

A Discrete Weight Language for Large Language Models: Compressing Gemma 4 31B with a 5-Step Ternary Route

Arman Aubakirov
Independent Research

April 2026

Abstract

We present Route B, a weight-compression scheme that treats every weight tensor of a pretrained language model as a sequence of short codes drawn from a shared, per-family code book. In effect, each scalar weight is spelled out by a 5-step ternary path through a learned ladder of amplitudes. Applied to `google/gemma-4-31b-it`, the method replaces the 60 GB bf16 weights of the 410 linear layers by 7.9 bits per weight (8-bit packed code plus a single fp32 scalar per layer, plus an optional 16-bit per-row scale), reducing the working memory of inference to about 31 GB with no change to the model I/O interface. A fused Triton matmul that pulls the 243-entry layer code book into shared memory reaches 37 tokens/s at batch size 4 on a single NVIDIA H200, without materializing the bf16 weight matrix. On 32k tokens of WikiText-2 at 2k context, the packed model is within +0.08 nats of the dense bf16 baseline (loss 6.906 vs. 6.829).

Adding a per-row scale reduces weight-reconstruction MSE by about $5\times$ and matmul output error by about $3\times$, but its end-to-end effect on Gemma 4 31B IT is strictly harmful from 1k context upward: the gap vs. dense widens from +0.25 nats at 512 context to +0.64 nats at 8k, even though per-layer activation-weighted MSE is locally better with row scale. A per-head prosody variant (one scalar ρ per attention head on q/k/v_proj, no row scale elsewhere) eliminates the regression entirely: it matches or beats the global-scale baseline at every context from 4k to 16k, and a brute-force end-to-end probe over the 3 locally rejected `k_proj` layers shows that one of them should actually be kept. Force-keeping block 37 `k_proj` yields strict wins over the global-scale baseline at all tested long contexts. We use these results to argue that weight-MSE, and even naive per-layer activation MSE, is an insufficient PTQ objective for long-context transformers. We also report the current external-evaluation status more conservatively than the accompanying research release: WikiText-2 and long-context loss remain the strongest settled results, while the new dense-vs-route downstream harness should still be read as preliminary.

Keywords. weight quantization, ternary codes, LLM compression, Gemma, discrete representations.

Artifacts. Project source package, paper sources, and reproducibility scripts: <https://github.com/AubakirovArman/discrete-weight-language-gemma4>. Full all-linear Hugging Face model export: <https://huggingface.co/armanibadboy/discrete-weight-language-gemma-4-31b-it-all-linear>.

1 Introduction

Production-grade LLMs at the 30B–70B scale remain memory-bound on mainstream GPUs. Even in bf16, Gemma 4 31B IT requires about 60 GB of weights alone, leaving little room for KV cache, activations, or batch parallelism on a single 80 GB card. Post-training quantization (PTQ) is the

pragmatic answer, and the literature spans from scalar INT8 all the way down to ternary and binary weights such as BitNet b1.58 [4]. The usual trade-off is familiar: the lower the bit budget, the larger the reconstruction error, and for sufficiently trained models the worse the downstream quality.

We revisit PTQ from a linguistic angle. Instead of asking how few bits can encode a scalar weight, we ask what is the shortest sentence in a small alphabet that spells the weight to within a chosen error. This is not just metaphor. It suggests a shared alphabet, variable word length, lightweight multiplicative prosody, and a fully discrete representation that can later be searched, diffed, or edited directly.

Into how few bits can I project a weight scalar?

What is the shortest sentence in a small alphabet that spells this weight to within a chosen error?

Concretely, this paper presents the simplest instance of this framework, which we call Route B: a fixed 5-step ternary word with a shared per-family \times depth-bucket ladder, packed into a single `uint8` per weight. We apply it to all 410 linear layers of Gemma 4 31B IT, provide a fused Triton kernel that decodes the LUT in shared memory, and release the public source package together with a 4-step reproduction recipe and a full all-linear Hugging Face export.

1.1 Contributions

1. **A concrete, reproducible 3-bit weight-compression recipe for Gemma 4 31B IT**, with all-linear coverage including the 10 shared-KV layers where `v_proj = None`, plus loading and reproduction code.
2. **A fused Triton matmul** for 5-step ternary weights. In the default `packed_triton_fused` mode it reaches 37.2 tok/s at about 31 GB resident memory on an H200 at batch size 4 and 64 new tokens. The slightly faster `packed_cached_decode` path only achieves that by re-materializing bf16 weights at about 60 GB.
3. **A per-row scale extension** that reduces weight-reconstruction MSE by about $5\times$ at a cost of one fp16 number per output row, together with a layerwise regression guard.
4. **An empirical diagnosis of where ternary codes fail and recover**, including the finding that a large improvement in weight-space MSE can still worsen end-to-end long-context loss, the per-head prosody correction that restores the regime, and an initial dense-vs-route downstream comparison harness for external validation.

1.2 Paper outline

Section 2 surveys related PTQ work and positions the discrete-language view. Section 3 gives the mathematical formulation of Route B and the per-row scale. Section 4 describes the runtime, including the Triton kernel and inference modes. Section 5 details the 4-step reproduction pipeline. Section 6 presents the empirical results: weight MSE, WikiText-2 loss, long-context behavior, memory, throughput, and the current downstream dense-vs-route status. Section 7 discusses limitations, and Section 8 outlines the broader discrete-weight-language program.

2 Related Work

Scalar PTQ. GPTQ [1], AWQ [2], SmoothQuant [3], and modern INT4/INT8 group-wise schemes are the current industrial default for serving large models. They spend 4–8 bits per weight, usually with per-group scales.

Ternary and binary weights. BitNet b1.58 [4] and earlier ternary/binary work show that models trained from scratch with low-bit discrete weights can match fp16 baselines surprisingly well. Post-hoc quantization of a pretrained instruction model to ≤ 2 bits without retraining remains substantially more fragile.

Look-up-table matmul. LUT-GEMM [5] and SqueezeLLM [6] materialize a small per-layer codebook and fuse dequantization into the matmul. This is the direct ancestor of our Triton kernel.

Product/vector quantization. QuIP [7] and later QuIP-style variants rotate weights into a more quantization-friendly basis before applying structured lattices. Route B is compatible with that idea; the discrete-sentence view is orthogonal to whether the alphabet lives in the original basis or in a rotated one.

Relative to these lines, Route B is fully post-training, stays near a 3-bit information budget, uses one ladder per family and depth bucket rather than per layer, and emphasizes a readable discrete encoding in which every scalar is spelled by an explicit word in $\{-1, 0, +1\}^5$.

3 Route B: Formal Description

Let $W \in \mathbb{R}^{O \times I}$ be a weight matrix and let $S = (s_1, s_2, s_3, s_4, s_5)$ be the ladder scales shared by the family of W .

3.1 Encoding

Given a real sample $x = W[n, k]$, the 5-step greedy ternary encoding produces a word $c = (c_1, \dots, c_5) \in \{-1, 0, +1\}^5$ such that

$$\begin{aligned} c_i &= \arg \min_{c \in \{-1, 0, +1\}} |r_{i-1} - c \cdot s_i|, \\ r_i &= r_{i-1} - c_i \cdot s_i, \\ r_0 &= x/\alpha, \end{aligned}$$

where $\alpha = \max_{n,k} |W[n, k]|$ is a per-matrix global scale. A small zero-bias $\zeta \in [0, 1]$ is used to prefer $c_i = 0$ when the residual is small; this reduces the number of non-zero steps and helps end-to-end quality.

3.2 Decoding and the layer code book

Decoding is

$$\widehat{W}[n, k] = \alpha \sum_{i=1}^5 c_i(n, k) s_i.$$

Because $c \in \{-1, 0, +1\}^5$, there are only $3^5 = 243$ possible words. We enumerate them once per layer, pre-multiply by α , and store the result as a 243-entry LUT. The full matrix is then described by the code matrix

$$\text{codes}[n, k] \in \{0, \dots, 242\},$$

which is the base-3 encoding of the 5-step word and is packed four or five weights per byte in the runtime.

3.3 Per-row scale

A single scalar α must cover the entire dynamic range of W , which under-resolves rows whose magnitude is far below the layer-level maximum. We therefore introduce a per-row scale

$$\begin{aligned}\alpha_n &= \max_k |W[n, k]|, \\ r_0(n, k) &= W[n, k]/\alpha_n, \\ \rho_n &= \alpha_n/\alpha,\end{aligned}$$

where the encoder is run on the row-normalized values and ρ_n is stored in fp16. At inference time,

$$\widehat{W}[n, k] = \rho_n \cdot \text{LUT}[\text{codes}[n, k]].$$

This adds one fp16 multiply per output scalar and can be fused into the same Triton kernel.

3.4 Ladder fitting

For each family and depth bucket we collect about 200k row-normalized samples and minimize

$$L(S) = \sum_{x \in \text{samples}} \left(x - \sum_i c_i(x; S) s_i \right)^2$$

by coordinate descent, starting from the balanced-ternary initialization $(1, 1/3, 1/9, 1/27, 1/81)$. The fitted ladder remains close to, but not exactly equal to, the balanced initialization.

4 Runtime

4.1 Packed storage

Base-3 packing fits five ternary digits in eight bits because $243 \leq 256$. The result is 8 bits/weight of storage, corresponding to about 3 bits/weight of information. Per-layer overhead is one fp32 global scale and, optionally, one fp16 per output row for the per-row scale vector.

4.2 Fused Triton matmul

The kernel loads the 243-entry per-layer LUT into shared memory once per block and then runs a standard K-sliced GEMM in which each packed code byte is dereferenced through the LUT on the fly. Because decoding is a single shared-memory lookup, no per-thread arithmetic depends on the ternary structure itself. The optional ρ_n multiplier is applied to the output accumulator before bias addition, so it does not increase the memory pressure inside the GEMM.

Mode	Matmul path	tok/s	Mem (GB)
<code>packed_decode_per_call</code>	decode full W each call	1.5	31
<code>packed_cached_decode</code>	decode once, cache bf16	~42	60
<code>packed_triton_fused</code>	LUT matmul	37.2	31
<code>packed_triton_autotuned</code>	LUT matmul, autotuned	26.9	31

Table 1: Runtime modes measured on H200, bf16, batch size 4, 64 new tokens.

4.3 Inference modes

`packed_triton_fused` is the default because it preserves the memory advantage of the packed representation while reaching about 88% of the throughput of the cached bf16 decode path.

4.4 Correctness

Output-level mean relative error between the fused kernel and a bf16 reference matmul is effectively zero in the kernel itself. The observed reconstruction error comes from the ternary encoding, not from the fused-matmul implementation. Against a dense bf16 reference, early-layer mean relative matmul error is typically 3–6% and drops to about 1.5% with per-row scale.

5 Pipeline

We ship the end-to-end pipeline as four independent scripts.

1. `01_fit_route_specs.sh` fits the 5-entry ladder for each family and depth bucket from about 200k samples per layer (about 2 minutes on one H200).
2. `02_export_packed_checkpoint.sh` encodes all 410 linear layers and packs the codes (about 40 minutes on one H200, and parallelizable across GPUs).
3. `03_add_row_scale.sh` recomputes the per-row scale with a regression guard (about 1 minute).
4. `04_validate_ppl.sh` measures dense-vs-packed WikiText-2 perplexity (about 5 minutes).

The runtime folder contains a `load_model.py` entry point that loads dense Gemma 4 31B IT and swaps every linear layer with its packed counterpart. The resulting `transformers` model remains a drop-in replacement at the API level.

6 Experiments

6.1 Setup

- Base model: `google/gemma-4-31B-it`, bf16.
- Hardware: NVIDIA H200 (141 GB HBM), CUDA 12.4.
- Packed coverage: all 410 linear layers inside the 60 transformer blocks, including the 10 shared-KV blocks where `v_proj = None` and `k_proj` doubles as `v_proj`.
- Ladder fitting: `family-depth`, coordinate descent, 200,000 samples per layer.

Variant	Mean relMSE	Best layer	Worst layer
Route B, global scale only	0.331%	0.083%	1.102%
Route B+ per-row scale	0.067%	0.014%	1.102%

Table 2: Average per-layer weight reconstruction error relative to $\sum W^2$.

Projection	Route Bonly	+ per-row scale	Reduction
k_proj	3.77%	1.17%	3.2×
q_proj	3.66%	1.16%	3.2×
v_proj	4.95%	1.88%	2.6×
o_proj	6.22%	2.01%	3.1×
gate_proj	4.70%	1.52%	3.1×
up_proj	3.59%	1.43%	2.5×
down_proj	5.90%	2.16%	2.7×
Mean	4.54%	1.57%	2.89×

Table 3: Mean relative output error on 8 test batches at layer 0.

6.2 Weight reconstruction

The per-row scale is kept in 381 of 410 layers. The 29 regressions are concentrated near the end of the stack and in `down_proj`, where the per-row maximum is a poor proxy for the typical row magnitude.

6.3 Output-level matmul error

6.4 WikiText-2 perplexity across context lengths

We evaluate on the WikiText-2 test split, fixing the total evaluated tokens at about 32k and sweeping the context length from 512 up to 16k. At 16k, the dense bf16 reference no longer leaves enough room for the full attention score tensor on a single H200, so we report packed-only numbers there.

The main findings are as follows.

1. **Route B without row scale is the strong short-path default.** The gap to dense remains moderate from 512 through 8k, and is smallest at 2k context.
2. **Per-row scale is monotonically worse from 1k upward.** This is the central negative result: much better weight-space and local matmul error does not imply lower end-to-end loss.
3. **Naive activation-aware selection remains misaligned.** The per-layer activation-aware criterion demotes only 9 of 381 layers and recovers less than 0.03 nats end-to-end. The damage is not local to a single layer output; it compounds through softmax and the residual stack.
4. **Per-head prosody eliminates the regression.** One scalar per attention head on q/k/v_proj preserves between-head dynamic-range adaptation while removing within-head row-wise multiplicative noise. The one-layer override on `layers.37.k_proj` improves the result further and beats the global-scale baseline at every tested long context.

Setting	Tokens	Variant	Loss	PPL	Δ vs. dense
8×512	4k	dense bf16	8.9196	7477.0	—
		Route B(global)	9.2035	9931.8	+0.2839
		Route B+ per-row scale	9.1666	9572.2	+0.2470
32×1024	32k	dense bf16	7.8438	2549.8	—
		Route B(global)	8.0771	3219.7	+0.2333
		Route B+ per-row scale	8.1734	3545.3	+0.3296
16×2048	32k	dense bf16	6.8292	924.4	—
		Route B(global)	6.9059	998.2	+0.0767
		Route B+ per-row scale	7.2812	1452.7	+0.4520

Table 4: Short-context WikiText-2 summary. The best settled short-path result is Route B without per-row scale at 2k context, within +0.08 nats of dense.

Setting	Tokens	Variant	Loss	PPL	Δ vs. dense
8×4096	32k	dense bf16	6.8528	948.2	—
		Route B(global)	7.0365	1137.5	+0.1837
		Route B+ per-row scale	7.3864	1612.2	+0.5336
		Route B+ act-aware select	7.3918	1620.9	+0.5390
4×8192	32k	dense bf16	7.3337	1530.7	—
		Route B(global)	7.5483	1897.3	+0.2146
		Route B+ per-row scale	7.9786	2919.7	+0.6449
		Route B+ act-aware select	7.9530	2846.9	+0.6193
2×16384	32k	Route B(global)	8.3255	4117.6	—
		Route B+ per-row scale	8.5604	5214.0	—
		Route B+ act-aware select	8.5390	5107.7	—
		Route B+ per-head prosody	8.3461	4213.9	+0.02 vs. baseline
		Route B+ per-head prosody + keep layers.37.k_proj	8.3172	4093.8	-0.008 vs. baseline

Table 5: Long-context WikiText-2 sweep. Per-head prosody and the single-layer override recover the long-context regime.

6.5 Memory and throughput

The single-batch latency penalty relative to the cached bf16 decode path is about $1.5\times$ in exchange for roughly a $2\times$ reduction in resident weight memory.

6.6 Preliminary downstream comparisons

Beyond WikiText-2, we built a common OpenAI-compatible serving path so that the dense baseline and the best current long-context Route B checkpoint (`checkpoint_perhead_force_k37.pt`) could be queried by the same harness. We report these numbers as early dense-vs-route comparisons rather than final benchmark claims. The completions-based path still underestimates absolute Gemma 4 31B IT quality relative to the official dense product sheet, GPQA remains parser-limited, and AIME remains prompt-limited in the current path.

Three observations matter.

1. **MMLU-Pro is the cleanest current external comparison.** On the capped sample harness, the packed model trails dense by about 9.3 points.

Context	Per-head prosody	Δ vs. dense	Δ vs. baseline
4096	6.9735	+0.1207	-0.0630
8192	7.5540	+0.2203	+0.0057
16384	8.3461	—	+0.0206
4096	6.9718 with keep k37	+0.1190	-0.0647
8192	7.5438 with keep k37	+0.2101	-0.0045
16384	8.3172 with keep k37	—	-0.0083

Table 6: Per-head prosody rows at the shorter long-context points.

Quantity	Dense bf16	Packed
Weight memory (410 linear layers)	60.1 GB	31.0 GB
Peak GPU memory (bs=4, 64 new tok)	~60 GB	58.3 GB
Decode throughput (bs=4, 64 new tok)	n/a	37.2 tok/s

Table 7: Memory and throughput in the default `packed_triton_fused` mode.

2. **HumanEval is roughly neutral at current scale.** The same harness shows a slight packed-model advantage.
3. **The harness itself is still incomplete.** A direct manual probe on the first AIME24 sample returned the correct answer 33 for both dense and Route B once prompted with a stricter final-answer cue. The current zeroes are therefore a benchmark-path artifact, not a clean model-quality measurement.

Separately, a full direct-HF MMLU-Pro evaluation for both dense and Route B was launched with `max_length=8192` and `max_gen_toks=64` while this TeX draft was being prepared. Those runs were still in progress at the time of writing, so we do not cite them here as completed results.

7 Limitations

- **Post-training only.** We do not fine-tune; Route B is evaluated strictly as a PTQ method on top of the public Gemma 4 31B IT weights.
- **Weight-only.** Activations and KV cache remain bf16. Reducing them is orthogonal to the present method.
- **Evaluation coverage.** The strongest settled quality table in this paper remains the WikiText-2 sweep. We now also have capped dense-vs-route comparisons on MMLU-Pro sample and HumanEval, but those are still regression-oriented harnesses rather than final leaderboard numbers. GPQA remains parser-limited, AIME remains prompt-limited, and the full direct-HF MMLU-Pro runs were still in progress at the time of writing.
- **Per-row scale ambiguity.** On Gemma 4 31B IT, per-row scale improves weight MSE but hurts long-context loss. The evidence now points away from naive per-layer activation-aware selection and toward trajectory-level or end-to-end criteria.
- **Shared-KV layers.** Ten transformer blocks use shared K/V structure. Our export handles them correctly, but does not jointly fit a K/V ladder, which likely leaves a small amount of performance on the table.

Benchmark	Dense baseline	Route B	Route - dense	Status
MMLU-Pro sample	0.5429	0.4500	-0.0929	usable relative regression
HumanEval	0.0854	0.0915	+0.0061	usable relative regression
GPQA Diamond sample	0.0625	0.0625	0.0000	parser-limited
AIME24 sample	0.0000	0.0000	0.0000	invalid under current prompt path
AIME25 sample	0.0000	0.0000	0.0000	invalid under current prompt path

Table 8: Current dense-vs-route downstream comparison under the shared capped harness.

8 A Discrete Weight Language Beyond Route B

8.1 Variable-depth words

Not every row of every layer needs all five ternary steps. Looking at the code histogram for Gemma 4 31B IT, the top 45 of the 243 codes cover 99% of all weights and the top 18 cover about 80%. This suggests two obvious follow-ups: variable-depth encoding that stops after 3 or 4 steps where possible, and a 7-bit truncated alphabet in which only the actually used codes are stored explicitly.

8.2 Per-row, per-column, and per-head prosody

The per-row scale is one-dimensional prosody on the output direction. A symmetric per-column scale would do the same on the input direction, and together they define a rank-2 multiplicative envelope around the discrete codes. For attention, however, the experiments in Section 6 show that per-head granularity is the right one for the current objective: it gives every head its own dynamic range while forcing rows *within* a head to share a scale, which removes the within-head multiplicative noise that compounds through softmax.

With this single change, we go from +0.53 nats at 4k with per-row scale to -0.06 nats at 4k with per-head prosody relative to the global-scale baseline. A one-layer end-to-end correction improves that further: force-keeping only `layers.37.k_proj` yields 6.9718 / 7.5438 / 8.3172 at 4k / 8k / 16k, beating the global-scale baseline at all three tested long-context points.

8.3 Adaptive alphabets

Instead of a single ladder shared by a family and depth bucket, one can fit per-row or per-group-of-32-row ladders at a small storage cost. Preliminary alternating-least-squares analysis suggests that moving from one ladder per family to one ladder per 32-row block can reduce MSE by another factor of 1.5–2 at well below 1% storage overhead.

8.4 Activation-aware selection and why the naive version fails

The failure mode of Section 6 is a direct consequence of optimizing the wrong objective. We implemented the obvious local fix, a per-layer activation-weighted MSE criterion,

$$L_{\text{act}}(\text{layer}) = \frac{\sum_x \|(W - \widehat{W})x\|^2}{\sum_x \|Wx\|^2},$$

computed on four calibration batches at 2k context. The selector demotes only 9 of 381 row-scale layers and recovers less than 0.03 nats end-to-end. The criterion correctly detects that local output error is locally lower with row scale, but the long-context damage accumulates through the softmax

and residual stream rather than through any single layer output. A correct criterion therefore needs to be trajectory-level or genuinely end-to-end.

8.5 Joint attention fit

For the 10 shared-KV layers where `v_proj = None` and `k_proj` doubles as `v_proj`, a joint objective over (q, k, o) that preserves $\text{softmax}(qk^\top)vo$ is a natural next step. The repository already contains the primitives for this, but we do not yet ship a joint-fit checkpoint.

8.6 Towards QAT-grade Route B

All of the extensions above are still post hoc. A natural endpoint is a short distillation phase on top of the packed model in which gradients flow only through the prosody parameters and the small set of actually used code-book entries. That would preserve the discrete structure while targeting the last remaining end-to-end gap directly.

8.7 What a weight language is actually good for

A ternary code is a finite discrete object. Unlike a bf16 weight, it can be compared, searched, or edited directly. We therefore view compression not only as a deployment benefit, but also as a route toward a manipulable discrete representation of model weights. In the long run, that may be the more important contribution of the program.

9 Conclusion

We presented Route B, a 3-bit weight-compression scheme with a fused Triton matmul, applied end to end to Gemma 4 31B IT. The method reduces weight memory from roughly 60 GB to 31–32 GB and preserves 37 tok/s at batch size 4 on a single H200. On WikiText-2 at 2k context, the packed model is within +0.08 nats of the dense bf16 baseline without any fine-tuning; across 1k–16k context, the plain global-scale variant stays within about +0.08 to +0.21 nats.

The main negative result is just as important as the positive one. Per-row scale reduces weight-space MSE by roughly $5\times$ and output matmul error by about $2.9\times$, yet makes the long-context model strictly worse from 1k context upward. This directly refutes weight-MSE as a sufficient PTQ objective for long-context transformers, and shows that standard per-layer activation-aware selection remains too local.

The corresponding positive result is that switching the attention prosody from per-row to per-head eliminates this regression. With per-head prosody and a one-layer end-to-end override on `layers.37.k_proj`, the packed model beats the global-scale baseline at all tested long-context points. That is the strongest evidence so far that the discrete-language program can move beyond a compression proof-of-concept into a genuinely competitive low-bit weight representation.

Outside WikiText-2, the present release should still be read as a strong compression and long-context evaluation paper with preliminary downstream coverage rather than a complete benchmark-paper replacement for the official Gemma evaluation sheet. The current dense-vs-route harness already gives a useful early signal on MMLU-Pro sample and HumanEval, but GPQA and AIME remain harness-limited, and the first full direct-HF MMLU-Pro runs were still in progress when this version was prepared.

A Reproducibility Checklist

- Base model: `google/gemma-4-31B-it` in `bf16`, obtained under Gemma’s public license.
- Hardware: NVIDIA H200 (141 GB HBM). The packed runtime is intended for any GPU with at least about 40 GB of device memory.
- Libraries: `torch==2.5.1`, `triton==3.1.0`, and `transformers==5.5.0`; full pins are listed in the release.
- Seeds: ladder fitting uses seed 0; reported evaluations are deterministic with `do_sample=False`.
- Scripts: the four pipeline stages can be run individually or via `reproduce/run_all.sh`. The total budget is about 50 minutes on one H200.

B File Layout

The research release is organized as follows.

```
archiv.org/
|- ARXIV_SUBMISSION_CHECKLIST.md
|- paper.md
|- paper_ru.md
|- main.tex
|- prepare_arxiv_bundle.sh
|- README.md
|- runtime/
|   |- load_model.py
|   |- generate.py
|   |- benchmark.py
|   |- openai_server.py
|   |- requirements.txt
|   ‘- ternary_route/
|- reproduce/
|   |- 01_fit_route_specs.sh
|   |- 02_export_packed_checkpoint.sh
|   |- 03_add_row_scale.sh
|   |- 04_validate_ppl.sh
|   ‘- run_all.sh
|- benchmarks/
|   ‘- results.md
‘- sections/
    |- 00_abstract.tex
    |- 01_introduction.tex
    |- 02_related_work.tex
    |- 03_routeb_formal.tex
    |- 04_runtime.tex
    |- 05_pipeline.tex
    |- 06_experiments_a.tex
```

```
| - 06_experiments_b.tex
| - 07_limitations.tex
| - 08_beyond_routeb_a.tex
| - 08_beyond_routeb_b.tex
| - 09_conclusion.tex
| - appendix_a.tex
| - appendix_b.tex
'- references.tex
```

The Markdown paper remains the editable narrative draft, while `paper_ru.md` is a full Russian reading/editing version. `main.tex` and the `sections/` directory remain the English arXiv-oriented LaTeX source version used for the upload bundle. The source-only code release vendors `runtime/ternary_route/` and omits `.pt` checkpoints; packed weights are distributed separately or rebuilt via `reproduce/`.

References

- [1] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers. arXiv:2210.17323, 2022.
- [2] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, and Chuang Gan. AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. arXiv:2306.00978, 2023.
- [3] Guangxuan Xiao et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438, 2022.
- [4] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits. arXiv:2402.17764, 2024.
- [5] Gunho Park, Baeseong Park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, and Byeongwook Kim. LUT-GEMM: Quantized Matrix Multiplication via Lookup Tables for Efficient Inference in Large-Scale Generative Language Models. arXiv:2206.09557, 2022.
- [6] Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-Sparse Quantization. arXiv:2306.07629, 2023.
- [7] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-Bit Quantization of Large Language Models With Guarantees. arXiv:2307.13304, 2023.