📚 Project Documentation: Production-Ready BERT Sentiment Analysis API

This document provides in-depth technical details, architecture overview, setup guides, and troubleshooting information for the "**Production-Ready BERT Sentiment Analysis API**" project. It complements the README.md by offering a deeper dive into the project's internal workings.

Contents

1. Project Overview

This project implements an end-to-end sentiment analysis solution for movie reviews using a fine-tuned **BERT** model. It demonstrates a complete machine learning pipeline from data acquisition and preprocessing to model training, evaluation, and deployment as a real-time **RESTful API** using **FastAPI**.

For a concise overview, key features, and initial setup instructions, please refer to the main README.md file.

2. Project Architecture

2.1. High-Level Flow

The project follows a standard machine learning pipeline:

- **Data Processing**: Raw IMDB movie review data is loaded, cleaned, and split into training, validation, and test sets.
- **Model Building**: A **BERT**-based classifier (using Hugging Face's TFBertModel) is constructed.
- **Training & Evaluation**: The model is fine-tuned on the training data, monitored using callbacks, and comprehensively evaluated on the test set.
- **API Deployment**: The trained model is served via a **FastAPI** application, allowing real-time sentiment predictions through HTTP requests.

2.2. Module Breakdown

The project is structured into modular Python files, each with a specific responsibility:

- config.py: Centralizes all configuration settings.
- data_loader.py: Handles IMDB dataset operations (loading, cleaning, splitting).
- model.py: Defines the **BERT** model's architecture.
- trainer.py: Manages the model training pipeline.
- evaluator.py: Conducts model evaluation and visualization.
- app.py: Implements the **FastAPI** inference **API**.
- main.py: Orchestrates the entire training and evaluation workflow.

- test_api.py: A separate script for programmatically testing the local **API**.
- requirements.txt: Lists all project dependencies.
- .gitignore: Specifies files/folders to be ignored by Git.

3. Configuration Details

All key parameters and paths are managed centrally in config.py using dataclasses for clarity and easy modification.

- **ModelConfig**: Defines model-specific hyperparameters (e.g., model_name, max_length, learning_rate).
- **TrainingConfig**: Controls training strategies and callbacks (e.g., epochs, save_strategy, metric_for_best_model).
- **ProjectConfig**: Manages project directory structure and file paths (e.g., models_dir, outputs_dir).

4. Local Setup & Installation

Follow these steps to set up and run the project on your local machine.

Prerequisites:

- **Python 3.11**: Recommended for stable **TensorFlow/Text** compatibility.
- **Git**: For cloning the repository.
- **Visual Studio Code** (or any IDE): For code editing and terminal access.
- **Microsoft C++ Build Tools** (for Windows users): Essential for compiling certain Python packages that contain C/C++ extensions. Download from [Microsoft Visual C++ Build Tools]() (select "Desktop development with C++" workload during installation).

Installation Steps:

1. **Clone the repository:**
2. git clone https://github.com/EmadAliEmad/BERT-Fine_tuning-for-Movie-Sentiment-Analysis.git
3. cd BERT-Fine_tuning-for-Movie-Sentiment-Analysis

4. **Download the Trained Model (best_model.h5):**
   The trained model is not included in this repository due to its large size. It must be downloaded separately.

   - Go to the Kaggle Notebook output associated with this project: Kaggle Notebook Output (Version with Trained Model)

   - Download best_model.h5 from the models/ folder within that specific version's output.

   - Create a models/ directory in your local project root: mkdir models

   - Place the downloaded best_model.h5 file inside the models/ directory.

5. **Create and activate a Python virtual environment:**
   It's highly recommended to use a virtual environment to manage project dependencies isolation.

6. # Using Python 3.11 directly (if installed as 'python3.11' or 'py -3.11')

7. py -3.11 -m venv venv_api

8. # Or simply 'python -m venv venv_api' if 3.11 is your default Python.

9.

10. # Activate the virtual environment:

11. .\venv_api\Scripts\activate   # On Windows (Command Prompt or PowerShell)

12. source venv_api/bin/activate # On macOS/Linux/Git Bash

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Bash

IGNORE_WHEN_COPYING_END

13. **Install the required dependencies:**
    All project dependencies are listed in requirements.txt.

14. pip install -r requirements.txt

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Bash

IGNORE_WHEN_COPYING_END

(You may see WARNING messages about dependency conflicts; these can usually be ignored as long as no ERROR prevents installation. For local Windows setup, ensure C++ Build Tools are installed if compilation errors occur.)

5. How to Run Components

5.1. Training & Evaluation Pipeline (main.py)

The main.py script orchestrates the entire training and evaluation process. It's designed to either load an existing model or train a new one if not found.

To run the full pipeline (training if model not found, then evaluation):

```
python main.py
```

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Bash

IGNORE_WHEN_COPYING_END

Output: This will log detailed progress to the console (using **Rich**) and save logs to logs/app.log, training history to outputs/training_history.json, and generate interactive HTML plots in outputs/.

If best_model.h5 is not in models/, this will trigger a full training run (which is resource-intensive and requires a **GPU** or significant **CPU** time).

5.2. FastAPI Inference API (app.py)

The app.py script runs the **FastAPI** server for real-time sentiment predictions.

To run the **API** locally:

```bash
uvicorn app:app --reload --host 0.0.0.0 --port 8000
```

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Bash

IGNORE_WHEN_COPYING_END

- uvicorn app:app: Tells **Uvicorn** to run the app instance (our **FastAPI** application) from the app.py file.
- --reload: Restarts the server automatically when code changes are detected (useful for development).
- --host 0.0.0.0: Makes the server accessible from all network interfaces (allows external connections).
- --port 8000: Specifies the port the **API** will listen on.

Output: The **API** will be running on http://127.0.0.1:8000. Keep this terminal open while using the **API**.

Note: If running on a remote server/notebook (like Kaggle), you would typically use **ngrok** (as demonstrated in the Kaggle Notebook) to expose this local port to a public URL.

6. API Reference

The **FastAPI** application automatically generates interactive **API** documentation.

6.1. Base URL

- Local: http://127.0.0.1:8000
- Documentation (Swagger UI): http://127.0.0.1:8000/docs

6.2. Endpoint: GET /health

- **Purpose**: Provides a simple health check for the **API**. It verifies if the server is running and if the **BERT** model and tokenizer have been successfully loaded into memory.
- **Request**:

- **Method**: GET
- **URL:** /health
- **Parameters:** None

- **Response**:
  - **Status Code**: 200 OK
  - **Body (JSON)**:
  - {
  - "status": "ok",
  - "model_loaded": true,
  - "tokenizer_loaded": true
  - }

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Json

IGNORE_WHEN_COPYING_END

6.3. Endpoint: POST /predict

- **Purpose**: Accepts text input(s) and returns sentiment prediction(s) (Positive/Negative) along with a confidence score.
- **Request**:
  - **Method**: POST
  - **URL**: /predict
  - **Content-Type**: application/json
  - **Body (JSON - adheres to PredictionRequest schema)**:
  - {
  - "texts": [

- "This movie was an absolute masterpiece! I loved every moment of it.",
- "The plot was confusing and the acting was terrible. A complete waste of my time and money."
-]
- }

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Json

IGNORE_WHEN_COPYING_END

- **Response**:
  - **Status Code**: 200 OK
  - **Body (JSON - adheres to PredictionResponse schema)**:
  - {
  - "predictions": [
  - {
  - "text": "This movie was an absolute masterpiece! I loved every moment of it.",
  - "sentiment": "Positive",
  - "confidence": 0.998
  - },
  - {
  - "text": "The plot was confusing and the acting was terrible. A complete waste of my time and money.",
  - "sentiment": "Negative",
  - "confidence": 0.995
  - }

-]
- }

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution. Json

IGNORE_WHEN_COPYING_END

- **Error Status Code**: 422 Unprocessable Entity (if request body format is incorrect).

7. Codebase Details (Module by Module)

Here's a detailed description of each Python module in the project:

- app.py:
  - **Role**: The core **FastAPI** application script.
  - **Key components**: Defines **API** endpoints (/health, /predict), handles model/tokenizer loading on startup (@app.on_event("startup")), and contains the inference logic for sentiment prediction. It includes PredictionRequest and PredictionResponse **Pydantic** models.
- config.py:
  - **Role**: Centralized configuration management.
  - **Key components**: Contains ModelConfig (model hyperparameters), TrainingConfig (training strategies), and ProjectConfig (file paths and directory management). Uses dataclasses for clean structure.
- data_loader.py:
  - **Role**: Data processing pipeline.
  - **Key components**: DataProcessor class with methods for load_imdb_dataset (decoding IMDB integer sequences to text), clean_text (standardizing text preprocessing), and create_data_splits (generating train/val/test sets).
- evaluator.py:

- **Role**: Model evaluation and visualization.
- **Key components**: ModelEvaluator class provides predict method (for test data), evaluate_model (calculates metrics like accuracy, F1-score, ROC AUC, and generates classification report/confusion matrix), and plotting functions (plot_training_history, plot_confusion_matrix, plot_roc_curve) using **Plotly** for interactive HTML outputs.

- logger.py:
  - **Role**: Professional logging setup.
  - **Key components**: setup_logging function configures **Loguru** with **RichHandler** for visually appealing console output and file logging (e.g., logs/app.log), ensuring all project activities are monitored.
- main.py:
  - **Role**: The main script orchestrating the full project pipeline.
  - **Key components**: Calls setup_logging, print_project_info, and controls the sequence of data loading, model building/loading (checking for existing model on disk), training, and evaluation. It links all other modules to execute the end-to-end workflow.
- model.py:
  - **Role**: Defines the **BERT** model architecture.
  - **Key components**: BERTSentimentClassifier (a custom tf.keras.Model subclass) which loads TFBertModel from **transformers** and adds a classification head. BERTModelBuilder provides a static method build_functional_model for constructing the model using **Keras Functional API**.
- trainer.py:
  - **Role**: Encapsulates the model training pipeline.
  - **Key components**: BERTTrainer class handles compile_model (optimizer, loss, metrics setup), setup_callbacks (e.g., ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, TensorBoard, CSVLogger), and the train method (fitting the model on data). It also manages save_training_artifacts for history and configurations.

- requirements.txt:
    - **Role**: Lists all Python library dependencies with specific versioning for exact environment replication.
- test_api.py:
    - **Role**: A standalone Python script for programmatically testing the locally running **FastAPI API** endpoints (health check and prediction).
- .gitignore:
    - **Role**: Specifies files and directories that Git should ignore and not include in the version control history (e.g., large model files, virtual environments, log files).

8. Common Issues & Troubleshooting

This section details common problems encountered during this project's development and their solutions.

- **ModuleNotFoundError: No module named 'xyz'**:
    - **Reason**: The required Python library (xyz) is not installed in the active virtual environment.
    - **Solution**: Run pip install -r requirements.txt after activating your virtual environment. Ensure !pip install pyngrok -q is run directly in the Kaggle cell that imports it.
- **TypeError: Object of type float32 is not JSON serializable**:
    - **Reason**: **TensorFlow/NumPy** float32 data types are not natively supported by the standard JSON format.
    - **Solution**: Convert float32 values to standard Python float before JSON serialization (implemented in trainer.py in save_training_artifacts).
- **huggingface_hub.utils._validators.HFValidationError: Repo id must use alphanumeric chars...**:
    - **Reason**: Attempting to re-initialize a tokenizer with an already-loaded tokenizer object itself, instead of its string name.

    - **Solution**: Ensure the tokenizer is passed directly as an initialized object where needed, or with its string name when initializing from scratch (fixed in evaluator.py).
- **NameError: name 'tf' is not defined**:
    - **Reason**: The **tensorflow** library was not imported (e.g., import tensorflow as tf) in the specific .py file where tf was being used.
    - **Solution**: Add import tensorflow as tf at the beginning of the relevant module (fixed in evaluator.py).
- **ValueError: Unknown layer: 'TFBertModel'. Please ensure you are using a keras.utils.custom_object_scope...**:
    - **Reason**: **Keras** does not natively recognize TFBertModel (which comes from Hugging Face **transformers**) when loading a saved model.
    - **Solution**: Pass custom_objects={'TFBertModel': TFBertModel} as an argument to tf.keras.models.load_model() when loading the best_model.h5 file (fixed in main.py and app.py).
- **ValueError: Input 0 of layer "bert_sentiment_classifier" is incompatible with the layer: expected shape=(None, 128), found shape=(None, X)**:
    - **Reason**: The tokenizer was not padding input sequences to the exact max_length (128) expected by the **BERT** model.
    - **Solution**: Explicitly set padding='max_length' in the tokenizer's call (fixed in app.py).
- **Health Check Status: 404 or JSONDecodeError during API testing**:
    - **Reason**: This often indicates the **API** is not reachable at the specified **URL**. Common causes include:
        - **Ngrok URL Mismatch**: The ngrok_url in the testing script/browser is outdated.
        - **Ngrok Service Down**: The **ngrok** tunnel itself is not active or crashed.
        - **Uvicorn Server Down**: The **FastAPI** application (**Uvicorn**) crashed or is not running.
    - **Solution**:

        - Always copy the NEW **Ngrok Tunnel URL** from the output of the **API** startup cell (Cell 14) and paste it into the testing cell (Cell 15).
        - Ensure no other **ngrok** sessions are running (check **ngrok** dashboard and kill any active sessions).
        - Ensure the **API** startup cell (Cell 14) executes successfully and continuously.

- **The ngrok process errored on start: authentication failed: Your account is limited to 1 simultaneous ngrok agent sessions. (ERR_NGROK_108)**:
    - **Reason**: Your **ngrok** free account only allows one active tunnel at a time. A previous session (from another notebook, local machine, or a crashed Kaggle session) might still be active.
    - **Solution**: Manually stop all active **ngrok** sessions from your **ngrok** dashboard. The ngrok.kill() command in our setup cells also helps in cleaning up residual processes within the Kaggle environment.

9. Future Enhancements

The project provides a solid foundation. Here are potential areas for further development:

Web User Interface (Frontend):

- Integrate a user-friendly web interface using **Streamlit** (already in requirements.txt), **Flask**, or **Dash** to allow users to input text and see real-time sentiment predictions.

Model Optimization & Fine-tuning:

- Experiment with different **BERT** variants (e.g., bert-large-uncased for potentially higher accuracy, but more resources) or other **Transformer** models.
- Explore advanced fine-tuning techniques (e.g., learning rate schedulers, weight decay).
- Fine-tune on a domain-specific dataset if targeting a particular industry (e.g., financial sentiment).

Advanced NLP Features:

- Implement more sophisticated text preprocessing (e.g., handling slang, emojis).

- Expand to emotion detection (e.g., joy, sadness, anger) or aspect-based sentiment analysis.
- Support multi-lingual sentiment analysis by using multi-lingual **BERT** models.

Deployment & MLOps:

- Containerize the application using **Docker** for easier packaging and deployment across different environments.
- Deploy the **API** to a permanent cloud platform (e.g., Hugging Face Spaces, **AWS Lambda**, **Google Cloud Run**, **Azure App Service**) for continuous availability.
- Add **API** authentication/authorization, rate limiting, and more detailed **API** logging for production readiness.

Monitoring & Alerting:

- Implement model performance monitoring and data drift detection.

10. Contributing Guidelines

We welcome contributions to this project!

1. Fork the repository.
2. Clone your forked repository: git clone https://github.com/YourUsername/BERT-Fine_tuning-for-Movie-Sentiment-Analysis.git
3. Create a new branch for your feature or bug fix: git checkout -b feature/your-feature-name
4. Make your changes, add comments, and ensure code quality.
5. Test your changes thoroughly.
6. Commit your changes: git commit -m "feat: Add new feature X" (use conventional commits).
7. Push to your branch: git push origin feature/your-feature-name
8. Open a Pull Request to the main branch of the original repository.

11. License & Acknowledgments

This project is licensed under the **MIT License**.

Acknowledgments:

- **Kaggle**: For providing the **GPU**-enabled notebook environment.
- **Hugging Face**: For the excellent **Transformers** library and pre-trained **BERT** models.
- **TensorFlow**: For the deep learning framework.
- **FastAPI**: For the fast and modern web framework.
- **Rich** & **Loguru**: For enhancing logging and console output.