Project Documentation: TEXTRON

Next Word Prediction Server with Model Evaluation Framework

# 1. Introduction

This document provides a comprehensive overview of the **Next Word Prediction Server**, a backend service designed to power a conversational frontend application built with **Next.js**. The server leverages lightweight language models (LLMs) such as **GPT-2** and others to generate next-word predictions in real-time. Initially, GPT-2 was used for this purpose, but during testing, issues like hallucinations and nonsensical outputs were identified. To address these challenges, we implemented a robust evaluation framework that uses various metrics to assess model performance and reliability.

The primary objective of this project is to provide a scalable, reliable, and accurate backend API endpoint that integrates seamlessly with the frontend application to deliver high-quality next-word predictions. This documentation outlines the project's objectives, development process, tools, evaluation metrics, and deployment strategy.

# 2. Project Objectives

The key objectives of this project are:

1. **Real-Time Next Word Prediction**: Develop an API endpoint that generates accurate and contextually relevant next-word predictions for user inputs.

2. **Model Reliability**: Address issues like hallucinations, factual inaccuracies, and grammatical errors by rigorously evaluating models using predefined metrics.

3. **Scalability**: Ensure the backend can handle multiple concurrent requests efficiently, supporting both local and cloud-based deployments.

4. **Integration with Frontend**: Provide a seamless integration between the backend API and the Next.js frontend, which includes a landing page and a conversation component for user interaction.

5. **Model Optimization**: Identify and deploy the most suitable lightweight LLM for next-word prediction tasks while maintaining low latency and resource efficiency.

# 3. Timeline

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Phase 1: Research | Investigated lightweight LLMs (e.g. | GPT-2 | TinyLlama | FLAN-T5) and identified evaluation metrics. | Week 1–2 |
| Phase 2: Development | Developed Flask-based backend API for next-word prediction and integrated it with Next.js frontend. | Week 3–5 |  |  |  |
| Phase 3: Testing | Tested the API with various test cases | identified hallucinations | and implemented evaluation metrics. | Week 6–7 |  |
| Phase 4: Optimization | Evaluated multiple models using metrics and fine-tuned parameters for optimal performance. | Week 8 |  |  |  |
| Phase 5: Deployment | Deployed the backend on a cloud platform and integrated it with the Next.js frontend. | Week 9 |  |  |  |

# 4. Tools and Technologies

## 4.1 Backend

- **Flask**: Lightweight Python web framework for building the API.

- **Transformers**: Hugging Face library for loading and running pre-trained models.

- **Sentence Transformers**: For semantic similarity calculations.

- **Rouge-Score**: For summarization quality evaluation.

- **Psutil**: For monitoring memory and CPU usage during inference.

- **LanguageTool**-Python: For detecting grammar and syntax errors.

## 4.2 Frontend

- **Next.js**: React-based framework for building the frontend application, including the landing page and conversation component.

- **Next Js** **Actions**: For making API calls from the frontend to the backend.

## 4.3 Additional Tools

- Google Colab: For testing and evaluating models locally.

- Hugging Face Hub: For accessing lightweight LLMs and datasets.

- Perspective API (Optional): For toxicity detection.

- Fairness Indicators (Optional): For bias evaluation.

# 5. Environment Setup

## 5.1 Prerequisites

- **Python Version**: 3.8 or higher.

- **Node.js Version**: 16 or higher.

- **Hardware Requirements**:

- Minimum: 8 GB RAM, 4 CPU cores.

- Recommended: GPU-enabled machine for faster inference (e.g., NVIDIA GPUs with CUDA support).

- **Operating System**: Linux, macOS, or Windows.

## 5.2 Backend Setup

1. Install Dependencies :

```bash

pip install flask transformers sentence-transformers rouge-score nltk psutil language-tool-python

```

2. Run the Flask Server :

```bash

python app.py

```

5.3 Frontend Setup

1. Install Dependencies :

```bash

npm install

```

2. Start the Next.js Application :

```bash

npm run dev

```

## 5.4 Integration

6. Metrics Overview- Model Evaluation Framework

To evaluate the performance of the models, we implemented the following metrics:

## 6.1 Accuracy Metrics

- Factual Accuracy: Measures the percentage of responses that are factually correct.

- Semantic Similarity: Evaluates how closely the generated text matches the expected output using cosine similarity.

- Hallucination Rate: Tracks instances where the model generates nonsensical or incorrect information.

## 6.2 Coherence and Fluency

- Perplexity: Assesses how well the model predicts the next word in a sequence.

- Grammar and Syntax: Detects grammatical errors using tools like LanguageTool.

## 6.3 Performance Metrics

- Inference Time: Measures the time taken to generate responses.

- Memory Usage: Tracks the computational resources consumed during inference.

## 6.4 Task-Specific Metrics

- Summarization Quality: Uses ROUGE scores to evaluate summarization tasks.

- Question-Answering Accuracy: Tests the model's ability to extract correct answers from context.

## 6.5 Ethical Metrics

- Toxicity: Identifies harmful or offensive language in outputs.

- Bias Detection: Analyses outputs for gender, racial, or other biases.

# 7. Testing and Results

## 7.1 Test Cases

We tested the API with various inputs to identify hallucinations and improve model reliability. Below are some examples:

Test Case 1: Factual Query

- Input: `"The capital of France is"`

- Output: `"Paris"` (Correct)

- Analysis: The model correctly predicted the next word.

Test Case 2: Creative Writing

- Input: `"Once upon a time"`

- Output: `"Once upon a time, there was a brave knight."` (Coherent)

- Analysis: The model generated a contextually relevant continuation.

Test Case 3: Hallucination

- Input: `"Artificial Intelligence refers to"`

- Output: `"AI is a type of fruit."` (Incorrect)

- Analysis: The model hallucinated an unrelated response.

## 

## 7.2 Evaluation Results

|  |  |  |
| --- | --- | --- |
| Factual Accuracy (%) | 66.67 |  |
| Semantic Similarity | 0.85 |  |
| Perplexity | 25.34 |  |
| Grammar Errors | 0 |  |
| Inference Time (seconds) | 1.23 |  |
| Memory Usage (MB) | 512.45 |  |
| Hallucination Rate (%) | 33.33 |  |
| Summarization Quality (ROUGE-L) | 0.78 |  |
|  |  |  |

# 8. Challenges and Solutions

8.1 Hallucinations

- Challenge: GPT-2 occasionally generated nonsensical or factually incorrect outputs.

- Solution: Implemented metrics like factual accuracy and hallucination rate to identify and mitigate this issue. Switched to lightweight models like TinyLlama for better control.

8.2 Latency

- Challenge: High inference time for large models impacted user experience.

- Solution: Optimized model parameters (e.g., `max\_length`, `temperature`) and deployed lightweight models for faster responses.

## 8.3 Integration

- Challenge: Ensuring seamless communication between the Flask backend and Next.js frontend.

- Solution: Used Axios for API calls and standardized JSON responses for consistency.

## 9. Future Enhancements

1. Advanced Models: Explore newer lightweight models like Llama 2 or Mistral for improved performance.

2. User Feedback Loop: Allow users to rate predictions and use feedback to fine-tune the model iteratively.

3. Multi-Language Support: Extend the server to support multilingual next-word predictions.

4. Monitoring and Alerts: Integrate monitoring tools like Prometheus and Grafana to track server performance and detect anomalies.

# 10. Conclusion

The TextRon project provides a reliable and scalable backend solution for powering conversational applications. By addressing challenges like hallucinations and latency through rigorous testing and optimization, this project ensures high-quality predictions while maintaining low resource usage. The integration with a Next.js frontend enables a seamless user experience, making this solution ideal for real-world applications.

For further details, visit the project repository on [[GitHub]](https://github.com/flying-spagetti/TextRoid).

Author: Gnaneswar Lopinti

Version: 1.0

Last Updated: 25-02-2025