Computation and Performance Studies
Volume 1 · Number 1 · 2026

Packed Weight Bytes and Decode Time in a NumPy Prototype

Cameron Peterson[1] · Andrew King[2] · Evan Anderson[3]

[1] France · [2] United States · [3] Australia

Original publication: October 28, 2026 | Article 695

Abstract

Four-bit packing reduced selected weight payloads from 2,359,296 to 626,688 bytes, including one float32 scale per 128 weights. In the same local session, reconstructing packed weights to float32 took modestly longer than converting source float16 payloads: the geometric mean ratio of matrix-median times was 1.064368. Both paths were measured after warmup with balanced execution order. Exact checks confirmed that unpacking reproduced the declared rounded values, rather than the original source weights. The result concerns a transparent memory-resident NumPy prototype. It measures neither a whole-model compression ratio nor inference acceleration through optimized low-bit kernels, and its timing cannot be assumed invariant across machines.

An actual payload and reconstruction benchmark for twenty-four public language-model matrices on one CPU session.

Packed Weight Bytes and Decode Time in a NumPy Prototype

An actual payload and reconstruction benchmark for twenty-four public language-model matrices on one CPU session.

Abstract

Four-bit packing reduced selected weight payloads from 2,359,296 to 626,688 bytes, including one float32 scale per 128 weights. In the same local session, reconstructing packed weights to float32 took modestly longer than converting source float16 payloads: the geometric mean ratio of matrix-median times was 1.064368. Both paths were measured after warmup with balanced execution order. Exact checks confirmed that unpacking reproduced the declared rounded values, rather than the original source weights. The result concerns a transparent memory-resident NumPy prototype. It measures neither a whole-model compression ratio nor inference acceleration through optimized low-bit kernels, and its timing cannot be assumed invariant across machines.

1. Storage savings and reconstruction work need separate measurements

A small numeric weight payload can reduce stored input bytes while requiring extra work before a computation consumes the values. If the next stage expects float32, a packed representation must be unpacked and scaled, while a float16 source must be converted. Those operations depend on implementation and hardware. A nominal bit width alone cannot establish which path is faster or how the tradeoff affects an application's full inference pipeline.

We ask how a transparent four-bit packing prototype compares with source-float16 conversion in payload size and memory-resident reconstruction time. The storage quantity is directly countable. The time quantity is measured for a specified implementation and session. We also verify whether the decoder returns the exact values defined by the rounding rule. These are complementary checks: a small payload, a fast call, and a correct reconstruction of its intended representation are different claims.

Compression work represented by Wang et al., 2026 motivates preserving those distinctions when discussing efficiency. We do not implement that framework or evaluate its speed. The present code packs selected public matrices and reconstructs them with ordinary NumPy array operations. It supplies a small inspectable resource experiment, without claiming a competitive inference system or treating a prototype decoder as representative of every optimized quantization implementation.

2. Source matrices and the declared rounded representation

The weights come from the final standard-Pile checkpoint associated with Biderman et al., 2023. We directly parse its numeric safetensors header and float16 byte ranges. The selected set contains four attention or multilayer-perceptron projection matrices in each of six layers, giving twenty-four matrices and 1,179,648 weights. No image, text, or activation data are needed to form the payloads. Embeddings, biases, normalization parameters, and other checkpoint components remain outside the selected storage comparison.

Within each matrix, row-major groups contain 128 values. We promote source values to float32, divide each group's largest absolute value by seven to form a float32 scale, and round divided values to codes between minus seven and seven. The scales are positive. Reconstructed values are the integer codes multiplied by those float32 scales. The operation is lossy relative to the released float16 parameters, even though the subsequent decoding check can be exact relative to this declared rounded reference.

The broader numerical methods in Frantar et al., 2022, Lin et al., 2023, Xiao et al., 2022, Dettmers et al., 2022, and Dettmers et al., 2023 use additional calibration, representation, or implementation choices. None is reproduced by the simple maximum-scale procedure. Their reported accuracy or throughput cannot be transferred to this prototype. We retain the narrow numerical convention so that byte accounting and decoder correctness refer to a fully specified object, rather than a generic label such as four-bit compression.

Numerical issues discussed in Higham, 2002 and Goldberg, 1991 motivate preserving the dtype, rounding order, and scale storage precision. A decoder can faithfully reconstruct a quantized reference while that reference differs materially from the original model. We do not equate exact decoding with lossless compression of source weights, and no downstream language score is inferred from an array-equality test. Prediction effects require their own model execution.

3. Real packing and explicit payload boundaries

The signed code is offset by seven into the range zero through fourteen. Two such codes occupy the low and high nibbles of each stored byte. Group scales follow the code bytes as little-endian float32 values. Every selected matrix has an even number of values divisible by 128, so the implementation needs no unused terminal half-byte. One nominal code remains unused within the four-bit representation.

The packed payload therefore contains four code bits per weight and thirty-two scale bits per 128 weights, giving 4.25 payload bits per weight. The source payload contains sixteen bits per weight. Across the selected matrices, sizes are 626,688 and 2,359,296 bytes respectively. The packed payload is 26.5625 percent of the float16 payload. This ratio follows directly from the selected code and scale format; the experiment does not present that arithmetic as an unexpected empirical discovery.

The saved byte arrays are not self-describing model files. Matrix identity and shape remain in the external source metadata and analysis record. Headers, file alignment, container structures, checksums, and the unchanged checkpoint tensors are outside the payload count. A whole-model file could therefore have a substantially different reduction. We do not multiply the selected payload ratio by the complete checkpoint size and describe that as an observed archive result.

The raw payloads are actually written to local files and assigned hashes. The packed decoder separates nibbles, converts their offset codes to float32, reshapes into groups, and multiplies by the stored scales. For every matrix, decoded values are exactly equal to the independently constructed rounded reference array. The float16 path is separately checked to reconstruct source values promoted to float32. These correctness checks use different intended references because the two representations have different fidelity.

4. What the timer includes

The implementation uses NumPy 2.5.4, in the array-programming context described by Harris et al., 2020, with Python 3.12.14 on an AMD64 processor reported as Family 25 Model 97. The performance counter has a reported resolution of 0.0000001 seconds. Both paths start from bytes already in memory and return float32 values. The measured call includes array conversion and output allocation. It excludes disk reads, source-header parsing, model execution, and any compilation stage.

Each matrix and method receives ten untimed warmup calls, followed by sixty timed observations. Within a repetition the method order alternates, yielding thirty measurements in each ordering. We summarize each method by its matrix-specific median duration. Timing a memory-resident payload repeatedly can

favor cache reuse; this is part of the local design rather than an approximation to cold loading from storage. Background workstation load is not independently controlled.

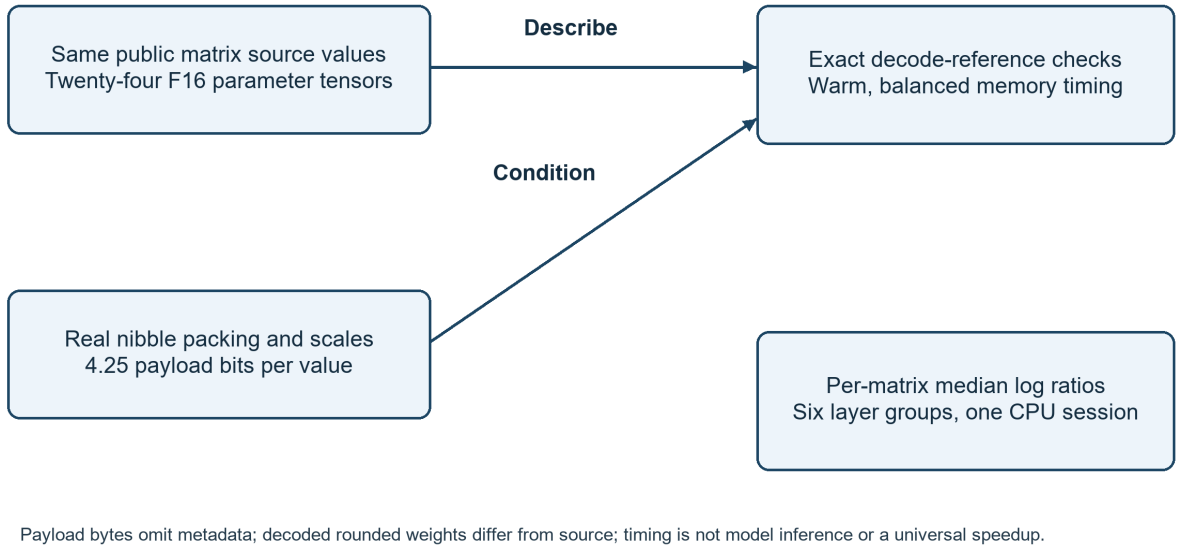


Figure 1. Source and packed bytes enter different float32 reconstruction paths; exact reference checks precede balanced warm timing and matrix-level summaries.

The hardware-cost perspective in Williams et al., 2009 emphasizes that an application's limiting resource may differ from the one measured here. Attention and serving methods in Dao et al., 2022 and Kwon et al., 2023 optimize other pipeline components. The decoding setting in Leviathan et al., 2022 is also distinct from unpacking stored weights. No such implementation is benchmarked by these array calls. Their relevance is to why a local reconstruction duration cannot alone supply a full language-model speedup estimate.

5. A matrix-level timing comparison

For every matrix we divide packed-decoder median duration by float16-conversion median duration and take the natural logarithm. An intercept-only ordinary least-squares model estimates the equal-matrix mean log ratio. There are twenty-four rows. Exponentiating the mean gives a geometric mean ratio, which differs from dividing arithmetic averages of the two sets of durations. The statistic weights matrix locations equally rather than weighting them by parameter count or payload bytes.

Covariance clusters by the six transformer layers, with a finite-sample correction and a t reference using five degrees of freedom. The grouping perspective in Cameron and Miller, 2015 motivates accounting for related matrices within a layer. Heteroskedasticity-consistent estimation in White, 1980 alone would not describe that grouping. These intervals summarize a conditional across-matrix comparison; six layers in one model and one workstation session are not six independent machines or independent model-training replications.

Table 1. Mean log packed-Q4 versus source-F16 reconstruction-time ratio, using per-matrix medians.

Model	Term	Estimate (SE)	95% CI	N
M1	Mean outcome/contrast	0.0624 (0.0039)	[0.0523, 0.0725]	24
M1: fit	R-squared = 0.0000	-	-	24

Model labels: M1 = Mean log packed-Q4 versus source-F16 decode time.

The mean-only fit has zero R-squared, without an implication that timing is unexplained in a broader performance model. The sixty within-matrix repetitions are not promoted to independent matrix observations. They support the median summaries used here. A separate repeated-session experiment would be required to quantify session-to-session timing variability or test whether the same ratio holds under changing system load.

6. A modest measured reconstruction overhead

The mean log ratio is 0.062380909, with layer-clustered standard error 0.003924130 and interval 0.052293611 to 0.072468207. Exponentiating the coefficient gives a ratio of 1.064367694: approximately 6.4 percent greater duration for the packed reconstruction under this equal-matrix summary. The interval is conditional on the measured session and the matrix-group interpretation. It should not be promoted into a universal penalty attached to four-bit weights.

Arithmetic averages of the matrix medians are 73,929.167 nanoseconds for float16 conversion and 76,922.917 nanoseconds for packed reconstruction, approximately 73.93 and 76.92 microseconds. Their ratio differs from the reported geometric mean because matrices have different sizes and timing ratios. We retain the definitions rather than switching summaries to enlarge or diminish the observed overhead. Both are descriptive quantities from the actual recorded calls.

The payload saving and local timing result are compatible. The packed path reads fewer stored code bytes but performs nibble extraction, offset conversion, and scaling to produce the same output dtype. It also allocates temporary arrays. The float16 path performs a different conversion operation and preserves different source fidelity. We have not isolated which instruction, cache effect, allocation, or dtype conversion accounts for the measured difference, so none is presented as a confirmed mechanism.

The published tool context in ggml-org contributors, 2026 illustrates a different kind of evidence involving actual quantized-model throughput. Those external numbers are not merged with this timing table. Runtime systems represented by Paszke et al., 2019 and Wolf et al., 2019 are also outside the measured decoder call. Merely loading a model through a separate library would not make these NumPy payload timings equivalent to an end-to-end comparison through that runtime.

7. Correctness, fidelity, and resource scope

Exact decoded equality verifies the implementation against the intended code-and-scale reference. It does not establish exact restoration of the source weights. The parameter-space reconstruction errors are retained in the data, but this report does not measure generated text, next-token loss, or task correctness. Measurement reasoning in Carroll et al., 2006 cautions against substituting one endpoint for another simply because both are numerical.

Scoring principles in Gneiting and Raftery, 2007 reinforce that performance claims require an outcome matched to the question. A decode call, a stored payload, and a language-model answer can each be evaluated correctly while leading to different practical conclusions. The benchmark therefore avoids a composite efficiency score that would combine bytes and nanoseconds with an unstated value judgment. A real deployment would need actual loading, execution, memory, and quality measurements under its intended workload.

Systems concerns in Sculley et al., 2015 are relevant to external metadata and integration assumptions. The prototype relies on known matrix shapes and a fixed grouping convention, so its raw files cannot be consumed as arbitrary standalone model files. An operational codec would need versioning, validation,

and an execution path for its reconstructed values. These are additional implementation requirements, not already measured benefits inferred from the saved byte arrays.

8. Limits of one-session timing evidence

Influence reasoning in Cook, 1977 motivates inspecting unusually slow matrix measurements, but no matrix is removed. Medians reduce sensitivity to individual timed-call outliers without establishing a controlled hardware experiment. Validation concerns in Roberts et al., 2017 and selection bias in Varma and Simon, 2006 also matter if the same timing session is used to tune implementation choices and then presented as independent final evidence. We retain the declared implementation and balanced schedule rather than searching multiple versions for a preferred ratio.

Interpretation under Wasserstein and Lazar, 2016 emphasizes the modest conditional effect and its measurement scope. Multiplicity considerations in Benjamini and Hochberg, 1995 and Benjamini and Yekutieli, 2001 would apply to a broader search over code paths, group sizes, precisions, and devices. The current comparison has one group size and two explicitly defined reconstruction paths. Its interval cannot compensate for an unmeasured system bottleneck or guarantee that another implementation will favor the same path.

Timing also requires a different reproduction standard from byte packing. Numeric payloads and their decoded reference values can be reproduced exactly under the declared format. Nanosecond measurements can change with software, hardware, allocation state, and background work. A successful rerun should preserve the protocol and report its new timing results, not be required to reproduce an old timing CSV byte for byte. Keeping these evidence types separate is necessary for honest performance reporting.

9. Report the tradeoff at the measured boundary

The completed prototype packs selected weight payloads to 4.25 bits per value and reconstructs their declared rounded values correctly. In one warmed CPU session, that reconstruction is modestly slower than source-float16 conversion under the reported matrix-level summary. These observations establish a local storage-and-reconstruction tradeoff, without determining the speed, fidelity, or size of a complete compressed inference system.

Documentation under Gebru et al., 2021 records source tensor identities, code offsets, endianness, scales, excluded metadata, timer resolution, warmups, and execution order. Preservation following Wilkinson et al., 2016 retains actual packed payloads, hashes, numerical checks, and timing repetitions. They permit the implementation and its measurements to be audited. A general compression-efficiency recommendation requires independent workload and hardware evidence beyond this transparent NumPy experiment.

References

1. Higham, Nicholas J.. (2002). Accuracy and Stability of Numerical Algorithms: Second Edition. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718027.
2. Goldberg, David. (1991). What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys. https://doi.org/10.1145/103162.103163.
3. Dettmers, Tim; Svirschevski, Ruslan; Egiazarian, Vage; Kuznedelev, Denis; Frantar, Elias; Ashkboos, Saleh; Borzunov, Alexander; Hoefler, Torsten; Alistarh, Dan. (2023). SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. https://arxiv.org/abs/2306.03078.
4. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.

5. Cameron, A. Colin, and Douglas L. Miller. “A Practitioner’s Guide to Cluster-Robust Inference.” *Journal of Human Resources*, vol. 50, no. 2, 2015, pp. 317-372. https://doi.org/10.3368/jhr.50.2.317.

6. Paszke, Adam; Gross, Sam; Massa, Francisco; Lerer, Adam; Bradbury, James; Chanan, Gregory; Killeen, Trevor; Lin, Zeming; Gimelshein, Natalia; Antiga, Luca; Desmaison, Alban; Köpf, Andreas; Yang, Edward; DeVito, Zach; Raison, Martin; Tejani, Alykhan; Chilamkurthy, Sasank; Steiner, Benoit; Fang, Lu; Bai, Junjie; Chintala, Soumith. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. https://arxiv.org/abs/1912.01703.

7. Wang, Junhao; Xu, Guoxi; Tong, Huimin; Xu, Gaochao; Chen, Yiwei. (2026). Dynamic Fusion for Large Language Models Compression. ICASSP 2026, 4711-4715. https://doi.org/10.1109/icassp55912.2026.11464073.

8. Frantar, Elias; Ashkboos, Saleh; Hoefler, Torsten; Alistarh, Dan. (2022). GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. https://arxiv.org/abs/2210.17323.

9. Wilkinson, Mark D.; Dumontier, Michel; Aalbersberg, IJsbrand Jan; Appleton, Gabrielle; Axton, Myles; Baak, Arie; Blomberg, Niklas; Boiten, Jan-Willem; da Silva Santos, Luiz Bonino; Bourne, Philip E.; Bouwman, Jildau; Brookes, Anthony J.; Clark, Tim; Crosas, Mercè; Dillo, Ingrid; Dumon, Olivier; Edmunds, Scott; Evelo, Chris T.; Finkers, Richard; Gonzalez-Beltran, Alejandra; Gray, Alasdair J.G.; Groth, Paul; Goble, Carole; Grethe, Jeffrey S.; Heringa, Jaap; ’t Hoen, Peter A.C; Hooft, Rob; Kuhn, Tobias; Kok, Ruben; Kok, Joost; Lusher, Scott J.; Martone, Maryann E.; Mons, Albert; Packer, Abel L.; Persson, Bengt; Rocca-Serra, Philippe; Roos, Marco; van Schaik, Rene; Sansone, Susanna-Assunta; Schultes, Erik; Sengstag, Thierry; Slater, Ted; Strawn, George; Swertz, Morris A.; Thompson, Mark; van der Lei, Johan; van Mulligen, Erik; Velterop, Jan; Waagmeester, Andra; Wittenburg, Peter; Wolstencroft, Katherine; Zhao, Jun; Mons, Barend. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data. https://doi.org/10.1038/sdata.2016.18.

10. Varma, Sudhir and Richard Simon. “Bias in error estimation when using cross-validation for model selection.” *BMC Bioinformatics*, vol. 7, no. 1, 2006. https://doi.org/10.1186/1471-2105-7-91.

11. Xiao, Guangxuan; Lin, Ji; Seznec, Mickael; Wu, Hao; Demouth, Julien; Han, Song. (2022). SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. https://arxiv.org/abs/2211.10438.

12. ggml-org contributors, . (2026). llama.cpp quantize tool: published Llama-3.1-8B throughput tables. Captured 7 November 2026, local snapshot hash recorded. https://github.com/ggml-org/llama.cpp/blob/master/tools/quantize/README.md.

13. Roberts, David R., et al. “Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure.” *Ecography*, vol. 40, no. 8, 2017, pp. 913-929. https://doi.org/10.1111/ecog.02881.

14. Gebru, Timnit, et al. "Datasheets for datasets" *Communications of the ACM*, 2021. https://doi.org/10.1145/3458723.

15. Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierric; Rault, Tim; Louf, Rémi; Funtowicz, Morgan; Davison, Joe; Shleifer, Sam; von Platen, Patrick; Ma, Clara; Jernite, Yacine; Plu, Julien; Xu, Canwen; Scao, Teven Le; Gugger, Sylvain; Drame, Mariama; Lhoest, Quentin; Rush, Alexander M.. (2019). HuggingFace's Transformers: State-of-the-art Natural Language Processing. https://arxiv.org/abs/1910.03771.

16. Biderman, Stella; Schoelkopf, Hailey; Anthony, Quentin; Bradley, Herbie; O'Brien, Kyle; Hallahan, Eric; Khan, Mohammad Aflah; Purohit, Shivanshu; Prashanth, USVSN Sai; Raff, Edward; Skowron, Aviya; Sutawika, Lintang; van der Wal, Oskar. (2023). Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. https://arxiv.org/abs/2304.01373.

17. Benjamini, Yoav and Yosef Hochberg. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 57, no. 1, 1995, pp. 289-300. https://doi.org/10.1111/j.2517-6161.1995.tb02031.x.

18. Dettmers, Tim; Lewis, Mike; Belkada, Younes; Zettlemoyer, Luke. (2022). LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. https://arxiv.org/abs/2208.07339.

19. Harris, Charles R.; Millman, K. Jarrod; van der Walt, Stéfan J.; Gommers, Ralf; Virtanen, Pauli; Cournapeau, David; Wieser, Eric; Taylor, Julian; Berg, Sebastian; Smith, Nathaniel J.; Kern, Robert; Picus, Matti; Hoyer, Stephan; van Kerkwijk, Marten H.; Brett, Matthew; Haldane, Allan; del Río, Jaime Fernández; Wiebe, Mark; Peterson, Pearu; Gérard-Marchant, Pierre; Sheppard, Kevin; Reddy, Tyler; Weckesser, Warren; Abbasi, Hameer; Gohlke, Christoph; Oliphant, Travis E.. (2020). Array programming with NumPy. Nature. https://doi.org/10.1038/s41586-020-2649-2.

20. Williams, Samuel; Waterman, Andrew; Patterson, David. (2009). Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM. https://doi.org/10.1145/1498765.1498785.

21. Leviathan, Yaniv; Kalman, Matan; Matias, Yossi. (2022). Fast Inference from Transformers via Speculative Decoding. https://arxiv.org/abs/2211.17192.

22. White, Halbert. “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity.” *Econometrica*, vol. 48, no. 4, 1980, pp. 817-838. https://doi.org/10.2307/1912934.

23. Gneiting, Tilmann; Raftery, Adrian E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association. https://doi.org/10.1198/016214506000001437.

24. Sculley, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, vol. 28, 2015. https://papers.nips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html

25. Carroll, Raymond J., David Ruppert, Leonard A. Stefanski, and Ciprian M. Crainiceanu. “Measurement Error in Nonlinear Models.” 2006. https://doi.org/10.1201/9781420010138.

26. Dao, Tri; Fu, Daniel Y.; Ermon, Stefano; Rudra, Atri; Ré, Christopher. (2022). FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. https://arxiv.org/abs/2205.14135.

27. Wasserstein, Ronald L.; Lazar, Nicole A.. (2016). The ASA Statement on p -Values: Context, Process, and Purpose. The American Statistician. https://doi.org/10.1080/00031305.2016.1154108.

28. Lin, Ji; Tang, Jiaming; Tang, Haotian; Yang, Shang; Chen, Wei-Ming; Wang, Wei-Chen; Xiao, Guangxuan; Dang, Xingyu; Gan, Chuang; Han, Song. (2023). AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. https://arxiv.org/abs/2306.00978.

29. Benjamini, Yoav and Daniel Yekutieli. “The control of the false discovery rate in multiple testing under dependency.” *The Annals of Statistics*, vol. 29, no. 4, 2001, pp. 1165-1188. https://doi.org/10.1214/aos/1013699998.

30. Kwon, Woosuk; Li, Zhuohan; Zhuang, Siyuan; Sheng, Ying; Zheng, Lianmin; Yu, Cody Hao; Gonzalez, Joseph E.; Zhang, Hao; Stoica, Ion. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. https://arxiv.org/abs/2309.06180.