Semantic Compression With Large Language Models

Henry Gilbert, Michael Sandborn, Douglas C. Schmidt, Jesse Spencer-Smith, and Jules White
Dept. of Computer Science
Vanderbilt University
Nashville, TN, USA
{henry.gilbert, michael.sandborn, douglas.schmidt, jesse.spencer-smith, jules.white}@vanderbilt.edu

***Abstract*—The rise of large language models (LLMs) is revolutionizing information retrieval, question answering, summarization, and code generation tasks. However, in addition to confidently presenting factually inaccurate information at times (known as "hallucinations"), LLMs are also inherently limited by the number of input and output tokens that can be processed at once, making them potentially less effective on tasks that require processing a large set or continuous stream of information. A common approach to reducing the size of data is through lossless or lossy compression. Yet, in some cases it may not be strictly necessary to perfectly recover every detail from the original data, as long as a requisite level of semantic precision or intent is conveyed.**

This paper presents three contributions to research on LLMs. First, we present the results from experiments exploring the viability of "approximate compression" using LLMs, focusing specifically on GPT-3.5 and GPT-4 via ChatGPT interfaces. Second, we investigate and quantify the capability of LLMs to compress text. Third, we present two novel metrics—Exact Reconstructive Effectiveness (ERE) and Semantic Reconstruction Effectiveness (SRE)—that quantify the level of preserved intent between text compressed and decompressed by the LLMs we studied. Our initial results indicate that GPT-4 can effectively compress and reconstruct text while preserving the semantic essence of the original text, providing a path to leverage more tokens than current limits allow.

***Index Terms*—large language models, prompt engineering, data compression, code generation**

I. Introduction

Emerging trends and challenges. Large language models (LLMs) have garnered significant attention with the recent release of OpenAI's ChatGPT [1], Google's Bard [2], and others [3]–[5]. These LLMs facilitate (potentially incorrect) information retrieval, whether in the form of concept clarification, question answering, text editing, code generation, summarization, or task planning. LLMs typically provide users with an interactive chat interface to engage in back and forth conversation about a concept, task, or goal. These topics can be referenced over time within a conversation that preserves the discussion context over some time horizon (e.g., number of prompts or total tokens input in the session).

A key to using LLMs effectively is the quality of the input *prompt*, which is the text provided as input to the LLM. In conventional LLMs, the prompt is used in an opaque manner by the LLM to produce an output, called a *response*, which is based on a non-linear function of the input prompt and the model's weights (the latter are opaque to users). Given some target task or output structural characteristics, the process of identifying high-quality prompts to feed an LLM is known as *prompt engineering* [6].

LLMs can be prompted with questions or directives, such as "who was the first president of the United States?" or "provide a vegetarian dinner recipe that takes 20 minutes to prepare," and they often generate high-quality answers. A key limitation of conventional LLMs, however, is that they are trained on knowledge with a cut-off date.[1] To reason or answer questions about newer information that an LLM was not trained on, the new information must be included in the prompt for the LLM. For example, a prompt could start with the text from a recent news article and then the LLM could be prompted with a question related to the information contained earlier in the prompt. The training cutoff increases the burden on the user to provide a sufficient context to the LLM about the nature of the desired output.

Challenge: LLMs have a maximum input size, which restricts the amount of information that can be put into a prompt. The maximum input size of an LLM is typically measured in tokens [7], [8] and corresponds to how much information can be input into the LLM at once. A token is a particular grouping of letters within a word. *Tokenization* refers to how a given text is divided from letters and spaces into groups as a pre-processing step for an LLM, and several tokenization methods exist. This input token cap determines the maximum number of words or symbols that can be included in a prompt provided by a user.

GPT-3.5, which was the model underlying an earlier release of ChatGPT, has an input token limit of 4,096 tokens, or ∼3,000 words [7]. The more recently released GPT-4 model has an increased input token limit of 32,768 tokens, or ∼24,000 words. It is likely, however, that LLMs will eventually operate on streams of data (e.g., daily customer activity or meeting transcripts) with much higher token counts. Moreover, while LLMs such as ChatGPT-4 can handle a conversation history, the same underlying token limit is applied. As such, the context the model is able to reason over is truncated by the internal token limit, regardless of prompt segmentation.

In many cases, the maximum input size is a significant limitation to the use of LLMs. For example, the source code for a large software application can often exceed the maximum input size to the LLM.

Various approaches exist to overcome input size limitations, including using the LLM to summarize information that should

[1]ChatGPT's cut-off was September, 2021.

be included in a prompt to shorten it. Other approaches range from (1) using semantic search to (2) only select contextually relevant information to include in a prompt to (3) structuring software into isolated components with clear interfaces that are easy to reason about using smaller segments of source code. Until the input size of LLMs becomes large enough to avoid becoming a concern in practice, however, determining how to use the limited space available in a prompt most efficiently remains an open research challenge.

Motivated by these limitations, we examine the viability of prompting LLMs to produce compressed responses that still preserve rich semantic information, so that the original information can be recalled sufficiently and the original intent is preserved. Potential benefits of LLM compression capabilities include source code manipulation, text retrieval, information distillation.

This paper compares the compression rates, semantic similarity, and reconstruction and recall quality of GPT-3.5 and GPT-4 for compressed and uncompressed text between different ChatGPT conversations. We restrict the compression format to characters only (i.e., no emojis or Unicode characters). We evaluate the compression capabilities of these models against a standard compression algorithm, Zlib's Deflate algorithm [9] using the compression ratio, edit distance, and our novel Exact and Semantic Reconstruction Effectiveness metrics. We use the cosine similarity between embeddings[2] of compressed and decompressed text to quantify the quality of LLM compression. This work offers the following contributions to research on prompt engineering:

- An initial exploration of LLM-based compression in GPT-3.5 and GPT-4, evaluated with two novel metrics: Semantic Reconstruction Effectiveness (SRE) and Exact Reconstruction Effectiveness (ERE)
- Experiments examining LLM compression of fictional short story excerpts
- Open source code of experiments, available at https://github.com/henrygilbert22/phd_chatgpt/tree/main/compression_analysis

The remainder of this paper is organized as follows: Section II provides background material that motivates our focus on LLM-based compression; Section III presents LLM compression experiments on excerpts from literary short story text; Section IV discusses prompt engineering approaches taken to facilitate compression; Section V compares our approach with related work on evaluating LLMs and data compression; and Section VI presents concluding remarks.

II. Background on Compression vs. Embeddings

Large language models (LLMs) are revolutionizing the field of natural language processing (NLP) by enabling more efficient and effective information storage and retrieval. A key research topic addressed by this paper is how well LLMs can perform their own prompt compression and manipulate compressed prompts, which enables the extraction and processing of information from shorter, condensed inputs while also maintaining semantic value and intent. In essence, an LLM is trained to iteratively predict the next word in its responses, which is intialized as a continuation of the input prompt. With this in mind, a smaller prompt size that still elicits desired behavior is valuable for tasks that require an extensive context about input data. For example, editing a large manuscript or reasoning about a large codebase may require that the prompt (e.g., "refactor this code to be compatible with this component") accompany a context that exceeds the allowed number of input tokens. This application scenario motivates our investigation of LLM capabilities in reducing the size of input text while also retaining semantic value.

We begin by highlighting the differences between embeddings and compression as they relate to the capabilites of LLMs. *Embeddings* [10] provide a one-way mapping for representing a high-dimensional, possibly sparse feature vector into a relatively low-dimensional space that still captures the semantics of the input. A common application of embeddings is to represent words of text [11] or source code [12].

Embeddings are typically obtained by extracting the weights of a trained neural network layer that is close to the output layer, but before a classification layer (such as softmax [13]) is applied. Embeddings are inherently non-invertible[3], owing to the information lost when reducing the dimensionality of the original input. In return, however, they offer the ability to quantify similarity of inputs in the embedding space, typically with a vector-based metric, such as cosine similarity.

In contrast, compression is concerned purely with minimizing the size of the data while also preserving the integrity of the information that is compressed. Depending on the algorithm used, compression may achieve perfect or near-perfect reconstruction of the compressed data into its original form. There are two main methods of compression: lossless and lossy [14]. Lossless compression removes only extraneous metadata, while lossy compression removes some of the original data. For example, PNG images use lossless compression, whereas JPEG images use lossy compression. Lossy compression usually optimizes the information lost so that the decrease in quality is perceptually minimal to humans and is therefore tenable in most cases.

To summarize, embeddings focus on transforming input into a different magnitude of dimensionality to facilitate comparison that is not easily achieved in the input space (e.g., semantic similarity of words or sentences). Conversely, compression focuses on minimizing the size of the input data so it can be reconstructed with original or near original fidelity. In other words, embeddings facilitate the quantitative comparison of data with unwieldy dimensionality, whereas compression minimizes the storage footprint of a piece of data while preserving as much of the original information as possible.

We focus primarily on GPT-3.5 and GPT-4, which are

[2]In this context, an embedding is a real-valued vector of reduced dimensionality derived from the input text data.

[3]Non-invertible means that the original text cannot be recovered from the embedding of the text.

TABLE I
TEXT EXCERPT IDENTIFIERS FOR THE FICTIONAL SHORT STORIES STUDIED

Text ID	Text Name	Author
a	A Good Man is Hard to Find	Flannery O'Connor
b	Break It Down	Lydia Davis
c	Cat Person	Kristen Roupenian
d	Cathedral	Raymond Carver
e	Flowers for Algernon	Daniel Keyes
f	Sticks	Karl Edward Wagner
g	Symbols and Signs	Vladimir Nabokov
h	The Bogey Beast	Annie Flora Steele
i	The Lottery	Shirley Jackson
j	The Veldt	Ray Bradbury

the two models provided by OpenAI in a browser-based chat interface (i.e., "ChatGPT") with a ChatGPT Plus subscription. We leverage the API endpoints for GPT-3.5 and GPT-4 where applicable, and otherwise produce prompts and responses for GPT-4 directly in the chat interface. The prompt text and results reside in our experiment notebooks available at https://github.com/henrygilbert22/phd_chatgpt/tree/main/compression_analysis.

III. ANALYZING LLM COMPRESSION PERFORMANCE ON LITERARY TEXT

This section provides initial results of experiments conducted to evaluate limitations of LLM compression. We use entropy, compression ratio, and edit distance to compare LLM compressed text to the baseline of Zlib's Deflate compression algorithm. Table I indicates the subject texts and their identifiers used in figures.

A. Experiment: Compression of Fictional Literary Text

We first compare the compression rate and reconstruction loss of GPT-4 with a standard compression method (Zlib's Deflate compression algorithm) to evaluate how well GPT-4 performs with respect to compressing textual information effectively while simultaneously retaining semantic information. The evaluation set is a collection of 10 excerpts from short story texts comprising a variety of genres and writing styles.

To initiate the compression process, we asked GPT-4 to generate a prompt that would facilitate text compression. By requesting that GPT-4 create its own prompt to facilitate compression and recall, we aim to mitigate human bias and provided a standard prompt for our experiments. In response to this request, GPT-4 generated the following prompt for compression:

> *Compress the following text into the smallest possible character representation. The resulting compressed text does not need to be human readable and only needs to be able to be reconstructed with a different GPT-4 model.*

In a similar manner, we prompted GPT-4 to generate a prompt for text decompression. This prompt ensured that a new, independent GPT-4 model instance would decompress the compressed input text effectively. In response to this prompt, GPT-4 generated the following prompt for decompression tasks:

TABLE II
GPT-4 COMPRESSED TEXT VS. ZLIB DEFLATE BY ENTROPY AND COMPRESSION RATIO (CR)

Method	Avg Entropy	Avg CR
GPT-4	0.933	0.825
Zlib Most	0.837	0.469
Zlib Least	0.838	0.453

> *Please decompress the following compressed text into its original form, as it was provided by a user and compressed by another GPT-4 model.*

A separate instance of GPT-4 independent from the original model (i.e., in a separate chat conversation), was used to decompress the compressed text. Prompt engineering tactics on compression quality are discussed further in Section IV.

To establish a baseline for comparison, Python's internal Zlib library [15] using the Deflate and Inflate algorithms was used for both compression and decompression to assess existing lossless compression methods. We compressed each short story excerpt twice, first with minimal compression (fastest) and second with maximal compression (slowest) by passing the `level=1` and `level=9`, respectively. These results provide a basis to compare the effectiveness of GPT-4's compression capabilities.

B. Analysis: Entropy

To better understand the measure of randomness in the distribution of characters compressed by the LLM, we calculate the frequency of each character in the compressed text and then the entropy of the distribution of compressed characters. Each compressed text was first converted to a byte stream representation before computing the distribution of its characters. The character distribution for a given text is given by:

$$P(x) = \frac{n_x}{N}, \tag{1}$$

where $P(x)$ is the probability of character x, n_x is the number of occurrences of character x, and N is the total number of characters in the byte stream representation. The entropy was then computed for each byte character distribution using the entropy equation [16]:

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x), \tag{2}$$

where $H(X)$ is the entropy of the character distribution of a compressed text excerpt.

As anticipated, ChatGPT-4 consistently results in the highest entropy of its compressed text. In contrast, Zlib's most compression method generates slightly higher entropy than the least compression method. Given the small sample size of text excerpts and the small differences between these values, we

cannot conclude that this trend holds with statistical significance. Table II displays the averaged compression entropy and compression ratio for GPT-4 and the Zlib baselines for the texts studied.

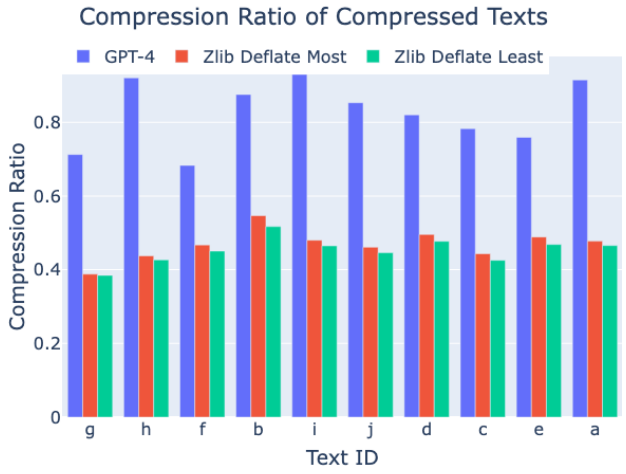
C. Analysis: Compression Ratio

To better understand the degree to which an input text was reduced in size, we compute the Compression Ratio (CR) between the original and compressed texts using the following equation:

$$CR = 1 - \frac{\text{\# compressed bytes}}{\text{\# original bytes}} \tag{3}$$

where CR is the compression ratio. For example, a compression ratio of 0.8 means that the original text size was reduced by 80% in its compressed form, or is 20% of the original text's size. Figure 1 shows the relative compression ratio for each method across all texts. Clearly, GPT-4

Fig. 1. Compression Ratio By Text



provides higher compression ratios for the all of the text excerpts studied compared to the baseline methods. Zlib's most (`level=9`) aggressive and slowest compression method narrowly outperforms Zlib's least (`level=1`) aggressive and fastest compression method.

As with Figure 1, Zlib's maximal compression remains marginally better than Zlib's minimal compression. GPT-4 continues to outperform both baseline methods with a near 60% increase in compression performance. While this result may appear noteworthy, our subsequent analyses in Section IV reveal that GPT-4 achieved this high degree of compression rate by discarding key information in the original text.

D. Analysis: Edit Distance

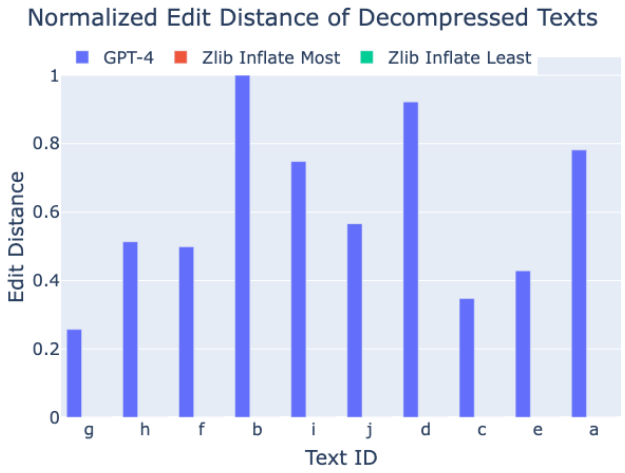
To better understand the exact closeness of the reconstructed text in relation to the originally compressed text, we use the Levenshtein edit distance metric [17]:

$$D(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ and } j = 0, \\ i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \text{else} & \\ \quad \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+(1-\delta(s_i,t_j)) \end{cases} \end{cases} \tag{4}$$

where $D(i,j)$ is the edit distance (number of characters that must be changed) between the original text of length i and the reconstructed text of length j, s_i and t_j are the characters at position i and j, respectively, and $\delta(x,y)$ is the Kronecker delta function which returns whether the characters at 2 possibly different indices are identical. All edit distances are normalized between 0 and 1, as the exact quantitative result is arbitrary in our case, and we only care about the relational analysis between methods.

Figure 2 shows the edit distances for each compression method over all texts. The Zlib baselines are lossless com-

Fig. 2. Edit Distance By Text



pression algorithms, so their edit distances are 0, as expected, and consequently not shown on the graph. From this figure we observe that GPT-4 compression is rather lossy.

Figure 2 also showcases the variance of exact reconstruction performance. In particular, GPT-4's performance varies greatly on different text excerpts. This edit distance variance in GPT-4's compression quality raises questions for future studies, such as what character distributions and captured features of input text influence a higher or lower edit distance in the compressed text. The reasons behind this variance could be due to inherent characteristics of the input text, such as language structure, semantic complexity, or specific dialect and choice of language. This variance requires further investigation into the factors influencing GPT-4's performance on text compression and decompression tasks.

E. Analysis: Semantic Retention Quantified by Cosine Similarity

Based on previous results, GPT-4 cannot currently be used as a reliable compression technique since it does not rival existing lossless methods based on the edit distance metric. This finding indicates that information is lost between compression and decompression when input text is passed to the LLM to reduce its size. However, we nevertheless want to explore the ability of an LLM to capture the *underlying semantic intent* of the original text in an approximately reconstructable manner.

We are not concerned whether the decompressed text exactly matches the original, as long as it retains the essence of what is originally intended to be communicated. For example, if the original message is: "Please send me an email on Monday", and the reconstructed message is: "On Monday, send me an email", then the resulting semantic similarity score should be relatively high as the underlying meaning of "send an email Monday" is represented in both messages, even though the character occurrence and alignment do not closely match.

To quantify similarity of compressed and decompressed texts, we use OpenAI's Embeddings API [18], and then apply the cosine similarity vector metric to pairs of embedded texts. Section I explained how embeddings are vectors that represent (1) the semantic information in the text learned by an LLM during its training and (2) a high-dimensional, possibly sparse vector in a low-dimensional representation. A text embedding captures a notion of distance in the embedding space, enabling similarity comparisons using vector-based metrics. The commonly used metric for embedding comparison is cosine similarity [19], which is calculated as follows:

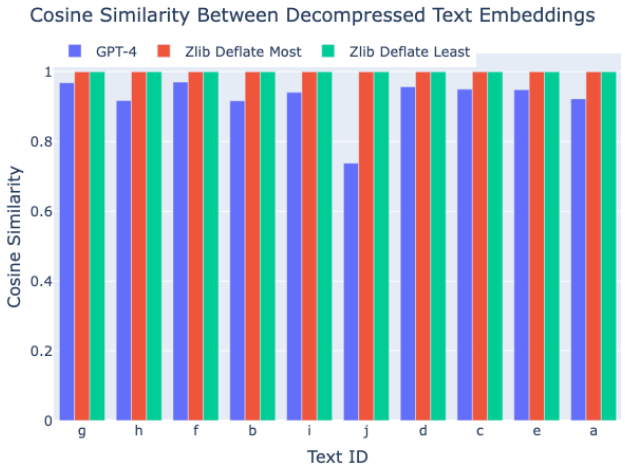
$$\text{Cosine Similarity}(\mathbf{A}, \mathbf{B}) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|} \tag{5}$$

Where $\mathbf{A}$ and $\mathbf{B}$ are n-dimensional vectors, $\mathbf{A} \cdot \mathbf{B}$ is the dot product between them, and $|\mathbf{A}||\mathbf{B}|$ represents the product of their magnitudes. The result of this operation is the angle between these vectors in the embedding space, with values ranging from -1 (indicating opposite vectors) to +1 (indicating proportional vectors). Zero indicates that the vectors are orthogonal. The angle between these vectors in the embedding space can be obtained with $\theta = \arccos(x)$ where x is the output of Equation 5 above.

Figure 3 shows the computed cosine similarity between embedding vectors across all texts for each compression method. Again, since Zlib's method is lossless it always captures the semantic meaning since the original input is reconstructed perfectly. GPT-4 does not perfectly preserve the semantic meaning of decompressed texts from the original, but performs relatively well across all texts, with an average angle between embedding vectors of $\arccos(0.923) \approx 22.6°$.

Interestingly, GPT-4's semantic reconstruction performance is consistent across texts, which is a stark contrast to GPT-4's high volatility in the edit distance metric. From these results, we conclude that GPT-4 may not be suitable for near lossless compression. However, it remains a compelling method to preserve semantic similarity in compressed and decompressed representations, based on the embedding methods we use.

Fig. 3. Cosine Similarity Between Embeddings of Decompressed Text



Although we can't quantify directly how this angular distance relates to the preservation of underlying semantical meaning in the texts, we can compare the relative performance to the relative performance in edit distance. By taking the relative difference between each model's cosine similarity score, ChatGPT-4 performed, on average, approximately 23% worse than Zlib's lossless methods for maintaining text semantics. However, when taking the relative difference between edit distances, ChatGPT-4 performed, on average, approximately 33% worse than Zlib's lossless methods for maintaining text character positions. This shows that ChatGPT-4 is relatively better at capturing semantical meaning when compared to maintaining exact text reconstruction.

To summarize, while GPT-4 is not a suitable replacement for traditional lossless compression, it demonstrates potential in preserving the semantic meaning of text. The next section explores the results of experiments we conducted to evaluate the influence of prompt structuring on compression quality.

IV. Prompt Engineering to Facilitate Compression Behaviour

This section present the results of our investigation into the role prompt engineering plays in terms of prompt content and wording in facilitating the compression performance of LLMs, specifically for GPT-4 and GPT-3.5. We examine the impact of three different meta-prompts for compression: *Base Compression* (simply direct to compress the input), *Guided Lossless Compression* (by specifying lossless compression of the input), and *Semantic Compression* (prioritizing semantic recovery).

We found that the choice of meta-prompt influenced the compression behavior of the LLMs studied. To evaluate the effectiveness of compression in relation to edit distance and semantic similarity, we introduced two novel metrics, *Exact Reconstruction Effectiveness* (ERE) and *Semantic Reconstruc-*

tion Effectiveness (SRE), respectively. The Exact Reconstruction Effectiveness metric revealed that while the Zlib Deflate lossless compression baselines outperformed GPT-4 and GPT-3.5, our meta-prompts for Guided Lossless Compression method outperformed both Base Compression and Semantic Compression in terms of compression ratio and edit distance minimization.

The Semantic Reconstruction Effectiveness metric, in contrast, demonstrated that the Semantic Compression meta-prompting approach outperformed the baseline lossless compression. Although the baseline lossless methods achieved slightly higher semantic similarity scores, the GPT-4 Semantic Compression model provided an improved compression ratio while preserving functionally equivalent semantic information in the input. This finding suggests that LLM-based Semantic Compression could offer considerable performance and cost gains over traditional compression methods in use cases where the exact reconstruction of the input is not crucial, as long as the underlying meaning remains intact.

A. Experiment Setup

The aim of this experiment was to distinguish LLM compression behavior and performance when optimized for lossless compression versus semantic compression. We applied the results from Section III to represent the baseline GPT-4 model performance when given no additional specification on compression requirements. The same analysis was therefore performed as outlined in Section III, but with two separate compression models distinguished by the meta-prompting performed to guide compression behavior.

GPT-3.5 was also given the same set of prompts over the same text excerpts to compare the compression quality between the different model versions. When requesting lossless compression, each model was fed the following prompt:

> *Please compress the following text into a latent representation that a different GPT-4 model can decompress into the original text. The compression model should be lossless, meaning that a different GPT-4 model should be able to perfectly reconstruct the original text from the compressed representation, without any additional context or information.*

The aim of this prompt was to instruct the model to prioritize lossless compression, thereby ensuring that the decompressed text was as close as possible if not identical to the original input. Similar approaches were used to formulate the decompression and semantic compression prompts, but these are omitted for brevity and can be found in our code.

GPT-4 was not available programmatically via an API endpoint at the time of this experiment. All prompts and results were therefore obtained manually using the chat interface feature on the ChatGPT website (`chat.openai.com`). For results from the GPT-3.5 model, the Chat Completion function of the `openai` Python library was used. This library requires providing the model with a "system" prompt to prime the model with expected behavior, as well as the actual chat prompt. For GPT-3.5, the following prompt is used when optimizing for lossless compression and similarly for decompression:

System Prompt:

> *You are a ChatGPT LLM trained by OpenAI to compress text. The compressed text should be able to be decompressed by a different ChatGPT LLM model into the original text. The compression must be lossless, meaning that a different ChatGPT LLM model should be able to perfectly reconstruct the original text from the compressed representation, without any additional context or information. The compressed text does not need to be human-readable, only decompressible by a different ChatGPT LLM model.*

Action prompt:

> *Compress the following text. Return only the compressed text with no additional text. Text to compress: ...*

The lossless compression prompts instructed the model to focus on preserving the original text in its entirety, allowing for perfect reconstruction. The lossless compression prompts attempted to accomplish a high-fidelity preservation of the original text, ensuring that no information was lost during the compression and decompression process. The semantic compression prompts instruct the model to prioritize semantic preservation while minimizing the character count. The semantic compression prompts attempted to accomplish a balance between reducing the text size and maintaining semantic integrity.

B. Compressed Text Entropy Analysis

Using Equation 1 and Equation 2, the entropy of the compressed text was computed for each method across all texts. To calculate entropy, the text was first converted into a byte stream representation and the relative probability of characters was then computed.

The resulting entropy metrics are fairly similar across all models and texts, with a notable exception being the lossless compression through GPT-3.5. The second column of Table III gives the averaged compressed text entropy by model.

TABLE III
AVERAGE EFFICACY METRICS OF COMPRESSED TEXT

Method	Entropy	CR	ED
Base (GPT-4)	**0.791**	**0.825**	0.510
Lossless (GPT-4)	0.758	0.423	**0.194**
Lossless (GPT-3.5)	0.755	0.383	0.573
Semantic (GPT-4)	0.741	0.772	0.556
Semantic (GPT-3.5)	0.737	0.768	0.556
Zlib Deflate Least	0.711	0.453	0
Zlib Deflate Most	0.710	0.469	0

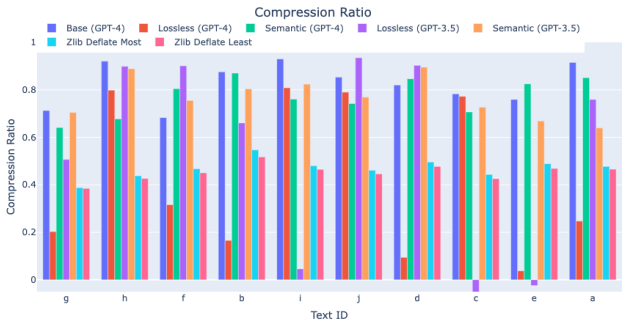
Interestingly, the base compression method, where GPT-4 was not explicitly prompted on expected compression behavior, demonstrates the highest entropy. This result was surprising given that ChatGPT-3.5 (which is not specifically

designed for this task) performs comparably well. The lossless compression maintains the lowest compressed text entropy, suggesting the efficacy of meta-prompting different models to achieve the desired outcome.

C. Compression Ratio

Similar to Section III, the compression ratio for each of the candidate compression methods was computed across all texts. Figure 4 plots the derived compression ratio of each method over each text. The graph shows high volatility in model performance across texts and between the models themselves. Interestingly, Lossless GPT-3.5 actually produces a compressed text that is approximately 71% larger than the original text for text c. Confoundingly, this text's edit distance from the original input text is the worst, which is another indicator that GPT-3.5 struggles to compress input text, highlighting the limitations of this particular model for compression tasks.

Fig. 4. Compression Ratio By Text



The third column of Table III includes the averaged compression ratio (CR) of each model across all texts. GPT-4 semantic compression maintains the best compression ability, closely followed by the base compression and GPT-3.5's semantic compression. This result was expected since semantic compression is not constrained by reconstructing the exact text and presumably the underlying semantic meaning can be captured in much less text when it can be arbitrarily decompressed. GPT-3.5's competitive semantic compression performance is surprising given its lossless compression performance.

The traditional lossless methods sit in the middle of the pack in performance, followed by both GPT lossless methods. The worse compression rates coming from the LLM lossless compression methods is expected and validates the impact of the meta-prompt. Clearly, the models must maintain a greater information density if the intended goal is exact reconstruction.

From these results, we conclude that LLMs like GPT-4 can achieve competitive compression rates when given appropriate meta-prompts, particularly in the case of semantic compression. However, further research is required to corroborate these results on a larger set of data.

D. Edit Distance

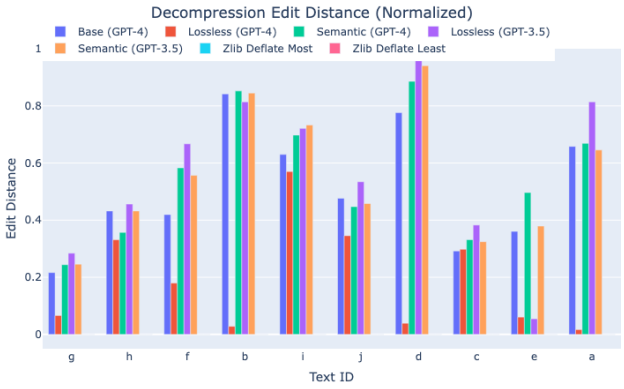
As discussed in Section III the edit distance for each compression method over each text was computed. This metric shows the model's ability to exactly reconstruct the original text from the compressed representation, based on the number of characters that must be inserted or deleted to achieve the original input text from the compressed representation of the text output by the LLM.

The fourth column of Table III shows the corresponding edit distance for each model over each evaluation text. While the magnitude of edit distance varies greatly between texts, the relative model performance is mostly stable, which suggests that the LLM models studied struggle on similar kinds of text. Future research should explore the semantic and syntactic classifications of texts that are "easier" versus "harder" for LLM models to compress effectively. Understanding these relationships enables targeted improvements in model performance and better adaptation to specific use cases.

The most notable model performance deviations are found in the GPT-4 lossless compression. It routinely performs the best of any LLM model, and on texts b, d, e, and a, the model performs exceptionally well with a normalized edit distance near 0. This result demonstrates that GPT-4 lossless compression can effectively reconstruct text with high accuracy in some cases. We therefore conclude that this model shows promising potential for text compression tasks.

Figure 5 averages the model edit distance over all texts. As

Fig. 5. Normalized Decompression Edit Distance by Text



expected, the traditional Zlib lossless compression methods maintain a distance of 0 as they losslessly reconstruct the text. GPT-4 performs better than all remaining models, with over ≈50% more accurate on average than the next closest LLM, GPT-3.5 Semantic Compression.

Interestingly, GPT-3.5 Semantic Compression and GPT-4 Semantic Compression are roughly split across the texts. Given the other performance differences between the models, this result is unexpected, but is likely justified by the limited evaluation data set.

GPT-3.5 Lossless compression clearly underperforms on this evaluation metric. This result aligns with expectations given its subpar performance in both the compression ratio

TABLE IV
AVERAGE QUALITY METRICS OF DECOMPRESSED TEXT

Method	CS	ERE	SRE
Base (GPT-4)	0.923	0.61	**1**
Lossless (GPT-4)	**0.976**	**0.945**	0.542
Lossless (GPT-3.5)	0.743	0.786	0.374
Semantic (GPT-4)	0.936	0.622	0.949
Semantic (GPT-3.5)	0.93	0.623	0.937
Zlib Deflate Least	1	1	0.594
Zlib Deflate Most	1	0.981	0.615

and entropy metrics. Evidently, GPT-3.5 struggles to achieve lossless compression and execute it effectively given the meta-prompt.

Figure 5 further validates that the lossless versus semantic meta-prompts in part induce the improved compression behavior in GPT-4. The lossless version performs ≈286% better than Semantic and Base compression for the same model. We therefore conclude from our assessment that meta-prompting for compression has a measurable effect on the quality of text compression tasks.

E. Semantic Retention

As discussed in Section III, the ability of a compression model to retain the underlying semantic meaning was evaluated using the cosine similarity metric between the embedding vectors from the original and decompressed text. Cosine similarity measures the angle between two embedding vectors, effectively capturing the degree of similarity between the original and decompressed text in the embedding space.

Both the overall magnitude and relative performance difference between models remain fairly stable across texts, which demonstrates the innate ability of LLMs to capture the underlying semantic context at a much higher level than exact text reconstruction. This behavior occurs because LLMs are designed to understand and generate meaningful text, so their internal representations inherently encode semantic information.

Again, the Zlib baseline methods using the Deflate algorithm exhibit perfect semantic retention since they are lossless. We also observe that the GPT-4 Lossless compression approach achieves perfect semantic retention in several cases. GPT-4 Semantic compression performs nearly as well, indicating that it can effectively retain the semantic content of the text even when the exact reconstruction is not required. This result highlights the potential of using LLMs for semantic compression tasks in cases where the exact reconstruction of the input data is not strictly necessary.

The second column of Table IV-D gives the average cosine similarity (CS) of the models across all texts. GPT-4 Lossless is the strongest performing LLM model. GPT-4 Semantic Compression follows closely behind, with a 4% drop in performance.

Surprisingly, GPT-3.5 Semantic Compression nearly perfectly matches the GPT-4 Semantic Compression performance, to the point where the exact ordering is likely dependent on internal model randomness. GPT-4 Base Compression and GPT-3.5 Lossless Compression perform considerably worse. These results reinforce that meta-prompting contributes at least in part to the overall performance in semantic retention of compressed text.

F. Exact Reconstruction Effectiveness

Exact Reconstruction Effectiveness is a novel metric proposed in this paper to capture compression performance with respect to both the compressed size and the ability to perfectly reconstruct the original text. This metric provides a means to compare LLM compression performance with traditional lossless algorithms. While traditional lossless algorithms can perfectly reconstruct the text, their resulting compression rate is often much lower than LLM compression.

In contrast, LLM compression derives a more efficient compressed representation, though it struggles to perfectly replicate the original input text. Exact Reconstruction Effectiveness reconciles these differences to provide a balanced evaluation metric. Exact Reconstruction Effectiveness is computed by taking the inverse of the log-normalized compression ratio multiplied by the inverse of the edit distance, as shown in the following equation:

$$\begin{aligned} &\textit{Exact Reconstruction Effectiveness} \\ &= 1 - (\log(\textit{Compression Ratio}) \times (1 - \textit{Edit Distance})) \end{aligned} \tag{6}$$

This equation maximizes the compression ratio and minimizes the edit distance (or in this case, maximizes the inverse of the edit distance as this accounts for numerical instability caused by 0 values in the edit distance). Taking the logarithm of the compressed ratio scales the relative importance of more effective methods. This metric is concerned with the ability to reconstruct the exact input, so we do not want to allow methods that have a disproportionately high compression rate, but terrible edit distance, to score highly.

For example, if a method discards 99% of the text, its compression rate alone could be enough to compensate for a very high edit distance. Taking the logarithm of the compression ratio mitigates the impact of such situations and balances between both the compression ratio and the edit distance. We take the inverse of the entire equation as the compression rates are strictly less than 1, resulting in negative values when log-normalized. Inverting this value is not mathematically necessary, but maximizes a more intuitive positive number, rather than minimizing a negative number.

The third column of Table IV-D shows the Normalized Exact Reconstruction Effectiveness for each model. Zlib's least compression method scores marginally higher than the most method due to the inherent trade-off in compression ratio. GPT-4 lossless compression scores second highest and validates both the metric formulation and the meta-prompting technique. GPT-4 semantic compression scores the worst as it is optimized for the opposite use case and thus its added benefit of best-in-class compression rate is mitigated and its average performance in exact reconstruction is unable to compensate.

The potential applications of our findings include efficient data storage and retrieval, particularly in situations where exact text reproduction is not strictly necessary, and maintaining the semantic information is of greater importance than preserving the exact structure of the text. Moreover, this research opens up new avenues for exploring LLMs and their potential in other domains where compression is essential, such as multimedia and sensor data. Future work could investigate the development of domain-specific prompts and techniques to further optimize LLM-based compression for a wider range of applications.

G. Semantic Reconstruction Effectiveness

Semantic Reconstruction Effectiveness is the other novel metric proposed in this paper. We use this metric to evaluate the performance of compression algorithms with respect to the captured underlying semantic context. This comparison is particularly useful for scenarios where exact text reconstruction is not a strict requirement, and the focus is instead on preserving the overall meaning of the original text.

$$\begin{aligned} &\textit{Semantic Reconstruction Effectiveness} \\ &= \text{Compression Rate} \times \text{Cosine Similarity} \end{aligned} \tag{7}$$

The fourth column of Table IV-D displays the computed Semantic Reconstruction Effectiveness for each model. The results validate our metric approach and the effectiveness of the meta-prompting technique, as GPT-4 Semantic Compression emerges as the highest-performing model under this metric by a considerable margin. In contrast, the two lossless compression models exhibit the worst performance since they are optimized for the exact opposite use case, which minimizes the data size at the expense of discarding semantically rich information from the input text.

These findings again highlight the potential of LLMs such as GPT-4 in semantic compression applications where exact text representation is not a priority. For example, in a scenario where an LLM generates a tailored sales pitch using a context document, maintaining the precise wording or structure of the context document is not crucial, as long as the core semantic selling points are retained.

To summarize, we demonstrate the value of our novel metric, Semantic Reconstruction Effectiveness, in assessing the performance of compression algorithms with respect to captured semantic context between input and output text. The superior performance of GPT-4 Semantic Compression in this regard, as well as the potential applications of semantic compression in various use cases, underscores the versatility and adaptability of LLMs for tasks where preserving meaning is more important than exact text reconstruction.

V. Related Work

This section compares our approach with related work on evaluating LLMs and data compression.

A. Neural Data Compression

Data compression aims to reduce the size of data in a way that maximally preserves the original raw data before compression. This problem has recently been addressed via neural models [20], [21] which can achieve more nuanced compression policies than rigidly defined algorithms or encoding schemes. These approaches may offer promise for complex compression tasks, such as for high-entropy data or intricately structured data formats, at the expense of increased performance overhead or limited generalization power over legacy approaches.

Our work leverages LLMs, which are treated as a black box with minimal interpretability from input prompt to output response. Our approach is based on the view that the model weights of an LLM represent a compressed representation of its training data and can thus serve as a compression mechanism for input data via strategic prompting. Motivated by fundamental constraints on input token counts [8] and request counts to an LLM, we explore the compression capabilities of LLMs in both lossless and lossy paradigms on both text-to-text and text-to-code tasks.

B. Large Language Model Evaluation

Empirical evaluation of LLMs is still a nascent discipline. In particular, there is both interest and urgency in improving our collective understanding of the ways an LLM produces an output response given an input prompt to (1) enhance transparency and confidence around LLM applications and (2) mitigate systematic bias [22]–[27]. Efforts in this area center on the global statistical trends in natural language usage as it applies to prompting and downstream use of output in LLMs.

There are a number of open questions with respect to fair evaluation of LLMs and mitigation of bias in these models, as well as the most productive techniques for ensuring high quality responses from an input prompt [6], [28]. Our work assesses the affects of prompt engineering on the measurable effects of compression quality for text compression. Evaluation methods of LLMs stand to further enhance and contextualize the findings in this work, but much additional research is required in this area.

VI. Concluding Remarks

This paper presents an initial evaluation of compression techniques in Large Language Models (LLMs), specifically ChatGPT-3.5 and ChatGPT-4. We propose two novel metrics in addition to compressed entropy, compression ratio, and edit distance for evaluating performance: Semantic Reconstruction Effectiveness (SRE) and Exact Reconstruction Effectiveness (ERE).

The following is a summary of key lessons learned from the research presented in this paper:

- *Evaluation metrics provide a sound basis for comparisons* Our SRE and ERE metrics provide a sound and standardized means of assessing the effectiveness of LLM compression techniques, considering both the semantic

aspects and the precise textual content of LLM outputs on compression directives.

- *Evaluations are limitation by resource constraints* The results presented in this paper are inherently limited by a small number of data samples, as well as limited resources.
- *Reproducibility challenges across releases* The LLM models are updated and modified on an unknown basis, which may affect the reproducibility of our results over time. In a similar vein, different users may observe different model outputs for identical input prompts, for reasons that are not entirely clear, which may limit reproducibility in some cases.

REFERENCES

[1] OpenAI, "Introducing chatgpt," 2023. [Online]. Available: https://openai.com/blog/chatgpt

[2] Google, "Bard," 2023. [Online]. Available: https://bard.google.com/

[3] Anthropic, "Introducing claude," 2023. [Online]. Available: https://www.anthropic.com/index/introducing-claude

[4] A. AWS, "Amazon titan," 2023. [Online]. Available: https://aws.amazon.com/bedrock/titan/

[5] AI21, "Announcing ai21 studio and jurassic-1 language models," 2023. [Online]. Available: https://www.ai21.com/blog/announcing-ai21-studio-and-jurassic-1

[6] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," 2023.

[7] OpenAI, "What are tokens and how to count them?" 2023. [Online]. Available: https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them

[8] ——, "Models overview," 2023. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[9] "Zlib technical details," *Zlib.net*, 2022.

[10] Google, "Embeddings," 2022. [Online]. Available: https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[12] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," 2018.

[13] R. S, A. S. Bharadwaj, D. S K, M. S. Khadabadi, and A. Jayaprakash, "Digital implementation of the softmax activation function and the inverse softmax function," in *2022 4th International Conference on Circuits, Control, Communication and Computing (I4C)*, 2022, pp. 64–67.

[14] Adobe, "Lossy vs lossless compression differences and when to use." 2023. [Online]. Available: https://www.adobe.com/uk/creativecloud/photography/discover/lossy-vs-lossless.html

[15] P. S. Foundation, "zlib — compression compatible with gzip," 2023. [Online]. Available: https://docs.python.org/3/library/zlib.html

[16] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[17] Wikipedia, "Levenshtein distance," 2023. [Online]. Available: https://en.wikipedia.org/wiki/Levenshtein_distance

[18] OpenAI, "Embeddings," 2023. [Online]. Available: https://platform.openai.com/docs/guides/embeddings

[19] Wikipedia, "Cosine similarity," 2023. [Online]. Available: https://en.wikipedia.org/wiki/Cosine_similarity

[20] Y. Yang, S. Mandt, and L. Theis, "An introduction to neural data compression," 2022.

[21] M. Alwani, Y. Wang, and V. Madhavan, "Decore: Deep compression with reinforcement learning," 2022.

[22] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. Cosgrove, C. D. Manning, C. Ré, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. Wang, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda, "Holistic evaluation of language models," 2022.

[23] C. Meister and R. Cotterell, "Language model evaluation beyond perplexity," 2021.

[24] S. Takahashi and K. Tanaka-Ishii, "Evaluating Computational Language Models with Scaling Properties of Natural Language," *Computational Linguistics*, vol. 45, no. 3, pp. 481–513, 09 2019. [Online]. Available: https://doi.org/10.1162/coli_a_00355

[25] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," 2023.

[26] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[27] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[28] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.