CSCE 689 Directed Studies – SummarizeME Project Report

Under the guidance of Prof. Duncan Walker
Akanksha Shah (UIN - 136005001)

Abstract

Text summarization is a fundamental task in natural language processing that aims to condense lengthy content into clear, concise summaries while preserving essential information. With the overwhelming volume of online material available today, efficiently extracting key insights from long web pages poses a significant challenge. SummarizeME addresses this need by providing a browser extension that generates brief, easily digestible summaries directly from any webpage. The system consists of a Chrome Extension as the front-end interface and a back-end summarization service powered by a specialized language model. Its ability to produce summaries of varying lengths offers flexibility for diverse user preferences and use cases. To handle larger documents effectively, the system employs a hierarchical summarization approach: the content is first divided into manageable segments, each segment is summarized independently, and the resulting partial summaries are then synthesized into a final, comprehensive summary. This report describes the system's design, architecture, model development, and implementation pipeline, establishing a strong foundation for ongoing enhancements and future extensions.

Introduction

Text summarization enables users to easily grasp the key elements of lengthy materials, increasing accessibility and reducing cognitive burden. With the exponential proliferation of online content such as blogs, news articles, documentation, research papers, and chat transcripts, tools that generate succinct, coherent summaries immediately within the browsing environment have become increasingly useful. SummarizeME is designed to address this need by providing an intuitive, browser-integrated summarization system that operates seamlessly within the user's workflow. Instead of requiring users to rely on external applications or manually transfer content into them, SummarizeME lets users highlight any portion of a webpage and instantly receive a summary. The system incorporates several key features to create a smooth and efficient summarization experience:

- **In-browser summarization of selected text:** Users can highlight any text directly on a webpage and request an immediate summary without leaving the site.
- **Dedicated side panel view for summaries:** Summaries are displayed inside Chrome's side panel, offering a clean, persistent space where users can read summaries while still viewing the original article.
- **Real-time, seamless interaction:** The summarization process occurs instantly upon user request, with no page reloads or external navigation.
- **Support for multiple summary lengths**: Users can generate *short*, *medium*, or *long* summaries depending on the level of detail desired.

A key design consideration of SummarizeME is its ability to handle longer or more complex documents, which often exceed standard model input limitations. To ensure summaries remain both coherent and comprehensive for such content, the system utilizes a hierarchical summarization approach:

- Segmenting long text into smaller, context-preserving chunks suitable for model processing.
- Summarizing each segmented chunk independently to maintain local coherence.
- Synthesizing the segment summaries into a unified final summary, improving global coverage and readability.

Overall, SummarizeME aims to make information consumption more intuitive, efficient, and aligned with natural browsing behavior. Through real-time text selection, a persistent side panel interface, adjustable summary granularity, and hierarchical processing for long documents, the system addresses the growing challenge of navigating and understanding extensive online information.

Technology Stack

SummarizeME is built using a combination of modern web technologies and NLP tools that work together. Each component plays a specific role in enabling real-time text extraction, communicating with the backend, and efficiently generating summaries.

1. Chrome Extension (Frontend)

The frontend interface is implemented as a Chrome Extension using Manifest V3. This allows the system to integrate seamlessly with the user's browser through features such as context menus, side-panel rendering, and active-tab access. The extension handles user interactions, such as selecting text, requesting summaries, and provides an intuitive, in-browser space to display results. Its lightweight design ensures that summarization fits naturally into the user's existing workflow without any extra steps.

2. FastAPI Backend

The backend service is built using FastAPI, a modern Python web framework known for its speed and simplicity. It exposes a summarization endpoint that receives text from the extension, processes it, and returns the generated summary. FastAPI's asynchronous capabilities, built-in validation, and compatibility with machine learning workloads make it well-suited for handling repeated summarization requests efficiently.

3. Summarization Model

At the core of the system is a fine-tuned FLAN-T5 language model. The model is trained using LoRA adapters and parameter-efficient fine-tuning (PEFT) techniques on a SAMSum dataset, enabling it to perform robust summarization across natural language. This method achieves strong summarization performance while maintaining computational efficiency during both training and inference.

Component	Technology
Browser Integration	Chrome Extension (Manifest V3)
Frontend Scripting	JavaScript / HTML / CSS
Background Services	Chrome Service Worker
API Layer	FastAPI (Python)
Machine Learning Model	FLAN-T5 Base Model + LoRA Adapters
Fine-Tuning Method	PEFT Fine Tuning
Model Serving	Transformers (HuggingFace)
Long-Document Handling	Hierarchical Summarization Framework
Datasets	SAMSum

Frontend Implementation

The SummarizeME frontend is implemented as a Chrome Extension built with Manifest V3. Its primary role is to integrate the summarization functionality directly into the browser environment. The extension serves as the user-facing component of the system and manages all interactions between the user and the backend.

At a high level, the frontend consists of three major components:

1. Extension Configuration (manifest.json)

The extension is initialized via the manifest.json file, which declares required permissions, enables context menu integration, and registers the background service worker. Key capabilities such as reading selected text, storing user input, displaying the side panel, and communicating with the backend API are enabled here. This configuration ensures that the extension can interact seamlessly with both the browser and external services.

2. Background Service Worker

The background service worker acts as the control center of the extension. It listens for browser-level events and performs system-wide tasks, such as:

1. Creating the “Summarize” context-menu option
2. Capturing the user’s selected text when the “SummarizeME” menu option is clicked,
3. Opening the Chrome side panel, and
4. Forwarding selection data to the panel for display.

The background service worker operates independently of the webpage itself, enabling consistent behavior across sites and tabs.

3. Chrome Side Panel Interface

The side panel serves as the main user interface for viewing summaries. Once the user initiates a summarization request, the side panel loads and displays:

1. Selected Website
2. Controls for choosing different summary lengths (short/medium/long)
3. The generated summary returned from the backend, along with word count, character count, and timestamp.

The side panel has the following features:

1. **Persistent and non-intrusive interface:** Allows users to view summaries alongside the original webpage, enabling easy comparison between the content and its summarized version.

2. **Automatic theme adaptation:** The panel adjusts to the user’s system appearance settings, supporting both light and dark modes for improved readability and visual consistency.

3. **State retention on close and reopen:** Users can close and reopen the side panel without losing the previously generated summary. This is achieved through Chrome’s storage API, which temporarily stores the last selected text and summary. When the panel is reopened, it automatically retrieves and displays the saved summary, removing the need to regenerate it unless desired.

4. **Backend Communication:** Finally, the panel manages backend communication by sending summarization requests, showing loading indicators during processing, and handling error states when network or backend issues occur.

User Interaction Workflow

At a high level, the frontend workflow operates as follows:

1. User selects text on any webpage.
2. On right click, the “SummarizeME” context-menu action triggers the extension to capture the selected text.
3. The background script opens the side panel and transfers the selected text for processing.
4. The side panel sends a request to the backend with the text and desired summary length.
5. The generated summary is returned and rendered inside the side panel.

Model Implementation

1. Datasets Details

To ensure that the model performs well across multiple content types found on webpages, SummarizeME was fine-tuned using the following dataset:

SAMSum Dataset

a. Link: https://huggingface.co/datasets/samsum
b. Contains human-written summaries of conversational dialog.
c. Improves the model’s ability to summarize informal, chat-like text found in emails, forums, messaging transcripts, and social platforms.
   Dataset Size
d. Training set: 14,732 samples
e. Validation set: 818 samples
f. Test set: 819 samples
g. Total: ~16,369 dialogue-summary pairs

2. Model Details

At the core of the SummarizeME system is the **Google/flan-t5-base** model, a 250M-parameter ecoder-decoder variant of the FLAN-T5 family. The model was selected because:

1. It offers significantly stronger summarization capability than the small variant.
2. It supports instruction-tuned behavior, making it well-suited for summarization tasks using natural-language prompts.
3. Its smaller size is ideal for browser-initiated requests, where latency must remain low.
4. It can be effectively fine-tuned using LoRA (Low-Rank Adaptation) and PEFT (Parameter-Efficient Fine-Tuning) to adapt to summarization tasks without requiring large compute resources.

This balance of performance and efficiency makes flan-t5-base a suitable base foundation for both single-pass and hierarchical summarization pipelines.

3. Training Process

The model was trained using a parameter-efficient fine-tuning approach to adapt FLAN-T5 for both conversational and long-form summarization. The process involved preparing and preprocessing the dataset, applying LoRA adapters, and training the model using supervised sequence-to-sequence learning. The steps are described below.

A. Dataset Preparation and Preprocessing: Before training, the dataset sample was converted into a standardized format suitable for supervised summarization. Preprocessing included:

1. Text normalization which involved cleaning unnecessary characters, fixing spacing, and ensuring consistent formatting across the dataset.

2. Instruction prompting, where each input document was prefixed with a guiding instruction, such as: “Summarize the dialogue:” (for conversational data).

3. Tokenization using the FLAN-T5 tokenizer, which transforms both the inputs and their reference summaries into sequences of subword tokens. This step is necessary to ensure that the model receives text in a standardized format. During tokenization, a maximum input length and a maximum summary length were enforced, with truncation applied when the text exceeded these limits. Padding was added to shorter sequences so that all samples within a batch shared the same length, enabling efficient and consistent training.

B. Parameter-Efficient Fine-Tuning (PEFT)

As transformer-based language models increase in complexity, it becomes increasingly difficult to fine-tune all model parameters, particularly on consumer-grade hardware. Parameter-Efficient Fine-Tuning (PEFT) addresses this issue by updating only a limited selection of parameters while freezing the rest of the model. This drastically reduces the memory requirements, computational costs, and training time. PEFT also addresses the issue of catastrophic forgetting, which occurs when a model fine-tuned for one job (e.g., summarization) loses performance on other tasks it previously handled well (e.g., translation or question answering). By freezing the majority of the pretrained weights, PEFT preserves the underlying model's general linguistic knowledge while still allowing it to learn task-specific behaviors. Several PEFT strategies exist, and the method employed in this project is LoRA (Low-Rank Adaptation), which is a widely used and computationally efficient strategy that is similar to classic fine-tuning but has a far smaller training footprint.

C. Low Rank Adaptation Mechanism

LoRA works by adding a small set of trainable weights into specific parts of the transformer instead of updating the entire model. These added weights act like “lightweight adjustment layers” that sit on top of the original pretrained layers.

During training:

1. The original model weights were kept frozen, so we do not modify or overwrite the knowledge the model already has.
2. Only the small LoRA weights were updated, which require far less memory and compute.

In this project, these LoRA adapters were attached to the query and value projection matrices inside the attention mechanism. These components help the model decide:

1. Which parts of the input are important, and
2. How information should be combined and summarized.

By allowing only these small adapter layers to learn, the model can pick up summarization-specific skills without needing to update millions of underlying parameters. Thus, LoRA teaches the model how to summarize better by adding a small trainable “patch” to the attention system, rather than changing the whole model.

D. Training Configuration

Once the datasets were prepared and LoRA adapters were integrated into the FLAN-T5 model, the fine-tuning process was carried out using the HuggingFace Seq2SeqTrainer framework. The Seq2SeqTrainer automates the core training loop for encoder–decoder models by handling batching, loss computation, gradient updates, evaluation, and checkpointing. It streamlines the entire fine-tuning workflow, allowing the model to learn the mapping from input texts to target summaries with minimal custom training code.

Training Hyperparameters: The following hyperparameters were used during training:

1. Number of training epochs: 2 (Due to limited compute available, the model was trained for 2 epochs; however, generally 3-5 epochs can be used)
2. Seed: 42
3. Learning Rate: 5e-5
4. Maximum source length: 1024 tokens (Maximum tokens in the input)
5. Maximum target length: 256 tokens (Maximum tokens in the output)
6. Optimizer Used: AdamW
7. Evaluation Strategy: At the end of each epoch (“Epoch”)
8. Beam search: 4 (uses four parallel candidate paths during decoding to generate higher-quality summaries)

LoRA Config:

1. r=8: the rank (size) of the low-rank matrices inserted into target layers.
2. lora_alpha=32: scaling factor;
3. lora_dropout=0.1: dropout on the LoRA path to reduce overfitting.
4. bias="none": No added bias.
5. task_type="SEQ_2_SEQ_LM": tells PEFT we're fine-tuning a seq2seq LM.
6. target_modules=["q", "v"]: for T5, these are the projection submodules in attention (query/key/value/output). LoRA adapters are injected there.

E. Evaluation and Model Selection

The performance of the fine-tuned FLAN-T5 Base model was evaluated using quantitative metrics computed on the validation set of the SAMSum dataset. Evaluation was carried out automatically at the end of each epoch.

1. Evaluation Methodology

a. Evaluation occurred once per epoch (eval_strategy="epoch") on the validation dataset.
b. Summaries were generated using beam search with 4 beams, consistent with the model configuration.
c. The best model was selected based on ROUGE-L, which was specified as the primary metric via metric_for_best_model = "eval_rougeL".

2. **Quantitative Metrics**

Evaluation used the standard ROUGE metrics through the HuggingFace evaluate library:

a. ROUGE-1: Measures unigram (single-word) overlap between the generated summary and the reference. It reflects how well the system captures the key content words from the original summary.

b. ROUGE-2: Measures bigram (two-word sequence) overlap, providing insight into how well the model preserves short phrases and local coherence.

c. ROUGE-L: Measures the longest common subsequence (LCS) between the generated and reference summaries, evaluating how well the summary maintains the overall structure and ordering of information.

d. ROUGE-Lsum: A sentence-level variant of ROUGE-L that compares summaries across whole sentences, offering a stronger reflection of overall summary organization and fidelity.

Together, these metrics assess how accurately, fluently, and structurally aligned the generated summaries are relative to the human-written targets.

3. Observed Metrics on Validation Dataset:

The fine-tuned model achieved the following results on the Validation dataset:

1. ROUGE-1: 54.49
2. ROUGE-2: 30.50
3. ROUGE-L: 45.76
4. ROUGE-Lsum: 45.79

4. Observed Metrics on Test Dataset:

1. ROUGE-1: 50.68
2. ROUGE-2: 25.77
3. ROUGE-L: 41.57
4. ROUGE-Lsum: 41.58

Compared to typical T5-Base baselines on SAMSum, the test results fall within a reasonable range. ROUGE-1 and ROUGE-2 are consistent with expected performance, while ROUGE-L is somewhat lower, likely due to limited training time and computing resources. Overall, the model performs respectably for its size and setup, capturing main ideas reliably even if it does not match the strongest fine-tuned baselines.

4. Model Deployment to HuggingFace

After completing training and evaluation, the LoRA adapter weights were merged with the base FLAN-T5-Base model to create a single consolidated checkpoint suitable for inference. This merging step integrates the learned adapter parameters into the underlying model weights, allowing the backend to load the trained summarization model just like any standard HuggingFace model.

To ensure seamless integration with the FastAPI backend, the merged model was then uploaded to the Hugging Face Hub. Hosting the model on Hugging Face offered several advantages:

1. Version control: Each update to the model is automatically preserved.
2. Standardized interface: The backend loads the model using AutoModelForSeq2SeqLM.from_pretrained().
3. Portability: The model can be accessed from any environment without manual file transfers.
4. Reusability: Future developers can easily access, reuse, or extend the model.

The upload process involves logging into Hugging Face from the training environment, creating a model repository (public or private), and pushing the merged model directory to the Hub. Once

uploaded, the backend retrieves the model directly from HuggingFace during initialization using the HF access token.

Backend Implementation

The SummarizeME backend is implemented using FastAPI. It is responsible for loading the deployed FLAN-T5-Base summarization model from HuggingFace, exposing a lightweight HTTP API for the Chrome Extension, and executing the inference pipeline for different summary lengths. Unlike traditional summarization services that return multiple metadata fields, the SummarizeME backend is intentionally minimal; it returns only the generated summary, while the Chrome Extension UI handles generating all additional metadata, such as timestamps, word count, and character count.

1. API Design

The backend exposes a single primary endpoint: *POST /summarize*

This endpoint accepts a JSON request body containing:

1. Text: the selected webpage content to summarize
2. summary_level: one of “short”, “medium”, or “long.”

Upon receiving a request, FastAPI validates the inputs and routes them to the summarization engine, and returns the JSON response with the status and the summarized content.

2. Model Loading

On startup, the FastAPI backend loads the merged FLAN-T5-Base model (integrating LoRA adapter weights) from HuggingFace. The tokenizer and model weights are cached locally, ensuring fast warm-up and low-latency inference. The backend is designed to run effectively on both CPU and GPU environments, allowing flexible deployment.

3. Inference Pipeline

The model inference pipeline is designed to efficiently generate summaries of varying lengths while maintaining high quality across both short and long inputs. When a summarization request is received, the backend first computes the true token length of the input text (without truncation). This length determines whether the system performs single-pass summarization or switches to a hierarchical approach. A cutoff of 1024 tokens is used because FLAN-T5-Base supports a maximum input size of 1024 tokens. This threshold also improves stability in the model’s attention patterns and keeps inference latency low enough for real-time browser interaction. For inputs within this token limit, the system uses single-pass summarization. The

input is paired with a fixed instruction prompt, and the model generates a summary using beam search with parameters tuned to control output length. The configurations are:

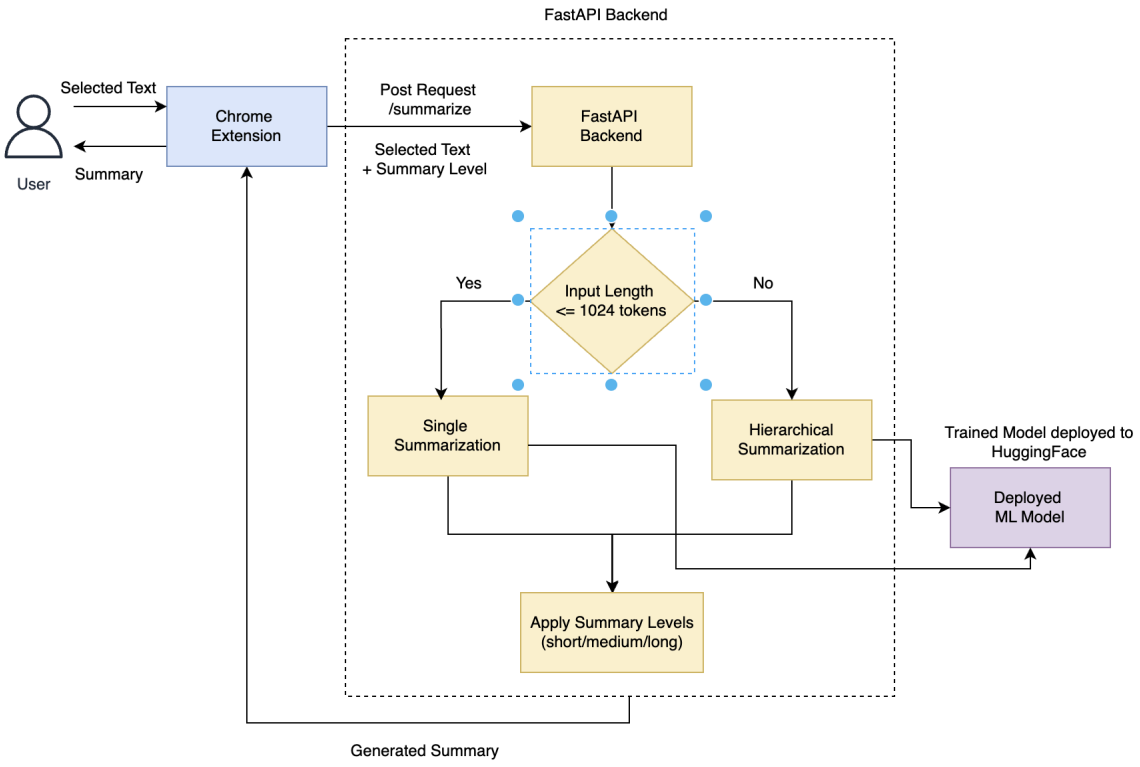
1. Short: 80–140 new tokens, length penalty 1.2, 3 beams
2. Medium: 140–260 new tokens, length penalty 1.0, 3 beams
3. Long: 220–380 new tokens, length penalty 0.9, 4 beams

These settings provide predictable summary lengths and ensure consistent coverage across different content types.

For longer inputs, the system applies hierarchical summarization. The text is first broken into sentence-aligned segments, using a sentence boundary detector to ensure that chunks do not split mid-sentence. These sentences are then aggregated into chunks of approximately 2500 characters, a size chosen to balance local context preservation with efficient model processing. Each chunk is summarized independently using the medium-length configuration, producing detailed yet concise intermediate summaries. These partial summaries are then concatenated and passed through a final summarization step using the user-selected length configuration (short, medium, or long), resulting in a summary that maintains the desired level of abstraction.

System Workflow

The figure below summarizes the workflow for generating summaries.



Limitations

Although SummarizeME performs reliably across simple everyday web content, several practical limitations arise from the design choices and computational constraints of the current system. These limitations do not detract from the functionality of the implementation but instead highlight areas where future enhancements could further strengthen performance.

1. Lack of Citation or Source Attribution

The FLAN-T5-Base model does not provide source-aware citations or highlight which portions of the input text support specific statements in the summary. As a result, the system cannot generate reference-style summaries or attribute information back to exact sentences. While this is typical of most lightweight abstractive models, adding citation capability would make the system more suitable for academic or research-heavy workflows.

2. Occasional Hallucination Due to Model Size

Because the backend operates under compute constraints, the system uses FLAN-T5-Base rather than larger variants such as FLAN-T5-XL/XXL or GPT-class models. The base model performs well for simple summarization, but like most compact models, it may occasionally generalize or simplify details too aggressively, especially in highly technical domains. Larger models generally offer stronger factual grounding, but the current choice provides a good balance between accuracy, responsiveness, and deployment feasibility.

3. Selection-Dependent Context Awareness

The current summarization system is not section-aware; it does not recognize or utilize the structural organization of a webpage, such as headings, subsections, or document hierarchy, when generating the summaries. As a result, summaries reflect only the raw text provided rather than the broader section-based context.

4. Trade-offs in Hierarchical Summarization for Long Inputs

The hierarchical pipeline enables summarization of long documents well beyond the model's single-pass token limit. However, splitting text into sentence-aligned chunks and summarizing them independently can sometimes reduce cross-segment coherence or continuity. The final synthesis step mitigates this to a large extent, but subtle variations in tone or emphasis may occur. These trade-offs are expected with hierarchical methods and represent a practical compromise that allows efficient handling of lengthy text.

Conclusion

SummarizeME demonstrates that effective, real-time text summarization can be seamlessly integrated into a user's browsing experience through a lightweight Chrome Extension paired with an efficient FastAPI backend. By combining a fine-tuned FLAN-T5-Base model with parameter-efficient techniques and a hierarchical inference pipeline, the system is able to

generate coherent summaries across a wide range of input lengths while maintaining low latency suitable for interactive use. The design choices, such as handling summarization directly within the browser workflow, supporting multiple summary lengths, and enabling long-document processing through sentence-aligned chunking, reflect a careful balance between capability, responsiveness, and computational practicality.

While the system exhibits some limitations inherent to compact language models and the absence of structural or citation-aware summarization, the current implementation provides a strong and reliable foundation for simple summarization tasks. It successfully meets its core objective – allowing users to quickly understand complex or lengthy web content without leaving the page or relying on external tools.

Overall, SummarizeME highlights the potential for scalable, model-driven browser extensions and provides a solid platform for future enhancements, including stronger factual grounding, richer summary formats, and more advanced model architectures.

Resources

1. Github Link - https://github.com/shahakanksha-tamu/SummarizeME
2. Deployed Model - https://huggingface.co/akankshashah/flan-t5-base-samsum-merged

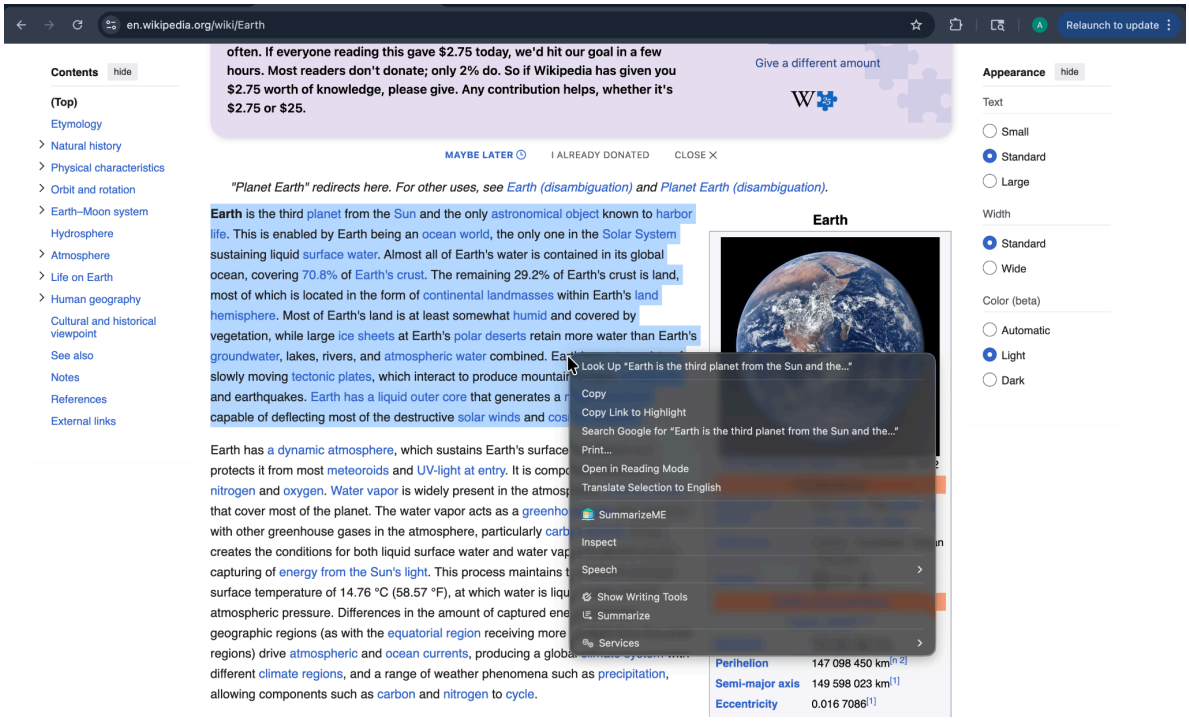
Because the FLAN-T5-Base model is relatively large, free hosting platforms such as Vercel or Railway do not provide enough memory to load and serve the model during inference, as these platforms are optimized for lightweight, stateless functions. As a result, the backend was executed locally rather than deployed to a cloud environment; full deployment would require a paid, higher-memory, or GPU-enabled service.

Similarly, the frontend is implemented as a Chrome Extension, which is not deployed on a traditional web hosting platform. Instead, it is packaged and loaded directly into the browser, since Chrome Extensions run locally within the browser environment rather than being hosted on a server.

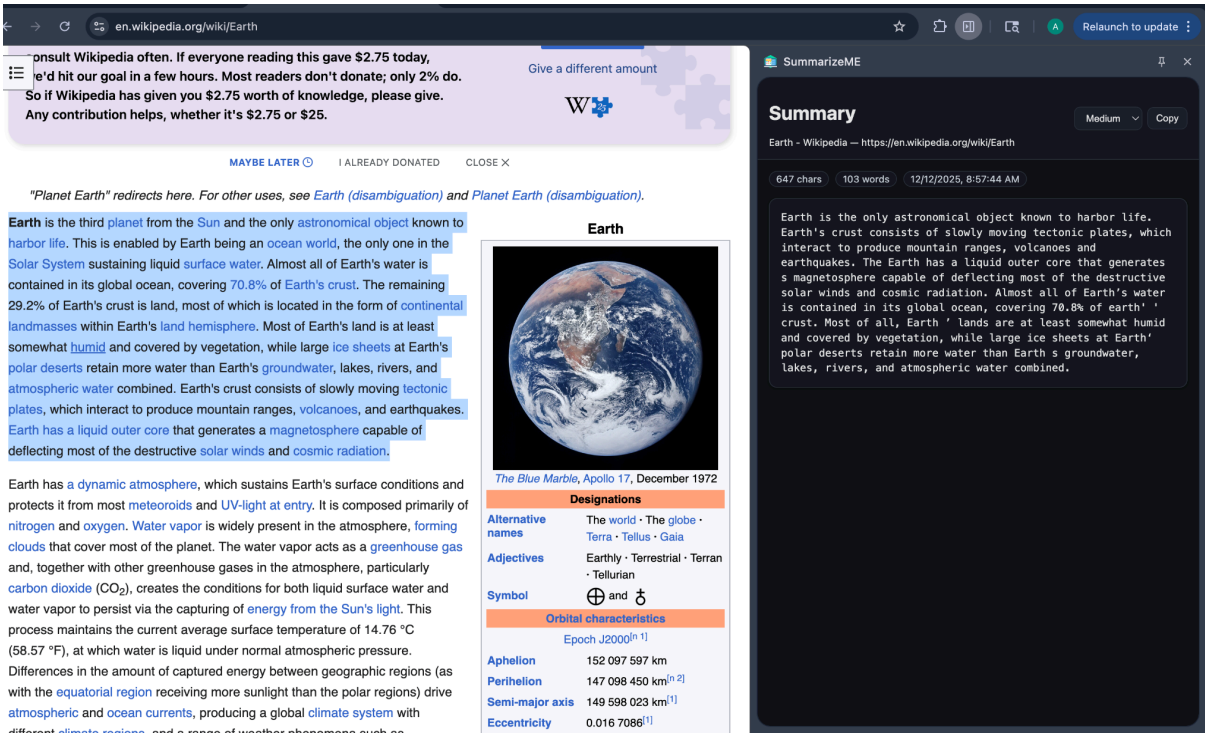
Screenshots and Demonstration

Video - SummarizeME Recording Link

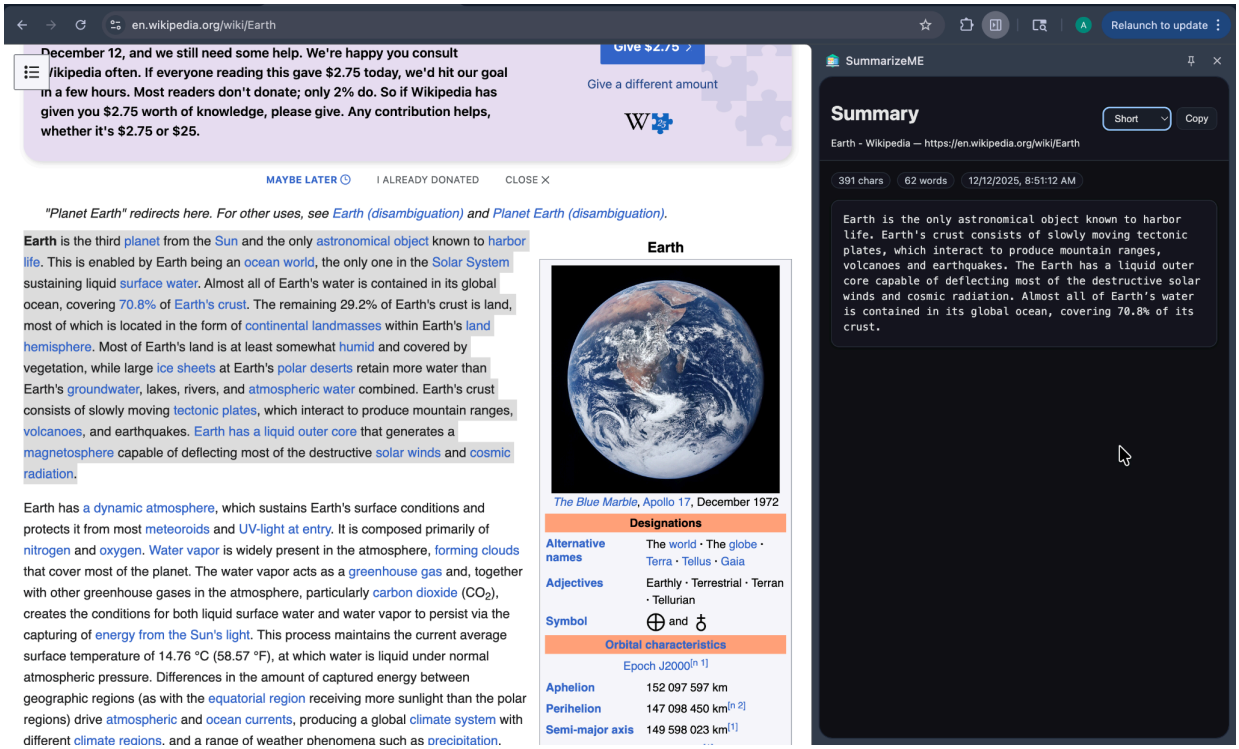
1. Chrome Extension Context Menu on Content Selection - Ability to view “SummarizeME” content menu option on right click on the selected text



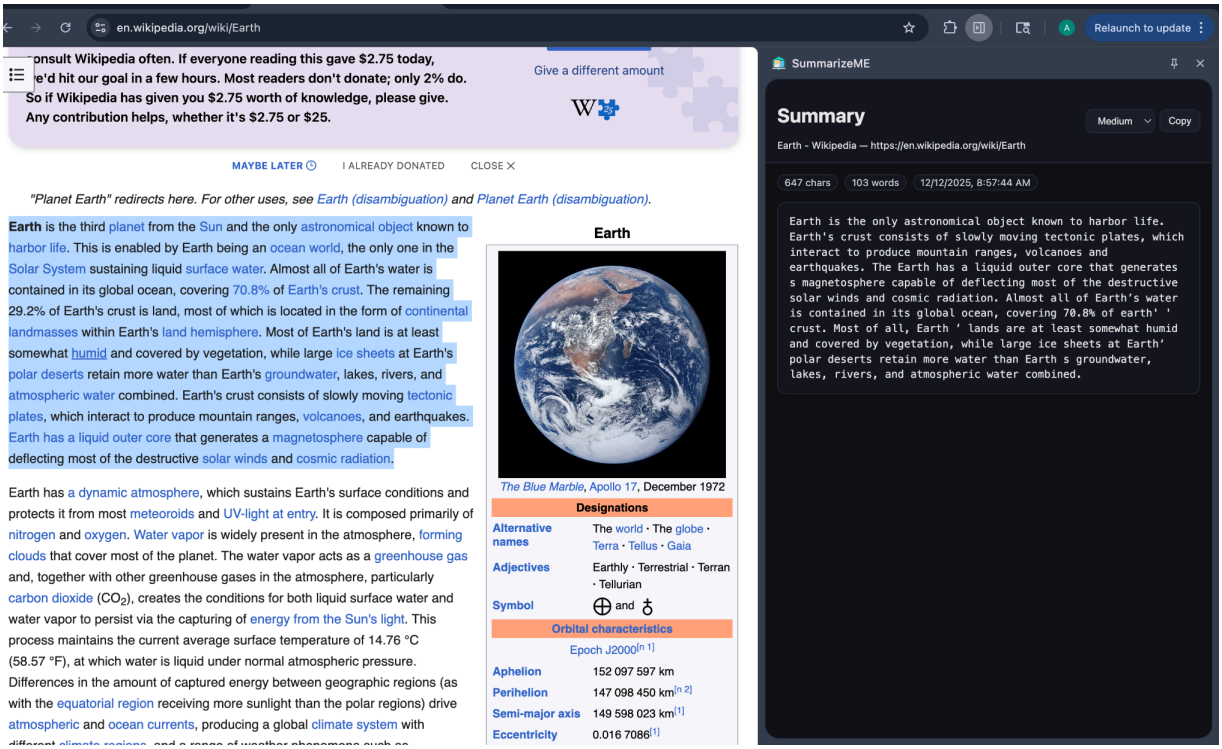
2. Chrome Extension Side Panel - Ability to view summary for the selected content in a side panel. Displays the character count, word count, timestamp and the website URL.



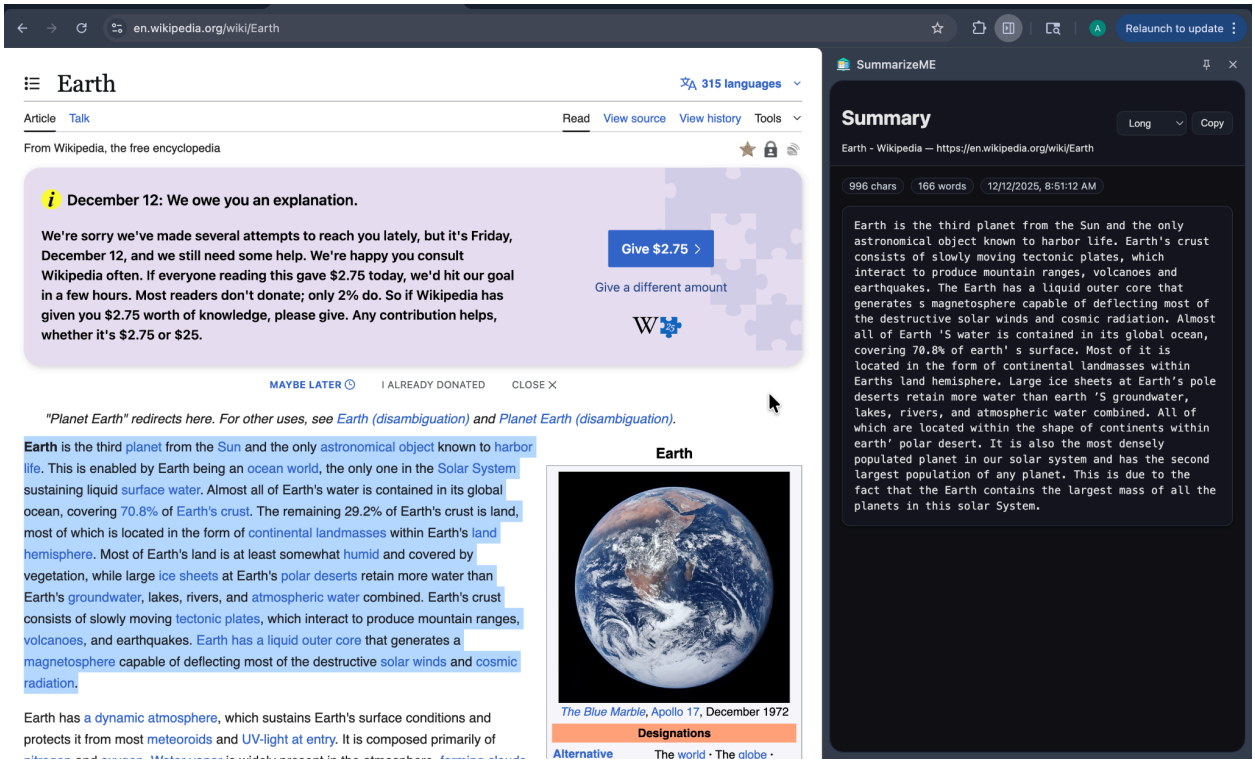
3. Short Summary



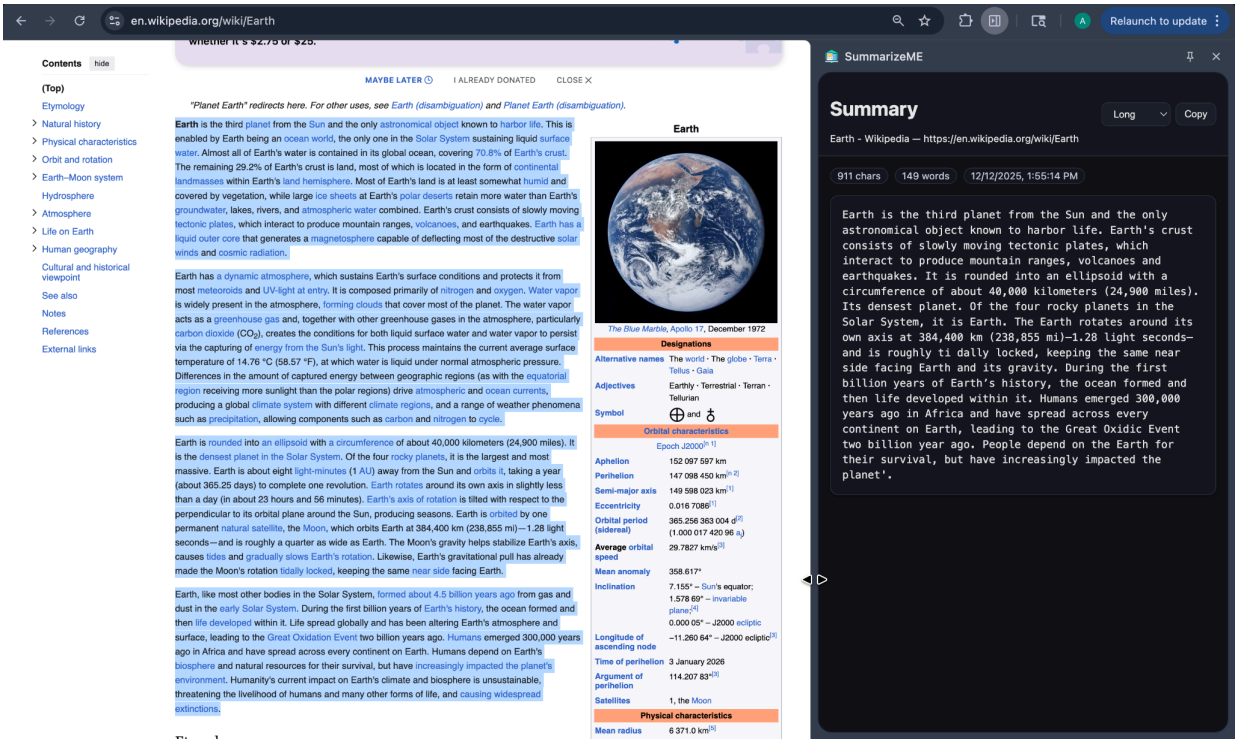
4. Medium Summary



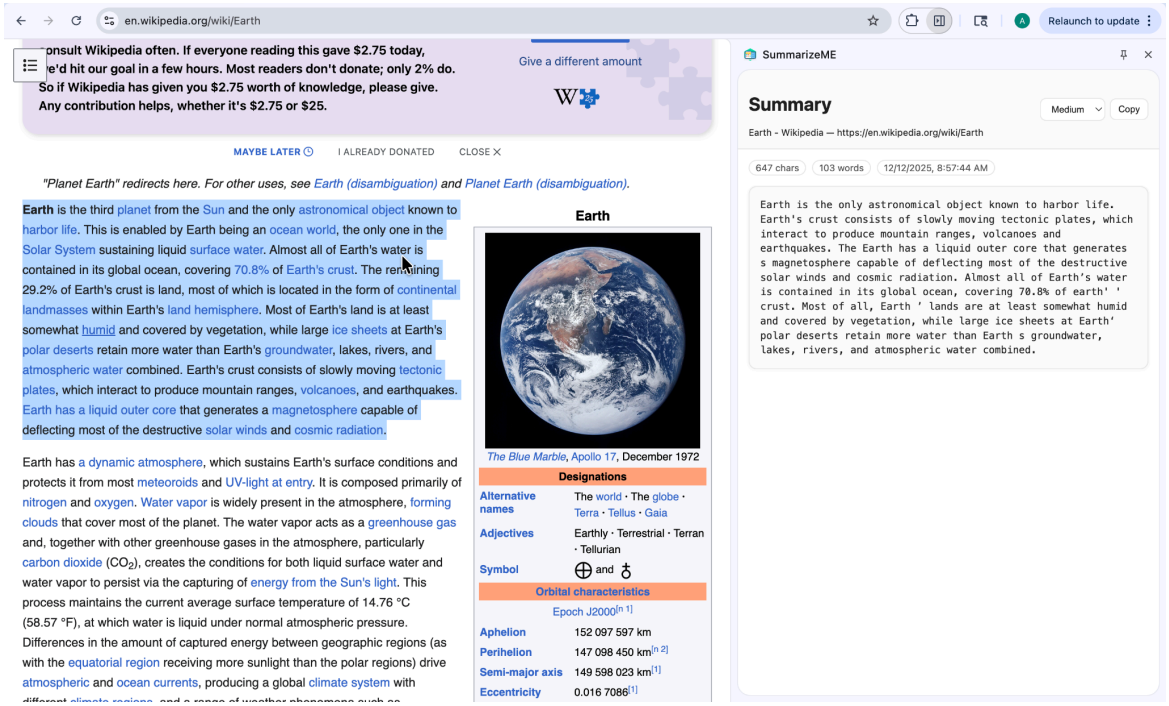
5. Long Summary



6. Hierarchical Summarization - Ability to Summarizes long content



7. Chrome Extension Dark/Light Theme Support - The side panel UI adapts to the user “Appearance” setting



8. Copy the generated Summary Content - The side panel provide as option to copy the summarized content