Google Colabでの実行ガイド

実行前の重要な準備

1. Colab設定の確認

```python
# 最初に実行するセル - 環境確認
import torch
print(f"PyTorch version: {torch.__version__}")
print(f"CUDA available: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"CUDA device: {torch.cuda.get_device_name()}")
    print(f"GPU memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.1f} GB")
```

2. ランタイムタイプの設定

- **重要**: 必ずGPUランタイムを選択
- メニュー → ランタイム → ランタイムのタイプを変更 → GPU(T4)を選択
- CPU環境では動作が非常に遅くなります

段階的実行プラン（推奨順序）

Phase 1: 基本環境構築（5-10分）

```python
# セル1: パッケージインストール
!pip install transformers>=4.21.0 torch>=1.12.0 tokenizers>=0.12.0 gradio

# セル2: インポートテスト
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import torch
import time
import re
print("✅ All imports successful")
```

Phase 2: 最小実装テスト（10-15分）

```python
# セル3: 最小チャットボットテスト
class MinimalChatbot:
    def __init__(self, model_name='gpt2'):  # 最初は小さいモデルでテスト
        self.tokenizer = GPT2Tokenizer.from_pretrained(model_name)
        self.model = GPT2LMHeadModel.from_pretrained(model_name)
        self.tokenizer.pad_token = self.tokenizer.eos_token

    def generate(self, text, max_length=50):
        inputs = self.tokenizer.encode(text, return_tensors='pt')
        with torch.no_grad():
            outputs = self.model.generate(
                inputs,
                max_length=inputs.shape[1] + max_length,
                temperature=0.7,
                do_sample=True,
                pad_token_id=self.tokenizer.eos_token_id
            )
        return self.tokenizer.decode(outputs[0][inputs.shape[1]:], skip_special_tokens=True)

# テスト実行
bot = MinimalChatbot()
response = bot.generate("Hello, how are you?")
print(f"Response: {response}")
```

Phase 3: 会話管理機能追加（15-20分）

```python
# セル4: ConversationManagerを追加
class ConversationManager:
    def __init__(self, max_history=5):  # 最初は少なめに設定
        self.conversation_history = []
        self.max_history = max_history

    def add_message(self, role, message):
        self.conversation_history.append({
            'role': role,
            'message': message,
            'timestamp': time.time()
        })
        if len(self.conversation_history) > self.max_history:
            self.conversation_history.pop(0)

    def get_context(self):
        context = ""
        for msg in self.conversation_history[-3:]:  # 直近3回
            role_name = "User" if msg['role'] == 'user' else "Bot"
            context += f"{role_name}: {msg['message']}\n"
        return context

# テスト
conv_manager = ConversationManager()
conv_manager.add_message('user', 'Hello')
conv_manager.add_message('bot', 'Hi there!')
print(conv_manager.get_context())
```

Phase 4: 統合チャットボット（20-30分）

```python
```

```python
# セル5: 統合版チャットボット
class SimpleChatbot:
    def __init__(self, model_name='gpt2-medium'):  # ここでmediumに変更
        print("Loading model... This may take 2-3 minutes.")
        self.tokenizer = GPT2Tokenizer.from_pretrained(model_name)
        self.model = GPT2LMHeadModel.from_pretrained(model_name)
        self.tokenizer.pad_token = self.tokenizer.eos_token
        self.conv_manager = ConversationManager()
        print("✅ Model loaded successfully!")

    def generate_response(self, user_input, max_length=100):
        # コンテキスト構築
        context = self.conv_manager.get_context()
        prompt = f"{context}User: {user_input}\nBot:"

        # 応答生成
        inputs = self.tokenizer.encode(prompt, return_tensors='pt')

        with torch.no_grad():
            outputs = self.model.generate(
                inputs,
                max_length=inputs.shape[1] + max_length,
                temperature=0.7,
                top_p=0.9,
                do_sample=True,
                pad_token_id=self.tokenizer.eos_token_id,
                repetition_penalty=1.1
            )

        response = self.tokenizer.decode(
            outputs[0][inputs.shape[1]:],
            skip_special_tokens=True
        ).strip()

        # 履歴更新
        self.conv_manager.add_message('user', user_input)
        self.conv_manager.add_message('bot', response)

        return response

# 初期化（時間がかかります）
chatbot = SimpleChatbot()
```

Phase 5: Gradio UI作成（5分）

```python
```

```python
# セル6: 簡単なUI作成
import gradio as gr

def chat_interface(message, history):
    if not message.strip():
        return history, ""

    response = chatbot.generate_response(message)
    history.append([message, response])
    return history, ""

# インターフェース起動
with gr.Blocks(title="My GPT-2 Chatbot") as demo:
    gr.Markdown("# 🤖 My First GPT-2 Chatbot")

    chatbot_ui = gr.Chatbot(label="Chat History", height=400)
    msg = gr.Textbox(label="Your Message", placeholder="Type here...")

    msg.submit(chat_interface, [msg, chatbot_ui], [chatbot_ui, msg])

demo.launch(share=True)  # share=Trueで外部からアクセス可能なリンク生成
```

重要な注意点とトラブルシューティング

メモリ関連の注意

```python
# メモリ使用量監視用コード（適宜実行）
import psutil
import os

def check_memory():
    process = psutil.Process(os.getpid())
    memory_mb = process.memory_info().rss / 1024 / 1024
    print(f"Current memory usage: {memory_mb:.1f} MB")

    # GPU メモリチェック
    if torch.cuda.is_available():
        gpu_memory = torch.cuda.memory_allocated() / 1024 / 1024
        gpu_cached = torch.cuda.memory_reserved() / 1024 / 1024
        print(f"GPU memory allocated: {gpu_memory:.1f} MB")
        print(f"GPU memory cached: {gpu_cached:.1f} MB")

# 使用例
check_memory()
```

よくある問題と解決策

1. CUDA Out of Memory エラー

```python
# エラー発生時の対処法
def clear_memory():
    import gc
    gc.collect()
    if torch.cuda.is_available():
        torch.cuda.empty_cache()
    print("Memory cleared")

# エラー時に実行
clear_memory()

# より小さいモデルに変更
# chatbot = SimpleChatbot('gpt2')  # mediumの代わりにsmallを使用
```

2. セッションタイムアウト対策

```python
# セッション維持用コード（30分間隔で実行）
import time
from datetime import datetime

def keep_session_alive():
    print(f"Session alive check: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    return "OK"

# バックグラウンドで定期実行（オプション）
keep_session_alive()
```

3. モデル保存・読み込み

```python
# Google Driveにモデルを保存
from google.colab import drive
drive.mount('/content/drive')

# 保存
def save_model_to_drive(model, tokenizer, name):
    save_path = f'/content/drive/MyDrive/chatbot_models/{name}'
    model.save_pretrained(save_path)
    tokenizer.save_pretrained(save_path)
    print(f"Model saved to: {save_path}")

# 使用例
# save_model_to_drive(chatbot.model, chatbot.tokenizer, 'my_chatbot_v1')
```

段階的デバッグ戦略

デバッグ用ミニ関数

```python
# 各段階でのテスト関数
def test_tokenizer(text="Hello world"):
    tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
    tokens = tokenizer.encode(text)
    decoded = tokenizer.decode(tokens)
    print(f"Original: {text}")
    print(f"Tokens: {tokens}")
    print(f"Decoded: {decoded}")

def test_generation(prompt="The weather is", max_length=20):
    model = GPT2LMHeadModel.from_pretrained('gpt2')
    tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

    inputs = tokenizer.encode(prompt, return_tensors='pt')
    outputs = model.generate(inputs, max_length=max_length, do_sample=True)
    result = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print(f"Generated: {result}")

# テスト実行
test_tokenizer()
test_generation()
```

パフォーマンス最適化のコツ

1. 段階的モデルサイズアップ

```python
```

```
# 最初は small → medium → large の順で試す
MODEL_PROGRESSION = ['gpt2', 'gpt2-medium', 'gpt2-large']

for model_name in MODEL_PROGRESSION:
    try:
        print(f"Trying {model_name}...")
        chatbot = SimpleChatbot(model_name)
        print(f"✅ {model_name} loaded successfully")
        break
    except Exception as e:
        print(f"❌ {model_name} failed: {e}")
        continue
```

2. 応答品質チェック

```
python

def test_conversation_quality():
    test_inputs = [
        "Hello, how are you?",
        "What's the weather like?",
        "Tell me a joke",
        "What can you help me with?"
    ]

    for test_input in test_inputs:
        response = chatbot.generate_response(test_input)
        print(f"Input: {test_input}")
        print(f"Response: {response}")
        print("-" * 50)

# 品質テスト実行
test_conversation_quality()
```

実行タイムライン（目安）

初回実行（約30-45分）

- 00:00-05:00 環境構築・パッケージインストール
- 05:00-15:00 最小実装テスト
- 15:00-30:00 フル機能チャットボット構築
- 30:00-45:00 UI作成・テスト

2回目以降（約10-15分）

- 00:00-02:00 環境確認
- 02:00-10:00 モデル読み込み（キャッシュ有効）
- 10:00-15:00 機能テスト・UI起動

トラブル別対処法クイックリファレンス

エラー	原因	解決策
CUDA out of memory	GPU メモリ不足	より小さいモデル使用、clear_memory()実行
Module not found	パッケージ未インストール	!pip install [パッケージ名] 再実行
Slow response	CPU モード	GPU ランタイムに変更
Session disconnected	長時間非アクティブ	keep_session_alive() 定期実行
Model download failed	ネットワーク問題	再実行、または別モデル試行

成功のポイント

1. **段階的に構築**: 一度に全てを実装せず、少しずつ機能追加
2. **小さく始める**: gpt2-smallから開始してメモリ使用量を確認

3. **こまめな保存**: 重要な進捗はGoogle Driveに保存
4. **エラーハンドリング**: try-except文でエラー対処を組み込む
5. **リソース監視**: メモリ使用量を定期チェック

実行時は必ずこの順序を守って、各段階での動作を確認してから次に進むことを強く推奨します。