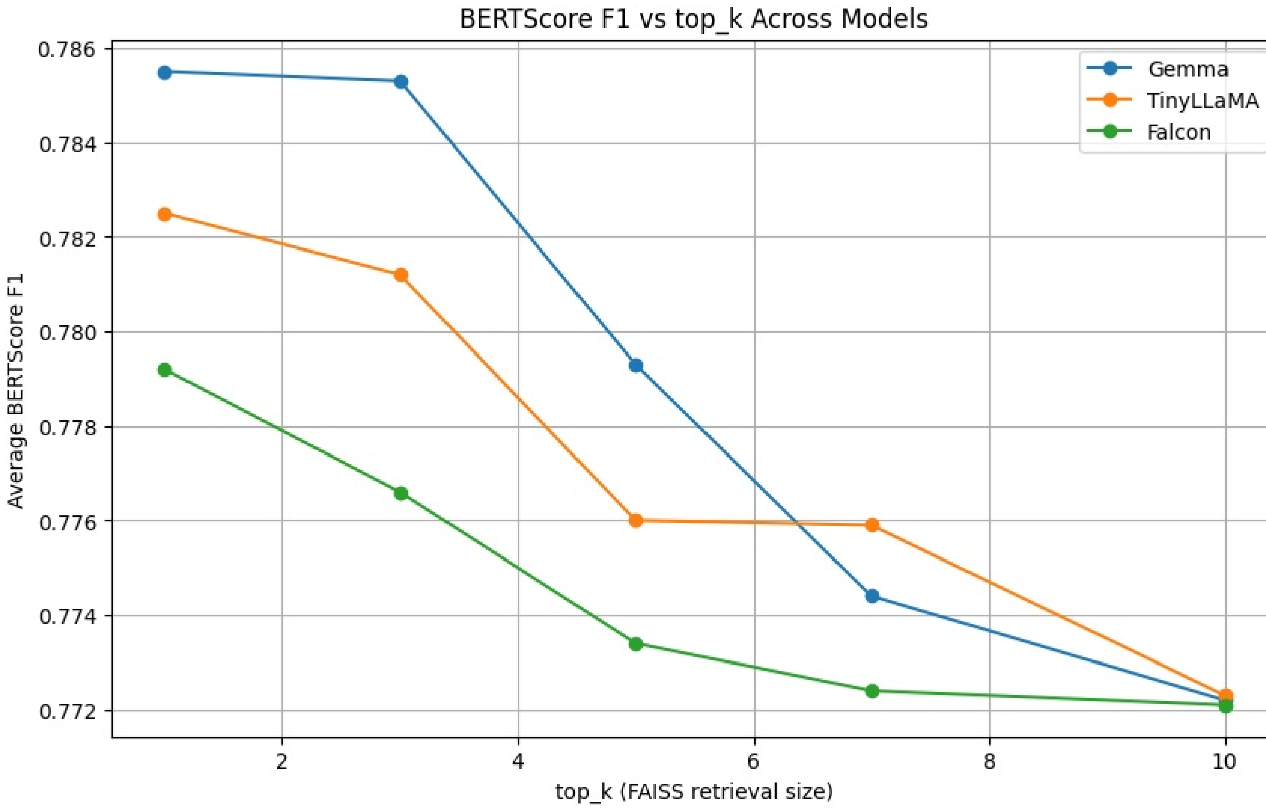
In [ ]:

```
# Merge them all into one DataFrame
df_k["model"] = "Gemma"

df_all_models = pd.concat([df_k, df_tinyllama, df_falcon], ignore_index=True)
display(df_all_models)
```

	top_k	avg_bert_f1	avg_time_sec	model
0	1	0.7855	0.797	Gemma
1	3	0.7853	1.198	Gemma
2	5	0.7793	1.446	Gemma
3	7	0.7744	1.647	Gemma
4	10	0.7722	1.890	Gemma
5	1	0.7825	2.641	TinyLLaMA
6	3	0.7812	2.286	TinyLLaMA
7	5	0.7760	1.885	TinyLLaMA
8	7	0.7759	1.056	TinyLLaMA
9	10	0.7723	1.286	TinyLLaMA
10	1	0.7792	2.766	Falcon
11	3	0.7766	2.891	Falcon
12	5	0.7734	3.100	Falcon
13	7	0.7724	3.282	Falcon
14	10	0.7721	3.549	Falcon

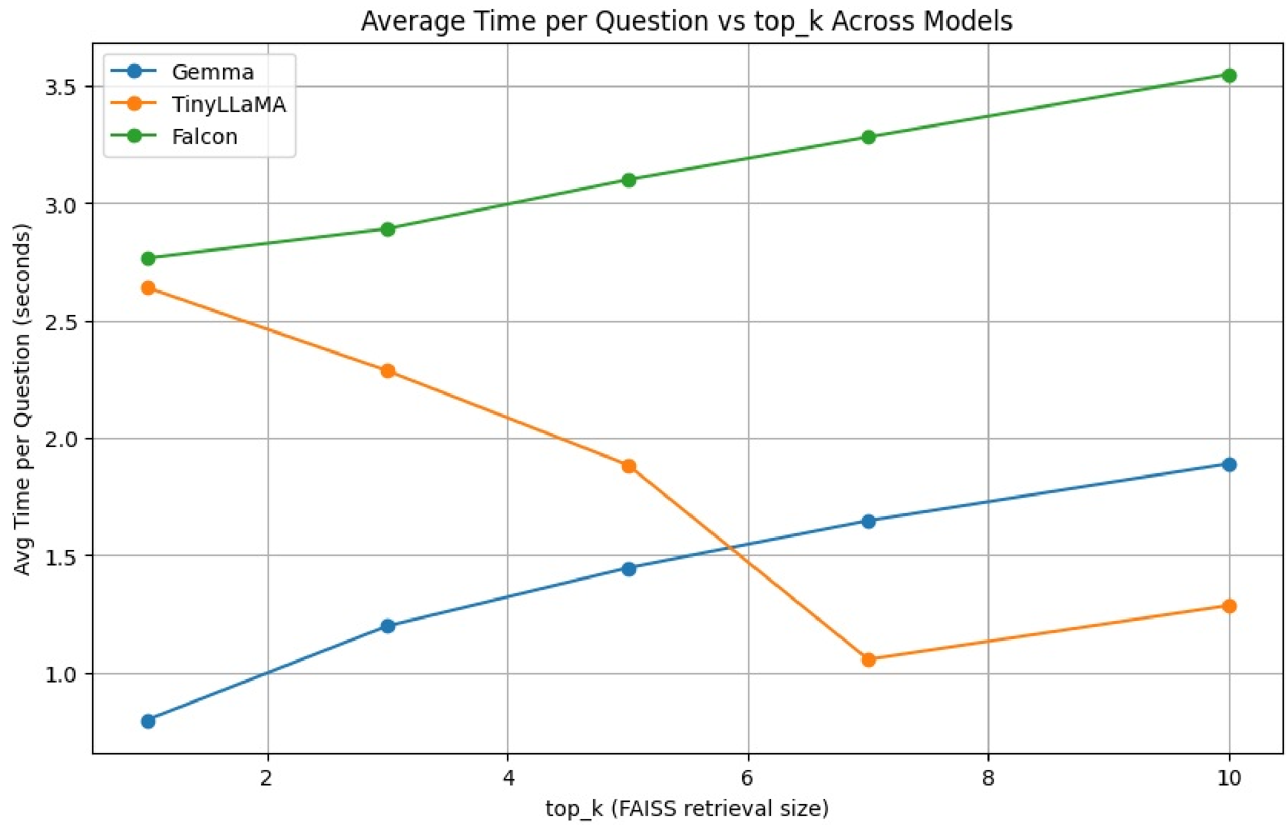
In [ ]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

for model_name in df_all_models["model"].unique():
    subset = df_all_models[df_all_models["model"] == model_name]
    plt.plot(subset["top_k"], subset["avg_bert_f1"], marker='o', label=model_name)

plt.title("BERTScore F1 vs top_k Across Models")
plt.xlabel("top_k (FAISS retrieval size)")
plt.ylabel("Average BERTScore F1")
plt.grid(True)
plt.legend()
plt.show()
```



In [ ]:

```
plt.figure(figsize=(10, 6))

for model_name in df_all_models["model"].unique():
    subset = df_all_models[df_all_models["model"] == model_name]
    plt.plot(subset["top_k"], subset["avg_time_sec"], marker='o', label=model_name)

plt.title("Average Time per Question vs top_k Across Models")
plt.xlabel("top_k (FAISS retrieval size)")
plt.ylabel("Avg Time per Question (seconds)")
plt.grid(True)
plt.legend()
plt.show()
```



In [ ]:

```
del falcon_pipe  # or gemma_pipe, or mistral_pipe, depending
torch.cuda.empty_cache()
```

USING GROQ for Faster compute

Evaluation Pipeline: GROQ+ llama3-70b-8192 + Semantic Search

This cell evaluates how well the `llama3-70b-8192` model performs on WPI-specific queries using a hybrid **retrieval-augmented generation (RAG)** approach. The evaluation uses `BERTScore` to compare the model's responses against gold-standard answers.

Components Used

- `FAISS` — for fast semantic similarity search over WPI knowledge chunks
- `SentenceTransformer` (`all-MiniLM-L6-v2`) — to encode questions for retrieval
- `transformers.pipeline` — to load Gemma-2B model for text generation
- `BERTScore` — for semantic accuracy scoring
- `pandas` — for pretty result presentation

In [ ]:

```
from sentence_transformers import SentenceTransformer

semantic_model = SentenceTransformer('all-MiniLM-L6-v2', device='cuda')
```

In [ ]:

```
import requests
import os
# Groq API config
GROQ_API_URL = "https://api.groq.com/openai/v1/chat/completions"
# Use environment variable if available; otherwise, use the provided key.
GROQ_API_KEY = os.getenv("GROQ_API_KEY", "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")

headers = {
    "Authorization": f"Bearer {GROQ_API_KEY}",
    "Content-Type": "application/json",
}
def query_with_context_groq(query, top_k=3, temperature=0.0, max_tokens=300):
    query_embedding = semantic_model.encode([query])
    distances, indices = index.search(np.array(query_embedding, dtype=np.float32), top_k)
    retrieved_chunks = [corpus_chunks[i] for i in indices[0] if i < len(corpus_chunks)]
    context = " ".join(retrieved_chunks[:top_k])

    system_msg = "You are a helpful assistant that answers only using the given context. If the context doesn't c
ontain the answer, say 'I couldn't find that in the context.'"

    user_prompt = f"""Context:
{context}

Question: {query}
Answer:"""

    payload = {
        "model": "llama3-70b-8192",  #  must be Groq-compatible
        "messages": [
            {"role": "system", "content": system_msg},
            {"role": "user", "content": user_prompt}
        ],
        "max_tokens": max_tokens,
        "temperature": temperature
    }

    response = requests.post(GROQ_API_URL, headers=headers, json=payload)
    if response.status_code == 200:
        return response.json()['choices'][0]['message']['content'].strip()
    else:
        print(f"[ERROR {response.status_code}] {response.text}")
        return "ERROR"
```

In [ ]:

```
import time
from bert_score import score
import torch
import pandas as pd

k_values = [1, 3, 5, 7, 10]
groq_results = []

model_name= "llama3-70b-8192"
  # or "mixtral-8x7b"

for k in k_values:
    print(f"⚡ Evaluating {model_name} on Groq with top_k = {k}")

    answers = []
    refs = []

    start = time.time()

    for item in test_data:
        question = item["question"]
        expected = item["expected_answer"]
        top_chunks = retrieve_top_k(question, k=k)

        try:
            response = ask_groq_model(model_name, question, top_chunks)
        except Exception as e:
            print(f"❌ Error: {e}")
            response = "ERROR"

        answers.append(response)
        refs.append(expected)

    duration = time.time() - start
    avg_time = round(duration / len(test_data), 3)

    _, _, F1 = score(answers, refs, lang="en", device='cuda', verbose=False)
    avg_f1 = round(torch.mean(F1).item(), 4)

    groq_results.append({
        "model": f"Groq-{model_name}",
        "top_k": k,
        "avg_bert_f1": avg_f1,
        "avg_time_sec": avg_time
    })

# Final DataFrame
df_groq = pd.DataFrame(groq_results)
display(df_groq)
```

```
⚡ Evaluating llama3-70b-8192 on Groq with top_k = 1

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

⚡ Evaluating llama3-70b-8192 on Groq with top_k = 3

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

⚡ Evaluating llama3-70b-8192 on Groq with top_k = 5

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

⚡ Evaluating llama3-70b-8192 on Groq with top_k = 7

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

⚡ Evaluating llama3-70b-8192 on Groq with top_k = 10

Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are
newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.
```

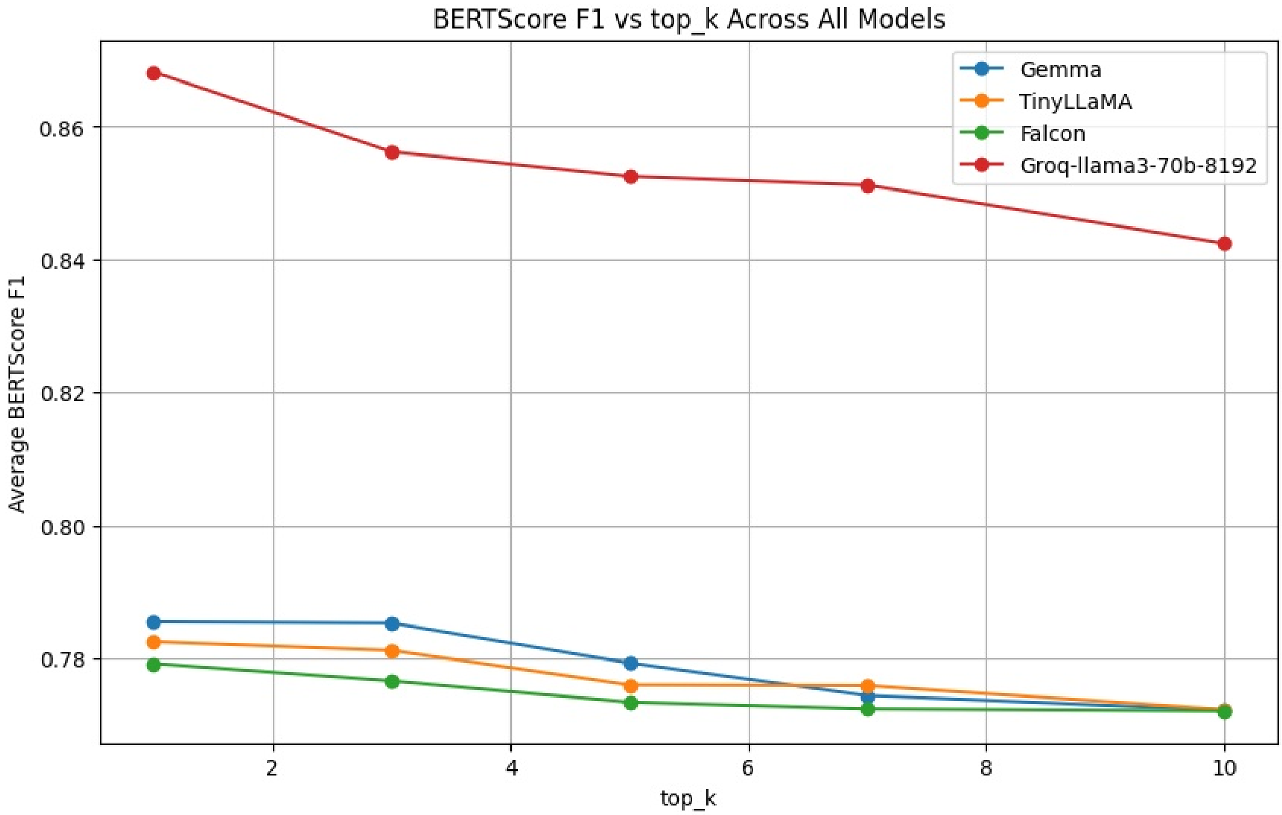
	model	top_k	avg_bert_f1	avg_time_sec
0	Groq-llama3-70b-8192	1	0.8682	0.545
1	Groq-llama3-70b-8192	3	0.8562	0.620
2	Groq-llama3-70b-8192	5	0.8525	0.583
3	Groq-llama3-70b-8192	7	0.8512	0.600
4	Groq-llama3-70b-8192	10	0.8424	0.680

In [ ]:

```
df_all_models = pd.concat([df_all_models, df_groq], ignore_index=True)
```

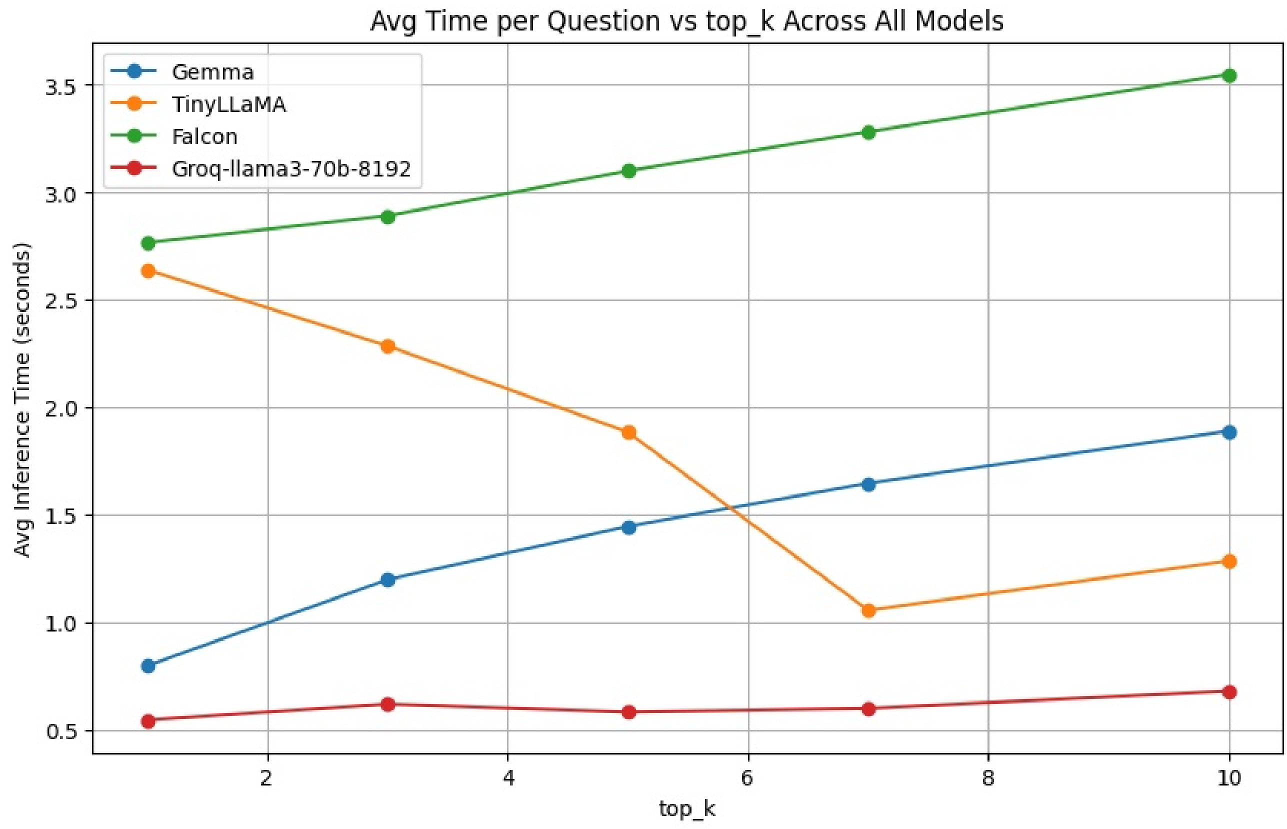
In [ ]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

for model_name in df_all_models["model"].unique():
    subset = df_all_models[df_all_models["model"] == model_name]
    plt.plot(subset["top_k"], subset["avg_bert_f1"], marker='o', label=model_name)

plt.title("BERTScore F1 vs top_k Across All Models")
plt.xlabel("top_k")
plt.ylabel("Average BERTScore F1")
plt.grid(True)
plt.legend()
plt.show()
```



In [ ]:

```
plt.figure(figsize=(10, 6))

for model_name in df_all_models["model"].unique():
    subset = df_all_models[df_all_models["model"] == model_name]
    plt.plot(subset["top_k"], subset["avg_time_sec"], marker='o', label=model_name)

plt.title("Avg Time per Question vs top_k Across All Models")
plt.xlabel("top_k")
plt.ylabel("Avg Inference Time (seconds)")
plt.grid(True)
plt.legend()
plt.show()
```



Conclusion

After testing multiple open-source models like Gemma-2B, TinyLLaMA, and Falcon-1B, one thing became super clear: **Groq's LLaMA3-70B absolutely stands out**.

- **Performance-wise**, it consistently gave the most accurate answers, even when we increased the number of context chunks. Other models started to get confused with too much information, but Groq held up really well.
- **Speed is where Groq shocked us**—Groq's custom hardware (their LPU), the responses were fast. Even faster than smaller models running locally on a GPU.
- **Clean deployment**: No worrying about GPU memory, torch cleanup, or huggingface downloads. It just works through an API, making it perfect for production and scalable setups.

Finally, **Groq** gave us big-brain answers with wait times which is what we need for a realtime campus chatbot like **WPIBot**.