Why Naive Binary Representation Fails for LLM Efficiency

Sushanth Tiruvaipati

July 2025

Abstract

We investigate the impact of representing code in binary and assembly form when prompting large language models (LLMs). Contrary to the intuitive belief that machine code may be more efficient for models to process, our analysis across 60 code tasks and four prompt types reveals a staggering token inflation of over 14,000% for binary representations. We show that naive binary encoding leads to massive cost overheads, making it a poor choice for efficient LLM inference.

1 Introduction

Token economy is critical in optimizing LLM workflows. With increasing interest in low-level code representation—such as hexadecimal binary and assembly—for compression or obfuscation, we explore the practical implications on token count and API cost. This paper presents empirical evidence that naive binary prompts are not only less efficient but also significantly more expensive than their source or assembly counterparts.

2 Methodology

We used a pipeline that compiled C code into binary and assembly formats and fed them into GPT-4 via prompts for four tasks: explanation, optimization, debugging, and conversion. For each prompt, we measured:

- Token counts for source, binary, and assembly formats.
- Percentage savings (or inflation) in tokens.
- Estimated API costs assuming 500 output tokens per prompt.

3 Results

3.1 Token Inflation

- **Binary representation**: 14,543% token increase on average.
- **Assembly representation**: 903% token increase on average.
- **Worst-case binary inflation**: 48,045%.

3.2 Cost Impact

- Total cost for source prompts: $2.15.
- Cost for binary prompts: $42.88 (1896% increase).
- Assembly prompts cost: $5.44 (153% increase).

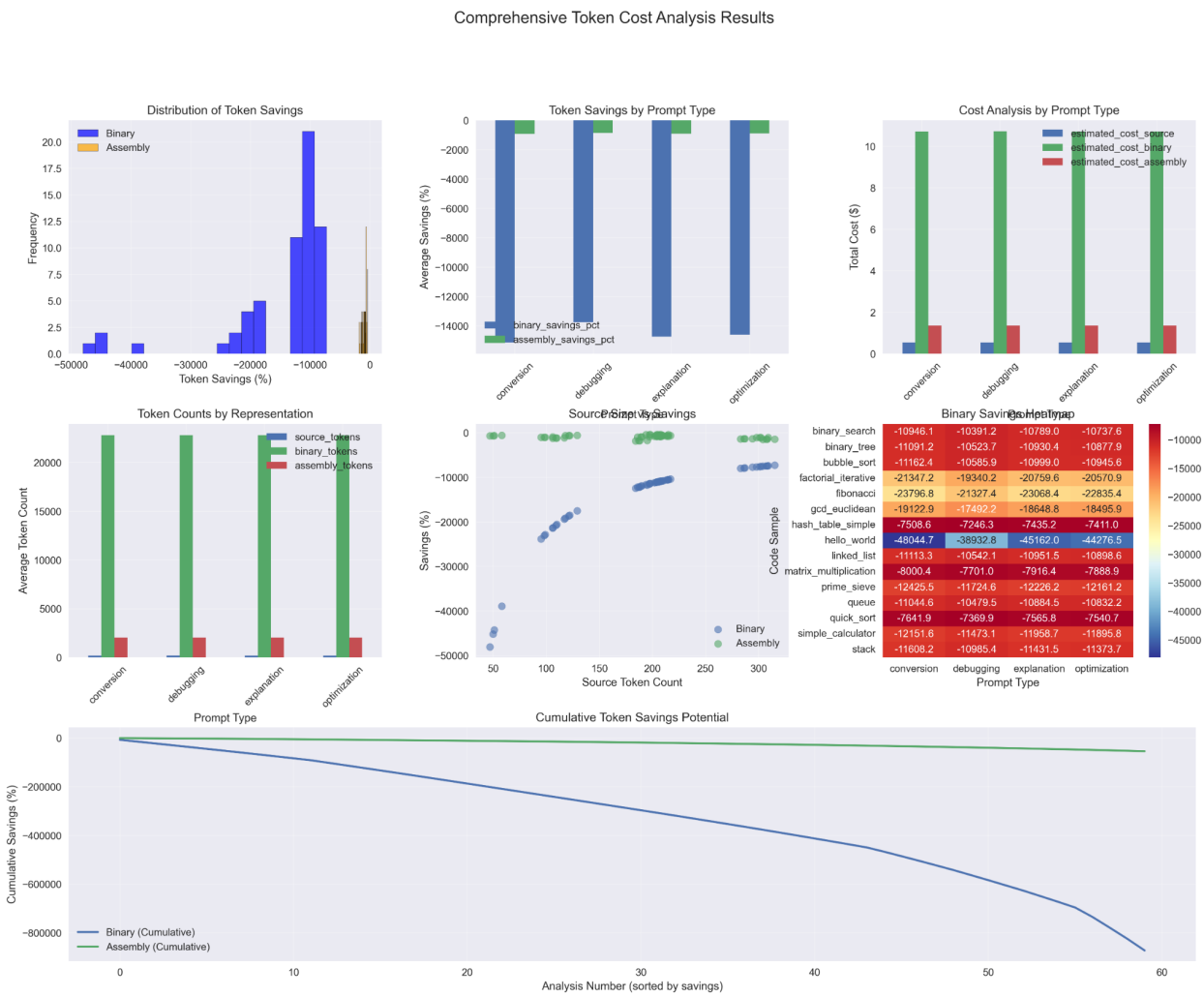

Figure 1: Visualization of token inflation and cost comparison across binary, assembly, and source code representations.

4 Discussion

The results clearly show that naive binary representation not only fails to reduce token count—it explodes it. This is due to the inefficiency of LLM tokenizers with hexadecimal strings and the loss of high-level structure in compiled formats. While binary may seem "smaller" on disk, it expands dramatically in tokenized form.

Assembly offers a middle ground, being more readable and still somewhat more efficient than binary, though worse than source.

5 Implications

- Binary prompts should be avoided in production LLM applications aiming for efficiency.
- Source and assembly formats remain superior for interpretability and cost.
- Cost savings alone do not justify low-level prompts—model accuracy must also be considered.

6 Conclusion

Naive binary prompts catastrophically increase token count and cost when interacting with LLMs. These findings advocate for caution when exploring token compression through binary encoding. Future research should examine model performance and accuracy with binary and assembly inputs before considering such representations viable.

Acknowledgements

Thanks to the open-source and AI research communities for tools that made this analysis possible.