Replicating DecSum: From Text Summaries to Decision Consistency

Zongqi (Lawrence) Lu

1 Executive Summary

Decision-focused summarization algorithm seeks to generate summaries that optimize decision quality rather than textual similarity. DecSum (Hsu & Tan, 2021) formalizes this by combining three objectives – *faithfulness, representativeness,* and *non-redundancy* – to guide summary selection through a beam-search optimization.

This project replicates the full DecSum architecture, including data pre-processing, model training, and summary generation, to test whether the decision-focused framework can be reproduced and whether its performance trends generalize. The Yelp restaurant reviews dataset was reprocessed following the same filtering criteria, and a Longformer-base-4096 model was fine-tuned to predict restaurant ratings from the first ten reviews. The DecSum search algorithm was implemented with its three objective components and optimized via a recursive beam search (beam size = 4, summary length = 5).

The replicated model achieves a mean squared error compared to full summary (MSE) of around 0.00001, comparable in magnitude to the original paper, and reproduced same qualitative trends:

- MSE decreases with increasing beam size, with optimal summary length around 3 sentences.
- The model balances faithfulness and representativeness when the beam size is sufficiently large.
- Total runtime for the full test set (4,205 restaurants) was 9 hours on one A100 GPU, consistent with the paper's reported efficiency.

Metric	Paper (Hsu & Tan, 2021)	Replication	Note
MSE with full	0.0005	0.00001	
Runtime per restaurant	15.5s	7.5s	Replication uses 5 sentences instead of 6
Optimal summary length	3	3-4	

Table 1: Key results and comparison between paper and replication

More discussion on the architecture and results can be found later. My implementation uses a recursive search for DecSum, vectorized batched inference for efficiency, some caching to prevent redundant computation across recursion levels, and an optional normalization of the loss term to balance scaling differences. These design choices preserved algorithmic integrity and improves code readability and efficiency.

Major challenges I encounter involved mastering unfamiliar modules such as PyTorch Lightning and SentenceTransformers, optimizing GPU utilization, and debugging recursive search logic. Debugging has been especially difficult in the early implementation because of the recursive design, multiple components of loss, and unfamiliar functions. These frustrating moments forced me to develop better habit of modular coding, logging, and systematic testing. Optimizing runtime was very rewarding, seeing the runtime decrease from over 3 minutes to under 10 seconds. This process taught me that in a scientific context, having a running program is only the first step – careful performance engineering, extended testing, and proper logging are also crucial for a successful implementation.

All code, pretrained models, and sample outputs are available in my GitHub Repository. Overall, this replication confirms the reproducibility of decision-focused summarization framework DecSum and its ability to achieve both accuracy and performance in decision-making tasks that rely on textual input.

2 Introduction and Background

In domains like report aggregation and policy analysis, human decisions often rely on summaries of long texts. While standard NLP summarizers focus on textual relevance and conciseness, they often overlook *decision utility* - whether conclusion drawn from summaries align with those based on full documents.

A decision-focused summary addresses this gap by optimizing summaries not only for readability but also for faithfulness to decisions. The DecSum framework (Hsu & Tan, 2021) formalizes this approach through three objectives:

1. Faithfulness: decision predicted from the summary should be close to that from the full text.
2. Representativeness: the summary should capture the diversity of opinions across all reviews.
3. Non-redundancy: the summary should minimize semantic overlap.

In the original paper, the authors applied DecSum to Yelp restaurant reviews, where a regression model was trained to predict average restaurant ratings from text. Summaries were evaluated by the mean squared error (MSE) between the predicted ratings based on summaries versus full reviews, with smaller MSE indicating better decision faithfulness.

This project replicates DecSum to assess its reproducibility and practical performance by:

- Reconstructing preprocessing and data sets from Yelp reviews,
- Reimplementing the fine-tuned regression model for decision prediction,
- Building the DecSum beam search with faithfulness, representativeness, and redundancy, and
- Evaluated summary quality quantitatively and qualitatively

3 Implementation and Design Choices

3.1 Data Preprocessing

The preprocessing pipeline closely follows Hsu and Tan (2021). I filtered the Yelp Dataset to include only businesses categorized as "Restaurants" with at least 50 reviews to generate a reliable prediction benchmark. The dataset contained 21,021 restaurants after filtering and was split into train/validation/test sets in a 64/16/20 ratio.

For each restaurant, the earliest 10 reviews and their corresponding ratings were used as model input, while the mean of the first 50 reviews served as the baseline rating for supervision. This setup simulates a decision-maker reading a limited number of reviews before forming an overall opinion.

3.2 Model Architecture

The score prediction model is a fine-tuned transformer encoder based on Hugging Face Transformers Longformer-4096 (Beltagy et al., 2020), chosen for its sparse attention that allows to process the 10 summaries (1,000 - 1,500 tokens long). The model was trained as a regression task to predict the restaurant's average score from the concatenated reviews. Because model training is not the core novelty of DecSum, I followed the paper's reported hyperparameters rather than performing a hyperparameter search.

Hyperparameter	Value	Note
Base model	allenai/longformer-base-4096	Transformer with sparse global attention
Max input length	3000	
Learning rate	5×10^{-5}	AdamW optimizer
Max epochs	3	Save best model on validitation MSE
Batch size	4	
Scheduler	Linear warmup	warmup step = 500
Percision	half	

Table 2: Model hyperparameters and training configuration

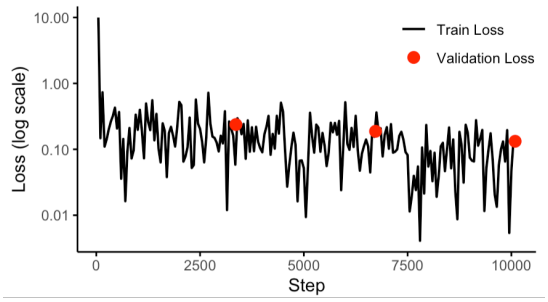

Figure 1: Model Training Loss and Validation Loss

3.3 DecSum Algorithm

The DecSum search procedure forms the core of this project. Its goal is to identify a subset of sentences (or sliding-window chunks) that minimizes the combined loss of faithfulness, representativeness, and non-redundancy:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{faith}} + \beta\mathcal{L}_{\text{repre}} + \gamma\mathcal{L}_{\text{redun}} \tag{1}$$

Each component was implemented as follows:

- Faithfulness ($\mathcal{L}_{\text{faith}}$): Log distance between the summary's predicted score and the full-text prediction, using the trained Longformer model.
- Representativeness ($\mathcal{L}_{\text{repre}}$): Log Wasserstein distance between the distribution of sentence-level predictions and that of all reviews.
- Non-redundancy ($\mathcal{L}_{\text{redun}}$): Sum of cosine distance among embedded sentence representations.

My implementation employs a *beam search* strategy: at each iteration, all remaining candidate sentences are scored, and the top beam combinations are expanded recursively until the target summary length is reached or if there are no candidates left. The key configuration parameters used in this implementation are summarized below.

Parameter	Value	Note
Default beam size	4	
Default summary length	5	Reduced from 6 (paper default)
Window size	3	Sliding window for chunking
Min tokens	3	Discard short sentences
Embedding model	distilbert-base-nli-stsb-mean-tokens	Sentence encoding for redundancy
Parser	en_core_web_sm	Sentence segmentation

Table 3: DecSum replication configuration

3.4 Efficiency Improvement and Tradeoffs

Initial implementation required over 3 minutes per restaurant due to repeated inference calls within nested beam loops. I applied several optimizations to make the algorithm computationally efficient:

In the production run permutations of same sentences are recomputed because the effect of rearrangement is not clear. However, the candidate that has overlap with selected sentences were eliminated. Furthermore, I noticed very little performance gain on parallelization because the task is GPU bound and prone to out-of-memory error without queue management. The replication model uses threading instead. Overall, my implementation replicates the full DecSum pipeline, differing mainly in engineering detail and optimization strategy.

4 Results and Analysis

To evaluate the replication quality of the DecSum framework, I conducted extensive tests across varying summary lengths, beam sizes, and scoring configurations. All results are from a single A100 GPU

Strategy	Description	Rationale	Implication
Batched prediction inference	Aggregate all candidate sentences into one model call	Reduces kernel launch overhead	Potential out-of-memory error when summary is too long
Precompute sentence score and cosine matrix	Compute once at outmost loop and pass in down recursion	Avoid redundant computation	None
Index-based lookup	Use integer indices instead of text keys	Stability, low memory cost	None
Parallelization trials	Multiprocessing on CPU	Spread out CPU-bound tasks	Racing for embedding model; potential out-of-memory error
Candidate pruning	Delete candidates that overlap with selected sentence or has the permutation checked before	Exclude overlapping chunks and rearrangement of same summary	Rearrangement may affect prediction result

Table 4: Summary of optimization strategies and expected impact

(40GB CUDA memory). Runs with sentence length 5 and beam size 4 are run on the full Yelp test set (n=4205), and most test runs use the first 512 restaurants due to limited computational power.

4.1 Runtime Performance

The replicated DecSum implementation achieves comparable speed to the original paper despite architectural differences. On the full test set (beam size = 4, summary length = 5), inference completed in 8.75 hours (7.5s per restaurant). This is nearly twice as fast as the paper's reported 15.5s per restaurant with one more sentence.

4.2 Accuracy and Representativeness

The replicated model achieved a mean squared error compared to full review (MSE with full) of 0.00001, comparable to the paper's benchmark (0.0005). The summary-level predictions closely matched those from the full 10-review model.

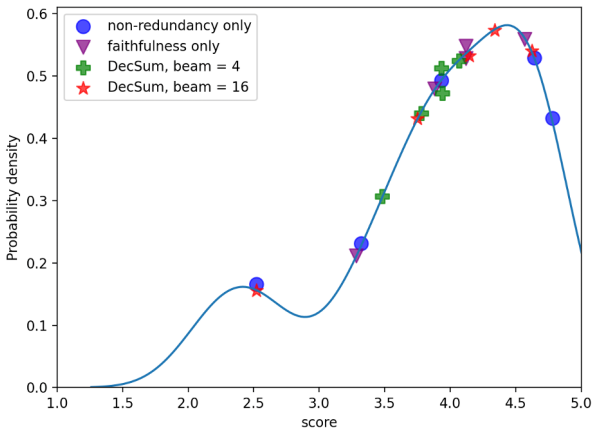

Figure 2: Distribution of sentence-level prediction scores

Visual analysis of sentence-level prediction distributions revealed that faithfulness-only models tend to concentrate around the main rating peak, whereas full DecSum captures both high- and low-score regions. Surprisingly, DecSum with beam size 4 ignored the secondary peak, but increasing beam size from 4 to 16 improved representativeness by recovering it.

4.3 Components Analysis

To isolate the contribution of each loss term, I ran tests with subsets of the three components of $\mathcal{L}$. All experiments use beam size = 4 and sentence length = 5.

Configuration	(α, β, γ)	MSE	Runtime
Full model	$(1, 1, 1)$	0.000011	7.49s
Faithfulness only	$(1, 0, 0)$	0.000005	7.04s
Non-redundancy only	$(0, 0, 1)$	0.200138	0.96s
Faithfulness and non-redundancy	$(1, 0, 1)$	0.095282	0.90s

Table 5: Model hyperparameters and training configuration

One potential reason for this the faithfulness in fact has the highest weight, as its loss range from -13.81 to -5.54 while the representativeness and non-redundancy score have a range of less than 1. If the three loss terms are individually normalized a significant increase in MSE can be observed. Adjusting the weight could balance the tradeoff between representativeness and accuracy.

4.4 Sensitivity to Length and Beam Size

To study scalability, I varied both summary length and beam size. Increasing either parameter yields roughly linear runtime growth. Higher beam size leads to lower error, but after a certain threshold longer summary leads to larger error. Empirically, summary length = 3 and beam = 4 provides the best tradeoff, with MSE with full < 0.02 and runtime of around 3s per restaurant.

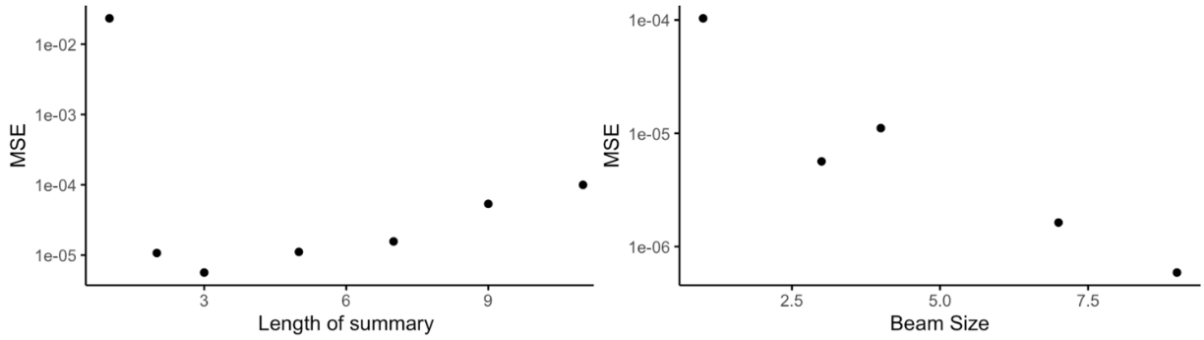

Figure 3: MSE vs length with beam size = 4 (left) and MSE vs length at length = 5 (right)

4.5 Sentence Order

In agreement with Hsu and Tan (2021), DecSum exhibits sensitivity to sentence order. Reordering summaries into the original chronological order, contrary to the paper result, increased MSE from 0.00001 to 0.00081. This suggests that the Longformer's attention weights encode local sequence dependencies, even though the summarization objective is order-invariant.

4.6 Lightweight Approximation Model

To further improve efficiency, I developed a "lightweight" version of DecSum that averages precomputed sentence-level scores instead of recomputing them during recursion.

With the same length and beam size, this variant is 7 times faster (1.08s) but produced an MSE = 0.012, over 1000 times higher than the main model. Larger beam sizes partially mitigated this: at beam = 16, the MSE fell below 0.02 while still being 4.4 times faster than the main model (1.69s).

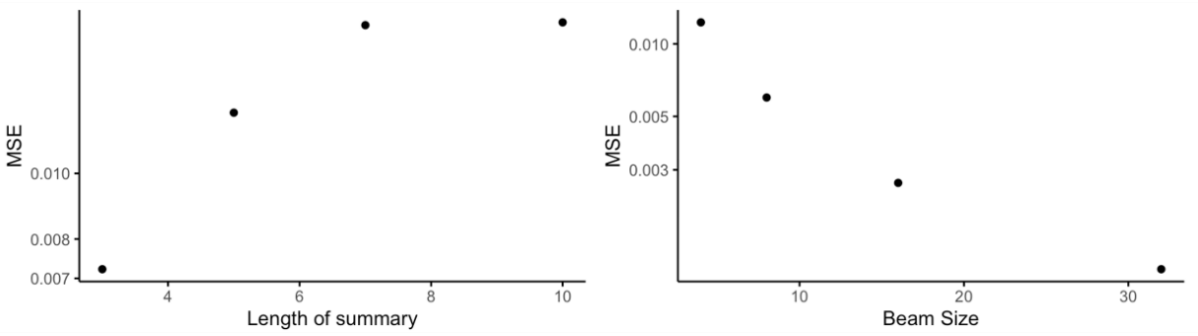

Figure 4: Lightweight MSE vs length with beam size = 4 (left) and MSE vs length at length = 5 (right)

These experiments confirm that while the lightweight model provides dramatic speedups, it sacrifices faithfulness accuracy. A greedy search (main model with beam size 1) is slightly slower (2.27s per restaurant) but has 50 times lower MSE error. For practical deployment, reducing beam size in the full model yields better speed–accuracy tradeoffs.

4.7 Summary of Key Findings

Experiment	Beam	MSE	Runtime	Takeaway
Main DecSum	4	0.00001	7.5s	Matches paper accuracy and performance
Greedy DecSum	1	0.0001	2.3s	Good performance-accuracy tradeoff
Chronological Order	4	0.0008	6.9s	Order matters
Faithfulness only	4	0.000006	7.0s	Dominates objective
Lightweigth	16	0.012	1.7s	Accuracy drop

Table 6: Summary of main results

Overall, the replication matches the original DecSum both qualitatively and quantitatively. This confirms that DecSum is replicable and outperform text-based extractive approaches.

5 Discussion and Reflection

This project provided me a deep, hands-on understanding of decision-focused summarization and the technical challenges of reproducing a complex NLP pipeline. Through replicating Hsu and Tan (2021)'s DecSum framework, I learned to implement, test, and optimize models built upon advanced libraries such as PyTorch Lightning and SentenceTransformers. The experience revealed how much of real-world machine learning work lies not in algorithm design but in engineering efficiency, numerical stability, and reproducibility.

5.1 Implementation Differences and Design Choices

While the overall structure and objective of my DecSum replication followed the paper, several implementation details differ by design.

- **Recursive search**: I implemented DecSum as a recursive beam search (`decsum_search`) that explicitly tracks candidate and selected lists at each level. This structure made the algorithm conceptually clean and easy to debug, though it introduced additional Python overhead. The original paper used an iterative beam-update loop within a single function.
- **Faithfulness computation**: My model evaluates faithfulness by repeatedly calling the Longformer predictor on all partial summaries (`faith_score`). The original paper calls the model only once per beam expansion, which might be computationally lighter.
- **Sentence segmentation**: My implementation uses a consistent SpaCy-based window segmentation, whereas the paper employed multiple heuristics (e.g., `window_k` that uses a sliding window

of k sentences, `longest_n`, that selects n longest sentences, and `summary_r` that keep r ratio of text). My implementation simplifies hyperparameter control but is less versatile.

These design choices make my version easier to extend and cleaner at the cost of more overhead and more inference calls.

5.2 Future Directions

Three main improvements could further strengthen the model:

- **Robustness**: The current implementation assumes valid data set input and sufficient candidate sentences for all beam expansions. This might fail for long summaries (length ≥ 10). Adding exception handling and adaptive beam trimming would make the model more robust.
- **Parameter modularity**: Most hyperparameters are currently hardcoded. Converting them to external configuration files or command-line argument would streamline parameter sweeps and facilitate reproducibility.
- **Caching for repeated runs**: Implementing caching of cosine matrices, sentence embeddings, and per-sentence prediction scores would yield major runtime savings. These elements are deterministic across runs and can easily be reused. I expect this to bring the greatest performance gain.

Beyond these engineering improvements, the project highlights an important scientific insight: decision-focused summarization depends strongly on the faithfulness component. My experiments showed that even with balanced (α, β, γ) weights, the faithfulness term numerically dominates, implying that future work should focus on dynamic reweighting or normalization to achieve better representativeness without sacrificing accuracy.

5.3 Overall Reflection

I faced several challenges during development. At first, integrating high-level libraries such as PyTorch Lightning and SentenceTransformers felt overwhelming, as I had to deal with the complicated libraries and their jargon. There were moments when debugging a single recursive loop took hours, but these difficulties forced me to write cleaner, more modular code.

This replication confirmed that the DecSum framework can be reproduced faithfully, with comparable MSE accuracy and faster runtime under certain configurations. The main lesson is that small architectural and implementation decisions – such as caching, normalization, and recursion – can have large effects on efficiency and stability. These are the kinds of details that rarely appear in papers but matter enormously for reproducibility and scalability.

More broadly, this project changed how I think about research code. I start by trying to "make it run," but ended up learning how to make it efficient, transparent, and trustworthy. It taught me that replication is not just about reproducing numbers but a way to understand the deeper logic of a model by rebuilding it from the ground up. I now have a much clearer sense of how decision-focused NLP connects modeling, optimization, and real software engineering, and how thoughtful replication can both validate and extend existing research.

References

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. *arXiv:2004.05150*.

Hsu, C.-C., & Tan, C. (2021). Decision-focused summarization. *arXiv preprint arXiv:2109.06896*.