



Lossless Text Compression Using LLMs

Revisiting Shannon's Symbol Ranking in the LLM Setting

Pesaresi Seminars

Department of Computer Science - University of Pisa

Angelo Nardone

April 10, 2026

Introduction

Problem definition and basic notation

Lossless Compression

Formal definition of compression and decompression

Let Σ be a finite alphabet and let Σ^* be the set of all finite strings over Σ .

Formal definition

A **lossless compression scheme** consists of two mappings

$$C : \Sigma^* \rightarrow \{0, 1\}^*, \quad D : \{0, 1\}^* \rightarrow \Sigma^*$$

satisfying

$$D(C(x)) = x \quad \forall x \in \Sigma^*$$

Objective

The goal is to encode text exactly, but using fewer bits on average than the original representation.

$$\mathbb{E}[|C(x)|] < |x|$$

Why Lossless Compression Matters

Storage, communication, and exact reconstruction

Why study compression?

- **Storage:** reducing data size lowers storage costs at scale.
- **Transmission:** fewer bits enable faster communication over limited bandwidth.
- **Energy:** less data movement reduces energy consumption.

When lossless compression is needed

- **Exact reconstruction** is required
- Typical examples:
 - source code
 - databases and logs
 - medical or legal data

When lossy compression is acceptable

- **Small distortions** are tolerable
- Typical examples:
 - images (JPEG)
 - audio (MP3)
 - video (H.264)

What Do We Mean by Text?

Different textual domains exhibit different structure

Text

In compression, text is any symbolic sequence $S = s_1, \dots, s_n$ over a finite alphabet Σ .

Many kinds of symbolic sequences can be treated as text for compression purposes.

- **Natural language:** books, articles, web pages
- **Source code:** syntax-rich and highly repetitive
- **Structured data:** logs, JSON, XML
- **Biological sequences:** DNA over $\{A, C, G, T\}$

Key observation

These domains exhibit different statistical regularities. In this talk, we focus on **source code**, while many of the ideas extend more broadly to natural language texts.

Standard Text Compressors

Main approaches and their limitations

Typical approaches

- **Statistical coding:** Huffman coding, arithmetic coding
- **Dictionary-based:** gzip, zstd — replace repeated patterns with back-references
- **Block-sorting:** bzip2 — reorder text via BWT, then encode

Key limitation on text

Traditional compressors capture **local regularities**, but struggle with:

- long-range dependencies
- semantic and grammatical structure

Method

Symbol Ranking text compression approach

Symbol Ranking Text Compression

Shannon's idea: prediction-driven encoding (1951)

Symbol Ranking Compression was introduced by Shannon (1951).

Key idea: predict before encoding

- given $S = (s_1, \dots, s_{n-1})$ (over Σ), a **predictor** estimates the next symbol s_n
- it produces a **probability distribution** over Σ (conditioned on the context)

Rank

The **rank** $r_n \in \mathbb{N}$ is the position of s_n in the symbols sorted by predicted probability.

- **Idea:** encode s_n via its rank r_n
- $(s_1, \dots, s_n) \longrightarrow (r_1, \dots, r_n)$ (from text to integers)

Prediction Schemes

Shannon's approaches to symbol ranking

Shannon proposed three strategies for the prediction step:

(i) **First Shannon Method** - *Single guess* ($T = 1$)

- guess the most likely symbol
- output 0 if correct, otherwise ESC s_n

(ii) **Second Shannon Method** - *Sequential guessing* ($T = \infty$)

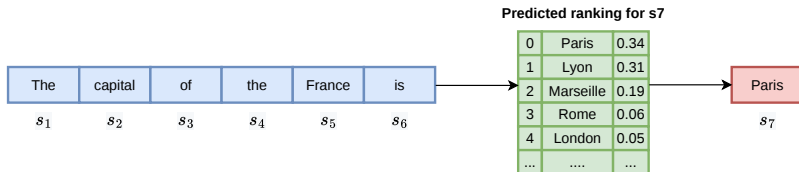
- keep guessing until the correct symbol is found
- output the **number of attempts** (the rank)

(iii) **Third Shannon Method** - *Thresholded guessing* ($1 < T < \infty$)

- as in (ii), but stop after T wrong guesses
- output ESC s_n if not found within T

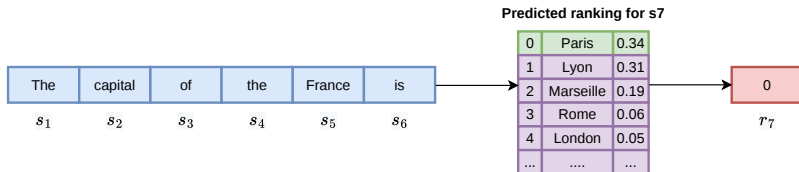
Shannon's Prediction Schemes

Three predictor-based encoding methods



Shannon's Prediction Schemes

Three predictor-based encoding methods

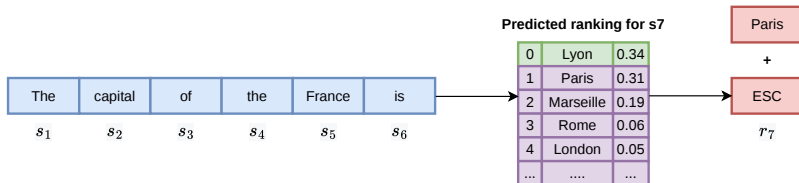


First method: single guess ($T = 1$)

Predict only the most likely symbol. Output 0 if the guess is correct, and ESC followed by the true symbol otherwise.

Shannon's Prediction Schemes

Three predictor-based encoding methods

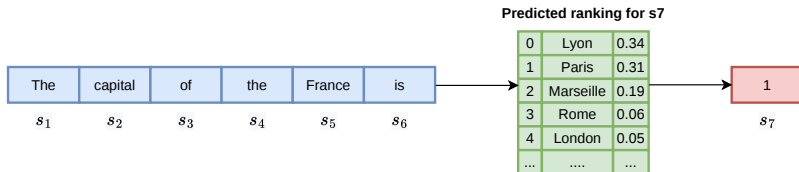


First method: single guess ($T = 1$)

Predict only the most likely symbol. Output 0 if the guess is correct, and ESC followed by the true symbol otherwise.

Shannon's Prediction Schemes

Three predictor-based encoding methods

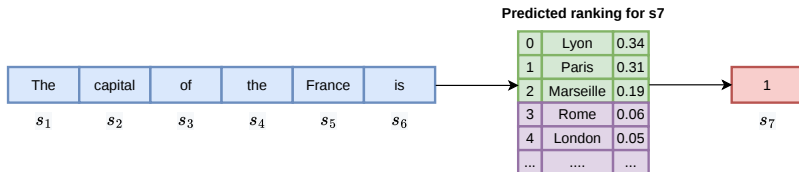


Second method: sequential guessing ($T = \infty$)

Guess symbols in ranked order until the correct one is found. Output the **rank** of the true symbol.

Shannon's Prediction Schemes

Three predictor-based encoding methods



Third method: thresholded guessing ($1 < T < \infty$)

Guess symbols sequentially up to a threshold T . Output the rank if found within T guesses; otherwise output ESC followed by the true symbol.

From Prediction to Compression

Why good predictors lead to better compression

After the prediction step, the rank sequence $(r_1, \dots, r_n)$ is compressed using **standard compressors**.

Golden Rule

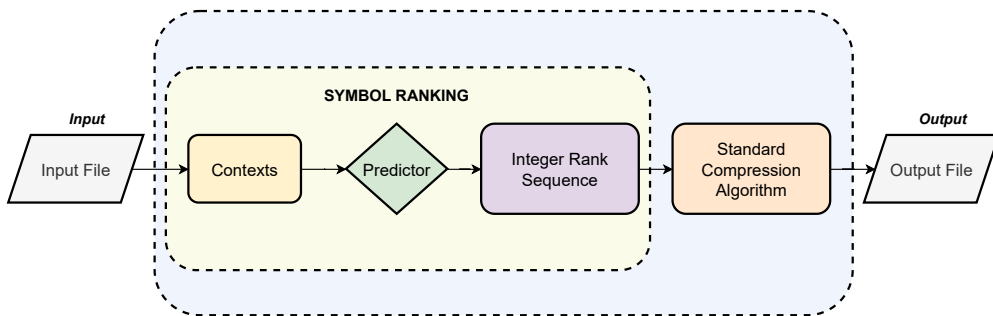
The following principle holds:

- Most symbols are guessed within a few attempts
- $\Rightarrow$ low entropy of the rank sequence
- $\Rightarrow$ high compressibility

Takeaway: good prediction $\Rightarrow$ good compression

Pipeline

Symbol Ranking Text Compression



LLMs as Predictors

From language modeling to compression

Core idea

Use a **LLM** as the predictor in the Symbol Ranking pipeline.

Key insight:

language modeling $\approx$ compression

LLMs as Predictors

From language modeling to compression

Core idea

Use a **LLM** as the predictor in the Symbol Ranking pipeline.

Key insight:

language modeling $\approx$ compression

Existing approaches

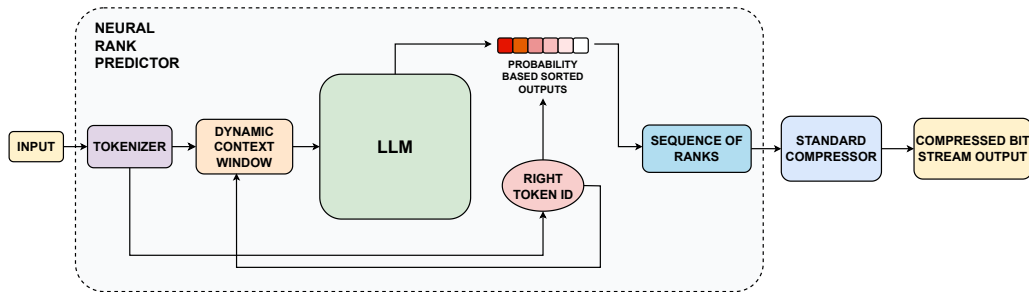
- RNNs / LSTMs (early works)
- Transformers (turning point)
- **LLMZip**, **FineZip**, **AlphaZip**

Pros / Cons

- + strong compression ratios
- very low throughput
- limited practical usability

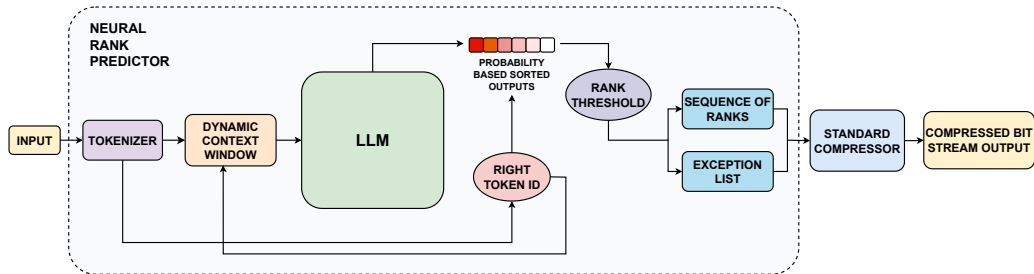
Known Pipeline

Shannon's Second Method: keep guessing ($T = \infty$)



Extended Pipeline

Shannon's First and Third Method: keep guessing up to a fixed threshold $T = 1,63$



Results

Experimental setup and results

Datasets

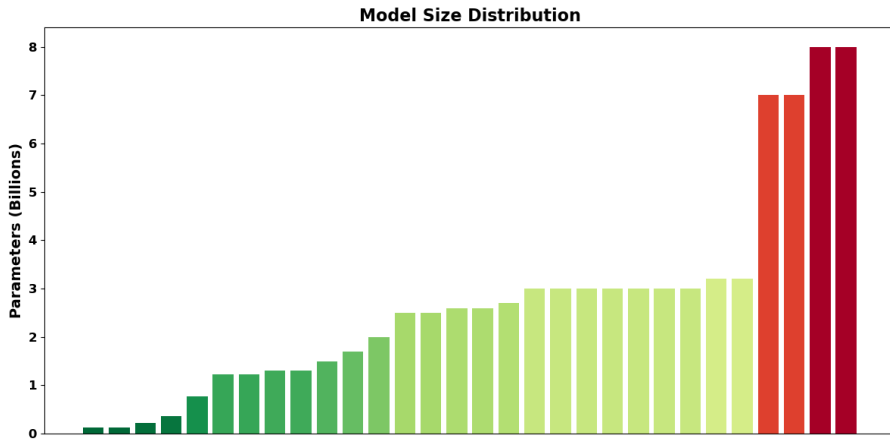
Data source and evaluation setup

- **Source:** **Stack-Edu**, multilingual source code from **Software Heritage**
- **Languages:** Python, C, C++, C#, Java, JavaScript

Language	# Files	Size (MB)	Avg (B)
Java	34k	100	3048
C#	34k	100	3097
Python	38k	100	2737
JavaScript	39k	100	2689
C++	36k	100	2948
C	39k	100	2708
Total	220k	600	–

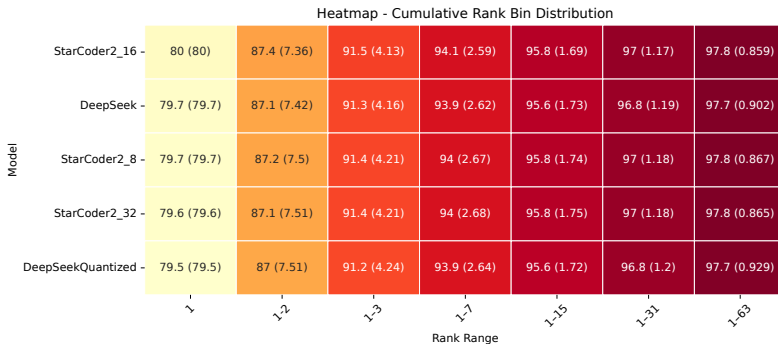
Model Size Distribution

Models Overview



Prediction Performance

Rank distribution across models



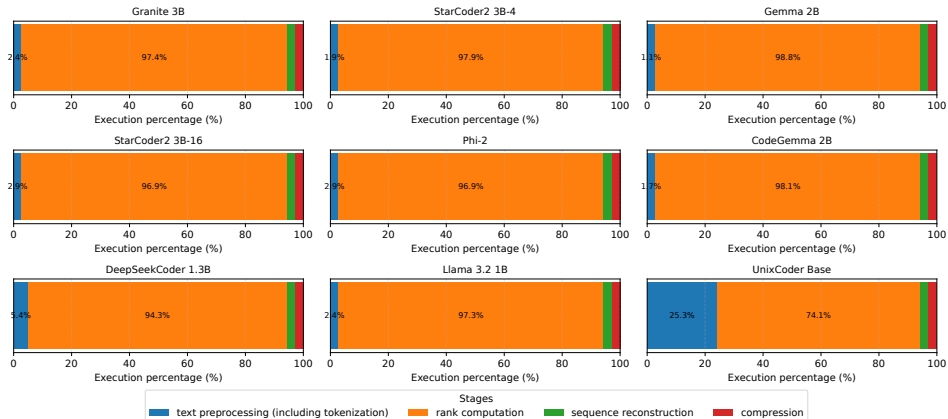
Key observation:

- most symbols are predicted with low rank
- symbol ranking is well supported by modern LLMs

Runtime Analysis

Rank computation dominates execution time

Execution share per model (normalized to 100%) - $T=\infty$



Compression Results

Impact of different Shannon schemes

Key finding: the **third Shannon scheme** achieves the best performance among LLM-based approaches.

vs. Second Shannon method

- **+37%** compression ratio
- **+40%** throughput

vs. Standard compressors

- **+20%** compression ratio
- up to 100× slower

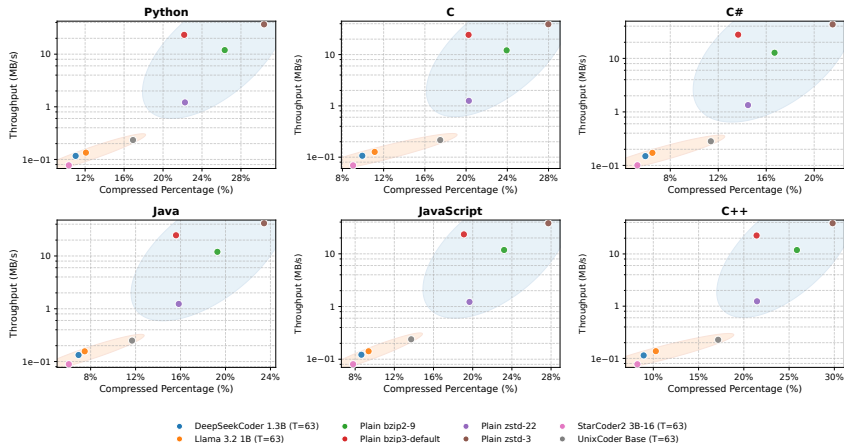
Performance depends on: model choice, hardware, and pipeline parameters

Better compression than standard tools, at the cost of speed.

LLM-Based Compression ($T = 63$) vs Standard Compressors

Throughput vs. compressed percentage

Throughput (log) vs Compressed Percentage - $T=63$ (bzip3) vs Plain (bzip3, zstd-3, zstd-22, bzip2-9)



Conclusion

Open problem sand future direction

Conclusions

Main findings

- Studied **lossless compression** using LLM-based predictors
- **Results**
 - up to **+20%** compression ratio vs standard compressors
 - up to **100×** **slower** in throughput
 - $T = 63$: **best**, improving both compression (+37%) and throughput vs $T = \infty$
- **Takeaway**: better compression than classical methods, but **speed is the main bottleneck**

Open question

How can we reduce runtime without sacrificing compression?

Future Work

Towards higher throughput and practical deployment

Pipeline improvements

- Introduce **Prefix-Free Parsing (PFP)** to reduce redundancy before LLM prediction
- Skip the **first k tokens** to avoid costly low-accuracy predictions
- Adapt encoding to **non-uniform probability distributions**

Model improvements

- **Distillation** of large models into lightweight variants
- **Language-specific fine-tuning** (e.g., with LoRA)
- Combine LLMs with **fast statistical models** (hybrid approaches)



Thank you for your attention
Q & A

Angelo Nardone

Pesaresi Seminars

Department of Computer Science - University of Pisa

angelo.nardone@phd.unipi.it