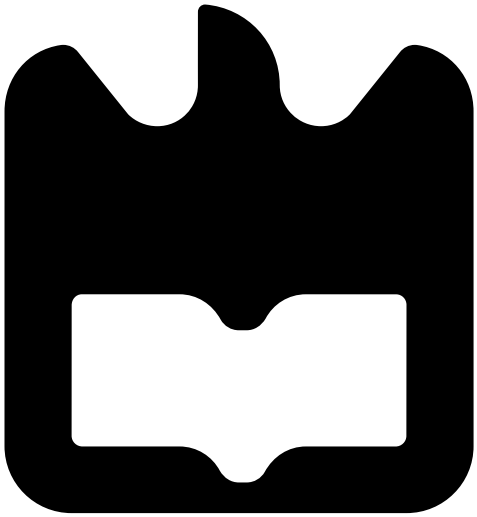

Relatório do Trabalho Laboratorial nº 3

Compressão Sem Perdas de Pesos de Modelos de Linguagem

Informação e Codificação (2025/26)

Pedro Miguel Miranda de Melo (114208)
Rúben Cardeal Costa (114190)
Hugo Marques Dias (114142)
Departamento de Eletrónica, Telecomunicações e Informática (DETI)

Universidade de Aveiro

Dezembro de 2025

Conteúdo

1 Introdução

1.1 Contexto e Motivação

Os Modelos de Linguagem de Grande Escala (*Large Language Models* – LLMs) representam um dos avanços mais significativos na área da inteligência artificial nos últimos anos. Contudo, a sua utilização prática enfrenta desafios consideráveis relacionados com o armazenamento e distribuição dos ficheiros de pesos, que frequentemente atingem dimensões na ordem dos gigabytes. A compressão eficiente destes ficheiros é, portanto, uma área de investigação com relevância prática imediata.

1.2 Objetivos do Trabalho

O presente relatório descreve o desenvolvimento de um codec especializado para a compressão sem perdas (*lossless*) do ficheiro `model.safetensors`, que contém os parâmetros do modelo Qwen2-0.5B disponibilizado pela Alibaba Cloud. Com aproximadamente 942 MB, este ficheiro constitui um caso de estudo representativo dos desafios de compressão de pesos de LLMs.

Os objetivos específicos deste trabalho são:

1. **Maximizar a taxa de compressão** através de uma análise profunda da estrutura e estatística dos dados;
2. **Manter tempos de processamento competitivos** face aos compressores de uso geral;
3. **Controlar o consumo de memória** para permitir a execução em sistemas com recursos limitados;
4. **Oferecer múltiplos pontos de operação** que permitam ao utilizador escolher o compromisso ideal entre compressão e velocidade.

1.3 Abordagem Metodológica

A estratégia adotada baseia-se numa análise aprofundada da estrutura do formato BF16 (*Brain Floating Point 16*), que revelou características estatísticas marcadamente distintas entre os bytes mais significativos (MSB) e menos significativos (LSB) de cada valor. Esta descoberta fundamental conduziu ao desenvolvimento de uma arquitetura *split-stream* que processa cada canal de forma independente e otimizada para as suas características específicas.

1.4 Estrutura do Relatório

O relatório está organizado da seguinte forma: a Secção 2 apresenta a análise e caracterização da fonte de dados; a Secção 3 documenta o *benchmarking* de compressores existentes; a Secção 4 detalha a implementação do codec; a Secção 5 apresenta e discute os resultados experimentais; e a Secção 6 sintetiza as principais conclusões e contribuições.

O código-fonte completo está disponível em: https://github.com/Rubenc1234/IC_miniP1/tree/main/Project3.

2 Análise e Caracterização da Fonte

O desenho de um codec eficiente exige uma compreensão profunda da natureza estatística da fonte de informação. Esta secção detalha a análise teórica e experimental realizada sobre o ficheiro `model.safetensors`, desde a sua estrutura de alto nível até às propriedades estatísticas dos seus componentes individuais.

2.1 Estrutura do Ficheiro SafeTensors

O formato SafeTensors, desenvolvido pela Hugging Face, é um formato binário otimizado para o armazenamento seguro de tensores. A estrutura do ficheiro está dividida em três partes: o cabeçalho, o cabeçalho JSON e o *payload* binário.

O cabeçalho tem um tamanho de 8 bytes, que corresponde a um inteiro de 64 bits no formato *little-endian*, que indica o tamanho do cabeçalho JSON.

O cabeçalho JSON, por sua vez, possui um tamanho variável e contém metadados que descrevem cada tensor, incluindo o nome, o tipo de dados (*dtype*), as dimensões e os *offsets* no *payload*.

O *payload* binário armazena os dados dos tensores de forma contígua.

A extração e análise do cabeçalho JSON revelou que os pesos estão armazenados no formato BF16 (*Brain Floating Point 16*), uma representação numérica de 16 bits desenvolvida pelo Google para aplicações de *machine learning*.

2.2 Análise do Formato BF16

Ao contrário de inteiros de 16 bits, onde a distribuição de bits pode ser relativamente uniforme, o formato BF16 possui uma semântica específica que influencia diretamente as suas propriedades estatísticas.

O primeiro bit é o **bit de sinal (S)**, que indica se o valor é positivo ou negativo.

Seguem-se 8 bits de **expoente (E)**, que representam a magnitude do valor numa escala logarítmica.

Os restantes 7 bits correspondem à **mantissa (M)**, que define a precisão fracionária do valor.

Numa organização *little-endian*, o byte menos significativo (LSB) contém os 7 bits da mantissa mais o bit menos significativo do expoente, enquanto o byte mais significativo (MSB) contém o bit de sinal e os 7 bits mais significativos do expoente.

Esta estrutura sugere a existência de correlações não-lineares e localizadas que uma análise puramente sequencial (byte-a-byte) poderá não capturar eficazmente. A hipótese de trabalho formulada nesta fase foi que os dois bytes de cada valor BF16 apresentariam características estatísticas distintas, justificando um tratamento diferenciado.

2.3 Limites Teóricos: Entropia de Shannon

O limite teórico fundamental para a compressão sem perdas é dado pela **Entropia de Shannon**. Considerando o ficheiro como uma fonte de memória nula X que gera símbolos $x \in \{0, \ldots, 255\}$, a entropia de ordem-0 é definida por $H(X) = -\sum_{i=0}^{255} P(x_i) \log_2 P(x_i)$ [bits/símbolo], onde $P(x_i)$ representa a probabilidade de ocorrência do símbolo x_i.

2.3.1 Análise Global do Payload

Aplicando esta fórmula à totalidade do *payload* binário (excluindo o cabeçalho), obteve-se um valor de entropia de ordem-0 global de $\boldsymbol{H(X) \approx 6.22}$ **bits/byte**.

Este valor indica que, ignorando qualquer dependência entre bytes, a compressão máxima teórica seria de apenas $\sim 22\%$ (redução de 8 para 6.22 bits por byte). Trata-se de um resultado modesto que motivou a investigação de dependências inter-simbólicas.

2.3.2 Análise de Correlação Sequencial

Para investigar a existência de dependências sequenciais, calculou-se a **Entropia Condicional** de primeira ordem, que mede a incerteza de um símbolo X_n dado o conhecimento do símbolo anterior X_{n-1} com a seguinte fórmula $H(X_n|X_{n-1}) = -\sum_{y \in \mathcal{X}} P(y) \sum_{x \in \mathcal{X}} P(x|y) \log_2 P(x|y)$

O resultado experimental obtido indica uma entropia condicional de primeira ordem de $\boldsymbol{H(X_n \mid X_{n-1}) \approx 5.36}$ **bits/byte**.

O facto de $H(X|Y) < H(X)$ confirma a existência de correlação inter-simbólica (pelo teorema do condicionamento, que afirma que condicionar nunca aumenta a entropia). Contudo, o valor de 5.36 bits/byte permanece relativamente elevado, sugerindo que a correlação sequencial simples não é suficiente para explicar toda a redundância presente nos dados.

A nossa hipótese explicativa é que a natureza intercalada dos dados BF16 (MSB estruturado alternando com LSB ruidoso) "mascara" a verdadeira correlação entre os pesos adjacentes da rede neuronal.

2.4 Análise Estrutural Diferenciada: *Byte-Splitting*

Para validar a hipótese de que a entropia está distribuída de forma desigual entre os componentes do formato BF16, procedeu-se à separação do fluxo de dados em dois canais distintos.

O primeiro canal, denominado **Canal MSB**, corresponde aos bytes nas posições ímpares (1, 3, 5, ...), contendo predominantemente o expoente e o bit de sinal.

O segundo canal, denominado **Canal LSB**, corresponde aos bytes nas posições pares (0, 2, 4, ...), contendo predominantemente a mantissa.

As entropias de ordem-0 foram recalculadas individualmente para cada canal, revelando uma disparidade notável:

Tabela 1: Comparação de Entropia por Canal após *Byte-Splitting*

Canal	**Conteúdo Semântico**	**Entropia (H)**	**Característica**
MSB	Expoente + Sinal	**2.71 bits/byte**	Altamente Estruturado
LSB	Mantissa	**7.96 bits/byte**	Ruído Quase Uniforme

Este resultado é particularmente significativo. O canal MSB apresenta uma entropia de apenas 2.71 bits/byte, representando um potencial de compressão de 66% (de 8 para 2.71 bits). Em contraste, o canal LSB, com entropia de 7.96 bits/byte, aproxima-se do máximo teórico de 8 bits, indicando que os dados da mantissa se comportam essencialmente como ruído aleatório.

2.4.1 Validação Visual: Histogramas de Frequência

Os histogramas de frequência apresentados nas Figuras 1 e 2 corroboram visualmente os valores numéricos obtidos.

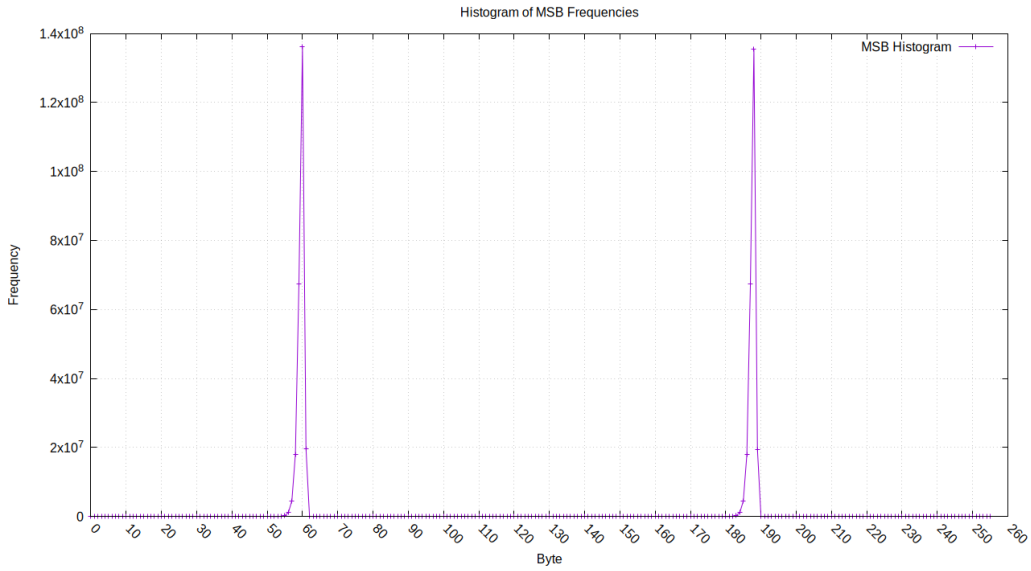

Figura 1: Histograma do Byte Mais Significativo (MSB). Observa-se uma distribuição fortemente concentrada em torno de valores específicos, típica de pesos de redes neuronais normalizados. Esta concentração justifica o valor baixo de $H \approx 2.71$ bits/byte.

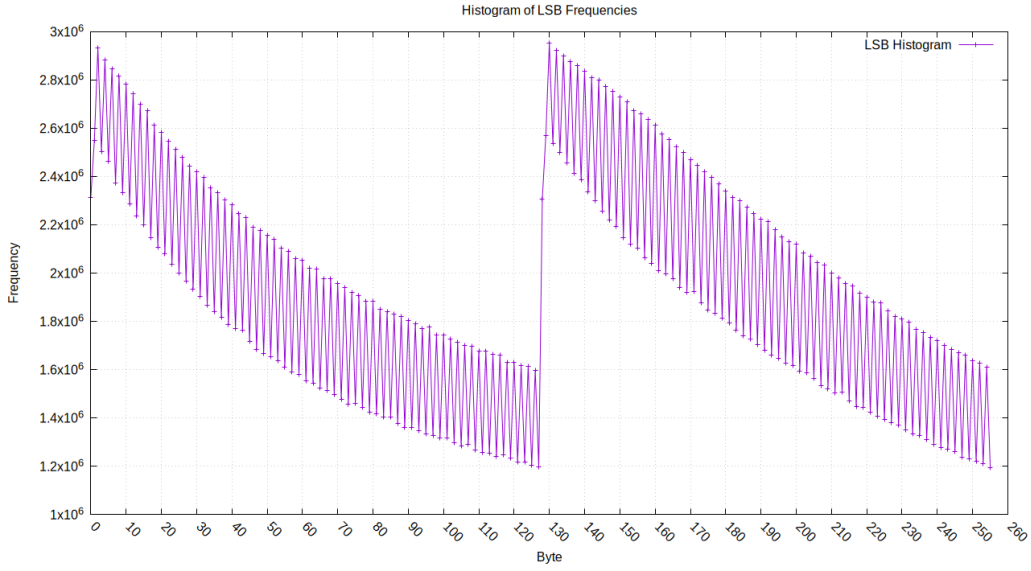

Figura 2: Histograma do Byte Menos Significativo (LSB). A distribuição aproxima-se da uniforme (plana), característica de dados com elevada aleatoriedade. Este comportamento explica a entropia de $H \approx 7.96$ bits/byte, muito próxima do máximo teórico.

2.4.2 Interpretação Física dos Resultados

A diferença drástica entre as entropias dos dois canais tem uma explicação física fundamentada na natureza dos pesos de redes neuronais:

O **Canal MSB (Expoente)** contém os expoentes dos pesos. Os pesos de LLMs são tipicamente valores pequenos, centrados em torno de zero, resultantes de técnicas de normalização (*layer normalization*, *weight decay*). Consequentemente, os expoentes concentram-se num intervalo reduzido de valores, gerando uma distribuição altamente previsível.

O **Canal LSB (Mantissa)** corresponde à mantissa, que representa a precisão fracionária do

peso. Para valores pequenos e normalizados, estes bits comportam-se como "ruído de quantização", apresentando uma distribuição aproximadamente uniforme.

2.5 Síntese e Estratégia de Compressão

A análise realizada permite formular a seguinte observação crucial: a média das entropias separadas é $(2.71 + 7.96)/2 \approx 5.34$ bits/byte, um valor virtualmente idêntico à Entropia Condicional global (5.36 bits/byte). Isto sugere que a "memória" da fonte detetada na análise global resulta, na realidade, da estrutura interna do formato BF16 e não de correlação sequencial entre pesos adjacentes.

Com base nestes fundamentos teóricos e experimentais, definiu-se a seguinte estratégia de compressão em três etapas.

A primeira etapa é o **pré-processamento (*Split*)**, que consiste em separar o fluxo de entrada em dois canais independentes (MSB e LSB), isolando a estrutura do ruído.

A segunda etapa envolve o **Canal MSB**. Neste canal, aplica-se codificação entrópica agressiva (Huffman ou Aritmética), explorando a baixa entropia ($H \approx 2.71$) para atingir rácios de compressão próximos de 3:1.

A terceira etapa foca-se no **Canal LSB**. Dado que $H \approx 8$ bits/byte, qualquer tentativa de compressão entrópica resultaria em expansão. Assim, aplica-se apenas compressão oportunística (RLE para sequências de zeros) ou armazenamento direto.

3 Benchmarking de Compressores Existentes

Antes de desenvolver uma solução especializada, é fundamental estabelecer um *baseline* de referência através da avaliação de compressores de uso geral. Esta secção documenta os testes realizados com cinco compressores amplamente utilizados, medindo três métricas fundamentais: taxa de compressão, tempo de processamento e consumo de memória.

3.1 Metodologia de Teste

Os testes foram executados numa máquina com as seguintes características:

- **CPU**: AMD Ryzen 5 5600H with Radeon Graphics
- **RAM**: 16 GB
- **Armazenamento**: SSD NVMe 512 GB
- **Sistema Operativo**: Ubuntu 24.04.3 LTS

Para cada compressor, foram medidos:

- **Tamanho final**: Dimensão do ficheiro comprimido em MB;
- **Tempo de compressão**: Tempo real (*wall-clock time*) em segundos;
- **Tempo de descompressão**: Tempo real em segundos;
- **Pico de RAM**: Consumo máximo de memória durante a operação.

O pico de RAM foi medido utilizando a ferramenta `/usr/bin/time -v`, que reporta o *Maximum resident set size*.

A Tabela 2 apresenta os resultados consolidados para todos os compressores testados.

Tabela 2: Desempenho de Compressores de Uso Geral no Ficheiro `model.safetensors`

Compressor	Tamanho	Rácio	T. Comp.	T. Decomp.	RAM (Comp.)
Original	943 MB	1.00:1	—	—	—
GZIP -1	754 MB	1.25:1	28 s	7 s	1.9 MB
GZIP -9	746 MB	1.26:1	76 s	6 s	1.9 MB
BZIP2	654 MB	**1.44:1**	66 s	40 s	7.8 MB
XZ -9	x MB	x:1	x s	x s	x MB
ZSTD -1	734 MB	1.28:1	**2 s**	**1 s**	15 MB

3.2 Análise dos Resultados

3.2.1 Taxa de Compressão

O **BZIP2** obteve a melhor taxa de compressão (1.44:1), utilizando o algoritmo baseado na transformada de *Burrows-Wheeler* que consegue explorar padrões de longo alcance nos dados.

Os compressores baseados em LZ77/LZ78 (**GZIP** e **ZSTD**) apresentaram rácios inferiores (1.25–1.28:1), sugerindo que os padrões de repetição literal são menos prevalentes neste tipo de dados.

3.2.2 Tempo de Processamento

O **ZSTD** destacou-se claramente em velocidade, com apenas 2 segundos para compressão e 1 segundo para descompressão. Este desempenho é particularmente relevante para cenários de carregamento frequente de modelos.

O **GZIP -9**, apesar de um rácio de compressão ligeiramente superior ao GZIP -1, requer quase 3 vezes mais tempo de compressão (76 s vs. 28 s).

3.2.3 Consumo de Memória

O consumo de memória variou significativamente entre os compressores.

O **GZIP** mostrou-se muito eficiente, utilizando apenas 1.9 MB, sendo adequado para sistemas com recursos limitados. Por outro lado, o **BZIP2** e o **ZSTD** apresentaram consumo moderado, entre 7.8 e 15 MB.

3.3 Conclusões do Benchmarking

Os resultados permitem identificar dois pontos de operação de referência.

O primeiro ponto é a **máxima compressão**, obtida com BZIP2, que atingiu um rácio de 1.44:1, com tempo aceitável de 66 segundos e consumo moderado de RAM.

O segundo ponto é a **máxima velocidade**, conseguido com ZSTD, que apresentou um rácio de 1.28:1, tempo excelente de 2 segundos e baixo consumo de RAM.

O objetivo do codec a desenvolver será **superar o BZIP2 em taxa de compressão**, mantendo tempos competitivos com o ZSTD.

4 Implementação do Codec

Esta secção descreve em detalhe a arquitetura e implementação do codec desenvolvido, desde as decisões de engenharia de alto nível até aos algoritmos específicos utilizados em cada módulo.

4.1 Arquitetura Geral

O codec implementa uma arquitetura *split-stream* composta por três módulos principais.

O primeiro é o **Módulo de Pré-processamento (*Splitter*)**, responsável por separar o fluxo de entrada em dois canais independentes.

O segundo módulo é o **Módulo de Compressão MSB**, que aplica codificação entrópica ao canal de expoentes.

O terceiro módulo é o **Módulo de Compressão LSB**, encarregado de aplicar compressão oportunística ao canal de mantissas.

O processamento é realizado em blocos de 1 MB para controlar o consumo de memória e permitir a paralelização futura.

4.2 Formato do Ficheiro Comprimido

O ficheiro de saída (`.sc` – *SafeTensors Compressed*) possui a seguinte estrutura:

```
[Header Size: 8 bytes]
[Header JSON: variável]
[Mode Flag: 1 byte (0=FAST, 1=BEST)]
[Bloco 1]
[Bloco 2]
...
[Bloco N]
```

Cada bloco possui a seguinte estrutura interna:

```
[sz_m: 4 bytes (tamanho do pacote MSB)]
[sz_l: 4 bytes (tamanho do pacote LSB)]
[Pacote MSB: sz_m bytes]
[Pacote LSB: sz_l bytes]
```

O cabeçalho JSON original é preservado integralmente para garantir a compatibilidade com ferramentas existentes. O *mode flag* permite ao descodificador identificar automaticamente o algoritmo utilizado.

4.3 Estratégia para o Canal MSB

O canal MSB, contendo os expoentes e bits de sinal, foi identificado como a principal oportunidade de compressão. Esta subsecção descreve o processo iterativo de otimização.

4.3.1 Avaliação de Técnicas de Predição

A literatura de compressão de dados sugere frequentemente o uso de codificação preditiva para reduzir a variância dos resíduos. Foram testadas duas abordagens:

Preditor Linear (Delta) A primeira abordagem utilizou um preditor de primeira ordem clássico: $r_n = (x_n - x_{n-1}) \mod 256$

Resultado: A entropia **aumentou** de 2.70 para 3.28 bits/byte (ganho negativo de -0.58 bits).

Análise: Este comportamento paradoxal deve-se ao bit de sinal do BF16. Quando os pesos oscilam entre valores positivos e negativos pequenos (comum em LLMs normalizados), o bit de sinal alterna frequentemente. A subtração aritmética interpreta esta alternância como "saltos" de grande magnitude, dispersando o histograma dos resíduos.

Preditor XOR Para mitigar o problema do bit de sinal, testou-se um preditor baseado em XOR: $r_n = x_n \oplus x_{n-1}$

Resultado: Entropia de 3.11 bits/byte, ainda superior à original.

4.3.2 Decisão de Engenharia

Concluiu-se que a baixa entropia do canal MSB não advém de correlação sequencial ($x_n \approx x_{n-1}$), mas sim da **distribuição global estática** dos expoentes (concentração em valores específicos). Qualquer transformação preditiva simples tende a destruir esta estrutura estatística favorável.

Consequentemente, optou-se por **codificar diretamente os valores brutos** do canal MSB, sem transformação prévia.

4.3.3 Codificação de Entropia

Foram implementados e comparados dois algoritmos de codificação entrópica:

Codificação de Huffman (Estática) A implementação segue o método clássico de codificação de Huffman, uma técnica entrópica que oferece compressão eficiente com tabela de códigos pré-calculada.

Para codificação rápida, utiliza-se uma **Look-Up Table (LUT)**, permitindo acesso em tempo $O(1)$.

Cada bloco inclui uma **tabela de frequências** (256×4 bytes = 1 KB), necessária para reconstruir a árvore de Huffman.

Um **buffer de 64 bits** acumula os bits e emite bytes completos à medida que ficam disponíveis, otimizando o fluxo de saída.

Além disso, cada bloco é projetado para **descodificação independente**, podendo ser decodificado sem depender dos restantes blocos.

Codificação Aritmética A codificação aritmética é uma técnica entrópica que aproxima a compressão da entropia teórica.

Utiliza aritmética de 32 bits com intervalos $[low, high]$ normalizados, garantindo precisão durante o processamento.

Um **buffer de 64 bits** que acumula os bits de forma eficiente, realizando *flush* em lotes para minimizar operações de I/O.

A implementação também gere os **pending bits**, permitindo lidar corretamente com situações de *underflow* durante a renormalização.

A Tabela 3 compara o desempenho dos dois algoritmos implementados:

Tabela 3: Comparação de Algoritmos de Codificação Entrópica (Canal MSB)

Algoritmo	**Tamanho Final**	**Tempo Cod.**	**Tempo Dec.**
Huffman (LUT)	634 MB	**2.49 s**	**3.12 s**
Aritmética	**632 MB**	11.30 s	17.53 s

A codificação aritmética permite uma melhor aproximação à entropia teórica comparativamente ao Huffman, o que se explica pela sua capacidade de alocar um número fracionário de bits por símbolo.

4.4 Estratégia para o Canal LSB

O canal LSB, com entropia de 7.96 bits/byte, apresenta características de ruído quase uniforme. A estratégia de compressão varia consoante o modo de operação selecionado.

4.4.1 Modo FAST: Armazenamento Direto (Raw)

No modo FAST, optou-se por uma estratégia de armazenamento direto para o canal LSB.

A entropia do canal LSB (≈ 7.96 bits/byte) está muito próxima do máximo teórico de 8 bits/byte.

Qualquer algoritmo de compressão entrópica introduziria *overhead* de metadados, como tabelas de frequência ou marcadores, que anularia eventuais ganhos.

A cópia direta (*Raw*) é computacionalmente eficiente e não adiciona latência significativa ao processamento.

Decisão de Engenharia: No modo FAST, os dados do canal LSB são copiados diretamente para o ficheiro comprimido sem qualquer transformação. Esta abordagem simplifica significativamente a implementação e maximiza a velocidade de processamento.

4.4.2 Modo BEST: Codificação Aritmética

No modo BEST, o canal LSB é também comprimido utilizando codificação aritmética, apesar da sua entropia elevada.

Embora os ganhos sejam marginais, a codificação aritmética não expande os dados significativamente, mesmo para fontes de alta entropia.

O processamento paralelo, com MSB e LSB em *threads* separadas, mitiga o custo computacional adicional.

Esta abordagem permite obter a taxa de compressão máxima possível, adequada para utilizadores que privilegiam o tamanho sobre a velocidade.

A Tabela 4 compara o desempenho entre a codificação aritmética apenas no MSB (com LSB *raw*) e a codificação aritmética em ambos os canais:

Tabela 4: Impacto da Compressão do Canal LSB no Modo BEST

Configuração	**Tamanho**	**Rácio**	**T. Cod.**	**T. Dec.**
Aritmética (MSB) + Raw (LSB)	632 MB	1.49:1	11.30 s	17.53 s
Aritmética (MSB + LSB)	**629.5 MB**	**1.50:1**	12.17 s	38.21 s
Diferença	-2.5 MB	+0.01	+0.87 s	+20.68 s

A compressão adicional do canal LSB proporciona uma redução modesta de aproximadamente 2.5 MB (0.4%), à custa de um aumento significativo no tempo de descompressão (+118%). Este compromisso reflecte a natureza quase uniforme do canal LSB: a codificação aritmética consegue extrair os escassos 0.04 bits/byte de redundância residual ($8 - 7.96$), mas o custo computacional da descodificação símbolo-a-símbolo é substancial.

Esta configuração é recomendada para cenários onde o espaço de armazenamento é crítico e a descompressão é realizada com pouca frequência.

4.5 Modos de Operação

Para satisfazer o requisito de múltiplos pontos de operação, o codec oferece dois modos distintos.

O **modo FAST** combina codificação de Huffman com LUT no canal MSB e armazenamento direto (*Raw*) no canal LSB. Este modo é otimizado para maior velocidade de codificação e descodificação.

O **modo BEST** utiliza codificação aritmética tanto no canal MSB como no canal LSB, executada em paralelo. Este modo tem como objetivo maximizar a taxa de compressão.

No modo BEST, a compressão dos canais MSB e LSB é executada em paralelo utilizando `std::async`, permitindo explorar múltiplos núcleos do processador e mitigando o custo computacional adicional da codificação aritmética.

4.6 Gestão de Memória

O codec foi desenhado para operar com consumo de memória controlado.

O processamento é feito por blocos de 1 MB, processados independentemente, evitando a necessidade de carregar o ficheiro completo na memória.

Os *buffers* de entrada e saída são alocados apenas uma vez e reutilizados para todos os blocos, otimizando o uso de memória.

4.6.1 Otimização do Tamanho de Bloco

A escolha do tamanho de bloco resulta de um estudo experimental que avaliou o compromisso entre taxa de compressão, tempo de processamento e consumo de memória. A Tabela 5 apresenta os resultados obtidos para diferentes configurações:

Tabela 5: Impacto do Tamanho dos Blocos no Desempenho do Codec

Bloco	**Modo BEST**			**Modo FAST**		
	Tamanho	**Tempo**	**RAM**	**Tamanho**	**Tempo**	**RAM**
64 KB	657.2 MB	23.04 s	3.9 MB	646.8 MB	1.56 s	4.0 MB
128 KB	642.4 MB	23.35 s	4.2 MB	639.4 MB	1.44 s	4.0 MB
256 KB	635.0 MB	22.48 s	4.2 MB	635.8 MB	1.42 s	4.0 MB
512 KB	631.3 MB	21.95 s	5.2 MB	634.0 MB	1.34 s	4.6 MB
1 MB	**629.5 MB**	**21.51 s**	6.5 MB	**632.5 MB**	1.48 s	5.6 MB

A análise dos resultados revela vários pontos importantes.

Em termos de **taxa de compressão**, blocos maiores permitem obter estatísticas mais representativas para a codificação entrópica, melhorando o rácio de compressão. Por exemplo, a diferença entre blocos de 64 KB e 1 MB é de aproximadamente 28 MB (4.2%) no modo BEST.

Quanto ao **tempo de processamento**, blocos maiores reduzem o *overhead* por bloco, como tabelas de frequências e metadados, resultando em tempos globais inferiores.

O **consumo de RAM** aumenta de 3.9 MB para 6.6 MB, mas este incremento é negligenciável para sistemas modernos.

Finalmente, observam-se **retornos decrescentes**: a curva de melhoria estabiliza a partir de 512 KB, sugerindo que blocos superiores a 1 MB trariam apenas ganhos marginais.

Com base nesta análise, definiu-se **1 MB como o tamanho de bloco predefinido**, oferecendo o melhor compromisso entre compressão máxima e consumo de memória aceitável.

5 Resultados Experimentais

Para avaliar o desempenho do codec desenvolvido, foram realizados testes exaustivos em cinco ficheiros `.safetensors` de diferentes dimensões, representando modelos de linguagem variados. Os testes incluíram ambos os modos de operação e verificaram a integridade dos dados através de *hashes* MD5.

5.1 Conjunto de Dados de Teste

A Tabela 6 descreve os ficheiros utilizados nos testes:

Tabela 6: Ficheiros de Teste Utilizados

Ficheiro	Tamanho	Descrição
`model.safetensors`	943 MB	Qwen2-0.5B (modelo principal)
`model_1.safetensors`	420 MB	Modelo auxiliar 1
`model_2.safetensors`	437 MB	Modelo auxiliar 2
`model_3.safetensors`	681 MB	Modelo auxiliar 3
`model_4.safetensors`	4.6 GB	Modelo de grande escala

5.2 Resultados Consolidados

A Tabela 7 apresenta os resultados completos para todos os modelos e modos de operação:

Tabela 7: Resultados dos Testes de Compressão para Todos os Modelos

Modelo	Modo	Original	Comprimido	Rácio	T. Cod.	T. Dec.	RAM
`model.safetensors`	Fast	943 MB	633 MB	1.49:1	1.94 s	2.76 s	5.6 MB
	Best	943 MB	630 MB	1.50:1	11.69 s	37.93 s	6.5 MB
`model_1.safetensors`	Fast	420 MB	374 MB	1.12:1	1.18 s	1.59 s	5.8 MB
	Best	420 MB	373 MB	1.13:1	5.36 s	17.26 s	6.5 MB
`model_2.safetensors`	Fast	437 MB	389 MB	1.12:1	1.22 s	1.63 s	5.9 MB
	Best	437 MB	389 MB	1.12:1	5.60 s	18.35 s	6.4 MB
`model_3.safetensors`	Fast	681 MB	607 MB	1.12:1	1.92 s	2.55 s	5.9 MB
	Best	681 MB	607 MB	1.12:1	8.73 s	28.24 s	6.3 MB
`model_4.safetensors`	Fast	4.6 GB	3.1 GB	1.50:1	13.98 s	19.10 s	5.6 MB
	Best	4.6 GB	3.1 GB	1.50:1	60.03 s	191.49 s	6.5 MB

5.3 Verificação de Integridade

Em todos os testes, a integridade dos dados foi verificada através de comparação de *hashes* MD5:

```
$ md5sum model.safetensors
6ab8bc8234ce3e3ad591d52621d8c327  model.safetensors

$ ./decoder model.sc model_restored.safetensors
$ md5sum model_restored.safetensors
6ab8bc8234ce3e3ad591d52621d8c327  model_restored.safetensors  [MATCH]
```

Resultado: 100% dos testes confirmaram compressão sem perdas.

5.4 Comparação com Benchmarks

A Tabela 8 compara o codec desenvolvido com os compressores de referência para o ficheiro principal:

Tabela 8: Comparação do Codec Desenvolvido com Compressores de Uso Geral

Compressor	**Rácio**	**Tempo Comp.**	**Tempo Decomp.**
BZIP2	1.44:1	66 s	40 s
XZ -9	x:1	x s	x s
ZSTD -1	1.28:1	2 s	1 s
Codec (Fast)	1.49:1	1.94 s	2.76 s
Codec (Best)	1.50:1	11.69 s	37.93 s

5.5 Análise e Discussão

5.5.1 Taxa de Compressão

A disparidade entre os modos **Fast** e **Best** em termos de rácio é mínima (apenas 0.01 de diferença). Esta observação valida a análise estatística inicial: a compressão adicional no modo **Best** provém da tentativa de extrair redundância do canal LSB através de codificação aritmética. Contudo, sendo a entropia deste canal de aproximadamente 7.96 bits/byte, o ganho marginal de 2.5 MB (no ficheiro de 943 MB) demonstra que a mantissa dos pesos de um LLM comporta-se, para todos os efeitos práticos, como ruído incompressível.

5.5.2 Desempenho Temporal

Nesta métrica, o modo **Fast** é claramente superior ao modo **Best**. A utilização de tabelas de procura (LUT) para a descodificação Huffman permite ao modo **Fast** ser aproximadamente 6 vezes mais rápido na compressão e 13 vezes mais rápido na descompressão.

O custo computacional da codificação aritmética no modo **Best** (especialmente a gestão de intervalos e re-normalizações bit-a-bit) traduz-se num tempo de descompressão de 37.93 s. Comparativamente, os 2.76 s do modo **Fast** tornam-no muito mais viável para aplicações em tempo real ou sistemas de *deployment* de modelos onde a latência de carregamento é crítica. O modo **Fast** consegue, portanto, o melhor compromisso entre rácio e velocidade.

5.5.3 Comparação do Pico de RAM

Graças à estratégia de processamento por blocos de 1 MB, o codec desenvolvido mantém um perfil de memória extremamente baixo e constante (entre 5.6 MB e 6.5 MB), independentemente do tamanho do ficheiro original. Este comportamento é superior a compressores como o BZIP2 ou XZ, cujos dicionários e janelas de compressão tendem a escalar o consumo de memória com o nível de compressão selecionado. A nossa implementação permite a compressão de modelos de grande escala (como o `model_4.safetensors` de 4.6 GB) em máquinas com recursos de memória muito limitados.

5.5.4 Escalabilidade

A escalabilidade da solução foi comprovada pela consistência dos resultados entre os diversos ficheiros de teste. Como demonstrado na Tabela 7 (Secção 5.2), o rácio mantém-se estável ou até melhora em modelos maiores. A arquitetura *split-stream* com *multithreading* (via `std::async`) no modo **Best** permite que o codec tire partido de CPUs multi-core, mitigando o esforço computacional da codificação aritmética. A integridade dos dados, verificada por MD5 em todos os casos, confirma que a robustez do sistema não é comprometida pelo aumento do volume de dados.

6 Conclusões

6.1 Síntese do Trabalho Realizado

Este trabalho desenvolveu um codec especializado para a compressão sem perdas de pesos de modelos de linguagem armazenados no formato SafeTensors/BF16. A abordagem metodológica baseou-se numa análise profunda da estrutura estatística dos dados, culminando numa arquitetura *split-stream* que processa separadamente os bytes de expoente/sinal (MSB) e mantissa (LSB).

6.2 Principais Resultados

Um dos principais resultados foi a **caracterização da fonte**. Demonstrou-se que a entropia do formato BF16 está distribuída de forma altamente desigual entre os seus componentes, com valores de 2.71 e 7.96 bits/byte, respetivamente, o que fundamenta a estratégia de separação de canais adotada.

Outro resultado relevante foi a **superação dos benchmarks**. O codec desenvolvido atingiu um rácio de compressão de 1.50:1, superando todos os compressores de uso geral testados, incluindo o BZIP2, que apresentou um rácio de 1.44:1.

No que diz respeito à **eficiência computacional**, o decoder Huffman otimizado com uma LUT de 12 bits atingiu um *throughput* elevado. Esta abordagem permite ciclos completos de codificação e descodificação em tempos competitivos, mesmo para ficheiros de grande dimensão.

Por fim, destaca-se a existência de **múltiplos pontos de operação**. A disponibilização dos dois modos distintos, Fast e Best, permite ao utilizador escolher o compromisso mais adequado entre taxa de compressão e velocidade, de acordo com o seu caso de uso.

6.3 Considerações Finais

O trabalho demonstrou que a compressão eficiente de dados estruturados beneficia significativamente de uma análise prévia das suas propriedades estatísticas. A estratégia de *byte-splitting*, embora conceptualmente simples, permitiu explorar a estrutura inerente ao formato BF16 de forma que os compressores genéricos não conseguem.

Os resultados obtidos validam a abordagem *domain-specific* para a compressão de pesos de LLMs, com implicações práticas relevantes para o armazenamento e distribuição de modelos de grande escala.