Title: Text-to-Prompt Compression: Solving the Context Window Problem in LLMs

---

1. Introduction

Large Language Models (LLMs) such as GPT-4, Claude, and Gemini have a limited context window—a maximum number of tokens they can process in a single prompt. When users try to work with large documents or entire codebases, this limit often leads to truncated inputs, loss of critical context, and ultimately hallucinations or incomplete responses.

Text-to-prompt compression is a solution to this problem that focuses on reducing the size of the input while preserving the essential meaning and information.

---

2. What is Text-to-Prompt Compression?

Text-to-prompt compression is the process of transforming long or complex textual inputs into shorter, more focused prompts that fit within the model's token limits. This is done by identifying and preserving the most relevant information needed to answer the user query or complete the task.

---

3. Why Use It? - **Solve token overflow issues** - **Preserve core information for reasoning** - **Reduce cost and latency** - **Enable multi-document or multi-module input handling**

---

4. How Does It Work?

There are three main approaches to text-to-prompt compression:

A. Extractive Summarization

- Pull out the most relevant sentences or paragraphs.
- Techniques: Sentence embedding + cosine similarity, keyword matching.
- Tools: `LangChain`, `SpaCy`, `Transformers`, `SBERT`

Example: > Original Text: 10,000 tokens from a research paper. > > Compressed: Top 5 paragraphs with highest similarity to query.

B. Abstractive Summarization

- Generate a compressed version using another LLM.
- Preserves semantic meaning, rephrases content.
- Tools: GPT-3.5-turbo, Pegasus, BART.

Example: > LLM input: “Summarize this doc in under 500 tokens” > Output: A condensed, fluent summary.

C. Hybrid Compression

- Combine extractive + abstractive.
- Extract top sentences, then summarize.
- Best for multi-chunk documents or chat history.

Bonus: Task-specific Compression - Instead of summarizing, transform into task-specific prompt. - E.g., Convert code comments + docstrings into instruction format for bug fixing.

---

5. Implementation Pipeline

1. **Chunk Long Input** (e.g., using LangChain or NLTK)
2. **Rank Chunks** by relevance to query (via embeddings)
3. **Select Top-N Chunks** based on token budget
4. **Summarize if needed** (abstractive)
5. **Format into Final Prompt** (task-specific)

---

6. Example in Enterprise Coding Copilot

Problem: LLM can’t read the whole monolithic repository (100k+ tokens).

Compression solution: - Use embedding-based retrieval to get top 10 relevant files. - Extract method/function comments and key lines. - Generate a 300-token instruction to fix a bug in auth module.

---

7. Benefits Over Naive Truncation | Naive Truncation | Text-to-Prompt Compression | |——————|——————————| | Arbitrarily cuts content | Intelligently selects and rewrites | | High hallucination risk | Lower risk, preserves intent | | May miss important details | Highlights critical elements |

---

8. Tools & Frameworks - LangChain (`DocumentCompressorPipeline`, `SummarizationChain`) - HuggingFace Transformers - LLMs: GPT-3.5, Claude, Gemini - Embedding models: `all-MiniLM`, `text-embedding-ada-002`
