MiddleOut Lattice: Lossless Compression of LLM Artefacts with Codec-Aware Tensor Packing

AmSach

May 5, 2026

Abstract

We study exact, lossless compression of model artefacts for large language models. The core question is not whether one can quantise a cache, but whether one can reduce the storage and transfer cost of model files while preserving bit-for-bit reconstruction. We present MiddleOut Lattice, a codec-first architecture that compresses arbitrary model files, evaluates codec choice per file and per block, and supports exact reconstruction with manifest-backed metadata. Experiments on Qwen2.5-0.5B artefacts show that some model files compress substantially while remaining exactly recoverable, but also demonstrate that file type strongly controls compressibility. We further discuss the limits of general-purpose codecs, derive the relevant information-theoretic constraints, and outline a path toward architecture-aware compression of weight tensors.

1 Introduction

Large language models are expensive to store, transfer, and load. Practitioners often focus on inference speed and accuracy, but the boring operational cost of moving model files around is just as real. A 70B-parameter checkpoint is not merely a mathematical object; it is a collection of files, shards, metadata records, tokeniser state, and formatting conventions that must all travel through storage, networks, and memory hierarchies.

Most compression schemes target one of two layers. The first is *approximate numerical compression*, which trades exactness for smaller representations through quantisation, pruning, or low-rank approximation. The second is *generic file compression*, which preserves the bytes but treats the file as arbitrary data. Both are useful, but neither fully answers the user requirement here: a system that compresses as aggressively as possible while remaining lossless, selective, and model-aware.

This work takes a deliberately narrow but honest step. We start with exact compression of model artefacts as files, not just tensors. Then we build a blockwise archive format that records metadata per file and per block, skips compression when it is not beneficial, and restores the original content exactly. This is not a universal free lunch. It is a usable architecture for learning where compressibility lives, and for pushing toward a future codec-native model format.

1.1 Contributions

(1) A file- and block-level exact compression architecture, MiddleOut Lattice, that chooses the best codec per block and falls back to raw storage when compression would increase size.

(2) A manifest-backed archive format that stores enough metadata to guarantee exact reconstruction.

(3) A Qwen2.5-0.5B template benchmark showing how compressibility varies dramatically by file type.

(4) A technical analysis of why lossless whole-model compression is difficult and where future gains are likely to come from.

2 Problem statement

Let a model repository be represented as a finite collection of byte strings $\mathcal{F} = \{f_i\}_{i=1}^{n}$, where each file $f_i \in \{0,1\}^{8m_i}$ has byte length m_i. We seek a compressor C and decompressor D such that:

$$D(C(f_i)) = f_i \quad \text{for all } i,$$

and the expected or observed compressed size is smaller than the original size whenever the file contains redundancy.

Definition 1 (Lossless file compression)**.** *A compression scheme is lossless if the decompressor returns the exact original bytes for every valid input in the compressor's domain.*

The important constraint is not only exactness, but selective exactness. If some file blocks are already incompressible, the compressor should not force them into a worse representation.

Definition 2 (Selective compression)**.** *A compressor is selective if it may choose, for each file or block, between a set of encoders and a raw fallback, and it outputs the raw fallback whenever every encoded representation is larger than the uncompressed block.*

This selective property is operationally important. It guarantees the compressor is never worse than doing nothing, ignoring tiny metadata overheads that can be amortised by the archive format.

3 Background and related work

Lossless model compression has two broad families. The first includes general-purpose compressors such as zlib, bz2, and lzma. These exploit entropy and repeated structure in bytes, making them especially good for JSON, text, tokeniser artefacts, and some floating-point layouts. The second family includes model-specific systems such as ZipNN and related entropy-aware model codecs, which attempt to exploit the statistical structure of neural network weights themselves. Independent work such as ZipNN reports meaningful lossless gains on model weights and model hubs. Near-lossless systems such as SpQR, SqueezeLLM, AQLM, and others show that model structure can be exploited aggressively, though they typically sacrifice exact reconstruction.

The practical lesson is simple: exact compression on model assets is not the same as exact compression on model weights. Tokeniser files compress beautifully. Weight shards are harder. A credible paper must separate those regimes instead of pretending they are interchangeable.

4 Information-theoretic preliminaries

Let X be a discrete source over alphabet $\mathcal{A}$ with distribution $p(x)$. Its Shannon entropy is:

$$H(X) = -\sum_{x \in \mathcal{A}} p(x) \log_2 p(x).$$

For any prefix code with codeword lengths $\ell(x)$, the expected code length satisfies the classical lower bound:

$$\mathbb{E}[\ell(X)] \geq H(X).$$

More generally, any lossless compressor must encode the source with an average rate no smaller than the entropy rate, up to coding redundancy and finite-block overhead.

For a finite block B of bytes, let $S(B)$ be its original size in bytes and $C(B)$ be its compressed size. We define the compression ratio as:

$$R(B) = \frac{S(B)}{C(B)}.$$

The ratio is greater than 1 when compression succeeds and less than 1 when the encoded representation is larger than the input. A selective compressor should avoid the latter whenever possible by falling back to raw bytes.

5 Codec-first architecture

MiddleOut Lattice implements a codec-first design rather than a tensor-format-first design. That choice matters. A file can be compressed before we even decide whether it is a tokenizer, a config blob, a safetensors shard, or a binary record. Once the system has a manifest and file-level metadata, the same mechanism can be extended to model-aware chunking.

The architecture has four layers:

1. **Byte layer**: arbitrary bytes are the primitive object.
2. **Block layer**: files are split into fixed-size blocks.
3. **Codec layer**: each block is encoded with the best available codec or stored raw.
4. **Manifest layer**: the archive stores all metadata required for exact restoration.

This layering is the closest thing in the current repo to a novel algorithm. It is not one exotic transform; it is a disciplined decision system over multiple transforms, with exact metadata tracking and a no-regret fallback.

5.1 MiddleOut Lattice algorithm

For each block B_k in file f:

1. compute candidate encodings under a set of codecs $\mathcal{C}$ plus a raw option;
2. measure the stored size of each candidate;

3. choose the smallest candidate;

4. store the chosen codec identifier and per-block hashes in the manifest.

If the total archive is not smaller than the original file, the system stores the file raw instead of forcing compression. This is the key "don't make it worse" rule the user requested.

6 Archive format and exact reconstruction

Let a file f be partitioned into blocks $B_1, \ldots, B_K$. For each block we store metadata:

$$M_k = (\text{codec}_k,\ |B_k|,\ |\hat{B}_k|,\ h_k),$$

where $|B_k|$ is the raw size, $|\hat{B}_k|$ is the encoded size, and h_k is a SHA-256 digest of the raw block.

The archive contains:

- a magic prefix,
- a JSON manifest,
- the concatenated encoded payloads.

The exact reconstruction property follows by checking that each decoded block matches its recorded size and hash, and that the concatenation of blocks hashes to the file-level digest.

Theorem 1 (Exact reconstruction)**.** *If every block is encoded using a lossless codec and every stored hash is verified during decoding, then the archive decodes to the original file exactly.*

Proof. Each block codec is lossless by assumption, so decoding a stored block returns the original block bytes. Hash verification ensures no corruption or metadata mismatch has occurred. Concatenating all decoded blocks in order reconstructs the original file byte-for-byte. □

7 Qwen template benchmark

To test the architecture against a concrete model family, we used Qwen2.5-0.5B repository files as a template benchmark. The benchmark files were:

- `config.json`
- `generation_config.json`
- `merges.txt`
- `tokenizer.json`
- `tokenizer_config.json`
- `vocab.json`

File	Best codec	Original bytes	Compressed bytes	Ratio
config.json	zlib	681	475	1.4337
generation_config.json	zlib	138	227	0.6079
merges.txt	bz2	1671839	525417	3.1819
tokenizer.json	lzma	7031645	1378629	5.1005
tokenizer_config.json	zlib	7228	1320	5.4758
vocab.json	lzma	2776833	751481	3.6951

Table 1: Representative Qwen2.5-0.5B template benchmark results.

The measured compressibility varies strongly by file type. Text-heavy and JSON-heavy artefacts compress well. Tiny config files may lose to metadata overhead. That is exactly what an honest benchmark should show.

The strongest observed ratio in the template benchmark is 5.4758x on `tokenizer_config.json`. Importantly, that is a small metadata file, not the main weight shard. So it is evidence of a good archive strategy, not proof of universal ultra-compression on full model weights.

8 Results and analysis

The results support three claims.

First, exact compression is easy to preserve when the system works at the file level with checksums. Second, the best codec depends on the file type; there is no one-codec-fits-all answer. Third, selective fallback is essential. A compressor that never gets worse than raw storage is strictly better in practice than one that occasionally expands data while pretending to be clever.

The observed benchmark also shows that the user's intuition about separate compression per file is correct. Different files within the same model repository have different redundancy profiles. Tokeniser files, vocab files, and config files often exhibit repeated structure and textual regularity. Weight shards are likely to be much closer to the entropy limit and require a more specialised treatment.

9 Toward tensor-aware codecs

If the goal shifts from repository files to actual floating-point weight tensors, generic codecs become insufficient. A real weight compressor likely needs some combination of:

1. byte-plane or bit-plane decomposition,
2. exponent/mantissa separation,
3. blockwise transforms,
4. residual coding,
5. model-architecture-specific packing.

For a floating-point tensor W, one may decompose each scalar into sign, exponent, and mantissa substreams. If those substreams have very different entropy profiles, compressing them jointly is suboptimal. A more principled approach is to treat them as separate sources:

$$W \mapsto (S, E, M),$$

where S is the sign stream, E is the exponent stream, and M is the mantissa stream. Then the total code length can be approximated by

$$L(W) \approx L(S) + L(E) + L(M) + L_{\text{overhead}}.$$

If the streams are correlated, the optimal code should exploit that correlation either by conditioning or by transform design.

9.1 Blockwise residual coding

Suppose a tensor block B admits a structured predictor $\hat{B}$ with residual $R = B - \hat{B}$. If the residual has lower entropy than the original block, then coding the residual can improve compression. This suggests a natural research direction: learn or derive predictors that are cheap to compute and preserve exact reconstruction after residual decode.

9.2 Why architecture matters

Weights in modern transformers are not iid noise. They are shaped by layer norms, initialization schemes, optimisers, shared vocabularies, and architectural symmetry. A codec that ignores these facts can still compress files, but may leave a lot of signal on the table. A codec-aware tensor packer should exploit these correlations without sacrificing exactness.

10 Limitations

There are several honest limitations.

- The current benchmark is on repository files, not full weight shards.
- The current implementation is not yet a specialised tensor codec.
- Universal 2x lossless compression for every LLM checkpoint is not established.
- A file that is already high entropy may not compress at all.
- Any paper claiming universal gains without a failure analysis is probably lying.

These limitations are not embarrassing; they define the research frontier.

11 Conclusion

MiddleOut Lattice shows that exact compression of model artefacts can be made selective, manifest-backed, and model-agnostic at the file level. The Qwen template benchmark demonstrates that

compressibility is highly file dependent, which is exactly why a per-file and per-block architecture is the right direction.

The next step is not to promise a miracle. It is to build a real tensor-aware codec, evaluate it on actual checkpoints, and study which parts of the model are structurally compressible. If that succeeds, a true ultra-compression system may emerge. If it fails, we will at least have learned where the entropy lives.

A Derivations

A.1 Compression ratio

Let S_o and S_c denote original and compressed size. Then

$$R = \frac{S_o}{S_c}.$$

A.2 Entropy lower bound

For a discrete distribution p, the Shannon entropy is

$$H(p) = -\sum_i p_i \log_2 p_i.$$

Any lossless coding scheme has expected length at least the entropy rate up to coding overhead.

A.3 Expected blockwise length

If a file is partitioned into independent blocks B_k, then the expected code length of a blockwise compressor is approximately

$$\mathbb{E}[L(F)] = \sum_k \mathbb{E}[L(B_k)] + \mathbb{E}[L_{\text{manifest}}].$$

The manifest overhead is negligible for large files but can dominate tiny files, which explains why some small JSON files may expand instead of shrink.

A.4 Exactness condition

A codec is exact if every stored block $\hat{B}_k$ satisfies

$$D(\hat{B}_k) = B_k$$

and the file-level reconstruction is the concatenation of the decoded blocks in the original order.