Dialogue Summarization with Large Language Models: ACME Project

Pitch by Alejandro Silva [alejandroiq@gmail.com](mailto:alejandroiq@gmail.com)

## Problem Description - Overview

Modern messaging platforms face a critical issue of information overload. In active group chats, users often return to dozens or even hundreds of unread messages. Studies of user behavior suggest:

* Users can spend **10–15 minutes** **average** catching up on just one day’s worth of missed group chat content *(Source: Deloitte Digital Media Trends, 2023).*
* Roughly **40–50% of users report missing important details** when returning to ongoing conversations *(Source: Pew Research Center, Online Communication Study, 2022).*
* Engagement drops significantly when users feel overwhelmed; churn risk increases as users disengage from active channels. In fact, platforms have observed up to a 15–20% decrease in daily active engagement among overwhelmed users, with churn risk rising by approximately 12% in this segment *(Source: McKinsey Consumer Insights, Digital Engagement Report, 2022)*.

This creates a **pain point** for Acme Communications: essential context gets buried in lengthy dialogue, reducing user satisfaction and threatening retention.

## Impact Assessment

The consequences of unchecked information overload are significant:

* **User Experience & Satisfaction**: Frustration arises when users cannot quickly locate important details. This diminishes perceived value of the platform.
* **User Retention & Engagement**: Overwhelmed users are less likely to engage actively, leading to fewer interactions and declining daily active user metrics.
* **Competitive Disadvantage**: Rival platforms that offer AI-powered summarization or enhanced information management gain an advantage in attracting and retaining users.

## Business Question

*Is it possible to leverage automated dialogue summarization while ensuring that summaries remain accurate, concise, and aligned with user expectations?*

## Justification for ML

The choice of the **SAMSum dataset** provides a strong foundation for this proof-of-concept. SAMSum mirrors real-world group chat dynamics with concise, informal dialogue and accompanying human-written summaries. This ensures that the model is trained on data highly relevant to ACME’s target use case, bridging the gap between technical feasibility and user-centered needs. By leveraging this dataset, the project can demonstrate not only summarization accuracy but also practical business value.

In research on human-computer interaction, response delays under **1 second** maintain seamless user engagement, while delays longer than **10 seconds** typically cause users to lose focus or abandon the task *(Source: UX User Experience, how long should the user wait before providing reassurance? 2018)*.. For our summarization tool, this means our **efficiency goal** should be generating summaries in under **1 second** — with fallback performance under **10 seconds** if providing loading indicators or progress feedback

## Dataset Description

The **SAMSum dataset** contains over 16,000 messenger-like conversations paired with human-written summaries. Conversations are short, informal, and multi-turn, reflecting the dynamics of real-world group chats. Each summary captures the essential points of the dialogue, making the dataset particularly useful for training and evaluating dialogue summarization models. Its size, diversity of topics, and naturalistic style make it an industry-standard benchmark for abstractive summarization tasks.

The dataset contains only three columns: the **sample ID**, the **dialogue**, and its corresponding **summary**. Below are two histograms showing the distribution of word counts for dialogues and their summaries.

A graph of a distribution of words

AI-generated content may be incorrect.

A graph of a number of words

AI-generated content may be incorrect.

Dialogues are often very long and highly variable, while summaries are consistently short and concise. This sharp contrast highlights the core business problem: without automation, users must sift through lengthy, noisy conversations to extract key information. Effective summarization directly addresses this pain point by saving time, improving clarity, and enabling faster decision-making.

## Solution Vision

We propose developing an **automated dialogue summarization system** using **large language models (LLMs)**. This feature will:

* **Reduce cognitive load on users** by condensing conversations into concise, accurate summaries.
* **Improve accessibility** by allowing users to catch up on conversations in seconds rather than minutes.
* **Enhance value proposition** by positioning ACME as an innovator in AI-powered communication tools.
* **Enable premium features** such as smart conversation highlights and personalized daily digests.

While the solution is designed to deliver strong performance, it is important to acknowledge **resource limitations**. Training medium-large language models can be computationally expensive, and if a single optimization step requires **30+ minutes of training time**, the cost in time and resources outweighs the marginal gains. As a result, while there may still be room for further improvement, practical constraints require stopping at a point where the model already meets success criteria. This is a **trade-off**: optimization beyond this point yields diminishing returns for the business case.

To address this, I am investing in **GPU capacity in Colab**, since relying on CPU is entirely impractical given the extreme slowness of the process.

## Success Criteria

To measure success, we set clear **technical** and **business** goals:

* **Summarization Quality**: Achieve **ROUGE-1/ROUGE-2 ≥ 0.35–0.40** against human-written benchmarks on the SAMSum dataset.
* **Efficiency**: Generate summaries in short time, ensuring seamless user experience. Additionally, target a compression ratio (summary length ÷ dialogue length) of approximately 20%–30%.
* **Human Judgment (qualitative):** Select 2–3 dialogues from the validation set, compare the generated summaries against both the input dialogue and the reference summary. We could also manually judge clarity, conciseness, and faithfulness. Success is defined as the generated summary being *understandable and aligned with the original conversation* in most test cases.

**Measurement Plan**: Throughout development, we will track ROUGE on the validation set after major changes in parameters to guide model selection. Efficiency will be measured only on the finalized, optimized model to reflect realistic usage. For human judgment, we will do a single side-by-side example early on to compare candidate models, and a final 2–3 example reviews on the best model to confirm clarity, conciseness, and faithfulness before sign-off.

## Stakeholders

**Acme Communications Product Team** will be the primary stakeholder. They are responsible for defining requirements and evaluating whether the summarization feature aligns with user needs and strategic goals.

Other stakeholders’ worth mentioning are Engineering & Data Science Teams, UX/Design Team and End Users.

## Problem Solving Process – Process Framework

### Data Exploration and Preparation:

* 1. **Load the SAMSum dataset and explore its structure.**

1. Import and dataset loading: The SAMSum dataset was imported from Hugging Face (knkarthick/samsum)
2. Shape for train (14,732), validation (818), and test (819) subsets. This provided the foundation for all subsequent exploratory analysis and preprocessing.
3. Shape for train (90%), validation (5%), and test (5%) subsets. This provided the foundation for all subsequent exploratory analysis and preprocessing.
   1. **Analyze the characteristics of the dialogues and summaries.**
4. Data quality: A single missing dialogue was identified in the training set and removed. No missing values were found in the validation or test splits. This ensures data integrity across all partitions.
5. Describe the data: (count, mean, percentiles, std, min, max) and show first 5 rows of the train split as a DataFrame (head)
6. Turns distribution: Conversations typically involve multiple speakers turns, with an average of ~10–15 words per turn. This indicates that the dataset captures genuine back-and-forth interactions rather than monologues. Human-written summaries are much shorter than the dialogues, with an average compression ratio of ~0.15–0.20. This confirms that the summarization task requires distilling long input into concise outputs.
7. Length variability and Edge cases: Dialogues range from extremely short (just a few words) to several hundred words. This reflects real conversational diversity, from quick exchanges (“pizza or pasta?”) to long multi-turn discussions. The dataset includes both minimal dialogues (3 words) and very long ones (hundreds of words), which justifies setting high maximum input lengths during preprocessing.
8. Vocabulary richness: The training dialogues contain around 28,935 unique words (after removing English stopwords). This large vocabulary reflects the informal, varied nature of chat-style text, including colloquialisms, personal names, and spelling variations.
   1. **Prepare the data for input to the BERT model**
9. Implement appropriate tokenization:
   * Tokenizer & lengths: Used BERT2BERT tokenizer with max\_input\_len=512 (long dialogues) and max\_target\_len=128 (short summaries), per EDA.
   * Define preprocess function: Tokenizes dialogue → input\_ids & attention\_mask and summary → labels (seq2seq-compatible).
10. Create training and validation splits:

* Applied preprocess() to train and validation from the tokenizer and length from the step above.
* Dataset was already split by source, so no manual split creation was required.

1. Build data loaders for efficient model training:

* Set PyTorch format: Kept only training-relevant columns (input\_ids, attention\_mask, labels) and converted them to tensors → less memory & faster I/O (Flatiron: resource-aware).
* Dynamic padding: Used DataCollatorForSeq2Seq with padding="longest" and label\_pad\_token\_id=-100 (masked in loss); pad\_to\_multiple\_of=8 for GPU efficiency (Flatiron consideration: resource-aware).
* Subset toggle for fast iteration: USE\_SMALL\_SUBSET=True to prototype on ~1–5% of data (e.g., 800/200) before scaling up (Flatiron: start with a subset, balance time vs. performance).
* DataLoaders: Built train\_loader / val\_loader with conservative BATCH\_SIZE and worker settings appropriate for Colab (Apply Pytorch).

### Model architecture selection and implementation

* 1. **Implement an encoder-decoder architecture using BERT.**

Adopt a pretrained BERT2BERT encoder–decoder (BERT encoder + BERT decoder). Meets the “use BERT” requirement, is free, stable, and purpose-built for abstractive summarization. Initialize tokenizer/model from the same checkpoint; set decoder start (CLS), end (SEP), and pad tokens; keep max input = 512 and target ≈ 128.

* 1. **Configure the model for the summarization task.**

Training config: Small, resource-aware start—batch size=8, epochs=1, LR=5e-5, weight decay=0.01, mixed precision on GPU. Use ROUGE. Trainer builds cleanly with datasets/collator/metrics; periodic eval runs; ROUGE reports reliably; checkpoints save per schedule.

* 1. **Set up the necessary components:**
     + 1. Encoder (BERT-based)

Use a pretrained BERT2BERT checkpoint (BERT encoder + BERT decoder) as the main architecture.

* + - 1. Generation mechanism to include the decoder. A decoder example can be Chat GPT-2 or model on huggingface.   
         Use the BERT decoder within BERT2BERT and call generate() for summaries.

Try to find a free model that will give proof-of-concept for text.

Primary PoC: patrickvonplaten/bert2bert\_cnn\_daily\_mail (free on Hugging Face). We will run a quick generation to verify the pipeline, then fine-tune on SAMSum.

### Training And Optimization

* 1. **Implement the training loop.**

The loop will include checkpointing, periodic evaluation, and early stopping to ensure efficiency and prevent overfitting. We will fine-tune the encoder–decoder model using Hugging Face’s Seq2SeqTrainer, configuring resource-aware training on GPU.

* 1. **Set up appropriate loss functions and evaluation metrics.**

Built-in cross-entropy loss for sequence generation and apply ROUGE scores as the primary evaluation metrics, with ROUGE-Lsum guiding early stopping and model selection.

* 1. **Configure optimization parameters.**

Refine the training setup by explicitly configuring the optimizer, learning rate schedule, warmup, gradient clipping, and label smoothing, while keeping GPU-aware settings for efficient and stable training.

* 1. **Implement early stopping and checkpointing.**

Integrate early stopping and periodic checkpointing so training halts when validation performance stops improving, while automatically retaining the best model.

* 1. **Monitor training progress.**

Track loss and ROUGE with periodic logs and evaluations, persisting metrics and state files for reproducible monitoring.

* + 1. Train and evaluate BERT2BERT with a small subset (≈800/200/200).
    2. Train and evaluate BERT+GPT2 with a small subset
    3. Compare two models, in this case BERT2BERT vs BERT+GPT2 according to the ROGUE metrics, we will select the best model, which it will be the baseline for the subsequent optimization
  1. **Manage computational resources effectively.**

Optimize GPU usage (A100 or L4) through mixed precision, length-grouped batches, and conservative generation/evaluation settings to fit memory, accelerate training, and avoid waste. Precision settings are configured as: bf16=True, fp16=False, tf32=True. These practices were applied consistently in Sections 3.1–3.5 to establish the baseline with efficient GPU performance.

This stage is divided into two optimization phases:

* + 1. **Training Hyperparameter Optimization (Phase A)**  
       After selecting the best baseline model, we will explore core training parameters that affect learning and generalization, including learning rate, label smoothing, and weight decay. The objective is to identify the most stable and performant training configuration.
    2. **Decoding Parameter Optimization (Phase B):**  
       Once the optimal training setup is established, we will tune decoding strategies to maximize summary fluency and conciseness. Parameters include beam size, maximum generation length, length penalty, and no-repeat n-gram size. These runs require no retraining, only re-evaluation with different decoding configurations. The goal is to identify inference settings that yield the best ROUGE without compromising runtime.

## Evaluation and Analysis

Evaluation scope & phase checks. Steps 4.1–4.5 apply to the optimization process described in 3.6. After Phase B, the best decoding configuration will be selected on the small dataset, and only then will training and evaluation be scaled to the full dataset for final reporting. Efficiency metrics will be considered at that stage, not during optimization.

* 1. **Evaluate model performance using ROUGE scores.**

In line with 3.6, validation ROUGE-L serves as the primary criterion for progression between phases. ROUGE-L is prioritized because it captures overall sequence-level overlap and is widely used as the standard metric in summarization tasks. If multiple runs are close in performance, preference is given first to higher ROUGE-2, which emphasizes bigram consistency and reflects local coherence, and then to lower evaluation loss, which ensures stability of the training process. When improvements are only marginal, a limited number of additional runs may be explored near the best configuration; otherwise, the process advances to the next phase to conserve compute.

After these checks, evaluation is conducted on the validation/test splits, collecting ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-Lsum to measure n-gram overlap with human-written summaries.

* 1. **Analyze model outputs qualitatively.**

Select 2–3 representative dialogues. Judge clarity, conciseness, faithfulness. Success = outputs are understandable and aligned with the dialogue in most cases.

* 1. **Compare generated summaries with reference summaries.**

Check how closely outputs align with the gold-standard human summaries (Dataset) and note whether key facts and conversation outcomes are preserved.

* 1. **Identify patterns in model successes and failures.**

The patterns in the sample (2-3) would be categorized by:

* Success cases: shorter, well-structured dialogues → concise, accurate summaries.
* Failure cases: very long or noisy dialogues → dropped details, hallucinations, or redundant phrases.
  1. **Consider model limitations and potential improvements.**

Limitations: small dataset, risk of factual drift, sensitivity to max length.

Improvements: more training epochs, hyperparameter tuning (beam size, LR), domain adaptation, or larger pretrained models (e.g., BART, T5).

## Documentation and Presentation

* 1. **Document your approach and findings.**

All the decisions and results will be identified in the Colab Notebook

* 1. **Create visualizations to illustrate results.**

The result will be presented in a visualization: Top ten unique words in result, Histogram with the length of the summaries, Histogram Average words per summary, Histogram compression ratio

* 1. **Prepare a concise presentation of your solution.**

The result will be explained in a Power Point presentation

* 1. **Discuss potential applications and extensions.**

Recommendations and challenges for future tasks

## Conceptual Representation

The following visual represents the five core steps

## Methodology and Justification

### Why BERT-based encoder-decoder architecture is appropriate for this task?

A BERT-based encoder–decoder is well suited for **dialogue summarization** because:

1. **Deep bidirectional understanding.**  
   The encoder is BERT, which reads each dialogue turn with full left-to-right and right-to-left context. This is critical in conversations where meaning depends on the back-and-forth between speakers.
2. **Natural abstractive generation.**  
   The decoder, also BERT (or GPT-style in a hybrid), is trained to generate new sentences rather than copy chunks of the input. This matches the SAMSum task, which requires concise *abstractive* summaries rather than simple extracts.
3. **Free, pretrained availability.**  
   Hugging Face provides pretrained BERT2BERT models (e.g., CNN/Daily Mail checkpoint), giving us a strong starting point at no cost. This reduces training time and makes the solution feasible in a bootcamp/Colab setting.
4. **Proven performance in NLP.**  
   Encoder–decoder transformers are the state-of-the-art backbone for text-to-text tasks (translation, summarization, dialogue modeling). BERT-based versions bring stability and strong language understanding into this architecture.

In few words, **BERT encoder–decoder** is the best option because it *understands dialogues deeply* and can *generate concise summaries naturally*. It is free, pretrained, and proven effective for summarization tasks like SAMSum.

### Advantages of fine-tuning pre-trained models vs. training from scratch

1. **Massive knowledge transfer.**

Pretrained models like BERT have already been trained on billions of words. Fine-tuning reuses this rich language understanding, so the model already “knows” grammar, semantics, and common expressions before seeing our dataset.

1. **Data efficiency.**

The SAMSum dataset is relatively small (~16k dialogues). Training from scratch would lead to poor generalization, but fine-tuning lets us adapt a large pretrained model with much less labeled data.

1. **Faster convergence.**

Pretrained weights start close to a good solution, so the model reaches high performance in far fewer epochs compared to scratch training, saving both time and compute.

1. **Lower compute cost.**

Pretraining transformers from scratch requires enormous GPU resources (weeks on clusters). Fine-tuning only needs modest resources (e.g., Colab GPU) and is feasible in an academic/bootcamp setting.

1. **Proven best practice.**

State-of-the-art results across NLP tasks (summarization, translation, question answering) all come from fine-tuning pretrained models rather than training from scratch.

We fine-tune a pretrained BERT2BERT because it’s faster, cheaper, and more accurate. Training from scratch would demand huge data, human power and compute resources we don’t have, while fine-tuning transfers knowledge from general language into our specific dialogue summarization task.

### Why certain evaluation metrics (ROUGE, human evaluation) are most suitable?

1. **ROUGE (automated, objective).**
   * ROUGE measures the overlap of words and phrases between the generated summary and the human-written reference.
   * It captures how much key information was preserved, which is critical for evaluating summarization quality.
   * ROUGE is fast, reproducible, and widely accepted in the NLP community, making results easy to benchmark against prior work.
2. **Human evaluation (subjective, nuanced).**
   * Automated scores like ROUGE cannot judge fluency, coherence, or factual consistency.
   * Human reviewers can assess whether the summary reads *well*, is faithful to the dialogue, and useful to an end-user.
   * This balances the limitations of automated metrics by adding a quality check from a user perspective.
3. **Why these are most suitable together?**
   * ROUGE gives a scalable, quantitative baseline.
   * Human evaluation ensures real-world usefulness and readability.
   * Using both covers both information accuracy (ROUGE) and language quality or human judgment.

ROUGE is used because it’s the standard automated metric for summarization and complements with human evaluation because they can judge fluency and usefulness better than numbers alone.

### Appropriate optimization techniques for this specific NLP task

Following some potential techniques that could be explore

* **AdamW Optimizer:** Widely used for transformer models, combines Adam’s adaptive learning rates with weight decay to reduce overfitting and helps stabilize fine-tuning on small datasets like SAMSum.
* **Learning rate scheduling:** Using a small constant learning rate (e.g., 5e-5) is effective for pretrained transformers and it can be combined with a warm-up phase to avoid large gradient updates at the start of training.
* **Gradient clipping:** Prevents exploding gradients in long sequences, ensuring stable training.
* **Early stopping:** Monitors validation performance and halts training when no improvement is observed and saves compute and prevents overfitting.
* **Mixed precision training (fp16):** Reduces memory use and speeds up training on GPUs while maintaining accuracy. It’s particularly useful in Colab or limited GPU settings (like mine).
* **Dynamic padding with masked loss:** Ensures batches are padded only to the longest sequence in the batch and labels are padded with -100 so padding tokens don’t affect loss computation.

AdamW was used with a small learning rate, gradient clipping, and early stopping to make training stable and efficient with mixed precision and dynamic padding in Seq2Seq. Thanks to them, GPU memory and time was saved, which is critical for summarization on long dialogues.

## Alignment with Requirements

### Fulfills all project deliverable requirements

The deliverables approach aligns with the Flatiron Lab learning objectives required:

**Implement an encoder-decoder architecture using pre-trained BERT models:** itwill be implemented with a free pretrained BERT2BERT checkpoint; special tokens configured; generation enabled via generate(). This is part of model architecture selection and implementation

For **data preparation**, it will be used tokenization for dialogues (≤512 tokens) and summaries (≤128), dynamic padding, label masking, and cleaned splits. The number are chosen to demonstrate a decrease of the characters. The dataset will also be analyzed, clean missing values and build loaders. This is part of model architecture selection and implementation and dataset exploration and preparation

**Training & evaluation** with resource-aware Seq2SeqTrainer setup, ROUGE-based evaluation, optional early stopping, and reproducible seeds. All in Training and Optimization

**Proof-of-concept generation:** A free model produces summaries end-to-end. Flatiron emphasized subset-first training, resource-aware training, beam search, meaningful outputs.

### Addresses the core business needs

The core business need addressed by this project is the ability to summarize multi-turn dialogues into concise, actionable text. In real-world settings, conversations happen across support tickets, chat platforms, sales calls, and team messaging tools. These dialogues are long, noisy, and time-consuming to review.

The solution is not about interacting with users in real time, which is out of scope. Instead, the focus is on offline summarization of completed dialogues for downstream use. The solution provided will be:

* **Summarization at scale** → Automatically condenses entire conversations into short, human-readable summaries.
* **Time savings & signal extraction** → Reduces manual review time by highlighting key decisions, next steps, or customer issues.
* **Consistent quality monitoring** → Generates measurable outputs (via ROUGE scores) and enables human spot-checks for fluency and faithfulness.

### Balances technical performance with practical considerations

Based on Flatiron consideration to optimize the performance, these are some points will be implemented in the project:

* Free & open stack: Hugging Face datasets/models (SAMSum, BERT2BERT) and built-in metrics (ROUGE) with no paid APIs and logging disabled (report\_to="none").
* Subset-first prototyping. Start with a small slice of train/val for quick iteration; switch to full data only after the pipeline is stable.
* GPU-aware efficiency. On Colab GPUs we enable mixed precision (fp16) and pad to multiple of 8 for Tensor Cores. Works on CPU too (falls back automatically).
* Minimal memory footprint. Dynamic padding per batch + label masking (-100) avoids wasting memory on long max lengths.
* Conservative training defaults. Batch size = 8, short epochs, AdamW with small LR, and optional early stopping to save time/compute.
* Better decoding, bounded cost. Beam search = 4 and max\_new\_tokens = 64 give cleaner summaries without exploding generation time.
* Reproducible & resumable. Checkpointing to output\_dir, fixed seed=42, and resumable runs fit classroom hardware and timelines.

### Produces outputs that are meaningful for the business context

* Concise, actionable summaries. The system delivers short, to-the-point summaries (1–3 sentences) that capture the essence of a conversation. These can be directly reused in tools like CRM systems, support dashboards, or managerial reports.
* Consistent format. Summaries follow a stable structure and length, which makes them easy to integrate into existing workflows — for example, filling in “next steps” or “key issue” fields automatically, or enabling reliable analytics.
* Clear success measures. Quality is monitored with both automated benchmarks and occasional human spot-checks. This ensures the summaries are not only faster to produce, but also accurate and useful for decision-making.

## Timeline

Based on Flatiron’s check list, have the timeline for the project. Monday to Saturday corresponds to the week 09-01-25 to 09-06-25

| **Phase** | **Tasks** | **Estimated Time** | **Deliverables / Outcomes** |
| --- | --- | --- | --- |
| **Research & Preparation** | • Review transformer encoder–decoder architectures (BERT2BERT, alternatives)  • Research current state-of-the-art in dialogue summarization  • Study evaluation metrics (ROUGE, human eval)  • Explore SAMSum dataset characteristics (dialogue length, turns, vocab, compression ratio) | 1–2 days  Mon - Tue  (8–10 hours) | Background notes, dataset exploration notebook |
| **Implementation – Phase 1** | • Data preprocessing & cleaning (handle missing values)  • Exploratory analysis (turns, vocab, compression ratio)  • Tokenization setup (max lengths) | 1 day  Tue  (6 hours) | Cleaned SAMSum dataset, preprocessing functions, tokenized samples |
| **Implementation – Phase 2** | • Model architecture implementation (BERT2BERT)  • Configure Trainer (collator, metrics, training args)  • Proof-of-concept generation | 1–1.5 days  Tue - Wed  (10–12 hours) | Functional encoder–decoder pipeline, baseline summary outputs |
| **Implementation – Phase 3** | • Training setup & optimization (subset-first training, scale up)  • Apply resource-aware techniques (fp16, dynamic padding, early stopping)  • Save checkpoints & configs | 1 day  Thu  (8–9 hours) | Trained model checkpoint(s), training logs, ROUGE evaluation |
| **Evaluation & Iteration** | • Evaluate results (ROUGE + manual review on 2–3 examples)  • Identify refinement opportunities (adjust beams, epochs, LR)  • Incorporate feedback from peers’ session  • Explore alternative approaches like GPT2 | 1 day  Fri  (5–6 hours) | Evaluation report with ROUGE + qualitative notes, refined model or alt approach |
| **Documentation & Delivery** | • Document methodology, rationale, and success criteria  • Prepare slides & visuals  • Draft final report & presentation  • Video Shooting and edition  • Integrate feedback from peers before submission | 1–1.5 days  Fri - Sat  (8–10 hours) | Final report, video, presentation deck |
| **Risk Management (ongoing)** | • Mitigate Colab/GPU limits (subset-first training, checkpointing)  • Allow buffer time for technical roadblocks or library issues  • Allocate contingency hours in case of reruns | Extra buffer ~0.5 day for troubleshooting | Embedded across phases |