Semantic Compression: Streamlining Text Transmission with Large Language Models

Luning Yang, Yacun Wang
University of California, San Diego
{l4yang, yaw006}@ucsd.edu

Abstract

Text compression is a well-studied area in computer science, and traditional lossless algorithms focused on storage optimization and network load reduction. While such algorithms are able to recover the original data exactly, there are plenty of applications where lossy compression could be accepted by achieving higher compression ratios while sacrificing some information. With the rapid improvements in the capabilities for Large Language Models (LLMs) to understand natural language, this paper explores semantic text compression using these LLMs. We propose a baseline method and a more advanced LLM text compression pipeline aimed at improving the tradeoff between retained information and compression efficiency. Experiments across multiple datasets demonstrate that the proposed pipeline significantly reduces text size while maintaining satisfactory semantic similarity between original and decompressed texts. In addition, the compressed texts show that key information are retained as the performance on downstream tasks mostly remain level.

1 Introduction

General data compression algorithms, specifically text compression, has been a traditional computer science research area that has been extensively studied. Popular text compression algorithms usually discovers patterns in strings and store the patterns efficiently. Such algorithms are widely used in data archives to save storage spaces or data transmissions to reduce network burden. However, in the recent era where massive text data is prevalent, and transferring such data becomes increasingly harder, there still exists the need to even more aggressively reduce the size of such data.

Traditional data compression algorithms (Deutsch, 1996; Seward, 1998) are usually lossless, meaning that they could retain 100% of the original information after decompression. In the contrary, there are also lossy algorithms that could potentially compress the original data into smaller sizes, but also suffers from the fact that the original data cannot be fully recovered. There exists a trade-off between the ultimate size and the amount of information retained. In the context of text compression, while users might choose the compression algorithms for specific use cases, we argue there would be plenty of applications under either setting:

- **Lossless**: Where exact recovery is necessary, such as code repositories, literature pieces, etc.
- **Lossy**: Where understanding the meaning of texts suffice, such as lecture notes, online messaging archives, reducing LLM context size, etc.

In recent years, Large Language Models (LLMs) have been improving at a high pace with the an increasingly better capability to understand natural language. Such capability enables LLMs to become perfect agents to help understand and compress text. While most recent studies (Gilbert et al., 2023; Fei et al., 2024) attempted compression using LLMs, both focus on the objective to reduce LLM input context to avoid the degrading performance of LLMs when a long context input is given.

In this paper, we aim to explore the capabilities of text compression directly using large language models under the objectives that: (1) the original text and the generated text after decompression holds similar semantic meanings; (2) the compressed text is as short as possible. We wish to improve the Pareto frontier on the information retained-compression size tradeoff. In particular, we proposed a simple baseline compression method and a more advanced compression pipeline that utilizes the power of LLMs to achieve the objectives. From the experiments on multiple datasets, the advanced pipeline shows a promising compressed

text size while a reasonably satisfactory similarity between the original and decompressed text. The code repository could be found here[1].

2 Problem and Evaluation

We attempt to perform text compression using some large language model compressor f_c and decompressor f_d. For a particular document D,

- The compression stage should produce the compressed text $D_c = f_c(D)$
- The decompression stage should then take the compressed text D_c to produce $D_d = f_d(D_c)$

where we expect

$$D_c \approx \underset{s(D,D_d)\approx 1}{\operatorname{argmin}} |D_c|$$

Here $|\cdot|$ is the text size, and s is a similarity function. In full,

$$f_c(D) \approx \underset{s(D,(f_d \circ f_c)(D))\approx 1}{\operatorname{argmin}} |D_c|$$

To evaluate the effectiveness, we simply follow the objectives. Regarding the retained information, we simply used the score $s(D, D_d)$ where we choose the cosine similarity score for sentence transformer (SBERT) embeddings (Reimers and Gurevych, 2019). Specifically, we compute the SBERT embeddings for D and D_d respectively, denoted as v_D, v_{D_d}. Then the similarity score (i.e. information retained) is computed by

$$s_{D,D_d} = \frac{\langle v_D, v_{D_d} \rangle}{\|v_D\| \cdot \|v_{D_d}\|}$$

Regarding the compressed size, we define:

$$\text{Compression Ratio (CR)} = 1 - \frac{|D_c|}{|D|}$$

In other words, it represents the percentage of original text size that is saved during the compression process.

To further experiment the information retained and its effects on downstream tasks, we prompt the language models to perform simple classification tasks according to the labels given in the dataset and report their accuracies. Specifically, we use the simplest prompt to directly ask the LLMs to pick an option. The prompt could be found in Appendix A.3.

[1] https://github.com/colts661/LLM-Semantic-Compression

Dataset	Algorithm	CR
NYT	gzip	52.1%
	bzip2	52.2%
Yelp	gzip	49.2%
	bzip2	48.8%

Table 1: Compression Ratio for Traditional Lossless Algorithms

3 Background

Here we present some background information about text compression and related literature.

3.1 Traditional Algorithms

There exists plenty of traditional rule-based text compression algorithms, and we give two examples:

- **gzip**: Based on the DEFLATE algorithm (Deutsch, 1996) that combines LZ77 (Ziv and Lempel, 1977) redundant string elimination and Huffman coding (Huffman, 1952) bit reduction algorithms. It's widely considered as one of the most effective lossless text compression algorithms;
- **bzip2**: Stacks several layers of compression techniques that includes popular run-length encoding, Burrows-Wheeler transform (Burrows and Wheeler, 1994) that utilizes block sorting, Huffman coding (Huffman, 1952), etc. and sometimes considered even more effective than DEFLATE.

Both algorithms are considered as effective lossless text compression baselines. Using the datasets in Section 4, we present the compression rate results in Table 1. In general, both algorithms are able to reduce the size of each dataset to about half of its original size.

3.2 Compression vs. Embedding

Since the introduction of Word2Vec word embeddings (Mikolov et al., 2013), there emerged plenty of effective semantic embeddings including word-level BERT (Devlin et al., 2018) and sentence transformer (SBERT) (Reimers and Gurevych, 2019) that are showed in a wide variety of research to encode natural language in a lower-dimension semantic vectors while maintaining semantic meaning.

However, also mentioned in (Gilbert et al., 2023), there are clear distinctions between the concepts of compression and embedding. While embeddings

are able to encode text data into lower dimension latent numerical vectors, such process is one-way and those vectors don't allow a direct decoding into words. To recover the words, one needs to make additional predictions; it's almost impossible to recover the entire text just by looking at the semantic numerical vectors.

On the other hand, compression aims to minimize the size of the compressed text while maintaining the integrity of the original text. The process should be two-way, even if lossy compression algorithms might not recover the exact content. Any compressed text should be able to at least be recovered to a similar meaning compared with the original text.

3.3 Related Work

Some recent studies have looked into text compression using LLMs mainly as a preprocessing step for LLM inference. In particular, (Fei et al., 2024) attempted to compress LLM input context before feeding the compressed text for downstream tasks. The study proposes a compression pipeline by segmenting input into topic-based chunks then refine each chunk to only keep key information. Such compression is showed to perform well in downstream tasks, but the work focuses less on the compression size itself. A similar work (Ge et al., 2024) attempts to extend LLM context by compress the input into memory tokens which are then used to perform downstream tasks.

On the other hand, (Gilbert et al., 2023) works directly on semantic text compression using LLMs on short stories and code generation tasks. Specifically, the work showed LLMs' capability to compress literature into around $1/4$ of its original size while retaining around 75% of the original information. This work serves as a motivating work for this paper, as LLMs should be able to further compress the text on applications that don't require exact information retained.

4 Experiments

Due to the limited computational resource and the latency of receiving results from APIs serving LLMs, we uses the most cost-efficient and accessible `GPT-4o-mini` to conduct all the compression, decompression, and related evaluations.

To confirm on the effectiveness of the compression pipeline, we picked two datasets representing different text genres:

Dataset	# Classes	Avg # Characters
NYT	5 (Fine: 26)	3792.46
Yelp	10	2869.18

Table 2: Dataset Statistics

- **New York Times (NYT)**: The NYT dataset consists of news articles published by the New York Times. Such genre usually features well-written formal languages with relatively strong logic flow. The provided labels contain both coarse-grained and fine-grained ones, representing the topics of the documents

- **Yelp Reviews (Yelp)**: The Yelp dataset consists of user reviews on restaurants along with a few additional feature columns. We only use the review part as text compression. Such genre are user-written and should contain more colloquial language. The provided labels represent the restaurant cuisine, which is harder to predict than topic classification.

For each dataset, we randomly sample 1000 documents to perform all experiments, and the dataset statistics are showed in Table 2.

5 Baseline: Single-Stage Compression

In this section we present the baseline method and the discussion of its results and implications.

5.1 Method

Similar to (Gilbert et al., 2023), the simplest way to compress a text using LLMs is to create a detailed prompt specific to this task. To comply to the objectives, we specifically mentioned the expectations that:

1. Allows LLMs to generate non-human-readable texts that could be understood only by similar models;

2. Allows the mixed use of any symbols, emojis, etc. that could minimize the size after compression;

3. Request the model to keep all the information present in the original text

During the decompression stage, we explicitly mentioned that the text was compressed by the same family of models. Specific prompts could be found in Appendix A.1.

Dataset	CR	Similarity	Failure Rate
NYT	67.7%	0.777	35.8%
Yelp	69.7%	0.786	25.3%

Table 3: Compression Results for Baseline Compression Method

Figure 1: An example of compressed text using baseline method

5.2 Results and Discussion

We present the compression and decompression results in Table 3. Compared to traditional algorithm results in Table 1, we're able to compress each dataset to an average of only $1/3$ of their original size, but also sacrificed quite some information in the process. If we consider a similarity score between 0 and 1 as the percentage of information retained, then the baseline method only further compressed $1/6$ of the original information compared to the lossless algorithms but sacrificed more than $1/5$ information.

However, the largest problem with the baseline method lies in the failure rate. In the two datasets, while the compression rate is acceptable, the model itself is unable to decompress the text the same model just compressed in both NYT and Yelp datasets for a considerately large percentage. For example, the model will explicitly state that it was unable to process the compressed text instead of trying to generate incoherent non-sense language. Such possibilities highly degrades the baseline method as a failed decompression would lead to a failed data transmission or permanent data lost.

This situation happens mainly because we're allowing the LLM to compress as aggressively as it wishes, lacking specific guidelines. For example, the model would try to generate as many emojis as possible without considering negative effects. To alleviate this issue while exploring better compression rates, we present the multi-stage compression pipeline below.

5.3 Case Study

Figure 1 presents an example of compressed text in the NYT dataset using the baseline method. It shows that the LLM is aggressively trying to compress the text using as many continuous emojis as it could, but results in a decompression failure.

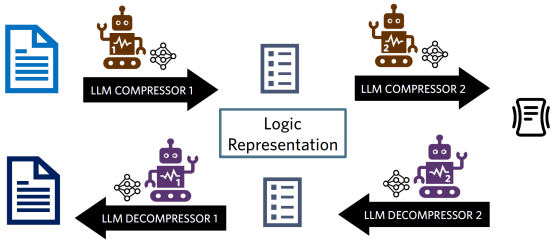

Figure 2: An Illustration on the Multi-Stage text compression pipeline where an intermediate "logic representation" stage is added to guide LLMs

6 Multi-Stage Compression

In this section we present the more advanced method and the discussion of its results and implications.

6.1 Method

To further improve on the compression ratio while alleviating the limitations of failed decompression and lack of guidance, we propose the multi-stage compression pipeline that breaks both the compressor and the decompressor into 2 stages while introducing an intermediate logical representation of text, as shown in Figure 2. In particular, instead of directly prompt the LLM to compress the text entirely, the intermediate stage intentionally allows LLMs to understand the document by arranging the scattered information into a more structured and logically coherent format. The logic representation, in theory, could be in any reasonable format that bears more structure than pure natural language, for example:

- A note-taking style similar to a lecture note which contains hierarchical bullet points with subsection titles and supporting details
- A program code style that could be similar to Python functions. It follows rigorous keywords and grammar
- A logic symbol style that mimics live presentation translator notes, which has to contain extremely concise logic symbols and acronyms as human translators need to record the note in a short time and translate based on the structure.

Dataset	Stage	CR	Similarity
NYT	After 1[2]	56.5%	0.791
	After 2	85.3%	0.826
	After gzip	88.5%	0.826
Yelp	After 1	51.2%	0.736
	After 2	87.2%	0.740
	After gzip	87.8%	0.740

Table 4: Compression and Decompression Results for the Multi-Stage Compression Pipeline

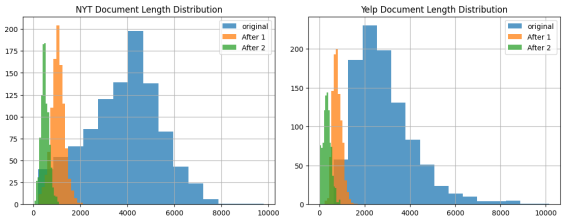

Figure 3: Distribution of text lengths after each stage of compression under the multi-stage pipeline.

In the experiments, we tested the first option by asking the language models to first compress into "concise, minimalistic notes, using abbreviations, symbols, and simple text-based diagrams to clearly represent essential points and key relationships". With the logic representation, the second stage (i.e. compressor 2) could potentially follow the logic flow to further compress into short text while preserving key words and keep words that are hard to abbreviate. In the decompression stage 2 (i.e. decompressor 2), the logic representation also allows the LLM to disentangle compressed information into the same logic structure before granting more freedom to generate natural language in stage 1 (i.e. decompressor 1). The entire pipeline utilizes the natural language understanding capabilities and guides the LLM compressor/decompressor in specific logic flows, ultimately retaining more semantic meaning.

In addition, as the LLMs are pre-trained on large amount of data and likely data under the same genre, we add the text genre (e.g. user reviews, news articles, etc.) to the prompts. This allows the LLMs to have clearer expectations of the logic flows usually baked into specific genres. For example, a news article will most likely contain formal languages and clear indication of the time and locations of an event before describing details about the event. All of the complete prompts used in the multi-stage pipeline could be found in Appendix A.2.

6.2 Results and Discussion

We present the compression and decompression results for the multi-stage pipeline in Table 4, and the distribution of text lengths after each stage in Figure 3. For a better analysis for each compression step, we also include the same metrics evaluated on the intermediate logic representations. From the results, we observe that:

1. The critical problem of decompression failure has been eliminated. We believe that the guidance given to LLMs does restrict the chances for them to freely use any symbols that ultimately leads to incomprehensible text

2. The multi-stage pipeline is able to reach much higher compression ratio while retaining about a similar amount of original information, if not more. It shows that the guidance does help LLMs to understand the texts in a correct and more structured way that leads to a more aggressive and efficient compression. Specifically, the experiments on both datasets show a significant gap comparing to baseline and lossless algorithms that users might consider worthy to sacrifice some meanings.

3. Comparing between the results after each stage, we see that arranging the original content into logic representation is indeed helping to extract a good amount of key information and makes proper preparation for the second stage. In both datasets, only about half of the original info is retained after this stage, and the second stage helps compress another 30%.

4. We could also observe from the figures that the distributions of text lengths are preserved relatively well by examining the shapes after compression. This confirms the consistency of the multi-stage pipeline as the stages help gradually reduce the size but selecting carefully on what information to retain.

In addition to the 2 compression stages, we also applied a lossless post-processing to investigate the remaining patterns in the text after LLM compression, also showed in Table 4. Since a lossless compression is 100% recoverable, a decompression stage only needs to add the gzip decompression step before using the same pipeline. The results show that after LLM compression, lossless algo-

[2]Abbreviated for After Compressor 1, similarly for below

Stage	NYT		Yelp
	Coarse (6)	Fine (26)	Cuisine (10)
Original Text	0.966	0.895	0.820
After 1	0.963	0.909	0.790
After 2	0.960	0.890	0.692

Table 5: Classification accuracy using texts compressed in different stages and direct prompting. The numbers in parenthesis means the number of possible classes the set of labels has.

rithms would only further compress the text by an insignificant amount, showing that while it's possible to gain a slightly better compression ratio using this post-processing step, multi-stage LLM compression has done a decent enough job to eliminate possible string patterns and only keeping key information.

Further analyzing the quality of compressed content, we show the classification accuracy done using texts compressed after different stages. Using the prompts presented in Appendix A.3, we directly ask the LLM to try reading the text and choose from one of the given class labels. Note while we emphasized in the prompt to only choose within the list, it would still be possible for LLMs to predict to other classes. In this situation, we simply consider the prediction to be incorrect.

As shown in Table 5, the classification performance on both the coarse-grained and fine-grained labels in the NYT dataset are great and consistent. It shows that such compression helps retain most key information needed for document topic understanding, and details are also retained as it could be used to predict into 26-class fine-grained labels. This further confirms that the compressed texts in either stage shows concentrated yet important information. On the other hand, we're observing a much harder task in cuisine classification based on user reviews. The most useful information for such task does not stem from the main summary for the review, but mostly from noisy dish names. The central ideas from the reviews are usually related more to emotions. We argue that while the logic representations after compressor 1 still maintain most dish information, a significant chunk of such details will be lost after compressor 2 if not using more careful prompting. Therefore, the performance for cuisine classification is expected to drop after stage 2.

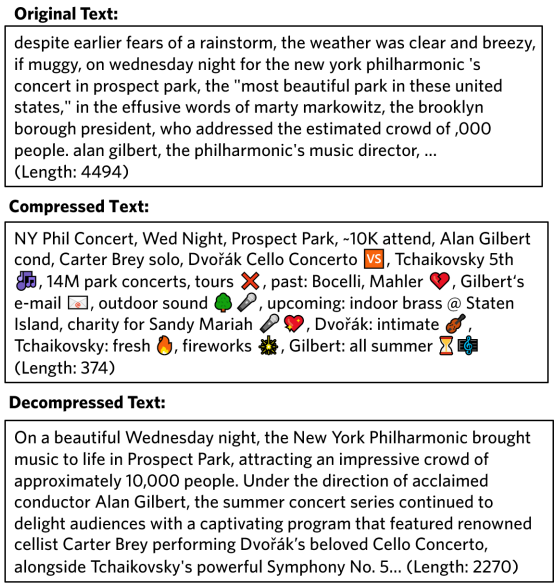

Figure 4: An example of compressed and decompressed text using multi-stage pipeline

6.3 Case Study

Figure 4 presents an example of compressed and decompressed text in the NYT dataset using the multi-stage pipeline. It shows that the compressed text is much more formatted compared to the baseline compression and intentionally maintained special names, and the decompressed text keeps most of the information in the starting paragraph, including the time, location, and performed music pieces for the music event. The example yields a compression ratio of 91.67% and a decompressed SBERT similarity of 0.824.

7 Limitations

From the discussions above, although the current multi-stage model is capable of compressing text into a reasonably minimal size and keeping most of the key information, there are plenty of drawbacks that this model failed to address:

- **Variety of LLMs**: While the study shows great capability of LLM compression, prompting black-box models such as `gpt-4o-mini` still requires monetary cost and access to only the generated text. It's highly possible to apply the same framework to open-source LLMs to reduce such cost while investigating the compression capabilities with regards to different model size and families. We're unable to conduct such experiments because of the computational resource limitation.

- **Further Compression**: The case study above shows that while we've already compressed the text by a significant compression ratio, further processing could be done by either eliminating more spaces or making the text more compact;

- **Preprocessing**: Since our semantic compression setting does not require a full text recovery, there leaves plenty of space to conduct text preprocessing before sending it to compression stages. For example, it's possible to remove stopwords or detect tightly connected phrases to reduce the information beforehand. The compression method presented in (Fei et al., 2024) also could be seen as preprocessing and could be possibly useful for better compression. In this case, there does not need to add another decompression stage as LLMs are capable of generating coherent text.

- **User-Guided Compression**: We could observe from the degraded cuisine classification accuracy that LLMs are more capable of keeping main information using a general prompt. It's possible for users to provide more guidance on specific information they wish to keep, thus leading to a more versatile pipeline.

- **Model Probing**: The proposed pipeline in this study directly utilizes LLM input and outputs, and still only has limited control over the LLM output. Potentially on open-source LLMs, we might finetune a model for a specific dataset to probe the softmax probabilities of next token prediction, and extract high probabilities for additional compression. This stems from the fact that with a high softmax probability, the model will most likely generate that token, thus there it's less necessary to directly store that token for semantic purposes.

8 Conclusions

In conclusion, this study investigates the wide applications of lossy text compression directly utilizing the natural language understanding capabilities of Large Language Models. Experiments show that by guiding LLMs through a structured intermediate step, the pipeline could achieve high compression rate and relatively satisfactory decompressed similarities. However, the work could be further improved by: (1) confirming the robustness of our pipeline by benchmarking on more LLMs, datasets, and tasks; (2) applying NLP algorithms as preprocessing or model probing to construct a better understanding of input documents before potentially reaching better compression efficiency.

References

M. Burrows and D.J. Wheeler. 1994. *A Block-sorting Lossless Data Compression Algorithm*. Digital SRC. Digital, Systems Research Center.

Paul Deutsch. 1996. Deflate compressed data format specification version 1.3. https://www.rfc-editor.org/rfc/rfc1951.html. RFC 1951.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. 2024. Extending context window of large language models via semantic compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5169–5181, Bangkok, Thailand. Association for Computational Linguistics.

Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 2024. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*.

Henry Gilbert, Michael Sandborn, Douglas C. Schmidt, Jesse Spencer-Smith, and Jules White. 2023. Semantic compression with large language models. In *2023 Tenth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 1–8.

David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Julian Seward. 1998. The bzip2 and libbzip2 manual. http://www.bzip.org/.

J. Ziv and A. Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343.

A Prompting Strategy

In this section we present the exact prompt we used for our experiments.

A.1 Baseline Prompts

The simple baseline prompts are showed as follows. In this baseline, we didn't discriminate between system or action prompts. We placed all context within the action prompt.

Compression Prompt

```
We want you to produce a compressed text that
fits a Tweet, such that you (GPT-4o) can
reconstruct it as close as possible to the
original. For each number, generate a single
token in the compressed text. This is for
yourself. Do not make it human-readable. Abuse
of language, mixing, abbreviations, and symbols
(Unicode and emojis) to aggressively compress
it are all encouraged. Make sure to keep ALL the
information to reconstruct it fully.

#### Text to Compress:

[TEXT]

Just give me the compressed text directly. Do
*not* provide any additional commentary at any
point in your response.
```

Decompression Prompt

```
I asked you to compress a long text using your
own abbreviations. You replied with:

[TEXT]

Reconstruct the original text. Just give me the
reconstructed text directly. Do *not* provide
any additional commentary at any point in your
response.
```

A.2 Multi-Stage Compression Prompts

In this section, we shared the detailed prompts for the multi-stage compression pipeline. The special tokens `[TEXT_GENRE]` will be replaced to represent the genre that we provide to the LLMs.

Compress-To-Formatted (Compressor 1) Prompt

```
### System Prompt:
You are an advanced AI tasked with interpreting
and summarizing complex [TEXT_GENRE] in a
symbolized, condensed, systematic format as the
first step in a two-stage text compression
process. Your goal in this step is to reduce the
original [TEXT_GENRE] into concise,
minimalistic notes, using abbreviations,
symbols, and simple text-based diagrams to
clearly represent essential points and key
relationships. Focus on preserving the main
ideas with symbols and arrows where useful.
Ensure that:
- The output could be further compressed by
  another GPT-4 model to an even shorter form.
- Another GPT-4 model could also expand your
  output back to closely resemble the original
  [TEXT_GENRE].

### Action Prompt:
Interpret the following [TEXT_GENRE] and
provide only the interpreted version. Create a
compact, readable summary that maintains the
core meaning. Only output the compressed text
and no extra explanations.

#### Text to Interpret:
[TEXT]
```

Compress-To-Short (Compressor 2) Prompt

```
### System Prompt:
You are an advanced AI tasked with aggressively
compressing logically organized [TEXT_GENRE] to
its minimal length as the second step in a two-
stage text compression process. Your goal in
this step is to compress the organized
[TEXT_GENRE] as much as possible, using any
method -- abbreviations, symbols, mixed
language, Unicode, or emojis. You doesn't need
to format the output nicely since it doesn't
need to be human-readable at all. It's for
yourself. As long as the information could be
reconstructed, make the output as short as
possible. Ensure that:
- Another GPT-4 model could reconstruct the
  compressed [TEXT_GENRE] to a semantically
  close version of the original text before
  logical organization.
- Achieves shortest compressed output as
  possible.

### Action Prompt:
ompress the following organized [TEXT_GENRE]
and return only the compressed version. Don't
format the output; doesn't need to be human
readable; be as short as possible. Only output
the compressed text and no extra explanations.

#### Text to Compress:
[TEXT]
```

Decompress-To-Formatted (Decompressor 2)

```
### System Prompt:
You are an advanced AI tasked with interpreting
highly compressed [TEXT_GENRE] that contains
symbols, abbreviations, and mixed language by
another GPT-4 model, and expanding it into
logically organized meeting-style notes. The
original content is a coherent [TEXT_GENRE],
so your reconstruction should follow a similar
flow. Aim to create a notes-style format with
minimal structure—small headers or titles with
```

```
supporting details in text flow, as if quickly
jotting down main points and relevant details.
Your goal is to produce an expanded format
that:
- Preserves enough detail for further expansion
  into a full, human-readable [TEXT_GENRE].
- Prioritizes a logical flow and clear
  organization while avoiding overly
  structured, sectioned formats like "Overview,
  Background, Risks, etc." Stick to an informal
  flow for easier reading and interpretation.

### Action Prompt:
Decompress the following text by reconstructing
it into logically organized, meeting-style
notes that maintain the flow, key points, and
essence of the original [TEXT_GENRE].
Prioritize clarity and coherence. Only output
the decompressed notes, without additional
explanations or indication that it's a note.

#### Compressed Text to Decompress:
[TEXT]
```

Decompress-To-Text (Decompressor 1)

```
### System Prompt:
You are an advanced AI tasked with reversing a
symbolized, condensed, systematic format into
a full, human-readable [TEXT_GENRE]. The input
is a set of notes-style format with minimal
structure, as if quickly jotting down main
points and relevant details. Your goal in
this step is to expand the compressed
information into a coherent [TEXT_GENRE],
while preserving the original logic and flow.
Ensure that:
- The output is a natural language [TEXT_GENRE]
  that reflects the uncompressed content
- The reconstructed article maintains clarity,
  coherence, and logical progression as it
  would appear in a typical [TEXT_GENRE].
- There should be no note-style elements such
  as little headers and bullet points.

### Action Prompt:
Decompress the following text by expanding it
into a full, human-readable [TEXT_GENRE].
The expanded article should closely resemble
the original content, written in a natural
style suitable for a [TEXT_GENRE] with clear
and coherent ideas. Only output the
decompressed text.

#### Compressed Text to Decompress:
[TEXT]
```

A.3 Classification Prompts

Topic Classification Prompt

```
Given the following [TEXT_GENRE]:

[TEXT]

You must classify it into one of the following
classes: [POTENTIAL_CLASSES]. You must respond
with one of the classes given, even if you feel
none is suitable. Only output the predicted
class name in lower case. Do not include any
explanations in any situation.
```

Cuisine Classification Prompt

```
Given the following restaurant reviews:

[TEXT]

You must classify the store the user reviews
into one of the following cuisines:
[POTENTIAL_CLASSES] You must respond with one
of the classes given, even if you feel none
is suitable. Only output the predicted class
name in lower case. Do not include any
explanations in any situation.
```