Modern ML network designs

Robert Currie

Model Optimization

- Training advanced large models is slow.
- Most of the time spent training is mostly in shuffling data around for massive matrix multiplications.

- How can we optimize or speed this up?

- **Higher level languages -> lower-level languages**
  (Fewer instructions to complete)
- **Generic code -> Accelerator specific code**
  (More instructions per cycle)
- **Reducing precision -> Less time per operation**
  (More instructions per cycle)
- **Advanced Caching/Compression -> Rewriting the model itself**
  (Fewer instructions to compute)

Easy

Difficult

Hard

Computing Throughput and Precision

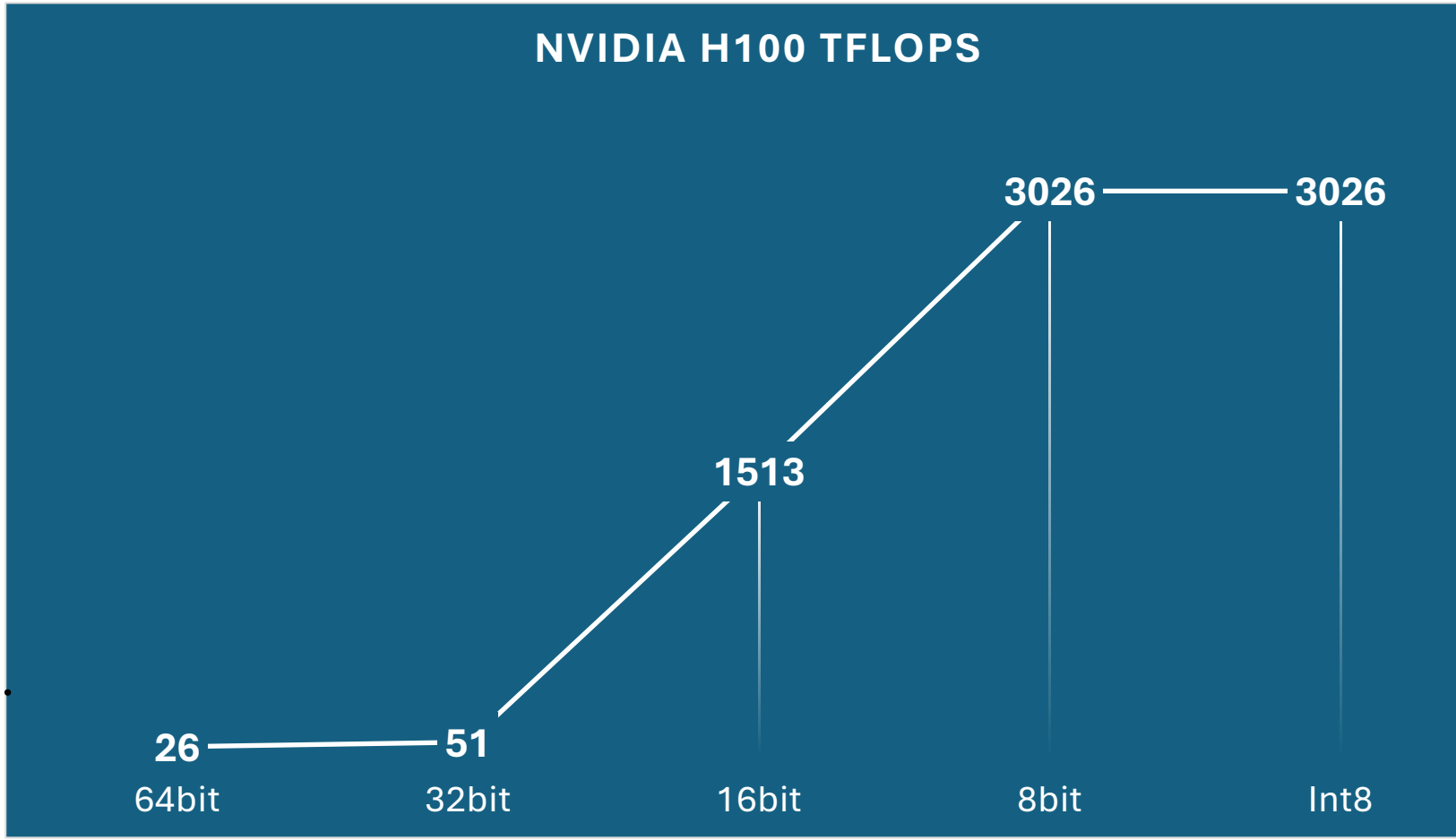
Intel i9 can perform 0.3TFLOPs @ 64bit. Best in class CPU.

GPUs can perform better.

nVidia arguably best in class.

Potentially 1,500x faster by reducing precision.

Reducing precision reduces, cost, power, thermal waste, ...



Computing Throughput and Precision

- Why can't we "just" use **8bit** models?
- Training models using only lower precision is more unstable.
- Production models may not be well-suited to using **FP8**.
  e.g. GAN accuracy can degrade rapidly with reduced precision.
- Larger models help restore accuracy of lower precision but are much more difficult to train.

Computing Throughput and Precision

- Modern approach to combat this is to use:

“Quantization Aware Training”

- This is when a full-precision model is trained with extra quantization aware components which gives enhanced stability.

- This also means that models need to be **“Quantized”** after training so that they can run in production.

Computing Throughput and Precision

- **QAT** models are often good in production where resources are limited.

- However, there's no "*free lunch*".

- Typically, these models have to be much wider in order to still store the same amount of information through training.

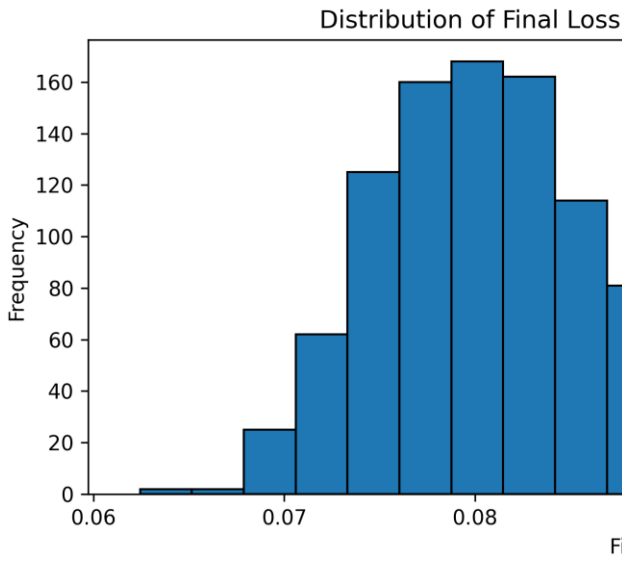
Now time for a Recap

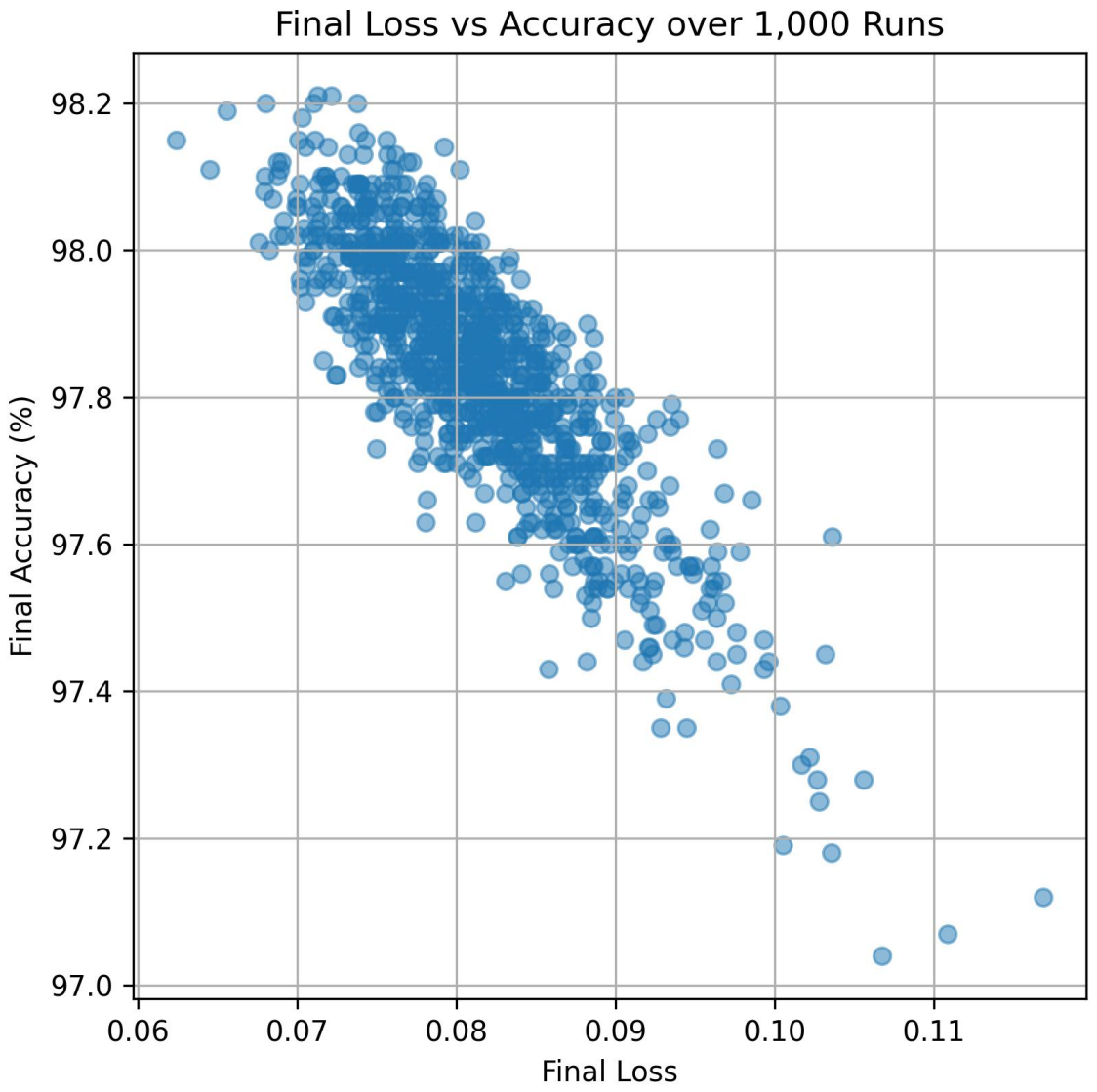
- We've covered how to build/train **DNN**, **CNN**, **VAE**, **GAN** and **conditional** models.

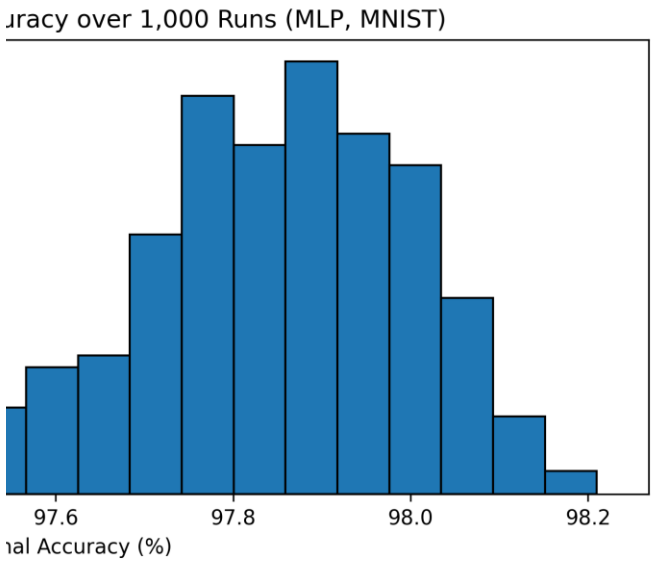
- We have covered how training in general works and how can improve this.

- We have not discussed pattern recognition or model long-range stability explicitly.

Complex Model Training – Luck

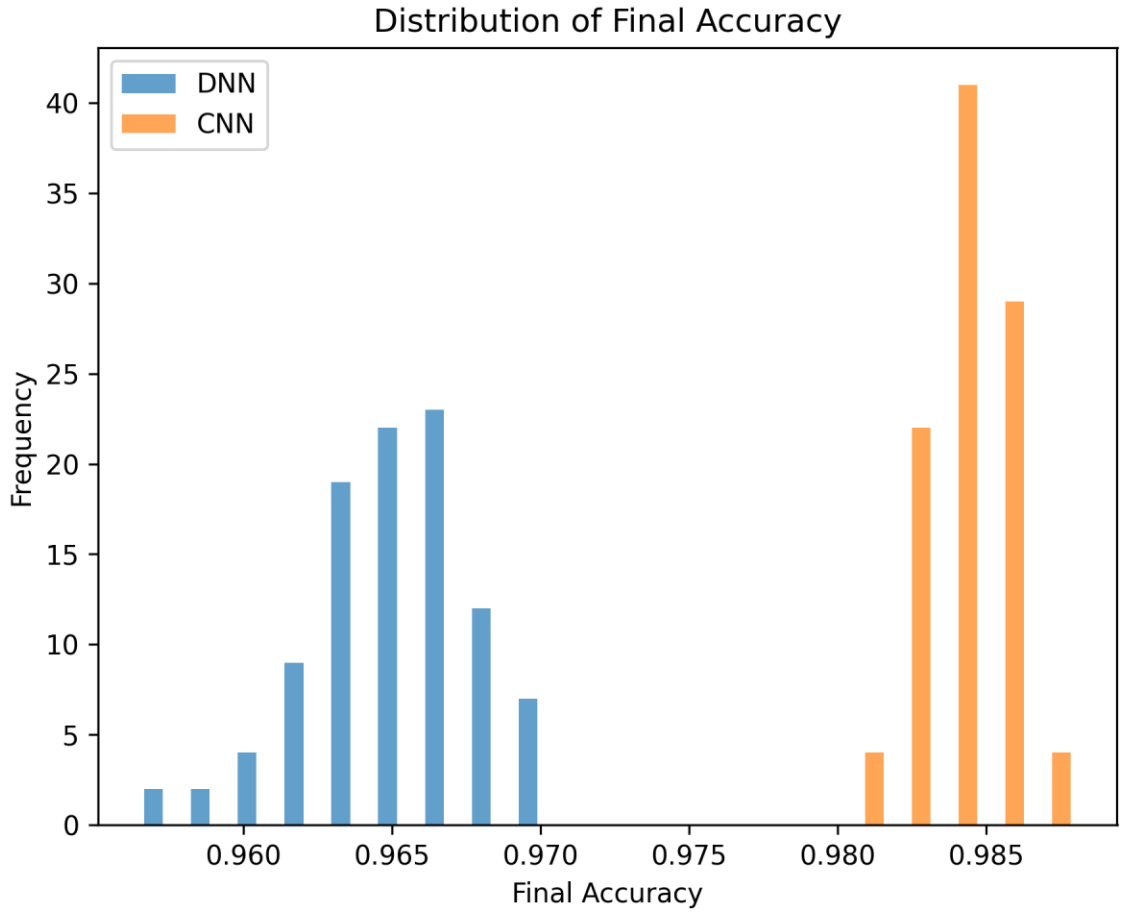


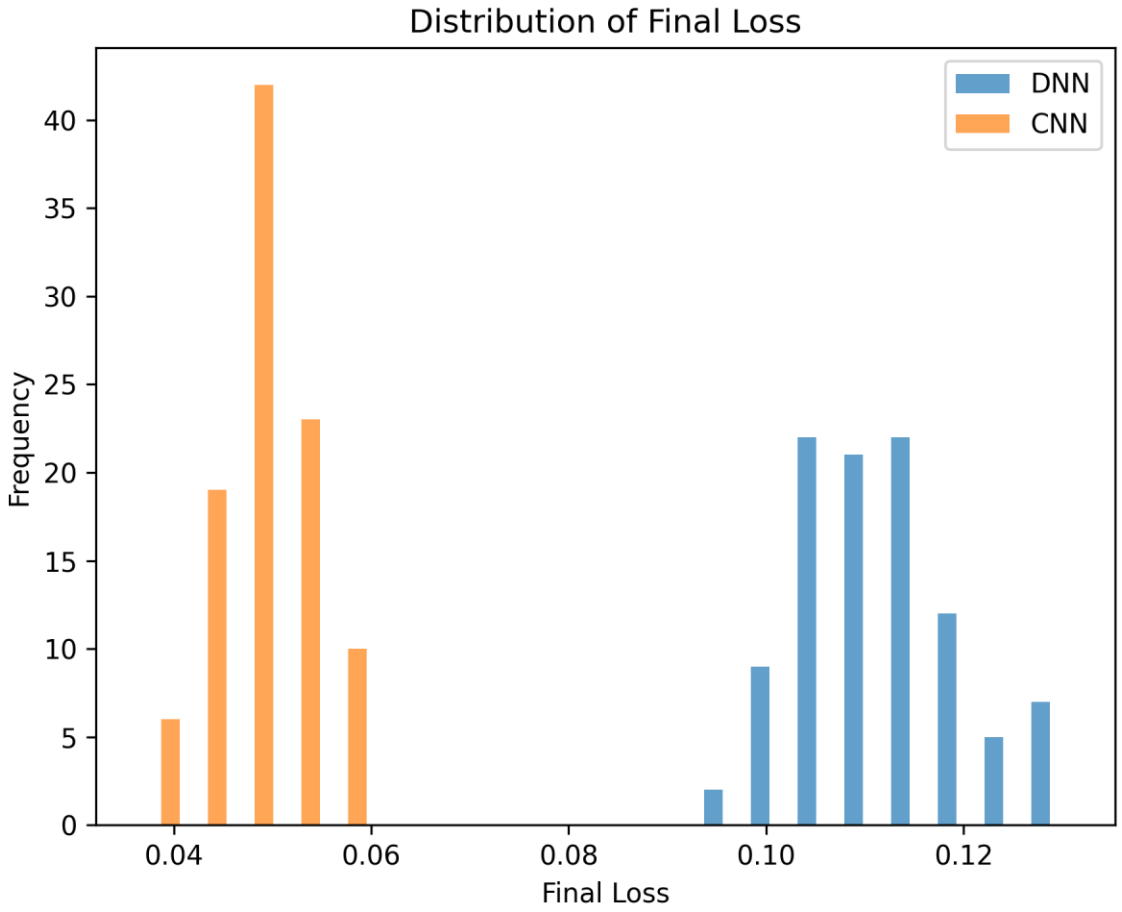




ML Evolution & Building Blocks

- Reducing the number of free parameters and being cleverer gives better model performance.
  - *(resulting accuracy/loss from 100 trained mnist classifier models)*



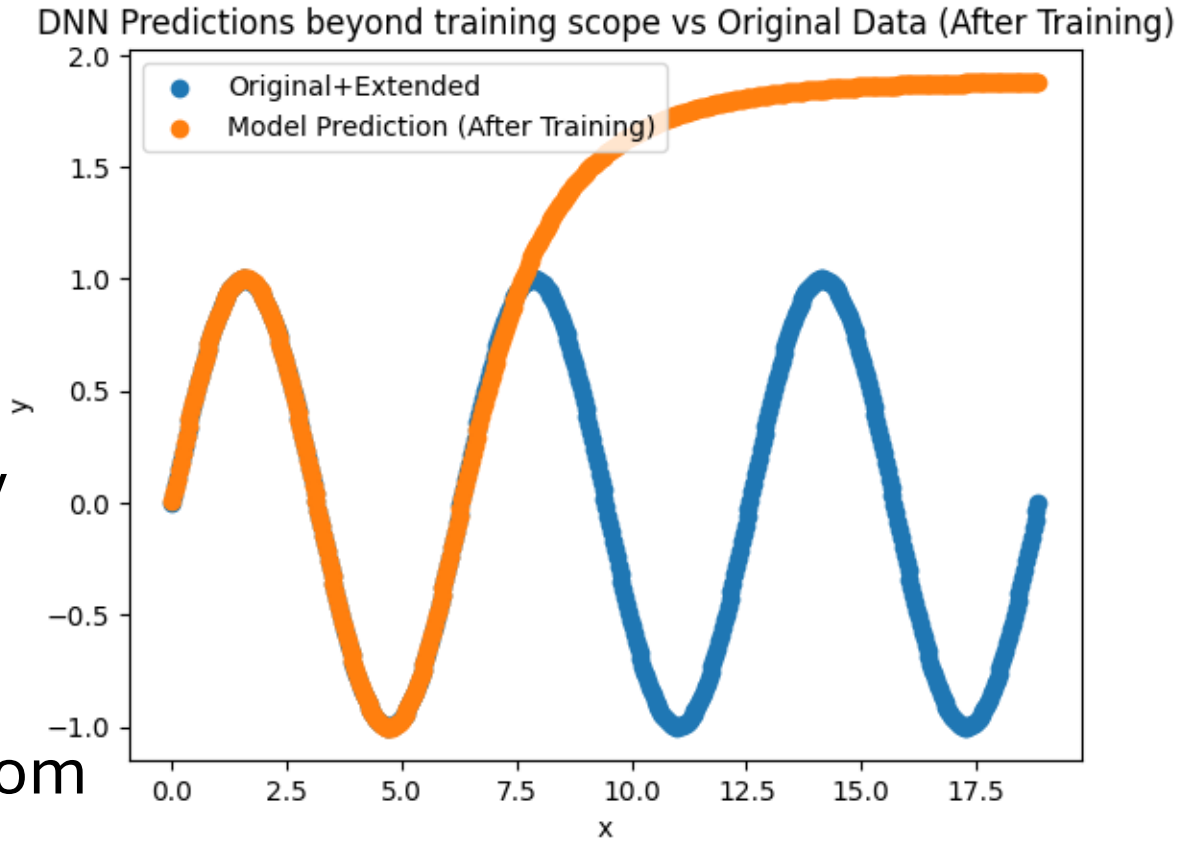


CNN/DNN Model Test – Failure

- **Vanilla DNN/CNN aren't designed to extract relationships between input events**

  These models can't tell time(!)

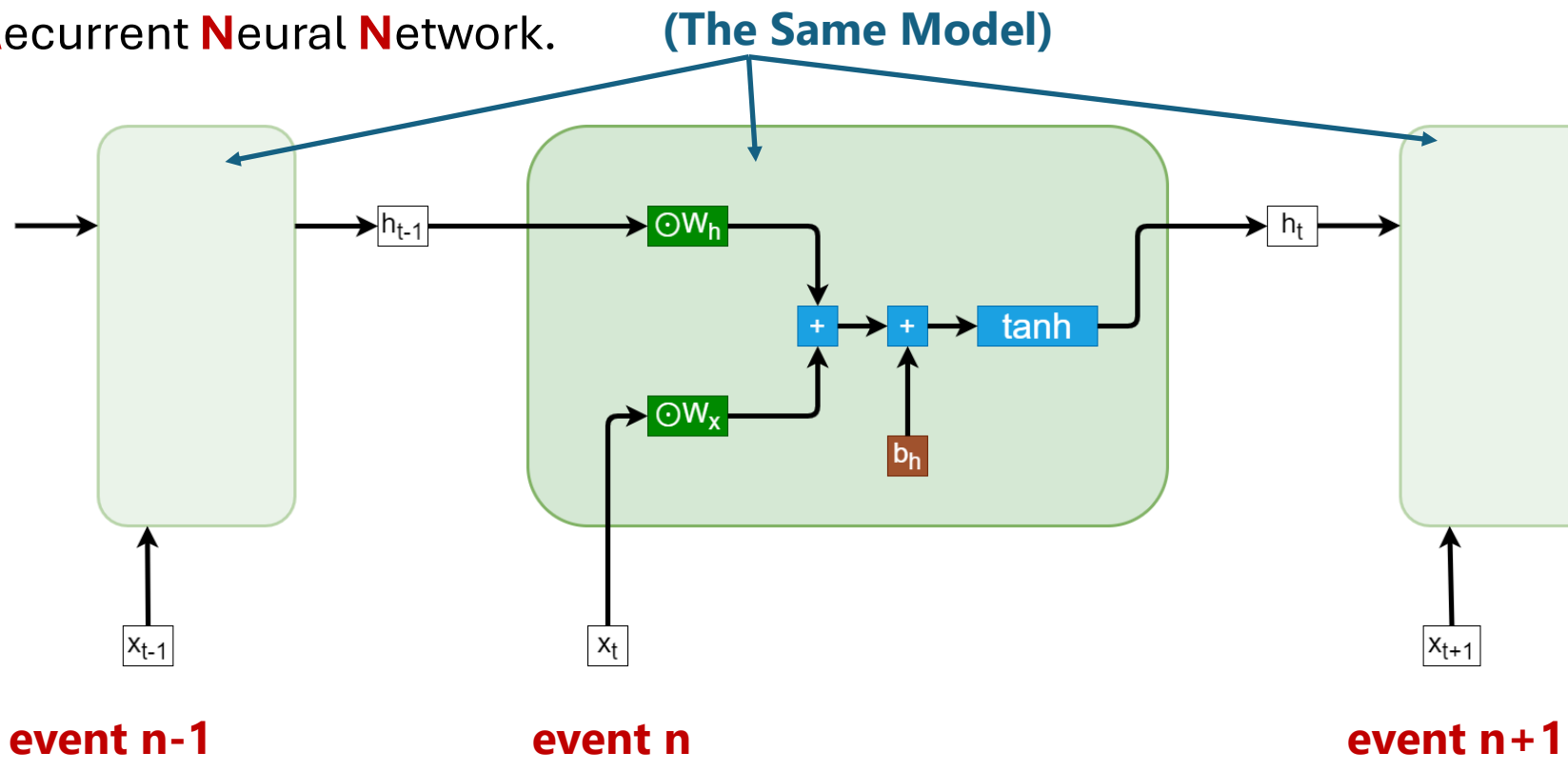
- Networks are not designed to take ordered input(s)
- Output from model will almost certainly not predict patterns or time-based behaviours
- Models are good for identifying if a random anomaly appears compared to training dataset.



ML Evolution & Building Blocks – RNN (1/4)

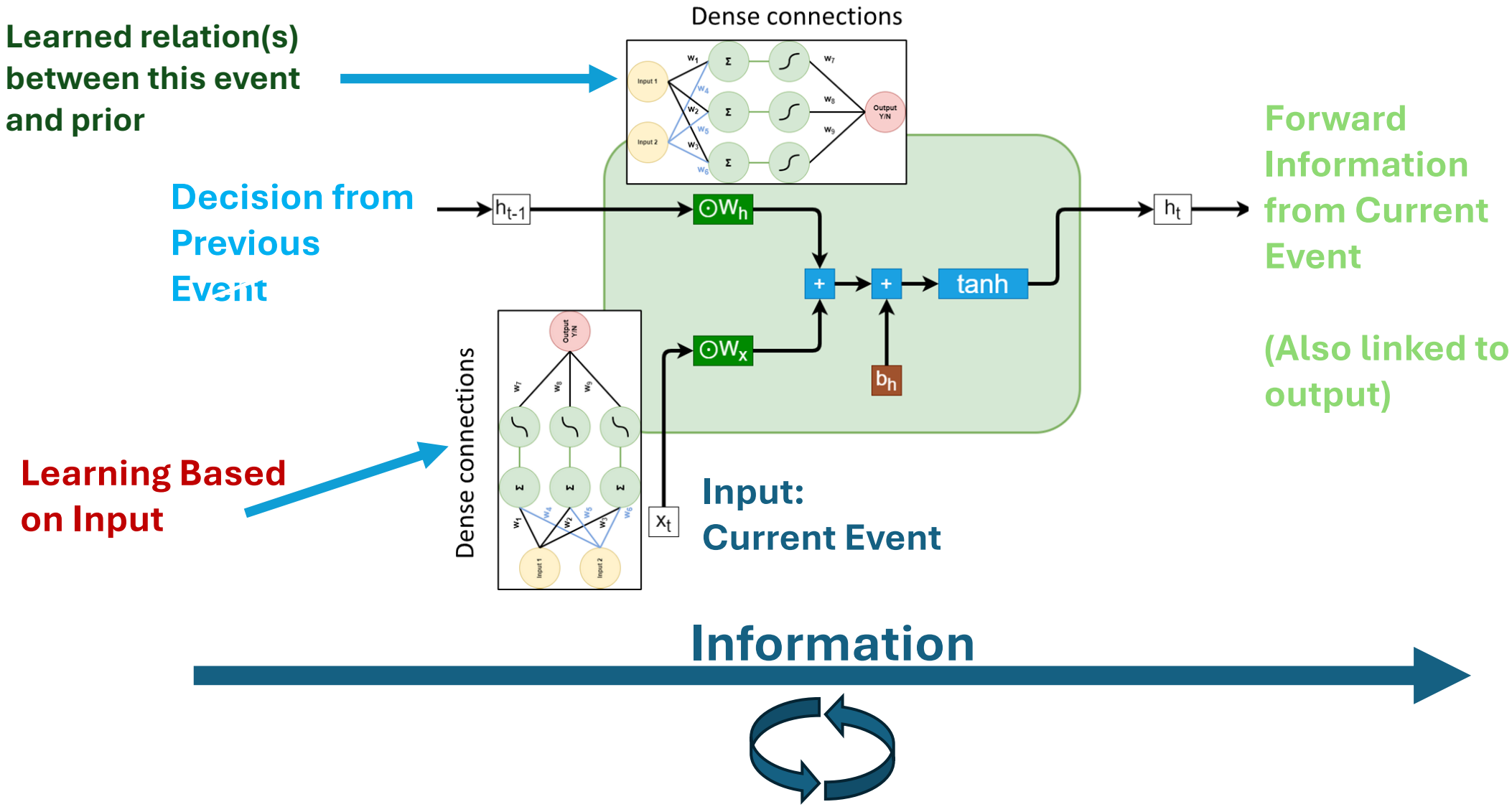
- Learning **relationships** **between** inputs requires analysing **multiple inputs** "***in sequence***".

- Simplest example of this is a **R**ecurrent **N**eural **N**etwork.
- Inputs are passed in **ordered** structure to single model.
- The model iterates over multiple inputs every iteration.



ML Evolution & Building Blocks – RNN (2/4)

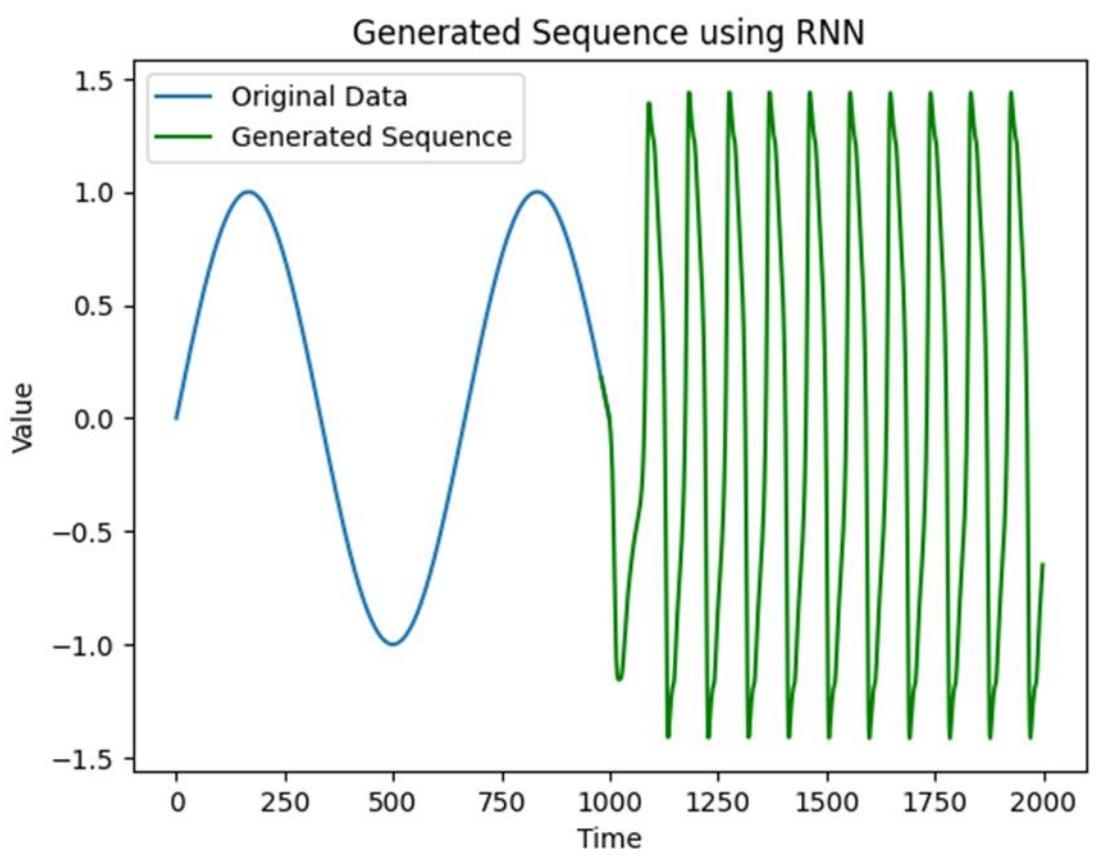
ML Evolution & Building Blocks – RNN (3/4)



ML Evolution & Building Blocks – RNN (4/4)

- However, **RNN** are notoriously difficult to train. With their generative abilities, potentially, unstable…

Seed: 42

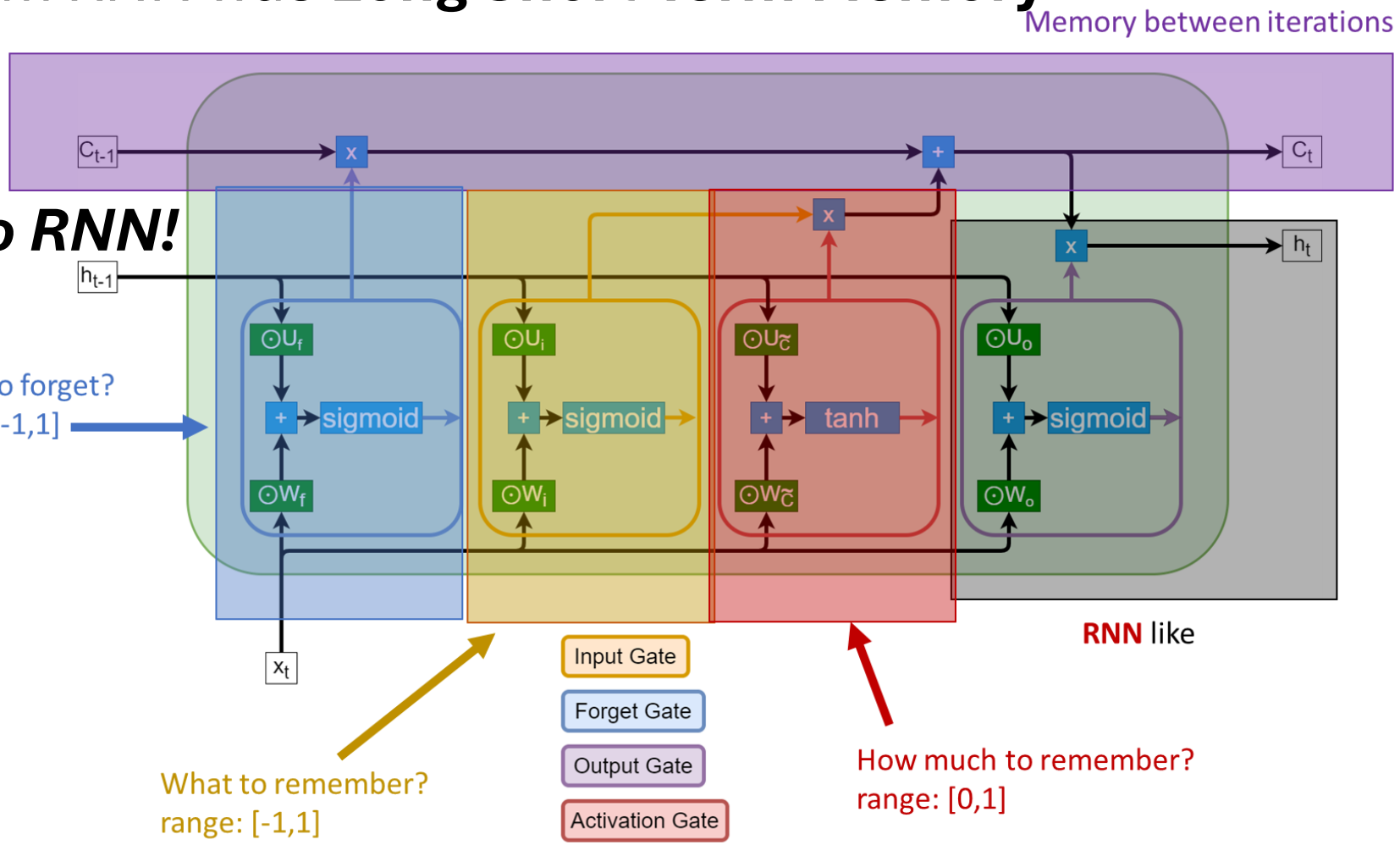


ML Evolution & Building Blocks – LSTM

- The next evolution from RNN was **Long Short-Term Memory** networks.

- ***We add "memory" to RNN!***

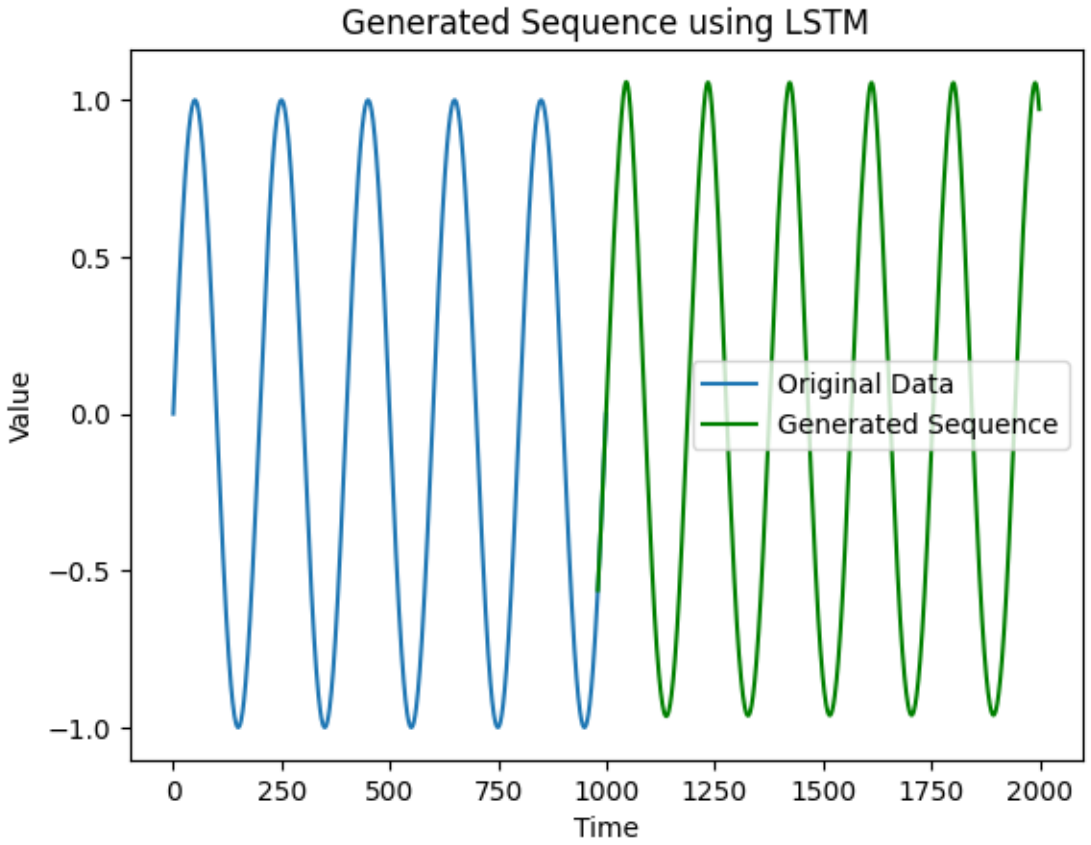
  - Better at learning relationships between inputs
  - Output is more stable, but still **very difficult to train**
  - Models can be scary and complex…



ML Evolution & Building Blocks – LSTM

- **LSTM** are more complex than **RNN**.
- Big difference is that this introduces the idea of an internal memory state to store learned relationships between inputs.
- Training is still difficult; outputs are still semi-unstable.

Seed: a

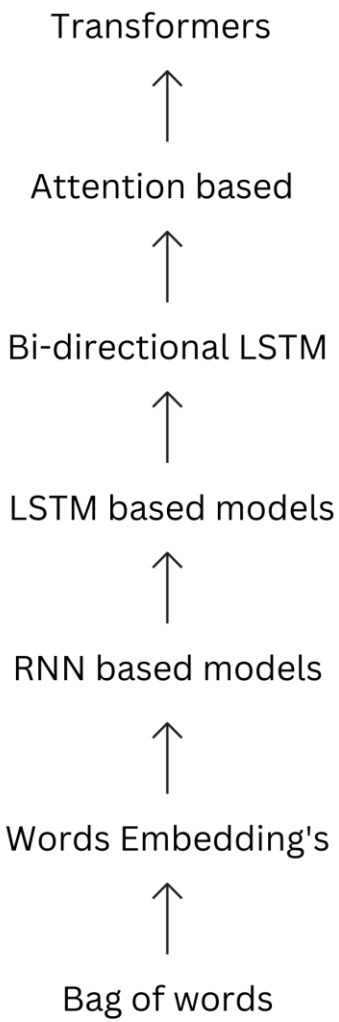


ML Evolution & State of the Art (2015)

- Cleverer models are more stable than simpler models.
- Models that get too big are often unstable.
- Keep models as big as they need to be, but don’t make them too big.
- Recursive model generation is unstable, and you need to use them carefully.
- Further training models likely yields no real benefit.
- Domain specific models will likely always out-perform generic models.

Analysing complex relationships

- Language can be broken down into a complex set of rules governing the ordering of words
- Ideally, to extract sentiment or guess what word might come next you need some model which can learn these rules or relationships and lead to a stable trained model.
- *Ideally*, we'd also like this model to be stable during generation too.



https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)

Attention in Transformers

- Before we get to "Attention in Transformers".

- What is "Attention"?

$$Attention = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right).V$$

Attention in Machine Learning

- **Query**, **Key**, **Value**. OK, that's English, but *what does it mean*?

- **Query:** What you're looking for.
  (focus of attention)

- **Key:** How well some reference matches what you're looking for.
  (relevancy score)

- **Value:** Information to retrieve from reference depending on relevancy.

Attention in Machine Learning

- **Query**, **Key**, **Value**. OK, that's English, but *what does it mean*?

  d_k : *Key Dimension*

- E.g.: **Ranking query (web search)**

- Attention can be used to produce a ranking system.
  *For a ranking model, **Key**/**Value** are fixed*.

- **Eg**:

  Query: "Keyword from user Search",
  Key: "Keywords already pre-computed",
  Value: "Indexed Web-Pages".

Web Query (User): Ferrari

Keys: 1990, blue, red, furry, mediterranean, engine, speed, leather, winter, raining

Values (Web-Page): Cars of 1990s www.cars.com, Best Boat www.bestboat.org, Cute Kittens www.kittens.net,

Transformer Attention as a Concept

- It's worth looking at **Attention** to understand what is going on within these models.

$$\mathbf{Attention} = \mathbf{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{\boldsymbol{d}_{\boldsymbol{K}}}}\right).\boldsymbol{V}$$

- At a high-level we have:

$$\mathbf{Attention} = \text{"}\mathbf{how\ alike}\ \boldsymbol{Q}\ \mathbf{and}\ \boldsymbol{K}\ \mathbf{are}\text{"} \times \text{"}\mathbf{input\ values}\text{"}$$

- This means that *(ignoring a lot of implementation details)*;

  every **Attention** layer contains residual *"**skip**-like"* connection and a *"**mask**-like"* filter to control the flow of information through the model.

Attention in Machine Learning

- OK, I can extract **Key** and **Value** from data.

- With a bit of imagination, I can make an algorithm to map "***user queries***" to "**Query**" and that gives me a search system which can find results in ranked web-pages.

- This is in a *"hand-wavy"* way how ***Google***, search engines and complex fuzzy database queries work.

Attention in Machine Learning

- So how do we extract this information from our data?

1. Build a billion-dollar company.

   Hire the worlds best computing experts.
   Spend millions each year.
   Keep updating refining many algorithms.

   Or,

2. Get the Machine Learning model to do it for us.

   This is called "**Self-Attention**"

Aka, map the **input data** to **Q**, **K** & **V** and have the model learn the relationships itself.

Attention as a Graph

- Input data is *"projected"* into the model.
- Like embedding, but **preserves positional ordering of input**.

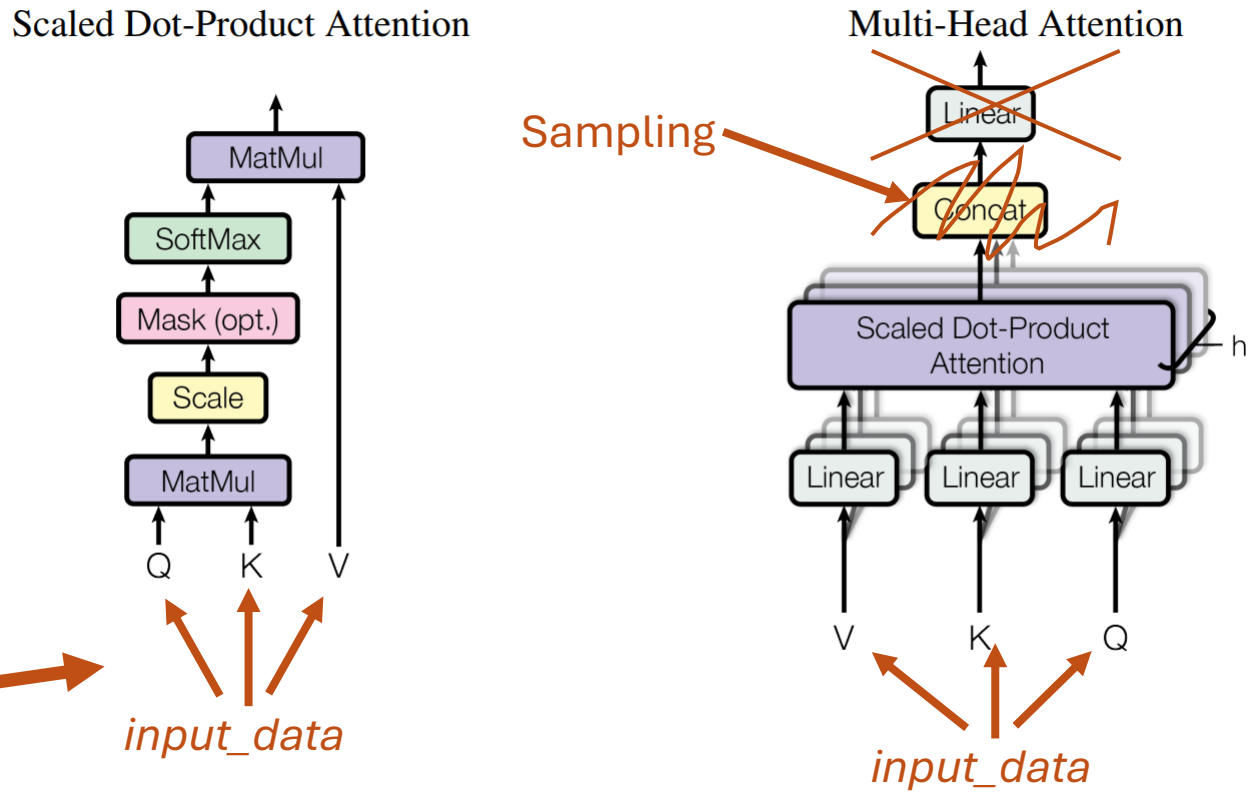


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Projection, or, "Embeddings"

https://arxiv.org/abs/1706.03762

Attention as a Graph

- When projecting to **Q**, **K** and **V** in a model with **Attention** these **3** matrices are looking for **3** different values

- For LLMs:

  Query extracts the current query being made
  Key extracts relationships between concepts
  Value contains a reference of concepts/values

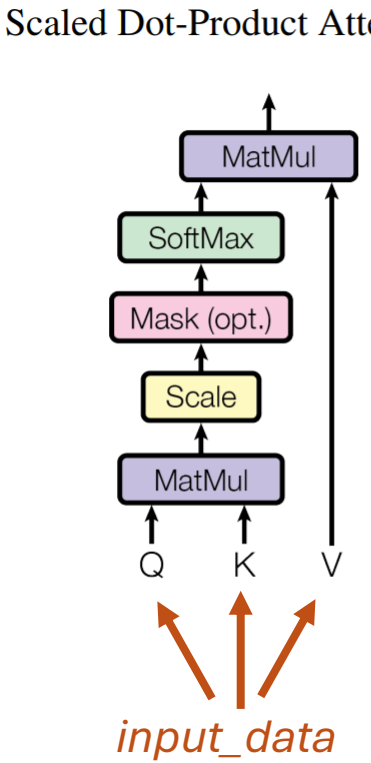
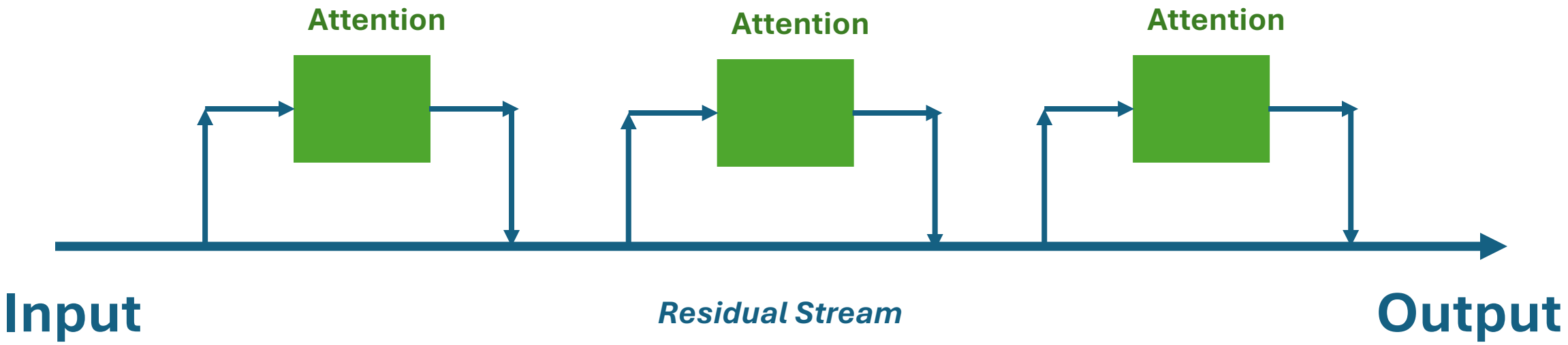


Figure 2: (left) Scaled Dot-Product Attention. attention layers running in parallel.

https://arxiv.org/abs/1706.03762

Attention based model design

- The simplest **Attention** based model is a model which acts on tokens and extracts out ("attends to" key features) and passes this back to the residual stream.



https://arxiv.org/abs/1706.03762

Transformers

- Now have all the pieces to build the transformer model. Famously introduced by Google in **2017**

- This model uses **3** very new and key advances:

1. **Self-Attention**
   Extracting **Q**, **K** & **V** directly from input

2. **Feed-Forward masks**
   Only the past is considered when generating

3. **Multi-Headed Attention**
   Multiple attention calculations in parallel

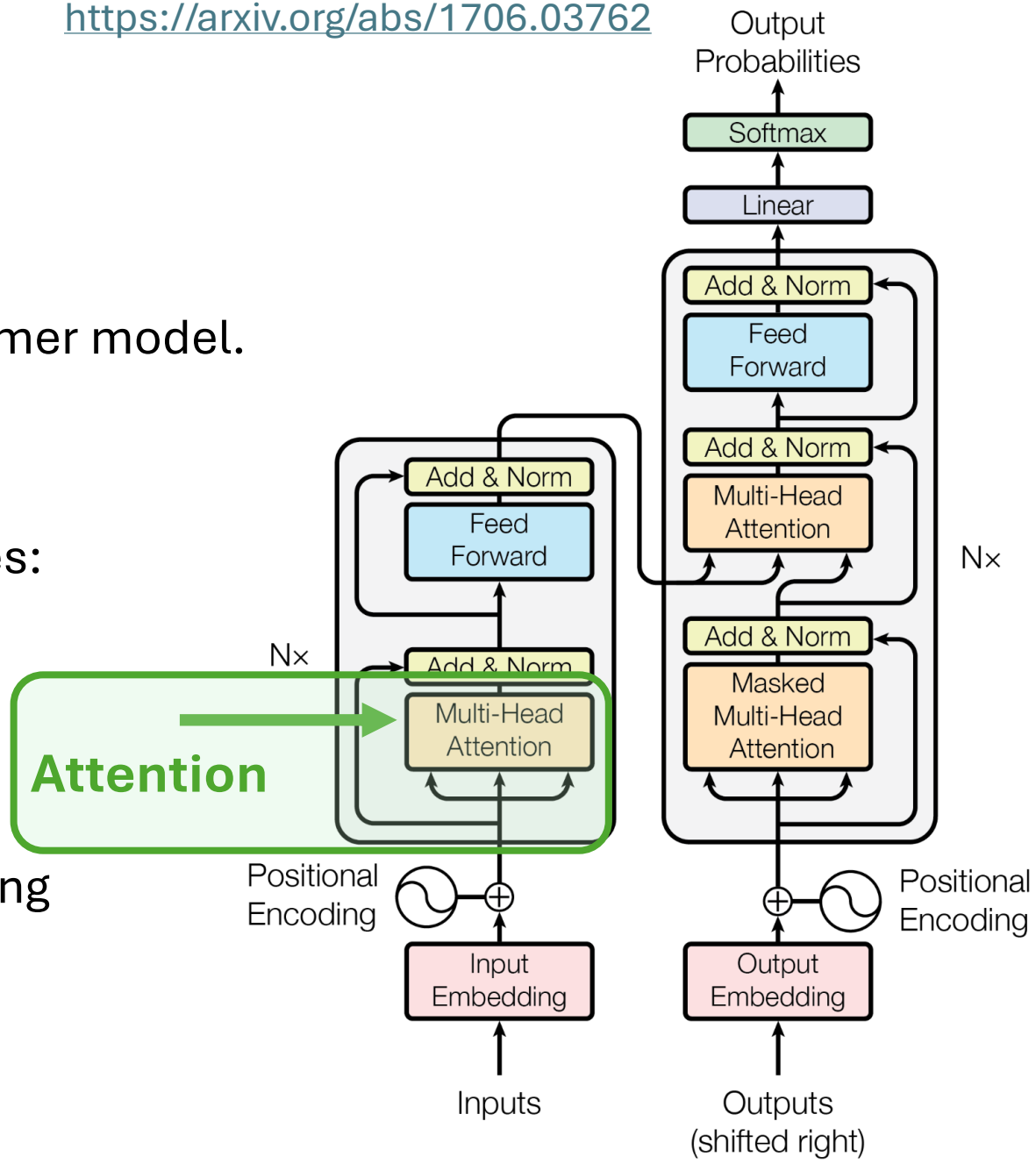


Figure 1: The Transformer - model architecture.

Transformers

- **Self-Attention** means we can extract relationships from our data.
- **Feed-Forward masks** help improve the model stability.
- **Multi-Headed Attention** means we can extract short- and long-range relationships in parallel easier.
- The graph of **MH(FF)A** shows this approach allows more information to flow through the model.

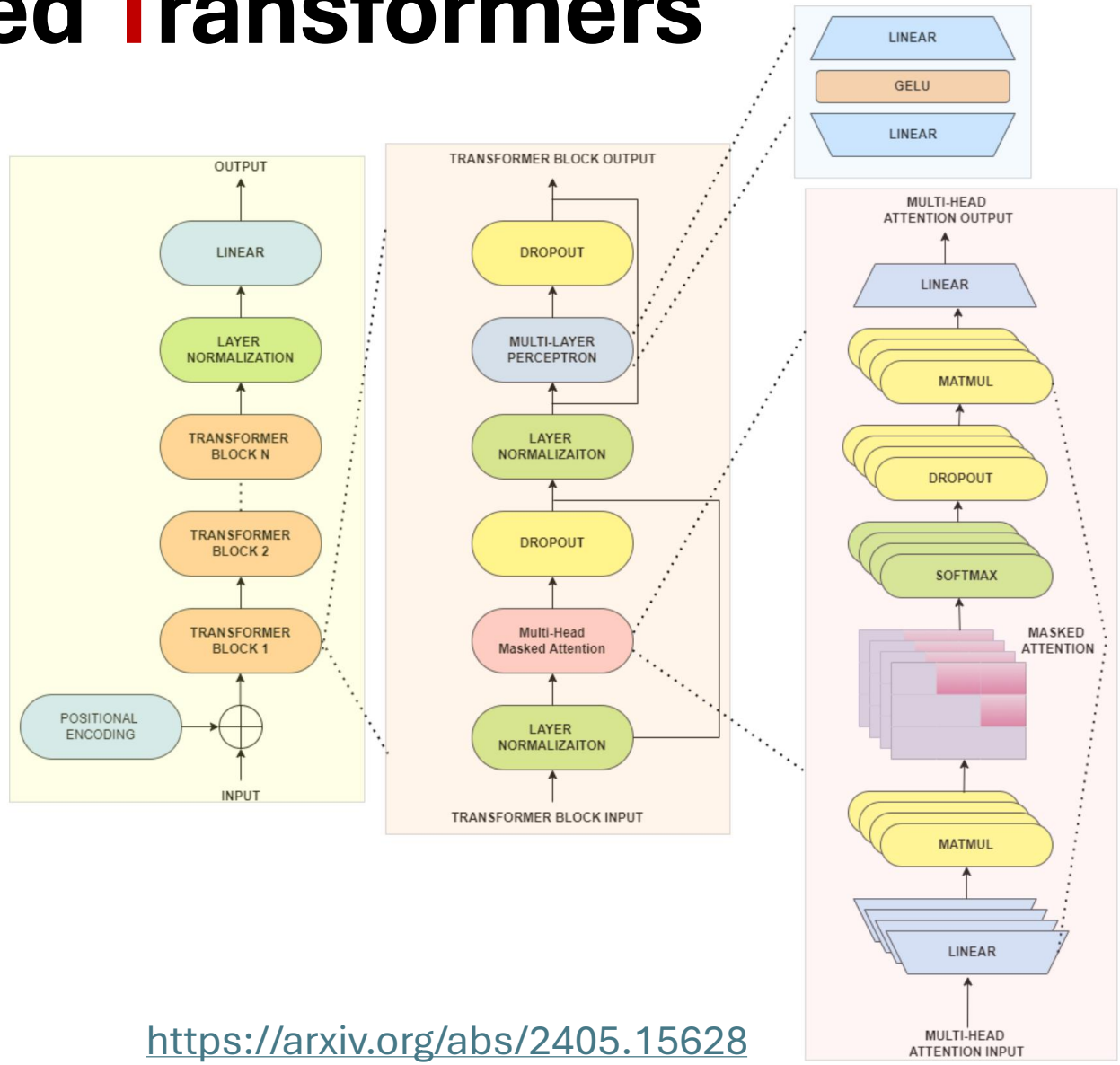
Transformers vs **Attention**

- **Transformers** models are an application of **Attention**.
- A single **Transformer** "**block**" is composed of multiple ***Attention*** "**heads**" which act in parallel on the same input data.
- Parallel **attention heads** increases the model's ability to attend to different concepts.
- Parallel **blocks** push this even further allowing a model to act upon the residual stream of data that flows through the model in different ways.

Generative Pre-Trained Transformers (GPT)

ChatGPT was built using the transformer components first introduced by Google.

ChatGPT2/3 were mainly stacked **Feed-Forward** **Self-Attention** based **Transformer** blocks.

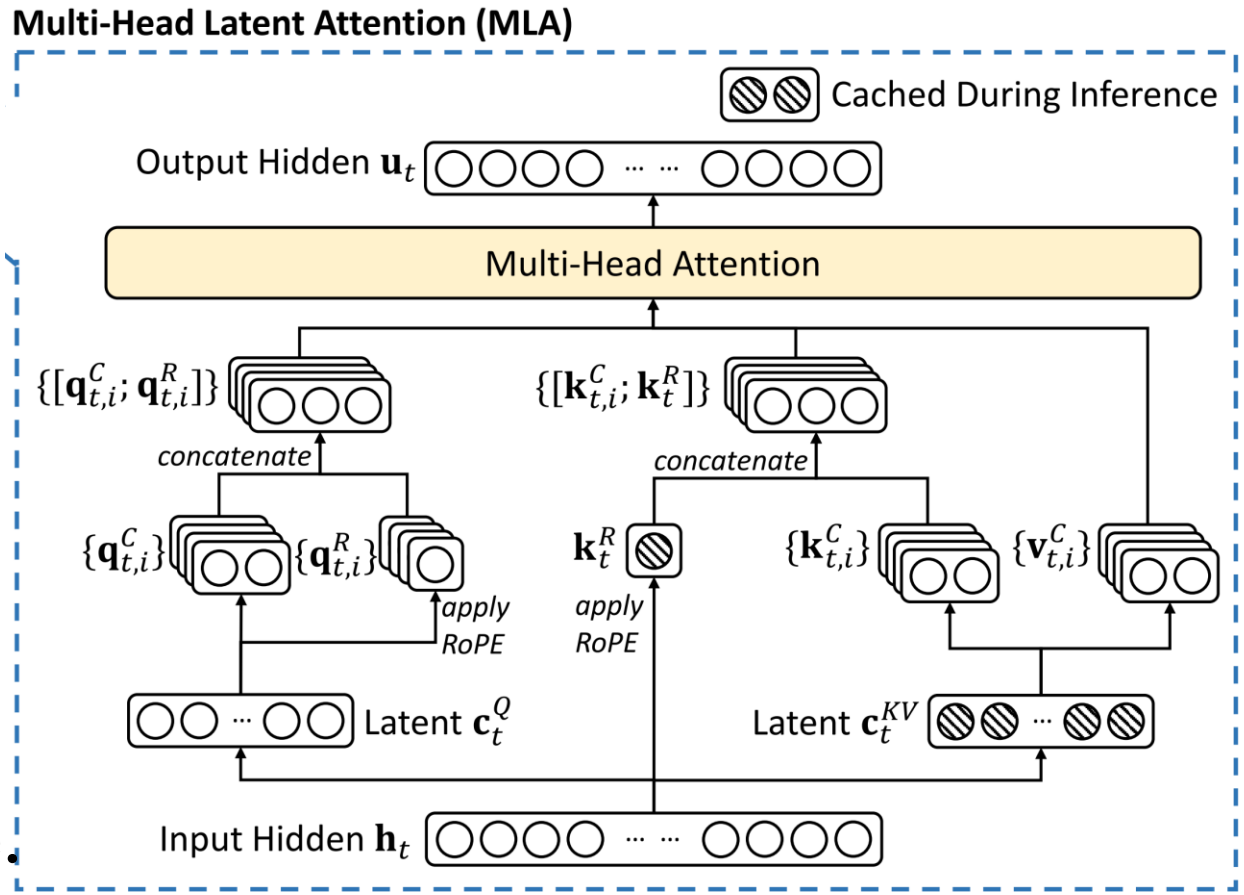


https://arxiv.org/abs/2405.15628

Fig. 2 GPT-2 Architecture

DeepSeek – new best in-class

- **MHA** models perform a huge amount of complex matrix multiplications in pipelines.
- This causes an extremely large amount of data to be shuffled.
- **Multi-Headed *Latent* Feed-Forward Attention** is an attempt to reduce this whilst preserving model performance.



https://arxiv.org/pdf/2405.04434

DeepSeek – new best in-class

- To train **LLM** models, data ordering is a strong constraint.

- **Query** and **Key** matrices in **Attention** calculations needed to have a positional encoding applied to them so that they can “learn” relative context based upon the input.
- **Value** in **Attention** however just needs to learn key features to extract and doesn’t need to positional information to do this.

- The simplest way to give **Q** and **K** what they need is to hard-code positional embeddings into the input data.

DeepSeek – new best in-class

- Hard-coded positional embeddings work.
- That's exactly what google did in their "***Attention is all you need***" paper.

- However, this has a strong limit that as soon as the model moves beyond its context window it can be come unstable and "*hallucinate*" more…

  Also, it's easy for this positional data to "*bleed*" into the **Values** being extracted…

- By adopting RoPE the model learns relative context without relying on hard-coded positional data.

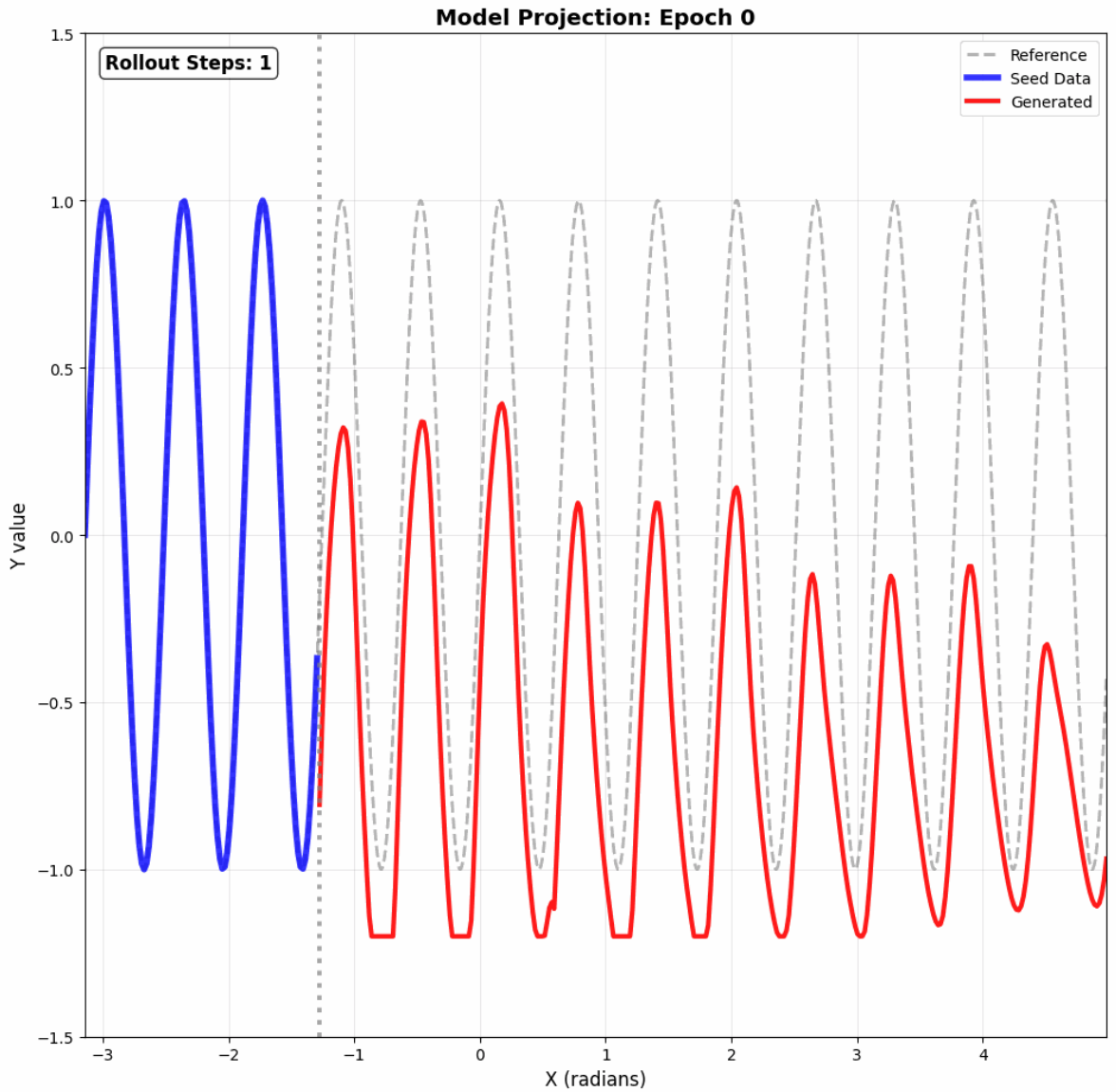
- This requires some extra computation; but ultimately tends to help the model better separate between context and values.

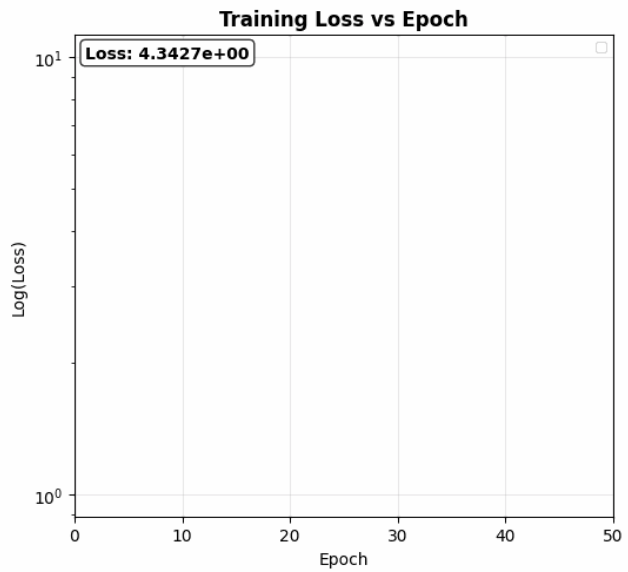
GPT Model training

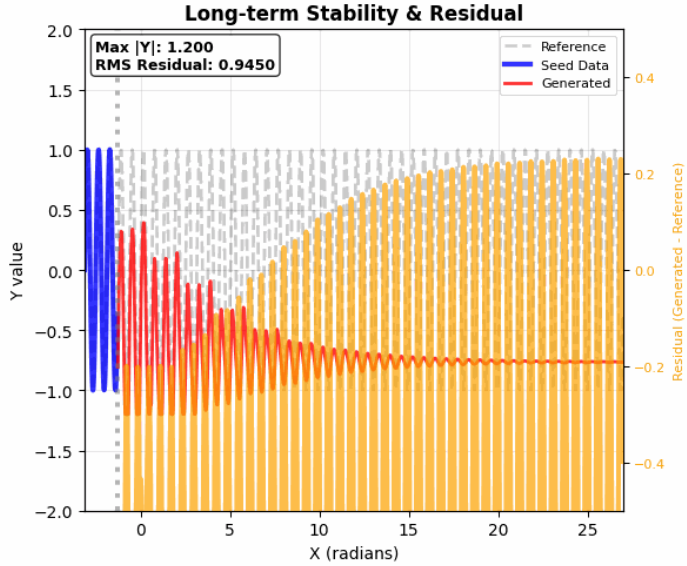
Noisy at first, **Transformer** models can be used to trained to generate new waveform data based on a dataset.

(This involved implementing:

statistical data sampling & augmentation, clipping, normalization, auto-regressive curriculum based training, consistency checking, positional encodings, correct initialization, …)

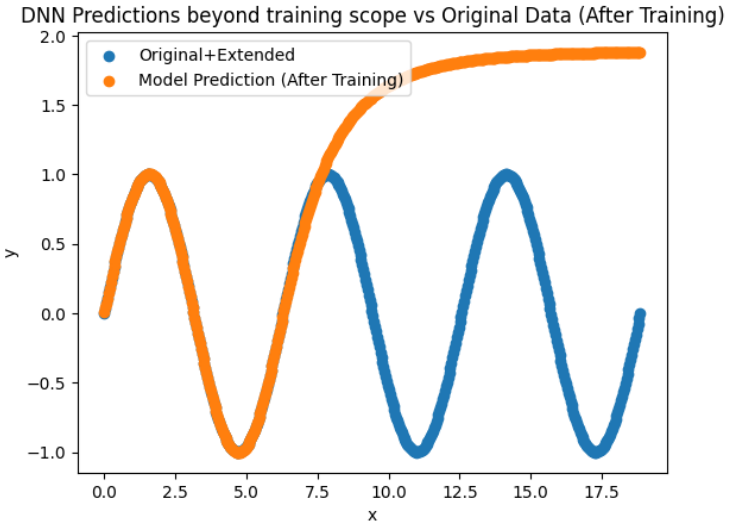




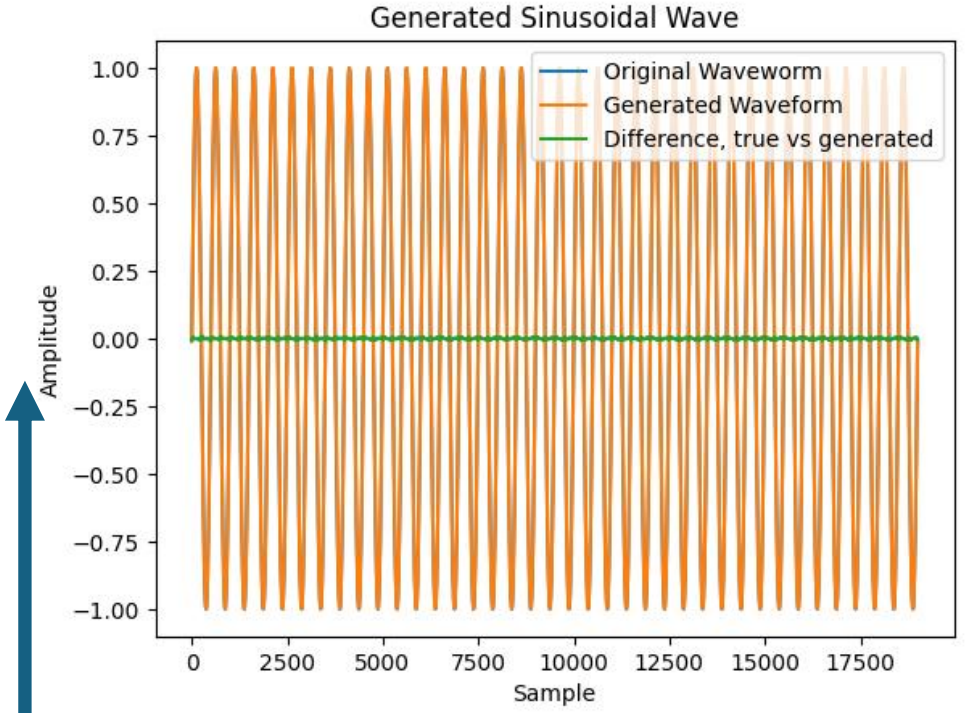


Model Performance

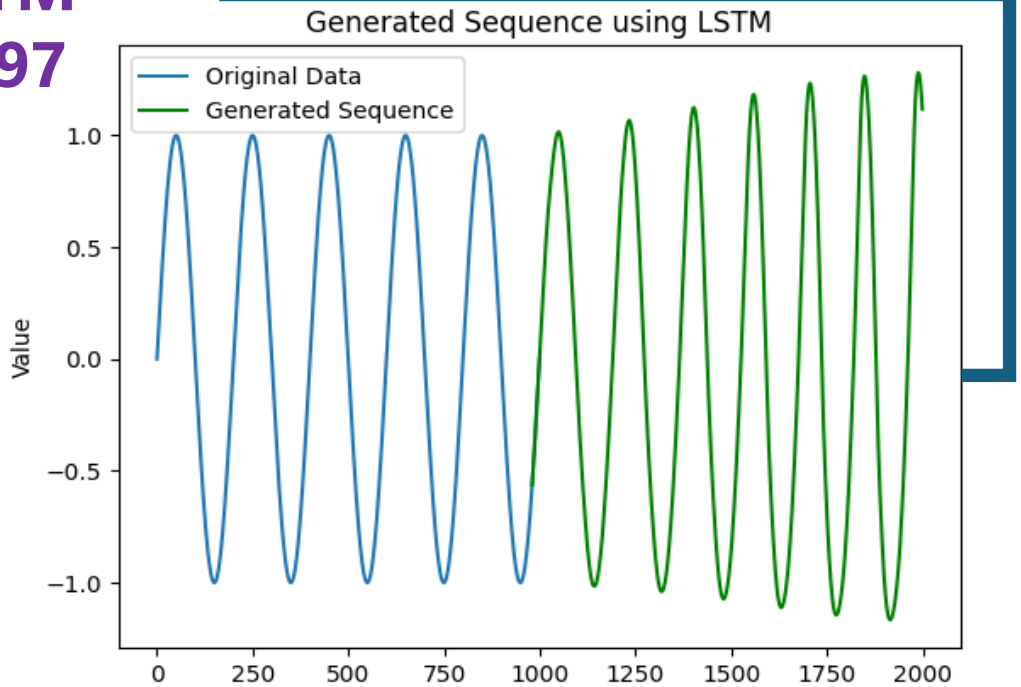
DNN/CNN
1958



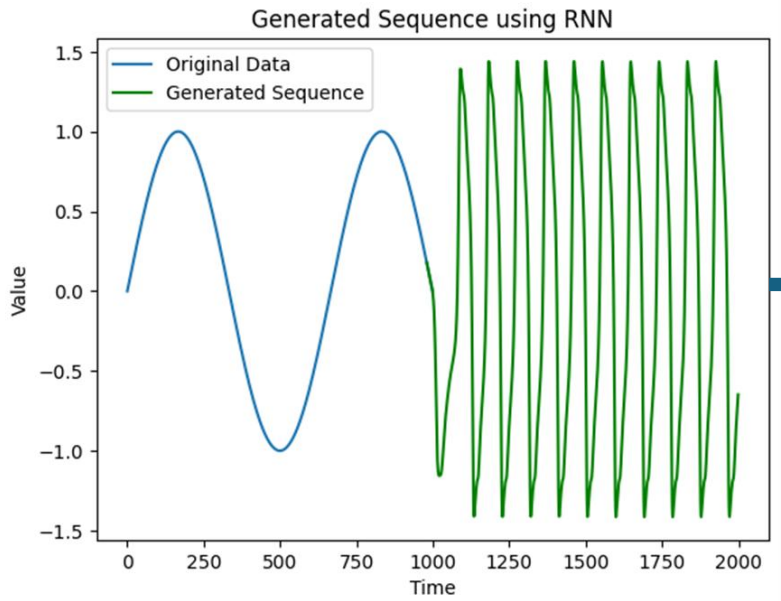
GPT
2018



LSTM
1997



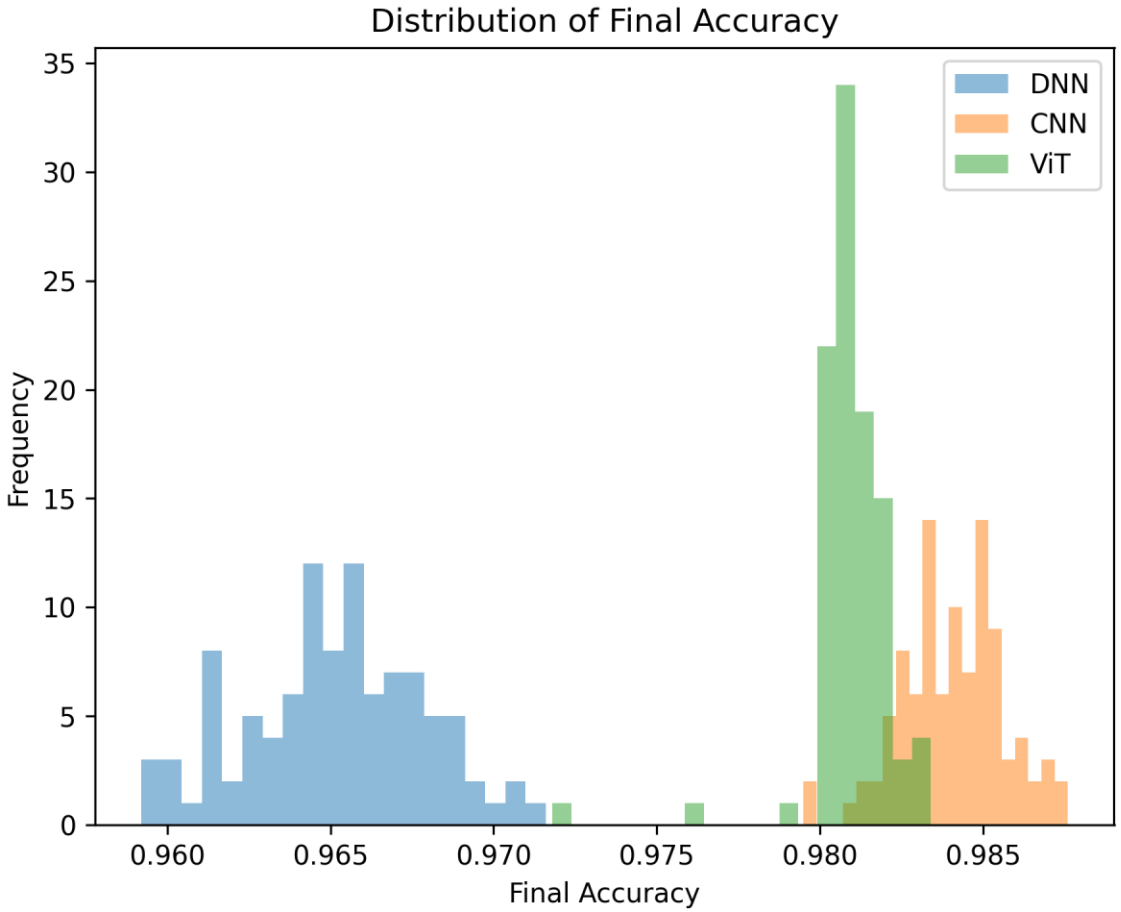
RNN
1986

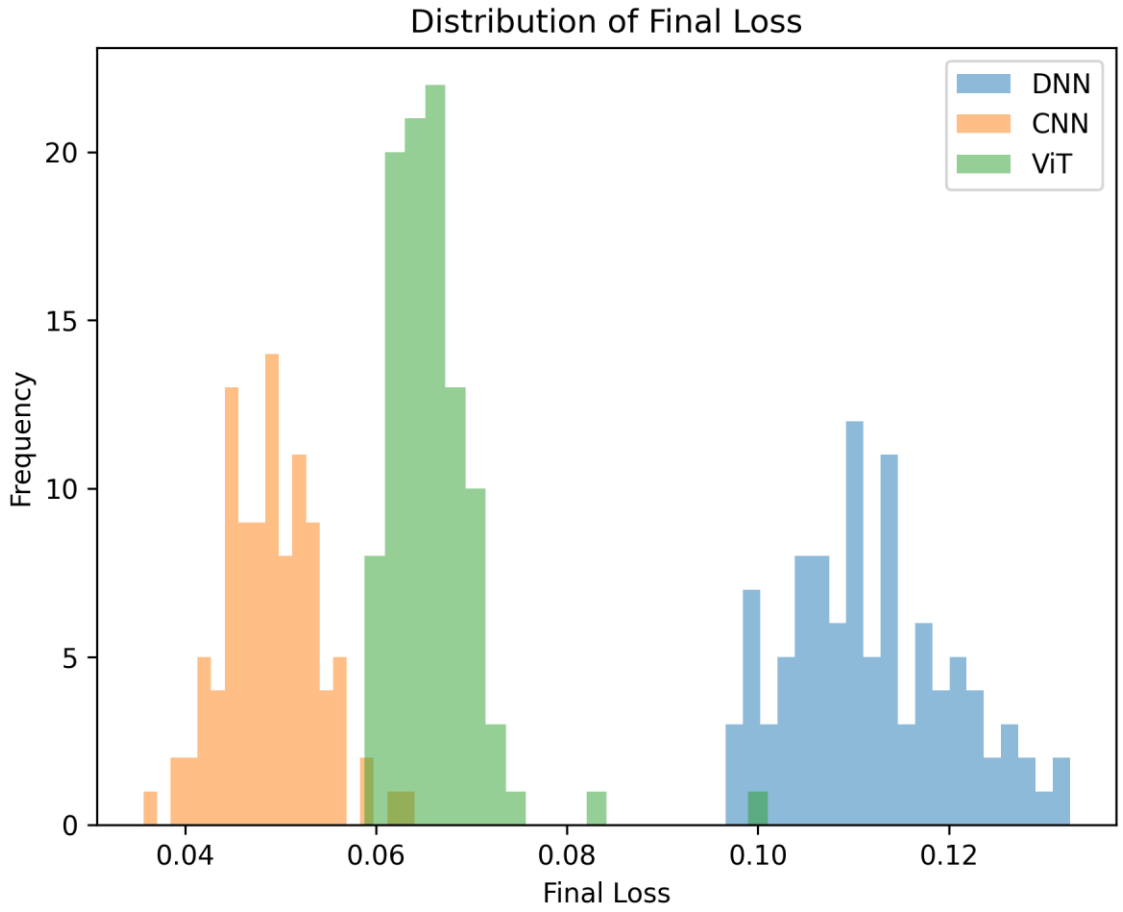


LLM stability (aside)

- Comparing **CNN** & **DNN** to **Transformer** based models.

  Training 100 classifiers for the mnist numerical data. **Transformer** models are **large** and very **stable**(!)





Transformer Models

- **Transformer** based models are models capable of extracting context from data. Hence, we say they are “capable of learning meaning”.

- Pretty much all **Transformer** based models are now based on **Attention** in some form given the performance of this type of model design.

- One of the often-touted advantages of ML models is ***“learn once apply everywhere”***.

  This means that once a single model has been designed and trained it can be used repeatedly at much lower cost.

- **Transformers** are also often touted as a class of model where you can:

  “Throw hardware at a problem and reduce the expert person-power needed to solve verify the result.”

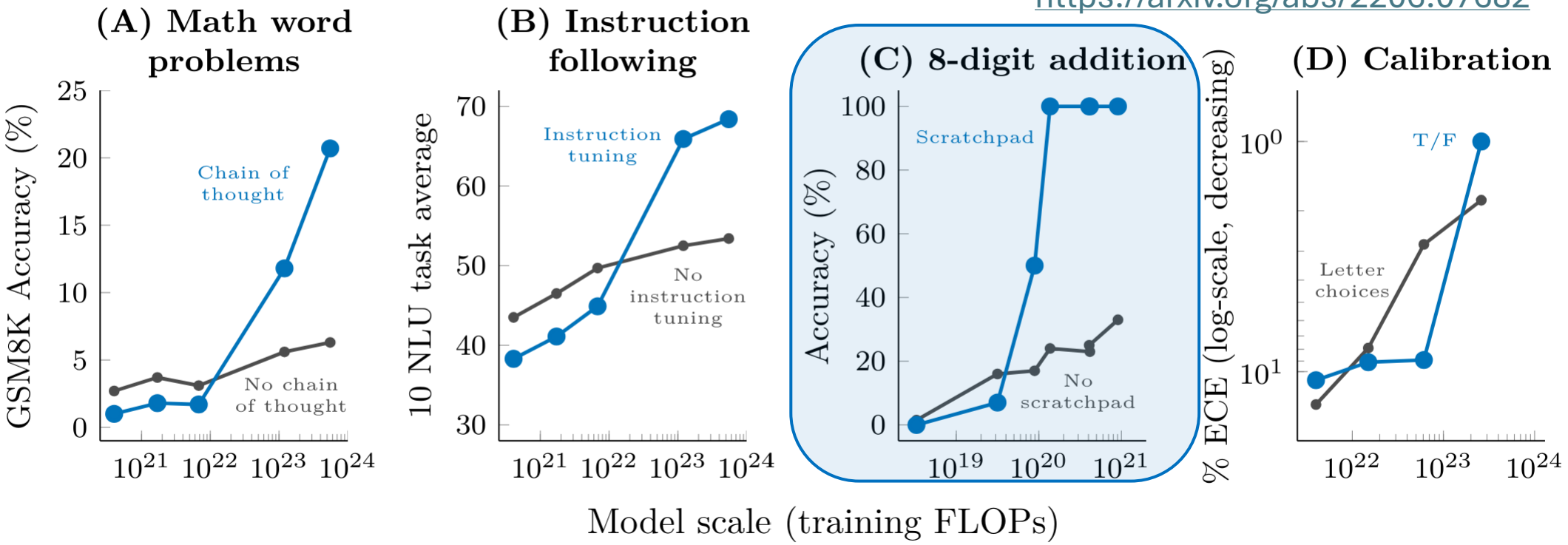
Emergent Properties (1/3)

- Emergent properties observed at around ~300ZFLOPs of processing in modern models. (2024)

  Most powerful '*single*' HPC system in the world is capable of 0.001ZFLOP/s …

- → Minimum training times of <u>days</u> on extremely expensive & power hungry (**1MW**!) kit.

https://arxiv.org/abs/2206.07682

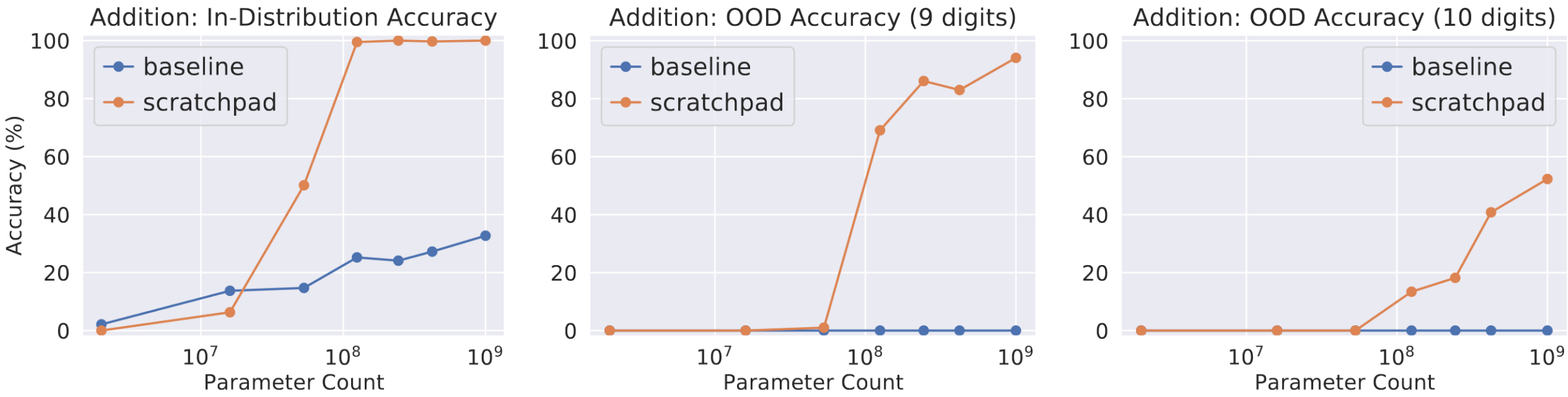


Emergent Properties (2/3)

- A model capable of 8-digit addition can perform this level of calculation when the model:

  a) has been sufficiently trained
  b) has sufficient complexity

https://openreview.net/pdf?id=iedYJm92o0a



Emergent Properties (3/3)

- Another example of an emergent property is that NLL models trained on English language can explain how/why a joke is funny.

- Or a model trained on coding websites can annotate, understand code, context and even identify basic coding problems and offer solutions.

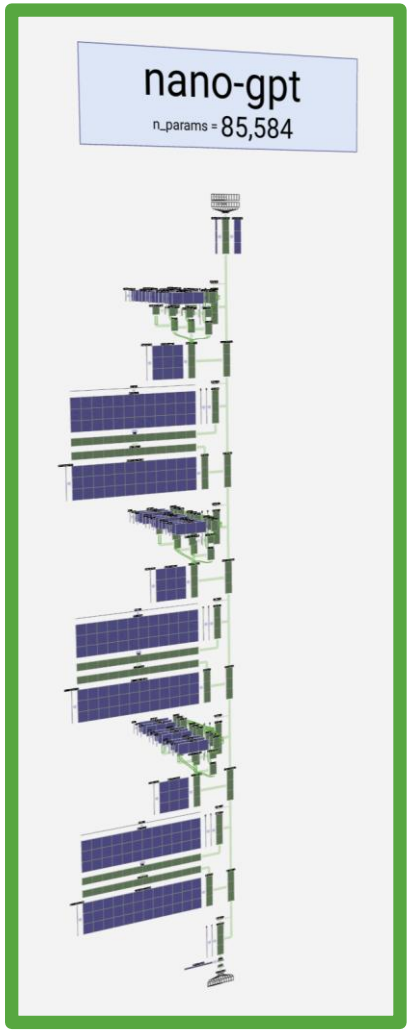
- **It's not fully always understood why/how these emergent properties emerge.**
  Or even why these properties suddenly emerge.

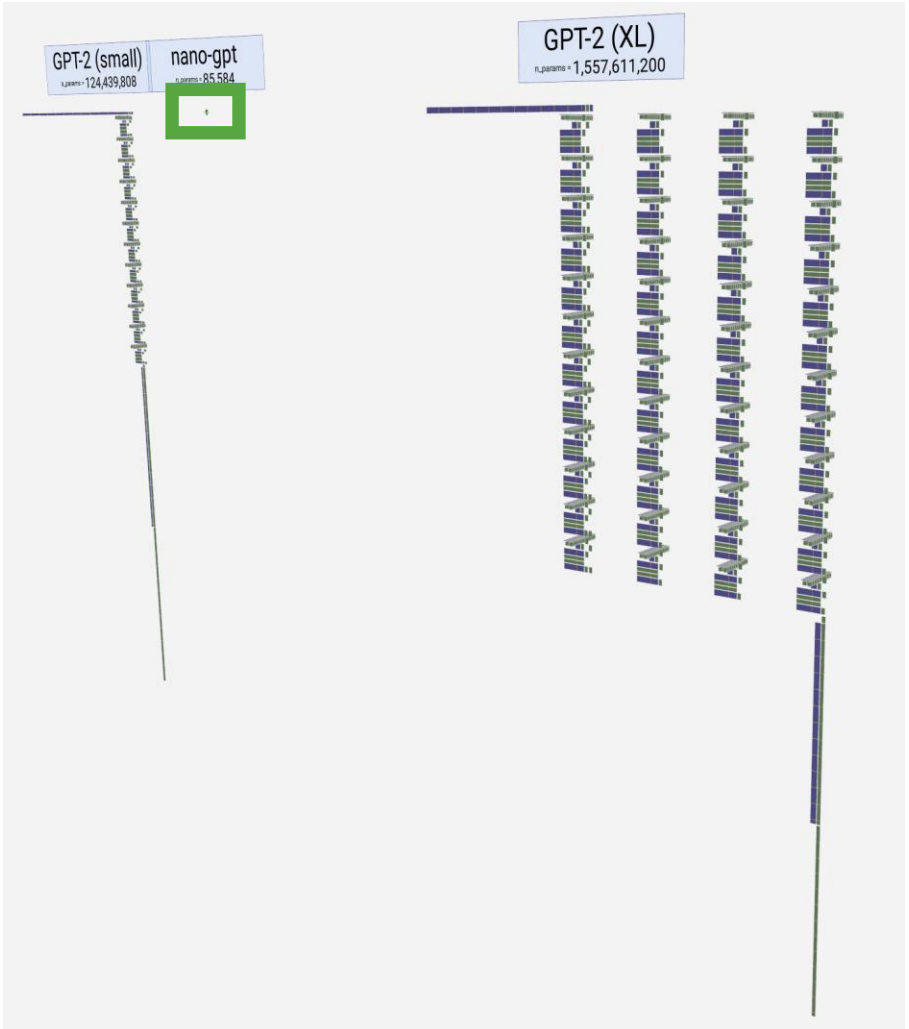
  Properties from trained models are well demonstrated, but it's not obvious during training when/how these properties have emerged.

  There are philosophical, mathematical & computational suggestions for how/why these properties arise…

GPT LLM Model Size Visualization

- So how complex are these models? Using https://bbycroft.net/llm





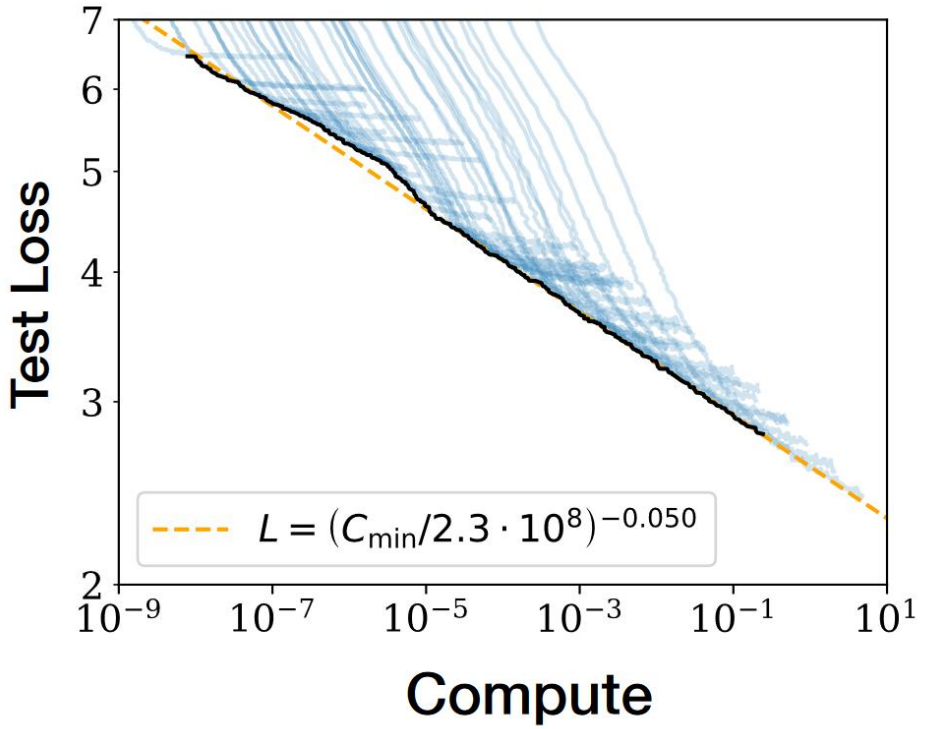
GPT LLM Model Size Visualization

- So how complex are these models? Using https://bbycroft.net/llm

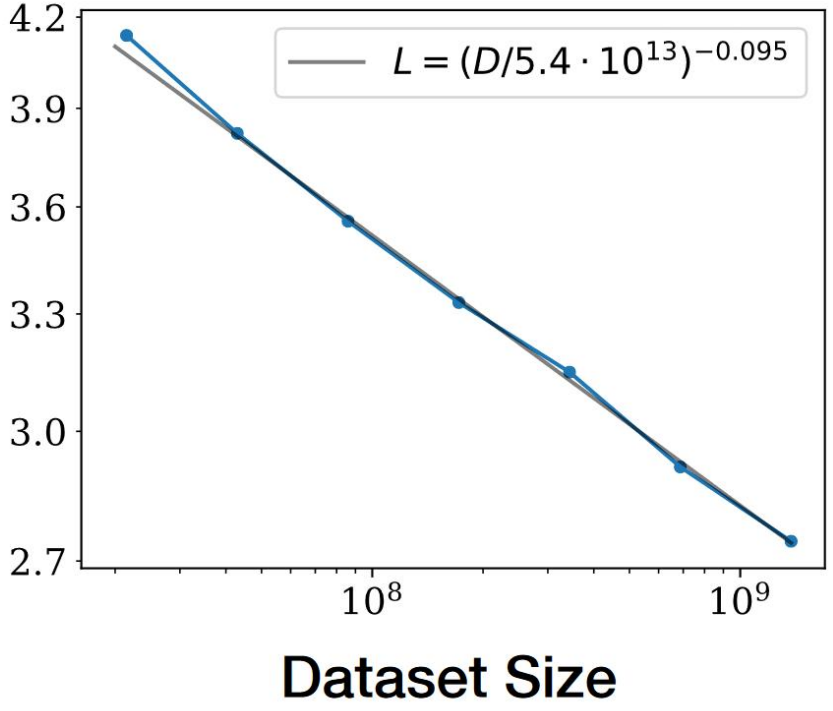
Will LLM just keep getting better?

- LLM model scaling is already being strongly investigated by **OpenAI** and others.

Error



Model Size



Dataset Size

https://arxiv.org/abs/2001.08361

Will LLM just keep getting better? (2)

- Trends observed from ***ChatGPT3*** training predicted ***ChatGPT4*** performance.
- *Not from empirical laws but observed trends.*

- LLM model scaling potentially follows the relationship: $\alpha \propto \frac{4}{d}$, https://arxiv.org/abs/2004.10802
  (here d is the dimensionality of the training data (text)…)

- OpenAI have observed their GPT models to follow a scaling of: $\alpha \propto 0.095$
  (vs English text dataset size)

- This ***could*** imply that the dimensionality of English text is*: $d \approx 42$

**I claim this potentially shows there's too many academics at OpenAI …* 😉

Will LLM just keep getting better? (3)

- Models will improve with a combination of more parameters, more data, or, longer training times.

- However, the scaling laws that these follow are not in our favour:

- Performance $\propto$ (Processing Time)$^{-0.050}$ ← **More Power**

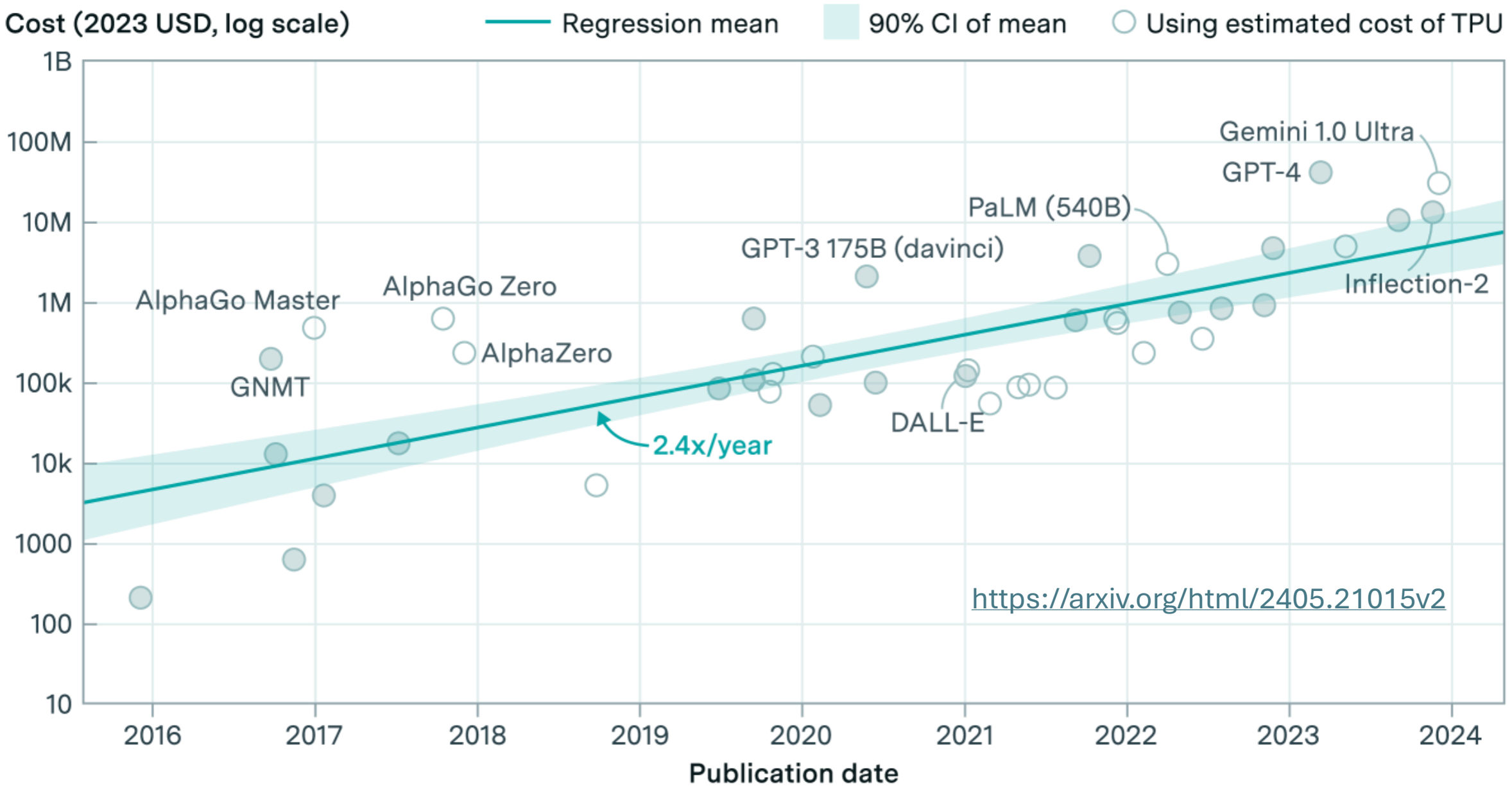
  $\propto$ (Dataset Size)$^{-0.095}$ ← **More Data**

  $\propto$ (Model Parameters)$^{-0.076}$ ← **More Hardware**

- *Reducing errors by 50%* for a (perfect) LLM requires either:

  ~10^6x CPU, ~1,000x Data, or ~1,000x Parameters (!!!)

Amortized hardware and energy cost to train frontier AI models over time



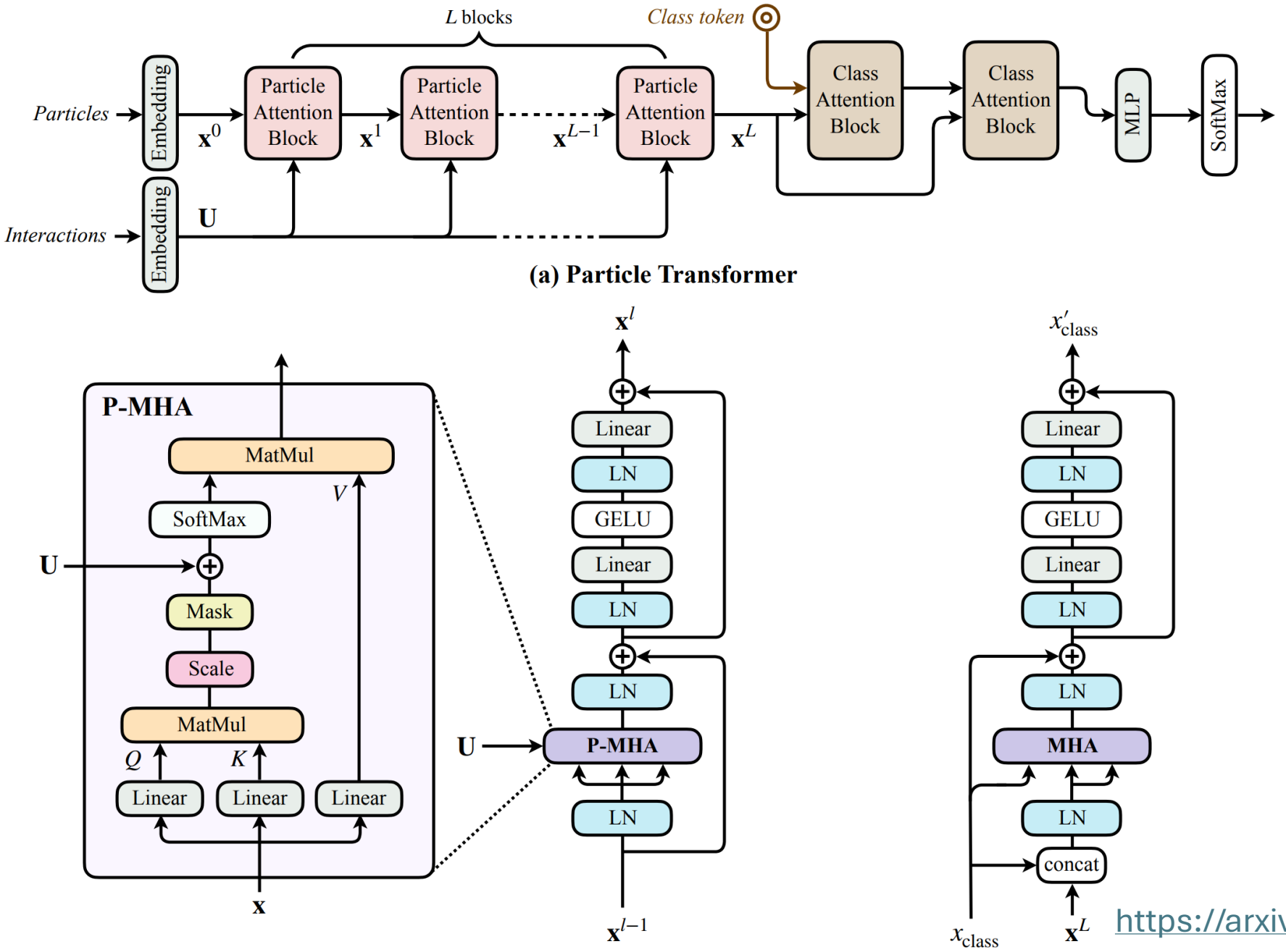
ParticleTransformer

- Transformer based models have demonstrated a huge amount of potential over normal DNN/CNN based models.

- One example model within Particle Physics is the ParticleTransformer model. https://arxiv.org/abs/2202.03772

- This transformer can take un-structured information of observed particle interactions and can be trained to

Pa



https://arxiv.org/abs/2202.03772

Figure 3. The architecture of (a) Particle Transformer (b) Particle Attention Block (c) Class Attention Block.

ParticleTransformer

- Particle Transformer as an example model has 2 “interesting” features not found in all Transformer based models:

1. Additional **U** connection between all Attention layers.

   This adds some global relationship between all the particles in the decay.

2. A **Class Token** injected toward the end of the model.

   Research from Facebook on multi-layer attention models has shown there is the performance of Transformers in models such as classifiers is better when the class tokens are inserted later into the model.

Today's Workshop

- Today we will be building an **Attention** based classifier for the CIFAR10 dataset.
  Low-resolution 32x32 pixel colour images in 10 categories.

- This model will only use 1 **Attention** stream with residuals.

- To use **Attention** in this way we need to go from image data (32x32x3) to a stream of tokens (Ax1).

Today’s Workshop

- To get tokens which we can Attend to we first pass the image data through a small CNN.

- After passing the data though a CNN the model will have trained to pick out key features.

- These key features can be then be re-shaped into a linear list of “tokens” that our Attention class can act on.

Today’s Workshop

- One of the key takeaways from today's workshop will be writing some of the key parts of your own Attention model from scratch.
- This model technically implements “***Feed-Forward***” Attention.
- This is where we want no values from the future modify what has been seen “so-far”.

  Softmax(-inf) == 0 → if we decrease output values from $Q.K^T$ to –inf we will get 0

- This is not ***needed*** as there isn’t any concept of “*future*” or “*past*” in this training data.
- However, forward masking helps reduce the amount of further computation required and has been shown to help with numerical stability.

Today’s Workshop

- Now for a quick example of running a jupyter notebook on Google colab.