**Software Design**

**Document**

**for**

**Self-Attention Mechanism of ChatGPT**

**Version 1.0.2 approved**

**Prepared by Fardeen Abir**

**Nathan Campos**

**Anthony Edeza**

**Jose Flores De Santiago**

**Edward (Jong Yeon Kim) Fjura**

**Kenneth Lieu**

**Kevin Mateo**

**Michael Nguyen**

**Roberto Reyes**

**Maggie Yang**

**Dr. Russ Abbott**

**Created 1/04/23**

Table of Contents................................................................................................................... <pg 2>

Revision History..................................................................................................................... <pg 3>

1. Introduction................................................................................................................ <pg 4>
   1. Purpose........................................................................................................... <pg 4>
   2. Document Conventions……………………................................................. <pg 4>
   3. Intended Audience and Reading Suggestions................................................ <pg 4>
   4. System Overview........................................................................................... <pg 4>
2. Design Considerations............................................................................................... <pg 5>
   1. Assumptions and dependencies..................................................................... <pg 5>
   2. General Constraints........................................................................................<pg 5>
   3. Goals and Guidelines.....................................................................................<pg 6>
   4. Development Methods................................................................................... <pg 6>
3. Architectural Strategies.............................................................................................. <pg 7>
4. System Architecture................................................................................................... <pg 8>
   1. ....................................................................................................................... <pg 8>
   2. ....................................................................................................................... <pg 8>
5. Policies and Tactics.................................................................................................... <pg 9>
   1. Specific Products Used.................................................................................. <pg 9>
   2. Requirements traceability............................................................................... <pg 9>
   3. Testing the software....................................................................................... <pg 9>
   4. Interfaces........................................................................................................ <pg 9
   5. System's deliverables..................................................................................... <pg 9>
   6. Abstraction..................................................................................................... <pg 9>
6. Detailed System Design...........................................................................................<pg 10>

6.1 Name of Module.........................................................................................<pg 10>

6.1.1 Responsibilities....................................................................................<pg 10>

6.1.2 Constraints...........................................................................................<pg 10>

6.1.3 Composition.........................................................................................<pg 10>

6.1.4 Uses/Interactions..................................................................................<pg 10>

6.1.5 Resources..............................................................................................<pg 11>

6.1.6 Interface/Exports..................................................................................<pg 11>

1. Detailed Lower level Component Design

7.1 Name of Class or File....................................................................................<pg 12>

7.1.1 Classification........................................................................................<pg 12>

7.1.2 Processing Narrative(PSPEC)..............................................................<pg 12>

7.1.3 Interface Description............................................................................<pg 12>

7.1.4 Processing Detail...................................................................................<pg 12>

7.1.4.2 Restrictions/Limitations................................................................<pg 12>

7.1.4.3 Performance Issues.......................................................................<pg 12>

7.1.4.4 Design Constraints.......................................................................<pg 13>

7.1.4.5 Processing Detail For Each Operation.........................................<pg 13>

1. Glossary....................................................................................................................<pg 14>
2. References................................................................................................................<pg 15>

**Revision History**

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Initial | 03/25/23 | Creation of Document | 1.0.0 |
| Premise | 04/10/23 | Layout and Project Scope | 1.0.1 |
| Secondary | 05/01/23 | Creation for Final Submission | 1.0.2 |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

The purpose of this Software Design Document (SDD) is to provide a detailed description of the design of the software system that will be developed based on Andrej Karpathy's miniGPT tutorial. The SDD will outline the software architecture, system components, and their interactions. It will also describe the design considerations, assumptions, and constraints that were taken into account during the development process.

**1.2 Document Conventions**

This SDD will use a combination of text, diagrams, and code snippets to describe the software design. The document will follow standard formatting conventions, including headings, subheadings, and bullet points for easy readability.

**1.3 Intended Audience and Reading Suggestions**

The intended audience for this SDD includes software developers, researchers, and students who are interested in the field of natural language processing and want to learn how to build a language model using the miniGPT tutorial. The reading suggestions include familiarity with Python programming language, neural networks, and the transformer architecture.

**1.4 System Overview**

The system that will be designed based on the miniGPT tutorial will be a language model that can generate coherent and relevant text based on a given prompt. The system will consist of several components, including a pre-processing module for text data, a training module that uses the transformer architecture and self-attention mechanisms, and a text generation module that produces output based on the learned patterns. The system will be designed to be modular, extensible, and easy to use, with clear documentation and well-defined interfaces between components.

**2. Design Considerations**

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

**2.1 Assumptions and Dependencies**

* Related software or hardware
* Operating systems
* End-user characteristics
* Possible and/or probable changes in functionality
* Programming language - Python/Pytorch
* Shakespearean English text as training data
* Word Embedding

**2.2 General Constraints**

* Hardware or software environment
* Availability or volatility of resources
* Complexity of the Model
* Data repository and distribution requirements
* Memory and other capacity limitations
* Performance requirements

**2.3 Goals and Guidelines**

The purpose of this software design document (SDD) is to provide a detailed description of the software architecture and design of the miniGPT language model implementation based on Andrej Karpathy's tutorial. The primary goals of this document are to:

* Clearly describe the design decisions and trade-offs made in the implementation of the miniGPT model, including the choice of programming language, libraries, and algorithms used.
* Provide a detailed overview of the software architecture of the miniGPT model, including the different components and how they interact with each other.
* Document the design patterns, coding conventions, and best practices used in the implementation of the miniGPT model.
* Identify potential areas for future improvements and optimizations to the miniGPT model and provide guidelines for making these improvements.

**2.4 Development Methods**

The miniGPT tutorial was developed through a process of research, coding, and testing, resulting in a step-by-step guide for building a simple language model from scratch using Python and PyTorch. Throughout the development process, different language models and neural network architectures were explored, while various hyperparameters and training techniques were experimented with.

The tutorial emphasizes the importance of understanding the underlying principles of natural language processing and neural network architecture, while also considering practical implementation and testing of the model. By highlighting key design decisions and trade-offs, the development methods used in the tutorial provide a comprehensive and accessible introduction to building language models, demonstrating the importance of a systematic and rigorous approach to development.

**3. Architectural Strategies**

The miniGPT tutorial follows a modular approach to the implementation of the language model. The main components of the architecture are the data processing module, the model architecture module, and the training and evaluation module.

* Using the transformer architecture as the basis for the language model.
* Implementing the transformer architecture using PyTorch.
* Using self-attention mechanisms to enable the model to learn long-term dependencies in the data.
* Using a custom training loop to train the model on the text corpus.
* Using a softmax activation function to compute the probabilities of the next token in the generated text.
* Using a top-k sampling technique to generate text that balances coherence and diversity.
* Using a beam search decoding technique to generate higher quality text by considering multiple potential next tokens.

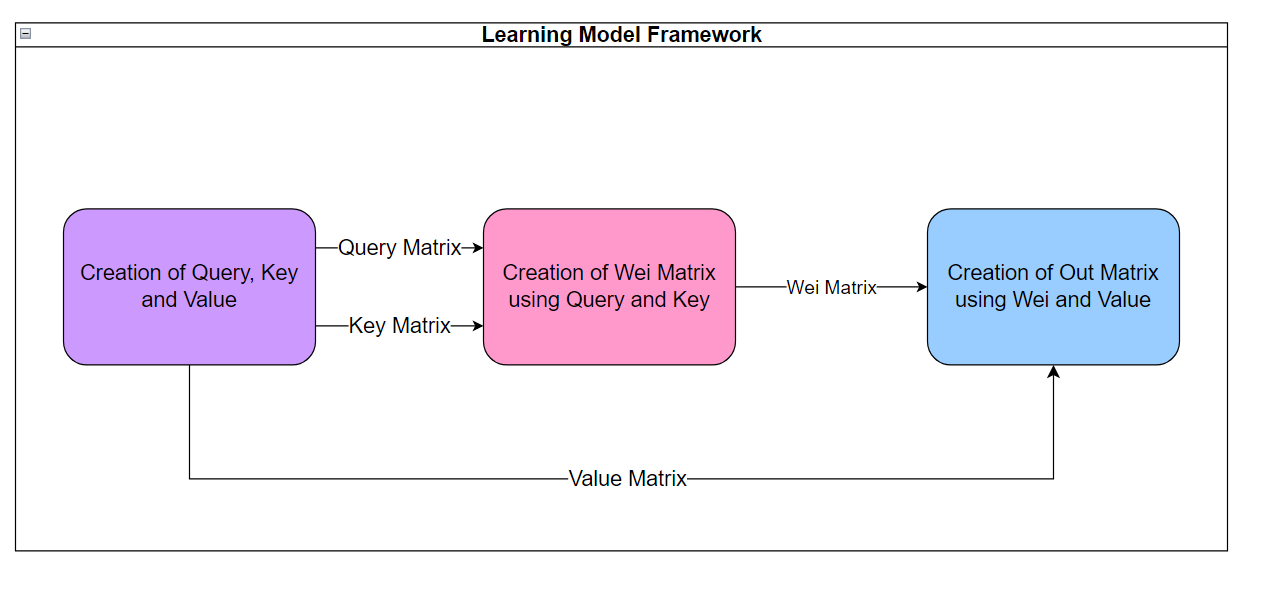
**4. System Architecture**

**4.1 Architecture for miniGPT and the Learning Model Framework**

The input to Andrej Karpathy's miniGPT is Shakespeare text. The miniGPT then learns the format and produces text of characters that is in a similar format to the input Shakespeare text.

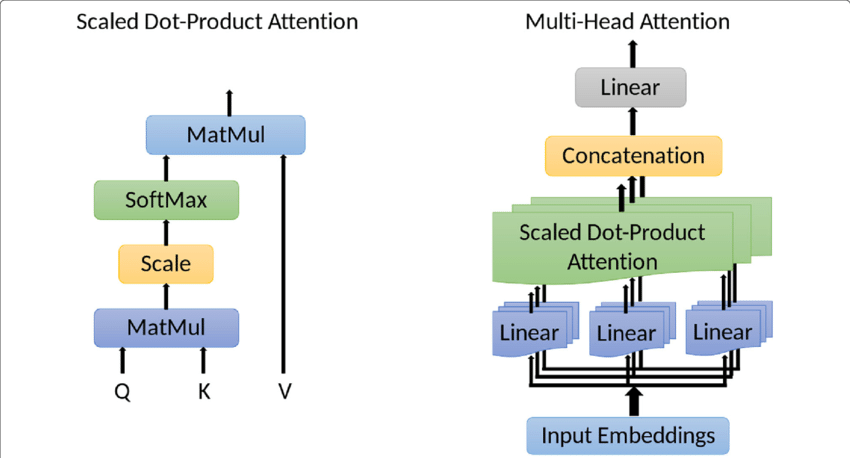


The architecture for the learning model can be broken down into three sections: data setup, creation of wei, and then the creation of out.

Here is a diagram that shows how this architecture works at a high level:

**4.2. Data Flow:**

Here is an overview of the self-attention mechanism described in the tutorial by Andrej Karpathy.



These are the technologies used in the overall tutorial as well as the self-attention mechanism:

**4.2.1 Text Corpus:**

The input data is a corpus of text documents that are used to train the model. The text corpus is preprocessed by tokenizing the text into individual words, converting them to lowercase, and removing any special characters or punctuation.

**4.2.2 Tokenization:**

The tokenized text is fed into the model in the form of sequences of fixed length. Each word in the sequence is represented as an integer using a tokenizer that maps each unique word to a specific integer value.

**4.2.3 Embedding:**

The integer values are then passed through an embedding layer that maps each integer to a vector of continuous values, representing the word in a high-dimensional space.

**4.2.4 Model Architecture:**

The embedded sequences are then fed into the GPT model architecture, which consists of a stack of transformer layers. Each transformer layer has a self-attention mechanism that allows the model to consider the context of each word in the sequence.

**4.2.5 Training:**

The model is trained using backpropagation and gradient descent to minimize the loss between the predicted and actual next word in the sequence.

**4.2.6 Generation:**

Once the model is trained, it can be used to generate new text by predicting the next word in a sequence given a prompt. The generated text is then fed back into the model as input, and the process is repeated to generate longer pieces of text.

**4.3. Implementation:**

Self-Attention Mechanism of ChatGPT consisted of four sections for efficient research: Python syntax, matrix multiplication, self attention, and ChatGPT output. Each section represents the progress of our research and steps we took to understand Karpathy’s version of ChatGPT.

**4.3.1. Python Syntax**

Collection of rules that dictate how a Python program will be structured and interpreted.

**4.3.2. Matrix Multiplication**

Matrix multiplication is a mathematical operation that combines two matrices to produce a new matrix.

**4.3.3. Self Attention**

A technique used in order to compute the next token in sequence. The mechanism computes three vectors for the input which are Key, Query, and Value. These vectors then use matrix multiplication in order to return an out matrix, which gives us how the next token in sequence looks like.

**4.3.4. ChatGPT Output**

Generation of Shakespeare-like text that was trained on sample Shakespeare text given from the user input.

**5. Policies and Tactics**

The miniGPT tutorial employs several policies and tactics to ensure the quality and integrity of the code and the tutorial itself. These include:

**5.1 Version Control:**

The tutorial code is managed using version control software such as Git, which allows for easy tracking and management of code changes, as well as collaboration with other developers.

**5.2 Code Review:**

Before committing changes to the codebase, the tutorial code is reviewed by other developers to ensure that it adheres to best practices and is free of errors.

**5.3 Error Handling:**

The tutorial code includes robust error handling and logging mechanisms to catch and report any errors that may occur during the training or inference process.

**5.8 Data Management:**

The tutorial emphasizes the importance of ethical and responsible data management practices, such as obtaining consent and ensuring privacy, when working with text data.

**5.9 Reproducibility:**

The tutorial provides detailed instructions on how to reproduce the results of the experiments and provides access to the training data and code, making it easy for other researchers and developers to verify and build upon the work.

**6. Detailed System Design**

The miniGPT tutorial implements a transformer-based language model with an encoder and a decoder. The input sequence is tokenized and embedded using a learned embedding layer. The encoder is composed of a stack of transformer blocks with self-attention and feedforward network, while the decoder has a masked self-attention mechanism and cross-attention mechanism. A linear layer and softmax activation function generate a probability distribution over the vocabulary. The model is trained with the cross-entropy loss function and stochastic gradient descent on a large corpus of text data, enabling it to generate coherent and relevant text based on the patterns it learns.

**6.1 Self-Attention**

**6.1.1 Responsibilities**

In the miniGPT tutorial, the self-attention mechanism is a crucial component of the transformer architecture used for the language model. This mechanism enables the model to capture dependencies between different positions in the input sequence and is responsible for attending to different parts of the input sequence during the encoding process.

**6.1.2 Constraints**

Currently, the ChatGPT program focuses only on generating Shakespeare-like text and there isn’t a chat box interaction like the traditional ChatGPT. Since there isn’t a chatbox, there’s limitations on what the user can do with the program.

**6.1.3 Composition**

Embedding - Each character in given input is represented as a vector and each vector is given a set amount of features. Each dimension in the vector will represent a numerical value for that feature. Characters with similar values will be more closely related to each other. Embedding will be used to create key, query, and value matrices.

Matrix Construction - Using matrix multiplication and dot product on key and query matrices will return the wei matrix, which represents how related the tokens/vectors are to each other.

Output Matrix - Using matrix multiplication on the wei and value matrices will return the out matrix, which will show how the next token/vector in sequence will look like.

**6.1.4 Uses/Interactions**

The data organization component of this program uses the input text in order to train on the data. This component will use libraries and software functions in order to use self attention to be able to generate Shakespeare-like text.

**6.1.5 Resources**

The self-attention mechanism software system requires memory, processors, databases, and software libraries for efficient operation. Memory optimization techniques, load balancing algorithms, efficient database management techniques, and proper synchronization of software libraries will be implemented for optimal system operation.

**6.1.6 Interface/Exports**

Self-attention is an important part of the program’s ability to produce Shakespearean-esque text. In the process, matrices termed out are given by self-attention in both the classes Head and MultiHeadAttention, with the latter being made from the former. out in MultiHeadAttention, along with the chain of classes it affects like Block and BigramLanguageModel through being the output of MultiHeadAttention’s function, employed in these other classes, is crucial to the calculations of logits and loss for iterations of training the system’s model. These portions of the code, with logits and loss especially necessary for optimizing the model, would not be present or complete without the implementation of self-attention.

**7. Detailed Lower level Component Design**

The tutorial mainly centers on explaining the overall structure and implementation of the language model utilizing the transformer-based architecture. Although there are a few code examples and explanations of certain functions and modules used in the implementation, there is no thorough breakdown of the lower-level components of the model or their design decisions. The tutorial aims to provide a hands-on introduction to constructing a basic language model using PyTorch and the transformer architecture, rather than a comprehensive analysis of the design and implementation of each component of the model.

**7.1 Data**

**7.1.1 Classification**

This component is a manipulated result from the text in input.txt, which the code downloads from GitHub. The code also outputs information in forms of numbers or words.

**7.1.2 Processing Narrative (PSPEC)**

The text file that is implemented contains some of Shakespeare’s work/text which is then processed by the program.

**7.1.3 Interface Description**

The program takes the Shakespeare text that is provided as input, then goes through the GPT model that outputs text similar to Shakespeare text.

**7.1.4 Processing Detail**

The text file goes through an encoder, decoder, embedding, and tokenization.

**7.1.4.2 Restrictions/Limitations**

Although the program is limited to the amount of text or size of text file, other text files can be used.

**7.1.4.3 Performance Issues**

No Performance Issues to report since all computational processing occurred on Google Collabs.

**7.1.4.4 Design Constraints**

The software depends on processors and GPUs for algorithm execution, and efficient operation also requires considering the system's memory availability and limitations. Since memory size depends on model size and data being processed, it's crucial to ensure sufficient memory for the software. However, it's important to note that Google Colab has a memory limit of 12GB.

**7.1.4.5 Processing Detail For Each Operation**

The Embedding Layer transforms input tokens to vectors while the Self-Attention Layer calculates the attention weights for each token, indicating their importance for other tokens in the sequence. The processing detail includes tokenizing and assigning unique IDs, looking up embeddings, computing query/key/value vectors, calculating dot product, applying softmax function, and multiplying attention weights with value vectors.

**11. Glossary**

1. Language model: a type of neural network model that can generate coherent text based on patterns learned from a large corpus of text data.
2. Transformer architecture: a type of neural network architecture that uses self-attention mechanisms to process sequential data, such as language.
3. Encoder: the part of the transformer model that processes the input sequence and generates a hidden representation of the sequence.
4. Decoder: the part of the transformer model that generates the output sequence based on the hidden representation generated by the encoder.
5. Self-attention: a mechanism in the transformer model that allows the model to attend to different parts of the input sequence when generating the output sequence.
6. Cross-attention: a mechanism in the transformer model that allows the decoder to attend to the output of the encoder when generating the output sequence.
7. Embedding: a learned mapping that maps each token in the input sequence to a high-dimensional vector representation.
8. Tokenization: the process of splitting a sequence of text into individual tokens, such as words or subwords.
9. Stochastic gradient descent: an optimization algorithm commonly used for training neural networks that updates the network weights based on the gradient of the loss function with respect to the weights.
10. Cross-entropy loss: a commonly used loss function for training classification models, such as language models. It measures the difference between the predicted probability distribution and the actual distribution of the target class.
11. PyTorch: an open-source machine learning framework used for building and training neural networks.

**12. References**

Karpathy, A. (2021). The Unreasonable Effectiveness of the Transformer Architecture. Retrieved from <https://karpathy.github.io/2019/04/25/recipe/>

Hugging Face (2021). Transformers Library Documentation. Retrieved from <https://huggingface.co/transformers/index.html>

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

PyTorch (2021). PyTorch Documentation. Retrieved from <https://pytorch.org/docs/stable/index.html>

Brownlee, J. (2021). How to Develop a Transformer Model for NLP. Retrieved from <https://machinelearningmastery.com/how-to-develop-a-transformer-model-for-nlp/>