Lecture Summarizer

A Tool for Personalized Learning and Assessment

Bachelor's thesis
Degree Programme in Computer Applications
Spring 2024
Martin Pavel Jaroš

HAMK
Häme University
of Applied Sciences

Degree Programme in Computer Applications — Abstract
Author Martin Pavel Jaroš — Year 2024
Subject Lecture Summarizer: A Tool for Personalized Learning and Assessment
Supervisors Deepak Kc

This thesis's endeavour ventured into the development of a semi-automatic, artificial note-taker. Customer's proposition was the summarization of attended academic lectures, due to their inability to mark down information in time, therefore, extensive research on the note-taking practices amongst students was conducted. The primary research question driving this thesis determined if the utility of combined modern technologies in a scenario described by the customer, effectively enhance students' engagement. Additionally, the following research question addressed the adaptations of various digital technologies in the education field and applied the best practices in order to provide the customer with the most suitable solution.

The purpose of the literature review presented in this thesis was to describe and abstract the note-taking practices among students through qualitative research of influential publications on this phenomenon. The critical literature regarding the potential negative effects of note-taking on conceptualization was highly considered to determine the product's potential viability for everyday academic use, however, these effects were found to be relatively infrequent among the average student. An overview of the most recent applicable technologies was provided, detailing their early recorded origins and contemporary abstractions. These technologies were applied to construct a prototype of the desired product according to the researched noting requirements.

The developed product proved to be a viable solution for students defined as low efficiency listeners. The software was designed to record and transcribe given lecture, reduce it into keywords, and generate informative descriptions of these keywords. The product was released as a free and redistributable software under the General Public Licence version 3 to empower individuals in their learning endeavours.

Keywords Education, Note-taking, Machine Learning, Text Generation, AI
Pages 36 pages and appendices 1 page

Glossary

ML	Machine learning
AI	Artificial Intelligence
TF-IDF	Term Frequency–Inverse Document Frequency
LLM	Language Learning Model
GUI	Graphical User Interface
GPU	Graphical Processing Unit
CUDA	Compute Unified Device Architecture
UTF-8	Unicode Transformation Format 8-bit
LLAMA	Large Language Model Meta AI

Content

Figures

Appendices

1 Introduction

In recent years, academic lecturing has shifted its medium from centralized and traditional classroom seminars and lectures to pre-recorded and globally available education materials. This shift has had a profound impact on increasing the amount of data that students can access after an introduction by their lecturer. (Rogers, 2000, pp. 21, 23) Students are usually allowed to take personal notes during lectures to help them make sense of the information they gather. After class, they may further examine and expand their understanding using additional resources like prescribed books or other materials related to the subject matter. The multimedia archives of academic lectures are determined by the institution, however, students can take notes on everything their lecturers say during class and record it in their notebooks or laptops. Furthermore, advancements in voice recording and other assistive digital technologies have provided significant benefits for students with learning disabilities, enhancing their ability to access and engage with course material. (Forgrave, 2002, pp. 122-126).

This thesis outlines the gathered background information and the methods for taking notes during academic lectures. This psychological analysis served as a foundation for the variable, personal, user experience and functionality of the final product. This thesis includes development of a software prototype. The prototype's components and core functions are established based on a prior analysis of the students' needs and their common practices.

The prototype utilizes a modern implementation of voice synthesis neural networks for full transcription of audio inputs, along with the deterministic algorithm TF-IDF for feature extraction. The extracted topics are broadcast into a natural language processing model, which was tasked with generating informative texts using the latest natural language generation methods. The system subjected the background knowledge gathered in the research of the processes and answer the following research questions:

- Can speech synthesis system combined with advanced text generation and fact checking be used to enhance student engagement during lectures, and if so, how does this impact their overall academic performance?

- How do combined modern technologies affect the learning experience for students, and what adaptations could be made to improve accessibility and usability?

2 Note-taking & fundamental concepts of education and learning

This chapter explores the relevant, fundamental concepts of academic education and pedagogy. This literature overview takes focus on the significance of note-taking through-out the more recent history of academic education. The second and the third part of the theoretical framework for this thesis, encapsulates the most up to date information accumulation and transformation technologies. Covered technologies combined with the education principles discovered by the analysis will serve as a guide for the selection of the final product's components and its implementation.

2.1 Lecture note-taking

Note-taking in higher education is the practice of recording and organizing clearly expressed information into a suitable medium (Friedman, 2014, p. 5). Educational psychologists also experiment and approve the idea that writing notes supports the engagement of the the student in the general topic they listen to (Bohay et al., 2011, p. 70). The following paragraphs reviewed the critical literature of this practice in the scope of instructional, live or pre-recorded lecture circumstances.

According to the results of a questionnaire, answered by the total of 123 American and British students, 92% of these students considered note-taking an important activity. To that account, only 25% of the questioned British students were ever encouraged or instructed to take notes by their mentors or peers. American students reported a greater inclination towards taking notes while studying, with 56% responding affirmatively when asked if they were encouraged or instructed to do so. (Hartley & Davies, 1978, p. 208) This suggests that even if note-taking is not taught it appears to be a natural function for many students. Although according to Marin and Sturm (2021, pp. 1, 7), other experts consider this function impersonal or a formal, collective gesture. The same questionnaire, done by Hartley and Davies (1978, p. 208) suggests to support this claim, with more than 50% of all the students filling out this questionnaire, answered "yes" when asked if their note-taking ever gets in the way of their direct comprehension of what their lecturers are saying. The support mentioned pertained to the sense of uncertainty that more than half of the questioned students had when they were attending a lecture on a personally difficult subject. Students may struggle with complex topics and choose to focus on actively listening rather than taking extensive notes. Alternatively, they might prioritize note-taking in order to review the material more thoroughly at a later time. Ultimately, it is up to each individual student to determine what works best for them when balancing their need for comprehension with their desire to retain information. (Kiewra 1989, pp. 151, 152)

According to Hartley and Davies (1978, pp. 209, 210), instances where students demonstrate enhanced comprehension of a subject without the aid of written verbalization of the lecture are uncommon. The researchers examined a total of 35 separate studies to determine whether note-taking enhances students' immediate comprehension of a subject and their subsequent recall of that material, both with and without revisiting the topic at later times. They found only two studies that provided positive feedback from the **absence** of taking notes. Among their reviewed literature, seventeen studies supported Hartley's and Davies's hypothesis, while sixteen reviewed papers did not provide conclusive evidence to substantiate it. (Hartley & Davies,1978, pp. 209, 210)

This collected literature review provides robust support for the widely-held notion that students are more likely to remember an item on a test if it is written in their notes, regardless of whether they review or elaborate upon them after the lecture. The validity of this phenomenon has been extensively documented by researchers such as Paul Weener (1974, p. 51) and Kenneth Kiewra (1989, pp. 149, 150), who have consistently found that students are more likely to remember the information when it is recorded in written form during a lecture, rather than simply relying on auditory or visual cues alone. Furthermore, Kiewra's (1989, p. 147) investigation into the "encoding function" of note-taking revealed that this process effectively assisted in learning about the subject matter by strengthening the cognitive connection of presented material. Moreover, his paper described the benefits of the generally agreed-upon "storage function", where he explained that reviewing one's notes could help solidify understanding of a subject and recover lost or undocumented details, thereby enhancing overall comprehension and retention of the information. (Kiewra, 1989, p. 147)

Mueller and Oppenheimer's (2014, pp. 1166) investigated the effects of taking notes on learning outcomes by conducting a series of comprehensive experiments that included the use of modern digital devices. The researchers' experiments approved that taking notes on paper can enhance factual content retention, even when there is no delay between the lecture and the assessment, as stated by other researchers before. However, upon comparing the outcomes of the study involving digital note-takers to those of traditional longhand note-takers, it became evident that using digital devices had a negative impact on student performance in academic evaluations. Digital note-takers were found to be more accurate in transcribing lectures but tended not to process and formulate the information into their own words as efficiently. Verbatim transcription of the lecture for note-taking purposes indicated lower performance levels. (Mueller & Oppenheimer, 2014, p. 1160)

They also noted that advising students not to transcribe the lecturer onto their digital devices was, in some instances, futile as the students often disregarded this oral guidance resulting in the detrimental effect (Mueller & Oppenheimer, 2014, pp. 1159, 1166). The use of digital devices can also lead to distractions for students, making it difficult for them to focus and understand lecture

materials. Students who are either struggling or uninterested in a lecture might be more inclined to bring their digital devices just to to pass the time. This was considered as epicentre for collateral distraction through loud messages and phone calls, or by playing network computer games with other classmates as pointed out by questioned students. (Goundar, 2014, pp. 212, 226) Goundar's study also described their experiment with a group of students that were allowed to use the world wide web during a lecture. The researchers analysed the internet traffic and found that many of these students were distracted by other unrelated internet activities such as online shopping or watching videos. (Goundar, 2014, p. 216)

Hartley and Davies (1978, p. 207), describe their findings on the different notation styles they have classified during their experiments. The first type are the students, who try to transcribe everything, or use abbreviations for words their lecturer says. Second type are the students who select the most important points out of the lecture. The last observed type of note-takers in this research were the students who condoned any reviewable note generation, but instead, these students chose to doodle and concentrate on listening to their lector. This suggests that students have different skills and practices for their note-taking depending on individual learning preferences and goals. While transcribing the lecturer's words verbatim may retain the most information, students who engage in this activity often focus more on precisely capturing every detail rather than truly comprehending the topic at hand. This can lead to a disconnect between what is being learned and its practical application. The second approach appears to be the most suitable in a world, where a wealth of information is readily accessible for students to review and update their notes accordingly. In this case, students mark down the highlights and hierarchy of the lecture, on which they are able to elaborate with the aid of valid resources during the revision of their notes. (Hartley & Davies 1978, p. 207; Mueller & Oppenheimer, 2014, p. 1160) The Lecture summarizer tool is targeted towards the third group mentioned in the research done by Hartley and Davies (1978, p. 207).

Peters (1972, pp. 278, 279) refers to this group as the "low-efficiency listeners". He classified this group in his study, by conducting criterion tests on one's individual, inherent ability to learn, to determine the differences in learning, instead of prior knowledge. According to Peters, the low-efficiency listeners have accomplished higher scores when they were not committed to note-taking, during the listening of tape recorded lecture (Peters 1972, pp. 278, 279).

Supportive findings by Kiewra (1989, pp. 150, 153) showed that some students who did not take notes or attend lectures, but borrowed and reviewed them from classmates before testing, performed similarly to, and occasionally even outperformed, the present students who took notes but did not review them. The lecture material necessary for establishing these findings demanded that students create connections between different concepts. Kiewra (1989, p. 153) suggests that

the act of reviewing notes allows for the formation of deeper internal connections then noting during lecture, potentially because note-takers were too focused on capturing information during the lecture to make these connections at the time. Backed up by Mueller & Oppenheimer (2014, p. 1166), these findings underscored the constraints of note-taking in fostering a deep comprehension as opposed to, for example, an independent review of the material. (Mueller & Oppenheimer, 2014, p. 1166)

2.2 Artificial intelligence

The "intelligence" part of the term was extensively studied in the past. Notably, revised Aristotle's (1986, pp. 39–44) studies of the soul. He defined knowing as an "art to its material," suggesting that comprehension is directed towards the object of thought when one possesses knowledge. Aristotle described this state as actual knowledge. Prior to acquiring actual knowledge, his logic suggested that an individual must possess potential knowledge, described as underlying knowledge of universally everything, which requires a cause to ascend. For instance, when someone wishes to learn a topic, their causes would be the topic material, reading skills, driving forces such as teachers or mentors, and the purpose behind their goal. These causes convert potential knowledge into actual knowledge. In nature, both potential and actual knowledge exist simultaneously, with one preceding the other, however, when considering the totality of knowledge as our universe, potential knowledge does not precede actual knowledge in our temporal understanding. Aristotle described the object of thought as an equal part of a "class of things." The class of things consists of causes or factors that bring potentials into actuality and matter - the potential to ascend to actuality. In the topic learning example mentioned earlier, the matter is the understanding of the topic itself. He concluded that intelligence is the empowerment of one's potential understanding and thought with a cause to abstract and conceptualize that understanding. (Aristotle, 1986, pp. 39–44)

In the "artificial" sense, researchers were making efforts to **simulate** and observe the patterns of the previously described, human intelligence. In his 1947 lecture, Alan Turing laid the foundational ideas for the notion that computational, programmable machines would be more beneficial to further research on intelligence, rather than biological entities. Mathematicians and AI researchers had considered philosophical definitions of intelligence and applied them in a simulated environment, composed by certain constraints sometimes designed to mimic real-world conditions, yet freed from the limitations imposed by our biological nature. (McCarthy, 2007, pp. 2–4)

Prior to the development of Turing's programmable machines, researchers explored and experimented with philosophical concepts related to intelligence through artificial implantation of knowledge and assessment of outcomes. For example, one of the considered principles of

intelligence was behaviourism. Thorndike (1898, p. 818) conducted experiments to answer the question of how a new skill is formed and learned. The experiment consisted of artificially constructed physical enclosures, with mechanical buttons and latches outside of the subject's comprehension, as the means of escape. The testing subjects were animals such as cats and dogs. The hungry subjects were placed into the enclosure and the food was placed outside of the enclosure. Thorndike learned that at first, these subjects shown no insight in understanding of the problem, meaning the mechanism to open the path to their food, but by the means of trial and error, they sometimes managed to open the door. After succeeding for the the first time, the subject's positive experience strengthened the following experiment trials in which, according to his research, the animals performed considerably faster. (Thorndike, 1898, p. 818)

Thorndike conducted a follow-up study to his initial investigation into animal intelligence, demonstrating that the "law of effect" also applied to humans. According to this principle, the actions and behaviours of living entities can be shaped by external stimuli, allowing for intelligence to develop and progress over time. In his study, Thorndike exposed human subjects to a set of words read aloud by an experimenter. The subjects were directed to respond to each said word with a number from 1 to 10. Each of the spoken words has a designated number on the same scale. If the subject manages to select the correct number for its corresponding word, they would be rewarded. If the subject fails to select the correct number, they would be punished. The series of these words with their corresponding correct numbers, was repeated multiple times although the paper refers to the results of only the initial, four trials. What the results clearly expressed, was that the action of the subject can be strengthened through positive reinforcement of the correct response, thus if the subject selected correct number for a word in a first trial, they would be influenced by the reward to do so again the next trial. (Thorndike, 1933, pp. 173, 174)

In addition to the same conclusion as the previous experiment on animals, the punished responses, did not all behave the same according to his measures. The influence of the rewarded answers also seemed to spread to other, near answers. He calculated the strengthening of this phenomena by measuring the percentages of answer repetitions in the following test trial. The measures indicated a regression of repetitions from the first trial with 26.4% repetitions, to 20.8% repetitions in the fourth trial. Comparing this data with the measures of the proximity of the rewarded answers, in terms of time and the number of all the combined answers, showed a slight tendency to be strengthened more than the responses further away from correct numbers. This indicated that the effects of rewarding mechanism, extend beyond the immediate action to influence other, near actions. (Thorndike, 1933, pp. 173, 174) These ideas, later served as the fundamental concepts for the “operand conditioning”, and later “reinforcement learning” machine learning process, with wide applications in the gaming field but it was even used to establish a set

of conditions for riding a bicycle using mathematical principles. (Agostinelli et al., 2018 pp. 299, 300)

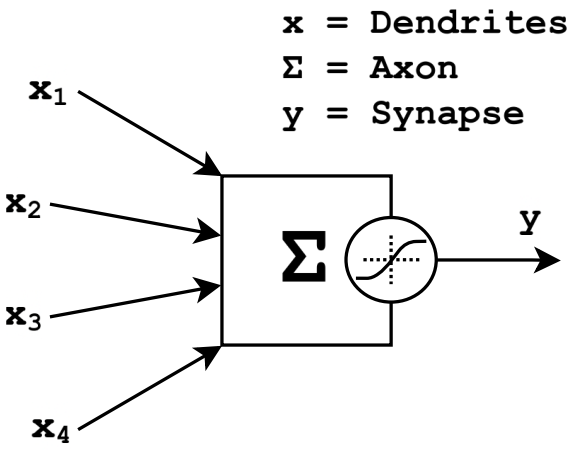
2.3 Machine learning

Machine learning is a branch of computer science, that investigates mechanically simulated learning and thought processes of living entities. Simulating the psychological behaviour of living beings in abstract environments allows researchers to develop more efficient systems for accomplishing predetermined tasks and potentially discover new theoretical possibilities. Researchers gain deeper understanding of human cognitive process, which may lead to the development of new approaches to solve complex problems or addressing various challenges by testing the consistency of collected knowledge. In the field of machine learning, this is achieved by apprehending the fundamental process specifics, also called sets, which rules out the historical impossibility of testing some theories. (Carbonell et al., 1983, pp. 3, 4)

To put things in perspective, number of tasks that are routinely performed by biological intelligence, such as driving a vehicle or recognizing imagery and sounds, cannot be strictly determined with clear mathematical specifications. However, by providing these algorithmic techniques with a memory to learn from their previous approaches or the ability to observe and imitate the actions of others, it is possible to make sufficient predictions of the defined end conditions. The second range of objectives where machine learning excels is in tasks that exceed human cognitive abilities, such as analysing large amounts of astronomical or weather data. By processing vast quantities of information and identifying patterns, machine learning algorithms can make predictions and recommendations that would be difficult for humans to replicate manually. (Shalev-Shwartz & Ben-David, 2014, p. 3)

At its core, the foundation for enabling machines to imitate biological cognition lies in a simplified mathematical model that describes an interconnected network of threshold switches functioning as neurons. This concept was first introduced by Warren McCulloch and Walter Pitts in their 1943 paper. The model serves as a basis for modern neural networks, which have since been widely adopted for various applications ranging from pattern recognition to natural language processing. The neural network is constructed by simulating the actions of a biological neurons (Figure 1). In a neural network, each artificial neuron mimics the function of a biological **dendrite** by collecting input weight streams from other neurons. These collected signals are summed in an artificial **axon**, which then determines if its predefined threshold has been crossed. If so, it sends out a corresponding positive signal, emulating the role of a biological **synapse**. This process enables efficient information transfer and decision making within the network, however, it does not inherently support concurrent learning. (Bishop, 1993, pp. 1804, 1805; Krogh, 2008, p. 195)

Figure 1 A model of artificial neuron



The concept of an artificial neuron within an interconnected network was further developed by F. Rosenblatt and subsequently expanded upon by Minsky and Papert (2017, p. Foreword 15). They described neural network as the “simplest type of learning machine”. According to Minsky and Papert (2017, p. Foreword 17, 18) the first prototype of neural network with a reinforcement learning strategy was constructed by Minsky in 1951. The initial model composed of forty artificial neurons, arranged as a single layer of axons, interconnected through a network of unidirectional links. When the device made a decision, neurons transmitted signals to each other, the human administrator judged the decision and translated it to weight differential. The network reinforced connections between units that were simultaneously active and made a favourable choice, thus improving its performance in following trials. (Minsky & Papert, 2017, p. Foreword 17, 18) Strengthening connections between neurons that are active at the same time was referred to as the Hebbian learning principle (Minsky & Papert, 2017, p. Foreword 17).

The name "Perceptron" was chosen by researchers to honour Rosenblatt’s groundbreaking research in this area (Minsky and Papert 2017, p. 4). Additional experiments in the book suggested that the functioning of the human brain is derived from various types of interacting mechanisms. As such, it is not possible to accurately simulate these functions using just one method of learning. In some scenarios, it is advantageous to gradually build up experience. In other scenarios, when time is a constraint, it becomes essential to draw rapid conclusions and act on them. (Minsky and Papert 2017, p. 269) The traditional method for tackling an issue is to devise a specific mathematical procedure to understand it. Computers are proficient at performing most fundamental tasks that require a well-defined set of instructions. These instructions must be very precisely encoded by typically large teams of programmers. Since the machine lacks a mathematical procedure to achieve undetermined actions, the Perceptron employs a speculative formulation of

the deterministic function. This allows it to create a mathematical procedure that can, for instance, determine whether an event matches a specific pattern. (Minsky and Papert 2017, p. 4)

Since their inception in 1943, neural network methods have been adapted and advanced by numerous researchers. Vaswani et al. (2017, p. 1) proposed a paper in which they described the revolutionary **transformer architecture.** They were investigating the encoding of input into an intermediate representation and the subsequent decoding of this representation into an output, examining various established machine learning models, including the recurrent neural network, which unlike the Perceptron, includes more layers and is able to communicate in loops between the neurons. The research also covered the encoding-decoding architecture of the long short term memory model developed by Sepp Hochreiter and Jürgen Schmidhuber, and according to Vaswani et al. (2017, p. 1), specifically Gated Recurrent Neural Networks architecture. They modified the recurrent neural network's additional layers with multi-headed self-attention layer, which allows the model to focus on different parts of the input sequence, such as syntax and semantic meaning simultaneously (Vaswani et al., 2017, pp. 3, 7).

In a dataset, the available values and their corresponding keys are represented as weights. When a query is subjected, it is translated into a corresponding weight vector and mapped across correlating key-value pairs. The output is determined by the degree of correlation between the query and each key. Keys that have a high correlation with the query receive more attention, meaning their corresponding values are given higher weights. The weighted values are then added together to generate the output. This led the researchers to develop the Transformer architecture, now one of the most popular models for natural language processing tasks. (Vaswani et al., 2017, pp. 3, 1)

2.3.1 Natural language generation

According to Touvron et al. (2023, p. 3) large language models are sophisticated artificial intelligence assistants, capable of reasoning with trained professional knowledge and able to generate creative text. The gradual development of individual machine learning techniques achieved by other researchers, has allowed the definition of a simple method for training and fine-tuning of large language models. The previously mentioned transformers were pre-trained on an extensive volume of human-generated textual data, without any human supervision during the initial weight assignment process. After the initial weight assignments, the model's weights were adjusted to align with specific, desired human needs through methods such as the before mentioned "reinforcement learning". (Touvron et al., 2023, p. 3) By employing this method, Touvron et al. (2023, pp. 3, 18) developed a series of large language models called Llama 2, which were able to outperform most other open source language models available at the time. This was

tested by the considered “gold standard” human evaluation tests, where humans judged the helpfulness of received responses (Touvron et al., 2023, p. 18). Touvron et al. (2023, p. 4) and Meta (n.d.) indicated that they developed and openly released this family of Llama 2 models with the goal of empowering individuals and thus provide a benefit to the society.

The initial unsupervised pre-training process was also referred to as the training of a foundation model. Foundation models were defined as large, flexible, reusable, transformer models that can be applied to serve many different purposes. These models consist of a large number of pre-labelled tokens that can then be directed towards accomplishing the desired behaviour through supervised learning techniques, like those used in the previously mentioned Llama 2 natural language processing for dialogue purposes. (IBM Research, n.d.)

A study conducted by Petroni et al. (2019, p. 1) demonstrated that the foundation of certain large language models can accurately recall correct factual information without requiring any additional fine-tuning or explicitly instructed behaviour. These pre-trained models do not inherently offer dialogue or other conditioned capabilities, therefore, Petroni et al. (2019, p. 4) developed an assessment probe to compare set of supervised facts to the factual base knowledge of the model using the masking method. The masking technique involves posing a statement that requires filling in a missing piece of information, and the foundation model supplies the required detail. (Petroni et al., 2019, pp. 1, 4) They concluded that large language models that would be continuously trained with new textual data, might become an alternative to knowledge bases as they are known to users today. (Petroni et al., 2019, p. 9)

2.3.2 Large text summarization

Summarization is a natural language processing task that involves compressing extensive source material into concise statements mentioning the main points (Pu et al., 2023, p. 1). Pu et al (2023, p. 4) explored the various fine-tuned models, specifically designed for the task of summarization of textual data. They discovered that most of these models had been fine-tuned and validated using relatively short texts, which limited their effectiveness due to the constraints imposed on their generated outputs. These fine-tuned summarization models struggled to catch multiple important topics across the longer source texts that these scientists used as subjects. As a result, these models performed very poorly when compared to other summarization methods. (Pu et al., 2023, pp. 1–4) Additionally, they evaluated the summarization capabilities of raw large language models such as GPT-3 and others, which surprisingly outperformed even human generated summaries in terms of factual knowledge. (Pu et al., 2023, pp. 1, 4)

2.3.3 Speech recognition

Research conducted by Forgrave (2002, pp. 124, 125) described the assistive technologies in education for students with learning disabilities, specifically the advantages of voice recognition software, also known as “speech-to-text” technology. The research states that such software enables its users to dictate their thoughts without having to physically write them down, making it an attractive for students who struggle with handwriting or require extra time to complete assignments. (Forgrave, 2002, p. 124) Her research into assistive technologies shown that using voice recognition software can have a positive impact on student motivation, as the ease of use and speed of production can increase enthusiasm for writing. Additionally, her research described improvements in reading comprehension, spelling, and word recognition scores among students who utilized this technology. The article states that continuous speech recognition software that allowed their users to speak at a regular, conversational rate without pausing between words have been shown to also enhance their working memory. The research concluded what voice recognition software has the potential to provide students with an accessible and effective means of expressing themselves more clearly through written communication. (Forgrave, 2002, pp. 124, 125).

One of the most modern adaptations of speech recognition is OpenAI’s Whisper, an automatic speech recognition model trained on a very large and diverse dataset of audio recordings. This general-purpose model was designed to perform multiple tasks at once, including multilingual automatic speech recognition, speech translation, language identification, and voice activity detection. According to the readme file, its core is composed of the previously described Transformer architecture which handles multiple speech processing tasks in a single machine learning model, replacing the pipeline stages with a unified approach. Whisper uses special sets of tokens that help it understand what task it needs to do for each section of provided audio recording. These tokens serve as signals to the model, indicating which specific task it should prioritize, such as transcribing spoken language or detecting distinct sounds like a cough. (OpenAI, 2023)

2.3.4 Retrieval-augmented generation

A study written by Lewis et al. (2020 p. 1) explored various methods and proposed a new concept, Retrieval-Augmented Generation models, which employ an approach that integrates the strengths of context-retrieval techniques and natural language generation methodologies. RAG combines contextual information retrieved from a vector database with the creative capabilities of generative models to generate text that is both informative and coherent. According to Lewis et al. (2020 p. 1), this approach has been shown to be effective in natural language processing tasks requiring

access to factual information, outperforming state-of-the-art pre-trained models in terms of effectivity measures. (Lewis et al., 2020 p. 1)

The ability of RAG to combine the advantages of retrieval and generation enables it to overcome some limitations of traditional generative models. Specifically, RAG can expand its memory and improve coherence or relevance issues when generating text, thereby addressing challenges that are being inherited in variations of such models. (Lewis et al., 2020 p. 1)

3 Methods

This chapter describes the methods taken to accomplish the construction of the theoretical foundation and the software prototype in order to investigate whether this supplementary tool, based on traditional longhand note-taking practices among students, could positively impact their learning outcomes. The investigation into artificial intelligence and machine learning was conducted with a goal of establishing an informational foundation for developing innovative software application capable of knowledge-based tasking scenarios.

3.1 Research methodology

This thesis was attempting to establish a critical theoretical foundation of academical note-taking and develop an advanced suitable solution, for any negative effects this practice might establish. The theoretical framework of the education field was constructed by qualitative case study methodology to gain deep insights and a comprehensive understanding of the note-taking phenomena. The qualitative case study involved diligent inspection of multiple publications and some of their corresponding references, which have been referenced frequently by other researchers in their own publications. These citation statistics were compared on the service Google Scholar, which serves as a search engine for academic publications. Google Scholar served as a primary method of acquiring theoretical background information for this thesis, by searching through its databases for mentions of input keywords. The keywords used to find publications consisted of the noun of the topics, such as “note-taking”, followed by a verb such as “review” or “critique”. If literature found on Google Scholar by a certain combination of keywords was insufficiently referenced by other scientific studies, the same keywords were used in Wikipedia to access the highly peer-reviewed articles from their reference lists. Scientific articles had the highest priority among other formats such as books or websites due to their compression of validated information and standard reference format for the ease of fact-checking. Some books and their underlying ideas were referenced due to their high importance to their relating fields, although only the referred chapters of the these books were exposed to the qualitative method of research. The description of utilized software libraries was paraphrased only from their official documentation or presentation pages.

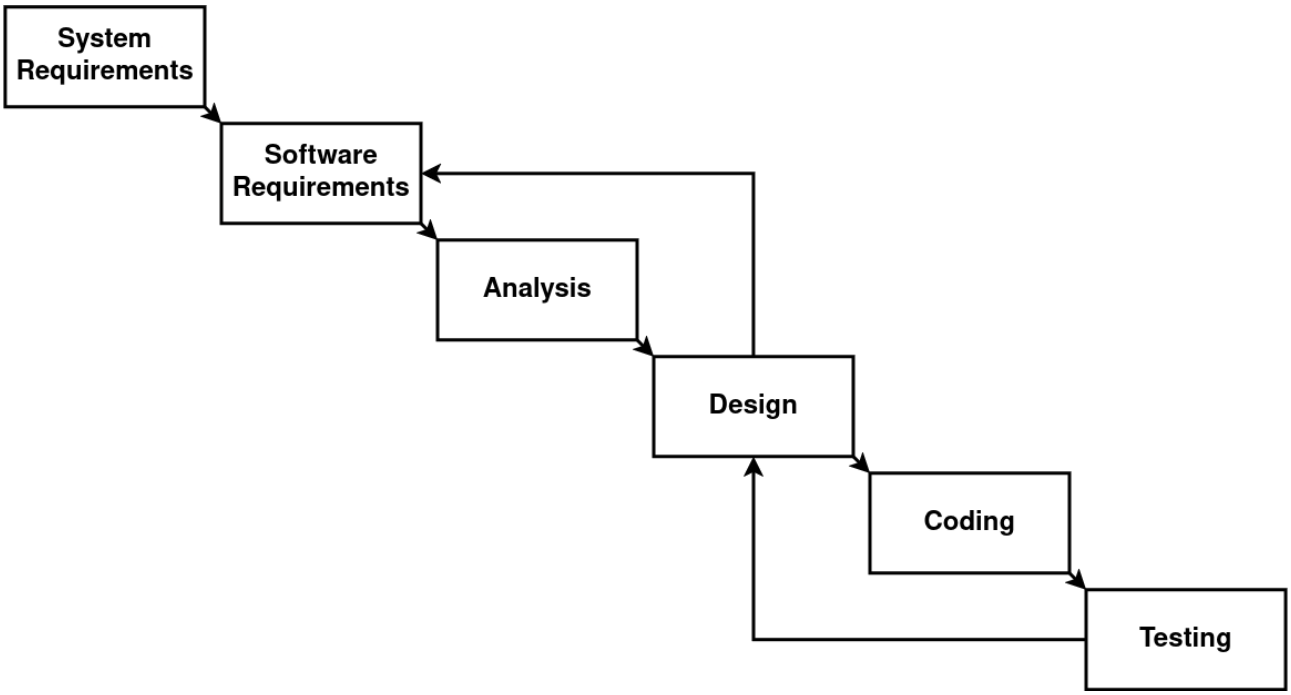
3.2 Product development strategy

The design and implementation approach employed in the prototyping phase of the product’s development process was informed by the principles of the Waterfall methodology. According to Royce (2021, p. 328), his publication presents a reflective account of his professional experiences

in overseeing software development initiatives for spacecraft missions, with particular emphasis on the application and validation of his project management methodology. He began by outlining the Waterfall methodology with two crucial stages that were considered essential to the overall software development process. The first stage was an **analysis phase**, where the problem was examined and a suitable solution was identified. The second stage involved the actual implementation of the solution through **coding**. He stated this development effort involved creative work and added to the value. Most customers were willing to pay for such efforts. (Royce, 2021, p. 328) But focusing only on only these two steps in development of sophisticated software products was according to Royce (2021, p. 328) “doomed to failure”.

Royce (2021, pp. 328–331) developed a more comprehensive model illustrated in Figure 2 that included five additional phases not visible to customers, yet essential for delivering large-scale computer software projects. Prior to commencing the analysis phase, he had designated the **system requirements** and **software requirements** phases as separate entities due to their unique objectives and outputs. Moreover, these distinct phases were closely linked to the **design phase** in that they established clear minimum hardware requirements and operational constraints imposed on the software design. (Royce, 2021, pp. 328–331) Royce (2021, pp. 328–331) considered the documentation of the design process as fundamental, since it functioned as a vital communication channel between the analysis phase and the following coding stage. After the precise and complete implementation of the documented design, the component was subjected to testing for verification of its functionality according to the original design described in documentation. If the testing process yielded unfavourable results, it was reasonable to attribute the outcome to potential design flaws in the original conceptualization. (Royce, 2021, p. 335) Similarly, during the Design Phase, the design team had identified any ambiguities or inconsistencies that existed in the originally documented system requirements. (Royce, 2021, p. 334).

Figure 2 Waterfall model according to Royce (2021, p 338)



This thesis's development project employed the Waterfall model in the manner previously described. The Waterfall model was considered suitable for multiple cooperating teams, however, the scope of the development project involved only the author of this thesis, who individually apprehended the predefined steps in the strategy. Initially, the author of the thesis acted as the customer, stating their problem with noting their lectures and composing a simple mock-up solution. Qualitative research into note-taking practices informed the definition of the product's final goals, which entailed transcription, emphasis on revision, and summarization tasks. Furthermore, the process identified the necessary resources required to execute the project, including the decision to deploy the application on hardware rather than cloud infrastructure. Subsequently, an examination was conducted to identify software tools that could potentially be applied for the required functionality. The analysis of requirements and propositional software was then documented and designed in Jupyter Notebook as internal narrative documentation. The employment of software within this testing environment allowed effortless backtracking from the testing phase to the design phase and to the software requirements phase. This process enabled a streamlined conceptualization, design, and implementation of individual components that had passed the testing phase and were ready to be implemented in the final development environment.

The code for implementing this product was uploaded to GitHub without providing a history of commits until the minimum viable state of the product had been achieved, due to the large amount of testing files and changes made to the product. The subsequent commits after the initial repository upload were small enhancements aimed at refining usability. The final development

product resided on the following hyperlink, making it available for anyone with access to GitHub, with freedom of redistribution and manipulation. https://github.com/JRoshthen1/LectureSummarizer

3.3 Product license

The developed product for this thesis was released under the GNU General Public License version 3. It is a license that requires any copied or adapted works to be licensed under the same open terms. Individuals creating software using GNU General Public License version 3 components are accountable for maintaining the open source nature of their work, by adhering to the specifications set forth in that license. The license includes provisions for inhibition of competitive legal action and leverages against extraction of licensing or settlement fees. This license aims to support the cause of developing software with the most use to the public, by allowing for redistribution, dissection and derivation of the underlying software. (Free Software Foundation, 2007)

4 Technologies

The following chapter describes the technological infrastructure established for the programming phase of the product development process. A comprehensive examination of the employed software tools, including details on their respective licenses and utilized versions, will be presented.

4.1 Python

Python is a high-level, modular, interpreted programming language. Python's open source code is written in the C programming language and it is designed to provide a set of accessible data structures to handle object oriented programming. Python code is being executed by the Python interpreter, which makes the development of applications faster, by cutting the code compilation times of C languages. Additionally, these interpreters can function virtually, offering a separate and isolated development environment from the system's existing setup. The Python's interpreters can also be run in terminal, interactively, and execute the written line by line for immediate results. Python aims to be easily usable by not only professional software developers, but its descriptive and indent oriented syntax allows even computer laymen quickly develop many automation tasks. The modularity of Python exponentially broadens the capabilities of this programming language. Its official package repository allows the community to contribute with a variety of ideas and approaches for further development. (Python Software Foundation, 2023)

These so-called modules are distributed with the help of package managers, allowing developers to substitute parts of their program with pre-built, community-developed components. One example is the package manager "Pip". Version 24.0 of Pip was used to collect packages throughout the development phase. The prototype for this project, was developed with the use of Python version 3.10.12. This version comes bundled with multiple standard, commonly used modules. The prototype utilizes threading, regular expressions, and the datetime module, all of which are standard, first-party modules. The **threading module**, is a high level programming interface for the lower level "_thread" interface, which handles the synchronization of multiple processes on a single processing unit. In the prototype, multiple other modules invoke the threading functionality to allow their resource heavy computation execute in the background, without blocking the graphical interface or other functions. The **relative expression module** provides accurate pattern matching operations on strings of text. This module has the Unicode and the UTF-8 character support making it useful for work with variety of natural languages. The **datetime module** provides functions for the extraction of date and time according to the proleptic Gregorian calendar. In the prototype, this module was used during the development stage to determine the efficiency of

different algorithms and machine learning models. In the production, the datetime module is used to compose names for the extracted documents. (Python Software Foundation, 2023)

The Python Software Foundation has approved and incorporated these modules into their Python interpreter and distributes them under its own license called the Python Software Foundation License version 2. This license is compliant to the GNU General Public license in most Python releases, including Python 3.10, which was utilized during the development process. (Python Software Foundation, 2024) In addition to these built in modules, the Lecture summarizer tool takes use of the following third party modules, approved by the community.

The **Python-sounddevice** module is an MIT-licensed software that offers user-friendly Python bindings to the widely used and versatile open-source audio processing library known as PortAudio. This higher level abstraction enables developers to effortlessly integrate audio capabilities into their Python applications, while still supporting multiple platforms such as Windows, Mac, and Unix. The Sounddevice library offers a method to discover input and output devices and record or playback audio files using those devices. (Python SoundDevice, 2023) Notably, the development process for this thesis relied on version 0.4.6 of the Python-sounddevice module, which provided a robust foundation for audio processing and manipulation within the context of this research driven development project.

The **Python-soundfile** module is an open-source software library that offers cross-platform solution for reading and writing various types of sound file formats with straightforward methods. This tool provides Python bindings to the lower-level "Libsndfile" C library, which enables developers effectively handle common audio file formats such as Waveform audio file and others. With access to its features, researchers and developers can seamlessly handle audio files of different types, thereby simplifying the audio incorporation process when working with Python. This module was licensed under the permissive BSD license, which ensured that users had unrestricted access to its source code and were free to redistribute it, as needed for incorporation into a GNU project. (SoundFile documentation, 2015)

The latest version of **NumPy** 1.26.0, served an important yet limited role in the development of the product. NumPy serves as a robust module that enables efficient and versatile numerical operations through its core features described as vectorization, indexing, and broadcasting. This functionality is possible by NumPy's supply of various mathematical functions, random number generators, and linear algebra routines. NumPy is released under their BSD-style license. (NumPy Team, 2024) This library constitutes as an indispensable tool for numerical computing in general, although this specific project only leveraged this library to concatenate arrays containing audio samples obtained from Sounddevice.

The **Tkinter** module served as Python's programming interface for the widely used "Tk" GUI toolkit. Tk was designed to easily create graphical interfaces across common operating systems, including Windows, Mac, and Unix. As an open-source project with a BSD-style license, Tk allowed many other programming languages to adopt it, aside from its native Tcl programming language. The development product was built using Tkinter version 8.6. Tkinter has been declared a standard toolkit for developing native desktop applications, ensuring usability across platforms without changes in appearance. (Roseman, 2022; Python Software Foundation, 2024)

Tkinter works on the basis of an event loop, running on a single thread, meaning that the program waits until an event is processed before proceeding with of the execution another. Tkinter's event-driven model encourages asynchronous programming practices, in this case multiple threads, allowing for more responsive GUI interactions. (Roseman, 2022)

Tkinter's interface required fewer resources to run compared to a web-based application interface and did not require an independent server to operate alongside the client. While Tkinter lacks some basic functionalities that come pre-built with web browsers, such as text-field pattern matching, it has been well-suited for the product's requirements due to its ease of use and flexibility.

Jupyter Notebook is a tool for interactive data analysis and scientific computing. The web-based notebook development environment was designed to create documents that contain lines of code, mathematical equations, graphical visualizations, and descriptive text. (Jupyter Notebook Documentation, n.d.) The most considered benefit of Jupyter Notebook in the development phase of this thesis, was its ease of use for testing the application components sequentially. By writing a series of cells containing varying Python code, the author of this thesis was able to execute each cell independently for iterative inspection and followed by potential debugging. This approach allowed for the implementation of the defined project management method, by fully embodying the the steps required for development of individual components before integrating them into their final environment. The extended version of Jupyter Notebook, which was released on Microsoft's development environment extension marketplace, was utilized to implement the aforementioned functionality.

4.2 Term frequency - inverse document frequency algorithm

TF-IDF algorithm was described by Ramos (2003, pp. 1, 2) as a statistical weighing method, however, its direct outcomes were characterized as naturally deterministic. According to Ramos, (2003, pp. 2, 4) TF-IDF is a well-established and effective weighting scheme, ideal for further incorporation into more complex implementations such as database search systems. The "TF" part

of the algorithm was designed to calculate the **frequency** of each **term** within a document, based on the length of that document. The latter IDF part stands for **inverse document frequency**, which was designed to calculate the logarithm of the proportion of documents containing a specific term, within the entire collection of documents. The multiplied product of TF and IDF computes the importance score of a measured term in relation to the entire collection of documents. The TF-IDF algorithm produced higher scores for words that occurred frequently within a small number of documents, or in just one document. This algorithm does not consider any lexical variations of the given tokens, such as inflections or derivations. (Ramos, 2003, p. 2)

The author of this thesis utilized the Scikit-learn 1.4.2 library, which is licensed under the BSD License and employed the "TfidfVectorizer" method class for its development. This class contains methods for straightforward application of the TF-IDF classification technique on all tokens found in documents from a given collection. Tokens with the highest scores were converted into "feature names" and returned as strings in a multi-dimensional array. TfidfVectorizer was designed to intake multiple parameters during initialization, the prototype utilized the English stop-words parameter for removal of uninformative words from the documents. Additionally, the “max_df” and “min_df” parameters, which determined the maximum and minimum document frequency thresholds for terms to be included in the training vocabulary for classification. The “max_features” parameter specified the maximum number of terms that can be included in the generated TF-IDF matrix. The “fit_transform” function in this class was designed for transforming the words of provided iterable list of sentences into TF-IDF weight matrix. Followed by the “get_feature_names_out” method, applied on the generated weight matrix to extract the specified number of top rankings in their verbal state. Scikit-learn is a Python library for machine learning and is licensed under the 3-Clause BSD license. (Scikit-learn. n.d.)

4.3 Whisper speech recognition

According to Whisper’s Github documentation (OpenAI, 2023), this project was introduced as a series of general-purpose, multitasking speech synthesis models, trained on variety of languages and accents. The Whisper model family employed the encoder-decoder transformer architecture, allowing users to load various model sizes on their hardware devices. These models required allocation memory ranging from approximately 1 gigabyte to 10 gigabytes, providing flexibility for deployment across different projects. Whisper was designed to be compatible with both regular computer processing units and graphical processing units (GPUs) leveraging CUDA technology. (OpenAI, 2023)

The development project utilized the Whisper Python module version 20231117, which directed the loading a pre-trained model using the “whisper.load_model” function. Subsequently, it employed a

“transcribe” method on the provided audio file. The “transcribe” method was designed by OpenAI to internally process the entire audio file by dividing it into 30-second sections and performing autoregressive sequence-to-sequence predictions on each section, resulting in a returned string of text generated from the audio file. (OpenAI, 2023)

4.4 PyTorch

PyTorch module was described as a usability first, high-performance framework that leverages both GPU and CPU processing to accelerate the development and execution of machine learning models through its efficient handling of tensors (PyTorch Contributors, 2023). According to PyTorch Contributors (2023), their software aimed to create an accessible ecosystem by avoiding strict upfront limitations. This allowed for a consistent and seamless user experience across various hardware and software environments, enabling effortless integration with other libraries while maintaining its position as a numerator software. (PyTorch Contributors, 2023)

On their official website, PyTorch Contributors (2023) stated that their second design philosophy principles were inherited from Python’s design Zen. They illustrated this by citing two principles: "Explicit is better than implicit" and "Simple is better than complex." This meant that when designing the building blocks of PyTorch, simplicity and explicitness were prioritized over ease-of-use. (PyTorch Contributors, 2023)

PyTorch Contributors (2023) explained that rather than offering high-level APIs that could necessitate examining numerous subprocesses during troubleshooting or prove too intricate to comprehend, they created explicit operators. While these operators were designed to be more challenging to set up in a development environment, they could easily be built into maintainable API’s. They referenced two highly influential works on their website, detailing the issues with incorporating excessive functionality into lower levels of the application stack, and cautioning against oversimplifying system design by ignoring the distinctions between the provided resources and their impact on performance. (PyTorch Contributors, 2023)

4.5 Llama.cpp

“Llama.cpp” is an open source language learning model interface, developed in the C class of programming languages, to deliver the highest efficiency, without any additional dependencies. The goal of the development of this module, is to provide the open language models such as LLaMA, Mistral 7b or Falcon, with stable, cross-platform environment. “Llama.cpp” works with a variety of hardware and can even combine processing units with a hybrid hardware interface, directed towards the large models that can’t fully fit on a consumer graphics card. (Gerganov,

2024) The development utilised the “Llama.cpp” Python binding module called “llama_cpp_python” Version 0.2.64. Both of these libraries are licensed under the same MIT license.

The “Llama.cpp” project provided a development foundation for the "ggml" tensor, which aims to offer efficient multidimensional array handling, and quantization or approximate weight scaling capabilities across different manufacturers’ processing units. Without depending on third-party dependencies, the goal of ggml library is to minimize runtime memory allocations. (Gerganov, 2024)

In an effort to reduce the processing power needed to run complex machine learning models on consumer hardware, Georgi Gerganov later developed the GGUF binary file format, which is optimized for fast loading and saving of models in a clear and unambiguous manner. With its ease of use, a single trained machine learning model file enables simple sharing and streamlined applications. GGUF not only encodes before mentioned tensors, but also includes various sets of standardized metadata key value pairs to enhance the quantization function. (Gerganov, 2024)

5 Design and implementation of the product

The lecture summarizer tool has been developed to tackle uncertainties that occur when students are unsure if they should take notes or concentrate on the lecture, particularly for individuals with learning difficulties. Drawing from extensive research that highlighted various cognitive benefits of longhand note-taking beyond mere information retention, the design of this software focused on facilitating detailed information retention, identifying important terms, and revising given information, as suggested by both note-taking research and artificial intelligence research. It achieves this by providing users with a selection of extracted documents from the lecture in different situations. This can help ensure that all students have access to the information they need in order to succeed. In other words, this software aims to complement, not replace, traditional note-taking by offering an interactively generated summary of the lecture's content after the student's attendance. The tool's design was informed by theoretical findings from educational and pedagogical research on the note-taking. The primary objective of this tool was to accurately convert spoken language into written text and summarize the transcription into informative notes.

This chapter provides a description of the development environment, as well as a comprehensive explanation of the program's various components and how they were integrated together. It also described the testing environment and conditions of the prototype, stating what was sought during the evaluation.

5.1 Development and production environment

The prototype was developed with the Python programming language using virtual python interpreter. The program structure is utilising object-oriented functionality for various graphical components and their corresponding styles. Python was chosen for the simplistic syntax and high community support in modules. The prototype's fundamental design can be translated into multiple programming languages, which may result in enhanced performance if developed using a lower-level language such as C. This would enable to discard multiple binding libraries and further enhance performance. Dependencies and testing packages were downloaded using the pip package manager.

The **development environment** consisted of virtual Python environment generated by the Pipenv library. The interpreter was executed via a desktop environment of GNU/Linux and a GPU-powered terminal running on version 6.8.0 of the generic Linux kernel. The GPU drivers used for this task were proprietary Nvidia version 550.67 and CUDA version 12.4. The choice of machine learning models was based on their performance results, which were evaluated by comparing their loading times, total memory allocations, and overall task-specific handling efficiencies. The

artificially generated results were evaluated by the human author of this thesis and were compared to the original source medium for factuality of information. With the help of official Python and Jupyter plugins, the initial testing of the components and models took place in Jupyter Notebooks. Jupyter Notebooks facilitated the sequential execution of numerous functions with varying parameters, making it convenient for testing different machine learning model sizes and adjusted versions. Variations of development environments were documented in the tool's source code, allowing for installation, inspection and potential contributions from others.

5.2 Software architecture

The user interface was built using Tkinter, which was chosen for its clean and minimalistic graphical design, simple implementation without requiring a server-client based programming interface, accessibility across multiple platforms while avoiding the distractions associated with web browsers. The following paragraphs provide comprehensive details on how the prototype functions behind the user interface, following the path of the user's interactions throughout their journey.

The application user journey commenced by launching the "main.py" file, which initialized the Tkinter window framework and set the appropriate title bar for each supported platform. The window was designed to operate within a Tkinter event loop on the main thread, ensuring it always remained responsive and ready for user input. Furthermore, the "main.py" file defined the styles and other global variables used consistently across the remainder of the interface frames. The final step of the initialization function in the main file involved displaying the starting page frame. The source code was organized into several files, each dedicated to a specific interface component of the application. During the initialization of the main Tkinter event loop, these files were mapped accordingly.

The initial screen of the prototype was designed to present users with a choice of an audio transcription device, sourced from the Sounddevice "query_devices" function. This enabled retrieval of all physical and virtual audio devices present on the user's system at the time of loading the initial frame. The "combobox" Tkinter widget was implemented to serve the user with a filtered set of input devices based on the positive count of their input channels. To capture output audio, such as speech from an online or pre-recorded lecture, users could create a loopback device using their audio driver interface, which redirected the output stream back into the input. Once the input device for transcription was chosen, the audio recording commenced following the pressing of the "Start Recording" button. This event initiated an isolated thread that began the Sounddevice audio recording method while updating status indication variables and awaiting further user input to terminate the recording process on the main Tkinter thread. The isolated thread's loop started by

continuously recording audio into arrays of samples using the input stream method, recorded at a 16 kHz frequency, until it was signalled to stop through a termination condition for the while loop. The rest of the function joined all the recorded arrays into a single Numpy array and saved the recorded audio in a specified format using the Soundfile's write method. Following the successful saving of the audio file on the device, a function executed that set a global variable containing the generated audio file's name and forwarded the user to the next application component."

The transcription page was created to choose the most recently generated audio file, load the small Whisper model, transcribe the audio file into text, and display the text in a modifiable text area for revision. This was achieved by retrieving the created audio file name from the global variable and executing the Whisper Python binding to transcribe given audio file. The transcription function operates on a separate, isolated thread, much like the audio recording process. This enabled for continuous updates of the text area with transcribed text and served to notify the user that their transcription is currently being produced.

After the transcription generation process had been completed, the resulting text is inserted into the text area for users to revise, edit, and, if necessary, highlight important sections of the lecture to cite in the final generated notes. Tkinter's text area widget was altered to incorporate a pattern-matching feature, allowing users to locate specific keywords or segments within their transcribed lectures. This function was designed to wait for the user to press the key combination "Ctrl+F" to reveal an input field at the bottom of the window. After the user entered their text into the field and pressed the Enter key, the subsequent match from the cursor's position of the character sequence was selected by the cursor. This was accomplished by utilizing the Tkinter search method within the text area field, eliminating the existing character tag, and marking the discovered sequence of characters with a new tag while simultaneously scrolling to that specific location. The purpose of this approach was to allow users to swiftly locate and revise keywords or phrases they recalled from a that lecture. As an illustration, if during a lecture the professor mentioned specific deadlines or crucial facts that students failed to mark down in their notebooks, then revisiting and searching through the transcribed text on this application frame fulfils this precise function. Additionally, a subsequent method was developed which enabled users to emphasize specific words or sections within a pair of asterisks. If the user had marked certain areas in their transcript, these highlighted parts were extracted with the aid of the "findall" function from the regular expression library and added to the top of the extracted transcription document for improved convenience. This function triggers upon pressing the “extract keywords” button which also leads the user to a new application page.

The keywords page was designed to automatically extract the keywords from the provided transcription on load. The extraction process began by saving the modified and sorted transcription

from the previous frame onto the computer's storage, which was then passed into the "extract_topics" function. The "extract_topics" function was created with the purpose of preprocessing the transcription and extracting the keywords. The function splits the transcription into a list of sentences, converts all words to lowercase, and groups together the inflected forms of words to improve the accuracy of the TF-IDF measures. Keywords were extracted based on a minimum document frequency requirement of at least two documents containing the given keyword, and a maximum term frequency threshold of 30% of all documents in the collection that contain the given keyword. By default, 20 keywords were selected for extraction based on the specified criteria. After the extraction, the keywords were automatically inserted into the text area, allowing users to review and modify them as needed. Once users have completed revisions on the extracted keywords, they can simply click a button to advance to the final step of the process.

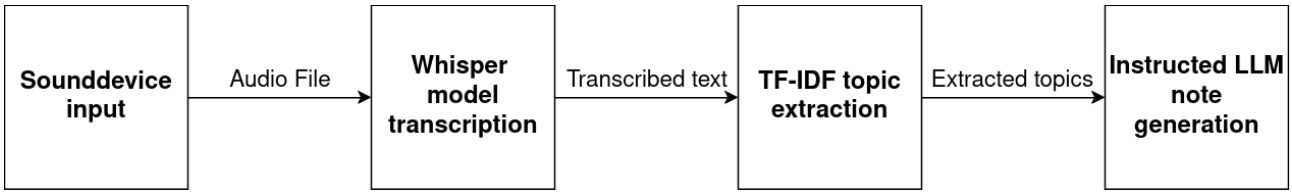
The final frame was added to enable users to make any last revisions on their generated notes by utilizing the same text area widget found throughout the rest of the application. Before this, users were prompted to select their desired GGUF language learning model, which then automatically generated their final text based on that choice. The generated text was inserted into the text area for additional modification, expansion, and most importantly, final revision, before saving the file with the same name prefix as all previously generated files for the recording and the transcription. Retrieval-augmented generation was intended to be integrated alongside language generation capabilities. This feature aimed to enable students to select books used during lecture and enable the LLM to access fragmented transcription for more accurate generation. However, due to limitations in author's expertise, the implementation of vector databases with document fragmentation features, required for successful RAG implementation was beyond author's current capabilities.

To develop the process for selecting the LLM, Tkinter's file dialog module was utilized, which was initiated when configuring the parameters for the chosen LLM. The "llama_cpp" Python binding provided a function that executed the chosen LLM using a context prompt. This prompt incorporated the extracted TF-IDF keywords and a directive for generating notes based on the supplied contextual data.

In summary, Figure 3 was created to clearly depict the structure of the software prototype. The initial audio recording interface was created utilizing the Sounddevice module, allowing users to easily capture audio on various operating systems. The resulting audio file was stored in a designated directory for future reference once the recording session ended at the user's command. The user was directed to the following application screen, where the Whisper model began transcribing the generated audio file in the background, keeping the user informed about the progress. Once the transcription process had concluded, the complete text was inserted into an

editing interface. The editing interface allowed users to revise the generated text and selectively emphasize specific passages. Following any necessary revisions or additions to the transcribed text, the user was able to proceed by clicking on a button that took them to the next application screen. The revised transcription was saved in the designated directory with the highlighted sections copied to the top of that document. As illustrated in Figure 3, the modified transcription output was passed into the TF-IDF topic extraction method, prior to being subjected to word lemmatization and document partitioning. The final component of this application was designed to incorporate the extracted keywords as contextual information in the note generation process. This feature enabled users to select their preferred GGUF language learning model, thereby allowing them to tailor the output to their individual needs and preferences. Additionally, the generated notes can be edited by the user, providing an opportunity for final revision of the material and refinement of the notes before saving them.

Figure 3 Software architecture flowchart



5.3 Software testing

The prototype was consistently tested during the development process, utilizing Tkinter's built-in event loop to display the interface within the development environment and enable debugging of ran components. This approach allowed for a direct subjecting method of the graphical interface to tests, leveraging Tkinter's built-in functionality to facilitate rapid iteration and refinement.

In addition to relying on Tkinter's event loop, sequential execution testing was also conducted as a means of verifying the behaviour of varying algorithms. By manually supervising the application's component execution within Jupyter notebooks, it was possible to thoroughly examine each component's performance and identify any issues or areas for improvement. The independent internal modules of the application were subjected to data visualisation and testing using Jupyter notebooks. This approach enabled to detach specific components and test their functionality in isolation, thereby ensuring that individual components functioned correctly before being integrated into the larger system.

The use of Tkinter's event loop and manual testing allowed for a comprehensive evaluation of the graphical interface, enabling the author of this thesis to instantly identify and rectify any issues that

may arise during the development process. This approach also facilitated rapid iteration and refinement, as changes could be quickly transformed and tested within the Jupyter notebooks environment.

For the purpose of testing the audio recording, transcription and topic extraction functionality, a set of 5 publicly available Youtube videos from topics like computer science and natural sciences were chosen and evaluated. The selection criteria included video duration of about 1 hour, without necessarily native english speakers as well as welcoming some background noise. To prepare the data for testing, a virtual audio loop-back device was created on a testing machine and streamed the output audio channel back into the input, resulting in successful audio recording.

The accuracy of transcribed text was verified by reviewing the transcript while simultaneously listening to the corresponding recording. This approach enabled to identify any differences or errors efficiently, thus streamlining the validation process. The outcome of different model variations also informed the decision regarding model size selection for release.

To assess the performance of TF-IDF, a manual inspection of the text corpus was conducted in order to identify key terms that were prominent based on the author’s understanding of the text data. This allowed for establishment of baseline data for evaluating the effectiveness of the TF-IDF algorithm. The manually identified keywords were compared with those generated by the TF-IDF algorithm, providing clear and conclusive results.

The evaluation of natural language generation was conducted using a pre-trained "gpt4all-falcon-newbpe-q4_0. gguf" large language model. This assessment did not aim to verify the factual accuracy of generated content, as multiple prior studies had evaluated variations of similar models, furthermore, the selection of LLM depended on end-user preference, allowing them to choose a model with the latest knowledge. The author assessed whether the instructed LLM correctly utilized the provided keywords as contextual information, and also examined in how much detail the generated text elaborated on each keyword.

6 Results

This chapter presents the results of the investigation into lecture note-taking accompanied by evaluation of the product's capabilities. The following sub-chapter summarizes the key findings from the product evaluation, which aimed to assess the capabilities and limitations of artificially generated transcription, topic extraction and natural language generation. The analysis provided insight into how well the individual components performed their respective tasks. These results serve as a foundation for drawing conclusions about the strengths and weaknesses of this approach, as well as informing about the tool's future plans to improve and expand its capabilities.

6.1 Developed product evaluation

The analysis of the product's testing results revealed consistent performance and accuracy measures, as defined by predetermined criteria. The exposed media was clearly transcribed with minimal errors, despite using the "small" Whisper model size. In retrospect, the processing time of the transcription proved highly dependent on the processing power of the end-user's machine; it utilized CUDA processing if available GPU drivers were found on the system, otherwise the model loaded to central processing unit.

Whisper model handled various accents and background noise exceptionally well, resulting in a precise transcript for student's revision. The "small" variation of model struggled with profession-specific homophones, words that only sounded similar but differed in meaning, one example being the "bash" words from Linux shell lectures, which were sometimes transcribed as "dash". The emphasis on revision enabled retrospective inspection of the lecture, thereby applying the findings reported by many referred researchers from the pedagogy field. This could also be interpreted as strengthening of connections described by Thorndike from the psychology field.

Despite the fact that the aforementioned proprietary summarization-capable large language learning models can process thousands of tokens, which would be the considered amount of tokens for a regular lecture transcription, these raw models are not applicable on consumer hardware. As a result, their potential application did not extend to the scope of this product. Instead of summarizing the lectures, the application of TF-IDF algorithm allowed for extraction of a specified number of topics. The algorithm was supplied with english stop words and lemmatization methods to reduce the amount of tokens for computation and gain better accuracy. The results achieved with TF-IDF met expectations, as determined through a human evaluation of specific relevance criteria on both the original source material and the generated output. The findings revealed that approximately 50% of the keywords matched those selected by human evaluator,

which was sufficient to meet the scope requirements for the product since users are able to revise and modify these words, facilitating in additional review of the lecture material.

The note generation assessment demonstrated that fine-tuned language models in consumer-applicable formats can generate long, detailed, and factual responses for the end-user. This testing involved variations of different keywords related to the same topic, as well as highlighted sections from transcriptions used for paraphrasing purposes. The presence of these highlights had a negative impact on note generation when inserted into prompt with instructions to paraphrase them. In some cases, this led the LLM to either generate text that closely followed the highlights or exhibit hallucinations due to an excessive context size, as there was no limit imposed on the highlighted sections. As a result the highlights were removed from the LLM prompt and appended to the top of the modified transcription file for ease of access. Each provided keyword was formatted as a heading to a section that described it in detail. Before saving the notes, the students should revise the material again and modify it if needed. Each provided keyword was formatted as a heading to a section that described it in detail. Before saving the generated notes, the students were able to revise the material again and modify the output.

6.2 Conclusion

With further development and advancements in local natural language generation machine learning models, the Lecture Summarizer tool has the potential to serve as a worthy alternative for automating note-taking tasks for students who struggle with simultaneous writing and listening. The tool is currently operational, however, it requires supervision of generated outputs from all machine learning models and statistical algorithms to ensure accuracy.

While this may be viewed as a limitation, research findings in the education field suggest that the **revision** of given material is one of the most important actions students can take after attending a lecture. Consequently, the lack of efficiency in generative AI technology could be interpreted as beneficial for student learning, as it forces them to correct and refine generated information. These findings provide strong evidence that answers the first research question: “Can speech synthesis systems combined with advanced text generation and fact-checking be used to enhance student engagement during lectures, and if so, how does this impact their overall academic performance?” The results affirm the product's usefulness in promoting retention and encouraging revision, as supported by both background education research as well as Thorndike’s findings.

The research into assistive modern technology yielded diverse outcomes. While some studies, such as those conducted by Forgrave, highlight numerous benefits associated with digital technologies, others emphasize the distracting nature of these mediums. To address the second

research question: "How do combined modern technologies impact the learning experience for students, and what adaptations can be made to enhance accessibility and usability?" The effectiveness of modern digital technologies in assisting users depends on their responsible use, avoiding the pitfalls of limitless internet access and maintaining focus on the lecture content. Designed as an isolated desktop program, this tool minimizes potential distractions. Adaptations such as voice commanding and personalization of GUI could provide an inclusive environment for various students.

6.3 Future plans

The future development of the Lecture Summarizer would most notably involve the application of RAG, leveraging pre-trained language models and retrieval techniques to generate notes that are accurate and contextually relevant to the student’s lecture. To improve readability and organization of generated notes, markdown formatting output would be implemented by fine-tuning a specific LLM, allowing users to format their notes using a syntax like the one in Markdown format. To appeal to more diverse user’s needs, selection of Whisper transcription model size would be implemented, allowing users to choose between smaller models for faster generation or larger models for more accurate results. The application currently lacks descriptive error handling mechanisms to detect and recover from errors effectively, application of effective error handling would supply the users with clear feedback about any issues encountered during the generation process might .

These development ideas were conceptualized during the analysis and development phases with the goal to enhance the overall functionality and usability of the automatic note-taking system, making it more efficient and effective for users.

7 Summary

The purpose of this thesis was to investigate note-taking practices and their impact on student comprehension of material. The thesis also explored the conception of artificial intelligence and other new technologies that could enhance the student learning process. The research began by qualitatively examining influential studies on the act of note-taking and described the practices for generating notes among students, including both traditional handwriting method and digital approaches like typing or dictating on laptops. This literature review revealed that effective **handwritten note-taking** is closely tied to improved memory connections and better comprehension, making it a crucial aspect of the learning process, therefore diminishing the purpose of the development product. Additionally, existing studies on digital technology usage within educational settings had revealed both the drawbacks, including distractions associated with these devices, as well as the benefits they provide to students who are differently abled, experience difficulties concentrating during lectures or write at an inadequate pace for effective retention. These concluded findings were employed in the software development strategy to conceptualize functionality of the product, providing these students with the best possible user experience.

The development of an AI tool designed to assist students in their note-taking endeavors was composed of applying a user-centered design approach, aimed at understanding user's requirements and preferences. The concept was developed according to the findings from prior qualitative research analysis of the target group, followed by prototyping and testing of the product. This iterative process allowed for the refinement of features such as topic extraction and overall quality-of-experience features that could be incorporated into a comprehensive, semi-automatic note-taking system.

The research questions guiding this study were answered early on during the qualitative research phase of this thesis. Research studies from the education field revealed that students are prone to distractions when using computer software during lectures, but if used correctly, digital devices and customized software can significantly enhance student engagement and positively impact their performance. In conclusion, this thesis explores variations of note-takers and examining how the note-taking practice impacts different students. It also explains how modern AI works and demonstrates its potential for integration with other software, highlighting opportunities for utilization in educational environments by addressing inadequacies, discovered in the education field.

The author acquired significant new knowledge in psychology, philosophy, and computer science. The study of education and note-taking revealed that taking notes by hand, even without intending

to retain the paper, can enhance memorization of written material. The research into philosophy proved personally valuable and also allowed for understanding of how researchers were able to abstract the patterns of intelligence. Moreover, the author gained hands-on experience with various machine learning models and technologies for future application or composition of new software.

References

Agostinelli, F., Hocquet, G., Singh, S., & Baldi, P. (2018). From reinforcement learning to deep reinforcement learning: An overview. In *Braverman Readings in Machine Learning. Key Ideas from Inception to Current State: International Conference Commemorating the 40th Anniversary of Emmanuil Braverman's Decease, Boston, MA, USA, April 28-30, 2017, Invited Talks* (pp. 298-328). Springer International Publishing. https://doi.org/10.1007/978-3-319-99492-5_13

Aristotle, A. (1986). *De anima (On the soul)*. London: Penguin. Retrieved April 6, 2024, from https://earlhaig.ca/departments/socialscience/downloads/Mr.%20Wittmann/2019-2020%20HZT4U1%20Handouts/HZT4U1%20Extra%20Readings/Aristotle,%20De%20Anima.pdf

Bishop, C. M. (1994). Neural networks and their applications. *Review of scientific instruments*, *65*(6), 1803-1832. https://doi.org/10.1063/1.1144830

Bohay, M., Blakely, D. P., Tamplin, A. K., & Radvansky, G. A. (2011). Note taking, review, memory, and comprehension. *The American journal of psychology, 124*(1), 63–73. Retrieved March 30, 2024, from https://memorylab.nd.edu/assets/258700/bohay_blakely_tamplin_radvansky_2011_american_journal_of_psychology_.pdf

Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983). An overview of machine learning. *Machine learning*, 3-23. https://doi.org/10.1016/B978-0-08-051054-5.50005-4

Forgrave, K. E. (2002). Assistive Technology: Empowering Students with Learning Disabilities. *The Clearing House: A Journal of Educational Strategies, Issues and Ideas, 75*(3), 122–126. https://doi.org/10.1080/00098650209599250

Free Software Foundation. (2007). *GNU General Public License version 3.0*. GNU Operating System. Retrieved April 10, 2024, from https://www.gnu.org/licenses/gpl-3.0.en.html

Friedman, M. C. (2014). Notes on note-taking: Review of research and insights for students and instructors. *Harvard Initiative for Learning and Teaching,* 1–34. Retrieved April 1, 2024, from https://u.osu.edu/rhodes-disalvo.1/files/2017/03/Notes-on-Note-Taking-grs2kq.pdf

Gerganov, G. (2024) *Llama.cpp README.md.* Github. Retrieved April 20, 2024, from https://github.com/ggerganov/llama.cpp/blob/master/README.md

Gerganov, G. (2024) *ggml documentation.* Github. Retrieved April 20, 2024, from https://github.com/ggerganov/ggml/blob/master/docs/gguf.md

Goundar, S. (2014). The distraction of technology in the classroom. *Journal of Education & Human Development*, *3*(1), 211-229. Retrieved April 10, 2024, from https://www.academia.edu/download/34860916/Published_Paper_-_The_Distraction_of_Technology_in_the_Classroom.pdf

Hartley, J., & Davies, I. K. (1978). Note-taking: A critical review. *Programmed Learning and*

Educational Technology, 15(3), 207–224. https://doi.org/10.1080/0033039780150305

Jupyter Notebook Documentation. (n.d.). *The Jupyter Notebook.* Read the Docs. Retrieved April 21, 2024, from https://jupyter-notebook.readthedocs.io/en/latest/notebook.html

Kiewra, K. A. (1989). A review of note-taking: The encoding-storage paradigm and beyond. *Educational Psychology Review, 1*, 147-172. https://doi.org/10.1007/bf01326640

Krogh, A. (2008). What are artificial neural networks?. *Nature biotechnology*, *26*(2), 195-197. https://doi.org/10.1038/nbt1386

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, *33*, 9459-9474. Retrieved March 22, 2024, from https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf

Marin, L., & Sturm, S. (2021). 'Why aren't you taking any notes?' On note-taking as a collective gesture. *Educational Philosophy and Theory, 53*(13), 1399–1406. Retrieved April 10, 2024, from https://doi.org/10.1080/00131857.2020.1744131

McCarthy, J. (2007). What is artificial intelligence. Retrieved March 30, 2024, from http://cse.unl.edu/~choueiry/S09-476-876/Documents/whatisai.pdf

Meta. (n.d.). Llama 2. Retrieved April 10 2024 from https://llama.meta.com/llama2/

Minsky, M., & Papert, S. A. (2017). *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press.

Mueller, P. A., & Oppenheimer, D. M. (2014). The pen is mightier than the keyboard: Advantages of longhand over laptop note taking. *Psychological science*, 25(6), 1159-1168. https://doi.org/10.1177/0956797614524581

NumPy Team. (2024). NumPy. Retrieved April 17, 2024, from https://numpy.org/

OpenAI. (2023). *Whisper documentation*. Github Retrieved April 23, 2024, from https://github.com/openai/whisper/blob/main/README.md

Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., & Riedel, S. (2019). Language models as knowledge bases?. *arXiv preprint arXiv:1909.01066*. https://doi.org/10.48550/arXiv.1909.01066

Peters, D. L. (1972). Effects of note taking and rate of presentation on short-term objective test performance. *Journal of Educational Psychology*, 63(3), 276–280. Retrieved April 10, 2024, from https://doi.org/10.1037/h0032647

Pu, X., Gao, M., & Wan, X. (2023). Summarization is (almost) dead. *arXiv preprint arXiv:2309.09558.* https://doi.org/10.48550/arXiv.2309.09558

Python Software Foundation. (2024). *Python license*. Retrieved April 25, 2024, from https://docs.python.org/3/license.html

Python Software Foundation. (2024). *tkinter — Python interface to Tcl/Tk.* Retrieved March 29, 2024, from https://docs.python.org/3/library/tkinter.html

Python Software Foundation. (2023). *Whetting your appetite*. The Python Tutorial. Retrieved April 1, 2024, from https://docs.python.org/3.10/tutorial/appetite.html#whetting-your-appetite

Python SoundDevice. (2023). *SoundDevice documentation*. Read the Docs. Retrieved April 21, 2024, from https://python-sounddevice.readthedocs.io/en/latest/index.html

PyTorch Contributors. (2023). *PyTorch Design Philosophy.* Read the Docs. Retrieved April 14, 2024, from https://pytorch.org/docs/stable/community/design.html

PyTorch Contributors. (2023). *PyTorch documentation.* Read the Docs. Retrieved April 14, 2024, from https://pytorch.org/docs/stable/index.html

Ramos, J. (2003, December). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning* (Vol. 242, No. 1, pp. 29-48). Retrieved April 10, 2024, from https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b3bf6373ff41a115197cb5b30e57830c16130c2c

Rogers, Donna L. "A paradigm shift: Technology integration for higher education in the new millennium." *AACE Review (Formerly AACE Journal)* 1.13 (2000): 19-33. https://www.learntechlib.org/p/8058/article_8058.pdf

Royce, W. W. (2021). Managing the development of large software systems (1970). https://doi.org/10.7551/mitpress/12274.003.0035

Roseman, M. (2022). TkDocs - Tk Tutorial - The event loop. Retrieved March 24, 2024, from https://tkdocs.com/tutorial/eventloop.html

Scikit-learn. (n.d.). TfidfVectorizer. Retrieved April 15, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press. https://doi.org/10.1017/CBO9781107298019.002

SoundFile documentation. (2015). *python-soundfile.* Read the Docs. Retrieved April 22, 2024, from https://python-soundfile.readthedocs.io/en/0.11.0/

Thorndike, E. (1898). Some experiments on animal intelligence. Science, 7(181), 818-824. Retrieved April 2, 2024, from https://www.jstor.org/stable/pdf/1624411.pdf

Thorndike, E. L. (1933). A proof of the law of effect. Science, 77(1989), 173–175. https://doi.org/10.1126/science.77.1989.173.b

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288.* doi.org/10.48550/ARXIV.2307.09288

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*. https://doi.org/10.48550/arXiv.1706.03762

Weener, P. (1974). Note taking and student verbalization as instrumental learning activities. *Instructional Science*, *3*(1), 51-73. https://doi.org/10.1007/BF00117026

Appendix 1: Material management plan

The author of this thesis collected research documents in digital form and stored them in the "/home/jrosh/Zotero/storage/" directory which was backed up onto a USB drive on weekly basis. These materials were analysed and paraphrased sections were highlighted with the help of Zotero and it's document marking functions. No sensitive data was collected during the making of this thesis. Narrative text was generated to accompany the development part of the thesis. Jupyter Notebook files were used to mark results of the various components of the software product which were stored in a local, offline git repository. The thesis document as well as the Jupyter Notebook files were backed up on the same USB drive as the research material on the weekly basis. The software product, along with its underlying source code, was published on GitHub under a redistributable license, accompanied by documentation explaining how to install and use the software on the following hyperlink. https://github.com/JRoshthen1/LectureSummarizer