



Building an LLM from Scratch

LM118 – Bachelor of Engineering in Electronic
and Computer Engineering

Project Interim Report

Shane Ginty
21234442

Prof. Colin Flanagan

24/10/2024

Abstract

This project aims to create a relatively small-scale large language model (LLM) from scratch using the Python programming language and the open-source machine learning framework Pytorch. This interim report is to demonstrate the progress on the project to date

The objective of this project is to develop a custom built generative artificial intelligence in the form of a large language model based on the transformer architecture that can be trained with a relatively small dataset of text data and implemented for autoregressive text generation to predict and generate the next word in a sequence or sentence based on the words currently present and the previous words generated. A Large Language Model (LLM) is a neural network that utilizes deep learning to process natural language and other sequential data to understand, generate and respond to human like or natural language text. [1]

This project begins from scratch and will begin at the foundations of LLMs and will address methods for data preparation including tokenization and embedding. It will introduce transformer architecture and its principles of self-attention. Pretraining will take place on the model with a small dataset followed by more training and finetuning to produce coherent and accurate outputs.

The motivation to undertake this project is in part due to the rapidly increasing uptake of artificial intelligence in many aspects of the IT industry. This uptake has led to massive employment growth for AI specialists. Undertaking this project aligns with the goal of developing foundational skills related to this growth in the industry capitalize on the expanding career opportunities in the sector.

Declaration

This interim report is presented in part fulfilment of the requirements for the LM118 Bachelor of Engineering in Electronic and Computer Engineering Bachelors Project.

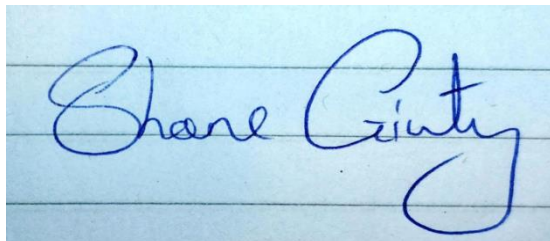
It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.

Where there has been made use of work of other people it has been fully acknowledged and referenced.

Name

Shane Ginty

Signature



Date

24 / 10 / 2024

Table of Contents

ABSTRACT	I
DECLARATION	III
TABLE OF CONTENTS.....	V
LIST OF FIGURES.....	VII
LIST OF EQUATIONS	IX
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 MOTIVATION FOR THE PROJECT	3
CHAPTER 3 BACKGROUND THEORY	5
3.1 Data Preparation	5
3.2 Transformer Architecture Introduction	6
3.3 Attention	7
3.4 Dropout	9
3.5 Loss Functions and Gradients	9
3.6 Normalisation	10
3.7 Backpropagation	11
3.8 Activation Functions	12
3.9 Postprocessing	13
CHAPTER 4 ACTION PLAN.....	15
CHAPTER 5 CONCLUSIONS AND FUTURE WORK.....	17

REFERENCES.....	19
APPENDIX A: PROJECT GANTT CHART	- 1 -
APPENDIX B: INTERIM PRESENTATION SLIDES	- 3 -
Appendix C: Weekly Reports	- 3 -

List of Figures

- (i) Page 8: Diagram representation of self-attention computation example. S. Raschka, Build a Large Language Model (From Scratch), Fig. 3.17, Shelter Island, NY: Manning Publications, 2024.
- (ii) GPT model including a transformer architecture block. S. Raschka, Build a Large Language Model (From Scratch), Fig. 4.15, Shelter Island, NY: Manning Publications, 2024.

List of Equations

Replace this text with a table showing equation numbers, captions and page numbers.

Attention: (Page 7)

$$Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}})V$$

Q: Query, K: Key, V: Value matrices

K^t : Key matrix transposed, d_k : Key vector dimensions

Softmax: function of converting attention scores to probabilities.

Layer Normalisation: (Page 9)

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2}}$$

\hat{x}_i : Normalised input vector

x_i : Input feature.

μ : Mean.

$\sqrt{\sigma^2}$: Variance.

Activation equations: (Page 11)

$$ReLU(x) = \max(0, x)$$

$$GELU(x) \approx 0.5x(1 + \tanh(\pi 2(x + 0.044715x^3)))$$

Chapter 1 Introduction

There has been an explosion in popularity and research into the topic of artificial intelligence in recent years. Due in part to the emergence of what is known as the transformer architecture, the underlying model behind many of the popular AI applications used today such as OpenAIs ChatGPT, Microsoft Bings CoPilot and tools for applications such as language translation, chatbots, text summarization and sentiment analysis.

A transformer is a deep learning architecture model that can process sequential data in parallel rather than sequentially and can identify data's importance in a sequence. This is an improvement over past deep learning models and significantly improves the efficiency, performance and scalability of a deep learning, natural language processing tool such as a large language model (LLM).

This project aims to create an LLM from scratch. By doing so we explore the foundations and underlying principles of the transformer architecture for machine learning and natural language processing.

LLMs are the technology behind much of the popular AI technology right now, including applications used for language translation, chatbots, summarization and sentiment analysis. The foundations of LLMs is the revolutionary transformer architecture

The project will be written in the Python programming language and developed mainly in the visual studio code integrated development environment. The project relies on many of the tools present in Pytorch, an open-source machine learning library based in Python.

While many utilize general purpose LLM models like ChatGPT or Bing Copilot, that have been trained with a massive number of parameters and perform adequately at many operations, sometimes these massive models may not be the most efficient for industry specific operations. The advantage of a custom built LLM is that when it is trained with a specific task it can outperform a large GPT model in regard to a narrowly defined purpose. For example, MedPaLM, an LLM developed by Google DeepMind is fine-tuned to medical datasets and has been shown to outperform general purpose LLMs on tasks such as medical question and answering and performance in medical exam questioning. [2]

Chapter 2 Motivation for the project

The motivation to undertake this project is in part due to the rapidly increasing uptake of artificial intelligence in many aspects of the IT industry. This uptake has led to massive employment growth for AI specialists. The World Economic Forums “Future of Jobs Report 2023” lists AI and machine learning specialists alongside data analysts and scientists as the most prominent emerging roles in the economy [3]. It predicts a 40% jump in the number of AI and machine learning specialist along with a 30% rise in data scientists and 31% increase in information security analysts leading to a combined 2.6 million jobs created by 2027. Undertaking this project aligns with the goal of developing foundational skills related to this growth in the industry and to capitalize on the expanding career opportunities in the sector.

This rapid growth in AI specialists is driven by massive investment into the technology from companies and governments internationally. Major IT companies are investing billions of dollars into LLM development and talent acquisition. In 2023 Microsoft agreed a multi-year, multi-billion-dollar investment into OpenAI with the goal of accelerating breakthroughs in AI and supercomputing at scale [4]. The European Union has also invested heavily into AI including the Digital Europe Program which aims to fund AI with a total of 2.1 billion euro from 2021 to 2027 [5] This activity in the industry showcases how prevalent AI and LLM development is at the moment and how relevant it will remain in the future giving rise to many opportunities and career paths for developers and others in the industry.

Other motivation for this project is an interest in AI developments and its rising popularity in recent years. In 2023 OpenAI released GPT-3.5 allowing users interact with its human like question-and-answer model and recognise its potential practical use in a range of everyday applications such as helping to write emails, answering queries, helping to explain complex subjects, assistance with problem solving and much more. For companies and businesses, they can utilize LLMs to power chatbots that can handle basic customer queries and troubleshooting, reducing wait times for customer support. It can also be utilised to analyse customer data and generate more accurate advertisements and recommendations to enhance customer engagement at an individual level. These are only scratching the surface

of LLM use. Developing a foundational understanding of how some of these operations work serves as motivation to undertake and complete this project.

Chapter 3 Background Theory

This chapter will touch upon the background theory of the project. It will focus on the transformer architecture that acts as the backbone of this project and why it is an improvement over the previous industry standard used for natural language processing (NLP). It will dive into the different aspects that allow the model to function and importance of each step from input to output.

This project relies on the transformer architecture, a deep learning model that can handle sequential data. It encodes input tokens into dense vector representations that can capture the relationships between tokens and their positions relative to each other in the sequence. From these relationships the transformer can deduce the next token in a series using its self-attention mechanisms and mathematical operations to select the token to follow the sequence and then decode these tokens into human readable text.

3.1 Data preparation

This project centres around the construction of an LLM based on the transformer architecture and capable of handling text data to generate replies and answers. The first step in the process is understand how raw data is handled by the system. The system can't handle basic text, it is not compatible with the many mathematical operations that take place in a neural network. Text must be converted into vector format. This conversion is known as embedding. Many different forms of data such as text, video or audio can be embedded using specific embedding models. In this project we will work exclusively with text data. The first step in text embedding is tokenization. A string of text is split into individual words and special characters. These words and characters correlate to an integer value. A popular tokenisation concept is Byte Pair Encoding (BPE). BPE has been used to train LLMs like GPT-2 and GPT-3 [8]

For this project we will implement BPE using an existing open-source library named tiktoken [9]. With tiktoken we can access the publicly available vocabulary used in the GPT-2 model consisting of 50,257 tokens and token IDs, including special characters and sub-tokens.

Tiktokens BPE with the GPT-2 encoding scheme is powerful and flexible. If faced with an unknown word not present in the vocabulary it can construct a token id by breaking up the word into individual characters and assigning sub-tokens that come together to represent a token ID of an unknown word.

With the tokenizer in place the next step in data preparation is mapping token IDs to embedding vectors. This vector has multiple dimensions detailing weights relative to each other vector present, the more dimensions present the better it can represent semantic meaning and inter relationships with other data. In this project the embedding layer is handled by calling the Pytorch neural network embedding class on the vocabulary. In the beginning the embedded vectors weights are random until initialized or optimized during training. The token embeddings are prepared with positional embeddings to help the model understand the order of the tokens in a sequence. Since transformers process data in parallel rather than in sequence it's important to add information to retain sequence order. This is achieved by creating a position vector layer using Pytorch and adding it directly to the token embedding layer, the output of this sum is a layer of input vectors with positioning encoded.

With text data now prepared, the next step is to address the subject of the self-attention mechanism that underlies the transformer model. For context We will address the predecessor of the transformer model and the issues that led to the development of the current industry standard.

3.2 Transformer architecture Introduction

Before the transformer architecture came onto the scene, Recurrent Neural Networks (RNNs) were the standard for deep learning models. RNNs are an encoder-decoder based architecture that handles sequential data one token at a time and saves each step in a hidden state vector as it moves forward in the sequence. RNNs worked well when processing short sequences but the model begins to experience issues when faced with large single sequences since its hidden states become larger and more complex to process, leading to more relevant data in the sequence having its importance diluted. To overcome this, researchers developed Bahdanau attention mechanisms in 2014, which modifies the encoder-decoder relationship

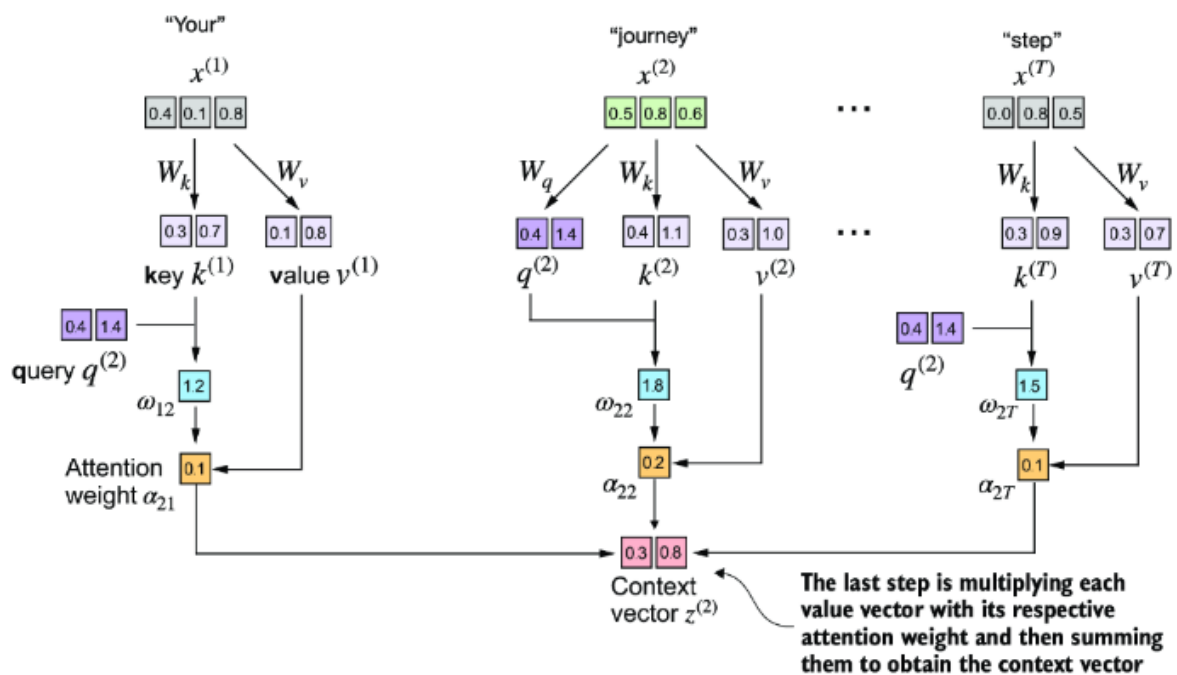
so that the decoder can select specific parts of the sequence at each decoding step instead of the entire sequence. The importance of specific parts of the sequence is determined by attention weights assigned to each token. In the years following, the transformer architecture was proposed with a self-attention mechanism inspired by the Bahdanau attention. This meant data within a transformer model could consider its relevancy compared to all other data in the sequence and weight itself accordingly and dynamically. Allowing for more efficient handling of long and complex data leading to its suitability for natural language processing and LLMs [7].

3.3 Attention

Inside of the transformer block, input vectors to the system are assigned an attention weight. With attention calculated each element in the input vector becomes a context vector which can be thought of as an even further enriched embedding vector that contains information from all other elements in the input sequence to help further improve semantic and inter relationships of an input vector and its relevancy compared to other embeddings in the system. Attention is calculated by breaking up an input vector into three separate representations. Query, key and value vectors. Where the query accesses the relevance of other tokens in the sequence compared to itself to determine relationships and relevance with other tokens. The key identifies tokens within a sequence and value represents the specific feature of the token in question.

The attention score is calculated by taking the dot product of a vectors query with every other input token value. The dot product yields a scaler value that quantifies how similar two vectors are, the higher the dot product the higher the similarity and thus the attention score between two elements. When the attention scores are calculated they are then normalised. Normalization is covered later on in this document but its goal here to adjust the attention weights so that the sum of them all becomes equal to 1. This helps maintain stability in the system. In this project we use Pytorch SoftMax function to implement our normalization layer in attention scoring. The SoftMax function will be covered later again when addressing transformer operations. The final calculation that results in our context vector is achieved by multiplying the embedded input tokens with corresponding attention weights. The sum of these multiplied vectors results in a context vector.

The next topic related to this projects LLM development is to modify the above self-attention to masked attention. For LLM tasks such as next word prediction and sentence completion, self-attention should only consider the current and previous tokens when calculating the next token in a sequence. This prevents the model from accessing data further in the sequence. To do this we utilize the Pytorch trill function which sets elements above the diagonal of a matrices to 0. Attention weights are then renormalized to redistribute attention scoring appropriately.



- (i) Diagram representation of self-attention computation example. S. Raschka, Build a Large Language Model (From Scratch), Fig. 3.17, Shelter Island, NY: Manning Publications, 2024.

The final step we undertake in attention scoring in the project is the implementing multi-head attention. In multiheaded attention multiple attention modules are ran independently in parallel outputting their own context vectors that then combine together and transformed to output a unified representation of the context vector.

3.4 Dropout

A method that we will implement in the transformer foundation is dropout. It is a technique used during the training phase of the LLM where randomly selected units are selected and “dropped” or ignored during training. This helps avoid issues like overfitting where the model becomes over reliant on the training data. We will apply a dropout mask after calculating attention weights to drop randomly selected units.

3.5 Loss Functions and Gradients

Before going any further will discuss some basics on how weights are dynamically adjusted during the training phase. This includes touching upon “loss function” and “gradients”.

Loss function is a mathematical function used in machine learning to quantify the difference between the predicted output of a model and the actual target value expected. It is a measure of a model’s predictions power to true test data. The goal in model training is to minimise loss function thereby improving the model’s accuracy and coherence.

Gradient is measure of loss function with respect to the models’ parameters. When we know the gradient, we know the discrepancy of the current model vs the expected outcomes. By adjusting weight values in the opposite direction of the gradient the process helps the model learn by adjusting weights to reduce error in predictions and increase model performance.

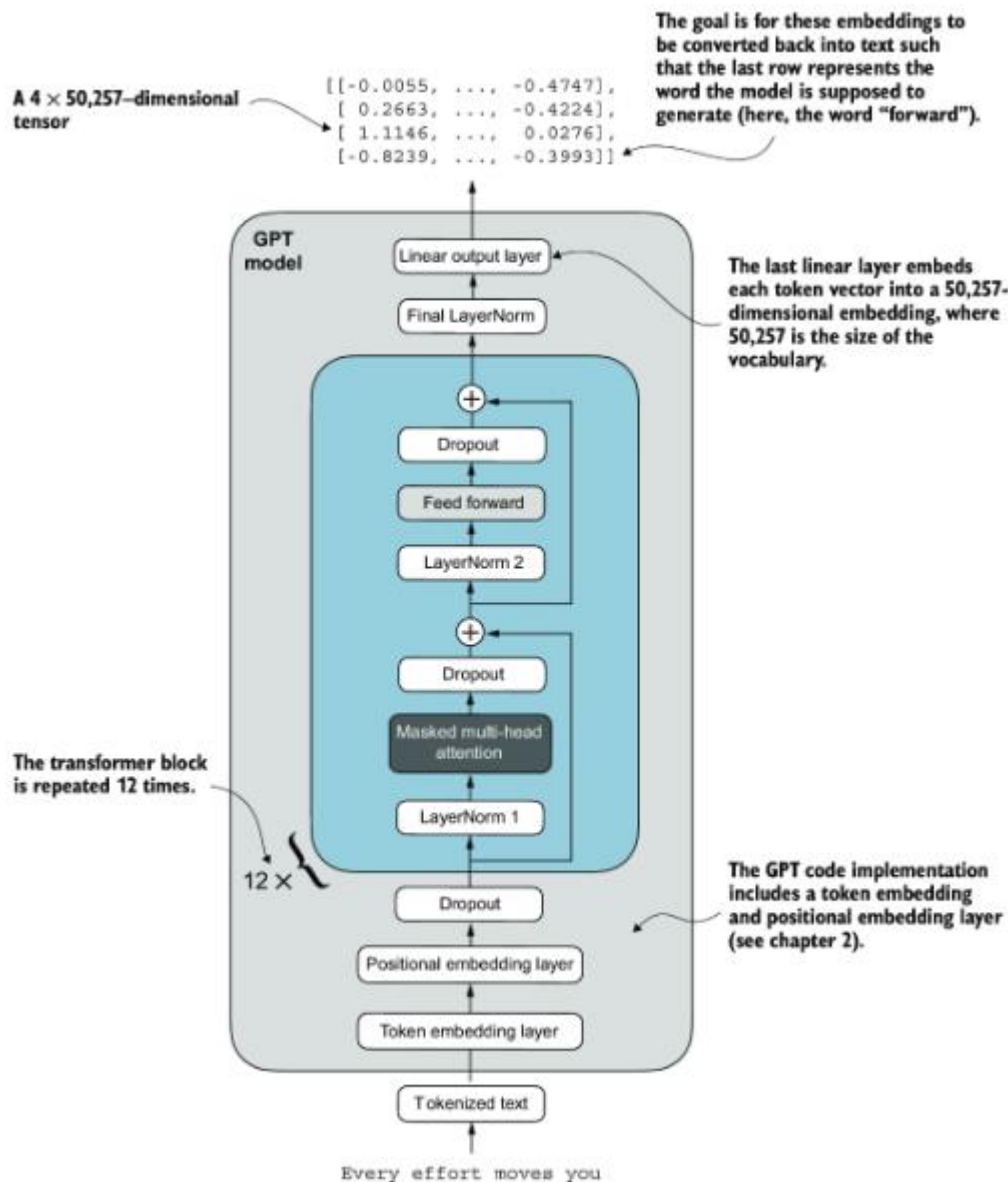
Some common issues to consider when working with gradients are exploding and vanishing gradients. Vanishing gradient is an issue when the gradient becomes very small as they propagate back through the layers during training. This slows down learning because the updated gradient values are too small, and the weight changes are negligible. Its more prominent in models with many layers. One method of avoiding vanishing gradients is to utilise shortcut connection within the transformer which will appear later in the document. Exploding gradients is the opposite, when gradient calculations are too large which can lead to an unstable model that has dramatic and unpredictable changes to the weights. One method of mitigating these issues is normalisation.

3.6 Normalisation

The first step of the transformer block is layer normalisation. Normalisation is a technique to adjust input values so that they fall within a specific range. The goal of normalisation is to make training more efficient and stable by ensuring input data follows a consistent scale. Layer normalisation leads to a mean of zero and a variance of 1. This is accomplished by calculating the initial mean of the input features, then subtracting the mean from each value. This creates a new vector with a mean of 0. We then calculate the variance of these new “centred” values. To achieve a variance of 1 we divide the centred values by the standard deviation resulting in a variance of 1. This process of normalisation helps avoid issues like vanishing or exploding gradients which can destabilise the training process.

Following normalisation, the transformer conducts masked multi-head attention calculations on its input vectors followed by a dropout mask. Within the transformer block we will implement a shortcut connection. The shortcut connection allows gradient calculations to flow through the layer and can be added back post attention calculation. This helps prevent gradients becoming too small during training phases and allows stable gradient calculations.

After incorporating the shortcut connection, another layer of normalisation is applied followed by backpropagation.



(ii)GPT model including a transformer architecture block. S. Raschka, Build a Large Language Model (From Scratch), Fig. 4.15, Shelter Island, NY: Manning Publications, 2024.

3.7 Backpropagation

Backpropagation is a technique for training deep neural networks. It consists of two steps, the forward pass and the backwards pass. During the forward pass, the input data is passed forward through the network layer by layer. Each layer processes the inputs using weights and activation functions and produces an output. When the final output is generated the loss function error is calculated.

In the backward pass, the loss function error is passed back through the network. The gradient of each weight within each layer is calculated. This indicates how much each weight contributes to the error, they are then adjusted to reduce loss.

The forward pass computes attention scores and normalises them using SoftMax. It then produces a context vector. SoftMax is a mathematical function used in machine learning and neural networks. It converts the raw model output vectors, or logits into a probability distribution. For this project we can access SoftMax as a function of Pytorch.

3.8 Activation functions

During the feed forward process of the transformer an activation function is introduced. An activation function is a mathematical function applied to neurons in a network to introduce non-linearity. This is important because if a model remained entirely linear, then no matter how many layers it consisted of, the model would still behave as a single layer model and limit in its ability to complete complex tasks. If the whole system was a linear model, it wouldn't have the ability to learn complex relationships between inputs and outputs

Activation functions help control what neurons in the system should "activate" at a given time and which should not. This allows a network to focus on relevant information in input data and enables the model to "learn".

ReLU (Rectifies Linear Unit activation has been commonly used in deep learning in the past, it thresholds negative inputs to 0. Ensuring only positive values however for the construction of this project's LLM we will utilize GELU (Gaussian Error Linear Unit). GELU is more complex but smoother and more efficient than ReLU.

Following the feed forward loop another layer of dropout and a shortcut connection to previous gradients are introduced. The transformer block then passes its context vectors through a final layer of normalisation.

3.9 Postprocessing

In the post processing stage of the LLM, the context vector is extracted from the transformer block. For next token generation, a softmax function is applied to convert vectors into a probability density that can interpret the likelihood of what will be the next token in the sequence. The feature is decoded from its embedding to token ID which is then mapped back to the vocabulary and outputs that word.

With the foundation of data preprocessing, a transformer block and data post processing, the project has a foundational LLM machine capable of generating text. But without weight initialisation and pre training the model generates incoherent text. This is the state of the project at the time of writing this interim report.

Chapter 4 Action plan

The timeline of this project ranges from week one of the autumn semester 2024 through the winter holiday 2024 up until the final report deadline in the midpoint of the spring semester 2025. (See Appendix A. Project Gantt Chart)

Milestones of the project to date include:

- Researched fundamentals of LLM machines and transformer architecture.
- Basic text preprocessing with tokenisation using tiktoken Byte Pair encoding
- Implementation of transformer sublayers with Pytorch neural network functions.
- Construction of a fundamental LLM with incoherent text.
- Final draft writeup of autumn interim presentation and report

Future target milestones:

- Development of a pretrained model with improved text generation coherence. Using publicly available pretrained weights from OpenAIs GPT-2 model.
- Presentation of the autumn interim report and submission of the interim report document.
- Further development of LLM model with adapting pretrained weights for more capable text generation
- Further research to deepen understanding of machine learning fundamentals, deep learning and Pytorch utilities

Chapter 5 Conclusions and future work

This interim report has hopefully successfully conveyed the work that has gone into the project so far. The main focus so far has been Research and construction of a foundational LLM with the goal of outlying the fundamental steps, layers and sub-layers of the operations that occur in the processing of an LLM as well as the pre and post-processing for text generation based on an input string of text and a vocabulary.

One of the goals of this project was to become familiarised with the technologies that act as the building blocks of LLMs and the transformer architecture model. At this stage of the project, I am confident of that significant progress has been made towards that goal and that further development of the project will strengthen and increase that knowledge.

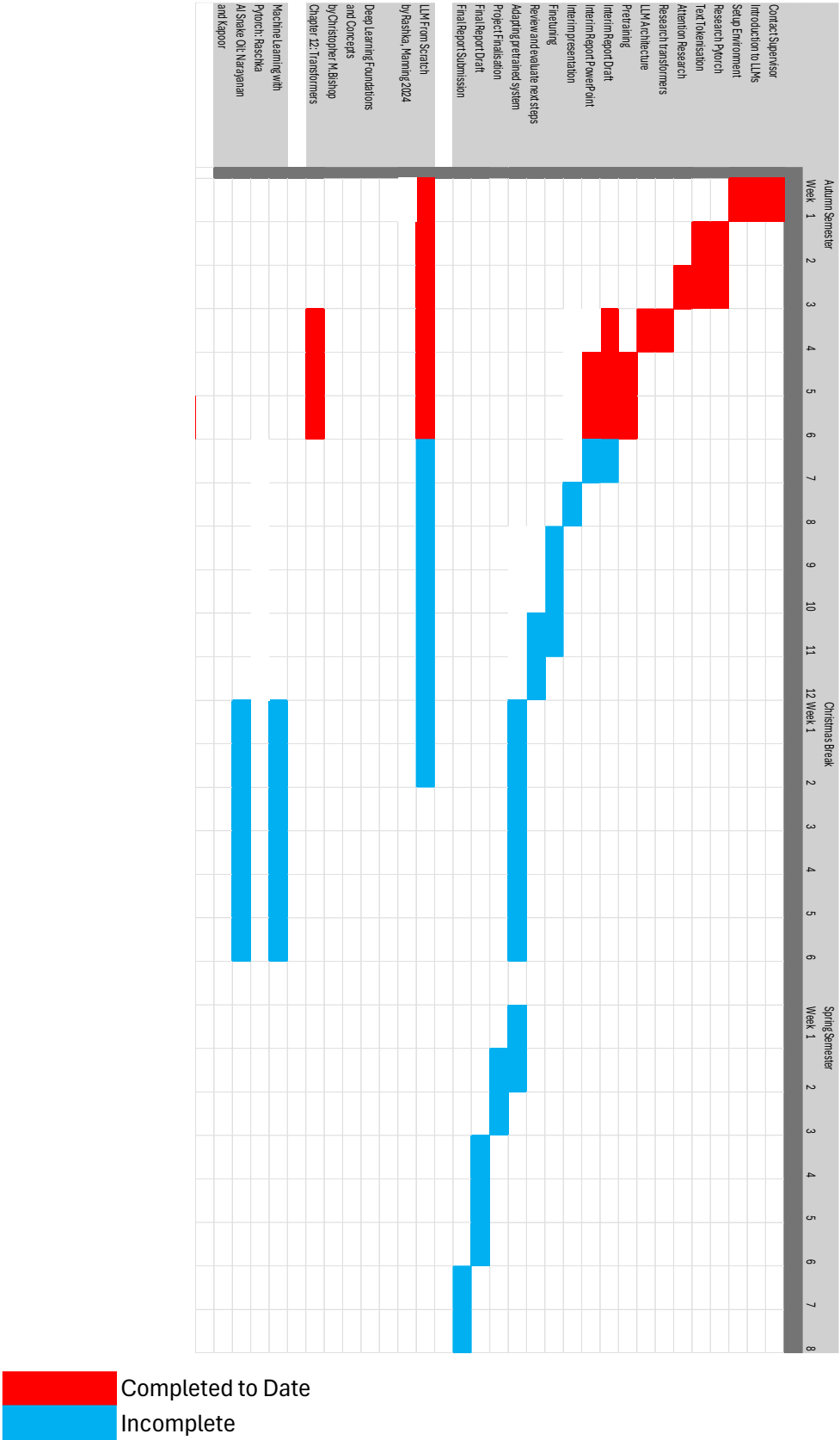
The future goal of the project is taking the foundations and developing a more capable model through pretraining and finetuning and adjusting the model to be capable of coherent text generation and basic question and answer tasks by developing instructions models for the system. This will require further research and study into the subject.

Chapter 6 References

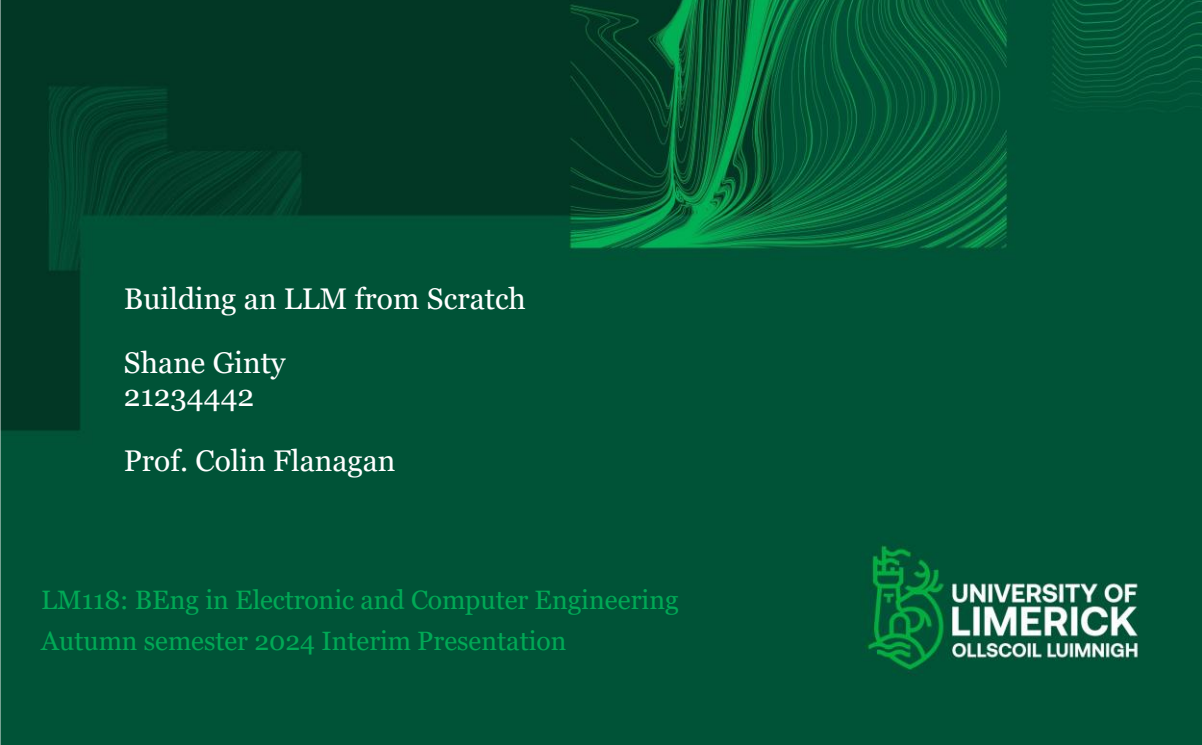
- [1] S. Raschka, *Build a Large Language Model (From Scratch)*. Shelter Island, NY: Manning Publications, 2024.
- [3] World Economic Forum, *The Future of Jobs Report 2023*. World Economic Forum, Geneva, Switzerland, 2023. [Online]. Available: <https://www.weforum.org/reports/the-future-of-jobs-report-2023>
- [2] **Google Research**. "Our latest health AI research updates," *The Keyword* (Google Blog), Mar. 14, 2023. [Online]. Available: <https://blog.google/technology/health/ai-ilm-medpalm-research-thecheckup/>.
- [4] **Microsoft Corp.**, "Microsoft and OpenAI extend partnership," *The Official Microsoft Blog*, Jan. 23, 2023. [Online]. Available: <https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/>.
- [5] **European Parliament Research Service**, "Artificial intelligence: Tackling challenges and promoting innovation," *European Parliament*, Jan. 2024. [Online]. Available: [https://www.europarl.europa.eu/RegData/etudes/ATAG/2024/760392/EPRS_ATAG\(2024\)760392_EN.pdf](https://www.europarl.europa.eu/RegData/etudes/ATAG/2024/760392/EPRS_ATAG(2024)760392_EN.pdf).
- [6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014. [Online]. Available: <https://arxiv.org/abs/1409.0473>
- [7] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>.
- [8] A. Radford et al., "*Language Models are Unsupervised Multitask Learners*," OpenAI, 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
- [9] OpenAI, "*tiktoken: Fast Byte-Pair Encoding (BPE) Tokenizer for use with OpenAI models*," GitHub repository, 2023. [Online]. Available: <https://github.com/openai/tiktoken>.

[10] Christopher M. Bishop, *Deep Learning: Foundations and Concepts*. Cham, Switzerland: Springer, 2023.

Appendix A: Project Gantt chart



Appendix B: Interim Presentation Slides




Building an LLM from Scratch

Shane Ginty
21234442

Prof. Colin Flanagan

LM118: BEng in Electronic and Computer Engineering
Autumn semester 2024 Interim Presentation



UNIVERSITY OF
LIMERICK
OLLSCOIL LUIMNIGH



Presentation overview

- Project Objective
- Introduction to Large Language Models and the Transformer Architecture
- Data Preparation (Tokenisation & Embedding)
- Self-attention
- Transformer Block
- Post-processing / Decoding
- Gantt Chart
- Future Work
- Conclusions

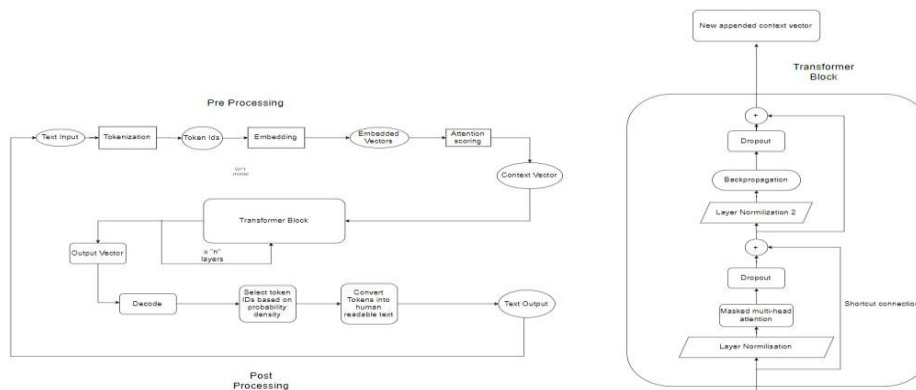


Project overview (1)

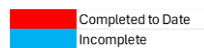
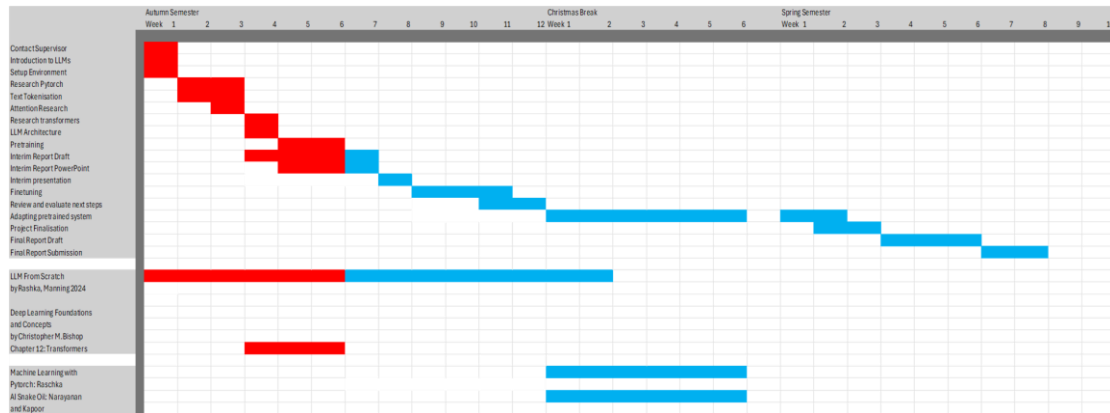
- The objective of this project is to construct a large Language model (LLM) from the ground up.
- An LLM is a type of AI designed for Natural language processing (NLP) and enabled by advancements in deep learning and neural networks.
- They are trained on vast quantities of data that when processed allows the model to capture semantic and contextual meaning and generate information in a human readable form for tasks such as language translation, sentiment analysis and classification tasks.
- Currently the project application generates text to complete a sentence, but it is incoherent. The next step is to train the model.
- The project report will go into detail of the transformer architecture and discuss its advantages over the previous natural language processing (NLP) method of Recurrent Neural Networks (RNNs) and discuss transformers foundational concept of attention.
- The project report will detail the advantages and disadvantages of using a custom built LLM, the breakthroughs in machine learning and neural network technologies in recent years and the popularization of LLM based services such as OpenAIs GPT model.
- The expected outcome from this project application is to have a small LLM capable of coherent text generation and classification tasks. Through the development of a transformer model with Pytorch utilities. Engaging with these technologies aims to enhance knowledge of the functions of these models and provide a deeper understanding of their mechanics, capabilities and challenges.

Project overview (2)

- Block diagram of a basic LLM procedure to generate text to append to a sequence



Project Gantt chart



Conclusions

- Project summary:
 - To date the project utilizes tokenization, embedding and transformer architecture blocks to take an input of text data and generate following tokens that convert back into text to complete the sentence
- Work completed to-date:
 - Research and implantation of Pytorch library for deep learning and neural networks
 - Pieced together a foundational LLM that incorporates text tokenisation, embedding, self-attention and transformer architecture for encoding and decoding resulting in simple text generation to complete a text sequence.
- Next steps:
 - The next steps for the application is to implement pretraining and begin some finetuning to the system. This will be achieved with further research and implementation throughout the remainder of the autumn semester. Over the Christmas break, more research and adapting the pretrained application will take place to understand and execute tasks. Finally, over the Spring semester I will produce an end version of the system along with the final report and presentation of the project.



**Thank you.
Any questions?**



**UNIVERSITY OF
LIMERICK**
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 TP9X,
Ireland.
Ollscoil Luimnigh,
Luimneach,
V94 TP9X, Éire.
+353 (0) 61 202020

ul.ie

Appendix C: Weekly Reports

Week 1, 09/09 – 15/09:

Contacted project supervisor for introduction and advice on a starting point.

Investigated some of the recommended reading [1]

Set up the development environment. Decided on using vscode for python development and running python files from a windows subsystem for Linux (WSL) terminal using python3.

Investigated other potential IDEs such as Jupyter lab and notebook with interactive python notebook files.

Began reading the introduction of Build a Large Language Model (From Scratch) by S. Raschka, Manning publications [1]

Week 2, 16/09 – 22/09:

Began reading chapter two of “Build a Large Language model from scratch”. This chapter spoke about data tokenisation and embedding. Did some research on embedding.

Developed test programs to split a text file into list of words and characters using regular expressions. These words and character were then converted into integer values by taking their index position in the list and using that position as a token id in the vocabulary. The function.

Installed Pytorch to my PC

Researched Byte Pair encoding and make some test programs for practise using tiktoken BPE with GPT-2 vocabulary encoding scheme and vocabulary.

Week 3, 23/09 – 29/09:

Investigated Google CoLab as a development environment. Being able to run programs over the cloud with a dedicated GPU system could be useful. Decided for the beginning of the project, my local PC with Pytorch installed will be sufficient.

Reviewed some basic Pytorch functions and tensor operation.

Reviewed “Build an LLM from Scratch” chapter three which introduced self-attention mechanisms with attention weights and context vectors.

Week 4, 30/09 – 6/10:

Continued Chapter three of “Build an LLM from scratch” with information on multi-head attention

Researched further into transformer architecture with the book “Deep Learning foundations and concepts” [10]. Chapter 12 speaks exclusively on transformers and attention mechanisms including its advantages over Recurrent Neural Networks (RNNs)

Completed chapter three of “Build an LLM from scratch” and began reading chapter four which spoke about LLM architecture including normalisation, and backpropagation.

Week 5, 7/10 – 13/10:

Began the draft document for the autumn interim report and presentation.

Continued chapter four of “Build an LLM from Scratch”.

Researched backpropagation and activation functions including ReLU and GELU functions

Week 6, 14/10 – 20/10:

Completed chapter four of “Build an LLM from Scratch and developed a foundational LLM that when given a string input and a number of words to add to the string will predict and generate text to complete the sequence. Without any pretraining or weight initialisation the generated text is incoherent, but it is a significant milestone that shows promise for the project.

Week 7, 21/10 – 27/10:

Constructed final draft of the autumn interim report and presentation with preparations to submit.

