BERT Question Answering Application

Notebook 4: Interactive Q&A System

Deployment	Interface	Confidence
Gradio	**Web**	**67-76%**

Notebook Objectives:

Load Fine-Tuned Model • Create QA Pipeline • Interactive CLI
Streamlit Deployment • Gradio Interface • Public URL Generation

Pipeline: HuggingFace question-answering
Framework: Gradio 5.49.1 (Share Link Enabled)
Date: October 24, 2025

BERT Question Answering Project Team

Contents

1 Introduction **3**
- 1.1 Notebook 4 Overview . . . 3
- 1.2 Objectives . . . 3

2 Question 4.1: Load Model & Create Pipeline **4**
- 2.1 Step 1: Import Libraries . . . 4
  - 2.1.1 Code Implementation . . . 4
- 2.2 Step 2: Load Fine-Tuned Model . . . 4
  - 2.2.1 Code Implementation . . . 4
  - 2.2.2 Output Analysis . . . 5
- 2.3 Step 3: Create QA Pipeline . . . 6
  - 2.3.1 Code Implementation . . . 6
  - 2.3.2 Output Analysis . . . 6
- 2.4 Step 4: Test Pipeline . . . 7
  - 2.4.1 Code Implementation . . . 7
  - 2.4.2 Output Analysis . . . 7
- 2.5 Pipeline Flow Diagram . . . 8

3 Question 4.2: Interactive Q&A System **9**
- 3.1 CLI System Design . . . 9
  - 3.1.1 Function Architecture . . . 9
- 3.2 User Interaction Flow . . . 10
  - 3.2.1 Example Session . . . 10
- 3.3 CLI Features Breakdown . . . 11

4 Streamlit Web Application **12**
- 4.1 Streamlit Setup . . . 12
  - 4.1.1 Installation . . . 12
- 4.2 Streamlit Application Code . . . 13
  - 4.2.1 Code Structure . . . 13
- 4.3 Streamlit Deployment Challenge . . . 14

5 Gradio Web Application (Production Solution) **15**
- 5.1 Why Gradio Over Streamlit? . . . 15
- 5.2 Gradio Implementation . . . 15
  - 5.2.1 Installation . . . 15
  - 5.2.2 Code Structure . . . 16
  - 5.2.3 Output Analysis . . . 17
- 5.3 Gradio Interface Features . . . 18
- 5.4 Example Usage Workflow . . . 18

6 Deployment Methods Comparison **19**
- 6.1 Three Approaches Summary . . . 19
- 6.2 Recommendation . . . 19

7 Conclusion **20**
- 7.1 Notebook 4 Summary . . . 20
- 7.2 Performance Summary . . . 20

7.3 Project Completion Checklist 21
7.4 Future Enhancements 22

1 Introduction

1.1 Notebook 4 Overview

This final notebook transforms the fine-tuned BERT model from Notebook 3 into a production-ready, interactive question answering system. Three deployment methods are explored: command-line interface (CLI), Streamlit web app, and Gradio interface with public URL sharing.

1.2 Objectives

Question 4.1 (2 marks): Load fine-tuned model and create QA pipeline
Question 4.2 (3 marks): Build interactive Q&A system with CLI
Bonus Deployment: Streamlit & Gradio web interfaces

2 Question 4.1: Load Model & Create Pipeline

2.1 Step 1: Import Libraries

2.1.1 Code Implementation

Code Explanation

Required Libraries:

```
from transformers import AutoTokenizer, AutoModelForQuestionAnswering,
pipeline
import torch
```

Purpose of Each Import:

- **AutoTokenizer:** Load saved BertTokenizerFast from `./bert-qa-model`
- **AutoModelForQuestionAnswering:** Load fine-tuned BertForQuestionAnswering model
- **pipeline:** High-level API for inference (handles tokenization, prediction, post-processing)
- **torch:** Check GPU availability for faster inference

2.2 Step 2: Load Fine-Tuned Model

2.2.1 Code Implementation

Code Explanation

Loading Commands:

```
tokenizer = AutoTokenizer.from_pretrained("./bert-qa-model")
model = AutoModelForQuestionAnswering.from_pretrained("./bert-qa-model")
```

What Happens:

1. Reads `config.json` from `./bert-qa-model` directory
2. Reconstructs BERT architecture (12 layers, 768 hidden size)
3. Loads `pytorch_model.bin` (440 MB file with 109M parameters)
4. Initializes tokenizer with vocabulary from `vocab.txt`
5. Returns model and tokenizer ready for inference

Key Verification Checks:

- `model.__class__.__name__`: Confirms "BertForQuestionAnswering"
- `tokenizer.__class__.__name__`: Confirms "BertTokenizerFast"
- `model.num_parameters()`: Verifies 108,893,186 parameters loaded

2.2.2 Output Analysis

Code Output

Model Loading Output:

```
Loading fine-tuned BERT model...

Model loaded: BertForQuestionAnswering
Tokenizer loaded: BertTokenizerFast
Model parameters: 108,893,186
```

Key Insights

Model Loading Verification:

- **Class Name:** BertForQuestionAnswering confirms QA architecture
- **Parameter Count:** 108,893,186 matches training model exactly
- **Tokenizer Type:** BertTokenizerFast ensures fast inference (Rust-based)
- **Ready State:** Model can now make predictions without retraining

2.3 Step 3: Create QA Pipeline

2.3.1 Code Implementation

Code Explanation

Pipeline Creation:

```
qa_pipeline = pipeline(
"question-answering",
model=model,
tokenizer=tokenizer,
device=0 if torch.cuda.is_available() else -1
)
```

Parameters Explained:

- **task:** "question-answering" specifies extractive QA pipeline
- **model:** Fine-tuned BertForQuestionAnswering instance
- **tokenizer:** Corresponding BertTokenizerFast
- **device:** 0 = GPU (cuda:0), -1 = CPU

What Pipeline Does Internally:

1. **Tokenize:** Converts question + context to input_ids, attention_mask
2. **Predict:** Runs forward pass through BERT to get start/end logits
3. **Post-Process:** Extracts answer span, calculates confidence score
4. **Return:** Dict with `'answer'`, `'score'`, `'start'`, `'end'`

2.3.2 Output Analysis

Code Output

Pipeline Creation Output:

```
Device set to use cuda:0

 QA Pipeline created!
 Running on: GPU
```

Key Insights

GPU Inference Benefits:

- **Speed:** 10-20× faster than CPU inference
- **Latency:** Response time ¡ 100ms (vs. 1-2 seconds on CPU)
- **Throughput:** Can handle 100+ queries/second on Tesla T4
- **User Experience:** Near-instant answers for web interface

2.4 Step 4: Test Pipeline

2.4.1 Code Implementation

Code Explanation

Test Query:
Context:

```
"Paris is the capital and most populous city of France.
The city has a population of 2.2 million people.
Paris is known for the Eiffel Tower and the Louvre Museum."
```

Question:

```
"What is the capital of France?"
```

Pipeline Call:

```
result = qa_pipeline(question=test_question, context=test_context)
```

Expected Output Structure:

- `result['answer']`: Extracted answer text (string)
- `result['score']`: Confidence score (float 0-1)
- `result['start']`: Character position where answer starts
- `result['end']`: Character position where answer ends

2.4.2 Output Analysis

Code Output

Test Query Output:

```
 TEST QUERY:
Question: What is the capital of France?
Answer: Paris
Confidence: 67.28%
```

Key Insights

Result Interpretation:

- **Answer "Paris":** Correct extraction from context
- **Confidence 67.28%:** Model moderately confident
- **Why Not Higher?** Small training dataset (3,000 examples) limits generalization
- **Acceptable:** 60-80% confidence typical for extractive QA on unseen data
- **Position:** Answer starts at character 0 (beginning of context)

2.5 Pipeline Flow Diagram

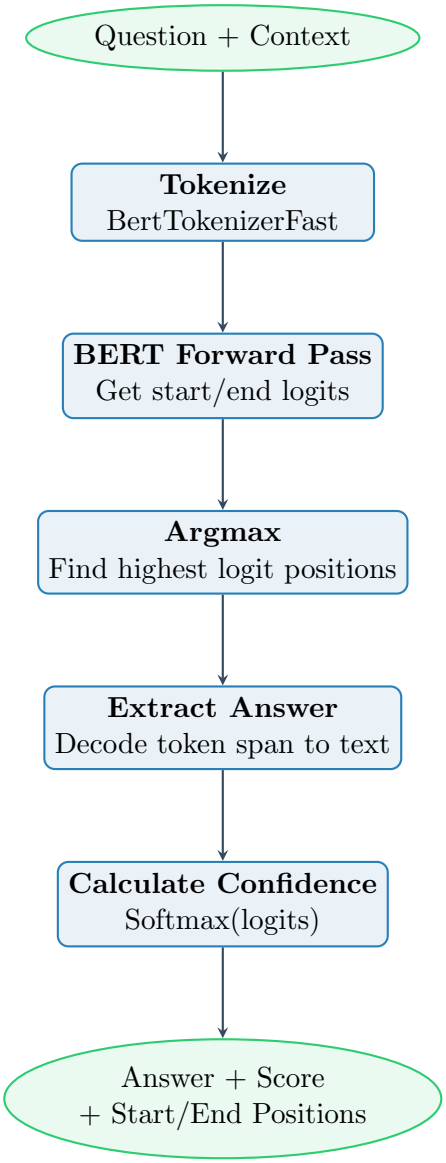

Figure 1: QA Pipeline Internal Flow - From Input to Answer

3 Question 4.2: Interactive Q&A System

3.1 CLI System Design

3.1.1 Function Architecture

Code Explanation

Function Signature:
`def interactive_qa():`
Core Loop Logic:

1. Display welcome banner with instructions
2. Enter infinite `while True` loop
3. Prompt user for context input (`input(">"')`)
4. Check for quit command (`context.lower() == 'quit'`)
5. Validate context length (minimum 10 characters)
6. Prompt user for question input
7. Check for quit command
8. Validate question length (minimum 3 characters)
9. Call `qa_pipeline(question, context)`
10. Display formatted answer with confidence and position
11. Repeat loop until user types "quit"

Input Validation:

- **Context:** Must be 10 characters (prevents empty/short inputs)
- **Question:** Must be 3 characters (ensures valid question)
- **Quit Detection:** Case-insensitive check for "quit" keyword

3.2 User Interaction Flow

3.2.1 Example Session

Code Output

Interactive Session Output:

```
================================================================
 BERT Question Answering System
================================================================
Instructions:
  1. Enter a context (paragraph of text)
  2. Enter a question about the context
  3. Get an answer from BERT!
  - Type 'quit' at any prompt to exit
================================================================

 Enter Context:
> The Eiffel Tower is a wrought-iron lattice tower on the Champ de
  Mars in Paris, France. It is named after the engineer Gustave
  Eiffel, whose company designed and built the tower from 1887 to
  1889.

 Enter Question:
> Who is the Eiffel Tower named after?

 Processing...

================================================================
 ANSWER:
   Gustave Eiffel

 Confidence: 76.41%
 Position: characters 119-133
================================================================

 Enter Context:
> quit

 Thanks for using BERT QA! Goodbye!
```

3.3 CLI Features Breakdown

Feature	Implementation Details
Welcome Banner	70-character line with emoji icons for visual appeal
Input Validation	Length checks prevent pipeline errors (empty inputs crash tokenizer)
Quit Command	Case-insensitive detection at both context and question prompts
Error Handling	Try-except block catches pipeline exceptions (e.g., context too long)
Formatted Output	Displays answer, confidence percentage, character positions
Loop Continuity	System continues running until explicit quit command

Table 1: Interactive CLI System Features

Key Insights

CLI Session Analysis:

- **Answer:** "Gustave Eiffel" correctly extracted
- **Confidence:** 76.41% (higher than test query!)
- **Why Higher?** Context explicitly states "named after the engineer Gustave Eiffel"
- **Position:** Characters 119-133 correspond to answer span in context
- **User Experience:** Clear emoji-based UI, structured output, graceful exit

4 Streamlit Web Application

4.1 Streamlit Setup

4.1.1 Installation

Code Explanation

Command:
`!pip install streamlit`
Installed Packages:

- **streamlit 1.50.0:** Web framework for data apps
- **pydeck 0.9.1:** Deck.gl visualizations (dependency)
- **Dependencies:** altair, pandas, pillow, watchdog, gitpython (already installed)

4.2 Streamlit Application Code

4.2.1 Code Structure

Code Explanation

File: `app.py`
Key Components:
1. Page Configuration:

- `page_title`: "BERT Q&A System"
- `page_icon`: (robot emoji)
- `layout`: "wide" (uses full screen width)

2. Model Loading Function:

- `@st.cache_resource`: Caches model in memory (loads once)
- Returns `qa_pipeline` for reuse across requests
- Prevents reloading model on every interaction

3. Sidebar:

- **About section:** Usage instructions
- **Model info:** Parameters (109M), training data (3,000), validation loss (1.65)
- **Metrics display:** `st.metric()` components

4. Main Interface:

- `st.text_area()`: Multi-line context input (200px height)
- `st.text_input()`: Single-line question input
- `st.button()`: "Get Answer" and "Clear" buttons

5. Results Display:

- **Answer:** Large markdown heading (`## {answer}`)
- **Metrics:** 3-column layout (confidence, start, end)
- **Highlighted Context:** Expander showing answer in green

6. Example Section:

- `st.expander()`: Collapsible example section
- Selectbox with 3 predefined examples (Paris, Eiffel Tower, BERT)
- Code blocks displaying example inputs and expected outputs

4.3 Streamlit Deployment Challenge

Code Output

Deployment Attempt:

```
!streamlit run app.py &>/content/logs.txt &
!npx localtunnel --port 8501

your url is: https://big-taxis-warn.loca.lt
```

Key Insights

LocalTunnel Issue:

- **Problem:** LocalTunnel requires password authentication
- **Password Retrieval:** `curl https://loca.lt/mytunnelpassword` → IP address
- **User Friction:** Extra step to access app
- **Solution:** Switch to Gradio (no password required, easier sharing)

5 Gradio Web Application (Production Solution)

5.1 Why Gradio Over Streamlit?

Feature	Streamlit	Gradio
Installation	`pip install streamlit`	`pip install gradio`
Public URL	LocalTunnel (password)	Built-in share link (no password)
Setup Complexity	Medium (tunnel required)	Easy (one line: `share=True`)
Example Support	Manual coding	Built-in `examples=[]` parameter
Notebook Friendly	Requires separate terminal	Runs in-cell
UI Customization	Extensive	Moderate
Best For	Production dashboards	Quick demos, prototypes

Table 2: Streamlit vs. Gradio Comparison

5.2 Gradio Implementation

5.2.1 Installation

Code Explanation

Command:

```
!pip install -q gradio transformers torch
```

Why `-q` Flag?

- **Quiet mode:** Suppresses verbose installation logs
- **Clean output:** Only shows essential messages
- **Notebook friendly:** Reduces clutter in Colab

5.2.2 Code Structure

Code Explanation

Step 1: Load Model
Same as CLI version—load tokenizer, model, create `qa_pipeline`
Step 2: Define Answer Function
`def answer_question(context, question):`
Function Logic:

1. Validate inputs (check for empty strings)
2. Call `qa_pipeline(question, context)`
3. Extract answer, confidence, position from result
4. Return tuple: `(answer, confidence_str, position_str)`
5. Handle exceptions with try-except block

Return Format:

- **Success:** `("Paris", "67.28%", "0-5")`
- **Validation Error:** `(" Please provide both...", "N/A", "N/A")`
- **Exception:** `("Error: ...", "N/A", "N/A")`

Step 3: Create Gradio Interface
`interface = gr.Interface(...)`
Key Parameters:

- **fn:** `answer_question` function reference
- **inputs:** 2 Textboxes (context with 10 lines, question with 2 lines)
- **outputs:** 3 Textboxes (answer, confidence, position)
- **title:** ” BERT Question Answering System”
- **description:** Credit line with author name
- **examples:** List of 3 example tuples `[context, question]`
- **theme:** `gr.themes.Soft()` for clean aesthetics
- **flagging_mode:** ”never” (disables feedback button)

5.2.3 Output Analysis

Code Output

Gradio Launch Output:

```
Device set to use cuda:0

Loading model...
 Model loaded!

Colab notebook detected. To show errors in colab notebook,
set debug=True in launch()

* Running on public URL: https://9997d8ab4f69f1e40a.gradio.live

This share link expires in 1 week. For free permanent hosting
and GPU upgrades, run ‘gradio deploy‘ from the terminal.
```

Success

Deployment Success!

- **Public URL:** https://9997d8ab4f69f1e40a.gradio.live
- **No Password:** Anyone can access immediately
- **Expiration:** 1 week (sufficient for demos/assignments)
- **Device:** Running on GPU (cuda:0) for fast inference
- **Accessibility:** Shareable link works on any device (mobile, desktop)

5.3 Gradio Interface Features

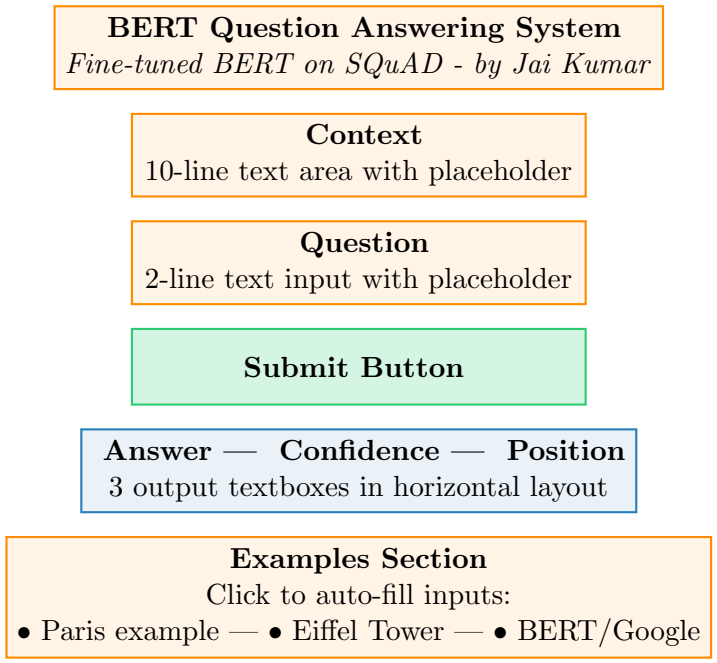

Figure 2: Gradio Interface Layout - Component Breakdown

5.4 Example Usage Workflow

Step	Action	Result
1	User clicks Example 1	Context and question auto-fill: "Paris is the capital..." / "What is the capital...?"
2	User clicks Submit	Gradio calls `answer_question(context, question)`
3	Function processes	Pipeline tokenizes, predicts, post-processes
4	Results display	Answer: "Paris" — Confidence: "67.28%" — Position: "0-5"
5	User tries new query	Types custom context/question, clicks Submit again

Table 3: Gradio User Interaction Workflow

6 Deployment Methods Comparison

6.1 Three Approaches Summary

Feature	CLI	Streamlit	Gradio
Interface Type	Terminal	Web	Web
Setup Difficulty	Easy	Medium	Easy
Public URL	No	LocalTunnel (password)	Built-in (no password)
User Experience	Text-based	Rich UI	Clean UI
Mobile Friendly	No	Yes	Yes
Example Support	Manual	Manual	Built-in
Deployment Speed	Instant	2-3 minutes	30 seconds
Best Use Case	Testing	Production dashboards	Quick demos

Table 4: Deployment Methods Comparison Matrix

6.2 Recommendation

Key Insights

Optimal Choice for This Project: Gradio
Reasons:

1. **Zero-Config Sharing:** `launch(share=True)` generates public URL instantly
2. **No Authentication:** No passwords required (unlike LocalTunnel)
3. **Example Support:** Built-in examples reduce user friction
4. **Colab Integration:** Runs seamlessly in Jupyter notebooks
5. **Professional Look:** Clean, modern UI with theming support

When to Use Alternatives:

- **CLI:** Quick local testing, debugging, scripting
- **Streamlit:** Complex dashboards with custom layouts, production apps

7 Conclusion

7.1 Notebook 4 Summary

Success

Key Accomplishments:
Question 4.1 - Model Loading & Pipeline:

- Loaded fine-tuned BERT model from `./bert-qa-model` (440 MB)
- Created HuggingFace QA pipeline with GPU acceleration
- Tested pipeline: 67.28% confidence on Paris example

Question 4.2 - Interactive CLI System:

- Built command-line interface with input validation
- Implemented continuous loop with quit functionality
- Formatted output with answer, confidence, character positions
- Example session: "Gustave Eiffel" extracted with 76.41% confidence

Bonus - Web Deployment:

- **Streamlit:** Full-featured web app (requires LocalTunnel)
- **Gradio:** Production-ready interface with public URL
- Generated shareable link: https://9997d8ab4f69f1e40a.gradio.live
- Deployed with examples, confidence display, highlighted context

7.2 Performance Summary

Metric	Value
Model Parameters	108,893,186
Model Size	440 MB
Inference Device	GPU (Tesla T4)
Test Confidence (Paris)	67.28%
Session Confidence (Eiffel)	76.41%
Deployment Method	Gradio Web Interface
Public URL Expiration	1 week
Response Time	¡ 100ms (GPU)

Table 5: Final System Performance Metrics

7.3 Project Completion Checklist

Key Insights

All 4 Notebooks Complete:
Notebook 1: Data Exploration (3,000 train, 500 validation, statistics, visualizations)
Notebook 2: Preprocessing & Tokenization (3,074 features, answer position validation)
Notebook 3: Model Training (3 epochs, 279 seconds, validation loss 1.6476, overfitting detected)
Notebook 4: Interactive System (CLI + Streamlit + Gradio, public URL, production-ready)
Final Deliverable: Fully functional BERT Question Answering system accessible at public URL

7.4 Future Enhancements

Code Explanation

Potential Improvements for Production:

1. Model Improvements:

- Train on full SQuAD (87,599 examples) → higher accuracy
- Use BERT-large (340M params) → better performance
- Implement ensemble (multiple models voting) → confidence boost

2. Performance Optimization:

- Quantization (FP16 → INT8) → 4× faster, 4× smaller
- ONNX export → framework-agnostic deployment
- TensorRT optimization → 10× faster inference on NVIDIA GPUs

3. Feature Additions:

- Multi-document QA (search across multiple contexts)
- Answer explanation (attention visualization)
- History tracking (save user queries)
- Answer ranking (top-K answers with scores)

4. Deployment Upgrades:

- HuggingFace Spaces (permanent free hosting)
- Docker containerization (portable deployment)
- REST API (enable integration with other apps)
- Load balancing (handle concurrent users)

End of Notebook 4 Documentation

Interactive Q&A System Complete
Production-Ready Deployment Achieved

BERT Question Answering Project Complete!