

FINAL MODULE'S PROJECT REPORT

Module : Advanced AI

FINE-TUNING BERT ON THE AG NEWS DATASET AND DEPLOYING A REAL-TIME TEXT CLASSIFICATION SYSTEM

Prepared By
CHAFIK Anas
BOUCHTI Abdelkebir
ATIF Youness

Supervised By :
Prof Soufiane HAMIDA

Master M2 SDIA

Table of content

Table of content	1
Table of figures	1
Abstract	2
Introduction	2
System Design	2
Methodology	4
Tools Used in This Project	5
1. Core AI & Data Processing Stack	5
2. Web & API Infrastructure	6
3. Deployment	7
4. Development Utilities	7
Web Interface Overview	8
Future Improvements	10
Conclusion	10
Resources	11

Table of figures

Figure 1 : use case diagram	2
Figure 2 : workFlow	4
Figure 3 : web interface	8
Figure 4 :Result panel	9
Figure 5 :Classification History Log	9

Abstract

This project presents a comprehensive implementation of fine-tuning a BERT model for multi-class text classification using the AG News dataset, followed by deployment as a production-ready web application. The AG News dataset contains 120,000 training and 7,600 testing samples across four news categories: World, Sports, Business, and Sci/Tech. The project demonstrates the complete machine learning lifecycle from data preprocessing and model training to API development and containerized deployment. A BERT-base-uncased model was fine-tuned using TensorFlow, achieving approximately 94.6% accuracy on the test set after three epochs of training. The trained model was then encapsulated within a Flask web application and containerized using Docker for scalable deployment. This end-to-end implementation showcases practical MLOps principles for transitioning machine learning models from development to production environments.

Introduction

Text classification remains a fundamental task in natural language processing with numerous practical applications including news categorization, sentiment analysis, and content moderation. The emergence of transformer-based architectures like BERT has revolutionized this field by enabling models to capture deep contextual relationships within text. This project focuses on fine-tuning a pre-trained BERT model specifically for the AG News classification task, which involves categorizing news articles into four distinct topics. Beyond model development, the project emphasizes the operationalization of machine learning models through web service deployment. The integration of modern deep learning techniques with production deployment practices addresses a critical gap between research prototypes and practical applications. By providing both the training pipeline and deployment infrastructure, this work offers a template for implementing similar text classification systems in real-world scenarios where automated content categorization is required.

System Design

The architectural design of the BERT News Classification System follows a client-server model with clearly defined roles and responsibilities for both end-users and system administrators. As illustrated in the use case diagram below, the system accommodates two primary actors with distinct interaction patterns: the end-user who submits news text for classification, and the system administrator responsible for deployment and maintenance operations.

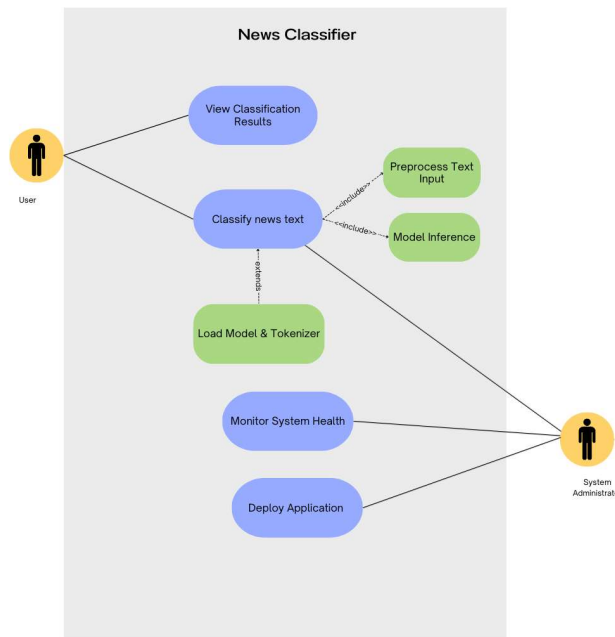


Figure 1 : use case diagram

The system's core functionality, represented by the "Classify News Text" use case, orchestrates a three-stage pipeline that begins with text preprocessing, proceeds through model inference, and concludes with result presentation. This primary workflow is initiated when users submit news articles through the web interface, triggering automated processes that clean and tokenize input text, perform BERT-based inference, and return classification results complete with confidence metrics and probability distributions across all categories. The system's design incorporates extension points for model initialization, ensuring that both the BERT model and tokenizer are loaded into memory during system startup, thereby optimizing inference latency for subsequent user requests.

Complementing the user-facing functionality, the system supports comprehensive administrative operations through three distinct use cases. System administrators can deploy the application container using Docker, monitor system health and performance metrics, and update model versions without disrupting service availability. This separation of concerns between user interaction and system management reflects sound architectural principles, enabling independent evolution of frontend features and backend infrastructure. The modular decomposition evident in the "include" relationships between primary and supporting use cases facilitates maintenance and testing, as each component—text preprocessing, model inference, and result formatting—can be developed and validated independently while contributing to the cohesive user experience of news classification. This design approach ensures that the system remains both accessible to end-users seeking news categorization and manageable for administrators responsible for its operational integrity.

Methodology

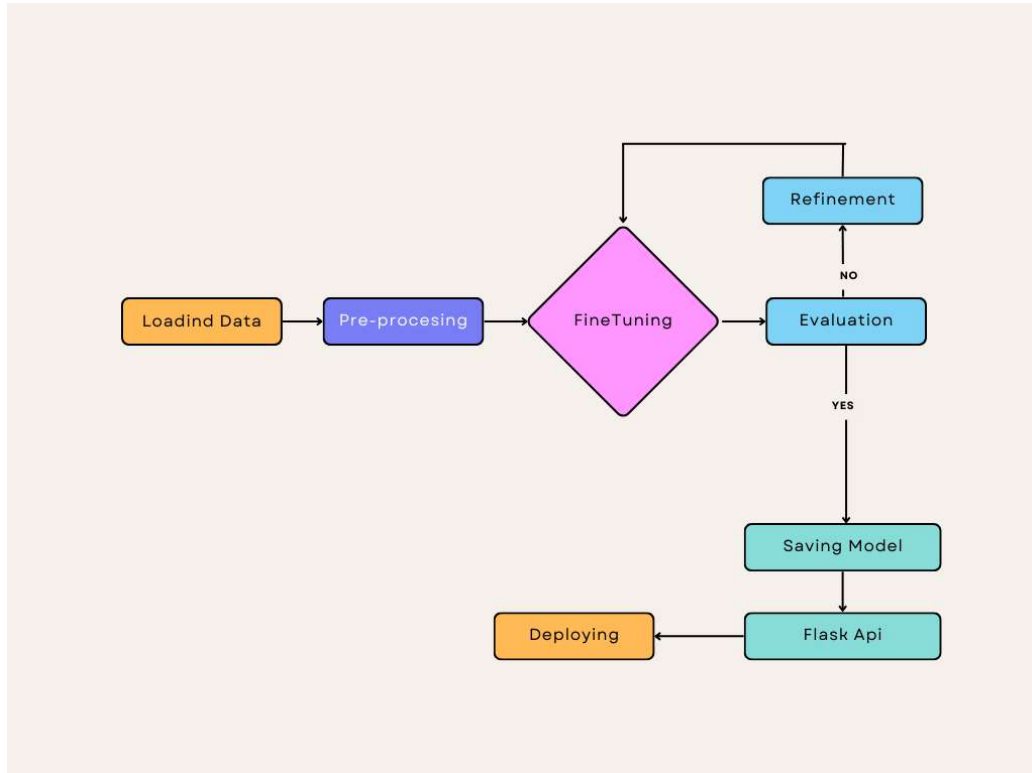


Figure 2 : workflow

The methodology follows a systematic pipeline beginning with data acquisition and preprocessing, proceeding through model development and evaluation, and concluding with API development and deployment. The AG News dataset was loaded from the Hugging Face datasets library, comprising 120,000 training examples and 7,600 test examples distributed equally across four categories. Each sample contains text content and corresponding categorical labels. A text cleaning function was implemented to standardize the input data by converting to lowercase, removing HTML tags, eliminating URLs and email addresses, and filtering non-alphanumeric characters while preserving essential punctuation. This preprocessing step ensures consistent input formatting for the BERT tokenizer and reduces noise in the training data.

The model architecture builds upon the BERT-base-uncased pre-trained model, augmented with a dense classification layer for the four news categories. The text was tokenized using the BERT tokenizer with a maximum sequence length of 128 tokens, and the dataset was converted into TensorFlow format with appropriate batching and shuffling. The model was trained for three epochs using the Adam optimizer with a learning rate of $1e-5$ and sparse categorical crossentropy loss function. Training was conducted on a GPU-accelerated environment, with

each epoch requiring approximately 50 minutes to complete. The model achieved a training accuracy of 95.8% and a test accuracy of 94.6%, demonstrating effective generalization without significant overfitting despite the relatively short training duration.

Following training, both the model and tokenizer were saved to disk in standardized formats. The model was saved as a TensorFlow SavedModel with timestamped naming for version tracking, while the tokenizer was saved using Hugging Face's serialization methods. A custom Flask web application was then developed to serve the model through a REST API. The application loads the saved model and tokenizer at startup, providing efficient inference without repeated loading overhead. The API accepts text input through HTTP POST requests, processes the input through the same tokenization pipeline used during training, and returns classification results with confidence scores. The frontend interface provides a simple web form for user interaction, displaying predicted categories and confidence levels.

For deployment, the application was containerized using Docker to ensure consistent execution across different environments. A multi-stage Dockerfile was created to install dependencies, copy application code and model artifacts, and configure the runtime environment. Docker Compose was employed to simplify orchestration, exposing the service on port 8000 with automatic restart policies. This containerized approach encapsulates all dependencies and configurations, enabling seamless deployment on various platforms including cloud services and local servers while maintaining reproducibility and scalability.

Tools Used in This Project

1. Core AI & Data Processing Stack

Tool	Role in Project
Python	Primary language for data preprocessing, model training, and API development.
TensorFlow	Deep learning framework used for building, training, and serving the fine-tuned BERT model.

Hugging Face (HF)	<p>Transformers: Provided pre-trained BERT weights and tokenizers.</p> <p>Datasets: Enabled efficient loading and processing of the AG News dataset.</p>
NumPy	Supported low-level tensor operations and numerical computations throughout the pipeline.
Scikit-learn	Provided evaluation metrics (F1-Score, Precision, Recall) and data splitting utilities.

2. Web & API Infrastructure

Tool	Role in Project
Flask	Served as the lightweight backend framework for the classification API.
Gunicorn	Acts as the WSGI HTTP Server to handle concurrent requests in a production-style environment.
Frontend Suite	<p>HTML/CSS: Structured and styled the web interface.</p> <p>JavaScript: Managed dynamic API calls and asynchronous UI updates.</p>

3. Deployment

Tool	Role in Project
Docker	Containerized the application to ensure a consistent environment across different OS platforms.
Docker Compose	Orchestrated the multi-container setup (e.g., linking the API and web services).

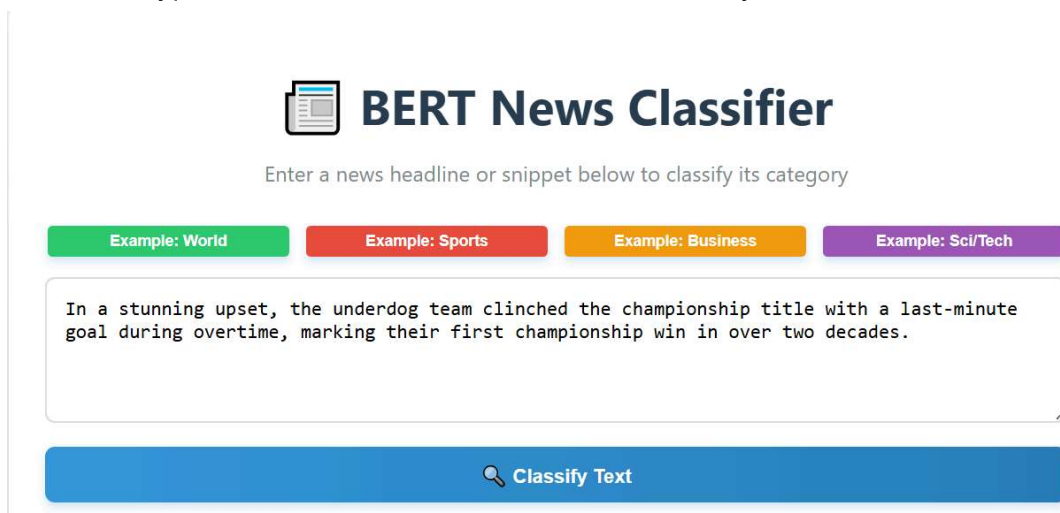
4. Development Utilities

Tool	Role in Project
VS Code	Primary Integrated Development Environment (IDE) for coding and debugging.
Git	Managed version control, enabling experiment tracking and code history preservation.
Google Colab	Provided cloud-based GPU acceleration (Tesla T4) for intensive BERT fine-tuning.

Web Interface Overview

The AG News BERT Classifier features a clean, user-friendly web interface designed for intuitive text classification. The interface includes:

Title & Input Area : This section serves as the starting point for user interaction. It is designed to immediately communicate the application's purpose and provide a clear, functional area for input. The prominent title, "BERT News Classifier," establishes the tool's context. Directly below, a concise instruction guides the user to enter their news text. The large, dedicated text input field is the primary interactive element. To aid the user and set expectations, example headlines for each potential category (World, Sports, Business, Sci/Tech) are displayed beneath the input box, offering a practical reference for the type of content the classifier is built to analyze.



The screenshot displays the 'BERT News Classifier' web interface. At the top, there is a logo of a document with a magnifying glass and the title 'BERT News Classifier' in a large, bold, dark blue font. Below the title is a subtitle: 'Enter a news headline or snippet below to classify its category'. Underneath the subtitle are four colored buttons: 'Example: World' (green), 'Example: Sports' (red), 'Example: Business' (orange), and 'Example: Sci/Tech' (purple). Below these buttons is a large text input field containing the example text: 'In a stunning upset, the underdog team clinched the championship title with a last-minute goal during overtime, marking their first championship win in over two decades.' At the bottom of the input area is a blue button with a magnifying glass icon and the text 'Classify Text'.

Figure 3 : Title & Input Area

Classification Results Panel : The Results Panel is dynamically updated to display the outcome of each classification request. Its purpose is to present the model's prediction in both a straightforward and detailed manner. The "Predicted Category" is highlighted prominently, showing the most likely news category alongside a confidence percentage, giving the user a quick, clear answer. Below this, the "Probability Distribution" provides a complete breakdown of the model's confidence across all possible categories. This transparency allows users to assess not just the top result but also the relative strength of alternative classifications, offering deeper insight into the model's decision-making process.

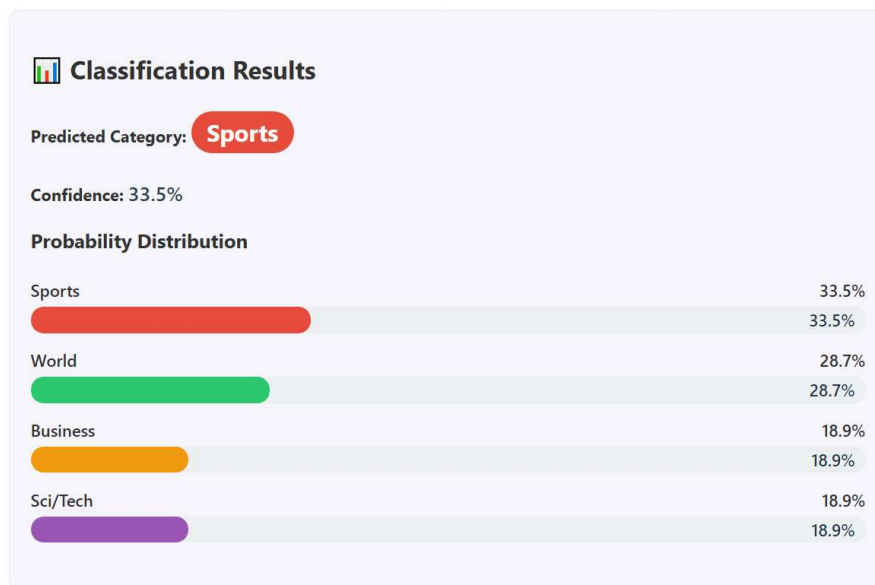


Figure 4 : Classification Results Panel

Classification History Log : Located at the bottom of the interface, the History Log maintains a persistent, chronological record of all classification activity within the current session. This feature enhances usability by allowing users to review past queries without needing to re-enter text. Each entry in the log includes a snippet of the submitted news text, the predicted category with its confidence score, and a precise timestamp. This creates an audit trail and a convenient reference point, making it easy to compare different results or revisit previous analyses.

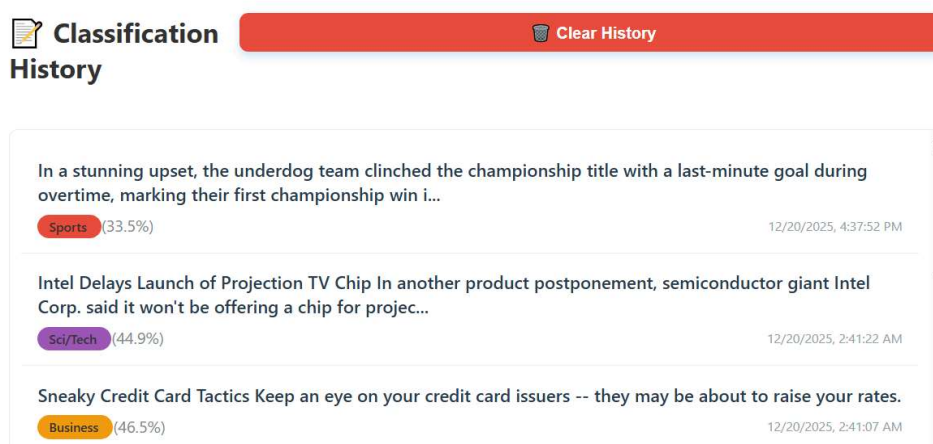


Figure 5 : Classification History Log

Overall Flow: The user enters text into the input area and submits it. The interface then updates the Classification Results panel with the new prediction and details, while simultaneously adding the query and result to the bottom of the Classification History log. The design uses clear headings, spacing, and visual hierarchy (like bolding the primary prediction) to make the information easily scannable.

Future Improvements

This BERT News Classifier provides a solid foundation for text classification. To enhance its capabilities further, consider these expansions:

Model Variants : Experiment with different BERT architectures like RoBERTa, DistilBERT, or ALBERT for improved accuracy or faster inference times.

Category Expansion : Extend beyond the current 4 classes by training on datasets with more specific categories like Politics, Entertainment, Health, or Technology subdomains.

Multilingual Support : Implement multilingual BERT models to classify news in different languages, expanding the application's global usability.

Real-time Updates : Integrate live news feeds to demonstrate real-world classification performance on current headlines.

Advanced Analytics : Add metrics dashboard showing model performance over time, confusion matrices, and accuracy trends across categories.

Conclusion

This project successfully demonstrates a complete workflow for developing and deploying a BERT-based text classification system for news categorization. The implementation showcases how state-of-the-art natural language processing models can be adapted to specific domains through fine-tuning and integrated into production environments via containerized web services. The achieved accuracy of 94.6% on the test set validates the effectiveness of the BERT architecture for this classification task, while the modular design of the training pipeline and deployment infrastructure ensures maintainability and extensibility. The containerized deployment approach addresses critical challenges in machine learning operations, including environment consistency, scalability, and reproducibility. This work provides a practical template for similar text classification applications, bridging the gap between experimental machine learning and production systems. The project highlights the importance of considering both model performance and operational requirements when developing machine learning

applications, ensuring that advanced algorithms can deliver value in real-world scenarios through reliable and accessible interfaces.

Resources

[The Secret to 90%+ Accuracy in Text Classification](#)

[BERT Model - NLP - GeeksforGeeks](#)

[DeepSeek](#)