ChatGPT

Project 3 – Part 1: Problem, Foundations & Tools

1. Introduction and Problem Statement

Modern news consumers face an overwhelming stream of information. Long-form articles provide depth but are time-consuming to read; many readers skim headlines and miss nuances. Editors and researchers need to quickly extract key points while ensuring important details are not lost. Your mission is to build an **automated text summarisation and deep-research system** that turns long news articles or multiple documents into concise, coherent bullet-point summaries tailored to human preferences. The system should:

- **Condense long articles** into 2–5 bullet points while retaining critical facts.
- **Optimise relevance, readability and factuality** based on user feedback, not just algorithmic metrics. Reinforcement learning from human feedback (RLHF) is one way to achieve this [1].
- **Support multi-document summarisation and open-domain research**, retrieving external facts via a retrieval-augmented generation (RAG) pipeline [2].
- **Scale to production** with robust deployment, monitoring and continuous improvement pipelines.

With this project you will gain hands-on experience with large language models (LLMs), vector search, human-in-the-loop training and DevOps best practices—all broken down into manageable steps for beginners.

2. Dataset & Preprocessing

For supervised training you will use the **CNN/Daily Mail** dataset available through Hugging Face Datasets. It contains thousands of news articles paired with human-written summaries. To prepare the data:

1. **Load the dataset** using `datasets.load_dataset("cnn_dailymail", "3.0.0")`.
2. **Select the** `article` **and** `highlights` **fields**, which correspond to the original article and its summary.
3. **Clean the text** by stripping HTML tags, extraneous whitespace and escaping HTML entities.
4. **Split into train/validation/test sets** (e.g. 90/5/5 %) and shuffle.
5. **Tokenise** using the tokenizer of your chosen model (e.g. `AutoTokenizer.from_pretrained("t5-small")`). Limit articles to about 512 tokens and summaries to about 150 tokens; apply truncation and padding.

Preprocessing with the Hugging Face Datasets API returns each example as a dictionary with token IDs and attention masks. Save processed datasets to disk using `dataset.save_to_disk()` for reuse during training.

3. Baseline Models & Parameter-Efficient Fine-Tuning

3.1 Baseline Model (T5)

You will begin with **T5 (Text-to-Text Transfer Transformer)**, a versatile sequence-to-sequence model for converting text to text. It excels at abstractive summarisation because it learns to map input sequences (articles) to output sequences (summaries). Steps:

1. **Load a pre-trained checkpoint** (e.g. `t5-small`, `t5-base` or `t5-large`). Smaller checkpoints train faster; larger ones yield higher quality if you have GPU resources.
2. **Prepare a summarisation prompt**: T5 expects tasks expressed as text. For example: `"summarize: {article text}"` as the model input.
3. **Fine-tune using supervised learning** with a cross-entropy loss on the reference summaries. Use the `Trainer` API from `transformers` or the `trl` library to manage training loops. Monitor training and validation loss; adjust learning rate, batch size and number of epochs accordingly.
4. **Evaluate** using ROUGE scores (ROUGE-1/2/L) and manually read a few summaries to assess readability.

3.2 Parameter-Efficient Fine-Tuning (PEFT)

Training large models from scratch is computationally expensive. **LoRA (Low-Rank Adaptation)** and **QLoRA** are PEFT methods that insert small, trainable low-rank matrices into a model's attention layers, dramatically reducing the number of trainable parameters. The PEFT library provides wrappers for transformers. You can apply LoRA to:

- **T5** – adapt a larger checkpoint (`t5-base`, `t5-large`) with limited compute.
- **BART** – an alternative encoder-decoder model that performs well on summarisation.
- **Mistral-7B-Instruct** and **Mixtral-8x7B** – modern open-source LLMs that use grouped-query and sliding window attention. Mistral 7B achieves strong benchmark results and is released under the Apache 2.0 licence [3].

By training LoRA adapters instead of all weights, you can fine-tune these models on a single GPU. During inference the LoRA adapter can be merged into the base model or kept separate.

3.3 Reinforcement Learning with Human Feedback (RLHF)

Once your baseline model generates reasonable summaries, you can further align it to human preferences using **RLHF**. In RLHF you:

1. **Collect human preference data** – sample multiple summaries for a given article and ask annotators to rank them. This yields pairs such as `(preferred summary, less preferred summary)` [1].
2. **Train a reward model** – fit a separate model to predict these preferences [4].
3. **Optimise the policy with RL** – fine-tune the summariser using an algorithm like Proximal Policy Optimisation (PPO) so that it maximises the reward predicted by the reward model [4]. Tools like TRLX simplify this process.

RLHF helps produce summaries that readers prefer, but obtaining high-quality human feedback is costly and can introduce bias [1]. Alternative approaches like **Direct Preference Optimisation (DPO)** and

Identity Preference Optimisation (IPO) bypass the reward model and treat alignment as a supervised problem [5]. In this project you'll start with RLHF but can explore DPO later.

3.4 Retrieval-Augmented Generation (RAG)

LLMs have a fixed knowledge cutoff and may hallucinate or omit important facts. **RAG** enhances accuracy by retrieving relevant documents from an external knowledge base and providing them to the model at generation time. NVIDIA's blog notes that RAG improves reliability by supplementing the model with information fetched from specific data sources [2]. It allows the model to cite sources and reduces hallucinations [6]. A typical RAG pipeline:

1. **Ingest documents** (e.g. news articles, papers) and split them into chunks.
2. **Encode chunks into vectors** using an embedding model.
3. **Store vectors** in a vector database such as `pgvector` within PostgreSQL [7].
4. **Retrieve relevant chunks** for a query via similarity search [7].
5. **Generate a summary or answer** by providing the retrieved context and the query to the LLM, with prompts that enforce citation and factuality.

Your project will implement this pipeline using `pgvector` as the vector store and incorporate retrieval both for summarisation and the Deep Research feature.

3.5 Multi-Document Summarisation

Real-world tasks often involve summarising multiple sources on the same topic. To avoid exceeding context limits, you can use two strategies [8] [9]:

1. **Map–Reduce:** summarise each document (or chunk) individually, then combine the summaries into a single global summary. This parallelises the first phase but may lose context when merging.
2. **Refine:** process documents sequentially; start with a summary of the first document and iteratively refine it as you incorporate each new document. This preserves context but is sequential.

You'll implement these strategies in later sprints to support multi-document summarisation and the Deep Research mode.

4. Tools & Technology Overview

Category	Tools & Libraries	Why They're Used
Data	Hugging Face Datasets, pandas	Loading and cleaning the CNN/Daily Mail dataset.
Modelling	Transformers, PEFT (LoRA/QLoRA), TRL/TRLX	Fine-tuning T5/BART/Mistral models; implementing RLHF.
Evaluation	Hugging Face Evaluate, ROUGE, BERTScore, QAFactEval	Compute summary quality and factuality metrics.

Category	Tools & Libraries	Why They're Used
Inference	vLLM	High-throughput inference server implementing the OpenAI API [10].
Backend API	FastAPI	Build async REST endpoints (`/summarize`, `/summarize_multi`, `/research`, `/rag/ingest`, etc.).
Vector Store	PostgreSQL + pgvector	Store embeddings and perform similarity search [7].
Front-End	Vite + React, Netlify	Provide a user interface for summarisation and deep research; Netlify offers branch deploys and deploy previews [11] [12].
Containerisation	Docker, Kubernetes, Helm	Package and deploy services across multiple environments.
CI/CD	GitHub Actions	Automate testing, build and deployment.
Observability	Prometheus, Grafana	Collect and visualise metrics; set alerts [13] [14].
Security & Secrets	External Secrets Operator, Kubernetes Secrets	Manage credentials and API keys.

Tool Summary

- **Hugging Face Datasets & pandas:** used for loading, cleaning and pre-processing the CNN/Daily Mail dataset. They provide convenient APIs for streaming large datasets and manipulating tabular data.
- **Transformers & PEFT:** the `transformers` library offers pre-trained models like T5, BART, Pegasus and Mistral, while PEFT (LoRA/QLoRA) allows you to fine-tune these models efficiently by training only a small subset of weights.
- **TRL/TRLX:** these libraries simplify RLHF and DPO training loops. You define a reward function (e.g. based on human preferences or ROUGE) and they implement algorithms like Proximal Policy Optimisation.
- **vLLM:** an open-source inference server that exposes the same API as OpenAI's Chat/Completion endpoints [10]. It uses efficient scheduling to serve many concurrent requests and supports streaming responses.
- **FastAPI:** a modern Python web framework for building asynchronous REST APIs. It provides automatic OpenAPI documentation, dependency injection and easy integration with pydantic for request/response validation.
- **PostgreSQL + pgvector:** PostgreSQL is a reliable relational database. The `pgvector` extension adds a `vector` data type and similarity operators so you can store embeddings and perform nearest-neighbour search within the database [7].
- **Prometheus & Grafana:** Prometheus scrapes metrics from your API, model server and database; Grafana visualises them and allows you to define alerts [13] [14]. Together they form the observability stack.

- **Docker, Kubernetes & Helm:** Docker packages your application into portable containers; Kubernetes orchestrates deployment and scaling across multiple nodes; Helm templates Kubernetes manifests and manages releases across environments.
- **Netlify & React:** React provides a reactive front-end; Netlify hosts your static site and automatically builds deploy previews and branch deploys for each pull request [11] [12].
- **External Secrets Operator / Kubernetes Secrets:** used to inject sensitive information (database passwords, API keys) into your pods without committing them to version control.

5. Conceptual Foundations: Hallucination & Quality Control

LLMs sometimes produce plausible but false statements—**hallucinations**. You'll mitigate hallucinations by:

1. **Using RAG & citations** – providing retrieved context and requiring the model to cite sources [6].
2. **Reward penalties** – including a term in the RLHF reward that penalises unsupported claims.
3. **Factuality evaluation** – running QAFactEval or NLI-based checks to detect inconsistencies.
4. **Monitoring** – tracking hallucination errors in Prometheus/Grafana and setting alerts.
5. **Prompt design** – instructing the model to answer concisely and avoid speculation.

Understanding these foundations prepares you to build a trustworthy summariser. Continue to Part 2 for architectural design and the RAG pipeline.

6. Implementation Workflow Summary

For beginners it helps to see the *whole journey* at a glance. A typical workflow for this project is:

1. **Initial setup:** install Python packages and set up Git & GitHub. On GitHub create a new repository (`New repository` → give it a name, description, choose public/private and add a README) [15]. Create a feature branch from `main` (`Code` tab → drop-down menu → enter branch name and click *Create branch*) [16]. Make your first commit by editing the README, writing a commit message and clicking *Commit changes* [17]. Open a pull request from your branch to `main` and merge it after review [18]. Install Docker on your machine (use the official installer for your OS) and verify the installation by running `docker run hello-world`; you should see a "Hello from Docker" message if it worked [19].
2. **Data preparation:** download the CNN/Daily Mail dataset, clean and tokenise the articles and highlights (Section 2).
3. **Baseline summariser:** fine-tune a T5 model on the dataset using supervised learning (Section 3.1). Evaluate with ROUGE and BERTScore.
4. **PEFT & alternative models:** apply LoRA/QLoRA to larger models (BART, Pegasus, Mistral 7B, Mixtral 8×7B) to improve quality with limited compute (Section 3.2). Record training cost and quality improvements. Consider running experiments with each model to see which suits your constraints [3].
5. **RLHF:** collect human preferences, train a reward model and optimise the policy with PPO or DPO (Section 3.3). RLHF aligns the model with human preferences but requires careful reward design and feedback [1] [4].
6. **RAG pipeline:** build a retrieval layer using embeddings stored in a `pgvector` table and integrate it into the summarisation loop to provide relevant context [7] [2].

7. **Multi-document & Deep Research:** implement map–reduce and refine summarisation chains to summarise multiple documents [8] [9]. Build a Deep Research feature that searches the web, ingests results and synthesises bullet-point answers with citations [6].
8. **Evaluation & hallucination control:** measure ROUGE, BERTScore and factuality; run QAFactEval and track hallucination rates. Use retrieval and citation prompts to reduce hallucinations.
9. **API & Front-End:** wrap your model in a FastAPI server with endpoints (`/summarize`, `/summarize_multi`, `/research`, `/rag/ingest`, `/rag/query`, `/feedback`). Create a React UI hosted on Netlify; Netlify automatically builds deploy previews for pull requests [11].
10. **Observability & CI/CD:** instrument the API with Prometheus metrics and visualise them in Grafana [13] [14]. Use Helm charts and GitHub Actions to deploy to dev/test/staging/prod and to run nightly training and evaluation.

This high-level plan shows how the different pieces fit together and guides you as you progress through the sprints in Part 3.

7. Learning Path for Beginners

If you're new to LLMs and DevOps, follow this learning sequence before or alongside the sprints:

1. **Python & Git basics:** practise variables, functions and classes; learn how to create a repository, branch, commit and open a pull request on GitHub [15] [16]. Familiarise yourself with command-line git (clone, add, commit, push) and remote collaboration.
2. **Transformers & summarisation:** read the original *Attention is All You Need* paper and explore pre-trained models on Hugging Face. Understand the difference between extractive and abstractive summarisation. Fine-tune a small model on a toy dataset.
3. **PEFT & LoRA:** study parameter-efficient fine-tuning (LoRA/QLoRA) and why it reduces compute requirements. The PEFT library documentation and blog posts are good starting points.
4. **Reinforcement learning & RLHF:** learn about Markov Decision Processes, rewards and policy gradients. Read Ziegler et al.'s paper *Fine-Tuning Language Models from Human Preferences* and experiment with TRLX examples.
5. **Vector search & RAG:** explore sentence embeddings, vector databases and similarity search. Read about `pgvector` and how it extends PostgreSQL with a `vector` type and distance operators [7]. Understand why RAG reduces hallucinations and improves factuality [2].
6. **Multi-document summarisation:** learn map–reduce and refine strategies [8] [9]. Read research papers on hierarchical summarisation and try implementing simple versions.
7. **Backend development:** study FastAPI for building async REST APIs; practise writing endpoints and integrating with a database.
8. **Frontend development:** learn React basics (components, hooks, state management); experiment with Netlify for continuous deployment. Understand how Netlify deploy previews and branch deploys work [11] [12].
9. **Containerisation & Kubernetes:** install Docker and practise running containers (`docker run hello-world`, `docker ps`) [19]. Learn the basics of Kubernetes (pods, services, deployments) and Helm templating.
10. **Observability & CI/CD:** read about Prometheus (pull-based metrics collection and alerting) [13] and Grafana dashboards [14]. Familiarise yourself with GitHub Actions and how to define workflows for testing and deployment.

Use this learning path as a checklist. As you progress through the project, revisit these topics to reinforce your understanding.

---

1 4 5 Reinforcement learning from human feedback - Wikipedia
https://en.wikipedia.org/wiki/Reinforcement_learning_from_human_feedback

2 6 What Is Retrieval-Augmented Generation aka RAG | NVIDIA Blogs
https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/

3 Mistral 7B | Mistral AI
https://mistral.ai/news/announcing-mistral-7b

7 The pgvector extension - Neon Docs
https://neon.com/docs/extensions/pgvector

8 9 Document Summarization Solution Patterns using Azure Open AI & Langchain - ISE Developer Blog
https://devblogs.microsoft.com/ise/solution-patterns-for-document-summarization-azureopenai/

10 OpenAI Compatible Server — vLLM
https://nm-vllm.readthedocs.io/en/0.5.0/serving/openai_compatible_server.html

11 Deploy Previews | Netlify Docs
https://docs.netlify.com/deploy/deploy-types/deploy-previews/

12 Branch deploys | Netlify Docs
https://docs.netlify.com/deploy/deploy-types/branch-deploys/

13 What is Prometheus? | New Relic
https://newrelic.com/blog/best-practices/what-is-prometheus

14 What is Grafana?
https://www.statsig.com/perspectives/what-is-grafana

15 16 17 18 Hello World - GitHub Docs
https://docs.github.com/en/get-started/start-your-journey/hello-world

19 A Docker Tutorial for Beginners
https://docker-curriculum.com/