Project Notes:

Project Title:
Name: Kafle, Ruchir

Note Well: There are NO SHORT-cuts to reading journal articles and taking notes from them. Comprehension is paramount. You will most likely need to read it several times, so set aside enough time in your schedule.

Contents:

Knowledge Gaps:

This list provides a brief overview of the major knowledge gaps for this project, how they were resolved and where to find the information.

Knowledge Gap	Resolved By	Information is located	Date resolved

Literature Search Parameters:

These searches were performed between (Start Date of reading) and XX/XX/2019.
List of keywords and databases used during this project.

Database/search engine	**Keywords**	**Summary of search**

Tags:

Tag Name	

Article #0 Notes: Title

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	
Source citation (APA Format)	
Original URL	
Source type	
Keywords	
#Tags	
Summary of key points + notes (include methodology)	
Research Question/Problem/ Need	
Important Figures	
VOCAB: (w/definition)	
Cited references to follow up on	
Follow up Questions	

Article #1 Notes: Researchers Uncover Hidden Ingredients Behind AI Creativity

Article notes should be on separate sheets

Source Title	*Researchers Uncover Hidden Ingredients Behind AI Creativity*
Source citation (APA Format)	Wright, W. (2025, June 30). *Researchers Uncover Hidden Ingredients Behind AI Creativity | Quanta Magazine*. Quantamagazine. https://www.quantamagazine.org/researchers-uncover-hidden-ingredients-behind-ai-creativity-20250630/
Original URL	https://www.quantamagazine.org/researchers-uncover-hidden-ingredients-behind-ai-creativity-20250630
Source type	Science News Site
Keywords	AI, machine learning, neural networks, diffusion model, locality, equivariance.
#Tags	#Diffusion Models #Locality & Equivariance
Summary of key points + notes (include methodology)	My article was about the creativity found in the results of diffusion model AIs. Diffusion models are often used in image generation and despite being trained to give perfect recreations of the material they've been trained on, they end up giving new, yet coherent images. Mason Kamb and Surya Ganguli, a graduate student and professor of physics respectively at Stanford University, found that this creativity comes as a by-product of the design of diffusion models. That architecture being two ideas called locality and equivariance. Locality is a system in which patches of pixels will be considered and generated at a time, with little to no attention to how the patch plays into the final image. Equivariance, on the other hand, is where changes in an input will correspondingly affect the output. These simple systems have allowed most diffusion model AIs to stray away from their training data and introduce some "creativity" to their outputs, to make something new. This article concerns my STEM project as a prospective idea I had was to use AI. While I'm not sure whether I want my project to be all about AI, I'm quite positive that I somehow want to involve AI in my project. If I am to involve AI in my project somehow, it will likely be quite important that I understand how it is able to come up with new solutions, so I can try to use those mechanics to my advantage in my project. It will be especially important to understand the creativity behind AI if I do choose to have my project centered around AI, and even more so if I want to use a diffusion model.
Research Question/Problem/ Need	What gives AI their creativity, when trained to replicate products?

Important Figures	N/A
VOCAB: (w/definition)	Locality - a system in which patches of pixels will be considered and generated at a time, with little to no attention to how the patch plays into the final image. Equivariance - where changes in an input will correspondingly affect the output.
Cited references to follow up on	N/A?
Follow up Questions	Is creativity in AI truly a good thing? Can creativity be replicated using variable amounts of randomness in certain factors? If so, what factors? What are the limits of the factors? How much randomness before the output becomes incoherent?

Article #2 Notes: Why the Latest AI Model Isn’t Always Best for Edge AI

Article notes should be on separate sheets

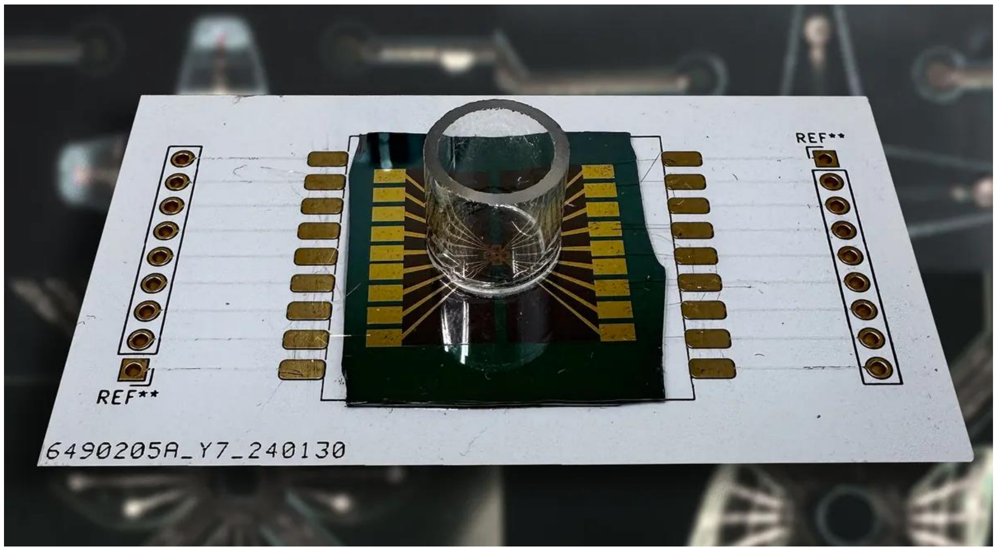
Source Title	*Why the Latest AI Model Isn’t Always Best for Edge AI*
Source citation (APA Format)	Chenna, D. (2025, July 20). *Edge AI: Navigating Hardware Constraints - IEEE Spectrum*. IEEE Spectrum. https://spectrum.ieee.org/edge-ai
Original URL	https://spectrum.ieee.org/edge-ai
Source type	Scientific Publication
Keywords	Edge AI, de-centralized, AI accelerators, hardware, NAS, MACS.
#Tags	#Centralized AI #Edge AI #Hindrances #Optimization #Device specifications
Summary of key points + notes (include methodology)	Most commonly, when interacting with AI, we are talking with centralized systems. That is to say that our devices do not house the trained AI system themselves, instead we push a request to a large, centralized server that houses the AI, one that lives a distance away from where we reside. "Edge" AI, on the other hand, refers to AI systems that live locally on a device, without the need to connect to the internet for communication. The implementation of edge AI in our daily lives has profound benefits, such as privacy and security as a result of all communication being done locally; accessibility as areas with poor internet infrastructure need little to no internet access to work with the AI; and finally, and maybe most importantly, processing speed, which will have major impacts in real-time applications. While edge AI seems to be the future, it is challenging to implement due to the devices it will be running on, that is the devices people have at home, have a limited amount of memory and processing power. This is a massive hindrance, especially for the training of AI, which is a field that typically requires large numbers of resources to ensure accurate results. As such, for edge AI to be a reality rather than a far-off dream, the architecture and production of AI models need to undergo serious change to ensure their optimization for low-power devices, without sacrifices to accuracy. Common methods of optimizing AI models include: neural architecture search (NAS), a

	search algorithm to find the best AI model for a task on a device; transfer learning, where a smaller model learns off a pre-trained model; pruning, reducing parameters that don't make large impacts on accuracy; and quantization, working with smaller numbers to conserve memory. Now while these kinds of optimization may increase the number of "multiply-accumulate" operations (MACs), basic mathematical operations at the core of AIs, there are other parameters that slow results, such as data transfer speeds. While having more MACs may be the best way to achieve faster results in some devices, in others, more MACs will provide the same, maybe even worse results. So, to make sure an AI is running as fast as possible on a device, that AI should be optimized with the device specifications in mind. Though hardware continues to look more favorably upon better AIs day after day, we must continue to take every bit of hardware into account when customizing, training, and deploying edge AI. This article pertains to my prospective ideas as I'm quite keen on the idea of creating an AI that can be used to provide some kind of service to help an end user. To ensure real-time communication, it will likely be best to have the AI run on the end user's device, making it essential I understand edge AI optimization. Optimization and researching AI's theory are also things that really fascinate me, and this article will serve extremely useful in making sure I understand techniques and limitations.
Research Question/Problem/ Need	How can edge AI be made more efficient to run better locally?
Important Figures	N/A
VOCAB: (w/definition)	Edge AI - AI systems that live locally on a device, without the need to connect to the internet for communication. Neural architecture search (NAS) - a search algorithm to find the best AI model for a task on a device. Transfer learning - where a smaller model learns off a pre-trained model. Pruning - reducing parameters that don't make large impacts on accuracy. Quantization - working with smaller numbers to conserve memory. Multiply-accumulate operations (MACs) - basic mathematical operations at the core of AIs.
Cited references to follow up on	N/A?
Follow up Questions	Are there other, better ways of optimizing AI/finding the best model to run on a specific piece of hardware? What are all the factors that should be considered when trying to determine the AI model that works best to an issue?

Article #3 Notes: Chips With Neural Tissue Aim to Make AI More Energy Efficient

Article notes should be on separate sheets

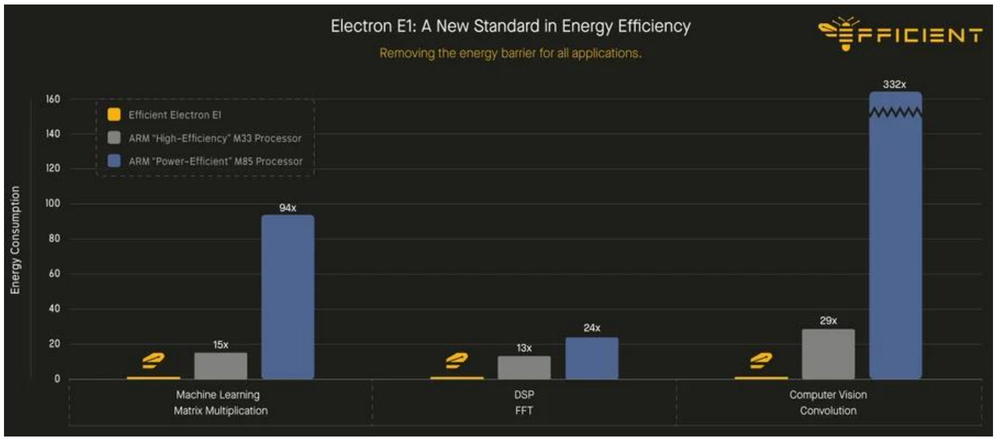
Source Title	*Chips With Neural Tissue Aim to Make AI More Energy Efficient*
Source citation (APA Format)	Mok, A. (2025, August 9). *Biochips Mimic the Brain to Cut AI Energy Use—IEEE Spectrum*. IEEE Spectrum. https://spectrum.ieee.org/biochip-organoid-intelligence-ai-processor
Original URL	https://spectrum.ieee.org/biochip-organoid-intelligence-ai-processor
Source type	Scientific Publication
Keywords	Organoid intelligence, biochips, organoids, EEG, neural structures.
#Tags	#Energy demands #Biochip structures #Reinforcement learning #Living conditions
Summary of key points + notes (include methodology)	As AI becomes larger and more prevalent, its demands for energy are only going to grow more. According to the article, it is already projected to take up 3% of the entire globe's electricity consumption in the next five years. Scientists have been considering new ways to minimize the power needs of AI. However, a promising method being the incorporation of live neurons into silicon chips, that is organoid intelligence on biochips. These biochips use lab-grown neurons to emulate the three-dimensional neural structures found in our powerful brains, increasing efficiency and adaptability from their more two-dimensional silicon chip counterparts. With a three-dimensional structure, more connections are possible in the chip, allowing ease in the transmission of signals and thus information processing. Professor David Gracias' team at Johns Hopkins University developed a electroencephalogram (EEG) shell that fits the curve of the organoid to allow for great stimulation and recording of electrical activity. The team was able to train the organoid using reinforcement learning with dopamine, not dissimilar to how us humans learn. Organoid intelligence has incredible potential that today's AI may not be able to replicate, but it isn't without its own set of issues. With the chips being alive, they, along with the conditions in which they are housed, need to be constantly regulated to ensure they don't die. As such, the systems are bulky, and other issues like latency and signal noise serve as major issues for

	commercial use. More funding and research are required in this subject to see if it's viable, but as of now, it's just a promising candidate. This article pertains to my prospective ideas as I am extremely interested in AI and possibly even its physical development. The process by which chips capable of running powerful AI on them are researched and created fascinates me and is a subject I would love to look more into. Though I don't think I will get the opportunity to investigate and do actual research into organoid intelligence, it is a fascinating subject that helps me understand the direction that AI may be going in.
Research Question/Problem/ Need	Can live neurons be incorporated into silicon chips? Can silicon chips be made in a way that rivals the efficiency and power of the human brain?
Important Figures	
VOCAB: (w/definition)	Organoid – lab-grown neurons used in biochips. Biochips – silicon chips utilizing lab-grown neurons (organoids) to emulate the three-dimensional neural structures found in our brains. Organoid intelligence – an AI running on a biochip or utilizing organoids. Electroencephalogram (EEG) shell – a shell that fits the curve of an organoid to allow for greater stimulation and recording of electrical activity. Dopamine – The reward neurotransmitter found in animals, and now being used in biochips.
Cited references to follow up on	N/A?
Follow up Questions	Can these biochips be mass-manufactured? Can organoids be modified to make them less fragile? How can biochips be made to support the necessities of the organoids living on them?

Article #4 Notes: Startup Claims up to 100x Better Embedded Computing Efficiency

Article notes should be on separate sheets

Source Title	*Startup Claims up to 100x Better Embedded Computing Efficiency*
Source citation (APA Format)	Moore, S. (2025, July 24). *Electron E1: Efficient Dataflow Architecture - IEEE Spectrum*. IEEE Spectrum. https://spectrum.ieee.org/efficient-computer-dataflow-architecture
Original URL	https://spectrum.ieee.org/efficient-computer-dataflow-architecture
Source type	Scientific Publication
Keywords	Von Neumann, general-purpose processor, embedded computing, dataflow, microcontrollers, energy efficiency.
#Tags	#Current processors #Tiles #Spatial pathways #E1 compiler #Dataflow-style architectures
Summary of key points + notes (include methodology)	Under the common von Neumann architecture, general-purpose processors will sequentially interpret instructions on how to work with data, which can lead to lots of overhead. In Electron E1 chips by Efficient Computer, instructions are given to the processor's network of tiles, each performing some instructions and handing off outputs to the next tile. The instructions are mapped out as arbitrary spatial pathways, which are subject to change as the program runs, making the E1 more flexible and thus general-purpose and efficient than rival processors.
Research Question/Problem/ Need	How can computer chips be made more efficient to use less energy?

Important Figures	
VOCAB: (w/definition)	Von Neumann architecture – a computer chip architecture where the chip sequentially takes an instruction from memory, follows it. and puts the result in memory. Overhead – computational resources expended on non-essential tasks rather than the primary one. Tiles – a small unit of computing power capable of performing a limited set of instructions, linked to other tiles in a network. Compiler – a program that takes human-readable code and translates it into instructions for a machine. Systolic array – a parallel computing architecture where a network of processing elements locally connected operate concurrently.
Cited references to follow up on	N/A?
Follow up Questions	• How do certain spatial pathways affect the efficiency of the chip? • How does the efficiency of running programs or the speed at doing so change with respect to the number of tiles? • Should all tiles be stripped down, or would it be beneficial to have some tiles with more functionality? o Should there be separate kinds of tiles for each category of functionality? Or just larger tiles? • Is parallelism truly the most efficient method in all cases?

Article #5 Notes: A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data*
Source citation (APA Format)	Sun, H., Ma, H., Ling, F., Xie, H., Sun, Y., Yi, L., Yan, M., Zhong, C., Liu, X., & Wang, G. (2025). A survey and benchmark evaluation for neural-network-based lossless universal compressors toward multi-source data. *Frontiers of Computer Science*, *19*(7), 197360. https://doi.org/10.1007/s11704-024-40300-5
Original URL	https://link.springer.com/article/10.1007/s11704-024-40300-5
Source type	Journal Article
Keywords	Lossless compression, benchmark evaluation, universal compressors, neural networks, deep learning. (Directly from the article.)
#Tags	#Purpose of Compression #Compressor Types #Related Work #NN Compressor Models #Implementations #Metrics #Experiments
Summary of key points + notes (include methodology)	• The amount of data in the world needing to be stored is going up rapidly. • This is an issue as high data sizes can slow transmission speeds and be costly. • This is why compression algorithms are important. • Algorithms are either universal or dedicated. ○ Dedicated are meant only to handle certain data. ○ Universal handle broad varieties of data. This article focuses on lossless universal compressors. • Since Shannon released Information Theory, many data compression algorithms have been created to minimize storage. • NNs have recently gathered attention in data compression.

- Used to exploit their ability to predict probabilities accurately.
- In a NN compressor, there are two blocks:
  - Probability predictor block.
    - RNNs, attention mechanisms, or LLMs.
  - Encoder block.
- Probability predictor block will take the terms in a sequence to find the probability distribution of the next.
- Probability distribution previously calculated goes into encoder block to encode the token using entropy coding techniques like Huffman or arithmetic coding.
  - Goal is to have the most accurate probability.
  - More accurate probability = better compression.
- Three categories of NNs:
  - Static (offline)
    - Pre-training stage
      - Model is trained on outside source.
      - Learned weights are kept during compression to compress data.
    - Compression stage (same as above)
    - Can't adapt to differing data, extra storage taken up due to having to store model weights, and overhead in training phase, needs outside data.
    - Many models are static.
  - Adaptive (online)
    - The model is trained actively as it goes through the data.
    - Back propagation can be slow.
    - Outside data isn't needed.
    - Weights don't need to be stored.
    - Many modern models are adaptive, performing better on heterogeneous data.
  - Semi-adaptive
    - Static and adaptive parts.
    - Two models
      - Bootstrap
        - Static.
      - Support
        - Adaptive.
    - Extremely resource costly due to being two neural networks.
    - Great compression ratios.
    - Relatively unimplemented.
- Implementation is often done in Python.
- RNNs, Transformers are most common in modern. FCs and FFNs are quite common before.
- Paper uses compression ratio (CR) and storage saving percentage (SSP) to evaluate algorithm effectivity and compression robust performance (CRP)

for robustness of compression effect.
- When tested on various datasets, NN based compressors consistently perform better than traditional methods on average. NN based compressors tend to have incredible compressions, though it is at significant time costs.
    - NNCP seems to do the best compression size wise.
- Compression ratio is highly dependent on various factors.
    - Parameters of model.
    - Alphabet size.

This paper conducted a comprehensive survey of architectures and performances of current NN data compression models. It went over many concepts aimed at familiarizing one with the broad field, like offline and online models. Using their metrics, the paper was able to benchmark competing models with traditional models to figure out how they compared to one another. The paper concluded that while NNs have relatively good compression ratios to traditional models, their compression times need improvement. The paper also suggests the creation of detailed classifications for NN compressors and an automated evaluation framework.

Research Question/Problem/ Need

What are the various ways that neural networks are being utilized in data compression?

Important Figures

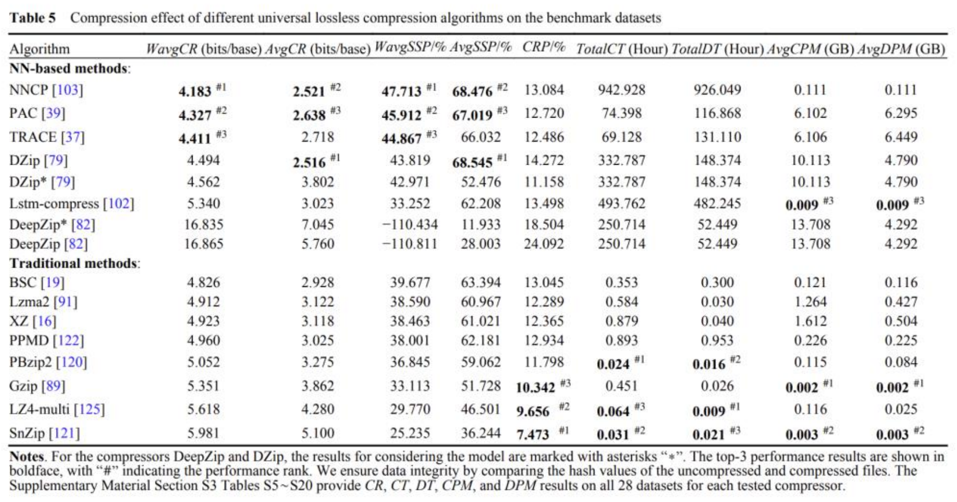


Table 5 Compression effect of different universal lossless compression algorithms on the benchmark datasets

Algorithm	*WavgCR* (bits/base)	*AvgCR* (bits/base)	*WavgSSP*/%	*AvgSSP*/%	*CRP*/%	*TotalCT* (Hour)	*TotalDT* (Hour)	*AvgCPM* (GB)	*AvgDPM* (GB)
NN-based methods:									
NNCP [103]	**4.183** #1	**2.521** #2	**47.713** #1	**68.476** #2	13.084	942.928	926.049	0.111	0.111
PAC [39]	**4.327** #2	**2.638** #3	**45.912** #2	**67.019** #3	12.720	74.398	116.868	6.102	6.295
TRACE [37]	**4.411** #3	2.718	**44.867** #3	66.032	12.486	69.128	131.110	6.106	6.449
DZip [79]	4.494	**2.516** #1	43.819	**68.545** #1	14.272	332.787	148.374	10.113	4.790
DZip* [79]	4.562	3.802	42.971	52.476	11.158	332.787	148.374	10.113	4.790
Lstm-compress [102]	5.340	3.023	33.252	62.208	13.498	493.762	482.245	**0.009** #3	**0.009** #3
DeepZip* [82]	16.835	7.045	−110.434	11.933	18.504	250.714	52.449	13.708	4.292
DeepZip [82]	16.865	5.760	−110.811	28.003	24.092	250.714	52.449	13.708	4.292
Traditional methods:									
BSC [19]	4.826	2.928	39.677	63.394	13.045	0.353	0.300	0.121	0.116
Lzma2 [91]	4.912	3.122	38.590	60.967	12.289	0.584	0.030	1.264	0.427
XZ [16]	4.923	3.118	38.463	61.021	12.365	0.879	0.040	1.612	0.504
PPMD [122]	4.960	3.025	38.001	62.181	12.934	0.893	0.953	0.226	0.225
PBzip2 [120]	5.052	3.275	36.845	59.062	11.798	**0.024** #1	**0.016** #2	0.115	0.084
Gzip [89]	5.351	3.862	33.113	51.728	**10.342** #3	0.451	0.026	**0.002** #1	**0.002** #1
LZ4-multi [125]	5.618	4.280	29.770	46.501	**9.656** #2	**0.064** #3	**0.009** #1	0.116	0.025
SnZip [121]	5.981	5.100	25.235	36.244	**7.473** #1	**0.031** #2	**0.021** #3	**0.003** #2	**0.003** #2

Notes. For the compressors DeepZip and DZip, the results for considering the model are marked with asterisks "*". The top-3 performance results are shown in boldface, with "#" indicating the performance rank. We ensure data integrity by comparing the hash values of the uncompressed and compressed files. The Supplementary Material Section S3 Tables S5~S20 provide *CR*, *CT*, *DT*, *CPM*, and *DPM* results on all 28 datasets for each tested compressor.

This picture is a table displaying all the results of running all the data compressors by the various data sets. The bold values represent the best performing model in a category. In the first half of the table, we see all the bolded values clustered in the top, the NN-based section, showing NN compressors usually operate better than typical traditional compressors. In the second half, however, most of the values are clustered in the bottom, showing that the traditional methods have much better run times than neural compressors. Overall, this data shows that while neural compressors have great compression rates, the time in which they compress is infeasible for practical use.

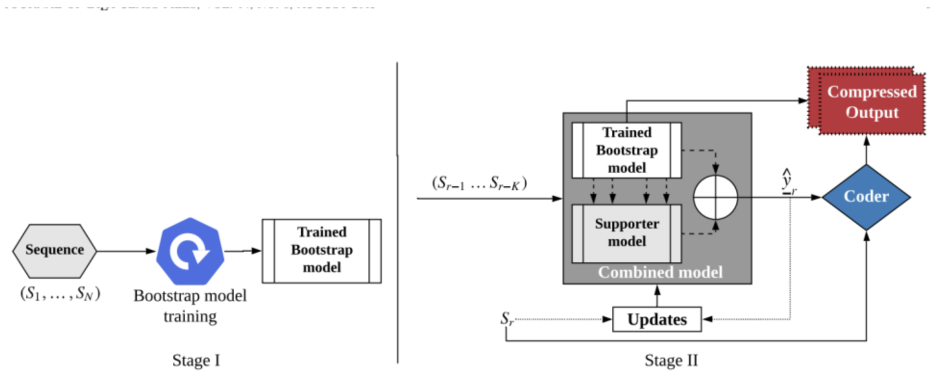
VOCAB: (w/definition)	Information theory – The study of finding in how little space information can be represented. Entropy coding – Methods of compressing data that bring storage size as close to the entropy limit as possible. Examples are Huffman coding and arithmetic coding. Neural network - A computer program with a structure that resembles that of neurons in the human brain. It is able to simulate human intelligence through this structure, making intelligent decisions, and remembering contexts. Recurrent neural networks (RNNs) - Neural networks with longer term memories. Attention mechanisms – Neural networks that are able to determine the importance of certain tokens in a sequence over others.
Cited references to follow up on	Mao Y, Cui Y, Kuo T W, Xue C J. TRACE: a fast transformer-based general-purpose lossless compressor. In: Proceedings of ACM Web Conference 2022. 2022, 1829−1838 Goyal M, Tatwawadi K, Chandak S, Ochoa I. DZip: improved general-purpose lossless compression based on novel neural network Modeling. In: Proceedings of the 2020 Data Compression Conference. 2020, 372−372 Shannon C E. A mathematical theory of communication. The Bell System Technical Journal, 1948, 27(3): 379−423
Follow up Questions	What other metrics may need to be considered for networks with different goals?

Article #6 Notes: DZip: improved neural network based general-purpose lossless compression

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*DZip: improved neural network based general-purpose lossless compression*
Source citation (APA Format)	Goyal, M., Tatwawadi, K., Chandak, S., & Ochoa, I. (2020). *DZip: Improved general-purpose lossless compression based on novel neural network modeling* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1911.03572
Original URL	https://arxiv.org/abs/1911.03572
Source type	Preprint
Keywords	Neural/learned compression, lossless compression, neural networks, arithmetic coding, adaptive training, and semi-adaptive training.
#Tags	#Related Works #Background #Training Types #Architecture #Combined model #Training #Results #Real Data #Synthetic Data
Summary of key points + notes (include methodology)	• Surge in data leads to a need for compression. • Prediction + entropy coding method. • NN predicts probability of next symbol based on context (previous symbols). • Probabilities and symbols are given to encoder to compress data. • DZip utilizes quantized semi-adaptive bootstrap model and adaptive supporter model training. • Final predictions used for compression are a combination of ^. • Bootstrap model utilized a GRU to capture long term relations. • Bootstrap model is an RNN that learns long-term relationships. Final weights must be saved. • Supporter model is an FC that is composed of 3 parts. Overall, it adapts quickly and provide better probabilities. • Combined model takes both sets of logits to get final logits with a sigmoid activated parameter. • Final logits then are scaled to probabilities via softmax activation.

	• DZip reads the input and selects hyperparameters, trains bootstrap model by doing multiple passes, then can be ran in two modes, each trading off either compression ratio and speed. • Encoding happens in 64 equally sized batches for quicker speeds. • Encoder and decoder are symmetrical. • DZip, run on a variety of real datasets, gave almost the best compression ratios compared to other compressors, except specialized compressors. ○ Especially strong on non-text data. • ZPAQ, a specialized text compressor, outperforms DZip consistently, however difference decreases as sequence length increases due to DZip amortization. • DZip sensitive to sequence length and alphabet size due to overhead of bootstrap model parameters. • Bootstrap model contributes heavily to final size. • DZip performs as well if not better than specialized compressors due to specialized compressors not being optimized for specific formats of data. • Nearly achieves theoretical entropy minimum for k < 70, but slight overhead from bootstrap model size. • From ablation, determined bootstrap has higher compression ratios on real datasets for higher time costs. • High computational requirements, but relatively good in comparison to other models. The authors of this article were able to create DZip, a neural network compressor capable of compressing data much better than previous neural network models. While still not quite comparable to specialized compressors, it performs much better than other neural networks and on the level of traditional models. DZip utilizes a two neural models for semi-adaptive and adaptive training. Doing this leads to compression ratios, DZip’s strong suit, but not compression times.
Research Question/Problem/ Need	Can a neural network based compressor be created to efficiently compress data without the use of additional datasets?
Important Figures	 Sequence $(S_1, \ldots, S_N)$ Bootstrap model training Trained Bootstrap model Stage I $(S_{r-1} \ldots S_{r-K})$ Trained Bootstrap model Supporter model Combined model Compressed Output $\hat{y}_r$ Coder S_r Updates Stage II The sequence of data is fed into the bootstrap model to train it. One the model is trained; its results are fed into the support model. Both outputs are added as part

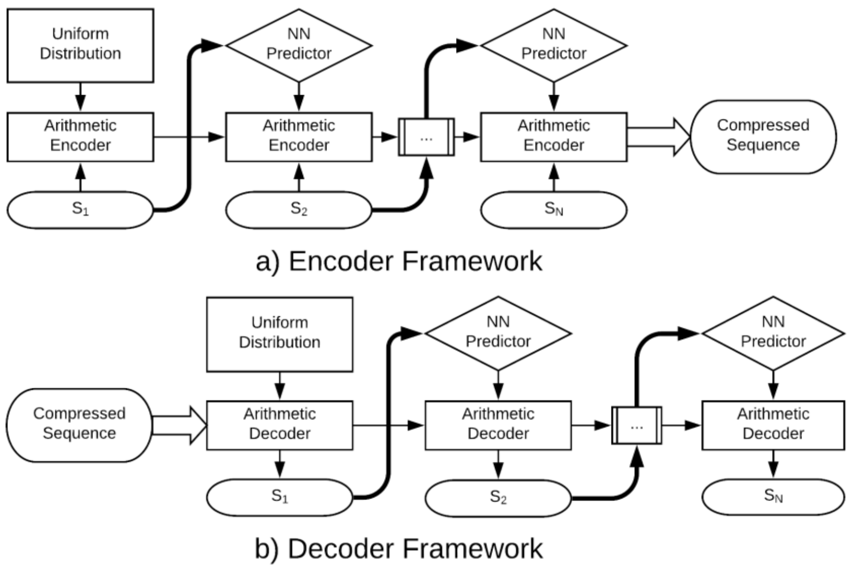
	of the combined model, where probabilities are calculated. These probabilities go into the arithmetic coder which will eventually result in compressed output. The supporter model is adaptive, so updates that need to happen will be checked and the combined model will be fixed. This is important as it is a show of the complex structure of this DZip.
VOCAB: (w/definition)	Static training - Trains the model on some external data; restricted to when similar data is available and not very universal. Model weights must be stored. Adaptive training - Trains the model as it goes through the sequence, based off randomly initialized values; doesn't need outside sources, can adapt, but is slow. Model weights don't need to be stored. Semi-adaptive training - Trains the model on the sequence; leads to more accurate models and thus better compression ratios at the cost of time. Model weights are stored. ADAM Optimizer – Optimization algorithm that updates model weights during back propagation. It combines momentum, the average of past gradients, and RMS, the average of the squared gradients. Amortized – Cost being spread out over multiple uses, leading to better cost ratios.
Cited references to follow up on	J. Devlin *et al.*, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018. A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019. T. Salimans *et al.*, "PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.
Follow up Questions	How does the activation function used change the result? How could scaling the models up or down individually affect overall compression?

Article #7 Notes: DeepZip: Lossless Data Compression using Recurrent Neural Networks

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*DeepZip: Lossless Data Compression using Recurrent Neural Networks*
Source citation (APA Format)	Goyal, M., Tatwawadi, K., Chandak, S., & Ochoa, I. (2018). *DeepZip: Lossless Data Compression using Recurrent Neural Networks* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1811.08162
Original URL	https://arxiv.org/abs/1811.08162
Source type	Preprint
Keywords	Recurrent neural network, long short-term memory, gated recurrent unit, fully connected neural network, neural network, arithmetic coding.
#Tags	#RNNs? #Related Work #Architecture #Encoding-Decoding #FC #LSTM/GRU #Experiments
Summary of key points + notes (include methodology)	• Surge in data leads to a need for compression. • Probability predictor block will take the terms in a sequence to find the probability distribution of the next. • Probability distribution previously calculated goes into encoder block to encode the token using entropy coding techniques like Huffman or arithmetic coding. ○ Goal is to have the most accurate probability. ○ More accurate probability = better compression. • Encoder block outputs final compressed bitstream. • RNN is trained for multiple epochs and weights are stored. • During inference: ○ Trained model used on each symbol in an input sequence to output probability distribution. ○ Distribution used to encode symbol. ○ Next symbol undergoes same process, but context is updated to include previous symbol. • Decompression is symmetrical. • FC model was also trained in place of the RNN to serve as a baseline due to having shorter term memory.

	• LSTM and GRU were used to get rid of vanishing gradients issue of default RNN. • Trained on real and synthetic data sets. • DeepZip proved comparable or better than traditional compressors on real data sets. • DeepZip proved much better than traditional compressors on synthetic data sets. • DeepZip wasn't quite able to compare to specialized compressors and is slow. In this paper, the authors successfully proved that RNNs can be used in data compression and are actually quite good. They point out that there is still work left to be done though.
Research Question/Problem/ Need	Can neural networks be used to create ways of effectively compressing data without loss?
Important Figures	 Uniform Distribution NN Predictor Arithmetic Encoder ... Compressed Sequence S_1 S_2 S_N a) Encoder Framework Arithmetic Decoder b) Decoder Framework Figure 1: Encoder-Decoder Framework. This is a diagram of what happens during inference. The arithmetic encoder will be procedurally fed two things: a symbol in the input sequence and its probability distribution. The probability distribution comes from the NN predictor, which takes the previous symbols, the context, in order to determine the distribution of the next symbol. This will happen and symbols will continue to be compressed until a final compressed sequence is output. In decompression, the exact same process happens. The decoder will output the next symbol based on the NN probability distribution, which is predicted via previous symbols. This is important

	because this is the basis of the model's memory and compression.
VOCAB: (w/definition)	Recurrent neural network (RNN) – A neural network with a longer term memory. Vanishing gradient issue – An issue in regular RNNs where the gradient exponentially decays during training, meaning some weights don't get altered, leading to no actual training being done. Long short-term memory (LSTM)/Gated recurrent unit (GRU) – RNN's that use gates to preserve gradients otherwise lost in the vanishing gradient issue in base RNNs. Inference – When the model is running.
Cited references to follow up on	David A Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952. Frans MJ Willems, Yuri M Shtarkov, and Tjalling J Tjalkens, "The context-tree weighting method: basic properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, 1995. John Cleary and Ian Witten, "Data compression using adaptive coding and partial string matching," *IEEE transactions on Communications*, vol. 32, no. 4, 1984.
Follow up Questions	Could some probabilities be truncated to prevent further storage of data? How could attention models add to these models? Could more epochs help with compression?

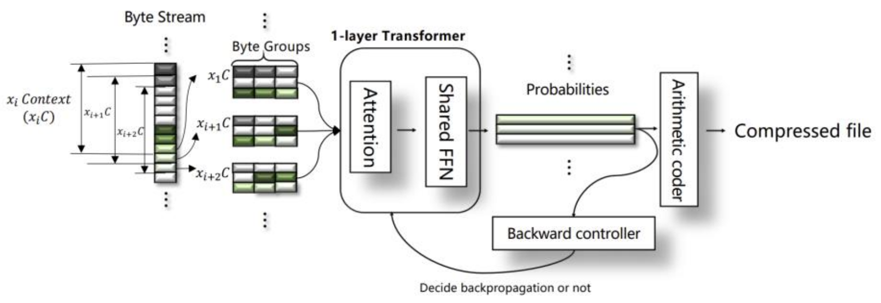
Article #8 Notes: A Fast Transformer-based General-Purpose Lossless Compressor

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*A Fast Transformer-based General-Purpose Lossless Compressor*
Source citation (APA Format)	Mao, Y., Cui, Y., Kuo, T.-W., & Xue, C. J. (2022). *A Fast Transformer-based General-Purpose Lossless Compressor* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2203.16114
Original URL	https://arxiv.org/abs/2203.16114
Source type	Preprint
Keywords	General-purpose compressor, byte stream, lossless data compression, neural networks, transformer, computational efficient model. (Taken directly from the journal article, which had listed keywords.)
#Tags	#Reason for using Transformer #Batch Size vs Memory Usage and Inference Throughput #Multi-Head Attention (MHA) #Feed Forward Network (FFN) #TRACE Structure #LCR #Byte Grouping #Shared-FFN #Dynamic Compression #Back Prop #Related Work
Summary of key points + notes (include methodology)	• Data volume has been boosted significantly, which can be costly for data centers. For storage in the PiB, energy bills can cost millions. • Standard compressors are limited in multi-modal data compression. Multi-specialized compressors can be used but aren't scalable. • Deep-learning models fix these problems but introduce a new issue: high compression times. • This paper attempts to decrease compression times and increase compression for multi-modal media. • Previous AI compressors used RNNs. • Uses single layer transformers to get rid of GPU utilization overhead. ○ Building history in parallel. • Studies transformer components' (Attention and Feed Forward Network (FFN)) impact on compression. • Proposes Back-Prop (BP) Controller to discard unneeded parameter

	updates. • Transformer has surprisingly good inference throughput and max GPU usage for batch sizes. • First find multi head attention matrix to determine what's important/long term dependencies. • Second uses an FFN to refine tokens and decide what to emphasize. • Uses common probability and entropy encoding blocks for NNs. • Standardize measurements with Latency-to-Compression Ratio (LCR, which compares model size (via latency) to the compression. o Higher value = faster latency growth for compression ratio. • Attention is computationally expensive while FFN is relatively cheap. • Currently, redundancy due to wasting hidden dimensions on each byte. • Gave groups of bytes rather than individual bytes hidden dimensions. • More layers = exponentially decaying returns in compression ratio, but much worse compression times. Few layers or shared-FFN is optimal. • Back propagation is slow but lets weights adjust for heterogeneous data. Paper proposes a Back Prop Control to stop redundant propagations via cache and thresholds of how new data is. • In experiments on real data sets, TRACE performed significantly better than all other NN compressors. With much less latency and GPU usage, it had much higher compression speeds, however, even these speeds were 3 orders of magnitude worse than traditional compressors. • Some competitors weren't running at best, so comparison seems a little bloated. • TRACE also seems to perform with significantly better compression ratios to even traditional models, which I am skeptical of. o Except heterogenous data, where Dzip wins. TRACE is a very computationally efficient model. It utilizes a transformer and various new algorithms to optimize and get rid of overhead. Some algorithms implemented are Back Prop Control, using a cache, byte grouping, optimized layer sizes, a shared-FFN, and more. This allows TRACE to have similar to decently higher compression ratios compared to competitor models while having significantly higher run times. Overall, TRACE has a leading efficiency, shown through the metrics the paper defined.
Research Question/Problem/ Need	Can a transformer model be used to efficiently compress data better than recurrent neural networks?
Important Figures	

	 Byte Stream Byte Groups 1-layer Transformer x_i Context (x_iC) $x_{i+1}C$ $x_{i+2}C$ x_1C $x_{i+1}C$ $x_{i+2}C$ Attention Shared FFN Probabilities Arithmetic coder Compressed file Backward controller Decide backpropagation or not **Figure 2: Overall architecture of proposed TRACE.** TRACE will take in a byte stream and separate it into groups of bytes. This is done to prevent redundancy in data. It is then fed into the transformer, which consists of the attention and shared FFN. The attention will figure out what is important, the FFN will refine it, and if back propagation is needed to fix weights, it will go through the backward controller. This process will output probabilities for the next terms in a sequence, which will be used in the arithmetic coder to compress the data. This diagram is important because it is a full and in-depth show of the model.
VOCAB: (w/definition)	Transformer – An AI model with two sections, an attention section and FFN. Attention – A section of a transformer that decides what tokens are important to consider for the final result. Filters out the unimportant data. FFN – A basic neural network that highlights and emphasizes important data to make more accurate predictions. Back propagation – A method of altering weights in a neural network, so the model better represents the data. Hidden dimensions – The number of nodes (neurons) in a given layer of the neural network (only one in the case).
Cited references to follow up on	Y. Tay, M. Dehghani, S. Abnar, et al. 2020. Long Range Arena: A Benchmark for Efficient Transformers. (2020). arXiv:2011.04006 preprint. James Townsend, Tom Bird, and David Barber. 2019. Practical lossless compression with latent variables using bits back coding. *arXiv preprint arXiv:1901.04866* (2019). Rianne van den Berg, Alexey A Gritsenko, Mostafa Dehghani, Casper Kaae Sønderby, and Tim Salimans. 2020. Idf++: Analyzing and improving integer discrete flows for lossless compression. In *International Conference on Learning Representations*.

Follow up Questions	How could more or less parameters affect compression ratio? Could more shared-FFN layers be utilized? Could a model other than an FFN be used in the transformer?

Article #9 Notes: Sequential Neural Text Compression

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*Sequential Neural Text Compression*
Source citation (APA Format)	Schmidhuber, J., & Heil, S. (1996). Sequential neural text compression. *IEEE Transactions on Neural Networks*, *7*(1), 142–146. https://doi.org/10.1109/72.478398
Original URL	https://ieeexplore.ieee.org/document/478398
Source type	Journal Article
Keywords	Lempel-Ziv, Huffman Coding, arithmetic Coding, neural compression, neural networks, feedforward network, adaptive training.
#Tags	#Need and Why Neural Networks #Probability compression #Offline and Online #Network Structure #Encoder Structure #Experiments #Results
Summary of key points + notes (include methodology)	• As of the article many text lossless encoding algorithms use Lempel-Ziv (LZ). • Aims to prove neural networks can be used to make “excellent” text compression algorithms. • Implements Huffman Coding, arithmetic coding, neural networks, and more. • Uses neural network to approximate probability distribution of next character given previous ones. • Probabilities go into coding algorithms that generate shorter codes with high probability. • Look up tables grow faster than neural networks for such a model. ○ Neural networks are optimal solution to problem. • Offline and online network from sender to receiver. ○ Offline network trained on training files, and after training weights are frozen and used to encode/decode data. ○ Online network learns during compression. ▪ Learned weights don’t need to be sent to receivers so long as initial conditions and learning algorithm is same. ○ Online performance > offline, but more computationally expensive. • Strictly layered feedforward network (FFN) trained by back propagation is

used.

- Every symbol in a sequence of characters is represented as a vector in the network.
- The vector is the same size as the alphabet size (unique characters), with each position representing a unique character in the alphabet.
- The vector is binary, so each position is either 0 or 1, with only 1 position being 1.
  - 0 represents the symbol is not the alphabet symbol at that position.
  - 1 represents the symbol and alphabet symbol at the position match.
- The neural network has *n* number of input nodes for each alphabet character, and output nodes matching the number of unique alphabet characters. *n* is the context size.
  - Since the neural network computes probability, each unique symbol will have its own output node to show its probability of being the next character.
- Method 1: Huffman bitstring coding tree is built up using probabilities.
- Must also decode after encoding.
  - Neural network predicts probabilities and feeds them into inverse Huffman coder (basically the same as encoding.)
  - For some characters, all previous characters must be decoded
- Method 2: Arithmetic coding is used rather than un-optimal Huffman.
- Same process as Huffman, though requires different math for actual encoding.
- Method 3: Change model weights based off of wrong predictions.
  - The model will predict the next character based on context, and if it gets the actual value wrong, adjust weights.
  - If the model gets character right, store nothing as it represents redundancy.
  - Eventually, model will have proper rates to properly construct probabilities.
  - Then goes through Huffman.
  - Similar to adaptive training.
- Methods 1 and 2 generally did the best.

Neural network data compressors are promising. They have great compression ratios, though this is at the cost of high compression times, which are not practical. There are multiple ways this paper discussed compressing data, including Huffman encoding, arithmetic coding, and adaptive training.

I chose to read an older article as I wanted to really understand the basis that so many of the articles I read were based on. Each of the articles I read talk about the concepts discussed in the paper due to using it as foundation, but they also add a lot more that they sought to change with their paper. In reading this paper, I wanted to truly understand the raw fundamentals and see how modern compression has deviated from the original ideas of how neural compression would be. I also wanted to see if there were any ideas which were not yet implemented, and their reasons for implementing what would become the foundation of modern neural compression.

Modern articles have changed to use different model types and structures to better predict probabilities. The probability predictor going into the encoder model idea hasn't changed. The encoder model also hasn't changed, indicating near perfect entropy encoding.

Research Question/Problem/ Need

Can neural networks be used to design text compression algorithms with compression ratios exceeding 2.0?

Important Figures

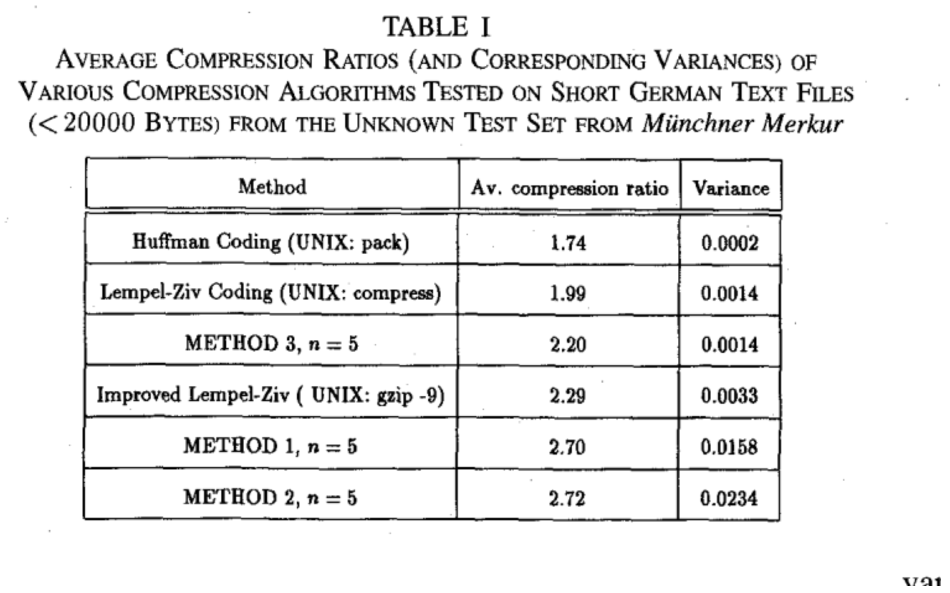


TABLE I
AVERAGE COMPRESSION RATIOS (AND CORRESPONDING VARIANCES) OF VARIOUS COMPRESSION ALGORITHMS TESTED ON SHORT GERMAN TEXT FILES (< 20000 BYTES) FROM THE UNKNOWN TEST SET FROM *Münchner Merkur*

Method	Av. compression ratio	Variance
Huffman Coding (UNIX: pack)	1.74	0.0002
Lempel-Ziv Coding (UNIX: compress)	1.99	0.0014
METHOD 3, $n = 5$	2.20	0.0014
Improved Lempel-Ziv (UNIX: gzip -9)	2.29	0.0033
METHOD 1, $n = 5$	2.70	0.0158
METHOD 2, $n = 5$	2.72	0.0234

This figure shows the results of an experiment where the neural network compressor was put head-to-head with traditional compressors. Each method compressed a small text file; in this case the text file was in German. In the table, it is seen that each of the neural networks are able to perform better than Huffman Coding and Lempel-Ziv Coding, and methods 1 and 2 were able to score the highest, beating out improved Lempel-Ziv. This data shows us that neural network' compression ratios are better than those of traditional compressors, showing their potential as data compressors. Additionally, method 2 scoring the highest shows Arithmetic Coding is likely the best compression method using neural networks, showing what method is best to use.

VOCAB: (w/definition)

Average compression ratio - Average ratio between original and compressed files.

Lempel-Ziv coding – A traditional method of data compression where repeated strings are replaced with a pointer to a dictionary.

Huffman coding – A method of entropy coding where Huffman trees are

	constructed to give high probability symbols less bits. Arithmetic coding – Same purpose as Huffman coding but does it through representing sequences as an interval from [0, 1). Neural network – A computer program with a structure that resembles that of neurons in the human brain. It is able to simulate human intelligence through this structure, making intelligent decisions, and remembering contexts.
Cited references to follow up on	S. Lindstädt, "Comparison of two unsupervised neural network models for redundancy reduction," in M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, Eds., in *Proc. 1993 Connectionist Models Summer School*, 1993, pp. 308-3 15. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1. Cambridge, MA: MIT Press, 1986, pp. 318-362. J. H. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computa.*, vol. 4, no 2, pp. 234-242, 1992.
Follow up Questions	Could storing each symbol as a vector of size alphabet size, where all but one element is the same be computationally inefficient? Maybe some quantization would help? How might each of the methods do better or worse as various parameters change? I.e. could adaptive learning perform better than arithmetic coding if alphabet size changed?

Article #10 Notes: Distributed Compression in the Era of Machine Learning: A Review of Recent Advances

Article notes should be on separate sheets

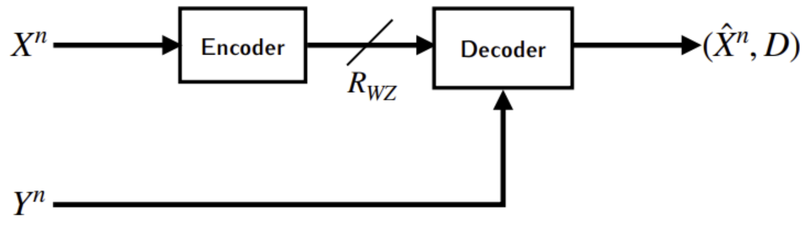
KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	*Distributed Compression in the Era of Machine Learning: A Review of Recent Advances*
Source citation (APA Format)	Ozyilkan, E., & Erkip, E. (2024). *Distributed Compression in the Era of Machine Learning: A Review of Recent Advances* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2402.07997
Original URL	https://arxiv.org/abs/2402.07997
Source type	Preprint
Keywords	Distributed source coding, Wyner–Ziv coding, lossy compression, binning, neural networks, rate-distortion theory, learning. (Directly from article.)
#Tags	#Lossless DSC #Lossy DSC #Wyner-Ziv #Media Sources #Abstract Data
Summary of key points + notes (include methodology)	• Information has a minimum limit based on its entropy. • Lossy compression, where data is lost for more compression, and lossless compression, where data is maintained for less compression. • Lossy compression complexity makes lossy compressors specialized to domains. • In lossy image compression, ○ Linear transformation and quantization ○ Then lossless compression entropy coding. • Deep learning neural networks (DNN) have increased and can perform better than JPEG. • Traditional compressors apply handmade transformations to images; however, DNN compressors learn patterns directly from the image to find the best transformation to approximate the data. • In distributed source coding (DSC), it's predicted that if compressed jointly, efficiency will increase rather than if done one by one; however complex correlations of data between sources are difficult to handle. • Authors think DNNs will be helpful to DSC due to quick adaptability. • Goal of paper to summarize DNN-aided DSC, specifically regarding Wyner-Ziv problem.

- Slepian-Wolf Theorem says the optimal rate region lossless DSC is the rate region which satisfies a set of conditions.
- Using random bins theory, multiple encoders can losslessly compress their sources with optimal rate regions, as though they were in communication with each other.
- Wyner-Ziv problem asks what the rate region will be for when lossy compression is allowed and one source is only available at the decoder.
- DISCUS aims to compress an unknown source efficiently when only one source is available to the decoder.
- Current practical compressors use linear transformations to quantize elements of data vectors.
- Nonlinear transformations used to be unfeasible, but with NNs they are much more feasible.
- NN model will look at the pixels of an image and transform them into latent space. The latent space values are quantized, giving final compressions.
- Network looks at distortion and bitrate and attempts to minimize both.
- Back propagation is not possible here due to quantization being nondifferentiable. Instead of rounding, additive noise is used during training to allow backpropagation, though quantization is kept during runtime.
- Using differentiable channels rather than quantization is becoming more popular, but it’s unsure how this model scales with complexity.
- Shown in recent works using a Quantized Variational Autoencoder (VQ-VAE), neural compressors can successfully compress the correlation between images in decoder-only side information.
- In joint source channel coding (JSCC), neural networks beat traditional JSCC models.
- NN can get optimal performance for abstract sources; however they struggle significantly on complex data.
- NN compressors seem to be good for the Wyner-Ziv problem, however don’t utilize side information effectively.

This article reviewed literature on distributed source coding. It aimed to show schemes and how they affected performance for DNNs that revolved around DSC. It gave valuable insight into these issues, like some of their bases, the Wyner-Ziv and Slepian-Wolf. Future work includes extending various neural schemes across fully distributed compression settings. This work has the impact of enhancing the efficiency of practical communication networks.

I chose this article because I wanted to get an understanding of other fields that use neural networks in data compression. I have mostly been looking at lossless text neural compression, but I also wanted to see lossy neural compression, which is usually applied to images and other similar media. Distributed source coding was also a fascinating topic to read about. I wanted to see if any concepts in either field could be applied to another to increase compression ratios and speeds.

Research Question/Problem/ Need	How are neural networks and deep learning being used in distributed source coding and image compression?
Important Figures	 X^n Encoder R_{WZ} Decoder $(\hat{X}^n, D)$ Y^n Fig. 2: Rate–distortion with side information, also known as *Wyner–Ziv* coding. The Wyner-Ziv problem. Where only the decoder has access to both sources. This model is important because it is the basis of many DSC models today.
VOCAB: (w/definition)	Distributed Source Coding (DSC) - Efficiently compressing information from sources that are physically separated, and then decoding them together. Wyner-Ziv problem – In lossy DSC, side information is only available at the decoder side. Rate region – A set of two bitrates, the amount of bits each symbol in a sequence takes up, for two sources of data. Achievability – Whether or not a region rate can be reached by DSC. Converse – Says if there is no way that a region rate can be better than those found in achievability.
Cited references to follow up on	J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *International Conference on Learning Representations*, 2018. J. Ballé, P. A. Chou, D. Minnen, S. Singh, N. Johnston, E. Agustsson, S. J. Hwang, and G. Toderici, "Nonlinear transform coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 2, pp. 339–353, 2021. D. C. Minnen, J. Ballé, and G. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Neural Information Processing Systems*, 2018.
Follow up Questions	Could similar systems be employed in text compression? When does quantization become too great?

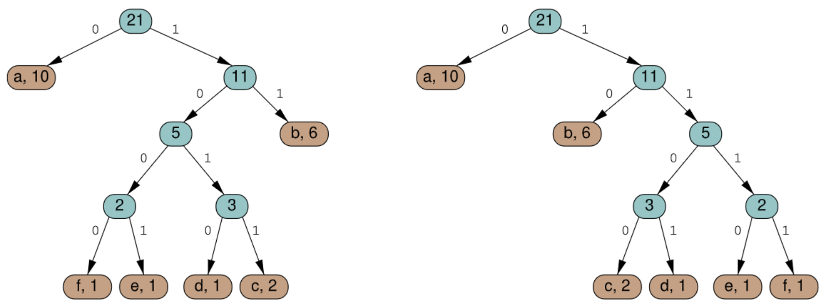
	How do differentiable channels change model size?

Article #11 Notes: Huffman Coding

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	Huffman Coding
Source citation (APA Format)	Moffat, A. (2019). Huffman Coding. *ACM Computing Surveys*, *52*(4), 85:1-85:35. https://doi.org/10.1145/3342555
Original URL	https://dl.acm.org/doi/10.1145/3342555
Source type	Journal Article
Keywords	Huffman code, minimum-redundancy code, data compression, Theory of computation, Design and analysis of algorithms, Data compression, Information systems, Data compression, Search index compression, Mathematics of computing, Coding theory.
#Tags	#HuffmanCoding
Summary of key points + notes (include methodology)	Huffman coding computes minimum-redundancy, prefix-free codes, which means the files encoded have no repeated data and no "codeword" in a code is a starting point of another, ensuring decryption. This article focuses on implementation rather than the tree-based understanding. In Huffman coding, you have a list of nodes, with each node being a unique symbol with its frequency. A greedy process is applied where the symbols with the least frequency are combined into a new internal node, where the symbol is [x, y], x and y being the nodes' symbols, and the frequency being x's frequency + y's frequency. This is done until only one node is left. The final node is a "tree," where the most likely symbol is at the top, and the least likely are at the bottom when you go through each node list. Now the simple textbook implementation can be applied to get the encoding for each symbol. Every time you go left, your binary codeword will add on a 0, every time you go right, your binary codeword will add on a 1. This means that in the tree in the important figures section, symbol "a" would have a binary representation as 0, "b" would have one that is 11, "c" is 1011, and so on. Theoretically, these are the smallest codelengths when considering one symbol at a time. This encoding means that whenever an "a" appears in a file, it will be replaced for "0," b for "11," c for "1011," etc.
Research Question/Problem/ Need	How does Huffman coding and its variations/interpretations work?

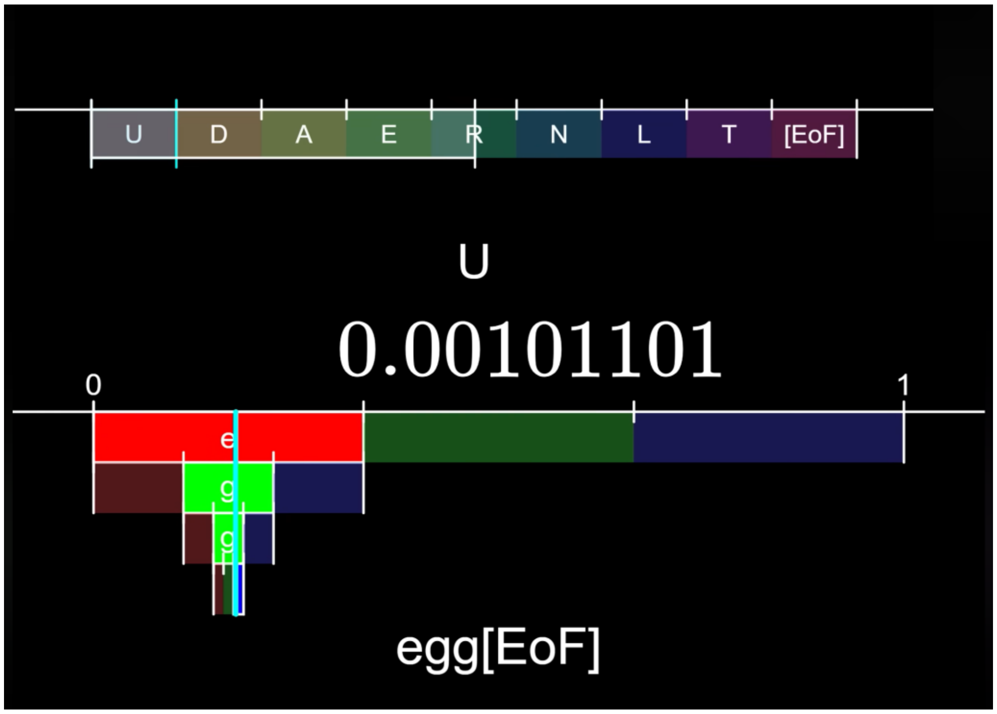
Important Figures	 21 0 1 a, 10 11 b, 6 5 2 3 f, 1 e, 1 d, 1 c, 2 A representative Huffman tree.
VOCAB: (w/definition)	Source alphabet – The data from which the symbols to be used in the algorithm with compression come from. Node – A point on a graph in graph theory. Edge – A line that connects multiple nodes in graph theory.
Cited references to follow up on	A. Turpin and A. Moffat. 1998. Comment on "Efficient Huffman decoding" and "An Efficient Finite-State MachineImplementation of Huffman Decoders."Inform. Process. Lett. 68, 1 (1998), 1–2. A. Turpin and A. Moffat. 2000. Housekeeping for prefix coding. IEEE Trans. Commun. 48, 4 (2000), 622–628. Sourcecode available from http://people.eng.unimelb.edu.au/ammoffat/mr_coder/
Follow up Questions	Why is Huffman coding worse than arithmetic coding? How do they vary?

Article #12 Notes: these compression algorithms could halve our image file sizes (but we don't use them) #SoMEpi

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	these compression algorithms could halve our image file sizes (but we don't use them) #SoMEpi
Source citation (APA Format)	JentGent (Director). (2024, August 17). *These compression algorithms could halve our image file sizes (but we don't use them) #SoMEpi* [Video recording]. https://www.youtube.com/watch?v=RFWJM8JMXBs
Original URL	https://www.youtube.com/watch?v=RFWJM8JMXBs
Source type	YouTube Video
Keywords	Arithmetic Numeral Systems, Data Compression, Arithmetic Coding, Huffman Coding, Entropy, Source Coding Theorem.
#Tags	#LosslessCompression #ArithmeticCoding
Summary of key points + notes (include methodology)	Arithmetic coding intakes a sequence of characters and an independent probability for each symbol in the sequence to appear. Arithmetic coding differs from Huffman coding since bits can represent multiple characters and can encode data into non-integer numbers of bits. Typically, data is transformed to look like a Discrete Memoryless Source (DMS) to better match the probability model. Source Coding Theorem states that a piece of data can get close to but never be less than entropy, the amount of information you need to learn on average about a random variable to know what it is. It encodes more frequent information into few bits. Entropy coders try to encode a piece of information with I information content into I bits. Arithmetic coding differs from ANS as to prevent sequences like “00123” and “123" from encoding to be the same, it uses an interval from 0 to 1 while ANS adds non-zero numbers onto the beginning. An entire message gets encoded into a single number by dividing a range into the different possible symbols, where the interval for each symbol changes depending on probability. The encoder encodes an entire message by selecting the exact number that perfectly falls into each symbol interval subdivision in the message. To prevent numbers with infinite expansions infinitely encoding, an EOF symbol is added to

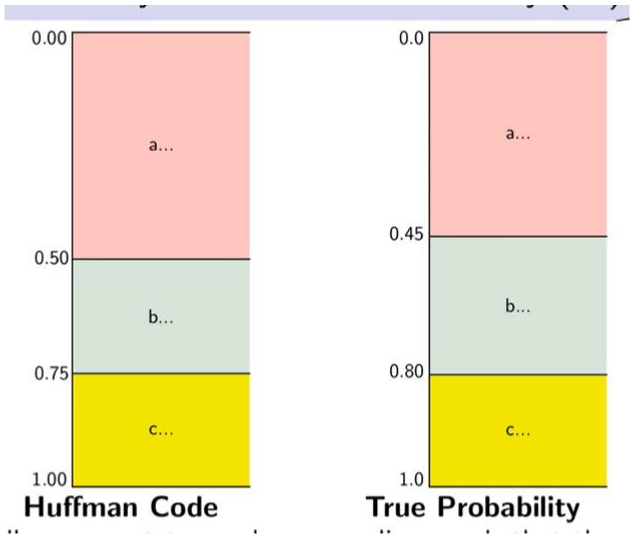
	the subdivisions. Each subdivision, it is checked if the code lies entirely in one half, at which point it renormalizes and repeats until it doesn't, allowing further symbol splicing. Decryption follows the exact, inverse cycle. Probabilities can shift dynamically in arithmetic encoding, which is where it relates a lot to neural compression. ANS works with whole numbers, where it spaces the same symbol on a number line proportional to that symbol's probability. To encode and decode, the symbol is counted x times until a final encoding/original sequence is reached. ANS isn't adaptive like arithmetic coding, but other forms like rANS and uANS were created to combat this.
Research Question/Problem/ Need	What are and ow do various forms of entropy coding work, specifically arithmetic coding?
Important Figures	 The interval system which arithmetic coding uses. Arithmetic coding aims for a certain interval to get the right number to have the correct string.
VOCAB: (w/definition)	Entropy – average number of bits per symbol based on information content.
Cited references to follow up on	Reducible (Director). (2021, July 30). *Huffman Codes: An Information Theory Perspective* [Video recording]. https://www.youtube.com/watch?v=B3y0RsVCyrw
Follow up Questions	How exactly does arithmetic coding incorporate into learned compression? Why exactly is arithmetic coding better than ANS in compression?

Article #13 Notes: Data Compression (Summer 2020) - Lecture 14 - Arithmetic Coding I

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	Data Compression (Summer 2020) - Lecture 14 - Arithmetic Coding I
Source citation (APA Format)	BillBird (Director). (2021, June 27). *Data Compression (Summer 2020) - Lecture 14 - Arithmetic Coding I* [Video recording]. https://www.youtube.com/watch?v=tLGo0Pfv9zQ
Original URL	https://www.youtube.com/watch?v=tLGo0Pfv9zQ
Source type	YouTube Video
Keywords	Arithmetic Coding, Huffman Coding, Entropy Coding, Bits, Entropy.
#Tags	#CompressingToEntropyWithArithmeticCoding #Intervals
Summary of key points + notes (include methodology)	Huffman coding gives you entropy encoding when the information content of every symbol is an integer, which is when probability is a negative power of 2. Huffman coding can be off by an average near bit per symbol, as sometimes information content is very small, much less than one, but Huffman still requires at least one bit per symbol. As the frequency of a symbol becomes higher, information becomes less as it's more guaranteed, and thus encoding should be smaller. Huffman coding is the best you can do for encoding one symbol. Huffman coding fails when doing multiple symbols as when you get large sequences; the combination becomes far too many to build the tree. Fractional values as used in arithmetic coding tell you things about the number you are being sent serially that integer values like in Huffman coding can't immediately. Also leading zeroes are insignificant with integers, which is bad, because it's scrapping possible information. Relates to arithmetic coding as when you are given information, you are given some set of 0s and 1s which is then translated into a new set where 0s and 1s don't necessarily indicate a number system, just information, and so to decode that, leading zeroes are significant. Unlike Huffman, each probability gets an accurate and exact amount of real estate on an interval, so no symbol is having its probability crammed into a smaller space which creates longer sequences for equal and proper distributions. Arithmetic coding approaches entropy with time; Huffman doesn't give you better entropy, but errors add over time. When the value falls in the range that represents the character combination,

	representative values can be chosen for just those first characters, without care for what comes next.
Research Question/Problem/ Need	How does arithmetic coding work, and how does it differ from Huffman?
Important Figures	 In Huffman code, the intervals are of fixed, powers of two sizes, which leads to overrepresentation of certain character combinations. This leads to large encodings, which is overall less efficient. Arithmetic coding fixes this issue, the true probability being the representation.
VOCAB: (w/definition)	Huffman coding – A method of entropy coding, refer to 11th article. True probability – the actual probability that a symbol has to appear. Prefix coding – A system in entropy coding by which all binary sequences are used either as a codeword or as a prefix to one to ensure the smallest possible codewords without having symbols that overlap.
Cited references to follow up on	N/A
Follow up Questions	Where exactly do the numbers plug into the interval come from? How does a letter sequence turn into something to use the interval for? How do you implement it?

Article #14 Notes: LLMZip: Lossless Text Compression using Large Language Models

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	LLMZip: Lossless Text Compression using Large Language Models
Source citation (APA Format)	Valmeekam, C. S. K., Narayanan, K., Kalathil, D., Chamberland, J.-F., & Shakkottai, S. (2023). *LLMZip: Lossless Text Compression using Large Language Models* (No. arXiv:2306.04050). arXiv. https://doi.org/10.48550/arXiv.2306.04050
Original URL	https://doi.org/10.48550/arXiv.2306.04050
Source type	Journal Article
Keywords	LLaMA, Large Language Model, Neural Compression, Asymptotic Upper Bound
#Tags	#LLMs #LLaMA #NeuralCompression
Summary of key points + notes (include methodology)	Learning, prediction, and compression are highly correlated, as explored by Shannon in 1951, giving machine learning a good foothold in data compression. Paper studies whether or not large language models can be used to compress to entropy in the English language. As with other models, the LLM splits a sequence into tokens, then scans through it to find the probability distribution of the next token using context, one at a time. With the probability distribution, it finds the ranking of the correct token and uses this probability ranking in the compression of the file. To decompress the file, it goes through decompression, then back through the LLM to find the correct tokens with their probabilities from the rankings. Many formulas are used to calculate entropy and the asymptotic upper bound. Recently, the bits per character of entropy and the upper bound of neural compressors have outperformed that of traditional compressors. The probabilities of each symbol are time-varying, which makes arithmetic coding a perfect candidate for encoding over options like Huffman coding. Statistical properties should be kept in mind when thinking about the entropy of various models using a finite number of tokens due to large alphabets and large memories. The article used LLaMA-7B, a pretrained LLM off of all sorts of characters, whereas the input was just lowercase letters, which may have slightly skewed results. The LLM was able to achieve entropy rates much greater than that of state of the art compressors like ZPAQ on a fraction of the text8 dataset (100 MB). The text equated to around 10 batches of 100,000 tokens. The authors note that running

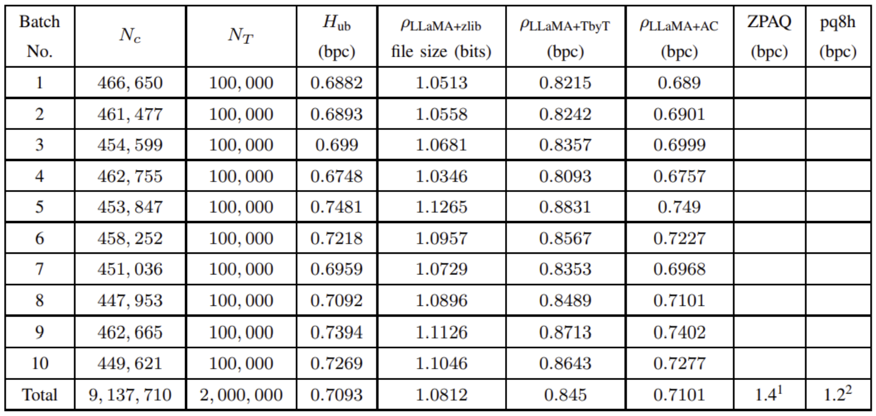
with a memory of 511 tokens was 16 times slower than batches with memory of 31 tokens.

Research Question/Problem/ Need

Can better compression results and better estimates of entropy of the English language be achieved using recent large language models?

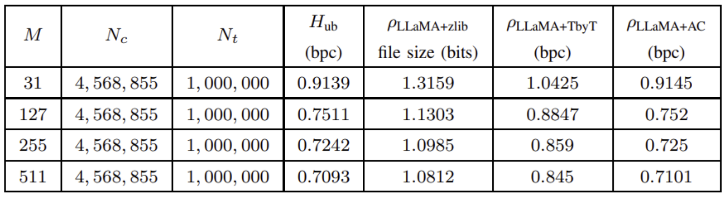
Important Figures

TABLE I
RESULTS FOR 1MB OF TEXT FROM TEXT8 DATASET



Batch No.	N_c	N_T	H_{ub} (bpc)	$\rho_{LLaMA+zlib}$ file size (bits)	$\rho_{LLaMA+TbyT}$ (bpc)	$\rho_{LLaMA+AC}$ (bpc)	ZPAQ (bpc)	pq8h (bpc)
1	466, 650	100, 000	0.6882	1.0513	0.8215	0.689		
2	461, 477	100, 000	0.6893	1.0558	0.8242	0.6901		
3	454, 599	100, 000	0.699	1.0681	0.8357	0.6999		
4	462, 755	100, 000	0.6748	1.0346	0.8093	0.6757		
5	453, 847	100, 000	0.7481	1.1265	0.8831	0.749		
6	458, 252	100, 000	0.7218	1.0957	0.8567	0.7227		
7	451, 036	100, 000	0.6959	1.0729	0.8353	0.6968		
8	447, 953	100, 000	0.7092	1.0896	0.8489	0.7101		
9	462, 665	100, 000	0.7394	1.1126	0.8713	0.7402		
10	449, 621	100, 000	0.7269	1.1046	0.8643	0.7277		
Total	9, 137, 710	2, 000, 000	0.7093	1.0812	0.845	0.7101	1.4[1]	1.2[2]

TABLE II
COMPRESSION PERFORMANCE OF THE LLM ON THE TEXT8 DATASET, AS A FUNCTION OF ITS MEMORY (M)



M	N_c	N_t	H_{ub} (bpc)	$\rho_{LLaMA+zlib}$ file size (bits)	$\rho_{LLaMA+TbyT}$ (bpc)	$\rho_{LLaMA+AC}$ (bpc)
31	4, 568, 855	1, 000, 000	0.9139	1.3159	1.0425	0.9145
127	4, 568, 855	1, 000, 000	0.7511	1.1303	0.8847	0.752
255	4, 568, 855	1, 000, 000	0.7242	1.0985	0.859	0.725
511	4, 568, 855	1, 000, 000	0.7093	1.0812	0.845	0.7101

From the table, we can see that LLMZIP comes much closer to the entropy limits than other compressors. It always outperforms the compared compressors in bits per channel.

VOCAB: (w/definition)

Large Language Model – A machine learning algorithm that specializes on text data. It works on a transformer model and tries to predict the next token.
Recurrent Neural Network – A machine learning algorithm based off a feed forward network with enhanced memory using gates.
Asymptotic upper bound – The upper bound for which something can be encoded within.

Cited references to follow up on

Cover, T., & King, R. (1978). A convergent gambling estimate of the entropy of English. *IEEE Transactions on Information Theory*, *24*(4), 413–421. https://doi.org/10.1109/TIT.1978.1055912

	Lutati, S., Zimerman, I., & Wolf, L. (2023). *Focus Your Attention (with Adaptive IIR Filters)* (No. arXiv:2305.14952). arXiv. https://doi.org/10.48550/arXiv.2305.14952
Follow up Questions	How exactly are the probability distributions fetched from the pre-trained LLM? Why LLaMA? How would lower parameter models affect the results? How does the arithmetic coder incorporate into the model?

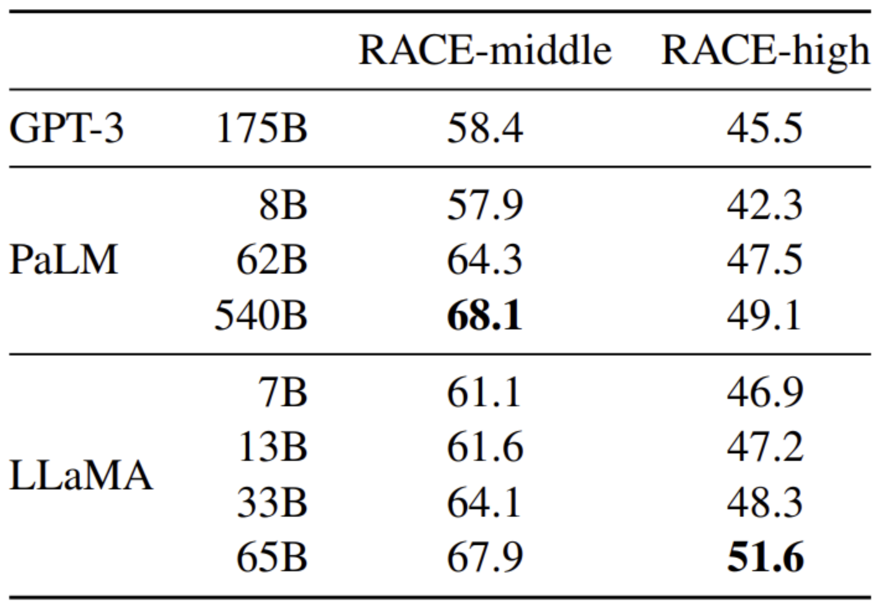
Article #15 Notes: LLaMA: Open and Efficient Foundation Language Models

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	LLaMA: Open and Efficient Foundation Language Models
Source citation (APA Format)	Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models* (No. arXiv:2302.13971). arXiv. https://doi.org/10.48550/arXiv.2302.13971
Original URL	https://doi.org/10.48550/arXiv.2302.13971
Source type	Journal Article
Keywords	Large Language Model, Tokens, GPU, Transformer.
#Tags	#LLaMA #LLM
Summary of key points + notes (include methodology)	Given a compute budget, best performances are not achieved by larger models, but rather by smaller models trained on more data. Hoffman tries to determine how best to scale the dataset and model sizes for a training budget, but not the inference budget. A smaller model trained for longer will ultimately be cheaper at inference, making smaller, properly trained models better than bigger ones. The authors train LLaMA on more tokens than often used to produce better inference results. The authors use commonly used datasets for training the LLM, but they have the restriction of only using publicly available ones, so the model can be open sourced. Some datasets include English CommonCrawl, Github, Wikipedia in various languages, and ArXiv, giving the model broad capabilities, however some sources were weighed heavier than others. The training dataset included roughly 1.4 trillion tokens. The model uses the transformer architecture, implementing some improvements used in other LLMs. The model is trained using the AdamW optimizer. Training a 65 billion parameter model took around 21 days with 380 tokens/sec/GPU on 2048 A100 GPUs with 80GB of RAM. The authors test LLaMA on various settings and compare them to other LLMs. In common sense reasoning, and despite a significant size decrease in comparison to other models, LLaMA outperforms on many benchmarks, such as outperforming ChatGPT-3. In closed-

	book question answering, LLaMA achieves state-of-the-art performance and competes with much larger models. LLaMA achieves similar or better performance in reading comprehension, mathematical reasoning, and code generation. The model does do worse in massive multitask language understanding. The authors shows that slightly instruction finetuning can make significant differences in results, showing that finetuning for data compression may lead to better compressions. The LLaMA models, however, are shown to be biased. This discourages use of the model as it means that compression ratio can become worse for biased pieces of media. Additionally, the model had significant carbon emissions and energy usage during training, however this should not affect inference energy usage. The authors plan to release larger models trained on larger pretraining corpora when they released the paper in 2023, which might be promising to use for my project. Considering multiple models have been released after this paper was published, there is a good chance some issues were resolved and further, improved results have been produced.
Research Question/Problem/ Need	Current models disregard the inference budget, instead prioritizing training, finding the optimal dataset and model size for a training budget. To train a series of language models that achieve the best possible performance at various inference budgets. Additionally, to publish them for open source use.
Important Figures	



		RACE-middle	RACE-high
GPT-3	175B	58.4	45.5
PaLM	8B	57.9	42.3
	62B	64.3	47.5
	540B	**68.1**	49.1
LLaMA	7B	61.1	46.9
	13B	61.6	47.2
	33B	64.1	48.3
	65B	67.9	**51.6**

Table 6: **Reading Comprehension.** Zero-shot accuracy.

This shows that LLaMA performs or competes significantly with much larger models on zero shot accuracy in reading comprehension.

VOCAB: (w/definition)	Zero-shot – 0 examples to model off. Few-shot – Few examples to model off. Large Language Model - A machine learning algorithm that specializes on text data. It works on a transformer model and tries to predict the next token.
Cited references to follow up on	Brants, T., Popat, A., Xu, P., Och, F. J., & Dean, J. (2007, June 1). *Large language models in machine translation*. ACL Anthology. https://aclanthology.org/D07-1090/ Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., . . . Fiedel, N. (2022, April 5). *PaLM: Scaling Language Modeling with Pathways*. arXiv.org. https://arxiv.org/abs/2204.02311
Follow up Questions	Are other models biased? If so, how does this model scale? How much of an impact can model bias make on compression?

Article #16 Notes: A Comparative Survey of Large Language Models: Foundation, Instruction-Tuned, and Multimodal Variants

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

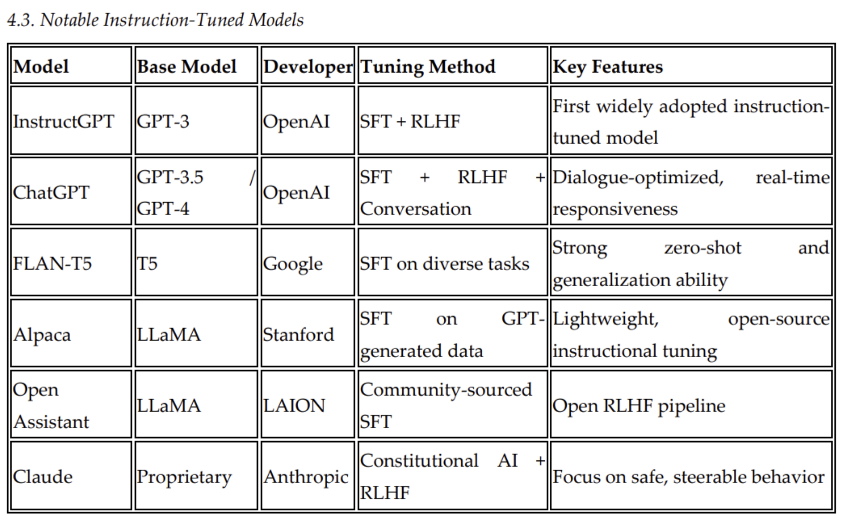
Source Title	A Comparative Survey of Large Language Models: Foundation, Instruction-Tuned, and Multimodal Variants
Source citation (APA Format)	Balford, J. (2025). *A Comparative Survey of Large Language Models: Foundation, Instruction-Tuned, and Multimodal Variants* (No. 2025061134). Preprints. https://doi.org/10.20944/preprints202506.1134.v1
Original URL	https://doi.org/10.20944/preprints202506.1134.v1
Source type	Journal Article
Keywords	Large Language Models, Multimodal Language Models.
#Tags	#LLMs #MLLMs
Summary of key points + notes (include methodology)	LLMs are statistical pattern learners. All they do is estimate the probability distribution of the next token appearing over a sequence of tokens, generating an appropriate output. There are many present issues with LLMs like hallucinations and following user intent, however most of these issues should be nulled in the implementation of my project. Hallucinations will play a significant role in data not properly compressing, however this is the purpose of incorporating further check steps for further compression. LLMs capture complex patterns and long range dependencies in text through self-attention. LLMs are presently pushing the boundaries, with early successes turning into higher model sizes and training data and such. Many models rely on billions of parameters and billions to trillions of tokens. BERT by Google seems to be an outlier, however, with just 340 million parameters and only 2 datasets being used, though it is much older. Instruction-tuning can better help models understand human intent. They started out with a general model and then refine it with instructions that pair completely to desired outputs. Multimodal language models (MLLMs) extend the abilities of LLMs to other tasks, essential for creating general purpose AI. They are still built around transformers, but handle different pieces of data different, meaning LLMs are more of a section than a base. There are many different ways people are trying to build multimodal information into LLMs, allowing for more overall purpose. They

may become significant if during the project I choose to diverge from just encoding text to also compressing images or videos. There are many issues such as bias, factuality, and sustainability.

Research Question/Problem/ Need

How are current Large Language Models structured, what are their strengths and flaws, and how can they be improved?

Important Figures



4.3. Notable Instruction-Tuned Models

Model	Base Model	Developer	Tuning Method	Key Features
InstructGPT	GPT-3	OpenAI	SFT + RLHF	First widely adopted instruction-tuned model
ChatGPT	GPT-3.5 / GPT-4	OpenAI	SFT + RLHF + Conversation	Dialogue-optimized, real-time responsiveness
FLAN-T5	T5	Google	SFT on diverse tasks	Strong zero-shot and generalization ability
Alpaca	LLaMA	Stanford	SFT on GPT-generated data	Lightweight, open-source instructional tuning
Open Assistant	LLaMA	LAION	Community-sourced SFT	Open RLHF pipeline
Claude	Proprietary	Anthropic	Constitutional AI + RLHF	Focus on safe, steerable behavior

Different methods of tuning each model, which will likely come significant during my project. During my project, the model may be altered very slightly to be more text tuned for data compression purposes. The results of doing so will be observed.

VOCAB: (w/definition)

Zero-shot – 0 examples to model off.
Few-shot – Few examples to model off.

Cited references to follow up on

Pahune, S., & Chandrasekharan, M. (2023). Several categories of large language models (llms): A short survey. arXiv preprint arXiv:2307.10188.

Nokhwal, S., Chilakalapudi, P., Donekal, P., Nokhwal, S., Pahune, S., & Chaudhary, A. (2024, April). Accelerating neural network training: A brief review. In Proceedings of the 2024 8th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (pp. 31-35).

Follow up Questions

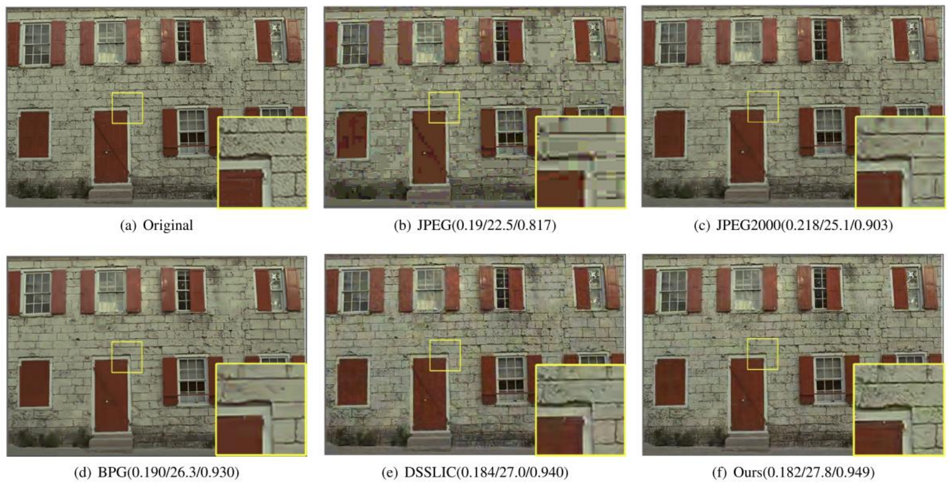
Could MLLMs be incorporated into data compression?
How would MLLM data compression be structured?
How computationally expensive are MLLMs in comparison to LLMs?

Article #17 Notes: Improved hybrid layered image compression using deep learning and traditional codecs

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	Improved hybrid layered image compression using deep learning and traditional codecs.
Source citation (APA Format)	Fu, H., Liang, F., Lei, B., Bian, N., zhang, Q., Akbari, M., Liang, J., & Tu, C. (2020). Improved Hybrid Layered Image Compression using Deep Learning and Traditional Codecs. *Signal Processing: Image Communication, 82,* 115774. https://doi.org/10.1016/j.image.2019.115774
Original URL	https://doi.org/10.1016/j.image.2019.115774
Source type	Journal Article
Keywords	JPEG, Kodak, Neural Image Compression, generative adversarial network, convolutional autoencoder, convolutional neural network.
#Tags	#LearnedImageCompression #Traditional+Neural
Summary of key points + notes (include methodology)	Since the successful application of convolutional neural networks in computer vision, neural image compression boomed, outperforming some traditional compressors. There have been many proposed compressors, especially those that combine many aspects to outperform codecs like JPEG. Quantization, one form of compression, has hindered compression of neural networks, showing how some compression can be directional and limit the overall. Many different compressors were proposed to fix distortion and dimensionality, by adding various things and adding layers; however, much work is still to be done as details are still not well preserved. Additionally, semantic segmentation sometimes fails, leading the authors to develop a hybrid coding scheme that is simplified. The network consists of two layers, which is used, and then the text is encoded by the FLIF codec, then it goes through another network, forming a computation autoencoder. Finally, it goes through the lossy BPG codec. The encoding and decoding phases are not completely uniform, like in text compression. During the training phase, both of the neural networks are trained: this idea of many codecs may be applicable to text data compression. The ultimate goal is to reproduce the image as close to the original as possible, but it's never going to be exact because some data was discarded. The Kodak and Tecnick datasets were used as one test, and it was often

	found that JPEG and other common or high-end compressors failed in detail compared to the work of the authors. The authors made many changes in the autoencoder network, the clipping method, which grants them great results. The combined use of various tools along with slight modifications to certain models and understanding directionality are all things that may lead to smaller compressions in text compression.
Research Question/Problem/ Need	Will hybrid models and changes to lossy compressors help increase overall image compression?
Important Figures	 (b) JPEG(0.19/22.5/0.817) (c) JPEG2000(0.218/25.1/0.903) (d) BPG(0.190/26.3/0.930) (e) DSSLIC(0.184/27.0/0.940) (f) Ours(0.182/27.8/0.949) **Fig. 9.** Example 1 in the Kodak dataset (bits/pixel/channel, PSNR, MS-SSIM). A comparison of image quality after compression. The article's compression has significant detail retained, which is good as it allows the user to see better their image with a compressed file size still.
VOCAB: (w/definition)	Codec – A compression decompression algorithm. Quantization – A method of image compression where a continuum is simplified into discrete values.
Cited references to follow up on	Rippel, O., & Bourdev, L. (2017). Real-Time Adaptive Image Compression. *Proceedings of the 34th International Conference on Machine Learning*, 2922–2930. https://proceedings.mlr.press/v70/rippel17a.html Ballé, J., Minnen, D., Singh, S., Hwang, S. J., & Johnston, N. (2018). *Variational image compression with a scale hyperprior* (No. arXiv:1802.01436). arXiv. https://doi.org/10.48550/arXiv.1802.01436
Follow up Questions	What parts of image compression might be the most significant in text

	compression? Can certain methods and networks in image or text compression be used in the other? Can an absolutely universal, completely efficient compressor be made? Can lossy compression be applied to text data?

Article #18 Notes: DEFLATE Compressed Data Format Specification version 1.3

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	DEFLATE Compressed Data Format Specification version 1.3
Source citation (APA Format)	RFC 1951 DEFLATE Compressed Data Format Specification ver 1.3. (n.d.). Retrieved December 18, 2025, from https://www.w3.org/Graphics/PNG/RFC-1951#introduction
Original URL	https://www.w3.org/Graphics/PNG/RFC-1951#introduction
Source type	Online Article
Keywords	LZ77, Huffman coding, DEFLATE, dynamic, bytes.
#Tags	#DEFLATE #Compress
Summary of key points + notes (include methodology)	The authors of the article are not attempting to allow random access to compressed data or compress specialized data like a specialized codec. Bytes do not have a bit order; they are always treated as one unless you let it represent an integer, in which case the left most bit is the most significant, and the right most bit is the least. Multi-byte numbers in the paper are stored the opposite way, where the least significant byte comes first and the most significant is last. All data is considered at the byte rather than bit level, differing from arithmetic coding. Significance of the bit from left to right can allow parsing from right to left, with the correct MSB-LSB order and Huffman codes in reverse order. The algorithm creates optimal prefix codes by giving the Huffman coder the symbol frequencies. Additional constraints to the Huffman coder in this paper is it must have lexicographically consecutive values and shorter codes precede longer ones. This changes a tree from A – 00, B – 1, C – 011, D – 010 to A – 10, B – 0, C – 110, D – 111. Using these rules, a Huffman code for an alphabet can be defined using just the bit lengths of the codes for each symbol in order. In implementation, this is three simple steps: ☐ Count # of codes for each code length. ☐ Find the value of smallest code. ☐ Assign values to all used codes, with codes that are the same length getting consecutive values with values in the previous step. Every block of compressed data has 3 header bits that define whether the current block is the last block of data and how the data should be compressed, either it

	wasn't, fixed Huffman codes, dynamic Huffman codes, or it errored. The two compressions differ by how the Huffman codes are defined. The authors had the intent not to reference any particular compression algorithms in "deflate," however LZ77 is closely related. A block is terminated when the compressor decides it useful to start a new block or the buffer fills up. The compressor can become slower and faster depending on using lazy matching or performing longer searches. There are some large impacts if the data gets corrupted, often losing the file unfortunately. As such, the author recommends some way of validating the integrity of the data.
Research Question/Problem/ Need	There are few universal compression algorithms that are able to compress data significantly. The authors of the article aim to create a better universal compression algorithm that can run efficiently on any computing architecture.

Important Figures

	Extra			Extra			Extra	
Code	Bits	Dist	Code	Bits	Dist	Code	Bits	Distance
----	----	----	----	----	------	----	----	--------
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

The amount of bits used for a code based on its distance.

VOCAB: (w/definition)	Byte – A sequence of 8 bits that is used to represent digital information. Huffman coding - A method of entropy coding, refer to 11th article.

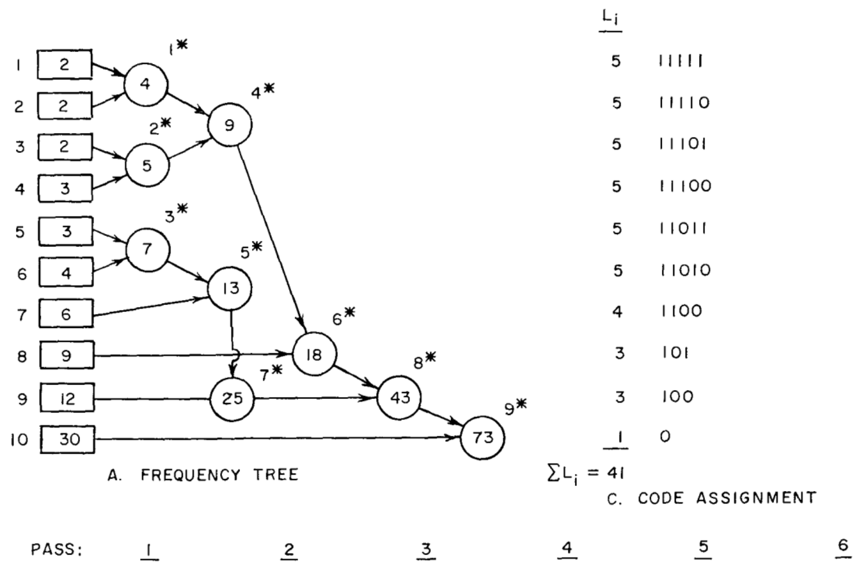
Cited references to follow up on	Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, *40*(9), 1098–1101. https://doi.org/10.1109/JRPROC.1952.273898 Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, *23*(3), 337–343. https://doi.org/10.1109/TIT.1977.1055714
Follow up Questions	Why is random access to compressed data a significant thing to address, or rather not address? What is the maximum code length that DEFLATE can operate on?

Article #19 Notes: Generating a Canonical Prefix Encoding

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	Generating a Canonical Prefix Encoding
Source citation (APA Format)	Schwartz, E. S., & Kallick, B. (1964). Generating a canonical prefix encoding. Communications of the ACM, 7(3), 166–169. https://doi.org/10.1145/363958.363991
Original URL	https://doi.org/10.1145/363958.363991
Source type	Journal Article
Keywords	Prefix Encoding, Huffman Coding, Frequency Trees.
#Tags	#CanonicalPrefixEncoding
Summary of key points + notes (include methodology)	The average length of the codes of symbols are equal to the sum from 1 to n of p_i * L_i, where L_i is the length of the code of symbol w_i and p_i is its relative frequency. Huffman coding ensures that every possible sequence of binary digits is used as a code or prefix for a code. This is an older article, so arithmetic coding had yet to be invented or popularized. Huffman trees generate canonical prefix encodings for its frequency trees. In the tree, the correct encodings, the correct probabilities, must be combined into nodes of a tree to get the optimum encoding. The goal is to obtain a minimum average length, minimum sum of code lengths, and minimum maximum L. There are four steps for encoding: preparation of rank-frequency table, generation of frequency tree, determination of code breakpoints, and assignment of codes. In the first step, the dictionary of frequencies is sorted and ranked. Then the frequency tree is generated where the two lowest frequencies are always merged. The node table will be generated with n - 1 entries. The tree will generate as many binary digits as it takes to go from the root of the tree to a certain digit. Breakpoints are generated with a simple check through the starred values in the node table, which is a value that represents the combination of two merged frequencies. A compact and dense set of codes are generated, using every successive storage location, assigned at break points, to compact storage. This routine is beneficial as only frequency distribution is needed, not minimum cost or anything else. Using these methods, researchers secure incredible efficiency and compressions for the time.
Research	What is the full process of an encoder that uses canonical prefixes?

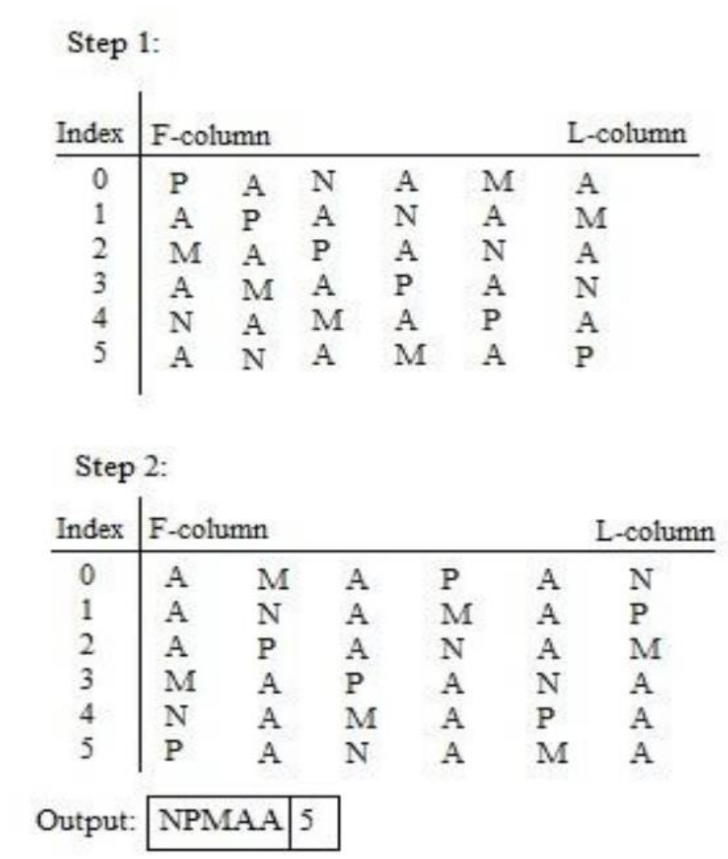
Question/Problem/ Need	
Important Figures	 A. FREQUENCY TREE C. CODE ASSIGNMENT PASS: 1 2 3 4 5 6 L_i 5 11111 5 11110 5 11101 5 11100 5 11011 5 11010 4 1100 3 101 3 100 1 0 $\Sigma L_i = 41$ A representative frequency tree from the article.
VOCAB: (w/definition)	Huffman coding – A method of entropy coding, refer to the 11th article. Frequency distribution – The probability that a certain symbol is going to appear in a given place. Rank-frequency table – The ranking of what symbol is most probable to come next.
Cited references to follow up on	Karp, R. S. Minimum-redundancy coding for the discrete noiseless channel. *Trans. IRE*, *IT-7* (1961), 27-38. Schwartz, E. S. An adaptive information transmission system employing minimum-redundancy word codes. Armour Research Foundation, Techn. Doc. Rep. ASD-TDR-62-265, Pt. 11, June 1963
Follow up Questions	How does prefix encoding apply to other entropy encoders?

Article #20 Notes: A Survey: Different Loss-less Compression Techniques

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	A Survey: Different Loss-less Compression Techniques
Source citation (APA Format)	Azeem, S., Khan, A., Qamar, E., Tariq, U., & Shabbir, J. (2016). A Survey: Different Loss-less Compression Techniques. International Journal of Technology and Research, 4(1), 1–4.
Original URL	https://www.proquest.com/docview/2067959624/fulltext/7FB5E19F6E54D1EPQ/1?accountid=29120&sourcetype=Scholarly%20Journals
Source type	Journal Article
Keywords	LZA, LZ77, LZ78, Byte Pair Encoding, Burrows Wheeler Encoding.
#Tags	#LZA #BurrowsWheeler #BytePairEncoding
Summary of key points + notes (include methodology)	Burrows Wheeler encoding consists of three consecutive main stages. The size of the data must be specified before compression. During the first step, BWT, characters are shifted from left to right. The number of steps is equal to the number of characters in the input. This burrows wheeler matrix is sorted alphabetically, and identical characters are combined. The second stage is move to front (MTF), where the most frequent letters are moved to the front of the global list of characters. Finally, Source Coding Algorithm, where Zero Run Length Code (in the article) is used to encode. Based on the number of zeroes, each symbol is assigned a certain code. The exact inverse can be done for decoding. Byte pair encoding is where repeated patterns are found in the text and replaced by a code. The main issue is searching for patterns, especially as file size increases. Byte pair encoding is very simple, simply looping through the text until no more patterns remain and the data is fully entropic. Finally, Lempel-Ziv Coding (LZA) is a group of algorithms. This is a dictionary based algorithm, where patterns are stored in a dictionary and used as a reference for each occurrence in the text. The algorithm uses a sliding window to determine inputs, having a search buffer and look ahead buffer. Decompression is easy, simply following pointers for keys, however compression can be a bit slow. This does mean that the dictionary can grow infinitely. LZ78 is the faster version in comparison to LZ77.
Research Question/Problem/ Need	How do various lossless data compression algorithms work?

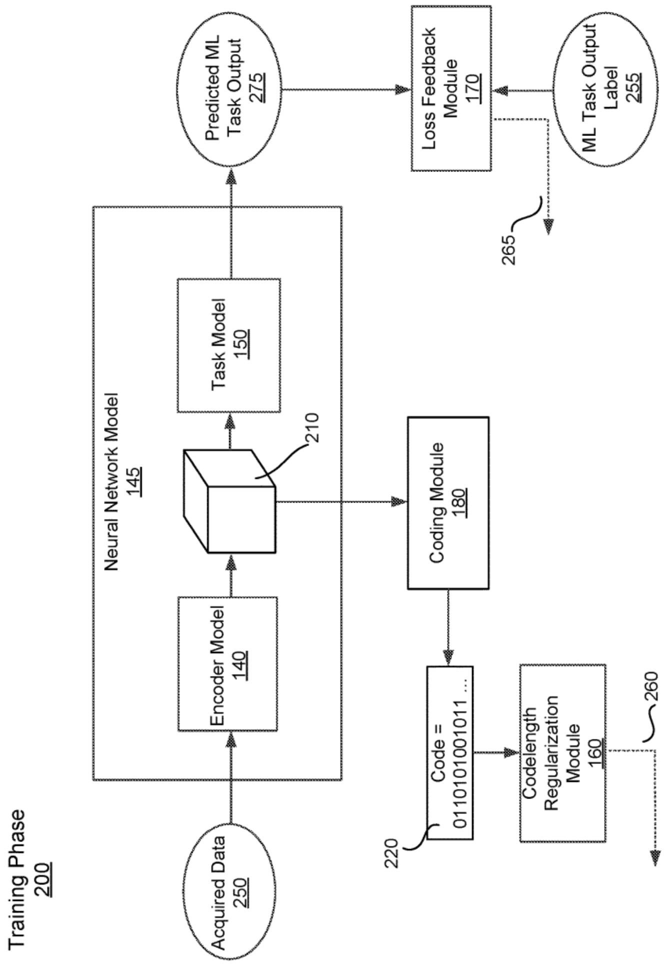
Important Figures	 Step 1: Index | F-column L-column 0 | P A N A M A 1 | A P A N A M 2 | M A P A N A 3 | A M A P A N 4 | N A M A P A 5 | A N A M A P Step 2: Index | F-column L-column 0 | A M A P A N 1 | A N A M A P 2 | A P A N A M 3 | M A P A N A 4 | N A M A P A 5 | P A N A M A Output: NPMAA | 5 The indices of the letters being set to one as the move to front algorithm scans from the sequence.
VOCAB: (w/definition)	Sliding window – A window in which everything is analyzed, which slides down, discarding old symbols for new ones.
Cited references to follow up on	Senthil S, Robert L., "A Comparative Study Of Text Compression Algorithms"., International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011, pp. 68-76. Yehoshua P, Venkat M, Nageshwar K., "THE CASCADING OF THE LZW COMPRESSION ALGORITHM WITH ARITHMETIC CODING", pp. 277-287.
Follow up Questions	Of these, is there one that encodes closer to entropy than the others?

Patent #1 Notes: Data compression for machine learning tasks

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

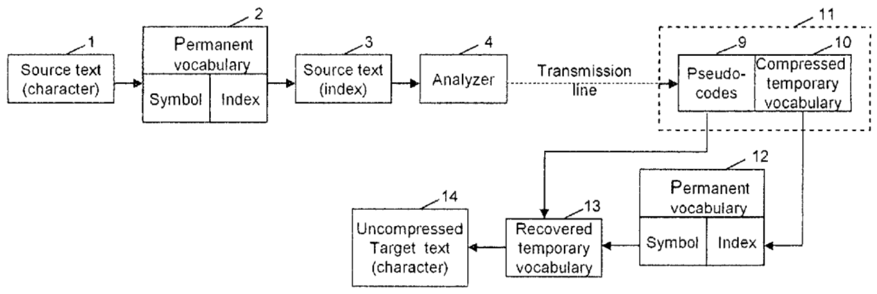
Source Title	Data compression for machine learning tasks
Source citation (APA Format)	Bourdev, L., Lew, C., Nair, S., & Rippel, O. (2022). *Data compression for machine learning tasks* (United States Patent No. US11256984B2). https://patents.google.com/patent/US11256984B2/en?q=(neural+text+compression)&oq=neural+text+compression
Original URL	https://patents.google.com/patent/US11256984B2/en?q=(neural+text+compression)&oq=neural+text+compression
Source type	Patent
Keywords	Data compression, loss.
#Tags	#DataCompression
Summary of key points + notes (include methodology)	A neural network is trained using error terms obtained from a loss function in gradient descent. In training, sends the data from training as input to the encoder for the compressed representation to then be calculated for loss through comparing codelengths to target codelengths. The target codelength is set based on available bandwidth. The network will attempt to maximize the accuracy of task outputs. It will also minimize codelengths while maintaining accuracy. Prior to encoding or anything, the neural network must first find the significant data with in the file. It will undergo extraction, finding the important data to be compressed, whether that be part of an image, text file, or video. It will then be sent to the encoder. Once compressed, the machine learning algorithm will run upon the compacted data in a “task” for a prediction. This then goes through the loss process where all that is done is loss is calculated.
Research Question/Problem/ Need	How can a general-purpose learned compressor be created to compact data more significantly than current processors?

Important Figures	 Training Phase 200 Acquired Data 250 Encoder Model 140 Neural Network Model 145 210 Task Model 150 Predicted ML Task Output 275 Loss Feedback Module 170 ML Task Output Label 255 265 Coding Module 180 Code = 0110101001011 ... 220 Codelength Regularization Module 160 260 FIG. 2 The structure of the patented algorithm.
VOCAB: (w/definition)	Loss – how far the model is off from its goal.
Cited references to follow up on	Dony et al . , “ Neural Network Approaches to Image Compression ” , Feb. 1995 , Proceedings of the IEEE , vol . 83 , No. 2 , pp . 288-303 (Year : 1995) .
Follow up Questions	Why feed the encodings into the task model or the coding module and not specifically one?

Patent #2 Notes: Method and system for text compression and decompression

Article notes should be on separate sheets

KEEP THIS BLANK AND USE AS A TEMPLATE

Source Title	Method and system for text compression and decompression
Source citation (APA Format)	Grinblat, Z. D. (2012). *Method and system for text compression and decompression* (United States Patent No. US8332209B2). https://patents.google.com/patent/US8332209B2/en
Original URL	https://patents.google.com/patent/US8332209B2/en
Source type	Patent
Keywords	Data Compression, Vocabulary, Dictionary-Based Encoding.
#Tags	#DataCompression
Summary of key points + notes (include methodology)	The patent aims to create a program that can turn a sequence of symbols into a compressed text. It will do this by assigning certain parts of the text numerical representations that can then be turned into a compressed bitstream. It also implements the "permanent vocabulary," which is a large collection of preprocessed words. It serves as a permanent reference vocabulary for encryption and decryption. It will finally implement the temporary vocabulary. This contains information contained within the current text, storing common symbols. These symbols are stored in one of two parts, the root of the tree or the main storages. This overall has the effect of translating a high bit length string into a low bit one that is present in the temporary vocabulary.
Research Question/Problem/ Need	Can various vocabularies be implemented to create a more efficient, lossless compression algorithm?
Important Figures	 1 Source text (character) 2 Permanent vocabulary Symbol Index 3 Source text (index) 4 Analyzer Transmission line 11 9 Pseudo-codes 10 Compressed temporary vocabulary 12 Permanent vocabulary Symbol Index 13 Recovered temporary vocabulary 14 Uncompressed Target text (character)

VOCAB: (w/definition)	Bit – 1 or 0 that represents information. Dictionary – A data structure with key value pairs that given a key, the value can be referenced. Works as a regular dictionary, if you know what word you want to find, you can search up the definition, or in this case, the reference/value. Reference – Something that points to the memory address of something else.
Cited references to follow up on	D. Huffman, “A Method for the Construction of Minimum Redundancy Codes, in Proc. IRE, Vol. 40, no. 9, pp. 1098 1101, 1952. Gonzalo Navarro and Mathieu Raffinot. A General Practical Approach to Pattern Matching over Ziv-Lempel Compressed
Follow up Questions	Does the permanent vocabulary not become expensive over time?