1. Import Library

Memasukkan library yang diperlukan untuk membuat model LLM

```
!pip install transformers datasets peft lion-pytorch accelerate
sentencepiece -q
!pip install huggingface_hub -q

import os
import json
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt

import torch
from torch.nn import functional as F
from torch.utils.data import DataLoader

from transformers import AutoTokenizer, AutoModelForCausalLM,
get_cosine_schedule_with_warmup
from datasets import Dataset
from peft import LoraConfig, get_peft_model
from lion_pytorch import Lion

print("✅ Semua library berhasil diimport!")
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

```
☐ Semua library berhasil diimport!
PyTorch version: 2.9.0+cu126
CUDA available: True
GPU: NVIDIA A100-SXM4-40GB
```

2. Dataset & Data Preparation

Dataset dibuat secara manual yang berisi tentang kuliner khas Yogyakarta, Sejarah Yogyakarta, dan rekomendasi destinasi tempat wisata. Dataset ini telah dipecah menjadi dataset_jogja_train dan dataset_jogja_test untuk keperluan pelatihan dan pengujian

```
from google.colab import files
print("✅ Upload file dataset_jogja_train.json dan
dataset_jogja_test.json")
```

```
☐ Upload file dataset_jogja_train.json dan dataset_jogja_test.json
```

```
print("✅ Memuat dataset...")

with open('dataset_jogja_train.json', 'r', encoding='utf-8') as f:
    train_data = json.load(f)

with open('dataset_jogja_test.json', 'r', encoding='utf-8') as f:
    test_data = json.load(f)

print(f"✅ Dataset berhasil dimuat!")
```

```
print(f"     Train: {len(train_data)} samples")
print(f"     Test: {len(test_data)} samples")
```

```
 Memuat dataset...
 Dataset berhasil dimuat!
    Train: 634 samples
    Test: 112 samples
```

```
print("✓ Contoh data training:")
print("="*60)

for i in range(3):
    print(f"\nSample {i+1}:")
    print(f"  Instruction: {train_data[i]['instruction']}")
    print(f"  Input: {train_data[i].get('input', '-')}")
    print(f"  Output: {train_data[i]['output'][:100]}...")
```

```
 Contoh data training:
============================================================

Sample 1:
  Instruction: Mengapa organisasi otonom seperti Aisyiyah penting?
  Input: -
  Output: Memberikan wadah perempuan untuk terlibat aktif dalam
pendidikan dan dakwah....

Sample 2:
  Instruction: Apa contoh kampung prajurit di luar benteng?
  Input: -
  Output: Wirobrajan, Dhaengan, Patangpuluhan, Jogokaryan,
Prawirotaman, Nyutran, Ketanggungan, Mantrijeron....

Sample 3:
  Instruction: Mengapa istilah sapaan sangat rinci dalam budaya Jawa?
  Input: -
  Output: Untuk menunjukkan hormat dan posisi sosial kerabat....
```

```
train_dataset = Dataset.from_list(train_data)
test_dataset = Dataset.from_list(test_data)

train_loader = DataLoader(
    train_dataset,
    batch_size=2,
    shuffle=True,
    drop_last=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=1,
    shuffle=False
```

```
)

print(f"✅ DataLoader siap!")
print(f"   Train batches: {len(train_loader)}")
print(f"   Test batches: {len(test_loader)}")
```

```
🚀 DataLoader siap!
   Train batches: 317
   Test batches: 112
```

3. Tokenization & Prompt Formatting

Menggunakan prompt yang khusus untuk model Gemma dengan format chat.

```
template_without_answer = "<start_of_turn>user\
n{question}<end_of_turn>\n<start_of_turn>model\n"
template_with_answer = template_without_answer +
"{answer}<end_of_turn>\n"

print("✅ Contoh format prompt:")
print("="*60)
print(template_with_answer.format(
    question="Apa itu gudeg?",
    answer="Gudeg adalah makanan khas Yogyakarta berbahan dasar nangka
muda."
))
```

```
🚀 Contoh format prompt:
============================================================
<start_of_turn>user
Apa itu gudeg?<end_of_turn>
<start_of_turn>model
Gudeg adalah makanan khas Yogyakarta berbahan dasar nangka
muda.<end_of_turn>
```

```
model_id = "unsloth/gemma-2-2b-it"

print("✅ Memuat tokenizer...")
tokenizer = AutoTokenizer.from_pretrained(model_id)
tokenizer.pad_token = tokenizer.eos_token

print(f"✅ Tokenizer siap!")
print(f"   Vocab size: {len(tokenizer.get_vocab())}")
print(f"   Pad token: {tokenizer.pad_token}")
```

```
🚀 Memuat tokenizer...
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94:  UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(

{"model_id":"f4124a98ce404858a56be16d307f144b","version_major":2,"vers
ion_minor":0}

{"model_id":"19ab759ea0104ef6929f11bdeec06b34","version_major":2,"vers
ion_minor":0}

{"model_id":"db3b56f3220548a48144cdfdf1fcf905","version_major":2,"vers
ion_minor":0}

{"model_id":"9c958d3ff1c1418e9aee4d1ac8ffed89","version_major":2,"vers
ion_minor":0}

 Tokenizer siap!
    Vocab size: 256000
    Pad token: <eos>

sample_text = "Apa rekomendasi makanan di Jogja?"
tokens = tokenizer.encode(sample_text, return_tensors="pt")

print(f"Original: {sample_text}")
print(f"Token IDs: {tokens[0].tolist()}")
print(f"Decoded: {tokenizer.decode(tokens[0])}")

Original: Apa rekomendasi makanan di Jogja?
Token IDs: [2, 46719, 189923, 35041, 751, 62064, 1663, 235336]
Decoded: <bos>Apa rekomendasi makanan di Jogja?

def collate_fn(batch):
    """
    Custom collate function untuk memproses batch data
    dengan padding dan attention mask
    """
    texts = []
    for item in batch:
        text = template_with_answer.format(
            question=item['instruction'],
            answer=item['output']
        )
        texts.append(text)
```

```
    encoded = tokenizer(
        texts,
        padding=True,
        truncation=True,
        max_length=512,
        return_tensors="pt"
    )

    return  encoded

print("✅ Collate function siap!")
```

```
✅ Collate function siap!
```

4. Load Pre-trained Model

Untuk proyek ini, kita memutuskan untuk menggunakan model Gemma 2B yang sudah di-pretrain. Alasan memilih model Gemma 2B karena ukuran model yang efisien, performa yang bagus untuk Bahasa Indonesia, dan gratis untuk digunakan.

```
print("✅ Memuat model Gemma 2B...")
print("✅ Ini membutuhkan waktu beberapa menit...")

model = AutoModelForCausalLM.from_pretrained(
    model_id,
    torch_dtype=torch.float16,
    device_map="auto"
)

print(f"✅ Model berhasil dimuat!")
print(f"   Model: {model_id}")
print(f"   Device: {model.device}")
print(f"   Parameters: {sum(p.numel() for p in
model.parameters()):,}")
```

```
✅ Memuat model Gemma 2B...
✅ Ini membutuhkan waktu beberapa menit...
```

{"model_id":"23b4701424ec46a0bc2b3243dd101875","version_major":2,"version_minor":0}

```
`torch_dtype` is deprecated! Use `dtype` instead!
```

{"model_id":"ec9259f9b50d4fe3b7e72bad52ec9b63","version_major":2,"version_minor":0}

{"model_id":"d27d55d3fb084f5b9be8849b22a9c2b9","version_major":2,"version_minor":0}

```
✅ Model berhasil dimuat!
   Model: unsloth/gemma-2-2b-it
```

```
    Device: cuda:0
    Parameters: 2,614,341,888

print("✅ Test model sebelum fine-tuning:")
print("="*60)

test_question = "Apa itu gudeg?"
prompt = template_without_answer.format(question=test_question)
tokens = tokenizer.encode(prompt,
return_tensors="pt").to(model.device)

with torch.no_grad():
    output = model.generate(tokens, max_new_tokens=50)

response = tokenizer.decode(output[0], skip_special_tokens=True)
print(f"Q: {test_question}")
print(f"A: {response.split('model')[-1].strip() if 'model' in response
else response}")

 Test model sebelum fine-tuning:
============================================================
Q: Apa itu gudeg?
A: Gudeg adalah makanan khas Yogyakarta, Indonesia.

**Deskripsi:**

* **Bahan utama:** Gudeg terbuat dari **sayur dan daging yang dimasak
dengan santan dan bumbu khas**.
* **Sayuran:** Biasanya, gu
```

5. LoRA Configuration

LoRA membekukan parameter model asli, menambahkan adapter kecil yang dapat dilatih, dan mengurangi kebutuhan memory dalam proses pelatihan.

```
lora_config = LoraConfig(
    r=32,
    lora_alpha=64,
    lora_dropout=0.05,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, lora_config)

print("✅ LoRA berhasil diterapkan!")
model.print_trainable_parameters()
```

```
 LoRA berhasil diterapkan!
trainable params: 41,533,440 || all params: 2,655,875,328 || trainable
%: 1.5638
```

```
NUM_EPOCHS = 3
ACCUMULATION_STEPS = 8
LEARNING_RATE = 2e-4
WARMUP_RATIO = 0.1

total_steps = (len(train_loader) // ACCUMULATION_STEPS) * NUM_EPOCHS
warmup_steps = int(total_steps * WARMUP_RATIO)

optimizer = Lion(model.parameters(), lr=LEARNING_RATE,
weight_decay=0.01)
scheduler = get_cosine_schedule_with_warmup(
    optimizer,
    num_warmup_steps=warmup_steps,
    num_training_steps=total_steps
)

print("✅ Training Configuration:")
print(f"   Epochs: {NUM_EPOCHS}")
print(f"   Total steps: {total_steps}")
print(f"   Warmup steps: {warmup_steps}")
print(f"   Batch size: 2")
print(f"   Accumulation steps: {ACCUMULATION_STEPS}")
print(f"   Effective batch size: {2 * ACCUMULATION_STEPS}")
print(f"   Learning rate: {LEARNING_RATE}")
```

```
 Training Configuration:
   Epochs: 3
   Total steps: 117
   Warmup steps: 11
   Batch size: 2
   Accumulation steps: 8
   Effective batch size: 16
   Learning rate: 0.0002
```

6. Training Loop

```
def compute_loss(model, tokens, mask, max_length=512):
    """
    Hitung loss hanya untuk bagian JAWABAN (bukan pertanyaan)
    """
    tokens = tokens[:, :max_length]
    mask = mask[:, :max_length]

    x = tokens[:, :-1]
```

```
    y = tokens[:, 1:]
    mask = mask[:, 1:]

    logits = model(x).logits

    loss = F.cross_entropy(
        logits.reshape(-1, logits.size(-1)),
        y.reshape(-1),
        reduction="none"
    )

    masked_loss = loss * mask.reshape(-1).float()
    return masked_loss.sum() / mask.sum().clamp(min=1)

print("✅ Loss function siap!")
```

```
✅ Loss function siap!
```

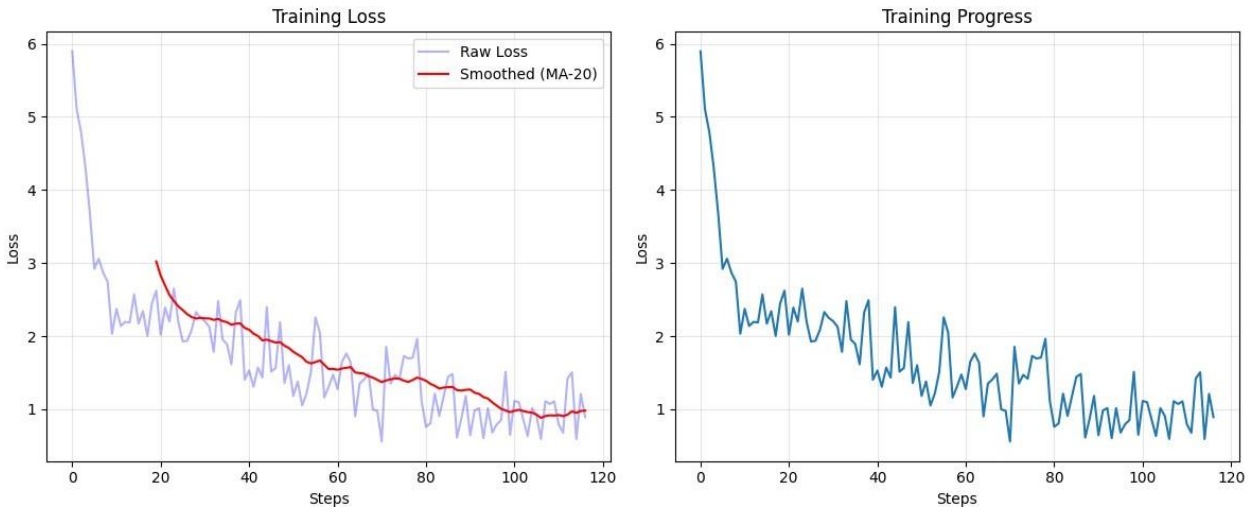
```
print("="*60)
print("✅ MEMULAI TRAINING")
print("="*60)

model.train()
optimizer.zero_grad()

losses = []
global_step = 0
best_loss = float('inf')

for epoch in range(NUM_EPOCHS):
    print(f"\n✅ EPOCH {epoch+1}/{NUM_EPOCHS}")
    print("-"*60)

    epoch_losses = []
    pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}")

    for i, batch in enumerate(pbar):
        q = batch["instruction"][0]
        a = batch["output"][0]
        text = template_with_answer.format(question=q, answer=a)

        encoded = tokenizer(
            text,
            return_tensors="pt",
            truncation=True,
            max_length=512,
            padding=False
        )

        tokens = encoded["input_ids"].to(model.device)
```

```
            answer_start = text.find("<start_of_turn>model\n") +
len("<start_of_turn>model\n")
            answer_tokens_start =
len(tokenizer.encode(text[:answer_start])) - 1
            mask = torch.zeros_like(tokens, dtype=torch.bool)
            mask[0, answer_tokens_start:] = True

            loss = compute_loss(model, tokens, mask)
            loss = loss / ACCUMULATION_STEPS
            loss.backward()

            epoch_losses.append(loss.item() * ACCUMULATION_STEPS)

            if (i + 1) % ACCUMULATION_STEPS == 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
                optimizer.step()
                scheduler.step()
                optimizer.zero_grad()
                global_step += 1

                avg_loss = np.mean(epoch_losses[-ACCUMULATION_STEPS:])
                losses.append(avg_loss)

                if avg_loss < best_loss:
                    best_loss = avg_loss

                pbar.set_postfix({"loss": f"{avg_loss:.4f}", "best":
f"{best_loss:.4f}"})

        epoch_avg_loss = np.mean(epoch_losses)
        print(f"\n✓ Epoch {epoch+1} Summary:")
        print(f"    Average Loss: {epoch_avg_loss:.4f}")
        print(f"    Best Loss: {best_loss:.4f}")

print("\n" + "="*60)
print("✓ TRAINING SELESAI!")
print("="*60)
print(f"Total Steps: {global_step}")
print(f"Final Loss: {losses[-1]:.4f}")
print(f"Best Loss: {best_loss:.4f}")
```

```
============================================================
🚀 MEMULAI TRAINING
============================================================

📊 EPOCH 1/3
------------------------------------------------------------

Epoch 1: 100%|████████████| 317/317 [01:18<00:00,4.02it/s,
loss=2.4914, best=1.6158]
```

```
📊 Epoch 1 Summary:
   Average Loss: 2.5903
   Best Loss: 1.6158

🔄 EPOCH 2/3
------------------------------------------------------------

Epoch 2: 100%|██████████| 317/317 [01:17<00:00,4.09it/s,
loss=1.7067, best=0.5579]


📊 Epoch 2 Summary:
   Average Loss: 1.4821
   Best Loss: 0.5579

🔄 EPOCH 3/3
------------------------------------------------------------

Epoch 3: 100%|██████████| 317/317 [01:17<00:00,4.09it/s,
loss=0.8921, best=0.5579]


📊 Epoch 3 Summary:
   Average Loss: 0.9889
   Best Loss: 0.5579

============================================================
🎉 TRAINING SELESAI!
============================================================
Total Steps: 117
Final Loss: 0.8921
Best Loss: 0.5579



plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(losses, alpha=0.3, color='blue', label='Raw Loss')
if len(losses) > 20:
    smoothed = np.convolve(losses, np.ones(20)/20, mode='valid')
    plt.plot(range(19, len(losses)), smoothed, color='red',
label='Smoothed (MA-20)')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
```

```
plt.plot(losses)
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.title('Training Progress')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('training_curve.png', dpi=150)
plt.show()

print("✅ Training curve tersimpan di training_curve.png")
```




```
✅ Training curve tersimpan di training_curve.png
```

7. Model Evaluation (LLM-based Scoring)

Evaluasi model menggunakan LLM sebagai juri untuk memberikan skor 0-10.

```
def chat(question, max_new_tokens=150):
    """
    Fungsi untuk chat dengan model
    """
    model.eval()
    prompt = template_without_answer.format(question=question)
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            temperature=0.7,
```

```
            do_sample=True,
            top_p=0.9,
            repetition_penalty=1.15,
            pad_token_id=tokenizer.eos_token_id
        )

    result = tokenizer.decode(outputs[0], skip_special_tokens=True)

    if "model\n" in result:
        return result.split("model\n")[-1].strip()
    return result.strip()

print("✅ Fungsi chat() siap!")
```

```
✅ Fungsi chat() siap!
```

```
print("✅ Generating jawaban untuk test set...")

n_samples = 10
generated_samples = []
reference_samples = []

for i, batch in enumerate(tqdm(test_loader, total=n_samples)):
    if i >= n_samples:
        break

    question = batch['instruction'][0]
    reference = batch['output'][0]

    answer = chat(question, max_new_tokens=100)

    generated_samples.append(answer)
    reference_samples.append(reference)

print(f"✅ Generated {len(generated_samples)} jawaban")
```

```
✅ Generating jawaban untuk test set...

100%|██████████| 10/10 [00:13<00:00, 1.32s/it]

✅ Generated 10 jawaban
```

```
import re

class LLMJudge:
    def __init__(self, model, tokenizer):
        self.model = model
        self.tokenizer = tokenizer

    def score(self, question, reference, answer):
```

```
        """
        Beri skor 0-10 untuk jawaban model
        """
        prompt = f"""<start_of_turn>user
Rate this answer from 0-10 based on accuracy and relevance.

Question: {question}
Reference Answer: {reference}
Model Answer: {answer}

Reply with only: SCORE: X (where X is 0-10)<end_of_turn>
<start_of_turn>model
"""
        inputs = self.tokenizer(prompt,
return_tensors="pt").to(self.model.device)

        with torch.no_grad():
            outputs = self.model.generate(
                **inputs,
                max_new_tokens=20,
                temperature=0.1,
                do_sample=False
            )

        response = self.tokenizer.decode(outputs[0],
skip_special_tokens=True)
        match = re.search(r'SCORE:\s*(\d+)', response, re.IGNORECASE)
        if match:
            score = int(match.group(1))
            return min(max(score, 0), 10)

        nums = re.findall(r'\b(10|[0-9])\b', response)
        if nums:
            return int(nums[-1])

        return 5

judge = LLMJudge(model, tokenizer)
print("✅ LLM Judge siap!")
```

```
 LLM Judge siap!
```

```
print("🔍 Mengevaluasi jawaban model...")

scores = []
for i in tqdm(range(len(generated_samples))):
    question = test_dataset[i]['instruction']
    reference = reference_samples[i]
    answer = generated_samples[i]

    score = judge.score(question, reference, answer)
```

```
        scores.append(score)

# Statistik evaluasi
avg_score = np.mean(scores)
min_score = np.min(scores)
max_score = np.max(scores)

print("\n" + "="*60)
print("✅ HASIL EVALUASI")
print("="*60)
print(f"Rata-rata Skor: {avg_score:.2f} / 10")
print(f"Skor Minimum: {min_score}")
print(f"Skor Maximum: {max_score}")
print(f"Total Samples: {len(scores)}")
```

```
🔬 Mengevaluasi jawaban model...

  0%|          | 0/10 [00:00<?, ?it/s]The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VERBOSITY=info` for more details.
100%|██████████| 10/10 [00:09<00:00,  1.01it/s]


============================================================
📊 HASIL EVALUASI
============================================================
Rata-rata Skor: 9.00 / 10
Skor Minimum: 7
Skor Maximum: 10
Total Samples: 10
```

8. Model Prediction & Save Results

```
print("="*70)
print("✅ PENGUJIAN 5 SKENARIO CHATBOT")
print("="*70)

test_scenarios = [
    "Apa itu gudeg kering?",
    "Dimana lokasi Sate Klathak Pak Bari?",
    "Siapa pendiri Muhammadiyah?",
    "Kapan Perang Diponegoro terjadi?",
    "Rekomendasi kuliner malam di Jogja?"
]

scenario_results = []

for i, question in enumerate(test_scenarios, 1):
    answer = chat(question, max_new_tokens=150)
```

```
        print(f"\n{'='*70}")
        print(f"✅ SKENARIO {i}")
        print(f"{'='*70}")
        print(f"✅ Pertanyaan: {question}")
        print(f"🤖 Jawaban: {answer}")

        scenario_results.append({
            "skenario": i,
            "pertanyaan": question,
            "jawaban": answer
        })

print("\n" + "="*70)
print("✅ PENGUJIAN 5 SKENARIO SELESAI")
print("="*70)
```

```
======================================================================
📊 PENGUJIAN 5 SKENARIO CHATBOT
======================================================================

======================================================================
📌 SKENARIO 1
======================================================================
✅ Pertanyaan: Apa itu gudeg kering?
🤖 Jawaban: Gudeg manis karena pengaruh budaya Mataram yang kaya gula. Manis melambangkan kemakmuran.

======================================================================
📌 SKENARIO 2
======================================================================
✅ Pertanyaan: Dimana lokasi Sate Klathak Pak Bari?
🤖 Jawaban: Di sebelah timur barat alun-alun selatan Tugu.

======================================================================
📌 SKENARIO 3
======================================================================
✅ Pertanyaan: Siapa pendiri Muhammadiyah?
🤖 Jawaban: Para pelajar STOVIA.

======================================================================
📌 SKENARIO 4
======================================================================
✅ Pertanyaan: Kapan Perang Diponegoro terjadi?
🤖 Jawaban: 1825–1830.

======================================================================
📌 SKENARIO 5
======================================================================
✅ Pertanyaan: Rekomendasi kuliner malam di Jogja?
```

```
🤖 Jawaban: Dekat area tersebut banyak warung lokal. Pakai Google Maps cari 'kuliner dekat saya' atau tanya warga.

================================================================================
🎉 PENGUJIAN 5 SKENARIO SELESAI
================================================================================

print("✅ Menyimpan hasil prediksi...")

predictions = []
for i in range(len(generated_samples)):
    predictions.append({
        "instruction": test_dataset[i]['instruction'],
        "input": test_dataset[i].get('input', ''),
        "expected_output": reference_samples[i],
        "model_response": generated_samples[i],
        "score": scores[i] if i < len(scores) else None
    })

with open("predictions.json", "w", encoding="utf-8") as f:
    json.dump(predictions, f, ensure_ascii=False, indent=2)

print(f"✅ {len(predictions)} prediksi tersimpan di predictions.json")

with open("test_scenarios.json", "w", encoding="utf-8") as f:
    json.dump(scenario_results, f, ensure_ascii=False, indent=2)

print("✅ Hasil skenario tersimpan di test_scenarios.json")

💾 Menyimpan hasil prediksi...
✅ 10 prediksi tersimpan di predictions.json
✅ Hasil skenario tersimpan di test_scenarios.json
```

9. Save & Upload Model

Untuk model, kita simpan di hugging face karena limit filenya yang tidak terhingga mengingat model LLM ini memiliki ukuran lebih dari 1 GB. Alasan kita menggunakan gradio bukan streamlit karena streamlit cloud tidak dapat menompang model yang berukuran besar sedangkan gradio mendukung model yang berukuran besar. Selain itu, gradio berintegrasi dengan hugging face dan mempunyai fitur *ChatInterface* untuk aplikasi chatbot

```
output_dir = "model_jogja_kuliner_sejarah"

print(f"✅ Menyimpan model ke '{output_dir}'...")
model.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

print("✅ Model berhasil disimpan!")

💾 Menyimpan model ke 'model_jogja_kuliner_sejarah'...
✅ Model berhasil disimpan!

from huggingface_hub import login

print("✅ Login ke Hugging Face...")
```

```
print("Dapatkan token di: https://huggingface.co/settings/tokens")
login()
```

```
 Login ke Hugging Face...
Dapatkan token di: https://huggingface.co/settings/tokens
```

{"model_id":"0c0832af6aa24f09a1d3389fde556ef8","version_major":2,"version_minor":0}

```
HF_USERNAME = "JSpiritGG"
REPO_NAME = "jogja-kuliner-chatbot"

print("✅ Uploading model ke Hugging Face Hub...")
print("✅ Ini membutuhkan waktu beberapa menit...")

model.push_to_hub(f"{HF_USERNAME}/{REPO_NAME}")
print("✅ Model uploaded!")

tokenizer.push_to_hub(f"{HF_USERNAME}/{REPO_NAME}")
print("✅ Tokenizer uploaded!")

print(f"\n✅ Model tersimpan di:
https://huggingface.co/{HF_USERNAME}/{REPO_NAME}")
```

```
 Uploading model ke Hugging Face Hub...
 Ini membutuhkan waktu beberapa menit...
```

{"model_id":"14d819029e074ba09a3ba06cd18d8527","version_major":2,"version_minor":0}

{"model_id":"fa099e348a3e46609caa71beea3b1e50","version_major":2,"version_minor":0}

{"model_id":"afcdf08982f9497e85621edfa7f6f42f","version_major":2,"version_minor":0}

{"model_id":"84e97841ab77439eb3a9d645b8a62361","version_major":2,"version_minor":0}

```
 Model uploaded!
```

{"model_id":"718fcc6b855845399b546018a6a9b23f","version_major":2,"version_minor":0}

{"model_id":"8d4b6427af4748d8bbb917826b0e9487","version_major":2,"version_minor":0}

{"model_id":"6e8e9a3eddd04117a519cb57b3616ff6","version_major":2,"version_minor":0}

{"model_id":"ea911b6f3b3a406786f18ca71a40119c","version_major":2,"version_minor":0}

```
No files have been modified since last commit. Skipping to prevent empty commit.
WARNING:huggingface_hub.hf_api:No files have been modified since last commit. Skipping to prevent empty commit.

 Tokenizer uploaded!

 Model tersimpan di: https://huggingface.co/JSpiritGG/jogja-kuliner-chatbot
```

10. Deployment ke Hugging Face Spaces

```
api = HfApi()

print("✅ Uploading ke Hugging Face Space...")

for filename in ["README.md", "app.py", "requirements.txt"]:
    api.upload_file(
        path_or_fileobj=filename,
        path_in_repo=filename,
        repo_id=f"{HF_USERNAME}/{SPACE_NAME}",
        repo_type="space"
    )
    print(f"✅ {filename} uploaded")

print(f"""
{'='*60}
 DEPLOYMENT SELESAI!
{'='*60}

 Link Aplikasi (PERMANEN):
   https://huggingface.co/spaces/{HF_USERNAME}/{SPACE_NAME}

 Tunggu 5-10 menit untuk build selesai.
{'='*60}
""")

 Uploading ke Hugging Face Space...
 README.md uploaded
 app.py uploaded
 requirements.txt uploaded

============================================================
 DEPLOYMENT SELESAI!
============================================================

 Link Aplikasi (PERMANEN):
   https://huggingface.co/spaces/JSpiritGG/jogja-kuliner-chatbot
```

```
 Tunggu 5-10 menit untuk build selesai.
============================================================
```