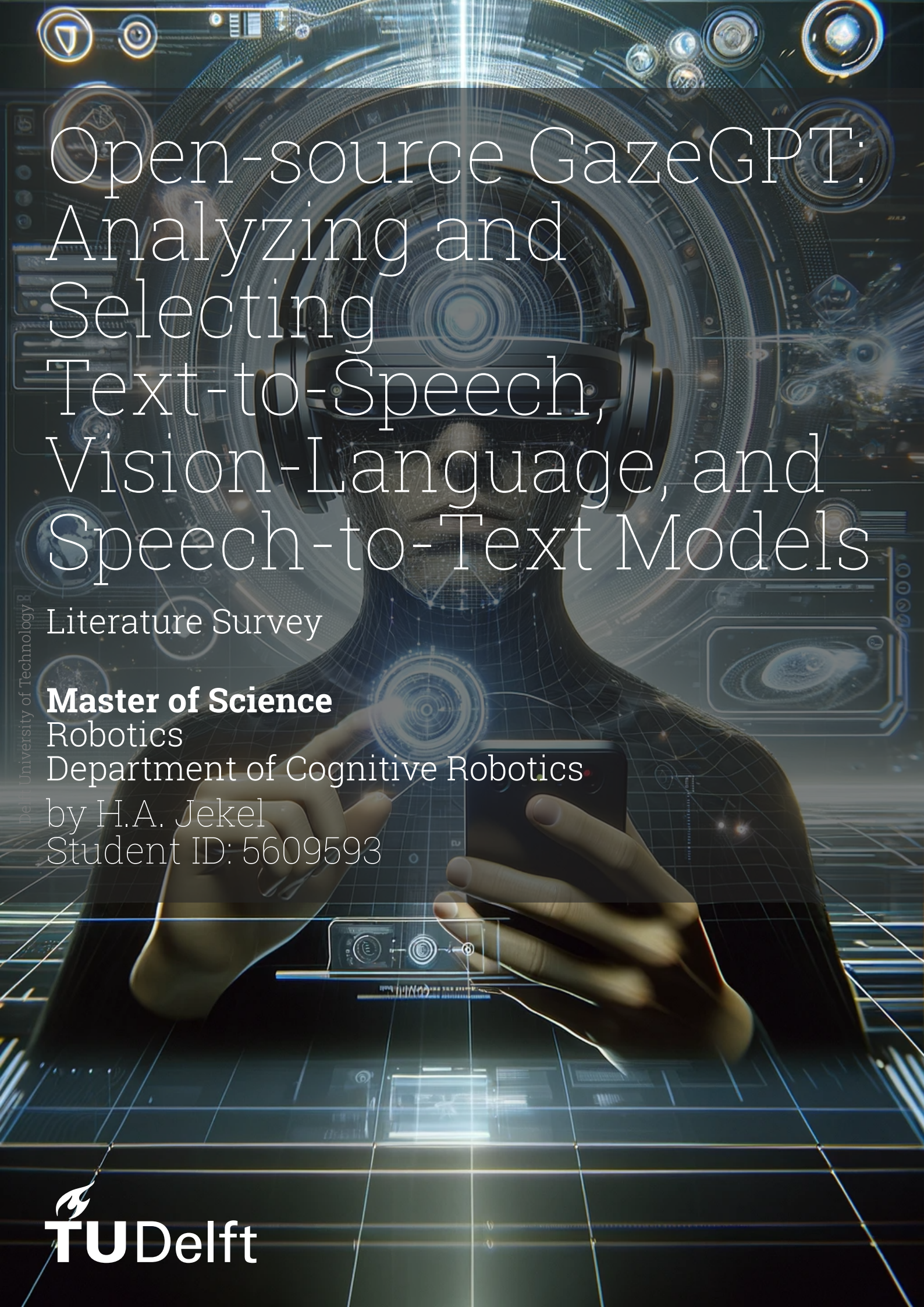

Open-source GazeGPT: Analyzing and Selecting Text-to-Speech, Vision-Language, and Speech-to-Text Models

Literature Survey

Master of Science
Robotics
Department of Cognitive Robotics

by H.A. Jekel
Student ID: 5609593

TU Delft

Open-source GazeGPT: Analyzing and Selecting Text-to-Speech, Vision-Language, and Speech-to-Text Models

Literature Survey

by

by H.A. Jekel
Student ID: 5609593

Student Name	Student Number
Hendrik Adrianus Jekel	5609593

Daily Supervisor: A.D (Alejandro Diaz) Rosales
Primary Supervisor: L. (Luka) Peternel
Project Duration: Nov, 2023 - June, 2024
Faculty: Mechanical Engineering, TUDelft

TU Delft

Abstract

This literature review presents the analysis and selection of text-to-speech, vision-language and speech-to-text models to replace the proprietary models used in GazeGPT. GazeGPT is a system that combines voice-enhanced smart glasses with eye tracking, functioning as a personal assistant aware of the user's gaze within a scene.

By wearing a head-mounted eye tracker, users can direct their gaze towards objects of interest and pose verbal questions. These questions, captured via a microphone in the headphones, are then processed in combination with the gaze data to provide contextual understanding of the user's intent. The scene camera integrated with the eye tracker captures the user's view, pinpointing the area of interest based on the gaze data. This multimodal input, comprising the visual scene and audio queries, is processed by the VLM to deliver precise and context-aware responses directly into the user's ears via speakers integrated into the glasses.

GazeGPT currently relies on proprietary models like ChatGPT for its VLM and Elevenlabs for its text-to-speech functionality. In line with the goal of democratizing AI and enabling free research, this literature review exclusively selects open-source models. Additionally, it aims to deepen the reader's understanding of these models by dissecting them into their fundamental components. The author believes this method is useful due to the rapid emergence of new, often overly complex models that are not always presented in an easily understandable manner. Despite their over-engineering and complexity, these models rely on the same basic principles.

Readers should come away from this literature review with a comprehensive understanding of the history of deep learning, including the progression from untrainable to trainable multi-layer perceptrons, the initial divergence of the field into natural language processing and computer vision, and the subsequent convergence through the transformer architecture. The review also details the author's choices for the best text-to-speech, vision-language, and speech-to-text models, along with their inner workings, including the inner workings of the transformer architecture.

Contents

1

Introduction

The introductory chapter highlights the revolutionary advancements in deep learning, focusing on the transformative impact of the transformer architecture.[1] It discusses various applications, such as speech-to-text (STT), text-to-speech (TTS), and vision-language models for multi-modal response generation. It underscores how instruction tuning of transformer-based large language models enabled the mainstream success of ChatGPT, which initially processed only text but evolved to handle images as a vision-language model in GPT-4V.[2][3] The chapter introduces the concept of GazeGPT, a context-aware smart glasses system integrating eye-tracking technology based on proprietary models.[4] The goal of this literature research is aligned with the philosophy of democratizing AI, aiming to identify the best open-source models to replace the proprietary models (such as GPT-4V) within GazeGPT, making this technology accessible to everyone. Subsequent chapters explore an intuitive understanding of the transformer architecture, identify state-of-the-art open-source components for GazeGPT, and synthesize these insights to address the main research question and propose future research directions.

1.1. Background

The field of Deep Learning has seen transformative advancements, notably with the introduction of the transformer architecture.[1] It has enabled a variety of applications such as speech-to-text (STT), text-to-speech (TTS), and image generation models like DALL-E.[5] A prominent example leveraging this architecture is ChatGPT, a conversational AI that generates responses based on user inputs.[6] Instruction tuning has been crucial in refining these models, making them more effective and practical for real-world applications.[7] This section explores these advancements, their integration into smart assistants, and the emergence of GazeGPT, which combines eye-tracking with smart glasses for enhanced context-aware interactions.

As highlighted, transformers can be used to build various types of models, such as speech-to-text (STT) and text-to-speech (TTS) models. STT models convert audio into text, while TTS models transform written sentences into synthetic human speech. Other examples include models that generate images from text descriptions, such as DALL-E.[5] DALL-E was used to generate the title page image for this literature review.

Perhaps the most mainstream technology that leverages a transformer as its backbone is ChatGPT.[6] ChatGPT, designed for conversational interactions, uses this architecture to comprehend user inputs and generate coherent, helpful responses, a process known as response generation.

GPT stands for Generative Pre-trained Transformer. "Generative" indicates that the model generates new text based on the input it receives. The user provides a text or a question, and GPT generates a response in text form. "Pre-trained" means the model has been initially trained on a different task prior to being trained for response generation. For GPT, this pre-training task involves generating text based on an initial sample provided to the model. Thus, given a text, the pre-trained model can predict the next word. A properly trained model would be able to finish this very sentence as the reader expects. If the model had substituted any of the words in this last sentence with "strawberry," it would be evident

that the model is not well-trained to generate coherent next words.

At first glance, it might not seem obvious that predicting the next word is the same as generating new text. However, once a prediction model has been trained, it can generate longer text by starting with an initial snippet, sampling a word from the probability distribution of the next word, appending the most likely word to the text, and then repeating the process to make new predictions based on the updated text.

The usefulness of such a model is questionable, as it continues writing a snippet of text until it is stopped. Instruction tuning, which is the training performed after pre-training, is what allowed these models to become mainstream.[7][8] Instruction tuning involves fine-tuning the next-word predictor on a labeled dataset of instructional user prompts and useful expected outputs. By learning from these examples, the model can execute user instructions, bridging the gap between its fundamental next-word prediction objective and the practical needs of human users.

Without instruction tuning, a model might respond to a prompt like "How do I bake bread?" with a continuation such as "...is a common question among new bakers." With instruction tuning, the model instead provides a useful response: "To bake bread, start by mixing flour, water, yeast, and salt, then let the dough rise before baking it in the oven."

OpenAI developed InstructGPT by applying instruction tuning to their general-purpose language model, GPT-3.[2][7] InstructGPT was instrumental in popularizing ChatGPT, serving as the core technology for the ChatGPT website and Application Programming Interface (API). This enhanced its ability to understand and respond to user queries. The API enables systems to interact with ChatGPT by allowing programmers to integrate their custom software with the InstructGPT response generator. Instead of visiting the ChatGPT website, developers can use the API to combine their applications with ChatGPT's capabilities, facilitating seamless software integration.

People quickly realized that such a response generator could significantly enhance smart assistants like Amazon Alexa, Apple Siri, Google Assistant, and Microsoft Cortana.[9] By integrating STT and TTS algorithms with ChatGPT through the API, these assistants can handle a wider array of inquiries. Users can ask questions on any topic and receive comprehensive answers, moving beyond the frequent "I don't know that" response familiar to smart assistant users.

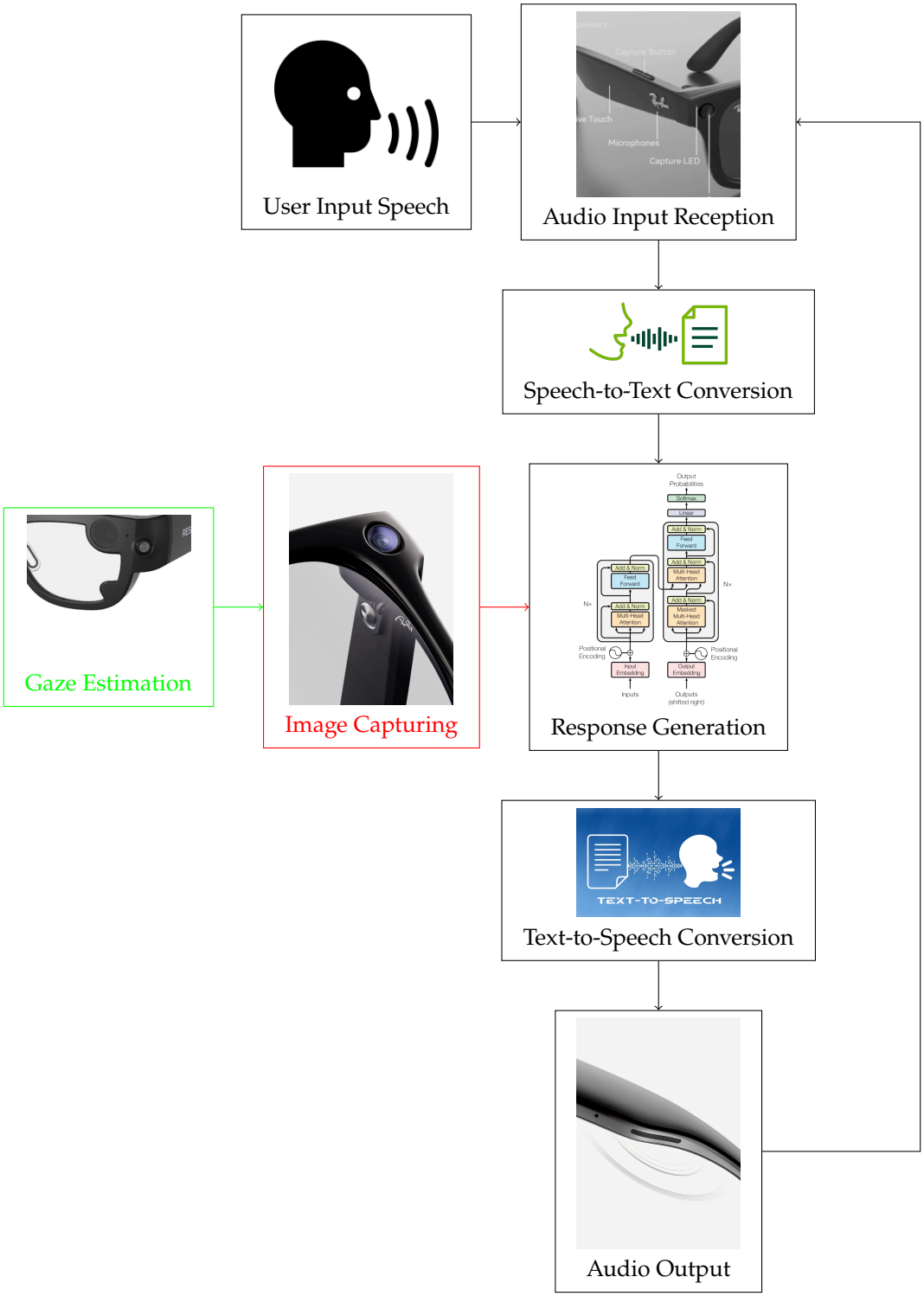

Figure 1.1: Standard smart assistant (black) to smart glasses (red) to GazeGPT's gaze enhanced smart glasses (green)

To demonstrate the interaction flow of the user and this improved smart assistant, observe only the black part of Figure 1.1. The user's spoken input is captured by a microphone, converted to text using a STT algorithm, processed by the response generator, and then converted back to speech using a TTS algorithm to be played to the user.

Recent advancements have enabled ChatGPT to process images and answer related questions.[10] The response generator now receives both text and images. This allows users to capture and upload images, facilitating questions about their immediate surroundings. Continuing with the example of baking bread, the user can now gather all the bread-baking ingredients they have at home, take a picture, and ask, "What types of bread could I bake with these ingredients?"

Alongside these advancements in STT, response generation, and TTS technology, wearable technology, especially smart glasses, has also progressed. Google Glass, introduced in 2013, faced privacy concerns and user acceptance issues.[11] However, the landscape has changed. By 2023, Meta introduced its smart glasses, renewing interest in this technology. Meta's smart glasses, shown in Figure 1.2, indicate a promising future for wearable technology.[12]

Figure 1.2: Meta's ray ban smart glasses[12]

These glasses function as wearable enhanced smart assistants, utilizing the black feedback loop in Figure 1.1 to interact with the user. Additionally, they incorporate the image processing capabilities, allowing users to ask questions about their surroundings using images captured by the glasses' integrated camera, as highlighted in red in Figure 1.1.

Despite their advancements, these smart glasses lack spatial awareness, posing a significant limitation. Users often ask questions related to specific objects or parts of their environment, but the captured images do not provide spatial context, making it difficult for the model to answer accurately. For example, if you face several plants and ask whether a specific one will thrive indoors, the model won't be able to identify which plant you're referring to.[4]

This limitation paved the way for GazeGPT, as detailed in a recent paper.[4] GazeGPT integrates eye-tracking with smart glasses, creating a context-aware system. Figure 1.1 shows gaze estimation (green) as an additional input on top of the captured image (red). Figure 1.3 exemplifies response generation combining gaze estimates, images, and text input. GazeGPT utilizes proprietary models for response generation through GPT-4V and human speech synthesis via Elevenlabs.[3][13] For automatic speech recognition, it employs Whisper, an open-source model from OpenAI's earlier open-source initiatives.[14]

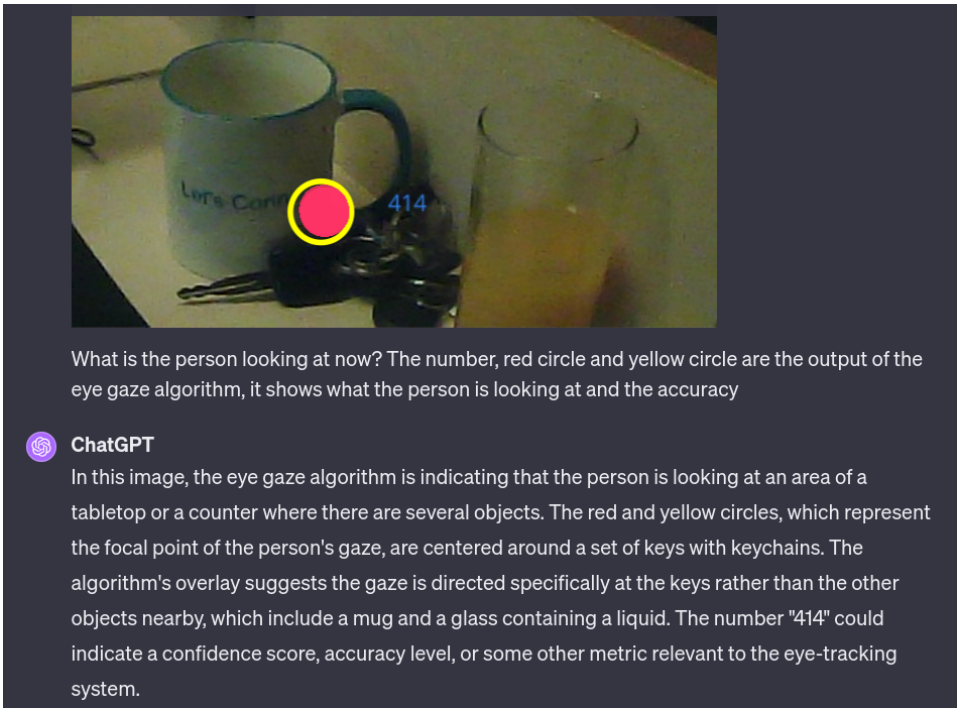

Figure 1.3: An example of response generation based on a deep learning model using input text, image, and gaze estimate.

In summary, this section has provided an overview of the revolutionary advancements in Deep Learning, particularly emphasizing the transformative impact of the transformer architecture. Key applications, such as speech-to-text, text-to-speech, and image generation models, were explored, highlighting their integration into smart assistants and the development of GazeGPT, which enhances context-aware interactions through eye-tracking smart glasses.

Transitioning to the problem definition, the focus shifts to addressing the reliance on proprietary models in GazeGPT. The overarching philosophy of democratizing AI is enforced by identifying the best open-source alternatives for text-to-speech and response generation, as well as finding a superior open-source speech-to-text model.

1.2. Problem Statement

This section introduces the critical issue of relying on proprietary models in the development of GazeGPT[4]. While proprietary models have driven significant advancements, they pose accessibility and transparency challenges that hinder the democratization of AI. This literature review aims to discourage the use of proprietary models, advocating instead for open-source solutions to make AI technologies more accessible, transparent, and inclusive. The goal is to identify the best open-source speech-to-text converters, response generators, and text-to-speech converters to make GazeGPT accessible to everyone. By informing readers about the top algorithms and their workings, this review seeks to demystify these complex technologies and promote their broader adoption.

The study goes beyond comparing performance measures, aiming to thoroughly understand the underlying models. With the rapid pace of AI research and an estimated market size of $1,185.53 billion by 2033 [15], continuous improvement of models is anticipated. Therefore, the author considers the theoretical foundations of these models to be as important, if not more so, than the specific models themselves.

In the paper "Attention is All You Need," which introduces the transformer architecture, the authors often favor precise mathematical formulas over intuitive explanations.[1] To make this material more accessible, this study aims to provide the necessary intuition as a solid foundation for understanding models built upon this architecture.

Once the intuition of the transformer architecture is developed, selecting the appropriate algorithms requires meeting several stringent criteria to ensure real-time, accurate, and human-like speech-to-speech interaction.

The system must operate in real time to provide immediate feedback, maintaining a seamless and natural interaction experience. For instance, if a user asks a question about an object they are looking at, the system needs to respond instantly to maintain the flow of conversation. If the system has noticeable delays, the interaction would feel disjointed and frustrating, reducing user satisfaction and the effectiveness of the assistant.

Additionally, the text-to-speech component must produce speech that is clear, natural, and expressive to avoid sounding robotic or uncanny. When the assistant answers a question, the voice should sound natural and expressive, making the interaction feel more personal and engaging. If the speech sounds robotic or unnatural, it can create an uncomfortable user experience, leading to lower acceptance and trust in the assistant.

Furthermore, the response generation model must deliver accurate, relevant, and engaging responses to user prompts. For example, when a user asks a factual question, the system should provide a precise and relevant answer, enhancing the user's experience and trust in the assistant. If the responses are inaccurate or irrelevant, it could lead to user frustration and decreased reliability, undermining the assistant's utility.

To summarize, the challenge is to offer readers an intuitive understanding of the transformer architecture, identify open-source models for GazeGPT capable of real-time processing with realistic, human-like speech, and generate accurate, engaging responses. This review aims to make these models more accessible by providing readers with the intuition needed to understand them. By enabling everyone to understand these models, the author hopes to move closer to the goal of democratizing AI.

1.3. Research question

The primary research question addressed in this literature report is:

How can open-source speech-to-text, response generation, and text-to-speech models be leveraged to develop a real-time, accessible, and human-like GazeGPT, while providing an intuitive understanding of their underlying architecture to promote the democratization of AI?

To address the main research question, the survey focusses on the following sub-questions:

> **Attention in transformers**:
> *How can the transformer architecture, the backbone of speech-to-text, vision-language, and text-to-speech models, be intuitively understood?*

This first sub-question is answered in Chapter 2. As stated in Section 1.1, these models are all based on the transformer architecture. Therefore, to gain an intuitive understanding of these models, the reader needs to first understand the transformer architecture.

> **Selection of Open-Source Components**:
> *Among the open-source models available, which text-to-speech, vision-language, and speech-to-text models stand out as the best, and what are the unique mechanisms and innovations that drive their performance?*

Chapter 3 answers this second sub-question. The primary research question is subsequently answered in chapter 4 using the findings from chapter 2 and 3.

1.4. Method

To address the primary research question and the sub-question in this literature research, a systematic and rigorous method was employed. The process entailed conducting targeted searches using academic databases, primarily focusing on recent advancements and developments in the relevant fields. The methodology used for this study is outlined below:

Search Strategy and Databases: The primary database utilized for sourcing relevant literature was the Web of Science, due to its comprehensive collection of scientific articles and papers. The search

was tailored to include papers that specifically discuss open-source speech-to-text and text-to-speech technology and vision language models.

Search Query Development: To ensure a thorough exploration of the topic, separate search queries were formulated for each sub-question. These queries were designed to capture a broad spectrum of relevant research, encompassing both foundational theories and cutting-edge developments. Searches were limited to the title, abstract, and keywords fields of the papers to maintain focus on the most pertinent information.

For chapter 2 the time span for the literature was set from 1950 to the present to ensure the inclusion of the historical development of the field of deep learning.

Queries included:

- multi layer perceptrons review
- feed forward neural network review
- natural language processing survey
- computer vision survey
- self attention review
- cross attention review

Chapter 3 explores the models in open-source literature that utilize self-attention mechanisms. Consequently, the timeframe is set from the publication date of the "Attention Is All You Need" paper in 2017 up to the present.[1]

Queries included:

- open-source speech-to-text
- open-source automatic speech recognition
- open-source vision language models
- open-source multi modal models
- open-source text-to-speech
- open-source speech generation

Cited Reference Search and Snowball Sampling: Additionally, a cited reference search was conducted to identify seminal works and foundational papers in the fields of speech-to-text, vision language models and text to speech. Snowball sampling was employed by reviewing the references of key papers, which often led to the discovery of additional relevant literature.

Selection of Papers and Analysis: The initial search yielded a substantial number of papers. These were then meticulously reviewed and narrowed down based on the relevance and contribution to the research questions. The selected papers were critically analyzed to extract insights and data pertinent to the research objectives.

Synthesis and Reporting: The final step involved synthesizing the findings from the selected papers to construct a comprehensive understanding of the current state of the art and to identify gaps in the literature. This synthesis forms the basis of the discussion and conclusions presented in the subsequent sections of the report.

Through this methodical approach, the research aims to provide a detailed and insightful analysis contributing meaningfully to the field of HCI and the advancement of open-source technology integration.

1.5. Report structure

The structure of this report is organized into chapters. The chapters are outlined as follows:

Chapter 2 Deep learning and Transformers: Understanding the transformer architecture is crucial for successfully interpreting the models used in GazeGPT. This chapter explores the transformer architecture

in a storytelling manner, emphasizing intuitive examples rather than heavy mathematical explanations. The focus is on building from basic components in the history of deep learning, synergizing them to create a seamless transition to the transformer architecture. This approach aims to provide the reader with an intuitive understanding, addressing the sub-question on the transformer architecture.

Chapter 3 State-of-the-Art Open-Source Components: This chapter identifies the best open-source models to replace the proprietary variants used in GazeGPT.[4] It covers the latest advancements in open-source speech-to-text, text-to-speech and vision language models. It also gives an intuitive explanation of the model architectures. This investigation aligns with the second sub-question.

Chapter 4 Discussion and future research: The final chapter, serving as the discussion chapter of this literature review, aims to synthesize the findings from Chapters 2 and 3 to answer the main research question. It integrates insights on the transformer architecture and the evaluation of open-source models to provide a comprehensive understanding. This chapter also discusses the implications of these findings, highlight the strengths and limitations of the identified models, and propose directions for future research in this innovative field.

Through this structured approach, the report aims to thoroughly investigate how the open-source models work and which are best, contributing valuable insights and directions for future research in this innovative field.

2

Deep learning and Transformers

This chapter addresses the sub-research question: How can the transformer architecture, the backbone of speech-to-text, vision-language, and text-to-speech models, be intuitively understood? To grasp the workings of the transformer architecture, it is essential first to understand the broader concept of deep learning on an intuitive level. Thus, the chapter embarks on a historical journey through the deep learning landscape from 1958 to the present. It begins with the origins of deep learning in section 2.1, progresses to an explanation of the multi-layer perceptron in section 2.2 and techniques to make networks deeper. Following this foundation, the chapter delves into a detailed and intuitive exploration of the transformer architecture in section 2.3. Chapter 3 will then address the second research question by explaining how the transformer architecture is implemented in the selected speech-to-text, vision-language, and text-to-speech models.

The next section introduces the origins of deep learning, a branch of artificial intelligence characterized by neural networks inspired by the human brain's functioning. The journey begins in 1958 with Frank Rosenblatt's introduction of the perceptron, a foundational concept in neural networks.[16] Despite its initial limitations due to the step-function activation, advancements like the sigmoid function and backpropagation algorithm by Geoffrey Hinton and colleagues in 1986 revolutionized training processes, enabling practical applications such as handwritten digit recognition.[17] This historical overview sets the stage for understanding more complex models like transformers, as these perceptrons are still part of the transformer architecture. Transformers are still based on the same training principles and inference computations as these early perceptrons.[1]

2.1. The origins of deep learning

Deep learning is the branch of artificial intelligence (AI) that has caused the current boom in AI.[18] It is characterized by algorithms known as neural networks, which are inspired by the workings of the brain.[19] The basis for neural networks was set in 1958 with the introduction of the multi-layer perceptron.[16] Figure 2.1 presents a single neuron of the input layer of such a perceptron.[1]

[1]Rosenblatt, who introduced the concept of the multi-layer perceptron, referred to these architectures simply as perceptrons. There is a common misconception in the field of deep learning that a perceptron is merely a single artificial neuron.[20][21] In this text, the terms perceptrons and multi-layer perceptrons are used interchangeably, both referring to an artificial network of neurons.[16]

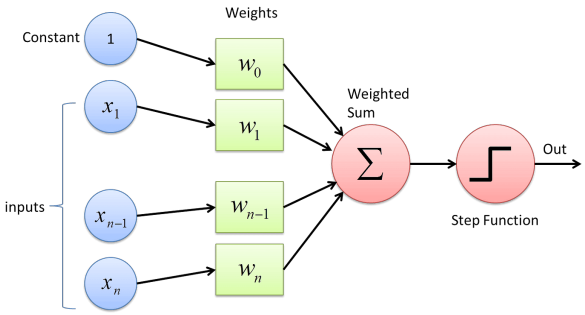

Figure 2.1: A single neuron of the hidden layer of a traditional perceptron[22]

When first observing Figure 2.1, one might initially notice its presentation as a mathematical model. However, instead of immediately diving into deep learning as a purely mathematical concept, it is more intuitive to consider the motivation behind its structure as envisioned by its creator, Dr. Frank Rosenblatt. Dr. Rosenblatt, a psychologist with extensive knowledge of neural functioning in the human brain, introduced the perceptron as an artificial analogue to a neural network found in the brain. In this context, "neural" refers to the neurons in the human brain, as illustrated in Figure 2.2.

The biological neuron illustrated in Figure 2.2 demonstrates key components that align with Rosenblatt's perceptron model. On the left of the image, the dendrites receive input signals from preceding neurons, analogous to the input values $x_1, x_2, \ldots, x_n$ in Figure 2.1. The preceding neurons either fire or do not. When they fire, they release neurotransmitters into the synapse, which are captured by neurotransmitter receptors in the subsequent neuron.[23] The amount of neurotransmitters released and the number of receptors available are analogous to the weights $w_1, w_2, \ldots, w_n$ in Figure 2.1, where a higher quantity signifies a stronger influence of the firing of that preceding neuron. These inputs are then integrated within the neuron cell body (soma), similar to how a perceptron sums the weighted inputs. The axon transmits the action potential, akin to the activation function's output in a perceptron, which, if excThese inputs are then integrated within the neuron cell body (soma), similar to how a perceptron sums the weighted inputs. The axon transmits the action potential, akin to the activation function's output in a perceptron, which, if exceeding a certain threshold, typically around 70mV for a biological neuron and analogous to w_0 in Figure 2.1, propagates the signal.[24].The axon tips (axon terminals) facilitate communication with subsequent neurons through synapses, feeding forward the perceptron's output. This structured flow of information from dendrites to axon terminals mirrors the computational steps in Rosenblatt's perceptron, highlighting the biological basis for artificial neural networks.

Summarizing, Rosenblatt's artificial neurons share several properties with biological neurons: they function as repeating units that behave uniformly, receive information from each other, process this information, and send information to each other.

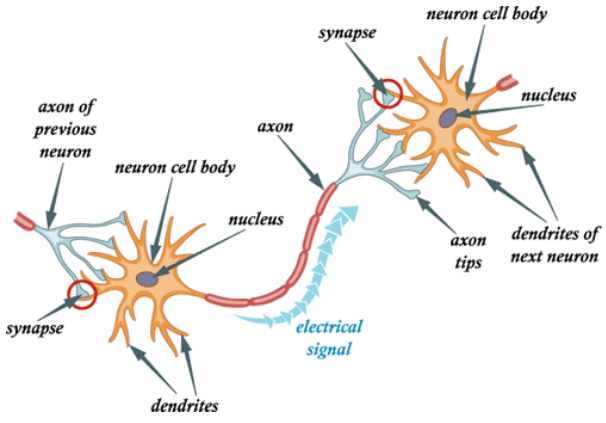

Figure 2.2: A schematic of a neuron in the human brain[25]

To illustrate why Rosenblatt believed such a structure could lead to artificial intelligence, consider the hierarchical structure of a construction project as an analogy. The construction workers represent the input neurons, each performing specific tasks and providing updates on their progress. These workers correspond to the inputs x_1 to x_n in Figure 2.1, and their numeric values indicate how much they feel the need to update the site manager. The site manager, who oversees the workers, assigns weights to each worker. These weights, represented by w_1 to w_n in Figure 2.1, reflect the importance of each worker's update to the overall project. The threshold w_0 represents the communication threshold for the site manager. When this threshold is exceeded, indicating a significant milestone, the step function in Figure 2.1 switches from 0 to 1, prompting the site manager to communicate the milestone to the project manager. Communication between the project manager and the site manager occurs infrequently, typically only when major tasks are completed. This mirrors the way information is processed and passed along in a perceptron as designed by Rosenblatt, with neurons in lower layers (x_1 to x_n) handling detailed tasks and higher layers (the site manager) integrating and passing on more abstract information.

As demonstrated in figure 2.1, Rosenblatt used a step-function as the models activation function. Based on the biological variant, the neuron either fired or not. This resulted in models that were almost impossible to train. Interest in the field of deep learning therefore died as it could not be used to solve any real-life problems.[26]

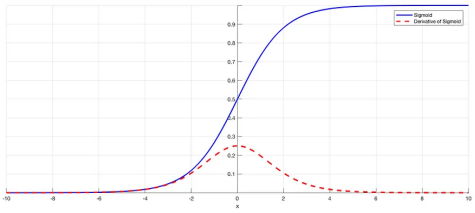

Figure 2.3: The sigmoid activation function and its derivative

In 1986, Geoffrey Hinton at the University of Toronto, along with colleagues David Rumelhart and Ronald Williams, addressed the training problem by publishing the now-famous backpropagation training algorithm.[17] They replaced the step function with the sigmoid function, as elaborated in Section 2.2.1. With the introduction of the sigmoid activation function, the outputs of artificial neurons continuously range between 0 and 1, see figure 2.3. An output close to 0 indicates that the neuron is inactive, while an output close to 1 indicates that the neuron is active. Figure 2.4 presents the updated neuron, replacing the step function in figure 2.1 with the sigmoid activation function.

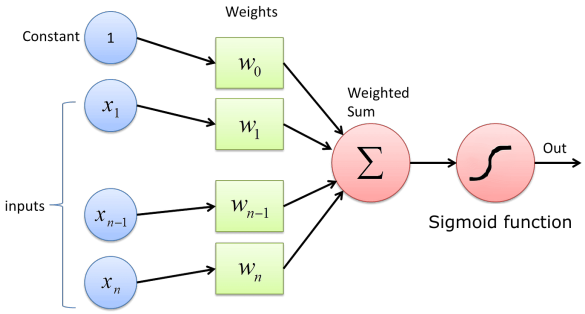

Figure 2.4: The improved neuron with the sigmoid activation function

This modification enabled the backpropagation training algorithm to function effectively, which is also discussed in Section 2.2.1.[17] This allowed these perceptrons to be trained to learn a certain task. Analogous to the human brain, these networks change through the process of learning as to better perform a certain task.[27] The human brain demonstrates remarkable capabilities, such as the ability to recognize handwritten digits (figure 2.5): a task that may appear simple but involves complex cognitive processes. Despite significant variations in handwriting styles across individuals, people can generally

identify digits correctly, even when encountering unfamiliar styles. For instance, imagine being shown 1000 different renditions of the number "5," each written by a different person. It is highly unlikely that these renditions would stimulate the exact same photo-receptor cells in the eye. Yet, the visual cortex of the brain successfully interprets these varied presentations as the same numerical concept. This skill underscores the brain's extraordinary ability to generalize and interpret similar items as representations of the same idea.[23]

Figure 2.5: A set of handwritten digits.[28]

The perceptrons or artificial neural networks mimic this capability of natural neural networks by learning to generalize from the data they are trained on. They achieve this by discovering a mathematical function, embodied by the entire network of trained neurons, which maps the input image of a handwritten digit to a prediction of a number.[29] When effectively trained, an artificial neural network can recognize not only the handwriting styles it has seen during training but also adapt to new, unseen styles. Thus, much like the human brain, a well-trained artificial neural network possesses the ability to generalize across variations, interpreting different instances of handwritten digit images as representing the same number. In the field of deep learning this is referred to as a generalized neural network: a network that classifies images both within and beyond its training dataset with similar accuracy. Conversely, if a neural network only accurately classifies digits it has seen during training and struggles with new images, it is considered to be overfitting the training data. Overfitting occurs when a network is excessively tuned to the training data, to the extent that it fails to generalize to new, unseen data.[30] The goal in deep learning is to create generalized neural networks through training that robustly perform useful tasks both within a training set and on unseen data outside the training set.

Going back to the example of handwritten digit recognition, an artificial neural network with a multi-layer perceptron architecture could be trained to recognize handwritten digits generalizing to digits that it has not seen before. Handwritten digit recognition significantly advanced automated mail sorting by accurately recognizing postal zip codes in the 80s. [29] This innovation enabled faster, more reliable mail processing, reduced human error, and lowered operational costs due to automation, highlighting the practical impact of neural networks in real-world applications.

This section traced the origins of deep learning, emphasizing the development and evolution of neural networks. Starting with Rosenblatt's introduction of the multi-layer perceptron in 1958, it detailed how biological neural mechanisms inspired artificial neurons.[16] The shift from the step function to the sigmoid function by Hinton and colleagues in 1986 addressed early training difficulties, enabling effective backpropagation and facilitating tasks such as handwritten digit recognition.[17] The transition to generalization and avoiding overfitting in neural networks was also discussed.[30] The next section delves deeper into the multi-layer perceptron, detailing the forward pass and training processes critical to understanding advanced architectures like transformers.

2.2. Multi-layer perceptron

Building on the foundational concepts of deep learning covered in the previous section, this section provides an intuitive explanation of the multi-layer perceptron (MLP). In subsection 2.2.1, the forward pass process is examined to demonstrate how an input image of a handwritten digit is processed and classified as a number between 0 and 9. Understanding the multi-layer perceptron is crucial for grasping the more advanced transformer architecture discussed in Section 2.3, as MLPs form a building block of transformers. Additionally, subsection 2.2.2 delves into the training methods of these perceptrons, offering insights that also apply to transformers.

2.2.1. Handwritten digit recognition using neural networks: forward pass

This section furthers familiarization with deep learning by exploring the forward pass in a multi-layer perceptron, using handwritten digit recognition as an example. The goal is to build intuition for this model architecture, which is a building block of the more advanced transformer architecture.[1]

Handwritten digit recognition is often referred to as the "Hello World" of artificial neural networks, serving as a common introductory problem for newcomers to the field.[31] Even Hinton's trainable multi-layer perceptrons from the 1980s, discussed in section 2.1, were capable of solving this task.[17] Section 2.1 also illustrated the appearance of a neuron in the first hidden layer of such a network, as shown in figure 2.4. However, the hidden layer of a multi-layer perceptron consists of an entire column vector of such neurons, as depicted in figure 2.7. These multi-layer perceptrons feature a layered structure comprising column vectors of neurons for each layer, starting with the input layer, progressing through an arbitrary number of hidden layers, and ending with the output layer. Figure 2.7 presents a variant with a single hidden layer.

In the specific context of handwritten digit recognition, preprocessing the image of the handwritten digit is a crucial first step.[31] This begins with transforming the grayscale image of the digit into a column vector, see figure 2.6. The transformation process involves concatenating all the pixel values from the rows of the image and then transposing these into a column vector. This vector, which contains grayscale values ranging from 0 to 255, is then normalized to range between 0 and 1 to enhance training stability.[31] This normalized column vector, which houses the input neurons, constitutes the input layer of the neural network (most left column of neurons in figure 2.7).[28]

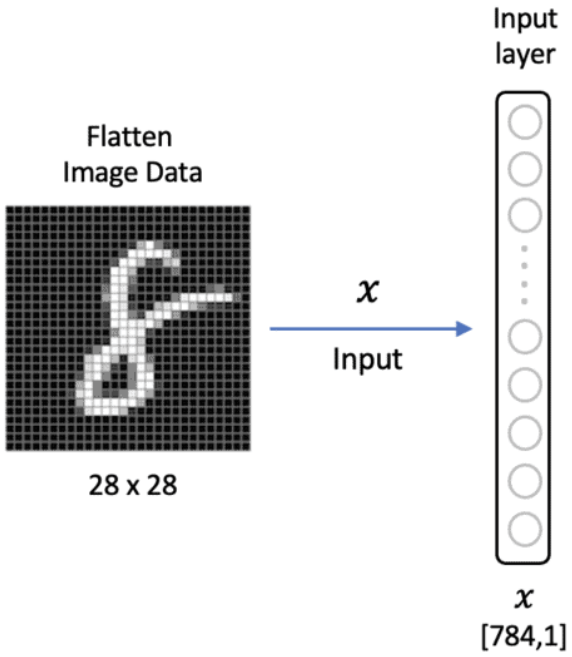

Figure 2.6: Flattening a handwritten digit to fit the input layer of a multi-layer perceptron

Subsequent layers are interconnected such that each neuron in a given layer is linked to every neuron in the immediately preceding layer, just like the single neuron in figure 2.4, see figure 2.7. Looking back at figure 2.4, the process of moving from one layer to the next involves scaling each neuron's output from the previous layer by a neuron-specific weight, summing these scaled values, adding a bias, and then applying a sigmoid activation function such that each neuron represents a value between 0 and 1. As explained in section 2.1, the addition of a bias adjusts the network's threshold for neuron activation, akin to how biological neurons require a net potential above a certain threshold (about 70mV) to activate.[24]

From a linear algebra perspective, the computation of the column vector of neurons for the next layer can be described as a linear transformation, where the input layer is transformed using the weight matrix associated with the previous layer's neuron vector. As presented in figure 2.4 each neuron in the hidden layer has its own weight vector, hence going from the input layer to the hidden layer involves a transformation with a weight matrix where each neuron has its own weight vector in this weight matrix.

To visualize this transformation, think of the 784-dimensional input column vector from Figure 2.6 as a vector in a 784-dimensional space. Although imagining such a high-dimensional space is impossible, an analogy can be drawn to a 3-dimensional space, which is more familiar. Picture this vector as an arrow with a tail and a tip, starting at one point and ending at another within that 784-dimensional space.

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,783} & w_{1,784} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,783} & w_{2,784} \\ w_{3,1} & w_{3,2} & \cdots & w_{3,783} & w_{3,784} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{14,1} & w_{14,2} & \cdots & w_{14,783} & w_{14,784} \\ w_{15,1} & w_{15,2} & \cdots & w_{15,783} & w_{15,784} \end{pmatrix} \tag{2.1}$$

This 784-dimensional vector is now transformed by the above weight matrix W. This weight transformation matrix has only 15 rows, which means that the 784-dimensional space in which the handwritten digit image vector lies is reduced to a lower dimensional space. In the case of a full rank weight matrix,

this results in a transformation with rank 15, mapping the original vector into a 15-dimensional subspace. This is analogous to how basis vectors that initially spanned a 2-dimensional space can collapse to a 1-dimensional space after a transformation that maps every vector within the 2-dimensional space onto a line.[32]

Adding the vector of biases, one for each neuron in the hidden layer, modifies this linear transformation into an affine transformation by translating the resultant vector. Although not common in deep learning implementations, this affine transformation can be conceptually represented using a homogeneous transformation in academic papers for theoretical simplicity.[33] To achieve this, the input vector $\mathbf{x}$ can be augmented with an additional dimension set to 1, resulting in equation 2.2. Correspondingly, an augmented weight matrix W' can be constructed that incorporates the bias vector $\mathbf{b}$ as represented in equation 2.3. The affine transformation can then be expressed using equation 2.4. This homogeneous transformation effectively combines the weight matrix and bias vector into a single matrix operation, simplifying the theoretical formulation of the perceptron layer. The output from this affine transformation, weather presented intuitively with a seprate weight matrix and bias vector or elegantly using a homogeneous transform then passes through a sigmoid function to form the neuron vector of the hidden layer.

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{2.2}$$

$$W' = \begin{bmatrix} W & \mathbf{b} \\ 0 & 1 \end{bmatrix} \tag{2.3}$$

$$\mathbf{y}' = W'\mathbf{x}' = \begin{bmatrix} W & \mathbf{b} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{2.4}$$

The sigmoid function introduces non-linearity, mapping any input number to a value between 0 and 1, with extreme inputs nearing the bounds and moderate ones smoothly transitioning across this range. This non-linearity is crucial, as without it, the neural network would be limited to learning only linear relationships, substantially narrowing the scope of functions it can model.[34] Viewing the neural network through this lens of linear algebra highlights the sequential transformations: a linear transformation followed by translation and a non-linear modification. In the multi-layer perceptron, each layer or column of neurons is computed in this manner, with the exception of the first layer. As described earlier, the first layer contains the input data, which, in the example of handwritten digit recognition, is the pre-processed image of the handwritten digit.[32] The multi-layer perceptron in figure 2.7 only contains a single hidden layer. However, perceptrons can have more than one hidden layers depending on the complexity of the task.

Moving to the end of the network, the last layer, known as the output layer, features a column with 10 neurons. Each neuron corresponds to one of the digits from 0 to 9. The activation of these neurons, scaled between 0 and 1, reflects the network's assessment of how likely it is that the input image matches each digit.

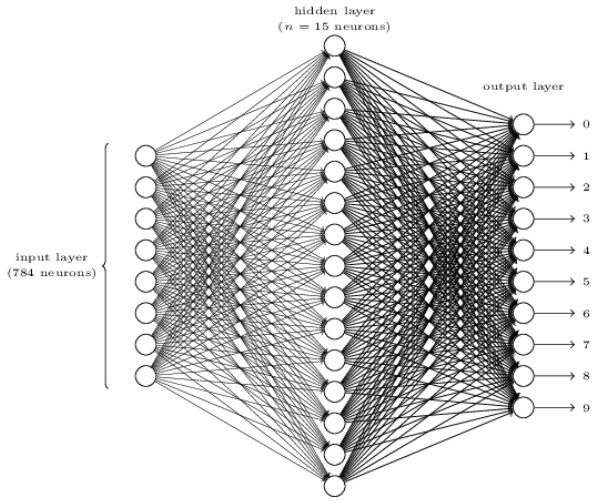

Figure 2.7: A multi-layer perceptron. [28]

To formalize this, consider a multi-layer perceptron with L layers. The input to the network is denoted by x, and the homogeneous weight matrix that includes both weights and biases for layer l is $W^{(l)}$, conform equation 2.4.

For the forward pass, the output of each layer can be written as:

1. First layer:
$$z_1 = \sigma(W'^{(1)}x) \tag{2.5}$$

2. Second layer:
$$z_2 = \sigma(W'^{(2)}z_1) = \sigma(W'^{(2)}\sigma(W'^{(1)}x)) \tag{2.6}$$

3. Third layer:
$$z_3 = \sigma(W'^{(3)}z_2) = \sigma(W'^{(3)}\sigma(W'^{(2)}\sigma(W'^{(1)}x))) \tag{2.7}$$

To get the output of the whole network, the input vector is repeatedly transformed using the homogeneous transform matrix $W^{(l)}$ and the sigmoid activation function at each layer. The output of the multi-layer perceptron with sigmoid activations is presented in equation 2.8:

$$\hat{y} = \sigma(W'^{(L)}\sigma(W'^{(L-1)} \cdots \sigma(W'^{(2)}\sigma(W'^{(1)}x)) \cdots)) \tag{2.8}$$

Instead of relying on handwriting experts to manually design image filters to classify each handwritten digit, the goal is for the neural network to autonomously develop weight matrices and bias vectors across its layers that are effective for digit recognition. The aspiration is that through training, the network will discern the relationship between specific combinations of these features in the input image and the corresponding digits from 0 to 9. For instance, it is hoped that for the digit "7," the network learns to identify it through the pattern of its constituent lines: two horizontal and one diagonal. When the network detects this particular arrangement of lines, it should recognize it as the number seven. The expectation is that the initial layers learn to detect fragments of lines (figure 2.8), subsequent layers learn to integrate these fragments into more coherent line structures, and the final output layer learns to associate these structures with the respective digits. This method of decomposing complex entities into simpler, understandable components mirrors everyday problem-solving strategies, where complex concepts are broken down into their basic elements and reassembled to understand the larger picture. For instance, calculating 3928 multiplied by 58219 would be overwhelming without breaking it down into smaller, more manageable multiplications.

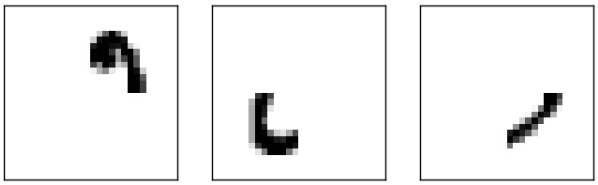

Figure 2.8: Line fragments of handwritten digits.[28]

In summary, this example illustrates a straightforward process where an image is converted into a normalized column vector. For each subsequent layer, this vector is linearly transformed with a weight matrix, translated using a bias vector, and processed through a sigmoid non-linearity until the output layer is reached. This structured approach enables the neural network to perform the complex task of recognizing handwritten digits, showcasing the power of a seemingly simple yet effective architectural design in neural networks.

Building on this understanding of neural network architecture, the next section delves into the learning process involved in training these networks, specifically focusing on how they can effectively recognize handwritten digits through iterative adjustments of weights and biases.

2.2.2. Handwritten digit recognition using neural networks: Learning in neural networks

During the training process, the computer seeks to determine the optimal values for the weights and biases, enabling the neural network to effectively solve the task at hand. Continuing with the example of handwritten digit recognition, the network is trained on a dataset that consists of images of handwritten digits, with each image accompanied by a corresponding label. Human annotators assigned these labels to categorize each image as a single digit, specifically a number between 0 and 9. The Modified National Institute of Standards and Technology (MNIST) dataset serves as a prime example of this type of training data.[35]

To train the network using the MNIST dataset, each image is processed through the untrained neural network. The forward pass, extensively explained in section 2.2.1, is applied to all the images in the dataset. For each image, the output layer neuron with the highest activation or numerical value indicates the network's prediction for that image, see figure 2.7. Initially, the weight matrices and bias vectors for the layers of the perceptron are set randomly, resulting in arbitrary classifications that generally do not align with the correct labels.[28] However, the goal of training is to adjust the weights and biases so that the network can accurately classify handwritten digits. This refinement is achieved through iterative adjustments based on the network's performance during training, striving for a model that generalizes well beyond the training samples.

Describing the training of a neural network as a learning process is somewhat anthropomorphic.[36] Unlike the complex ways in which neural networks learn in the human brain, an artificial neural network is adjusted through a series of mathematical calculations. To quantify the accuracy of the neural network, mathematicians have developed what is known as a loss function. This function measures the discrepancy between the network's predictions and the actual labels of the training data. During training, the network performs the forward pass on each digit in the training set, evaluates the prediction error for each instance using the loss function, and calculates the average loss across all examples to determine the mean error of the network on the training examples.[17]

This loss function lives in the loss landscape. Ideally, the loss function would be convex, simplifying the optimization process to finding the weights and biases that would cause this function to be at its minimum.[17] However, in perceptrons with one or more hidden layers, the loss function landscape is typically riddled with many local minima, similar to the multiple valleys in the Alps, with only one being the deepest.[37] The optimal performance of the network on the training data is achieved at the absolute minimum of this loss function. However, due to the characteristics of the optimization algorithm used, they typically converge to a local minimum, see figure 2.9. Ongoing research continues to explore the behavior of these local minima compared to the global minimum. In practical applications, a local minimum that is sufficiently low often provides adequate model performance and is therefore considered satisfactory.[38]

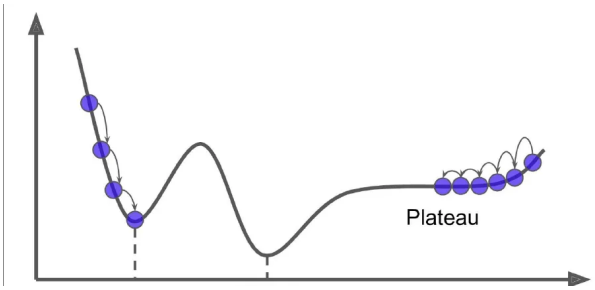

Figure 2.9: Two dimensional representation of the local minimum pitfall in loss function optimization[39]

To keep things straightforward, in the example of handwritten digit recognition, the multi-layer perceptron depicted in Figure 2.7 employs the sigmoid activation function for every neuron, including those in the output layer. Each output neuron operates independently, providing a probability for the presence of each digit class. The appropriate cost function for this setup, given the sigmoid activations in the output layer, is the binary cross-entropy (BCE) loss, see equation 2.9.

$$\text{BCE}(y, \hat{y}) = -\sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{2.9}$$

In the BCE loss, N represents the number of output neurons, which is 10 in our case for the digits 0 through 9. y_i denotes the true binary label (0 or 1), and $\hat{y}_i$ indicates the predicted probability of the respective class being present. This loss function effectively measures the dissimilarity between the predicted probabilities and the actual labels, thereby guiding the network's learning process through gradient descent.

For readers familiar with deep learning, it is worth noting that a softmax activation function with a cross-entropy loss would be more appropriate for mutually exclusive classification problems, such as handwritten digit recognition, where each image corresponds to only one digit. However, the use of softmax and cross-entropy is more complex and will be addressed in section 2.3.13. For now, we use the Binary Cross-Entropy Loss, and the local minimum of the resulting loss landscape must be found using an optimization algorithm called gradient descent.[40]

To shed light on the optimization algorithm, imagine finding yourself at a random spot uphill in the Alps and needing to locate your hotel, which is situated at the bottom of the nearest valley. The quickest way to reach your hotel would be to walk downhill in the steepest direction. This is where the calculus technique known as gradient descent comes into play, identifying the steepest descent path at each step to update the weights and biases.[29]

To draw on this analogy, initially, the untrained network conducts a forward pass using all the handwritten digits in the training set. The optimization algorithm calculates the average loss from all these examples, positioning the network with its weights and biases at a metaphorical random location in the Alps. It then determines the steepest downhill path to move a step closer to the hotel at the local minimum, adjusting the weights and biases accordingly. With each iteration, the network performs another forward pass over the training set to calculate the steepest descent path for the next step. In calculus, this iterative process of minimizing the loss function with respect to the weights and biases by continuously computing the gradient of the loss function is called gradient descent (Figure 2.10). Each step refines the network's parameters, gradually improving its accuracy and efficiency in predicting the right number for the image.[39]

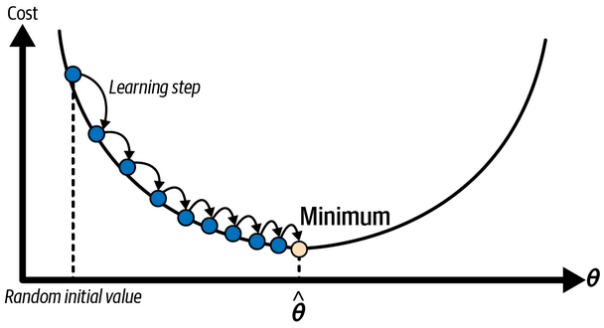

Figure 2.10: The process of gradient descent optimization, one step at a time closer to the local minimum.[39]

Due to the layered architecture of the neural network, where each layer is defined by distinct weights and biases, determining the steepest descent direction during training is not as simple as visually identifying the steepest path down a mountain. Rather than relying on direct observation, this process requires calculating the gradient, $\nabla_w \text{BCE}$.[28]

To calculate the gradient, consider the analogy of descending down a valley, now applied to a single neuron, such as the one depicted in Figure 2.4. In this scenario, computing the gradient downhill is straightforward. It involves calculating the partial derivatives of the BCE loss function (equation 2.9) with respect to the weights of the single neuron, see equation 2.10. In this equation $\hat{y}$ is the single

neurons output, z is the activation function and w_j is one of the neurons weights. Because this example considers a single neuron N is equal to 1 for the BCE from equation 2.9.

$$\frac{\partial BCE}{\partial w_j} = \frac{\partial BCE}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j} \tag{2.10}$$

Computing all partial derivatives of the BCE with respect to the weights w_j as per equation 2.10 and combining them as per equation 2.11 yields the gradient of the BCE with respect to the weights. This gradient represents the direction of the steepest ascent in the high-dimensional BCE loss landscape. Consequently, to minimize the loss, the weights are adjusted in the exact opposite direction of this gradient, effectively taking a step downhill.

$$\nabla_w \text{BCE} = \frac{\partial \text{BCE}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \text{BCE}}{\partial w_1} & \frac{\partial \text{BCE}}{\partial w_2} & \cdots & \frac{\partial \text{BCE}}{\partial w_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial \text{BCE}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_1} & \frac{\partial \text{BCE}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_2} & \cdots & \frac{\partial \text{BCE}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j} \end{bmatrix} \tag{2.11}$$

Sofar, this story build up to the multi-layer perceptron in figure 2.7 starting from a single neuron in figure 2.4. However, technically, this is not deep learning yet. Deep in deep learning refers to a deep neural network. The depth of a neural network refers to how many layers the network has. Looking at figure 2.7 it counts 3 layers, with only 1 hidden layer. The more hidden layers the network has the deeper the network is. However, there is a problem that comes with increasing depth.

The problem with increasing depth is referred to as the vanishing gradient. This issue was formally identified by Sepp Hochreiter in his 1991 diploma thesis, where he analyzed the difficulties in training deep neural networks.[41] During the backpropagation process, the gradients of the loss function with respect to the weights can either shrink (vanish) exponentially as they propagate through many layers.

To illustrate this, the intuition gained from calculating the partial derivatives for a single neuron, as shown in Equation 2.10, can be extended to understand the slightly more complex partial derivatives for multi-layer perceptrons. During backpropagation for a multi-layer perceptron, the gradient of the loss function with respect to each weight is calculated by applying the chain rule. To understand why, refer back to the forward pass as displayed in equation 2.8. That equation highlights that each layer transforms the network input x using the homogeneous weight matrix and a sigmoid activation function. A small change in $W^{(1)}$ causes a small change in the output of layer l_1, which in turn results in a small change in the output of l_2, and so on, all the way to l_L, ultimately resulting in a small change in the network output. With this intuition, for a network with sigmoid activations, the gradient for a weight w in layer l is computed as shown in equation 2.12.

$$\frac{\partial \mathcal{BCE}}{\partial w^{(l)}} = \frac{\partial \mathcal{BCE}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(l+1)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}} \tag{2.12}$$

Here, $z^{(l)}$ denotes the input to the sigmoid function at layer l. Each term $\frac{\partial z^{(l+1)}}{\partial z^{(l)}}$ includes the derivative of the sigmoid function at that layer according to equation 2.13

$$\frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \sigma'(z^{(l)}) \cdot W'^{(l+1)} \tag{2.13}$$

The derivative of the sigmoid function, $\sigma'(z)$, is given by equation 2.14.

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{2.14}$$

The derivative of the sigmoid function, $\sigma'(z)$, is always between 0 and 0.25, see figure 2.3. Consequently, as the gradients are propagated back through the layers according to the chain rule, the repeated multiplication of terms that are at most 0.25 leads to a significant reduction in gradient values. This phenomenon is known as the vanishing gradient problem. Substituting equation 2.13 into equation 2.12

highlights the repetition of $\sigma'(z)$, see equation 2.15. Furthermore, since the initial weights, referring to $W'^{(k+1)}$ in equation 2.15, are typically small and close to zero, they do not sufficiently compensate for the small derivative values of the sigmoid, leaving earlier layers untrainable.[41]

$$\frac{\partial \mathcal{BCE}}{\partial w^{(l)}} = \frac{\partial \mathcal{BCE}}{\partial z^{(L)}} \cdot \left(\prod_{k=l}^{L-1} \sigma'(z^{(k)}) \cdot W'^{(k+1)} \right) \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}} \tag{2.15}$$

To address the vanishing gradient problem, residual connections can be introduced into the architecture of a multi-layer perceptron, as shown in Figure 2.11. Residual connections, first proposed by He et al. in their seminal work on Residual Networks (ResNets), have significantly eased the training of very deep networks.[42] In Figure 2.11, the skip connection adds the input of a layer to its usual output. For intuition, consider the preprocessed handwritten digit from Figure 2.6. After undergoing an affine transformation and the sigmoid activation as per equation 2.5, the untransformed image vector is added to the transformed version. Generalizing this to an arbitrary layer in the multi-layer perceptron yields equation 2.18. Attentive readers will note that adding the original 784-dimensional image vector to the output of the 15-dimensional hidden layer in Figure 2.7 causes a dimensionality mismatch. To prevent this, researchers either match the number of neurons in the hidden layers or transform the input using the homogeneous transform from equation 2.3. In this text, we assume matching neuron numbers, but for the curious reader, the alternative using the homogeneous transform is presented in equation 2.16.

$$z^{(l+1)} = \sigma(W'^{(l)} \cdot z^{(l)}) + W'^{(l)} \cdot z^{(l)} \tag{2.16}$$

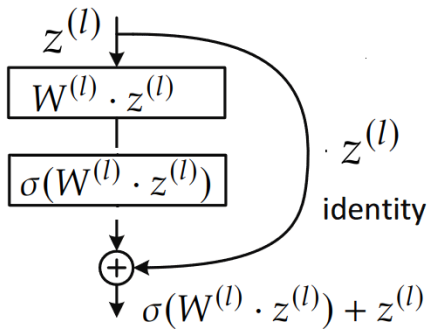

Figure 2.11: Reading from top to bottom, the diagram illustrates a skip connection in which the input to the multi-layer perceptron layer, $z^{(l)}$, is added to the output of the same layer. $W^(l)$ represents the homogeneous transform matrix that includes both weights and biases for that layer.

With skip connections, each layer's output is a combination of its input and the activation of the previous layer. For a given layer l, the output is defined according to equation 2.17. This modification affects the gradient calculation during backpropagation, as the partial derivative from equation 2.13 now includes an additional term due to the skip connection, see equation 2.18.

$$z^{(l+1)} = \sigma(W'^{(l)} \cdot z^{(l)}) + z^{(l)} \tag{2.17}$$

$$\frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \sigma'(W'^{(l)} \cdot z^{(l)}) \cdot W'^{(l)} + 1 \tag{2.18}$$

The presence of the constant term 1 in the derivative changes the dynamics of the gradient flow. Referring to equation 2.12, the product of partial derivatives across many layers typically causes the gradient to diminish, as discussed. However, with skip connections, each term in the product $\frac{\partial z^{(l+1)}}{\partial z^{(l)}}$ includes the constant 1, which mitigates the exponential decay of the gradient, as presented in equation 2.19.

$$\frac{\partial \mathcal{BCE}}{\partial w^{(l)}} = \frac{\partial \mathcal{BCE}}{\partial z^{(L)}} \cdot \prod_{k=l}^{L-1} \left(1 + \sigma'(W'^{(k)} \cdot z^{(k)}) \cdot W'^{(k)}\right) \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}} \tag{2.19}$$

The term $\sigma'(W^{(l)} \cdot z^{(l)}) \cdot W^{(l)}$ can be positive or negative, depending on $W^{(l)}$. Given that $\sigma'(z)$ is always between 0 and 0.25, this product is generally small, especially since weights are usually initialized small. If $W^{(l)}$ is positive, $1 + \sigma'(W^{(l)} \cdot z^{(l)}) \cdot W^{(l)}$ will be slightly greater than 1. If $W^{(l)}$ is negative but small, the term will be slightly less than 1, still close enough to 1 to prevent rapid gradient diminishment.

Another way of reducing the vanishing gradient problem involves the substitution of the sigmoid non-linearity with a different activation function called the rectified linear unit (ReLU), see figure 2.12.[43] The ReLU activation function, introduced by Nair and Hinton in 2010, is defined as $\text{ReLU}(x) = \max(0, x)$. Unlike the sigmoid function, the ReLU function maintains a gradient of 1 for positive inputs, thereby allowing gradients to propagate more effectively through the network.

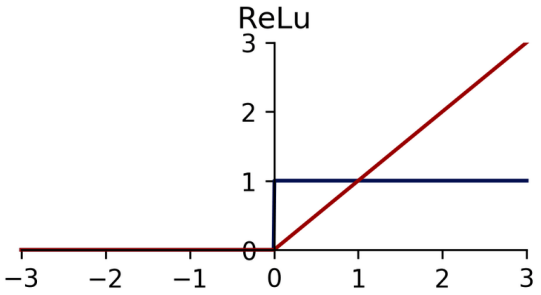

Figure 2.12: ReLU activation (red) and derivative (blue) for efficient gradient computation

As equation 2.15 highlights, the gradient contains the layer activations $z^{(k)}$ of the layers $k = l$ to $k = L - 1$. These are already computed during the forward pass of the network according to 2.5, 2.6 and 2.7, as these are the outputs of each layer. To save computation, people in deep learning use the backpropagation algorithm to update the weights and biases efficiently. This is done by reusing the activations from the forward pass. It also computes gradients from later layers first and then caches the left part of equation 2.15 to be reused in the calculation of gradients in earlier layers back propagating the derivatives of the loss function through the network. This avoids redundant calculations and saves computational resources. This highlights the increased complexity of finding the minimum of the loss landscape with an increasing amount of layers.[29]

As previously outlined, updates to the weights and biases are informed by the loss function, which reflects the average loss across all training examples in the dataset. However, executing a forward pass for every training example to compute this average loss each time the weights and biases need adjustment is computationally intensive.[39] To address this, the field commonly employs a technique known as stochastic gradient descent (SGD).[44] The term "stochastic" implies the introduction of randomness and an element of non-determinism in the training process. In SGD, this randomness is analogous to taking an approximate path down the steepest slope of a valley without precisely measuring each step with advanced tools. Similarly, the neural network updates are based on a random subset of the training examples, not the entire dataset. While each step taken using SGD might not exactly align with the steepest descent, the significant reduction in computational demand makes this method preferred. Although the direction of each step might not be perfectly accurate, the small size of the steps ensures that the network will eventually converge to the minimum. This process is akin to gradually descending to the bottom of a valley by foot, approximately following the direction of steepest descent. By adjusting the step direction to be approximately downhill at each step, you will, on average, follow the steepest descent path. [45]

To mitigate the zigzagging trajectory observed during the weight updates in gradient descent, an exponentially weighted moving average (EWMA) is employed. EWMA smooths out fluctuations in

data, which is particularly useful in managing the noise inherent in the gradient measurements. This method is recursive, building on the results from previous iterations to refine current updates. The EWMA formula is:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t \tag{2.20}$$

Here, v_t represents the EWMA of the gradient at time step t, β is the smoothing factor (typically close to 1), v_{t-1} is the EWMA from the previous time step, and g_t is the current gradient.

In the context of deep learning, this application of EWMA to the weight updates is known as momentum. Momentum helps to smooth the path of gradient descent by incorporating past gradients into current updates. Essentially, if the gradient direction alternates, shifting left and right down a slope, using momentum helps to average these oscillations, guiding a more direct route downward. Besides mitigating the zigzagging trajectory, this technique also proves beneficial when stochastic gradient descent (SGD) overshoots the minimum due to an excessively high learning rate, see figure 2.13. The learning rate is analogous to the step size one takes on the path downhill. Imagine being a giant in the Alps; due to your exceptionally large strides, you may step over the local minimum, finding yourself uphill on one side of the valley in one step, and uphill on the opposite side in the next. This scenario is akin to jumping from one side of a skateboard half-pipe to the other without ever touching the lowest point.

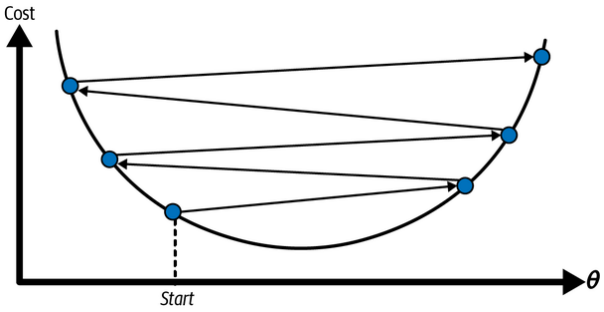

Figure 2.13: The learning rate is too large[39]

With the handwritten digits, it was already mentioned that they were pre-processed using normalization to have pixel values ranging between 0 and 1 instead of 0 and 255, which improves training stability by ensuring the inputs fall within a standard range of [0, 1] [31]. People in the field have asked themselves, why not normalize the output of every layer in the neural network? The preprocessing step of the input, so to say, normalized the output of the input layer. Researchers have applied the same idea to the activations of subsequent layers in the network, significantly improving training performance. They have experimented with different types of normalization, all showing substantial improvements in training performance [46] [47] [48]. The exact reasons why normalization enhances training so effectively are still debated [49] [50]. Nonetheless, it is common practice to use normalization techniques, and one specific method, layer normalization, is highlighted to provide an intuition for the process. Layer normalization, which is technically a form of standardization, is applied after each layer. It standardizes the outputs of a layer to have a mean of zero and a standard deviation of one across the outputs of the layer's neurons for each individual training example, see Figure 2.14. Equations 2.21, 2.22, 2.23, and 2.24 highlight the standardization process applied to the output of each layer before it is used as input to the next layer, resulting in layer normalization.

$$\mu = \frac{1}{H} \sum_{i=1}^{H} z_i^{(l)} \tag{2.21}$$

$$\sigma = \sqrt{\frac{1}{H} \sum_{i=1}^{H} \left(z_i^{(l)} - \mu \right)^2} \tag{2.22}$$

$$\hat{z}^{(l)} = \frac{z^{(l)} - \mu}{\sigma} \tag{2.23}$$

$$z^{(l+1)} = \sigma(W'^{(l)} \cdot \hat{z}^{(l)}) \tag{2.24}$$

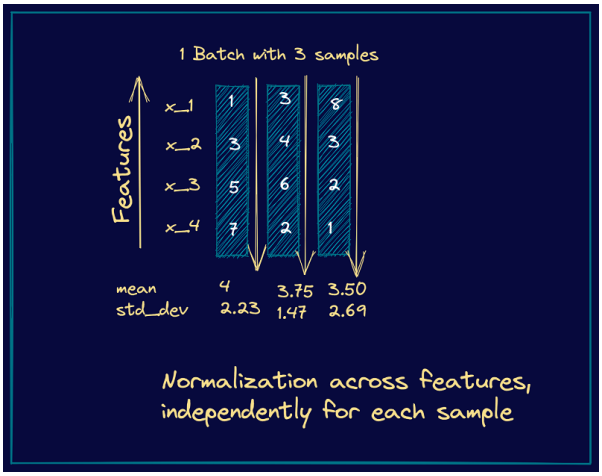

Figure 2.14: Dummy example of a layer in a multi-layer perceptron with 4 neurons considering 3 training examples. Layer normalization involves applying standardization to the output features x_i of the previous layer before using them as input to the next layer for each of the 3 training samples.

In summary, the training of neural networks, especially for tasks like handwritten digit recognition, involves iteratively adjusting weights and biases to minimize the loss function, which measures the discrepancy between predicted outputs and actual labels. Techniques such as stochastic gradient descent, skip connections, the incorporation of momentum via exponentially weighted moving averages, normalization, and the use of the ReLU activation function instead of the sigmoid function help navigate the complex landscape of the loss function, ensuring efficient convergence to a minimum.

While the multi-layer perceptron provides a foundational architecture for understanding neural networks, the field of deep learning has since evolved, first diverging into task specific architectures and recently converging to the transformer architecture. The next section delves into this transformer architecture as it is the basic building block of all the models in chapter 3.

2.3. Generative Pre-trained Transformer

With the foundational understanding of neural networks from sections 2.1, 2.2.1, and 2.2.2, which discussed the multi-layer perceptron architecture, the reader now has all the tools to understand the intuition behind the transformer architecture, the building block of all open-source models discussed in chapter 3.

Before the deep learning field arrived at the transformer architecture, the discipline evolved to design specialized neural network architectures tailored to specific tasks. This shift from a general-purpose design with the multi-layer perceptron to more sophisticated, purpose-built models has significantly enhanced performance on various data types and tasks.[51]

Natural language processing (NLP) and computer vision (CV) have emerged as prominent subfields of deep learning. NLP started with John Hopfield's recurrent neural networks (RNNs)[52], and later, the long short-term memory (LSTM) networks introduced by Jürgen Schmidhuber and Sepp Hochreiter in 1997, which improved long-term dependencies as standard RNNs had difficulties processing longer pieces of text, often "forgetting" the earlier parts while processing.[53] This subfield is also where the transformer architecture emerged. The seminal paper "Attention is All You Need" introduced the transformer architecture, which uses self-attention mechanisms to process entire texts simultaneously, overcoming the limitations of sequential models like RNNs and LSTMs.[1] This design allows each word

in a sentence to consider the relevance of every other word, enabling parallel computation and efficient handling of long-range dependencies. NLP underpins technologies like Google Translate, Alexa, and ChatGPT.

Within the CV subfield, Yann LeCun's pioneering work on convolutional neural networks (CNNs) in 1989 revolutionized image recognition by applying backpropagation to convolutional layers.[29] CV underpins technologies like autonomous driving, medical image analysis, and facial recognition software.

Later, it was discovered that transformers had the ability to combine the NLP and CV subfields. Transformers represent a significant convergence of NLP and CV capabilities. The emergence of Vision Language Models (VLMs) exemplifies this convergence. VLMs integrate visual and textual data using transformer architectures, tokenizing images and aligning them with word embeddings to analyze both modalities within a unified framework.[54] This capability makes transformers adept at handling multimodal inputs, producing coherent and contextually aware outputs across visual and textual domains.

This section will delve into the transformer architecture, focusing on the GPT model variant, specifically GPT-3. GPT stands for Generative Pretrained Transformer: "Generative" signifies the model's ability to produce new outputs, "Pretrained" indicates an extensive initial training phase on a broad dataset, and "Transformer" refers to the neural network architecture central to recent AI advancements.[55] While the original transformer model was designed for text translation,[1] GPT-3's large scale, with approximately 175 billion parameters, enables it to generate high-quality, coherent text by iteratively predicting and sampling the next word in a sequence.[56][57]

The following subsections will provide an overview of the data flow within the transformer architecture, using GPT-3 as a case study to explain how it predicts the next words based on a given text seed. This will address the mechanisms behind the success of transformer models, setting the stage for a deeper understanding of their application in NLP and beyond.[6][58][59]

2.3.1. Overview of Data Flow in Transformer Models: From Tokenization to Text Generation

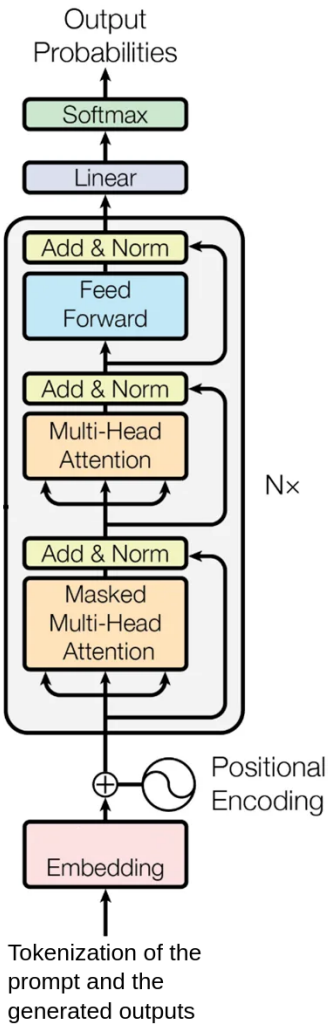

Figure 2.15: The decoder component of the transformer architecture underlying the GPT models[1]

To begin the high-level overview of how data flows through a transformer, the input text called the 'prompt' is first divided into smaller segments known as tokens using a tokenizer. The result is a set of tokens that represents common substrings in the dataset. In the context of text, these tokens typically consist of whole words, fragments of words, or common combinations of characters. This segmentation marks the initial step in the data processing sequence of the transformer, see figure 2.15.[60]

Each token is then associated with a vector, which is essentially a list of numbers intended to encode the meaning of that piece. These vectors can be thought of as coordinates in a high-dimensional space where words with similar meanings are positioned closely together. These vectors are called embeddings in Figure 2.15. The figure shows positional encodings are added to these embeddings, providing unique positional information to each input token to preserve sequence order, since transformers do not inherently capture this. Using sine and cosine functions of different frequencies for each embedding dimension, these encodings allow the model to differentiate between token positions in the sequence. This is important because the order of words can have great impact on the meaning of a sentence: "A man ate a big fish and was happy" or "A big fish ate a man and was happy".

The resulting series of vectors then enters a mechanism known as a multi-head attention block, see figure 2.15. This block enables the vectors to interact, exchanging information to adjust their values based on context.[61] For instance, the word "bank" has a different meaning in "people put more money in the bank" compared to "The wolf patiently waited on the bank of the river for a fish to swim by." The attention block determines the relevance of surrounding words in updating the meanings of each word, and how these updates should be applied. The term "masked" in figure 2.15 indicates that each token only has access to the information that came before it, ensuring it cannot see future tokens. This technique preserves the integrity of the sequence modeling process by preventing tokens from "cheating" and using information that had not been generated yet at the time of their selection. Consequently, this maintains the correct autoregressive property needed for accurate language modeling and prediction tasks.[62] [63] The next part involves two familiar concepts, the skip or residual connection and the layer normalization that were already explained in section 2.2.2. These two processes are highlighted in figure 2.15 with the "Add & Norm" and the feed forward arrow. The layer normalization improves training by providing numerical stability and the skip connection prevents the vanishing gradient effect.[47][42] Figure 2.15 shows that this process is repeated as the data enters another multi-head attention block with a skip connection and layer normalization.

Subsequently, these vectors are processed through a multi-layer perceptron, an operation previously described in section 2.2.1. In this phase, the token vectors do not interact with each other; instead, they are processed in parallel through the same operation. Analogous to the process used in handwritten digit recognition, the multi-layer perceptron block analyzes the updated word vectors for features. The interpretation of this block can be linked to posing a series of questions about each vector, with updates to the vectors being made based on the responses to these questions. These could be questions like: Is it part of a joke? Is it related to technology? Is it related to technology?[64]

The decoder shown in figure 2.15 systematically alternates between attention blocks and multi-layer perceptron (MLP) blocks, repeating this sequence Nx times, where Nx indicates the number of repetitions of these blocks. This sequential alternation is designed to progressively refine the representation of the input text. Initially, each vector in the sequence corresponds to an embedded representation of a word or subword. As these vectors pass through the successive layers of the transformer architecture, they are incrementally updated to reflect a more comprehensive understanding of the entire input text.[63]

By the end of this process, the final vector in the sequence ideally encapsulates the enriched meaning of the initial token, having integrated contextual information from the entire text. This enriched vector serves as the basis for the next step in the transformer's processing: generating a probability distribution. This distribution, calculated from the final vector, spans all possible subsequent tokens and represents the potential continuations of the text sequence through a linear layer and a softmax function, the better alternative of the output layer of the multi-layer perceptron for handwritten digit recognition that was hinted at in section 2.2.1. This softmax function treats the outputs as next-word prediction just like handwritten digit recognition involves a mutually exclusive classification problem.[40]

This decoder model in combination with an input and the generated output allows for an iterative process: it begins with a seed text or prompt, predicts subsequent text, samples from the resulting distribution, appends this sample, and repeats the cycle. This mechanism is reminiscent of early demonstrations of GPT-3, where the model would complete stories and essays from an initial text snippet. [57] To transform such a tool into a chatbot, one typically starts with a system prompt that sets up an interaction with a helpful AI assistant. The user's initial question or prompt serves as the initial dialogue piece, prompting the system to generate a suitable response from the AI assistant. While there are additional training steps necessary to refine this functionality, this overview outlines the basic concept.

Next sections break down the attention mechanism in detail, starting with the tokenization.

2.3.2. Tokenization

As stated in the overview of the transformer architecture given in section 2.3.1, the initial step involves segmenting the input text into smaller segments, see figure 2.16. In GPT-3's vocabulary, tokens are derived using the Byte Pair Encoding (BPE) algorithm, which iteratively combines frequently occurring character pairs into tokens.[65] This process results in whole word tokens, which are common words

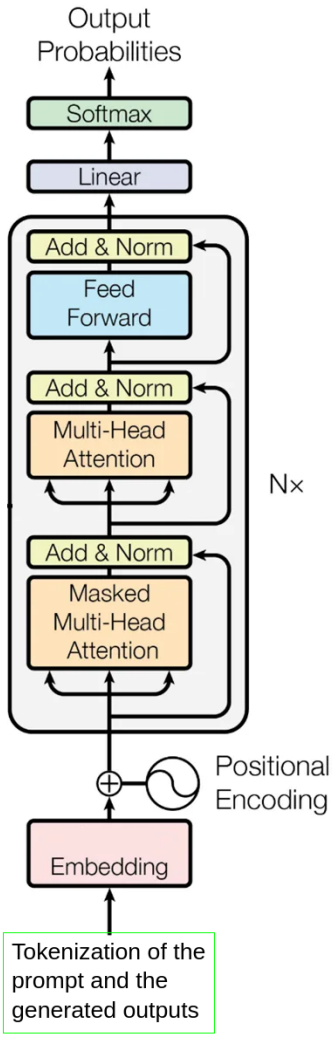

Figure 2.16: Highlight of the tokenization within the decoder of the transformer architecture underlying the GPT models[1]

encoded as single units, subword tokens, which represent segments of less frequent words, and byte tokens, used for encoding rare characters and symbols at the byte level for comprehensive text handling. Additionally, punctuation tokens are treated as separate entities, enabling the model to accurately capture syntactic nuances and sentence structures. Special tokens such as <|startoftext|> and <|endoftext|> delineate the boundaries of texts, crucial for text generation and interpretation tasks. This tokenization strategy allows GPT-3 to efficiently process a diverse array of linguistic elements, enhancing its understanding and generative capabilities.

BPE works by iteratively replacing the most frequent pair of bytes in a text with a single, unused byte. To illustrate how Byte Pair Encoding (BPE) works using a simple example with actual words, consider a small dataset consisting of the words: "low", "lower", "newest", "widest". The BPE algorithm starts by breaking these words into individual characters and frequently occurring sequences. Initially, every character is treated as a separate token. BPE then counts the frequency of pairs of adjacent tokens and repeatedly merges the most frequent pairs to form new tokens. In our example, suppose "e" and "s" appear adjacent to each other frequently across different words, as in "newest" and "widest". BPE would merge "e" and "s" into a single token "es". In subsequent iterations, other common pairs like "ne" in "newest" or "lo" in "low" and "lower" might be merged next, resulting in new tokens like "ne" and "lo". This merging continues until a specified number of merges has been reached or until further merging is non-beneficial.[66]

For simplicity of the explanation, in the following sections the tokens are considered to be whole words, while in reality they can also represent sub-word tokens, byte tokens and special tokens.

2.3.3. Word Embeddings

Figure 2.17 highlights that the next step involves the embedding of the words or tokens. A model such as GPT-3 employs a predefined vocabulary, usually comprising about 50,000 different tokens. The input text is segmented into tokens based on this vocabulary. To convert these tokens into vectors, the model utilizes an embedding matrix. This matrix features a column vector for each word in the vocabulary, which determines the initial vector transformation for each word. Marked as W_e, this matrix, like all matrices within the model, begins with random values but is subsequently refined through learning from data.[67]

The transformer internally learns this mapping from tokens to word vectors without the need for external algorithms. Nonetheless, the inspiration for this mapping comes from the Word2Vec algorithm, a specialized technique for performing this mapping.[68] Converting words into vectors is crucial for the subsequent processes in the model. This process is commonly referred to as "embedding" a word, promoting a geometric conceptualization of these vectors $\vec{E}_{\mathrm{i}}$ as points in a high-dimensional space. Analogous to the high-dimensional loss landscape in the example of handwritten digit recognition described in section 2.2.2, which enables the neural network to learn complex mappings from input image to output prediction, word embeddings typically exist in high dimensions. For example, in GPT-3, each word embedding comprises 12,288 dimensions. [57] Operating in such a high-dimensional space allows for many distinct directions, facilitating the capture of the nuances of language semantics.

Similar to relating the high dimensional loss landscape to the three dimensional alpes in section 2.2.2, the visualization of word embeddings can be simplified from a high-dimensional space by selecting a three-dimensional slice for projection. The critical concept here is that during training, as a model adjusts its weights, it optimizes how words are embedded into vectors. This adjustment tends to result in embeddings where locations within this space carry semantic meanings. For example, a search for words with embeddings nearest to "building" will reveal a cluster of words that share a similar, building-related essence. [69]

In the embedding space, directions also convey semantic meaning. Looking at figure 2.18, a well-known example of semantic directions in vector spaces involves examining the similarity between the difference vector of the vector representations of "woman" and "man," and those of "king" and "queen." [68] Theoretically, it would then be possible to determine the term for a female monarch by adding the woman-man difference vector to the vector representation of "king," and searching for the word embeddings closest to this resultant point. However, this example, while iconic, does not perfectly reflect reality. In practice, the actual embedding $\vec{E}_{\mathrm{queen}}$ often diverges from this predicted position,

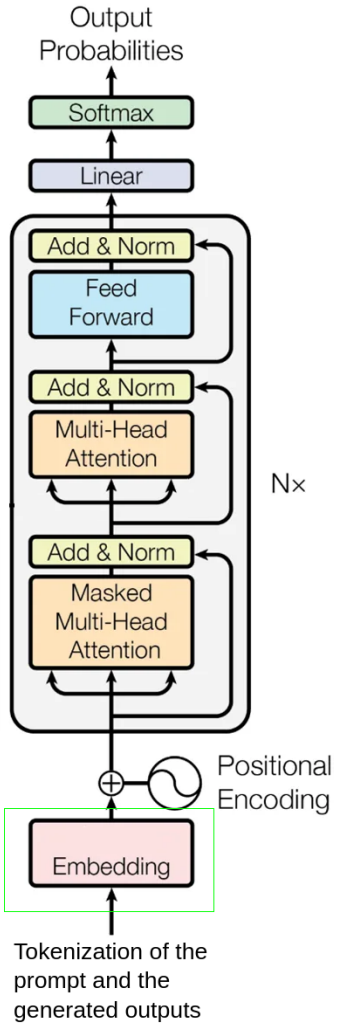

Figure 2.17: Highlight of the embedding within the decoder of the transformer architecture underlying the GPT models[1]

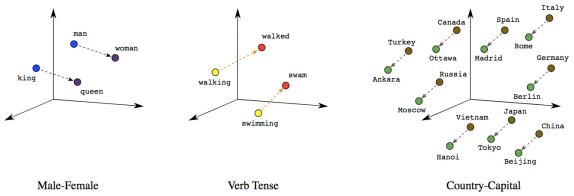

Figure 2.18: Low dimensional intuition between similarities between different word vectors highlighting their meaning and their relative distance showing directions of meaning[70]

likely because its usage in training data involves more than merely being a female equivalent of "king." For instance, "Queen" might refer to the band, or "queen" might refer to a drag queen.[71]

During training, it appears that the model strategically selects embeddings such that specific directions within the space encode particular types of information, such as gender in the above example (see figure 2.18. Geographical transitions are also captured in a specific direction as demonstrated by adding the difference vector obtained by subtracting the embedding of Madrid from the one of Spain and adding the one from Germany as this would land close to the embedding of Berlin (see figure 2.18.[68] This implies an association of directional vectors with countries and their capitals.

Confirming semantic directions in the embedding space utilizes the dot product of two embedding vectors to measure their alignment. This method emphasizes a geometric perspective, where the dot product yields a positive value when vectors point in similar directions, zero when they are perpendicular, and have a negative value when they point in opposite directions.[72]

To verify a semantic direction such as plurality using this method, one might hypothesize that the difference vector $\Delta\vec{E}$ derived from subtracting the embedding of "streets" from "street" represents plurality. To test this hypothesis, one can compute the dot product of this "plurality" vector with the embeddings of various singular nouns and their plural forms. Confirmation of the hypothesis would occur if the dot products with plural nouns consistently yield higher values than those with singular nouns, indicating that the plural forms are more closely aligned with the hypothesized "plurality" direction. The dot product thus serves as a useful tool for validating semantic relationships encoded within the vector space. [73] Exploring further, taking the dot product of the "plurality" vector with the embeddings of numerical words like 1, 2, 3, and so forth reveals increasing values. This suggests that the model quantitatively measures the "plurality" it attributes to each word. [68]

As highlighted in the overview section 2.3.1, positional embeddings are added to the word embeddings. These are used by the decoder to maintain the initial positions of the words relative to each other and require no further explanation.

To keep a parameter count for the decoder model, the embedding matrix W_e is recognized as the first set of weights in the model. Focusing on the specifics, GPT-3's vocabulary consists of 50,257 entries. Each token is represented in an embedding space of 12,288 dimensions. [57]. Multiplying the vocabulary size by the embedding dimension yields approximately 617 million weights. This is the part W_E takes up of the total of 175 billion parameters in GPT-3. With an understanding of word embeddings, the embedding space, and how directions within this space correlate to semantic meanings, the next section delves into the intuitive aspects of how context influences word meanings.

2.3.4. Contextual Dynamics and Context Limitations in Transformer Architectures

Within the framework of transformers, vectors in the embedding space $\vec{E}_{\text{i}}$ represent more than just individual words; they have the capacity to integrate contextual subtleties. Initially based on the embedding matrix W_e, a word vector undergoes transformations influenced by surrounding words that shape its semantic meaning. For instance, as discussed in section 2.3.1, consider the word "bank." In the context of "people put more money in the bank," the vector for "bank" shifts to a semantic space different from that in "The wolf patiently waited on the bank of the river for a fish to swim by," due to different surrounding words influencing its transformation.[74]

This concept aligns with the understanding of language: a word's meaning is shaped by its surrounding context, sometimes influenced by elements beyond the immediate text. In a transformer, each vector is initially derived directly from the embedding matrix, capturing only the standalone meaning of an individual word without any contextual influence. The network's main objective, however, is to enhance each vector with a nuanced and detailed meaning that goes beyond the capabilities of individual words alone.[61]

It is important to acknowledge that the network has a set limit on this number of surrounding words it can process to update a words meaning. This is known as the decoder's context window. For example, GPT-3 functions with a context window of 2048, meaning that the data limit traversing the network consists of an array of 2048 columns, each containing vectors of 12,288 dimensions. [57] This limitation

constricts the volume of text the transformer can evaluate for next-word prediction. Consequently, during extended interactions with a transformer, it may seem to forget earlier parts of the conversation as they exceed the context limit. Alternatively, if a single prompt exceeds the context window, the network will not be able to consider all elements of the input simultaneously.

within the decoder architecture the attention mechanism is responsible for transforming the initial word embeddings to contextualized embeddings as detailed in the next section.

2.3.5. Attention in transformers

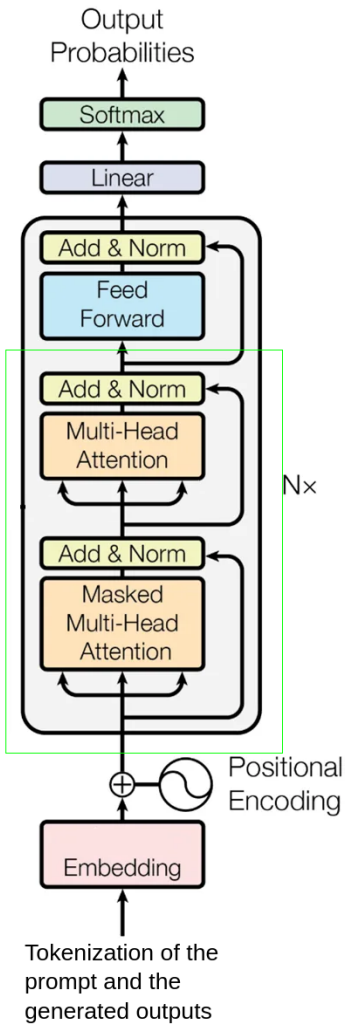

Figure 2.19: Highlight of the attention mechanism within the decoder of the transformer architecture underlying the GPT models[1]

Continuing with the word "bank," as detailed in section 2.3.4, the initial vector for "bank" remains constant across different inputs due to the fixed embedding matrix W_e after training, which functions as a basic lookup table without contextual influences. It is only in subsequent processing stages that surrounding embeddings can confer context-specific information on "bank." The model transforms the initial vector to move in various distinct directions in the embedding space, each corresponding to a different semantic meaning of "bank." A well-trained attention block then adjusts this generic embedding $\vec{E}_i$ towards a specific direction influenced by the surrounding context. Figure 2.19 highlights

the part of the decoder responsible for this process.

In a more abstract sense, the attention mechanism updates the meanings of words by combining their embeddings based on their interdependencies. As noted in section 2.3.1, once vectors pass through all the attention blocks of the transformer, the task of predicting the next token relies solely on the last vector in the sequence. For instance, if the input text includes a complete literature survey and the model needs to conclude the last sentence of the discussion with "therefore the best open-source models are:", the final vector, initially just representing "are," must undergo significant modifications by the attention blocks to incorporate comprehensive information from the entire text, thereby facilitating an accurate prediction of the subsequent word.

The Attention Pattern

To demonstrate the computational processes in attention mechanisms, consider the simplified example of the phrase, "A white and woolly sheep." This example focuses on how adjectives influence the meanings of their corresponding nouns through a single attention head that identifies nouns and their adjectives, adjusting the nouns based on the adjectives' relevance and meaning. In this scenario, the adjectives "white" and "woolly" modify the noun "sheep" according to their respective word vectors. Within one attention block, multiple attention heads operate simultaneously, allowing for various connections. For instance, another attention head might focus on the plurality of words, as seen when the article "A" indicates that "sheep" is singular. For now the focus is on a single head, continuing with the example of nouns and their adjectives.

The process starts with each noun generating a query vector, $\vec{Q}_i$, about potential adjectives modifying it. This is achieved by multiplying the noun's embedding by a query matrix, W_q, which transforms the high-dimensional embedding into a query vector in a reduced dimension, typically 128. The entries of W_q are model parameters that are optimized through training, and ideally, this matrix is designed to orient noun embeddings towards seeking out preceding adjectives. Equation 2.25 summarizes this process, showing that the query vector $\vec{Q}_i$ for the i-th noun is obtained by multiplying its embedding vector $\vec{E}_i$ by the query matrix $\mathbf{W}_q$.

$$\vec{Q}_i = \mathbf{W}_q \cdot \vec{E}_i \tag{2.25}$$

$$\vec{K}_i = \mathbf{W}_k \cdot \vec{E}_i \tag{2.26}$$

Simultaneously, a key matrix W_k is applied to all embeddings, producing a sequence of key vectors $\vec{K}_i$, which conceptually could provide the answers to the queries. Equation 2.26 summarizes this process, indicating that the key vector $\vec{K}_i$ for the i-th word is produced by multiplying its embedding vector $\vec{E}_i$ by the key matrix $\mathbf{W}_k$. The relationship between a key and a query is quantified by computing the dot product between each key-query pair. As each word in the input text now has its own key and query, it is possible to construct a square matrix visualized as a grid where the magnitude of each dot product reflects the alignment between keys and queries, see figure 2.20.

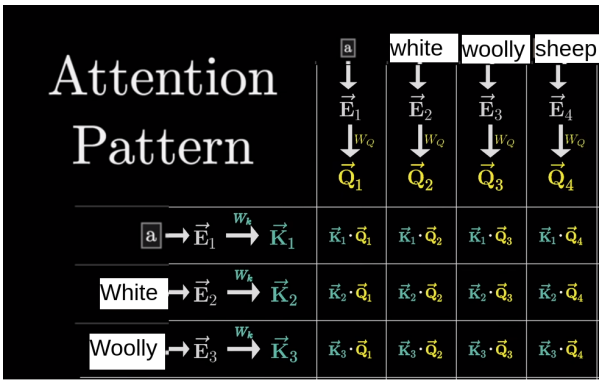

Figure 2.20: The attention pattern[75]

For example, in the adjective-noun relationship, if the keys for "white" and "woolly" align closely with the query for "sheep," the corresponding dot products in the key-query matrix would be large positive numbers, indicating strong relevance. Conversely, a key associated with an unrelated word like "and" would yield a low or negative dot product with the query for "creature," reflecting no meaningful relationship.

This key-query matrix of dot products, representing potential relevance scores between words, is then transformed into a normalized form. The normalization is achieved through the application of a softmax function along each column of the grid, converting the raw scores into a probability distribution that reflects the likelihood of each word's relevance to updating the meanings of others, equation 2.27. Similar to the neural network from section 1.1, most computations in the transformer take the form of matrix-vector products involving matrices filled with tunable weights W learned from data. For the example of a single head to update nouns based on their adjectives, the process starts with each noun generating a query vector, q, about potential adjectives modifying it. This is achieved by multiplying the noun's embedding by a query matrix, W_q, which transforms the high-dimensional embedding $\vec{E}_i$ into a query vector $\vec{Q}_i$ in a reduced dimension, typically 128. The entries of W_q are model parameters that are optimized through training, and ideally, this matrix is designed to orient noun embeddings towards seeking out preceding adjectives.

Simultaneously, a key matrix is applied to all embeddings, producing a sequence of key vectors, $\vec{K}_i$, which conceptually could provide the answers to the queries. The relationship between a key and a query is quantified by computing the dot product between each key-query pair. As each word in the input text now has its own key and query, it is possible to construct a square matrix visualized as a grid where the magnitude of each dot product reflects the alignment between keys and queries.

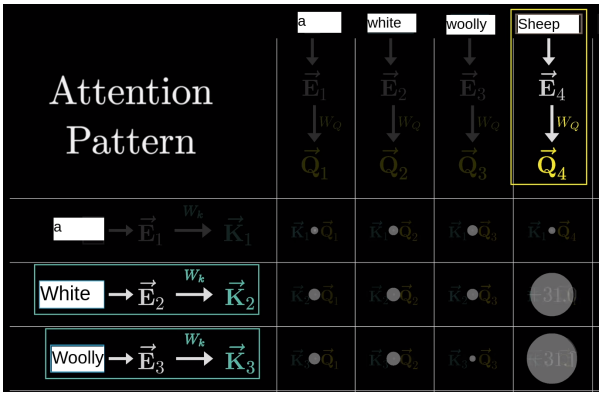

Figure 2.21: "A white woolly sheep" as an example for the attention pattern of an adjective-noun attention head, highlighting the high attention values of the adjectives white and woolly for the noun sheep [75]

For example, in the adjective-noun relationship, if the keys for "white" and "woolly" align closely with the query for "sheep," the corresponding dot products in the key-query matrix would be large positive numbers, indicating strong relevance, see figure 2.21. Conversely, a key associated with an unrelated word like "and" would yield a low or negative dot product with the query for "creature," reflecting no meaningful relationship.

This key-query matrix of dot products, representing potential relevance scores between words, is then transformed into a normalized form. The normalization is achieved through the application of a softmax function along each column of the grid, converting the raw scores into a probability distribution that reflects the likelihood of each word's relevance to updating the meanings of others. Equation 2.27 presents this softmax function. Here, "A" represents the attention factor that resulted from multiplying the keys with a query for a specific query.

$$\text{Softmax}(A_i) = \frac{\exp(A_i)}{\sum_j \exp(A_j)} \tag{2.27}$$

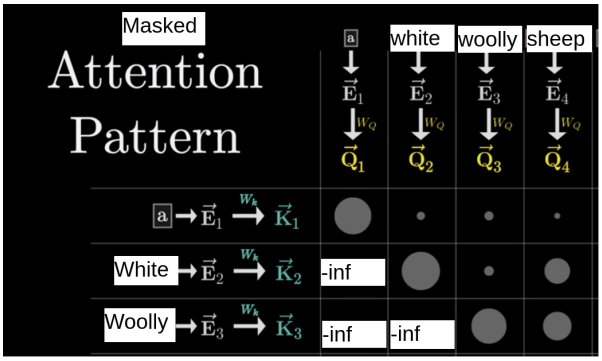

Figure 2.22: Masked attention where words can only use previous words to update their meaning [75]

The structured output of this process, known as an attention pattern, essentially weights each word's influence on another based on the calculated probabilities. [76] This is the mechanism that allows for the integration of context into word meanings as discussed in section 2.3.4.

2.3.6. Masking the attention pattern

Similar to the straightforward neural network discussed in 1.1, a transformer is also trained by adjusting its weights based on its performance in making predictions, specifically the prediction of the correct subsequent word in a text sequence. This optimization process is made more efficient by having the model predict the next possible token for each subsequence within the text. For example, in the phrase "A white and woolly sheep," the model not only predicts the word that follows "sheep" but also the subsequent words for subsequences such as "white," "and," "woolly" and "sheep." This approach multiplies the learning opportunities from a single piece of input text, thereby increasing the efficiency and effectiveness of the training process.

When generating attention patterns, it is vital to ensure that subsequent words in a sequence do not affect the meaning of preceding words. As presented thus far, the key-query matrix computations involved dot products between keys and queries, where dot products were also calculated for keys correspond to words appearing later in the text than those associated with the queries. This configuration is problematic during inference, as the model must predict the next word without access to subsequent words. Referring to the ongoing example, using the word "sheep" to update the embedding $\vec{E}_{\text{woolly}}$ of "woolly" would be inappropriate because "sheep" is the subsequent word the model is trying to predict. To maintain the predictive integrity of the model, it is necessary to neutralize entries in the attention matrix that allow later tokens to influence earlier ones, thus ensuring that the influence of subsequent words is not prematurely factored into the model's calculations.

The standard method to achieve this is by applying a masking technique before the softmax normalization. The masking process involves setting entries of the key-value matrix that should not influence predictions to negative infinity, see figure 2.22. When the softmax function is subsequently applied, these entries effectively become zero, maintaining the normalization of the columns such that each sums to one. This masking process is essential in models like GPT-3 during training to maintain the integrity of the predictive process, ensuring that only appropriate contextual influences shape the predictions. While this feature is particularly crucial during training to make sure the model is trained to predict the next word in the sequence, it is also applied in operational settings to ensure that the model behaves the same as it does during training.

2.3.7. The context window

The concept of a "context window" is what determines the operational limits of transformers. It defines the maximum number of input tokens that the model can process simultaneously. This capacity affects the model's ability to manage and interpret extensive sequences, which is important in applications like code generation where understanding complex structures and maintaining consistency are crucial.

However, the size of the context window also represents a significant bottleneck. The primary issue is that attention mechanisms in these models have quadratic time and memory complexities relative to

the length of the input. [77] As a result, expanding the context window not only requires exponentially more computational resources but also dramatically increases memory demands. These constraints have spurred recent advancements aimed at modifying the attention mechanism to enhance its scalability and manage larger context windows more efficiently. [77] [78] [79] [80] However, in the current discussion, the focus remains on the foundational principles of the attention mechanism without delving into these advanced variations. This foundational understanding is crucial for grasping how the basic attention mechanism functions and why adjustments to it are necessary for handling larger contexts in more complex modeling scenarios.

2.3.8. An intuition for values

Computing the attention pattern inside one attention head allows the model to determine which words are relevant to others in the text for a specific relationship. This step enables the model to update the word embeddings based on the identified relationships, thus allowing words to transmit information to other relevant words. For instance, the model might aim to adjust the embedding $\vec{E}$ of "sheep" based on the influence of "white," shifting it within the 12,288-dimensional embedding space to more accurately represent a "white sheep."

The typical approach to achieve this involves the use of a third matrix, known as the value matrix W_v. This matrix is multiplied by the embedding $\vec{E}$ of a word, such as "White," to produce what is termed a value vector $\vec{V}$. This vector represents the adjustments to be added to the embedding of another word, in this case, "sheep," to modify its meaning accordingly.

While it may appear unnecessary since the embedding vector $\vec{E}_{\text{white}}$ already encapsulates the meaning of "white," the utility of the value matrix becomes clear when considering the concept of plurality from section 2.3.5. In this section, it was highlighted that each attention head might focus on specific word relationships, such as noun-adjective or plurality. For example, in addressing plurality, the attention pattern ideally focuses on the word "A" within the context of its plurality to update "sheep" to be plural, and the value matrix is expected to isolate the plurality aspect of "A". It is critical to recognize that the initial embedding of "A" may also embody other meanings, such as the grade "A" or the musical note "A". The role of the value matrix, therefore, is to extract the aspect of the meaning relevant to the specific relationship the attention head is exploring.

In practice, once the attention pattern is established, the value matrix is applied to all embeddings of the input text to generate a series of value vectors. To update the embeddings, each value vector $\Delta\vec{V}$ is scaled according to the scaling factors determined by the attention pattern and then summed up. For the word "sheep," this means adding significant portions of the value vectors for "woolly" and "white," based on their relevance as indicated by the attention pattern, while effectively nullifying the contributions from less relevant words.

The final step in the update process involves summing these weighted value vectors for each word, resulting in a change vector, denoted here $\Delta\vec{E}$. This vector is then added to the original embedding $\vec{E}$ of the word, refining it to embody a more context-rich representation, such as that of a "woolly white sheep."

This process is not isolated to a single embedding but is simultaneously applied across all embeddings in the text. The cumulative effect of these updates across the entire sequence results in a set of refined embeddings that emerge from the attention block, each embedding now enriched with contextual information gleaned from related preceding words within the text. This method effectively transforms the initial context-independent embeddings into a dynamic representation of the text's semantic structure as understood through the interactions modeled by the attention mechanism.

2.3.9. More efficient values

This entire process is characterized as a single head of attention. Throughout the description, it has been emphasized that this process is parameterized by three distinct matrices W_k, W_q, and W_v, all of which contain tunable parameters. GPT-3's over-all parameter count started in section 2.3.3 and continues here by looking at the number of parameters in these key, query and value matrices respectively.

The key and query matrices each have 12,288 columns, which correspond to the embedding dimension,

and 128 rows, aligning with the smaller key-query space dimension. This configuration results in approximately 1.5 million parameters for each matrix. In contrast, the value matrix, as initially described, would be a square matrix with both dimensions equaling 12,288, suggesting a total of about 150 million parameters if it were to maintain the same dimensionality for both inputs and outputs.

However, for efficiency, especially when running multiple attention heads in parallel, it is common to adjust the structure of the value matrix. Instead of using a single large square matrix, the value transformation is often factored into two smaller matrices. Conceptually, one can still think of the value matrix as performing a linear map within the large embedding space, such as transforming the vector $\vec{E}_{\text{sheep}}$ in the 12,288 dimensional space to point in a direction that includes the fact that its white and woolly. [81]

Practically, this separation involves first reducing the large embedding vectors down to a smaller space, typically the same 128 dimensional space as the keys and querys and then transforming it back up to the original embedding space dimensions. This restructuring effectively constrains the value transformation to a low-rank modification, which is less resource-intensive and more manageable in terms of computational complexity. Intuition behind such low-rank approximations can be found in image compression using singular value decomposition, an optimal method for finding the best low-rank approximations.[82]

Applying the low-rank modification splits the value matrix into two smaller matrices leads to a more uniform and balanced parameter distribution between W_k, W_q, and W_v, where W_v now consists of 2 matrices. Summing the parameters from all four matrices yields approximately 6.3 million parameters for a single attention head.

2.3.10. Values in practice: the output matrix

In discussing the nuances of transformer architecture, it is crucial to clarify a particular aspect about the structuring of the value matrices, which might otherwise cause confusion when delving deeper into related literature. Previously, it was mentioned that the value map within each attention head is typically factored into two distinct matrices with a low-rank approximation. This division helps conceptualize how embeddings are first reduced in dimensionality and then projected back up into the original high-dimensional space.

However, the practical implementation in many transformer models, as described in academic papers, often differs from this straightforward conceptualization. While each attention head does indeed use a distinct matrix for the initial projection down to a 126 dimensional space, the subsequent expansion is commonly handled in a more consolidated manner across all heads.

In practice, the matrices that transform the values back up to the rank of the embedding space from all attention heads are combined into a single large matrix, referred to as the output matrix. This matrix is associated with the entire multi-headed attention block rather than individual heads. Consequently, when references are made to the "value matrix" in the context of a specific attention head in scholarly articles, they typically refer only to the matrix used in the dimensionality reduction step, not the ranking up step. [81] Recognizing this could prevent potential confusion when encountering different descriptions in the literature. This note aims to bridge the gap between the conceptual explanation provided here and the technical descriptions found in academic research on transformers.

2.3.11. Multi-headed attention

Expanding upon the concept of a single attention head, a full attention block within a transformer utilizes what is termed multi-headed attention. This involves running several attention processes in parallel, each with its own set of key, query, and value matrices. For instance, GPT-3 employs 96 attention heads per block, each contributing to a nuanced understanding of context.

Section 2.3.5 hinted at the necessity of multiple heads with the example of noun-adjective relation ships and plurality. In practice, each of these 96 heads generates what a single head generates: a unique attention pattern and corresponding value vectors based on the relationship that the head tries to find. Based on that each head finds a change $\Delta\vec{E}$ that it wants to add to the original embedding to change its meaning. For the case of multi-head attention, all these changes $\Delta\vec{E}_{\text{i}}$ are added to the original word

embedding, thereby refining the embedding at each position based on a composite of influences from all heads.

This process is essential for allowing the model to capture and learn from a multitude of ways in which context can alter meanings. Given the 96 distinct sets of matrices within each block of GPT-3's multi-headed attention, the total parameter count for this component alone approaches 600 million. The key takeaway is that by employing numerous parallel heads, the model gains the ability to discern and learn from diverse contextual interactions, thereby enhancing its linguistic and predictive capabilities.

2.3.12. Decoder architecture

With the understanding of the attention block, consider how data is processed beyond a single attention block. As previously introduced, data within a transformer not only passes through attention mechanisms but also encounters multi-layer perceptrons (MLPs), explained in section 1.1. Additionally, data does not simply pass through one set of these operations but cycles through numerous iterations of both attention blocks and MLPs.

This repetitive processing allows each word's embedding to absorb and integrate increasingly nuanced aspects of its context. As embeddings progress deeper into the network, they gather more sophisticated and abstract information from surrounding embeddings. The aim is to evolve beyond mere grammatical and descriptive details to encapsulate broader concepts such as sentiment, tone, literary form, and pertinent scientific or thematic elements.

Reflecting on the architectural scale, GPT-3 features 96 distinct layers. Consequently, the total count of key, query, and value parameters is amplified by these layers, cumulatively approaching 58 billion parameters dedicated solely to the attention mechanisms. This figure, while substantial, represents only approximately one-third of the total 175 billion parameters within the entire network. Thus, despite the significant focus on attention mechanisms due to their critical role, the majority of the model's parameters are actually located within other structural components between these attention processes. Concluding with the phrase: Attention is one third of what you need.

2.3.13. Predicting the next word: softmax, beam search and temperature

In the final sections of this chapter, the focus shifts to the application of the softmax function, as introduced in section 2.3.5, for predicting the next word. After the input text has been processed through all the model's layers, the transformed embedding of the last word dictates the subsequent word prediction. Here, softmax normalization transforms this vector into a normalized probability distribution. Unlike the standard softmax outlined in equation 2.27, this model incorporates a 'temperature' parameter as detailed in equation 2.28, which modulates the sharpness of the probability distribution. A higher temperature setting spreads the probabilities more evenly, increasing the chances of selecting less probable words and potentially enhancing creativity, albeit at the risk of reduced coherence. Conversely, a lower temperature concentrates the distribution around the most likely outcomes, fostering predictability in the generated text. This temperature-adjusted softmax thus introduces an additional layer of complexity, allowing for the fine-tuning of the model's output characteristics in applications like ChatGPT.

$$\text{Softmax}(A_i, T) = \frac{\exp\left(\frac{A_i}{T}\right)}{\sum_j \exp\left(\frac{A_j}{T}\right)} \tag{2.28}$$

Beam search plays a crucial role in refining the next-word prediction process in GPT-3, particularly when generating coherent and contextually appropriate text sequences. The algorithm interacts with several other components of the model, including the softmax function and temperature setting, to optimize output quality.

In GPT-3, after processing the input text through its layers, the model employs a softmax function at the output layer. This function generates a probability distribution over all possible next words in the model's vocabulary. Each word is assigned a probability score that quantifies its likelihood of being the appropriate next word in the sequence, based on the input context.

With these probabilities provided by the softmax function, beam search then takes over to select and extend potential sequences. Instead of merely choosing the single most probable next word (as in greedy search), beam search expands multiple sequence possibilities, each initiated by one of the top-B most probable words (where B is the beam width). For instance, if B is set to 3, the algorithm evaluates the top three initial words based on their probability scores. For each of these words, it again uses the softmax probabilities to determine the next set of probable words, combining them into new sequences with the existing ones and evaluating their joint probabilities. This evaluation often involves multiplying the probabilities of individual words in the sequence, thus preferring sequences that overall have higher cumulative probabilities.

Consider a scenario where the starting sequence is "The quick brown fox," and the initial output candidates from the algorithm are ["jumps", "runs", "sits"]. Beam search progresses by evaluating the next three most probable words that could follow each of these initial candidates. The algorithm calculates the combined probabilities of the possible sequences, such as $P(y_1|x) \times P(y_2|x, y_1)$, where y_1 is one of the initial candidates and y_2 is a potential following word. It then selects the top three combinations based on these probabilities, which might include sequences like ["jumps over", "runs quickly", "sits quietly"]. Importantly, the number of sequences considered at each step—known as beams—always remains at three. This means that sometimes an initially promising word like "sits" might be dropped in a subsequent round if the cumulative probabilities of sequences starting with "jumps" or "runs" become higher when combined with their respective next words. This method ensures that at each decision point, only the sequences with the highest overall potential are carried forward.

The temperature parameter modifies the probability distribution output by the softmax before it is processed by the beam search. A higher temperature results in a softer probability distribution, where differences in probability between words are less pronounced, allowing for more diverse and potentially creative or unusual word choices. Conversely, a lower temperature makes the distribution sharper, increasing the likelihood that higher probability words are selected more consistently, which can enhance coherence but reduce novelty.

In the same example of "The quick brown fox," where initial output candidates are ["jumps", "runs", "sits"], the temperature parameter used in the softmax function significantly influences the distribution of probabilities across these candidates. A higher temperature results in a softer probability distribution, where differences between the probabilities of the words are less distinct. This means less probable words like "sits" might receive a higher relative probability, increasing the diversity of the words considered for the next steps in beam search. Conversely, a lower temperature sharpens the probability distribution, making high-probability words like "jumps" even more dominant and likely to be chosen. This alteration can affect which sequences are extended and maintained in the beam search process. For example, with a high temperature, the sequence ["sits quietly"] might be more competitive against ["jumps over"] or ["runs quickly"], potentially altering the final selections in the beam, as the algorithm seeks to balance between exploring novel word combinations and focusing on more likely sequences.

By integrating these components, softmax for probability distribution, temperature for adjusting this distribution, and beam search for exploring and selecting the most promising sequences, GPT-3 efficiently balances between creativity and coherence in its text generation. This methodology ensures that the generated text not only continues logically from the given context but also maintains a level of unpredictability and variation, key for applications ranging from conversational AI to content creation.

Consider a scenario where the context concludes with the word "scientist," preceded by references to "Marie Curie" and the phrase "radioactive elements." In this case, a network well-versed in the history of science is likely to assign a high probability to the token "radium," assuming tokens correspond directly to complete words.

The process to arrive at this determination consists of two primary steps. A matrix is employed to transform the final vector in the sequence into a list of 50,000 values, each correlating to a token in the vocabulary. Subsequently, these values are normalized into a probability distribution using a the softmax from equation 2.27. It may appear unusual to base predictions solely on the last vector, given the presence of thousands of other vectors rich in contextual data. Nevertheless, it has been established through training that utilizing each vector in the final layer to predict the subsequent token introduces efficiencies.

The matrix used in this transformation is known as the Unembedding matrix, denoted as WU. As with all matrices in this model, the initial entries of WU are random but undergo refinement through training processes. In terms of parameter accounting, the Unembedding matrix, which is structured inversely to the embedding matrix, adds approximately 617 million parameters to the network. This increase brings the total count to just over a billion, representing a modest but noteworthy fraction of the overall 175 billion parameters in the model.

2.3.14. Cross-Attention

Zooming out and examining the concepts introduced thusfar, GPT 3 uses a decoder only architecture. Looking at figure 2.23, this implies that only the right side of the architecture is used, excluding the left encoder part. This decoder only architecture only attends to its own tokens for next word prediction without crossing over to an encoder. Therefore this type of attention is specifically referred to as self-attention. This form is distinct from another type of attention mechanism called cross-attention. Although cross-attention does not directly pertain to the GPT model currently under examination, it is relevant to the transcription model used for GazeGPT as explained in section 3.1.

As described previously, self-attention is used to predict the next word in an autoregressive manner: beginning with a prompt, the model identifies the most likely next word to append to the input text. Consider the task of translation. Suppose there is an input text in French that needs to be translated into English. This task differs fundamentally from self-attention because instead of appending to an input prompt, it involves generating an entirely new English text that corresponds to the French sentence. This is the exact problem which the authors of the "attention is all you need" paper faced and they solved it using cross-attention. [1] Looking at figure 2.23, the authors designed an architecture that contains an encoder (left) and decoder (right).

To highlight the similairities and differences between the encoder and decoder, the example of french-english translation is continued. They both use tokenization (section 2.3.2), word embeddings (section 2.3.3), the attention pattern (2.3.5), the context window.

Looking at the left part of figure 2.23, the encoder, similar to the decoder, employs sSimilarly, in speech recognition, cross-attention can link audio input directly to its corresponding text output. For these models, keys are generated from one dataset (e.g., source language text) and queries from another (e.g., target language text), enabling the model to identify correspondences between elements from two distinct data streams. To illustrate with the sheep example, consider the sentence in French: "Un mouton blanc et laineux." In this scenario, queries might be associated with the English sentence and keys with the French sentence. The attention pattern then functions to determine which words in the French sentence correspond to which words in the English sentence, effectively linking the two through their semantic relationships.

There is a clear distinction in the model architecture of transformers in which cross-attention is used compaired to the architecture of models in which only self-attention is used. This is clear from the original paper, as it used an encoder decoder architecture, figure 2.23. Models that are used to generate text, only need the decoder part of the architecture. [1]

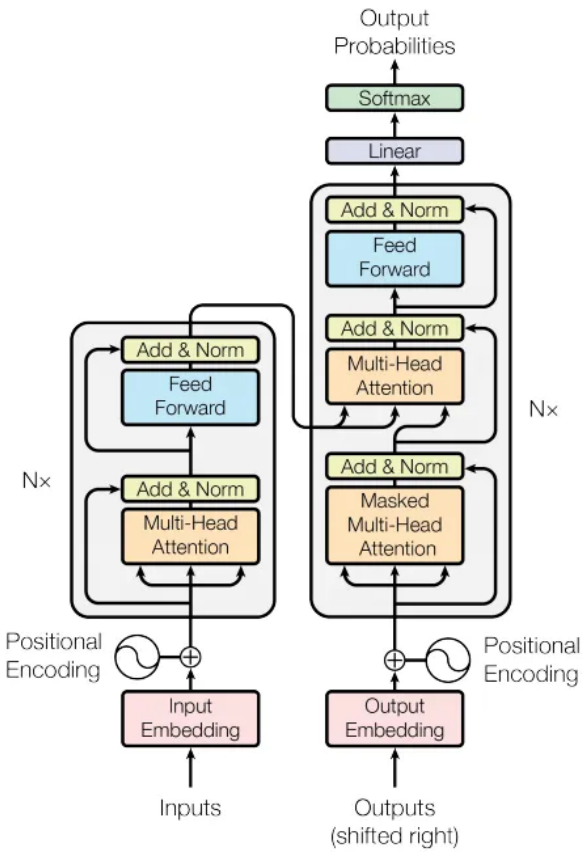

Figure 2.23: Original transformer architecture highlighting the encoder (left) and decoder (right) [1]

A key procedural distinction in cross-attention is the absence of masking. As discussed in section 2.3.6, masking is typically used in self-attention to prevent the model from accessing future tokens in the sequence, thus ensuring that predictions are based solely on previously seen data. However, in cross-attention, where datasets are processed but not in a sequential order relative to each other, there is no risk of future data 'leaking' into the present calculations. Consequently, there is no need for masking in cross-attention.

Understanding both self-attention and cross-attention is crucial for a comprehensive appreciation of the flexibility of attention mechanisms in managing diverse data interactions across neural networks. While the present discussion concentrates on self-attention as utilized in models like GPT, recognizing the operational dynamics of cross-attention enhances understanding of the extensive capabilities of neural architectures in complex computational tasks such as translation and transcription. To clarify terminology, the attention mechanism discussed so far is specifically referred to as a self-attention head, which is distinct from another variant known as cross-attention. This distinction is important to note, even though cross-attention is not directly relevant to the GPT model currently under discussion, it is in the transcription models discussed in chapter 3.

Cross-attention is utilized in models that process two distinct streams of data. For instance, in machine translation, cross-attention mechanisms allow the model to relate text from one language (source) to text in another language (target), facilitating the generation of translated content. This was wat the first transformer was designed for. [1] Similarly, in speech recognition systems, cross-attention could link audio input to its corresponding text transcription. In these scenarios, the keys are generated from one set of data (e.g., the source language), while the queries are derived from another (e.g., the target language), enabling the model to determine correspondences between elements of the two distinct datasets.

A key operational difference in cross-attention is the absence of masking. Masking is typically used in

self-attention to prevent the model from accessing future tokens in the sequence, thus ensuring that the predictions are based solely on previously seen data. However, in cross-attention, since the datasets involved do not follow a sequential order relative to each other, there is no risk of future data 'leaking' into the present calculations, and hence no need for masking.

Understanding both self-attention and cross-attention is vital for appreciating the versatility of attention mechanisms in handling various types of data interactions within neural networks. While this discussion focuses on self-attention as it applies to models like GPT, recognizing how cross-attention functions opens up broader insights into the advanced capabilities of neural architectures in complex tasks such as translation and transcription.

2.3.15. Discussion

The transformer architecture, a breakthrough in artificial intelligence, has significantly impacted speech-to-text, vision-language, and text-to-speech models. Understanding this architecture intuitively requires a brief look at the evolution of deep learning. Deep learning originated in the late 1950s with Frank Rosenblatt's perceptron, a model inspired by the brain's neural networks. Despite early limitations, the field advanced with the introduction of the sigmoid activation function and backpropagation algorithm by Geoffrey Hinton and colleagues in the 1980s, enabling more effective training of neural networks.

The multi-layer perceptron (MLP) is a foundational concept in deep learning, consisting of multiple layers of neurons that transform input data through linear transformations and non-linear activations. This iterative process of training and gradient descent in MLPs set the stage for more complex models like transformers. The transformer, introduced in "Attention is All You Need," leverages a self-attention mechanism that allows the model to weigh the relevance of different parts of the input data relative to each other. This mechanism involves generating queries, keys, and values for each input token, computing dot products between queries and keys, and normalizing these through the softmax function to produce an attention score matrix. This matrix guides how much each token influences others, capturing long-range dependencies and contextual relationships.

Transformers employ multi-headed attention to capture various relationships within the data by running multiple self-attention mechanisms in parallel. The outputs of these attention heads are combined to create a comprehensive representation of the input data. Additionally, positional encoding is added to the input embeddings to retain the order of tokens, crucial for understanding sequences in tasks like natural language processing (NLP).

In practice, transformers consist of an encoder and a decoder, although models like GPT use only the decoder for tasks such as text generation. The encoder processes input data into contextualized representations, while the decoder generates output sequences based on this processed data. Transformers have been adapted to various domains. In speech-to-text, they process audio signals to generate text transcriptions by capturing contextual dependencies. In vision-language models, transformers integrate visual and textual data for tasks like image captioning and visual question answering. For text-to-speech, transformers convert text into natural-sounding speech by capturing prosody and intonation patterns.

To answer the sub-question, "How can the transformer architecture, the backbone of speech-to-text, vision-language, and text-to-speech models, be intuitively understood?"—the transformer architecture can be understood by recognizing its foundation in neural network principles, specifically those of the multi-layer perceptron, and its innovative use of self-attention mechanisms. This chapter provided an intuitive explanation for these concepts thereby answering this sub-question.

3

Open source models for HCI interface

This chapter answers the sub-research question: Among the open-source models available, which text-to-speech, vision-language, and speech-to-text models stand out as the best, and what are the unique mechanisms and innovations that drive their performance? It presents the selection of the speech-to-text, VLM and text-to-speech open-source algorithms that can replace the proprietary models within GazeGPT. For each of the chosen algorithms it additionally provides a deep dive into its inner workings, aiming to demystify otherwise magical seeming capabilities of these mathematical structures. In order to do so, it builds on chapter 2 as all these algorithms are based on deep learning and they all include the transformer in their model architecture.

3.1. Comparative Analysis for Speech-To-Text Technologies

To enable real-time conversations with an AI interface, efficient automatic speech recognition (ASR) is crucial. OpenAI released Whisper in 2022, a model noted for its state-of-the-art accuracy but also for its substantial computational demands [83]. This complexity results in high latency, meaning Whisper takes significant time to process audio. The delays experienced with existing assistants like Siri and Google Assistant underscore the frustration of such latency. For an AI system to be practical and useful in daily tasks, its transcription model must surpass Whisper's efficiency. Fortunately, researchers introduced an alternative in November 2023, known as Distil-Whisper, which offers a more efficient solution [84].

In the course the literature research, the study titled *A Comparative Analysis of Real Time Open-Source Speech Recognition Tools for Social Robots* came up to be a leading work in the field.[85] The insights of this recent study were followed.

When assessing speech recognition software, several key factors are critical to ensure that the selected system aligns with the application's requirements defined in section 1.2. The following were the criteria considered:

- **Accuracy:** Measured by the metric Word Error Rate (WER).
- **Latency:** The time required for the speech recognizer to process spoken input and return the text output.
- **Language Support:** The number of languages the speech recognition software can understand and process.
- **Offline Capability:** Whether the speech recognition can function without an internet connection.
- **Ease of Integration:** How simply the software can be incorporated into existing systems.
- **Resource Requirements:** The computational power and memory needed for the software to operate effectively.
- **Cost:** The financial implications of using the software, including initial setup and ongoing usage fees.

Given these criteria, five open-source speech recognition tools were examined: Google Speech Recognition, Distil-Whisper, DeepSpeech, Vosk, and WhisperAI.

Criteria	Google Speech	Distil-Whisper	Deep-Speech	Vosk	WhisperAI
Word Error Rate (WER)	42.3%	14%	High	51.7%	9%
Latency	Low	Low	Medium	Low	Medium
Language Support	Extensive	Limited	Good	Good	Good
Offline Capability	No	Yes	Yes	Yes	Yes
Ease of Integration	Moderate	Moderate	Moderate	Easy	Easy
Resource Requirements	Moderate	Moderate	High	Low	Moderate
Cost	Free	Free	Free	Free	Free

Table 3.1: Evaluation of Speech Recognition Software

Based on the evaluation, selecting the software involved a balance between accuracy, functionality, and operational constraints. WhisperAI, with the lowest WER, was exceptional in terms of accuracy. However, Distil-Whisper was chosen for its faster execution time, despite its higher WER compared to WhisperAI. Vosk has a similar low latency, however its WER is much worse. As mentioned previously, the focus is on optimizing latency. During the initial development phase, WhisperAI will be used due to its ease of integration and will be replaced by Distil-Whisper once the minimal working example works.

This evaluation exemplifies that the superior software is not determined solely by accuracy metrics but also by how well it fits the specific constraints and aims of the project.

3.2. Distil-Whisper

Distil-Whisper, a model introduced by Hugging Face sponsored by Google and detailed in their publication dated November 1, 2023, represents an advancement in the field of Speech to Text (STT) technologies. As a distilled version of OpenAI's Whisper speech recognition model, Distil-Whisper aims to address the challenges posed by the deployment of large-scale STT technologies in environments constrained by latency.[84] Distillation, along with the other main processes needed to obtain the Distil-Whisper model from the original Whisper model are described in this section to equip the reader with an understanding of the technology. The section on the model architecture presents the link between this section and chapter 2, as this model is based on the original transformer architecture presented in figure 2.23.

The architecture of the Distil-Whisper model is presented in figure 3.1. It consists of an encoder-decoder transformer just like the original transformer presented in figure 2.23. The next subsections follow the same structure as section 2.3 to guide the reader through the process of transcribing speech using Distil-Whisper as the model architecture.

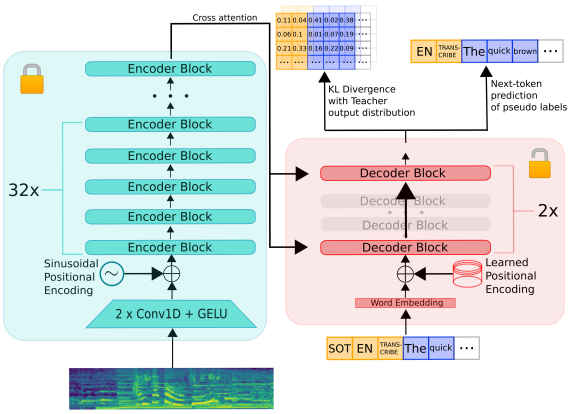

Figure 3.1: Architecture of the Distil-Whisper model. The encoder (shown in green) is entirely copied from the teacher to the student and frozen during training. The student's decoder consists of only two decoder layers, which are initialised from the first and last decoder layer of the teacher (shown in red). All other decoder layers of the teacher are discarded. The model is trained on a weighted sum of the KL divergence and PL loss terms.[84]

3.2.1. Tokenization

As highlighted in section 2.3.2 transformers exist of sequence to sequence models. Speech or audio data consists of a digitized sampled audio signal of 24000Hz. Section 2.3.4 highlighted the context window limitations, for GPT 3 this context window was limited to 2048.[57] First, the input speech needs to be tokenized. This process starts by splitting the audio fragments into 30 second chunk lengths. A fast Fourier transform (FFT) is applied to find the frequency

As discussed in section 2.3.14, the original transformer model was used to translate language. It used an encoder to encode the french sentence to generate keys designed to answer the queries from the encoder. The combination of the keys and queries then resulted in an attention map based on which the decoder could base its attention on specific words within the input french sentence to generate the next token for the English translation.

In the case of ASR transformer models, the encoder is used to produce the cross-attention keys for the decoder. The encoder

Audio is fundamentally a pressure wave propagated through air over time. When an individual speaks, they modulate air pressure, causing nearby air molecules to compress or expand. This modulation can be captured as a one-dimensional signal that varies over time, illustrating variations in loudness and pitch. For example, in a vinyl record, audio is encoded as a continuous groove on the disc's surface, with variations in the groove's height representing different sounds. This encoding allows a stylus reading the disc to reproduce the original audio, demonstrating that complex audio can be reduced to a simple, continuous one-dimensional signal. [86] The audio recordings relevant for speech recognition therefore primarily use 16 bit depth.

In modern digital systems, the continuous speech audio signal is converted into a digital format through a process known as discretization. This process involves sampling the continuous waveform at regular intervals and quantizing these samples into a set of discrete values, typically represented in binary code. This conversion is crucial as digital systems, including computers and digital signal processors, handle only discrete numerical values. The sampling rate, commonly 44,100 samples per second for CD-quality audio, is dictated by the Nyquist theorem. This theorem states that the sampling frequency must be at least twice the highest frequency to be recorded, ensuring each waveform cycle is captured by at least two samples—one at the peak and one at the trough. Thus, with a highest audible frequency of around 20kHz, the Nyquist frequency is achieved, capturing each waveform cycle adequately at this rate.

The bit depth, which determines the number of possible amplitude levels each sample can hold, varies by application. For audio recordings relevant to speech recognition, a 16-bit depth is primarily used, providing a dynamic range of 96dB. This bit depth is sufficient for capturing the nuances of speech without requiring the larger file sizes associated with higher bit depths. In contrast, action movies may

use a bit depth of 24 to accommodate a wider dynamic range of up to 144dB, essential for capturing both subtle conversations and loud explosions effectively. The choice of bit depth is therefore dependent on the required dynamic range and amplitude resolution of the audio content. [87]

As outlined in section 2.3.2, a transformer functions as a sequence-to-sequence model, necessitating sequences of input to produce sequences of output. A digital audio signal, as described, fits this description as it comprises a continuous stream of audio samples. However, feeding such high-resolution audio directly into a transformer network is not feasible. As explained in section 2.3.2, transformers process tokens. In the realm of text, Byte Pair Encoding (BPE) is employed to generate tokens from text, and as section 2.3.4 outlines, the context window for GPT-3 is 2048 tokens. Given a standard sampling rate of 44,100 samples per second for speech audio, this context window would encode just about 0.04644 seconds of audio. Even the most advanced version of GPT-4, featuring the largest context window of 128,000 tokens, could only process a 2.9-second audio signal. [54] This length is an order of magnitude shorter than the 30-second segments typically used in Automatic Speech Recognition (ASR) models. [83] Due to the need for longer durations to capture meaningful speech, these high-resolution audio samples are not directly used as tokens in the transformer.

Instead, the fourier tranform of the audio signal is used. The Fourier Transform decomposes a signal into sine waves, each defined by unique amplitude and frequency, enabling quantification of each frequency's presence in the signal. This is achieved by multiplying the signal by a sine wave of a specific frequency and integrating the product over time. A zero integral indicates absence, whereas a non-zero integral indicates presence and amplitude of that frequency.

When a single-frequency sine wave is analyzed without prior knowledge of its composition, testing it against various frequencies shows that only when the test frequency matches the signal's, the integral is positive, revealing the original frequency. Fourier transforms also involve cosine waves, allowing for phase analysis of the signal by multiplying it by both sine and cosine waves at each frequency.

Euler's formula is used to simplify these calculations into a single complex exponential function, splitting the results into real and imaginary parts to represent cosine and sine amplitudes, respectively.

Real-world signals require the Discrete Fourier Transform (DFT), which adapts the Fourier Transform to discrete data by segmenting the frequency spectrum based on the signal's sampling attributes. The sample spacing and duration influence the maximum detectable frequency and resolution.

In applications like processing speech data in Whisper, a DFT example with eight samples would create eight frequency bins. Each bin corresponds to a frequency, with its presence and amplitude in the signal determined by the calculations for both sine and cosine waves.

The DFT's computational burden, traditionally $O(n^2)$, becomes impractical for large datasets. The Fast Fourier Transform (FFT) addresses this by reducing the complexity to $N \log N$ through strategic decomposition of the signal into even and odd components and utilizing overlaps in sinusoidal functions. This reduction in computational demands is crucial for efficiently analyzing large-scale data in applications such as Whisper.

To further refine the analysis, these frequency components can be mapped onto the Mel scale during processing. The Mel scale is designed to reflect the logarithmic perception of frequency by human ears, where equal distances on the scale correspond to perceptually equal frequency differences.

Further downsampling occurs through the application of a convolutional layer, which processes the spectrogram by reducing its dimensionality even more.

3.2.2. Pseudo labeling

To obtain the distilled model, the authors employs pseudo labeling, a method where a high-performing model labels data based on its predictions. [88] Referring to the example of handwritten digit recognition from section 1.1, consider a system that can recognize handwritten digits with human-level accuracy or better. Such a model can label unlabeled data to provide labeled datasets for training another handwritten digit recognition system, saving enormous amounts of human labor in the process. This technique is used by the creators of Distil-Whisper, where the large, original high-performance Whisper model is utilized to generate speech-to-text labels.

3.2.3. Knowledge distillation

Knowledge distillation is described as a technique where a larger "teacher" model is used to train a smaller "student" model.[89] The objective is to enable the student model to mimic the teacher model's performance closely. The concept of knowledge distillation is gaining traction across various domains. Notably, Large language models such as Bert, and image segmentation models such as the "Segment Anything Model," have also undergone the distillation process.[90]. Section 3.2.2 discussed the generation of pseudo labels by the larger "teacher" model. In the case of Distil-Whisper, only these pseudo labels are used to train the smaller student model. Where the original model consisted of 1.5 billion parameters, the resulting distilled model utilizes 51% fewer parameters and achieves a substantial increase in processing speed being 5.8 times faster.

In the context of knowledge distillation, a larger model with greater capacity tends to exhibit a higher propensity for overfitting, particularly when the dataset is small. This increased likelihood of overfitting can sometimes be mitigated by distilling the model into a smaller version. Surprisingly, these smaller models often demonstrate improved generalization compared to their larger counterparts. This phenomenon suggests that distillation might implicitly enhance the model's ability to generalize from limited data. [89] [84]

The training of the model incorporates a combination of Kullback-Leibler (KL) Divergence and Perplexity Loss (PL) metrics. We will delve deeper into these loss functions shortly.

To train the Whisper model, a dataset composed of audio-text pairs is utilized. This model is trained using the standard cross-entropy loss, a common technique in supervised learning. Despite suggestions to the contrary, this method is clearly supervised, as each training instance consists of a paired audio and text segment. The confusion about it being unsupervised arises from the use of pseudo labels generated by another model, yet fundamentally, it remains an instance of supervised learning.

In this training process, the model is designed to predict specific instances by maximizing the estimated probability of the correct class label, which corresponds to the desired word or token. Essentially, this is akin to a classification task where each output, such as a word or token, represents a class out of many possible options. For instance, the model predicts the probability of each potential token—be it 'cat,' 'dog,' etc.—where the 'instance class' is the predicted token and the 'target class' is the correct token that the model aims to predict.

Knowledge distillation is recognized as a compression technique where a smaller, simplified student model is trained to emulate the behavior of a larger, more complex teacher model. According to the authors, this process represents a form of data compression, as discussed in [84]. Broadly, all neural networks, including the multi-layer perceptron detailed in sections 2.1, 2.2.1 and 2.2.2, compress training data into their weights and biases. Unlike methods that minimize cross-entropy loss between predictions and actual labels, such as the multi-layer perceptron in section 2.2, knowledge distillation utilizes the full predictive distribution from the teacher model, as outlined in section 3.2.2. This distribution encompasses the entire range of potential next tokens, offering a more nuanced approach than the binary categorizations typical of true labels.

Standard supervised learning methods train models to assign a high probability to the correct label while dismissing all other possibilities, which simplifies training but fails to capture the breadth of possible outcomes. For instance, in speech recognition, assigning a probability of one to 'new' while ignoring 'knew' overlooks nuanced distinctions between similar-sounding words and their related meanings, which a speech-to-text transformer could learn to differentiate.

In contrast, knowledge distillation feeds an input, such as audio, to a teacher model that then outputs a probability distribution across all possible tokens. The student model is trained not only to predict the most likely token but to approximate the full distribution provided by the teacher. This method enriches the training process by incorporating a broad spectrum of potential outcomes, enhancing the model's predictive capabilities.

The success of knowledge distillation critically depends on access to the teacher model's complete output. If one were attempting to distill knowledge from an inaccessible model, like a hypothetical 'GPT-5' available only via API, they would be limited to using one-hot encodings instead of the rich

distributions available through direct access to the model's outputs. This would significantly restrict the depth and effectiveness of the distillation process.

Kullback-Leibler (KL) Divergence is a fundamental concept in machine learning, extensively utilized to measure the difference between two probability distributions. Notably, KL Divergence is asymmetric, meaning that the divergence from distribution P to Q ($D_{KL}(P \parallel Q)$) is not equal to the divergence from Q to P ($D_{KL}(Q \parallel P)$). This asymmetry indicates that the order in which distributions are compared affects the resulting divergence value.

An illustrative example can demonstrate this asymmetry: consider two distributions, one depicted in green and the other in blue. When P is the reference and Q is the comparative distribution, the divergence value may differ significantly from when Q serves as the reference. This discrepancy becomes particularly evident when the distributions have non-symmetric properties or when one distribution partially overlaps the other.

Furthermore, in the context of machine learning, KL Divergence is used to align the probabilistic outputs of two models, typically in scenarios involving a teacher-student model setup. Here, the student model's probability distribution is trained to match that of the teacher model by minimizing the KL Divergence across the entire set of possible next tokens. This approach can be seen as a word-level application of KL Divergence, where knowledge is transferred from the teacher to the student model through the logits corresponding to potential tokens.

The utility of KL Divergence is particularly evident as it provides comprehensive insights across all classes, yielding less variance in gradients compared to other loss metrics such as categorical cross-entropy. This characteristic makes KL Divergence an attractive choice for tasks requiring robust probability distribution matching.

To optimize the model, KL Divergence is incorporated into the loss function. This metric is illustrated by comparing a pseudo label, generated by the teacher model, with the probability distribution approximation from the student model. Each represents probabilities assigned to every token in the model's vocabulary.

KL Divergence calculation begins by evaluating the ratios of the approximated probabilities to those of the pseudo labels and then averaging these ratios. However, this method poses a challenge: instances where the approximated probability significantly exceeds the pseudo label probability are weighted more heavily than those where the approximated probability is considerably lower. To address this imbalance, KL Divergence instead averages the logarithms of these ratios. Since the logarithm function (log) is reciprocal (i.e., $\log(a) = -\log(1/a)$), overestimations are penalized as severely as underestimations. The logarithm of one is zero, meaning no penalty is applied when the predicted probability precisely matches the pseudo label, which is a logical approach.

Further refining this calculation, the log of the ratio is multiplied by the pseudo label probabilities. This step effectively scales up the influence of higher pseudo label probabilities, assigning greater importance to them. This weighting is advantageous because it encourages the student model to focus more on the relationships between the correct word and its close variants rather than between unrelated words. For example, considering the word 'new', the pseudo label likely assigns a higher probability to 'knew' than to 'old'. The KL Divergence thereby directs the student model's attention towards the similarity between 'new' and 'knew' rather than between 'new' and 'old', compelling the model to discern and learn from phonetic and contextual similarities between words. With this the KL-divergence is mathematically written using equation 3.1.[91]

$$D_{KL}(\text{pseudo label} \parallel \text{student approximation}) = \sum_i \text{pseudo label}_i \cdot \log\left(\frac{\text{pseudo label}_i}{\text{student approximation}_i}\right) \quad (3.1)$$

600.000 hours vs 21000 hours training

3.2.4. Word Error Rate

Word Error Rate (WER) is an important metric for evaluating the performance of speech recognition systems, quantifying the accuracy of the transcribed text compared to the spoken words. Calculating WER involves identifying three types of transcription errors: substitutions, insertions, and deletions.

Substitutions occur when one word is inaccurately transcribed as another; for instance, if "cat" is spoken but "cap" is recorded. Insertions involve the addition of extra words not originally spoken; for example, if the phrase "I am home" is transcribed as "I am at home." Deletions happen when words that were spoken are left out of the transcription, such as "she is dancing" being recorded as "she dancing."

To compute WER, the total count of these errors is summed and then divided by the total number of words that were originally spoken. This calculation provides a numerical value, indicating the error rate of the transcription and reflecting how accurately the speech recognition system has performed. [92]

The performance of the distilled model closely approximates that of the original Whisper model, with a deviation of only 1% in word error rate on data that was not used during training such as people with different accents.

3.2.5. Speculative decoding

As shown in section 3.2.3, the distil-whisper model can be used inidividually to achieve a 5.8 times faster speech transcription generation. However, this speed up comes at a cost of a 1% increase in word error rate as stated in section 3.2.4. To find a better trade off between model speed up and transcription accuracy, the paper suggests the use of speculative decoding.

In the speculative decoding approach the original whisper model (figure 3.2) and the distilled version work in parallel. Comparing figure 3.2 and figure 3.1, the encoders are the same. This is intuitive, because for every audio sample, the encoder is ran once, while the decoder has to be ran 128 times. The decoder takes 90% of the computation time needed for the transcription of one audio sample and thus is the bottleneck in this process.[84] The decoder has to be ran this many times, because it works just like the autoregressive word generation of GPT-3 where each predicted word is added to the input prompt. Based on the encoded audio, it predicts the most likely first word in the audio, then it uses the encoded audio and this first prediction to make the prediction for the second word etc. So for a single audio sample, the audio only needs to be encoded once by either one of the models. Comparing the decoders, the distil-whisper model only contains the first and last layer of the original decoder, hence the speed up. With this, the idea is to run the smaller model x more times in an autoregressive fasion. The focus of this setup is to enable speculative decoding, which algorithmically ensures that predictions remain consistent with the predictions of the larger model. [93]

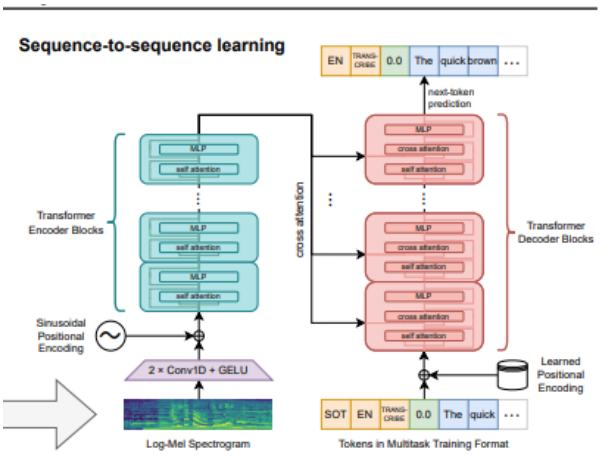

Figure 3.2: The architecture of the original whisper model [83]

The speculative decoding algorithm leverages a smaller model, in this case, distil-whisper, which is typically adept at handling common words reliably. It is optimal for this model to predict four tokens, as supported by recent studies [93] [94]. For instance, during a speech-to-text task, it might predict the

phrase "The cats did accept." The first three words are predicted accurately due to their straightforward acoustic properties, while the fourth word, "accept," presents a greater challenge, potentially due to its homophone "except."

The larger model proceeds to predict the next word based on these four predictions and simultaneously evaluates the accuracy of these predictions using the probability distributions from the last layer of its decoder. If the prediction for "accept" does not align with the expected probabilities, the larger model rejects this particular word while accepting the first three. This selective validation by the larger model ensures precision where necessary without compromising speed, intervening only when the output from the distilled model diverges from expected results. This approach enables a fine balance between maintaining high accuracy and operational efficiency in speech-to-text decoding.

The encoder, highlighted in green in figure 3.1, is directly transferred from the teacher model to the student model and remains unaltered during the training process. The student's decoder is simplified to consist of only two layers, specifically the first and last decoder layers of the teacher model. All remaining decoder layers from the teacher are not included in the student model, see figure 3.1.

The performance of in-distribution (ID) short-form audio does not seem to differ between whisper-medium and distil-whisper-medium, which corresponds to a Word Error Rate (WER) of 14.

3.3. Comparative Analysis for Vision Language Models

This subchapter delves into the comparative analysis of some of the most advanced AI language models: ChatGPT 3.5, ChatGPT 4, Bard, LLaMA, and Claude, which act as our conversation manager. This component is the brain of the conversation, interpreting the transcribed text and maintaining the context of the conversation.
The following criteria have been established to evaluate and compare the performance and capabilities of the different AI language models:

- **Accuracy:** The exactness of the model's responses to factual information and user prompts.
- **Ease of Use:** The user's ability to interact with and integrate the model into various platforms.
- **Cost:** The economic considerations involved in deploying and maintaining the model.
- **Integration:** The model's compatibility with other software and systems.
- **Response Time:** The delay between the user's request and the model's reply.
- **User Experience:** The overall satisfaction of users based on the model's interaction quality.
- **Creativity:** The model's capacity for producing innovative and imaginative responses.

Criteria	ChatGPT4V	CogVLM	LLaVA 1.5	LLaVA 1.6
Accuracy	Good	Good	Moderate	Good
Ease of use	Good	Moderate	Moderate	Good
Integration	Good	Moderate	Moderate	Good
Response time	Good	Moderate	Moderate	Good
User Experience	Good	Moderate	Limited	Good
Creativity	Good	Moderate	Limited	Moderate

Table 3.2: Evaluation of open-source VLM's Providers

To assess the models, the paper models were used and studied for critical evaluation.[95][96][97][98] Considering the specific requirements of our real-time conversational system as evidenced in our code, ChatGPT 4 proved to be the best choice among the AI language models evaluated. Its performance metrics, particularly in accuracy and the generation of contextually relevant responses, aligned very well with our system's architecture. The human-like responses and creativity of ChatGPT 4's interactions were especially crucial for the dynamic and varied queries our application is designed to handle. Furthermore, its scalability ensures that as our system grows and the complexity of user interactions increases, ChatGPT 4 will continue to provide intelligent conversational capabilities. The assesment is summarized in table 3.2.

However, ChatGPT 4 is not open-source and therefore cannot be chosen for this study, which aims to identify an open-source alternative for the Vision-Language Model (VLM) used in GazeGPT. Despite its superior performance across all evaluated criteria, the need for an open-source solution necessitated the evaluation of alternatives like CogVLM and LLaVA variants.

The evaluation criteria included accuracy, ease of use, integration, response time, user experience, and creativity. ChatGPT 4 consistently received high marks, demonstrating superior performance in all these areas. For instance, in terms of accuracy, both ChatGPT 4 and CogVLM performed well, but LLaVA 1.5 was rated moderate. LLaVA 1.6, however, matched ChatGPT 4 and CogVLM in accuracy, showing significant improvement.

Ease of use is another critical factor, where both ChatGPT 4 and LLaVA 1.6 excelled, while CogVLM and LLaVA 1.5 were found to be moderately user-friendly. Integration capabilities followed a similar pattern, with ChatGPT 4 and LLaVA 1.6 integrating seamlessly into existing systems compared to the moderate performance of CogVLM and LLaVA 1.5.

Response time is vital for real-time applications, and here, ChatGPT 4 and LLaVA 1.6 again outperformed the others, offering quick processing speeds necessary for fluid interaction. User experience was rated good for ChatGPT 4 and LLaVA 1.6, indicating a high level of user satisfaction, while CogVLM was moderate, and LLaVA 1.5 was limited.

Lastly, creativity in generating diverse and engaging responses was rated good for ChatGPT 4 and moderate for LLaVA 1.6, with CogVLM and LLaVA 1.5 showing limited creativity. This comprehensive evaluation underscores the balanced and robust capabilities of ChatGPT 4 and highlights the improvements seen in LLaVA 1.6 over its predecessor. LLaVA 1.6 performed slightly better than CogVLM in most criteria, making it a more favorable open-source alternative.

As a result the next section discusses the LLaVA architecture to gain intuitive understanding.

3.4. LLaVA

The LLaVA model, known as Large Language and Vision Assistant, represents a significant advancement in vision language models, particularly following the release of GPT-4 with vision capabilities in September 2023. Developed by Howan Leu and Chunan Lee, LLaVA quickly emerged as a leading model in this field, with its initial version outlined in the "Visual Instruction Tuning" paper released on April 17th, 2023.[96] This model was further refined, culminating in the release of LLaVA 1.5 on October 5th, 2023, which achieved state-of-the-art results across 11 different benchmarks with just about a day's training on a single A100 node.

The architecture of LLaVA consists of three core components: the Large Language Model (LLM), the Vision-Language Connector (an MLP), and the Vision Encoder (like CLIP). The LLM serves as the backbone for comprehending user instructions and generating responses, while the Vision-Language Connector bridges the gap between visual and textual domains. This MLP replaces the simpler linear projection originally used, enhancing the model's ability to understand and translate between visual inputs and their linguistic representations. The Vision Encoder, such as OpenAI's CLIP (Contrastive Language-Image Pre-training), encodes visual features from input images, providing the necessary visual understanding for the model to process and reason about images.

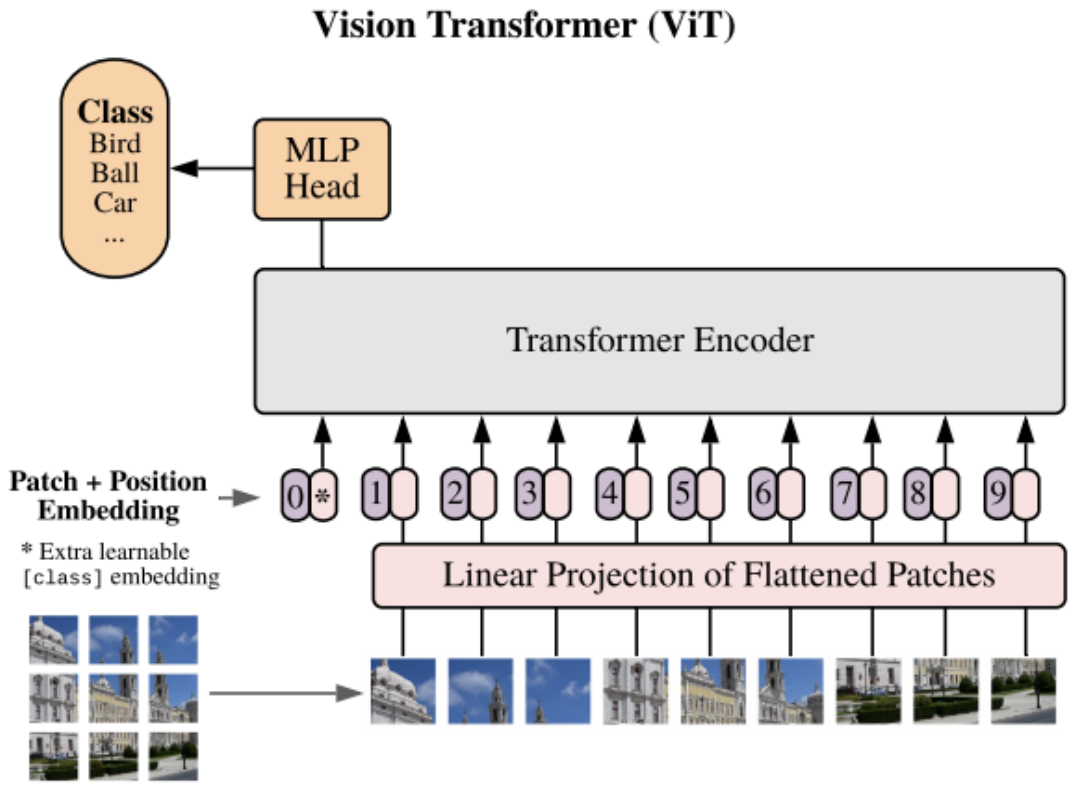

Figure 3.3: The Vision Transformer (ViT) architecture treats image patches as a sequence of tokens, processed similarly to word tokens in a language model.[99]

The Vision Encoder is based on the Vision Transformer (ViT) variant of CLIP, which processes images by segmenting them into patches and treating them as sequences of tokens, similar to words in a sentence (see Fig. 3.3). This method enables the model to handle complex multimodal tasks, including detailed descriptions and conversational-style question answering, with remarkable data efficiency and performance. CLIP uses pairs of text and images, employing contrastive loss during training to align related text-image pairs closely while distancing unrelated pairs. This technique forms an embedding space where related text and images are close together, enhancing the model's ability to understand and process multimodal inputs effectively.

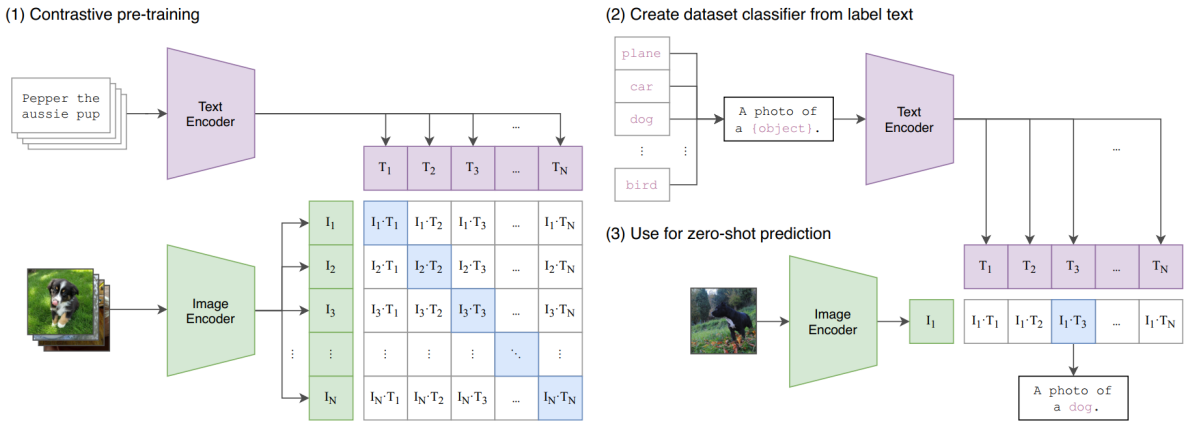

Figure 3.4: CLIP pre-trains an image encoder and a text encoder to predict which images were paired with which texts in the dataset, creating a shared embedding space for images and texts.[100]

The development of LLaVA leverages pre-trained models such as CLIP and GPT-4, which contribute significantly to its capabilities. While LLaVA can be rapidly trained, typically within a single day on an 8-A100 node, this does not include the extensive pre-training invested in foundational models like CLIP. This reliance on pre-trained models is a common trend in deep learning, where new models are often built upon existing ones, fine-tuning rather than training from scratch.

LLaVA's architecture and training methodology highlight the importance of integrating vision and language processing to achieve advanced AI capabilities. As an open-source large multimodal model, LLaVA is particularly attractive for its accessibility and potential for fine-tuning on custom datasets, making it a versatile tool for various applications. Its development underscores the significance of foundational research and pre-trained systems in achieving rapid and efficient advancements in AI.

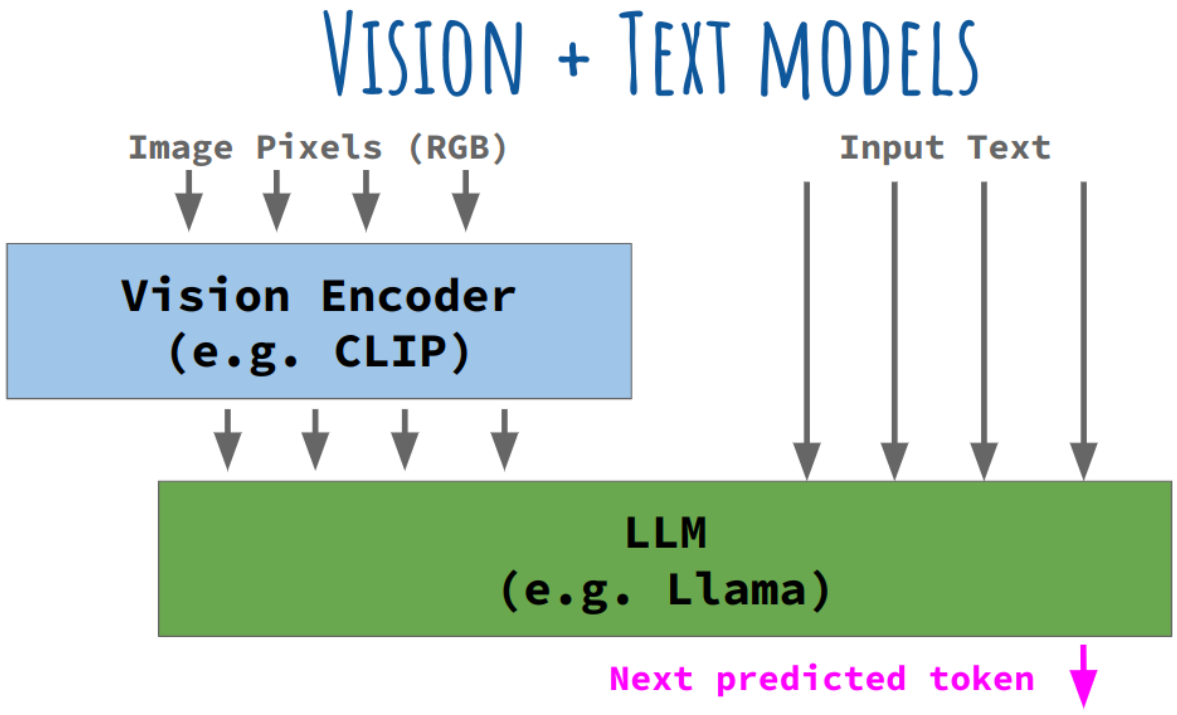

Figure 3.5: Architecture of a Vision Language Model (VLM), combining vision and language inputs for comprehensive multimodal understanding.

In summary, the LLaVA model combines the strengths of pre-trained vision and language models to handle complex multimodal tasks effectively. By leveraging the capabilities of models like CLIP and GPT-4, LLaVA provides a robust and efficient solution for applications requiring the integration of visual and textual information (see Fig. 3.5).

3.5. Comparative Analysis for Text-To-Speech Technologies

During the project, an independent research initiative was undertaken to identify a text-to-speech (TTS) provider capable of delivering a seamless and realistic conversational experience. Due to limited research available, a comprehensive testing of various TTS providers was conducted. The evaluation aimed to find a service that could overcome the uncanny valley often associated with synthetic voices, particularly in producing filler sounds and expressing emotions such as "uhm," "ehm," "ah," and laughter.

Leading TTS providers, including StyleTTS, Tortoise and the original ElevenLabs were examined through rigorous testing across several critical parameters.[101][102][13] These included the naturalness of the voice, expressiveness, ability to handle filler sounds naturally, diversity of voice options, capability for real-time streaming, ease of integration into our systems, and cost efficiency. The table 3.3 summarizes the results of the comparative analysis, highlighting ElevenLabs' superior performance and StyleTTS2 as the best open-source alternative.

Criteria	Tortoise	StyleTTS2	ElevenLabs
Naturalness of Voice	Moderate	Good	Excellent
Expressiveness	Limited	Moderate	Superior
Filler Sound Naturalness	Poor	Moderate	Excellent
Diversity of Voices	Moderate	Moderate	Extensive
Real-time Streaming	No	Limited	Yes
Ease of Integration	Good	Good	Very Good
Cost Efficiency	Good	Good	Good

Table 3.3: Evaluation of TTS Providers

Despite advancements in technology made by each provider, a robotic undertone was still present, particularly with the automated human-like sounds. This deviated from the conversational realism sought to be achieved. ElevenLabs provided a breakthrough with its extensive library of voices that closely emulate human tonality and expressiveness, earning it an "Excellent" score for naturalness of voice. Its superior expressiveness and excellent filler sound naturalness were also notable, making it stand out from other providers. Furthermore, the customizability offered by ElevenLabs allowed for tailoring the voice to the unique personality of the AI, and its capability for real-time streaming was crucial, ensuring responses with minimized latency and maintaining the natural flow of dialogue.

However, ElevenLabs is a proprietary model, and one of the goals of this literature review is to find an open-source replacement. As a result, an open-source model like StyleTTS2, which demonstrated the best performance among open-source alternatives, is better than Tortoise in the given criteria [101]. It is therefore considered a promising candidate. StyleTTS2 showed good naturalness of voice, moderate expressiveness, and the ability to handle filler sounds relatively well [102]. While it does not match the performance of ElevenLabs, it offers a viable open-source solution. Tortoise, another open-source model, did not perform as well, scoring "Moderate" for naturalness and expressiveness, and "Poor" for filler sound naturalness, making StyleTTS2 the more promising candidate.

In conclusion, while ElevenLabs provided excellent performance and was previously used by GazeGPT, the search for an open-source replacement has identified StyleTTS2 as the best alternative. StyleTTS2 stands out as the most promising open-source model available and represents a promising direction for future research and development. Additionally, the complex architecture of models like StyleTTS2, which involves diffusion, should be explored further to provide an intuitive explanation in future research.

3.6. Discussion

This chapter aimed to address the sub-research question: Among the open-source models available, which text-to-speech (TTS), vision-language (VLM), and speech-to-text (STT) models stand out as the best, and what are the unique mechanisms and innovations that drive their performance? Through an in-depth comparative analysis, we evaluated various models to determine the most suitable open-source alternatives for GazeGPT, focusing on their performance, architectural advancements, and practical applications.

In the evaluation of STT models, we considered several critical factors such as accuracy, latency, language support, offline capability, ease of integration, resource requirements, and cost. Distil-Whisper emerged as a noteworthy alternative due to its balance between efficiency and accuracy. Despite having a slightly higher Word Error Rate (WER) compared to WhisperAI, Distil-Whisper's reduced computational demands and faster execution time make it a viable choice for real-time applications where latency is critical. Key innovations like knowledge distillation, which involves training a smaller model using pseudo-labels generated by a larger model, and speculative decoding, which combines the predictions of both models to balance speed and accuracy, significantly enhance Distil-Whisper's efficiency. These mechanisms make Distil-Whisper a strong candidate for replacing proprietary models in GazeGPT.

For vision-language models, the comparison focused on ChatGPT4V, CogVLM, LLaVA 1.5, and LLaVA 1.6, evaluated on criteria such as accuracy, ease of use, integration, response time, user experience, and creativity. LLaVA 1.6 demonstrated notable improvements over its predecessors and alternatives, particularly in terms of accuracy, user experience, and integration capabilities. The Vision Transformer architecture, which processes images as sequences of patches similar to tokens in a language model, enhances LLaVA's ability to understand and generate contextually relevant responses based on visual inputs. Additionally, leveraging pre-trained models like CLIP for the vision encoder and GPT-4 for the language model allows LLaVA to achieve state-of-the-art performance with relatively short fine-tuning periods. These innovations contribute to LLaVA's effectiveness in handling complex multimodal tasks, making it a promising open-source solution for GazeGPT's vision-language requirements.

The assessment of TTS models included Tortoise, StyleTTS2, and ElevenLabs. Although ElevenLabs offered superior naturalness and expressiveness, its proprietary nature necessitated the exploration of open-source alternatives. StyleTTS2 emerged as the best open-source candidate, demonstrating good naturalness of voice and moderate expressiveness. StyleTTS2's architecture involves diffusion processes that gradually refine the audio output, enhancing the naturalness and fluidity of the generated speech. Despite not matching the proprietary models in expressiveness, StyleTTS2 offers considerable flexibility for customization, which is crucial for tailoring the voice to specific applications. These features make StyleTTS2 a viable open-source alternative for TTS in GazeGPT, particularly when customization and open-source benefits outweigh the slight performance compromises.

In conclusion, the comparative analysis conducted in this chapter identified Distil-Whisper, LLaVA 1.6, and StyleTTS2 as the leading open-source models for speech-to-text, vision-language, and text-to-speech tasks, respectively. Each model's unique mechanisms and innovations, such as knowledge distillation, Vision Transformer architecture, and diffusion processes, contribute significantly to their performance. These models provide a robust foundation for developing an open-source version of GazeGPT, capable of delivering efficient, accurate, and contextually relevant interactions in human-computer interfaces. Future research should focus on further refining these models and exploring additional innovations to enhance their capabilities and integration into practical applications.

4

Discussion

This last chapter aims to answer the main research question: " ". After attempting to answer this research question in section 4.1, the future work needed to further answer the research question is stated in section 4.2.

4.1. Discussion

The main research question seeks to explore how open-source speech-to-text, response generation, and text-to-speech models can be leveraged to develop a real-time, accessible, and human-like GazeGPT, while providing an intuitive understanding of their underlying architecture to promote the democratization of AI. To address this, we must consider the detailed findings from the chapters on open-source models and the transformative impact of the transformer architecture in deep learning.

Leveraging open-source models such as Distil-Whisper for speech-to-text, LLaVA for vision-language tasks, and StyleTTS2 for text-to-speech, GazeGPT can achieve a balance of efficiency, accuracy, and accessibility. These models have been identified as the best open-source alternatives based on rigorous evaluation criteria, including their performance, computational efficiency, and ease of integration into existing systems.

Distil-Whisper stands out due to its innovative use of knowledge distillation, which allows it to maintain high accuracy while being computationally efficient. This efficiency is crucial for real-time applications, reducing latency and enhancing the user experience. The model's speculative decoding mechanism further optimizes its performance, ensuring that speech transcriptions are both accurate and generated quickly.

LLaVA, a vision-language model, utilizes the Vision Transformer (ViT) architecture to handle complex multimodal tasks. By processing images as sequences of tokens and integrating them with textual data, LLaVA can generate contextually relevant and accurate responses. This capability is essential for creating a responsive and intelligent conversational agent like GazeGPT, which needs to understand and interact with users based on both visual and textual inputs.

StyleTTS2 offers a promising solution for text-to-speech with its diffusion-based architecture that enhances the naturalness and fluidity of generated speech. While it may not match the expressiveness of proprietary models, its flexibility and customization potential make it a strong open-source alternative. This ensures that GazeGPT can produce human-like, engaging, and emotionally expressive speech, crucial for creating a realistic conversational experience.

The underlying architecture of these models, primarily based on transformers, can be intuitively understood by tracing the evolution of deep learning. Starting from the perceptron in the 1950s to the development of the multi-layer perceptron (MLP) and the breakthrough of transformers, the field has seen significant advancements. The transformer architecture, introduced in "Attention is All You Need," employs a self-attention mechanism that captures contextual dependencies within the input data. This mechanism involves generating queries, keys, and values for each input token, computing dot products,

and using the softmax function to produce attention scores. These scores guide how much influence each token has on others, capturing long-range dependencies and contextual relationships.

Transformers also use multi-headed attention to capture various relationships within the data and positional encoding to retain the order of tokens, crucial for understanding sequences. The encoder-decoder structure of transformers, with the encoder processing input data into contextualized representations and the decoder generating output sequences, has been adapted to various domains, including speech-to-text, vision-language, and text-to-speech tasks.

By leveraging these open-source models and understanding their architecture, GazeGPT can be developed into a real-time, accessible, and human-like conversational agent. The use of efficient and accurate models like Distil-Whisper, LLaVA, and StyleTTS2 ensures that GazeGPT can interact seamlessly with users, providing a natural and engaging experience. Furthermore, the intuitive explanation of their underlying architecture promotes the democratization of AI, making advanced AI technologies more accessible and understandable to a broader audience.

In conclusion, the integration of these open-source models, underpinned by the transformative power of the transformer architecture, enables the development of GazeGPT as a sophisticated, real-time conversational agent. This approach not only meets the technical requirements but also aligns with the broader goal of making AI technologies more accessible and transparent.

4.2. Future work

For future work, a deeper understanding of the mechanisms behind text-to-speech models involving diffusion processes is essential. This includes studying their mathematical foundations and practical implementations to optimize both speed and quality, making the generated speech more human-like and engaging.

References

[1] A. Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 5998–6008.

[2] Tom B. Brown et al. "Language Models are Few-Shot Learners". Version v4. In: *arXiv preprint arXiv:2005.14165* (July 2020). Submitted on 28 May 2020, last revised 22 Jul 2020. eprint: 2005.14165 (cs.CL).

[3] OpenAI. *GPT-4V(ision) System Card*. https://openai.com/research/gpt-4v-system-card. Discusses the integration of vision capabilities into GPT-4, enabling the model to analyze and interpret image inputs. Sept. 2023.

[4] Robert Konrad et al. "GazeGPT: Augmenting Human Capabilities using Gaze-contingent Contextual AI for Smart Eyewear". Version v2. In: *arXiv preprint arXiv:2401.17217* (Jan. 2024). Submitted on 30 Jan 2024, last revised 31 Jan 2024. eprint: 2401.17217 (cs.HC).

[5] OpenAI. *DALL·E: Creating images from text*. We've trained a neural network called DALL·E that creates images from text captions for a wide range of concepts expressible in natural language. Jan. 2021.

[6] OpenAI. *ChatGPT: Optimizing Language Models for Dialogue*. OpenAI Blog. 2022. URL: https://openai.com/blog/chatgpt.

[7] Long Ouyang et al. "Training Language Models to Follow Instructions with Human Feedback". In: *arXiv preprint arXiv:2203.02155* (Mar. 2022). Submitted on 4 Mar 2022. eprint: 2203.02155 (cs.CL).

[8] Jason Wei et al. "Finetuned Language Models Are Zero-Shot Learners". Version v5. In: *arXiv preprint arXiv:2109.01652* (Feb. 2022). Submitted on 3 Sep 2021, last revised 8 Feb 2022. eprint: 2109.01652 (cs.CL).

[9] S. J. Shaikh. "Artificially Intelligent, Interactive, and Assistive Machines: A Definitional Framework for Intelligent Assistants". In: *International Journal of Human–Computer Interaction* 39.4 (2022), pp. 776–789. DOI: 10.1080/10447318.2022.2049133.

[10] *ChatGPT Can Now See, Hear, and Speak*. https://openai.com/index/chatgpt-can-now-see-hear-and-speak/. 2024.

[11] Peter-Paul Verbeek and Olya Kudina. "Ethics From Within: Google Glass, the Collingridge Dilemma, and the Mediated Value of Privacy". In: *Science, Technology, and Human Values* 44.2 (2019), pp. 291–314. DOI: 10.1177/0162243918793711.

[12] *Ray-Ban Meta Smart Glasses*. https://www.meta.com/smart-glasses. Ray-Ban Meta smart glasses are widely considered the best smart glasses in the world, allowing you to experience life to the fullest while being fully present. 2024.

[13] *ElevenLabs: AI Voice Generator and Text to Speech*. https://elevenlabs.io. Provides AI-driven text to speech and voice cloning services with a variety of voice options and multilingual capabilities. 2024.

[14] Alec Radford et al. "Robust Speech Recognition via Large-Scale Weak Supervision". In: *arXiv preprint arXiv:2212.04356* (Dec. 2022). Submitted on 6 Dec 2022. eprint: 2212.04356 (eess.AS).

[15] *Deep Learning Market*. https://www.precedenceresearch.com/deep-learning-market. 2024.

[16] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. DOI: 10.1037/h0042519.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[18] Arvid S. Lundervold and Alexander Lundervold. "An overview of deep learning in medical imaging focusing on MRI". In: *Zeitschrift für Medizinische Physik* (2019). DOI: DOIifavailable. URL: https://www.elsevier.com.

[19] David Daniel Cox and Thomas Dean. "Neural networks and neuroscience-inspired computer vision". In: *Current Biology* 24.18 (2014), R921–R929.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[21] Haohan Wang and Bhiksha Raj. "On the Origin of Deep Learning". Version v4. In: *arXiv preprint arXiv:1702.07800* (Mar. 2017). Submitted on 24 Feb 2017, last revised 3 Mar 2017. eprint: 1702.07800 (cs.LG).

[22] Sagar Sharma. *What the Hell is Perceptron? The Fundamentals of Neural Networks*. Towards Data Science. 3 min read. Sept. 2017. URL: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[23] A. Robert Martin et al. *From Neuron to Brain*. English. 6th ed. Sinauer Associates, 2020.

[24] Mark W Barnett and Philip M Larkman. "The action potential". In: *Practical Neurology* 7.3 (2007), pp. 192–197. DOI: 10.1136/jnnp.2007.120535. URL: https://pn.bmj.com/content/7/3/192.

[25] Branislav Holländer. *Natural vs Artificial Neural Networks*. Becoming Human: Artificial Intelligence Magazine. 11 min read. Aug. 2018. URL: https://becominghuman.ai/natural-vs-artificial-neural-networks-9f3be2d45fdb.

[26] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

[27] David A. Sousa. *How the Brain Learns*. Corwin Press, 2016. URL: https://books.google.com.

[28] Michael A. Nielsen. *Neural Networks and Deep Learning*. Vol. 25. San Francisco, CA, USA: Determination Press, 2015, pp. 15–24.

[29] Yann LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551.

[30] Thomas G. Dietterich. "Overfitting and undercomputing in machine learning". In: *ACM Computing Surveys (CSUR)* (1995).

[31] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: 2nd ed. O'Reilly Media, Inc., 2019. Chap. 3. ISBN: 978-1492032649.

[32] G. [3Blue1Brown] Sanderson. *Linear Algebra*. https://www.3blue1brown.com/topics/linear-algebra. Educational content on linear algebra concepts and applications. 2024.

[33] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. This document is the preprint version of Modern Robotics: Mechanics, Planning, and Control. This preprint is being made available for personal use only and not for further distribution. Original figures from this book may be reused provided proper citation is given. More information on the book, including software, videos, and a feedback form can be found at http://modernrobotics.org. Cambridge University Press, 2017.

[34] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. "Activation Functions in Neural Networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12 (2020), pp. 310–316. ISSN: 2455-2143. URL: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.

[35] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[36] Sven Nyholm. *Humans and Robots: Ethics, Agency, and Anthropomorphism*. Mar. 2020. ISBN: 1786612275.

[37] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A Learning Algorithm for Boltzmann Machines". In: *Cognitive Science* 9 (1985). *Cognitive Science 9, 147-169 (1985). Computer Science Department, Carnegie-Mellon University; Biophysics Department, The Johns Hopkins University, pp. 147–169.

[38] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016). Insight Centre for Data Analytics, NUI Galway; Aylien Ltd., Dublin. ruder.sebastian@gmail.com.

[39] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: 2nd ed. O'Reilly Media, Inc., 2019. Chap. 4. ISBN: 978-1492032649.

[40] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: 2nd ed. O'Reilly Media, Inc., 2019. Chap. 4. ISBN: 978-1492032649.

[41] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". Hochreiter's diploma thesis identifying the vanishing gradient problem. PhD thesis. Technical University of Munich, 1991.

[42] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[43] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. 2010, pp. 807–814.

[44] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951). JSTOR is a not-for-profit service that helps scholars, researchers, and students discover, use, and build upon a wide range of content in a trusted digital archive., pp. 400–407. URL: https://www.jstor.org/stable/2236626.

[45] Yann LeCun et al. *Efficient BackProp*. University of California San Diego. Although the tricks and analyses in this paper are primarily... 1998. URL: https://cseweb.ucsd.edu/Handouts/lecun-98b.pdf.

[46] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[47] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *arXiv preprint arXiv:1607.06450* (July 2016). Submitted on 21 Jul 2016. eprint: 1607.06450 (stat.ML).

[48] Yuxin Wu and Kaiming He. "Group Normalization". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19.

[49] Shibani Santurkar et al. "How Does Batch Normalization Help Optimization?" In: *Advances in Neural Information Processing Systems*. 2018, pp. 2483–2493.

[50] Jingjing Xu et al. "Understanding and Improving Layer Normalization". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[51] Md Zahangir Alom et al. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches". Version v2. In: *arXiv preprint arXiv:1803.01164* (Sept. 2018). Submitted on 3 Mar 2018, last revised 12 Sep 2018. eprint: 1803.01164 (cs.CV).

[52] John J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8 (1982), pp. 2554–2558.

[53] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[54] Joshua Achiam et al. "GPT-4 Technical Report". In: (2023). arXiv: 2303.08774 [cs.CL].

[55] L. Tunstall, L. von Werra, and T. Wolf. "Natural Language Processing with Transformers: Building Language Applications with Hugging Face". In: O'Reilly Media, 2022. Chap. 1. ISBN: 9781098103248. URL: https://books.google.nl/books?id=pNBpzwEACAAJ.

[56] OpenAI. *Better Language Models and Their Implications*. OpenAI Blog. 2019. URL: https://openai.com/blog/better-language-models/.

[57] OpenAI. *Introducing GPT-3*. OpenAI Blog. 2020. URL: https://openai.com/blog/openai-api/.

[58] Rohan Anil et al. "PaLM 2 Technical Report". In: (2023). arXiv: 2305.10403 [cs.CL].

[59] Seungwhan Moon et al. "Anymal: An efficient and scalable any-modality augmented language model". In: (2023). arXiv: 2309.16058 [cs.CL].

[60] L. Tunstall, L. von Werra, and T. Wolf. "Natural Language Processing with Transformers: Building Language Applications with Hugging Face". In: O'Reilly Media, 2022. Chap. 2, p. 29. ISBN: 9781098103248. URL: https://books.google.nl/books?id=pNBpzwEACAAJ.

[61] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: O'Reilly Media, 2019. Chap. 3. ISBN: 978-1492032649. URL: https://books.google.nl/books?id=pNBpzwEACAAJ.

[62] L. Tunstall, L. von Werra, and T. Wolf. "Natural Language Processing with Transformers: Building Language Applications with Hugging Face". In: O'Reilly Media, 2022. Chap. 3, p. 82. ISBN: 9781098103248. URL: https://books.google.nl/books?id=pNBpzwEACAAJ.

[63] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: 2nd ed. O'Reilly Media, Inc., 2019. Chap. 16. ISBN: 978-1492032649.

[64] L. Tunstall, L. von Werra, and T. Wolf. "Natural Language Processing with Transformers: Building Language Applications with Hugging Face". In: O'Reilly Media, 2022. Chap. 3, p. 70. ISBN: 9781098103248. URL: https://books.google.nl/books?id=pNBpzwEACAAJ.

[65] Philip Gage. "A new algorithm for data compression". In: *The C Users Journal archive* 12 (1994), pp. 23–38. URL: https://api.semanticscholar.org/CorpusID:59804030.

[66] Kaj Bostrom and Greg Durrett. "Byte Pair Encoding is Suboptimal for Language Model Pretraining". Version v2. In: *arXiv preprint arXiv:2004.03720* (Apr. 2020). Submitted on 7 Apr 2020, last revised 5 Oct 2020. eprint: 2004.03720 (cs.CL).

[67] Felipe Almeida and Geraldo Xexéo. "Word Embeddings: A Survey". Version v2. In: *arXiv preprint arXiv:1901.09069* (Jan. 2019). Submitted on 25 Jan 2019, last revised 2 May 2023. eprint: 1901.09069 (cs.CL).

[68] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". Version v3. In: *arXiv* (2013). arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[69] John Hewitt. *Note 10: Self-Attention and Transformers*. CS 224n: Natural Language Processing with Deep Learning, Stanford University. 2023.

[70] Shraddha Anala. *A Guide to Word Embedding: What is it? How is it more useful than a Bag-of-Words model?* Towards Data Science. 7 min read. Oct. 2020. URL: https://towardsdatascience.com/a-guide-to-word-embeddings-8a23817ab60f.

[71] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *arXiv preprint arXiv:1310.4546* (Oct. 2013). Submitted on 16 Oct 2013. eprint: 1310.4546 (cs.CL).

[72] Harald Steck, Chaitanya Ekanadham, and Nathan Kallus. "Is Cosine-Similarity of Embeddings Really About Similarity?" In: *arXiv preprint arXiv:2403.05440* (Mar. 2024). Submitted on 8 Mar 2024. eprint: 2403.05440 (cs.IR).

[73] Aurélien Géron. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow". In: 2nd ed. O'Reilly Media, Inc., 2019. Chap. 13. ISBN: 978-1492032649.

[74] Guillaume Genthial et al. *Neural Machine Translation, Seq2seq and Attention*. Lecture Notes for CS224n: Natural Language Processing with Deep Learning. Course Instructors: Christopher Manning and Richard Socher. 2017. URL: URL%20to%20the%20lecture%20notes.

[75] G. [3Blue1Brown] Sanderson. *Attention*. https://www.3blue1brown.com/lessons/attention. Educational content on attention mechanisms in neural networks. 2024.

[76] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. "A review on the attention mechanism of deep learning". In: *Neurocomputing* 452 (2021), pp. 48–62. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. URL: https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[77] Saurav Pawar et al. "The What, Why, and How of Context Length Extension Techniques in Large Language Models – A Detailed Survey". In: *Proceedings of a relevant conference or journal (if available)* (2023). Please update the journal, volume, number, pages, publisher, and DOI if available.

[78] Bowen Peng et al. “YaRN: Efficient Context Window Extension of Large Language Models”. In: *Proceedings of a relevant conference or journal (if available)* Specify if available.Specify if available (2023), Specify if available.

[79] Patrick Lewis et al. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. In: *Advances in Neural Information Processing Systems*. Vol. 33. NeurIPS. 2020.

[80] Jiayu Ding et al. “LongNet: Scaling Transformers to 1,000,000,000 Tokens”. Version v2. In: *arXiv preprint arXiv:2307.01234* (2023). URL: https://arxiv.org/abs/2307.01234.

[81] Nelson Elhage et al. “A Mathematical Framework for Transformer Circuits”. In: (Dec. 2021).

[82] Lijie Cao. “Singular Value Decomposition Applied to Digital Image Processing”. In: (2023).

[83] OpenAI. *Whisper*. OpenAI Research. 2022. URL: https://openai.com/research/whisper.

[84] Sanchit Gandhi, Patrick von Platen, and Alexander M. Rush. “Distil-Whisper: Robust Knowledge Distillation via Large-Scale Pseudo Labelling”. In: *arXiv preprint arXiv:2311.00430* (Nov. 2023).

[85] Akshara Pande, Bhanu Shrestha, and Anshul Rani. “A Comparative Analysis of Real Time Open-Source Speech Recognition Tools for Social Robots”. In: *Design, User Experience, and Usability*. Vol. 14033. Show All (4). 2023. ISBN: 978-3-031-35707-7.

[86] Richard P. Feynman, Robert B. Leighton, and Matthew Sands. *The Feynman Lectures on Physics*. Vol. 1. Reading, MA: Addison-Wesley, 1964. Chap. 47.

[87] T. Painter and A. Spanias. “Perceptual coding of digital audio”. In: *Proceedings of the IEEE* 88.4 (2000), pp. 451–515. DOI: 10.1109/5.842996.

[88] Dong-Hyun Lee. “Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks”. In: *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)* (July 2013).

[89] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the Knowledge in a Neural Network”. In: *arXiv preprint arXiv:1503.02531* (Mar. 2015).

[90] Haoxiang Wang et al. “SAM-CLIP: Merging Vision Foundation Models towards Semantic and Spatial Understanding”. Version v2. In: *arXiv preprint arXiv:2310.15308* (Oct. 2023). eprint: 2310.15308 (cs.CV).

[91] Jonathon Shlens. “Notes on Kullback-Leibler Divergence and Likelihood”. In: *arXiv preprint arXiv:1404.2000* (Apr. 2014). Submitted on 8 Apr 2014. eprint: 1404.2000 (cs.IT).

[92] Dietrich Klakow and Jochen Peters. “Testing the correlation of word error rate and perplexity”. In: *Speech Communication* 38.1-2 (Sept. 2002), pp. 19–28. DOI: 10.1016/S0167-6393(01)00041-3.

[93] Yaniv Leviathan, Matan Kalman, and Yossi Matias. “Fast Inference from Transformers via Speculative Decoding”. Version v2. In: *arXiv preprint arXiv:2211.17192* (May 2023). eprint: 2211.17192 (cs.LG).

[94] Charlie Chen et al. “Accelerating Large Language Model Decoding with Speculative Sampling”. In: *arXiv preprint arXiv:2302.01318* (Feb. 2023). eprint: 2302.01318 (cs.CL).

[95] Weihan Wang et al. “CogVLM: Visual Expert for Pretrained Language Models”. Version v2. In: *arXiv preprint arXiv:2311.03079* (Nov. 2023). Submitted on 6 Nov 2023, last revised 4 Feb 2024. eprint: 2311.03079 (cs.CV).

[96] Haotian Liu et al. “Visual Instruction Tuning”. In: (2023). arXiv: 2304.08485 [cs.CL].

[97] Haotian Liu et al. *Improved Baselines with Visual Instruction Tuning*. 2023.

[98] Haotian Liu et al. *LLaVA-NeXT: Improved reasoning, OCR, and world knowledge*. Jan. 2024. URL: https://llava-vl.github.io/blog/2024-01-30-llava-next/.

[99] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[100] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: (2021). arXiv: 2103.00020 [cs.CV].

[101] James Betker. “Better speech synthesis through scaling”. Version v2. In: *arXiv preprint arXiv:2305.07243* (May 2023). Submitted on 12 May 2023, last revised 23 May 2023. eprint: 2305.07243 (cs.SD).

[102] Yinghao Aaron Li et al. “StyleTTS 2: Towards Human-Level Text-to-Speech through Style Diffusion and Adversarial Training with Large Speech Language Models”. Version v2. In: *arXiv preprint arXiv:2306.07691* (June 2023). Submitted on 13 Jun 2023, last revised 20 Nov 2023. eprint: 2306.07691 (eess.AS).