Enhancing Digital Customer Journeys: A Comparative Analysis of Knowledge Retrieval Approaches

by

Teodor Nicola-Antoniu
B.S., Computer Science and Mathematics
Clark University, 2019

Submitted to the MIT Sloan School of Management and
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Business Administration

and

Master of Science in Electrical Engineering and Computer Science

in conjunction with the Leaders for Global Operations program
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 2024

Authored by:	Teodor Nicola-Antoniu MIT Sloan School of Management and Department of Electrical Engineering and Computer Science May 10, 2024
Certified by:	Dr. Randall Davis Professor of Electrical Engineering and Computer Science Thesis Supervisor
Certified by:	Dr. Rama Ramakrishnan Professor of the Practice, AI/ML Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students
Accepted by:	Maura Herson Assistant Dean, MBA Program MIT Sloan School of Management

Enhancing Digital Customer Journeys: A Comparative Analysis of Knowledge Retrieval Approaches

by

Teodor Nicola-Antoniu

Submitted to the MIT Sloan School of Management and
Department of Electrical Engineering and Computer Science
on May 10, 2024, in partial fulfillment of the
requirements for the degrees of
Master of Business Administration
and
Master of Science in Electrical Engineering and Computer Science

Abstract

Since its early days in 2003, Amazon Web Services (AWS) has evolved rapidly. From a single service created to support its parent company's e-commerce business, AWS became a leading cloud services provider. As AWS's product offerings and customer base expanded, its support knowledge base grew proportionally. Customers looking for self-service support solutions need novel solutions to navigate such a vast repository of information.

This study explores a set of knowledge retrieval architectures designed to surface the most relevant content to customers pursuing self-service solutions within the knowledge base of a large technology company. To recommend the best content that a customer should consume next in their journey, we leverage insights about the content already seen by the customer. Our research encompasses three methodologies: semantic search utilizing large language model embeddings, a frequency-based n-gram model, and a hybrid approach integrating semantic search within a deep neural network framework. Simulations on historical data display a significant percentage of scenarios where customers would be accurately directed to the desired solution.

Our findings suggest that organizations can adopt these methodologies internally to enhance digital customer journeys and pave the way for further innovations in this domain. This study addresses the immediate challenges of navigating large-scale company knowledge bases and presents the potential for scalable self-service models.

Thesis Supervisor: Dr. Randall Davis
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Dr. Rama Ramakrishnan
Title: Professor of the Practice, AI/ML

Acknowledgments

Studying at at the Massachusetts Institute of Technology (MIT) has been the crown jewel of my academic journey, a source of pride I will always cherish. Growing up in Romania, I was a high school student deeply passionate about Computer Science and Mathematics. I watched the prodigious Matt Damon in 'Good Will Hunting' effortlessly solve complex equations on a blackboard, and I imagined MIT as a sanctuary for the extraordinarily gifted. Not even in my most romanticized dreams did I envision myself walking down the Infinite Corridor as a student. As I prepare to join the ranks of MIT alumni, I am mindful of the privilege and responsibility that accompany this honor. I am committed to being a positive force within every community that I will be a part of.

I would like to extend my sincere gratitude to my advisors, Professor Davis and Professor Ramakrishnan. Our frequent conversations were not only intellectually stimulating but also a profound source of inspiration. Their remarkable achievements and profound knowledge have instilled in me a relentless drive for depth of understanding and pursuit of excellence. Their mentorship has been invaluable, constantly pushing the boundaries of my abilities, encouraging an unwavering commitment to my research.

I would like to thank Sally Smith (LGO Class of 2012), who made the internship opportunity in AWS a reality and started forging a relationship with this incredible business division as part of the LGO partnership with Amazon, a relationship which I am hoping to continue growing.

I would like to thank Steve Ham, my manager at AWS, whose support and enthusiasm for my vision were pivotal. His guidance in navigating the organization, forging connections, and effectively pitching my idea was invaluable. I am profoundly thankful to Schalk Vorster, and to the team of seasoned scientists who, despite having just met me, generously carved out time each week to mentor and guide me. Radhika Bhargava, Kyle Hinsz and Darrell Wong were truly instrumental in my project's success and in my growth as a professional.

My sincere appreciation is extended to the LGO program for recognizing my potential and providing me with a unique opportunity to advance my studies in engineering and management at MIT. My heartfelt thanks to Thomas, Ted, Patty, Josh, Kathy, Eileen, Katja, Janka and all the other members of the LGO staff for their continued support and for making this exceptional program a reality. Additionally, I am thankful for the LGO fellowship and the Taylor family's generous financial contribution, which have made my educational journey possible.

Most importantly, I want to thank my parents, who were my biggest supporters through thick and thin for the past 9 years and counting. Their love and patience have seen me through every up and down since I first came to the United States. Distance was a challenge, but nothing compared to my visits home which offered me the much-needed boost of energy to return to the United States and continue my pursuit to reach my full potential.

Note on Proprietary Information

This thesis reflects the work and opinions of the thesis author alone. Amazon takes no position on the accuracy, recommendations, or conclusions. In the interest of protecting Amazon's competitive and proprietary information, figures, numbers, and processes presented throughout this thesis are used solely for the purpose of illustration. All graphs have been formatted to show relationships and may omit data labels and other attributes to protect actual Amazon data. In addition, all numbers are rounded, scaled by a coefficient and mostly shown in ranges.

Contents

Note on Proprietary Information 6

List of Figures **11**

1 Introduction **15**

1.1 Problem Statement 15

1.2 Goals and Approach 15

1.3 Methods Overview 17

1.3.1 Finding the next best content with Semantic Search 18

1.3.2 Finding next best content with a frequency N-Gram 18

1.3.3 Finding next best content with a hybrid: Semantic Search and Neural Networks 18

1.4 Thesis Organization 19

1.5 AWS History 19

1.6 AWS Support 21

1.7 Self Service Technology 22

2 Background **23**

2.1 Defining a customer journey 23

2.2 AWS Support properties 26

2.3 Recap and Strategic Insights 27

3 Data **29**

3.1 Data Overview 29

3.2 Data Validation 31
3.3 Customer Journey Content Touchpoints 31
3.4 Touchpoint Mapping Confidence Level 32
3.5 Filtering Touchpoints with valid URL 33
3.6 Customer Journey Length 34
3.7 URL to Text Mapping 35
3.8 Removing Single Touchpoint and Target Repeat Journeys 37
3.9 The Golden Dataset 38

4 First Design: Semantic Search **41**

4.1 Overview 41
4.2 What are Embeddings? 42
4.3 What is Semantic Search? 43
4.4 What is BERT? 45
4.5 What is SBERT? 47
4.6 Building the Semantic Search model 48
4.6.1 Picking a Pre-Trained Model 49
4.6.2 Picking a loss function 51
4.6.3 Generating samples for contrastive loss 54
4.6.4 Fine Tuning 56
4.6.5 Building a Vector Database 57
4.6.6 Recommendation System Design 58
4.7 Model Evaluation - Semantic Search 61
4.7.1 Online Evaluation through A/B Testing: 62
4.7.2 Offline Evaluation with Historical Data: 63
4.7.3 Target of a journey 65
4.8 Semantic Search Recommendation Experiments 67
4.9 Semantic Search Key Takeaways 70
4.10 Coherence Study 72

5 Second Design: Frequency N-gram **75**

5.1 Overview . . . 75

5.2 What is a N-gram? . . . 75

5.3 Building the N-Gram model . . . 76

5.3.1 The N-Gram Frequency Dictionary . . . 76

5.3.2 Recommendation System Design . . . 77

5.4 Model Evaluation – Frequency N-Gram . . . 79

5.5 N-Gram Experiment Results . . . 80

6 Third Design: Deep Learning Hybrid Model **83**

6.1 Overview . . . 83

6.2 Building the Hybrid model . . . 84

6.2.1 Training the DNN . . . 84

6.2.2 Recommendation System Design . . . 86

6.3 DNN Experiment Results . . . 87

7 Chapter 7: Recommendations & Future Work **91**

7.1 Recommendations . . . 91

7.1.1 Overview . . . 91

7.1.2 Perform Cost-Benefit Analysis . . . 91

7.1.3 Explore Broader Applications of Semantic Search . . . 92

7.2 Future Work . . . 92

7.2.1 Overview . . . 92

7.2.2 Refinement of Customer Journey Data Models . . . 92

7.2.3 Enhancements to Semantic Search Models . . . 93

7.2.4 Development of n-Gram Models . . . 93

7.2.5 Advancements in Deep Neural Networks . . . 93

7.2.6 A/B Testing for Practical Insights . . . 94

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

1-1 AWS website as it appeared in 2006, sourced from the Wayback Machine 21

2-1 Customer Journey Structure 25
2-2 AWS re:Post, an AWS Support Property 26

3-1 Sample Customer Journey 30
3-2 Customer Journey Support Case Flag 30
3-3 Filtering for Text Content Touchpoints with URL 32

4-1 Word Embeddings Spatial Representation 43
4-2 Semantic Search 44
4-3 Transformer Architecture [23] 46
4-4 Cosine Similarity Search in the context of customer journeys 47
4-5 SBERT Pre-Trained Models [13] 49
4-6 Contrastive Loss [22] 52
4-7 Triplet Loss [22] 53
4-8 Loss Function Characteristics [22] 54
4-9 Vector Database Architecture 57
4-10 Semantic Search Architecture 58
4-11 Building a Customer Journey Summary Embedding 60
4-12 Back-testing Approach 63
4-13 Single and Multiple Target Touchpoints 67
4-14 Semantic Search Results 69
4-15 Coherence Study Results 73

5-1 N-gram Architecture Design 77
5-2 N-gram Results 80

6-1 DNN Architecture Design 86
6-2 DNN Results 87

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Problem Statement

A key attribute that sets Amazon Web Services (AWS) apart in the cloud services landscape is its exemplary customer support. To uphold its standard of excellence, AWS needs to complement its rapidly expanding support knowledge base with improved knowledge retrieval and recommendation systems designed to optimally guide customers during their digital self-service journeys. Continuously improving these content navigation mechanisms can facilitate shorter, more successful, and inherently more satisfying digital customer journeys, leading to business scalability and cost savings through improved self-service capabilities.

1.2 Goals and Approach

Our research goal was to enhance the AWS Support knowledge base navigation process by predicting the content that customers were looking for and then guiding them to it through recommendations. To reach this goal we decided to design the knowledge retrieval mechanism by which AWS support content is surfaced to customers, ensuring they can access the information they need both swiftly and efficiently. The foundation of our approach was the hypothesis that by capturing and analyzing text content visited by a customer in their journey across AWS Support websites, knowledge can be derived

to enhance the accuracy and relevance of content recommendations, facilitating the customer's progress towards their goals. This hypothesis relied on the core assumption that each touchpoint in a customer journey holds valuable insights into the customer's intent and needs, which, if properly captured, can lead to more personalized and goal-oriented content suggestions. It is helpful to note that throughout this paper we refer to 'customer journeys' as the sequence of interactions a customer has with the AWS Support knowledge base. A more detailed discussion of this definition is available in Section 2.1. To empirically test this hypothesis, we explored three approaches to capture the information carried by customer journeys:

1. **Semantic Search** - utilizes the context derived from customer interactions to surface relevant content, aligning recommendations closely with their recent activity.

2. **Frequency N-Gram** – identifies and analyzes patterns in touchpoint sequences to predict potential future steps within the customer's journey.

3. **Deep Neural Networks & Semantic Search Hybrid** - synergizes the contextual proximity of semantic search with the predictive power of deep learning

Capturing insights from a customer journey was only the first part of the system architecture. The knowledge derived from the customer journey was leveraged to retrieve the most relevant content that AWS customers should consume to accomplish their objective efficiently. In each of the three approaches, the retrieved content was ranked and then a delivery mechanism was devised in the form of a recommendation system. Through these three systems, we evaluated our ability to leverage historical interaction data effectively, aiming to support and prove the hypothesis that articles previously visited by customers can indeed inform the surfacing of highly relevant content. The systems developed sit at the intersection of two closely related yet distinct fields: information retrieval systems and recommendation systems. Information retrieval systems are ideal for sourcing relevant information based on a query, which

in this use case are the insights derived from customer journeys. Complementary to information retrieval, recommendation systems are ideal for predicting user preferences, which in this case are the customer objectives. The architectures built for this research aim to synergize these two fields to create a seamless and intuitive content discovery experience for AWS Support customers, ultimately guiding them more effectively towards successfully achieving their goals. To measure the effectiveness of our architectures, we used metrics tailored for evaluating recommendation systems. We chose the following set of metrics:

- **Hit Rate:** frequency with which recommended items match target items in the test set.
- **MRR (Mean Reciprocal Rank):** mean rank of the first correct recommendation.
- **MAP (Mean Average Precision):** mean precision of recommendation lists.
- **NDCG (Normalized Discounted Cumulative Gain):** score for quality of recommendations based on their position in the recommendation list.

1.3 Methods Overview

The customer journey data was analyzed and pre-processed as a first step in the research process. At the end of that stage, a new dataset of the knowledge base for the AWS Support properties was compiled, referred to throughout this paper as "the golden dataset". With this dataset available, three distinct systems were designed to recommend the next most relevant piece of content to customers. Each system will be described in detail in the subsequent chapters; however, an overview will be provided first.

1.3.1 Finding the next best content with Semantic Search

The first model explored the potential of semantic search to navigate the web of customer support documentation, aiming to identify and recommend the next most relevant piece of content based on semantic proximity to the text content previously visited by a customer. To build an intuition for this functionality, picture a hiker navigating a thick forest searching for a spring. Semantic search is akin to advising the hiker to move in a direction based on the patterns of tree markings they've followed so far and suggesting a continuation of that pattern.

1.3.2 Finding next best content with a frequency N-Gram

In contrast, the second model, inspired by the n-gram approach from natural language processing, opts for a different strategy than semantic search. This approach is equivalent to guiding the hiker by observing the most frequently traveled paths based on the footprints leading up to the current point—suggesting the next step in the direction most others have taken from the same spot. This model forgoes textual similarity, focusing instead on the sequence of steps taken by previous customers, providing a simpler, yet effective, form of guidance. The n in n-gram stands for the number of prior steps we consider when observing the most traveled path.

1.3.3 Finding next best content with a hybrid: Semantic Search and Neural Networks

The third and final design was a hybrid system, which merged the semantic search technique from the first model design with a deep neural network architecture to better facilitate the efficient retrieval of relevant content. Picture our hiker with a smart compass that not only recognizes the familiar tree markings but also predicts which of these signs have successfully led others to the water source. It combines an understanding of the hiker's path with the wisdom of successful journeys from the past.

1.4 Thesis Organization

The thesis is comprised of seven chapters which will walk readers through our approach on leveraging technological advancements to solve a large-scale business challenge for a large technology enterprise such as AWS.

Chapter One introduces the problem statement, outlines the objectives of this research, and describes our methodological approach. It also provides a historical overview of AWS, insights into the role of AWS Support within the company, and discusses the shift towards self-service in the IT support industry, including the factors driving this trend.

Chapter Two defines the concept of the customer journey, which is central to our research, and offers background information on the AWS Support knowledge base.

In Chapter Three, we delve into the data analysis and pre-processing steps undertaken to compile the dataset necessary for developing our knowledge retrieval systems.

Chapters Four through Six detail the architecture, development, and evaluation of the three systems we created. Specifically, Chapter Four focuses on the design of the Semantic Search system, Chapter Five explores the Frequency N-gram Design, and Chapter Six discusses the design of the Deep Learning Hybrid model.

Finally, Chapter Seven presents our recommendations and suggestions for future research.

1.5 AWS History

Jeff Bezos launched Amazon in 1995 from a garage in Bellevue, Washington, with the ambition to establish the world's most customer-centric company in the world, aiming to provide unmatched convenience, variety, and value to shoppers. Originating as an online bookstore, Amazon rapidly diversified, riding the wave of the internet revolution to become an e-commerce titan that has redefined retail. Amazon is a long-standing MIT LGO partner and the e-commerce giant is brimming with the program's Alumni,

from front-line managers to all the way up to the company's most senior leadership [19]. In the early 2000's, Amazon's quickly expanding e-commerce business started to face a new set of scaling challenges. While Amazon was hiring software developers at a fast pace, their internal teams were hitting a ceiling in accelerating software development because they all had to build their own resources for each individual project, from the database to the compute and storage components. It became clear to Amazon's leadership that their internal teams required a set of common infrastructure services which all their developers could tap into, avoiding the duplicate work of the initial set-up. While building and growing Amazon.com's e-commerce building, the company had become highly skilled at running reliable, scalable, cost-effective data centers out of need. With these strengths in mind, Amazon's leadership envisioned that AWS could potentially be a complementary business providing infrastructure services to their teams of engineers.[8] Their long-term vision was to allow any organization, company or developer to run their technology applications on top of Amazon's infrastructure. "Amazon's leaders did not think of it as an internal "cloud"—the term was not widely used in the tech world yet—but that's what it was"[4]. Amazon Web Services (AWS) started in 2006 with the first two services Simple Storage Service (S3) and Elastic Compute Cloud (EC2) quickly becoming essential tools for startups by offering affordable, scalable computing resources. Fast forward to 2010, Amazon migrated all its retail services onto AWS. In 2023, AWS marked a milestone by generating $90.8 billion [20], reflecting a consistent rise in annual revenue through the years. That same year, Amazon introduced Bedrock, a one-stop API for foundational models in generative AI.

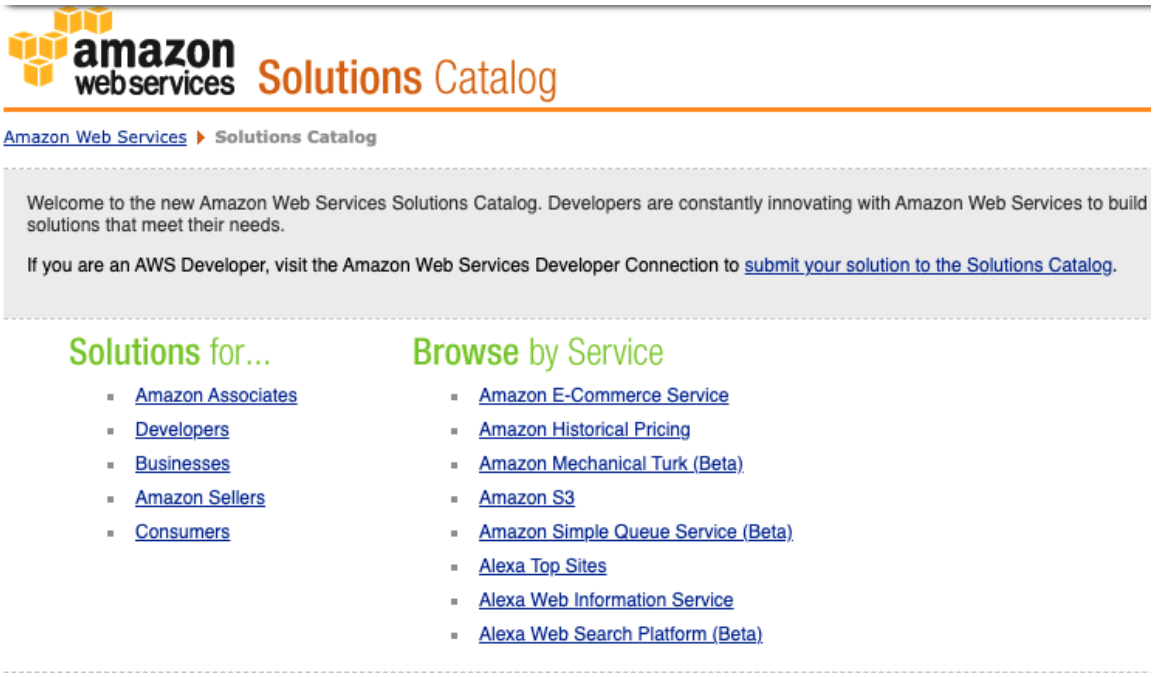

Figure 1-1: AWS website as it appeared in 2006, sourced from the Wayback Machine

1.6 AWS Support

As Amazon Web Services evolved from its founder's idea to a global cloud services provider, the sophistication and diversity of the services it provided required an equally evolved support mechanism. AWS Support is the organization that was created to fulfill this need. In the early days of AWS Support, customer queries were focused on billing, compute usage or outages associated with the relatively small number of services offered. Over time, as AWS increased its product offerings, the scope of customer queries expanded exponentially. As a result, AWS invested in a wide range of tools and resources designed to empower customers to resolve common issues independently. The AWS knowledge center, documentations, whitepapers, forums, and instructional videos form a comprehensive self-service support network. Following the "Customer Obsession" internal leadership principle, the support organization never stops focusing on further improving the customer experience by enhancing digital customer journeys.

1.7 Self Service Technology

An area of focus for AWS Support is the enhancement of self-service technologies as a means for increased customer satisfaction as well as a mechanism to increase operational efficiency. According to the "State of Global Customer Service" study undergone by Microsoft in 2017[5], 63% of customers under 35 went online to look for customer service rather than picking up the phone. In fact, people between the ages of 35 and 54 preferred going online first for troubleshooting or self-service rather than reaching out for human support. The preference for self-service has an upward trend for millennials, who first access a company's website and visit the FAQ troubleshoot section to find a solution for a problem they have before considering alternatives. Additionally, a 2018 study by Statista[21] found that 88% of U.S. customers prefer a self-service portal for a better customer experience and for troubleshooting their issues. A paper published in 2022, "Self-Service Technology: Benefits and Challenges" [11], explores how the Covid-19 pandemic has further accelerated the trends mentioned above. The paper shows that self-service technology has helped companies increase customer satisfaction, increase management efficiency and effectiveness, cut costs, and increase financial performance. More than 20 years passed since the dot-com boom and customers nowadays are technologically savvy and engaged with the digital content they navigate. Expectations increased over the years and customers care about their own consumer experience. Their patience to wait for customer care representatives and service agents to be available has decreased. People are now looking for prompt, trustworthy answers at their fingertips, which translates to self-service technology. The ability to provide customers with self-service tools is more than a competitive edge, it is an instrument for building and maintaining customer loyalty[3].

Chapter 2

Background

2.1 Defining a customer journey

We define a **'customer journey'** as a time-bound sequence of interactions that a customer engages in with AWS Support properties. Throughout this paper, the term 'AWS Support property', or simply 'property', will refer to any digital resource within the AWS Support knowledge base, including the Knowledge Center, Docs, etc. The term 'interactions' specifically denotes page visits or engagements with particular buttons or tools on a visited page.

Recognizing the diversity of customer objectives early in our research was pivotal. These objectives greatly influence the dynamics of their journeys through AWS Support properties and impacts our approach to enhancing these journeys, as we will detail in our methods section. For now, we present a breakdown of the major categories of objectives that customers pursue:

- **Troubleshooting:** Seeking solutions for specific issues or errors encountered.
- **Knowledge Building:** Learning more about AWS and best practices.
- **Research:** Investigating AWS capabilities for potential use in projects.
- **Refreshing Memory:** Revisiting topics or information previously learned.

- **Technical Reference:** Looking for specific technical details or documentation for programming purposes.

- **Decision Making:** Gathering information to decide on the best service or architecture for their needs.

- **Community Engagement:** Seeking advice or insights from AWS technical experts or other AWS users through forums.

Delineating the beginning and the end of a customer journey was a critical choice as it had significant downstream impact on the historical customer journey dataset. We chose the endpoints with a focus on the purpose of customer journeys in our research: to derive an insight about what the customer is searching for and to guide them there more efficiently. As a result, we decided that the beginning of a journey and the end of a journey are delineated by a period of inactivity from the customer, denoted by 'H'[1] hours. This period of inactivity suggests that the customer either accomplished the objective of their journey or failed the objective of their journey and requested human support.

- The beginning of a journey is defined by the first customer interaction with an AWS Support property after H hours of no interactions with AWS Support properties.

- The end of a journey is defined by the last customer interaction with an AWS Support property before H hours of no interactions with AWS Support properties.

- Consecutive interactions prior to an H hour gap are considered part of the same journey, generating a continuous chain of customer engagements with AWS Support resources, which we assume to be interconnected by a specific customer goal.

[1]*The specific value of H is not disclosed because it is proprietary data.

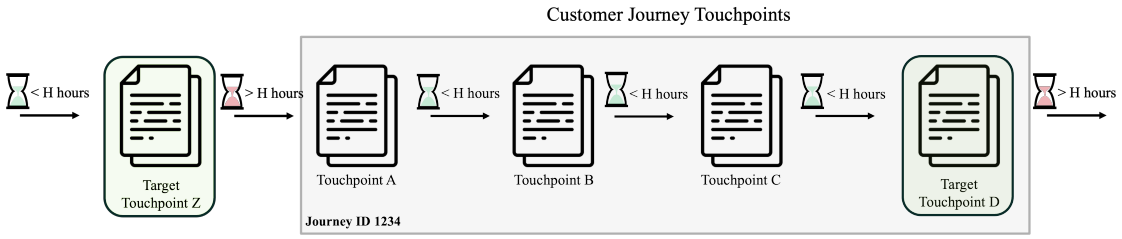

Figure 2-1: Customer Journey Structure

From here on, we will refer to each node in a customer journey as a touchpoint. A touchpoint is any point of interaction or engagement between the customer and the AWS Support knowledge base. The touchpoints within our journeys encompass a wide range of activities such as:

- **Web page Visit:** Accessing AWS Documentation, Knowledge Center articles, or blog posts.
- **Web page Interaction:** Clicking on buttons or links within AWS support pages, such as starting a new support case, subscribing to updates, or downloading whitepapers.
- **Multimedia Interaction:** Watching tutorial videos, listening to podcasts, or attending webinars offered through AWS resources.

In this research, we operate under the assumption that customer journeys within the AWS Support knowledge base hold valuable insights into customer objectives. These insights are expected to guide the recommendation of content that best meets customer needs. However, it is important to acknowledge the limitations in our dataset regarding the way these journeys are captured. The customer journey dataset is limited to interactions that occur directly within the AWS Support websites. However, customers could engage with external resources that are not recorded in our system. For example, a customer may begin their information search within the AWS knowledge base, subsequently consult additional resources on platforms such as Reddit or Stack Overflow, and later return to AWS. These external interactions, which could provide significant insights into the customer's intent and decision-making process, remain beyond the scope of our dataset.

As a result, while our models are designed to predict and recommend the next steps based on AWS interactions, they may not fully account for the influence of external content. This limitation is crucial for interpreting the effectiveness of our recommendations and underscores the need for future enhancements.

2.2 AWS Support properties

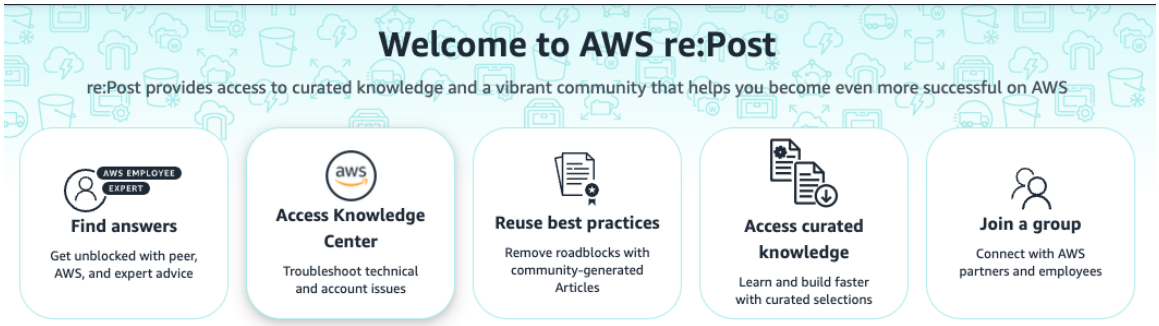

Figure 2-2: AWS re:Post, an AWS Support Property

The AWS Support knowledge base is comprised of multiple different properties that address various customer needs. These resources are designed to assist AWS customers to use the products they need and to resolve issues efficiently. Let's begin with a brief description of a few properties available at the time of our research:

- **AWS re:Post**: is an AWS-managed community featuring expert curated answers, articles and the most frequent questions and requests received from AWS customers.
- **AWS Documentation**: comprehensive manuals, developer guides, API references, and tutorials for AWS services. This documentation is crucial for both beginners and experienced users to understand and efficiently use AWS services. This content is public and free to access.
- **AWS Blogs**: updates, tutorials, and best practices from AWS experts and the community of AWS account holders. Blogs are a valuable resource for staying

informed about new features and learning about advanced use cases, available to all users.

- **AWS Whitepapers and Guides**: technical content authored by AWS and the AWS community, including technical whitepapers, technical guides, and other reference materials. This content is public and free to access.

- **AWS Support Center**: hub for managing support cases and accessing a range of troubleshooting tools. Available to account holders, it enables direct interaction with AWS Support professionals for more complex issues beyond the scope of self-service resources.

- **AWS Trusted Advisor**: resource to help customers reduce cost, increase performance, and improve security by optimizing their AWS environment Characteristic for large technology service organizations, a vast array of resources is offered to address different customer needs, from documentation to troubleshooting or simply research.

The properties within the AWS knowledge base are designed to facilitate seamless navigation between each other. For instance, "Documentations" might link to "re:Post" articles or discussion forums, generating a thick web of interconnected information. These properties are a mix of expert knowledge, written and published by AWS Support subject matter experts, and community answers in a forum vetted by other members of the community or by the internal subject matter experts.

2.3 Recap and Strategic Insights

We defined an AWS Support Customer journey as the time bounded sequence of customer touchpoints motivated by an objective. The beginning and the end of a customer touchpoints are determined by a period of H hours without any interaction between the customer and AWS Support properties. Consecutive interactions within the set timeframe are linked and become touchpoints belonging to the same customer

journey. Our touchpoint consists of a variety of different interactions with AWS properties, such as page visits, media engagement or simple page clicks.Understanding customer journeys provided critical insights into how customers use AWS Support resources and allowed us to build an appropriate software architecture that leveraged these insights and offers more tailored, responsive support solutions that meet diverse customer needs.

Chapter 3

Data

3.1 Data Overview

The AWS Support dataset obtained for this research offered a step-by-step breakdown of user engagements as they navigated across the division's properties. The dataset contained customer engagements over multiple years. Each record in the dataset represented a touchpoint, a digital interaction signifying a step in a customer's journey on one of the AWS Support properties described in Chapter 2.2. These journeys are emblematic for AWS customers' attempts to resolve their questions or challenges independently. Some touchpoints came enriched with the URL of the visited page, offering us a gateway into the text content that was consumed, while others did not, signifying other interactions such as clicks or reloads. All customer journey data used in this research did not include any personally identifiable information (PII), and it was structurally impossible to correlate any data points with PII, ensuring rigorous adherence to privacy standards.

Central to the dataset is the **'journey unique ID'**, a unique identifier tying together individual touchpoints into a cohesive journey, primarily dictated by the continuity of interactions within a timeframe of **H hours**. As described in the Customer Journey Section 2.1, the beginning and the end of a journey are delineated by a timeframe of H hours without interaction with an AWS Support property.

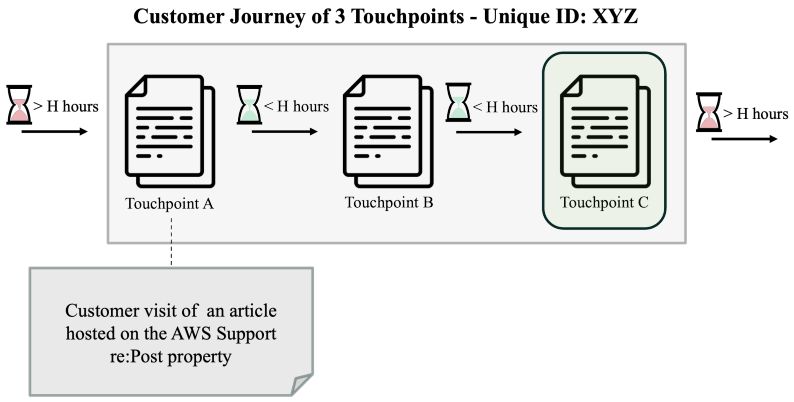

Figure 3-1: Sample Customer Journey

Another critical component of the dataset is a binary flag associated with each journey; this True or False flag indicates whether the journey culminated in the customer opening a case or not - a metric used to measure the success or failure of self-service attempts.

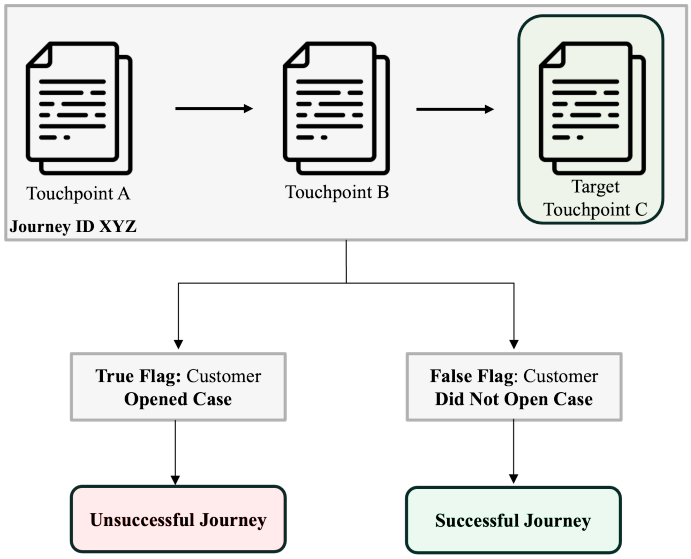

Figure 3-2: Customer Journey Support Case Flag

- An unsuccessful journey was marked by the initiation of a customer case
- A journey without an associated case was considered successful

We also identified the constraint that certain customer account types lacked the authorization to initiate technical support, thereby potentially inflating the successful

journey dataset with false positives. To mitigate this, a decision was made to exclude all touchpoints correlated with account types unable to open a case.

3.2 Data Validation

We took a methodical approach to validating the quality of the data and the structure of the customer journeys. One month of data was extracted and various metrics were calculated for reference:

- Number of unique customer journeys during the sample month
- Number of unique users visiting the AWS Support knowledge base during the sample month
- Average number of journeys undergone by users during the sample month
- Average number of customer journeys at a daily level during the sample month

The results were compared across two years of data to verify that our chosen sample month was not an outlier. The conclusion reached was that the month of data chosen to begin the design of the project was representative of a typical month within the dataset, adjusted for customer and services growth over time, and it was an ideal steppingstone toward the project objectives. The next step in the process was defining and creating a clean and reliable dataset that could be used for the modelling stage. We will refer to this clean and reliable dataset as the **"Golden Dataset"**.

3.3 Customer Journey Content Touchpoints

Each touchpoint has attributes describing the type of interaction the customer had with the support property, such as page access, interactions with media components or click actions. It quickly became clear that based on the available dataset, the most information rich touchpoints were those that had the URL of the visited page associated with them. The text content presented the potential to be processed

and used to distill the topic of interest for a customer and ultimately their [14]. In the pursuit of curating a definitive dataset of customer journeys for the modelling stage, focus was placed exclusively on touchpoints representing visits of AWS Support properties that displayed text content, incorporating the URL of the page visited. Therefore, we dropped intermediate touchpoints related to the same text content, resulting in customer journeys where each touchpoint corresponds to text content page visits.

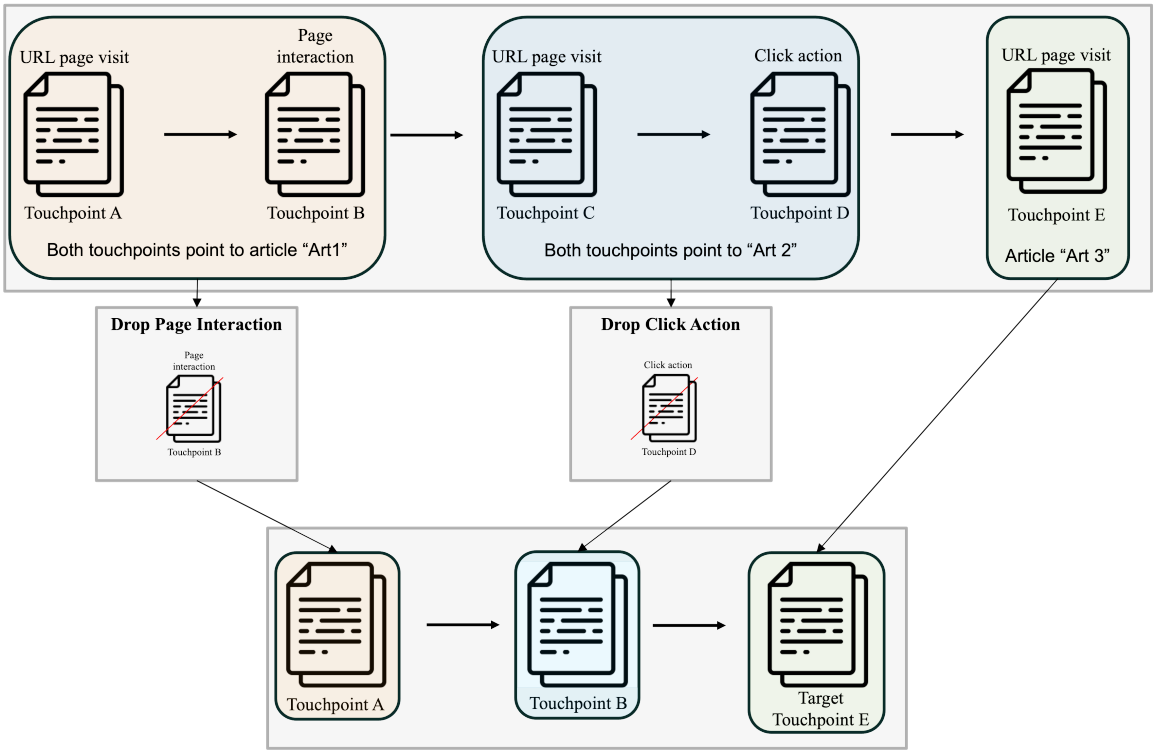

Figure 3-3: Filtering for Text Content Touchpoints with URL

3.4 Touchpoint Mapping Confidence Level

Another important aspect of the data validation and cleaning process was the customer journey confidence level. AWS customers could access Support properties through a multitude of channels. For example, customers could open a documentation property in one browser, such as Chrome, have a media property open in a Safari browser and have an AWS support blog open on their phone. Additionally, customers could have

multiple accounts so a single customer journey could be performed via multiple AWS accounts. Given all these factors, individual touchpoints were assigned to a customer journey id with a corresponding confidence level, calculated using internal business logic. For the purposes of this research project, access was only provided to the final confidence score assigned to each touchpoint. To appropriately use this information, it was important to understand how the dataset would be used in subsequent steps. Given the objective to extract insights about customer intent based on the content consumed along a series of touchpoints in a customer journey, it was essential to be certain that the sequence of touchpoints was executed by the same customer as part of the same journey. As a result, the dataset was pruned to include only those journeys where touchpoint association was ascertained with 100% confidence.

3.5 Filtering Touchpoints with valid URL

Upon analyzing customer journey touchpoints, it became clear that while they captured a wide variety of content engagement types, such as text, media, or page interactions, not all types of touchpoints carried information that could be leveraged to infer customer needs or intent. Central to our analysis is the sequence of text-based content consumed by customers, a focal point driven by our hypothesis that these interactions offer substantial insights. Therefore, the dataset had to be further filtered for touchpoints pointing to text content. To accurately identify touchpoints linked to text content consumption, it was observed that each touchpoint row carried two fields related to the AWS Support property visited, each of these fields possibly containing a valid URL. Regular expression (regex) functions were used to verify the presence of valid URLs in these fields, based on their expected format. A significant portion of touchpoints were excluded at this stage, primarily because many recorded mere interactions with a page, not necessarily the navigation to a new page, hence lacking a URL. Touchpoints without a valid URL were methodically removed from the dataset, further streamlining it for more focused analysis. While the dataset is trimmed significantly at this stage, the value lies in those touchpoints which contain

information for a customer's focus in their journey. In the available dataset, the page interactions were not mapped to specific instruments, buttons, or text within visited pages, information which could have contributed to the objective of this research. Such a potential improvement is discussed in the Future Work **??** Chapter. This careful dataset curation process is vital for the integrity and relevance of our analysis. It ensures we concentrate on those customer interactions that are most likely to yield meaningful insights into their engagement with text-based content.

3.6 Customer Journey Length

During the exploratory data analysis stage, some intriguing patterns were discovered in customer behaviors that were both insightful and critical for series of decisions made at the model development stage, described later in Chapter 4 Overview. Two patterns that particularly stood out were the customer journey length, alongside the frequency of monthly visits per unique AWS account. It was observed that the customer journey lengths exhibited a notable distribution, with a mean of 4.5 touchpoints per journey and a median of 2 touchpoints per journey. This pattern suggested a tendency among users to end their search in just a few steps, possibly due to finding what they were looking for, giving up on their objective to find an answer via self-service or losing patience and trying again later. While surprising, upon further reflection it was concluded that such behavior is representative of people's impatience for information[9], and it further validated the project's objective to maximize the efficiency of customer journey. The few steps that customers are looking to make along self-service journeys are the opportunities to guide them towards their target resource. While the analysis revealed a high concentration of short length customer journeys, with $\tilde{8}0\%$ shorter journeys than 5 touchpoints, it is worth noting that the number of users engaging in longer journey lengths remained substantial giventhe vast scale of the AWS Support operations. When it came to the number of journeys performed by unique AWS accounts, the data showed that customers perform around 1-5 journeys per day, with an exponential drop in the number of accounts who perform

6 or more journeys daily.

3.7 URL to Text Mapping

One of the core strategies we wanted to implement for finding the next best piece of support content to recommend to a customer was by analyzing the text that was read up to the point of recommendation in a customer journey. Naturally, the most important data for this approach was the text corresponding to each customer journey touchpoint. As described earlier, touchpoints provided URL's corresponding to the properties visited on AWS Support, but the text was not available in the initial dataset. To obtain this text dataset, we tried multiple approaches:

- First, tried scraping content from each page within the AWS Support knowledge base. However, the sheer volume of content led us to seek a more efficient approach.
- Next, got in touch with the different product teams who owned different AWS Support properties. This step revealed the value of connecting with the right people inside the organization.
- Lastly, teamwork, collaboration and appropriately navigating the AWS Support organization proved to be the most efficient solution. While discussing with product teams across AWS Support, we learned that a mapping between all the AWS Support URL's and their corresponding text had already been compiled and stored in an internal repository.

We proceeded to extract text from this repository, but a significant challenge arose in mapping the URLs from the git repository format to the valid URLs identified in the customer journey dataset. The internal repository URL's were partially altered from the format used in production. We overcame this by identifying the last string in the URL format as a potential unique key for mapping. We developed a function to convert between the two URL formats and managed to build our own text database with all the information we needed to extract the text corresponding to a customer

journey touchpoint. Our URL mapping method involved isolating the last string in the URL, which often corresponded to the article's name. This path varied significantly, ranging from variations in the HTTP/HTTPS prefix to different language markers or even the same article name being used across different services. To assess the quality of our matches, we employed the Levenshtein distance as a similarity score. The Levenshtein distance, also known as the edit distance, is a measure of the similarity between two strings. It's defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one word into the other[1]. This metric is widely used in applications such as spell checking, plagiarism detection, and bio-informatics, where it's essential to compare sequences of letters or nucleotides accurately. We employed this metric to find the minimum number of edits required to transform the value of the mapped URL to the key URL. Further analysis helped us set a similarity threshold, enabling us to discern what to retain or discard. To validate the completeness of the mapping, we analyzed a year worth of customer journeys, and we observed that customers visit a recurring number of unique URLs each month. This number closely matched the number of mapped URLs in the newly created text dataset. During the URL to text matching process, we also validated the URL universe, ensuring that we included only active URLs. For this, we used a used the Python requests package to ping each website and retrieve the status of the webpage. We identified that a fraction of the historical URL's in the dataset had become inactive and removed them, together with their corresponding customer touchpoints, from the dataset. This decision ensured the integrity and contextual relevance of our analysis, focusing on journeys that provided a comprehensive view of customer interactions and content consumption patterns.

3.8 Removing Single Touchpoint and Target Repeat Journeys

As part of the data refinement process, we implemented a strategic decision to exclude journeys that consist of only a single touchpoint. Such a journey would correspond to a customer accessing one AWS Support Webpage and without any additional step for H hours. The rationale behind this was our assumption that such journeys represent successful customer interactions where the customer swiftly locates the desired information and stops their search. This assumption is based on common customer behaviors, such as seeking specific references, consulting documentation for a known service, or confirming understanding of a concept. Customers in these 'one-shot' scenarios typically have a satisfying experience and do not require further guidance to navigate the AWS Support extensive information repository. Secondly, we also filtered out journeys where the final touchpoint was a repeat of an earlier touchpoint within the same journey. Picture a journey such as A – **B** – C – **B**, where each unique character corresponds to a different AWS Support property. In this example, a customer ends their journey after re-visiting a resource that they encountered earlier in their journey. This subset of data merits separate analysis as it indicates customer patterns of revisiting previously viewed resources. In line with the development of our system, one of our key criteria was to avoid recommending resources that the customer has already visited. Hence, these journeys, where the final touchpoint is a repeat visit, were not suitable for our recommendation model because we would never recommend a resource already encountered. The rationale is that customers are already aware of the resources they already visited. If they realize along the journey that a prior resource was the most beneficial towards accomplishing their goal, they would know about its existence and they would know where to find it, without the need of a recommendation system to be involved. Additionally, if a customer noticed that a recommendation system points them towards resources that they already visited, their experience could be negatively impacted, and they could end up classifying the recommendations as redundant or rudimentary.

3.9 The Golden Dataset

In the data pre-processing stage of our project, we first filtered out touchpoints that did not correspond to a page visit with an associated URL. Subsequently, we retained only those touchpoints that could be confidently assigned to a unique journey ID with 100% certainty. Given the dynamic nature of the knowledge base, we verified the validity of URLs linked to each page visit, ensuring they were still active and had not been updated, removed, or repurposed. We then mapped the text content from each unique page in the knowledge base to its corresponding URL. Additionally, we excluded journeys that consisted of only one touchpoint and journeys where the target touchpoint (the final page visit in a journey) had appeared earlier in the same journey.

At the conclusion of this thorough cleaning process, the refined "Golden Dataset" was organized into three input files, prepared for the subsequent modeling stages of the project:

- **Dictionary of Journeys**: This file contained journeys along with their associated touchpoints.
- **URL-to-Text Mapping for AWS Support URL's**: This file contained a mapping of URLs to their corresponding text content of the entire AWS Support corpus.
- **URL-to-Title Mapping**: This file contained a mapping of URLs to the titles of their corresponding AWS Support articles. The mapping was put together for a user-friendly display of a recommendation system.

Specific details about the dataset, such as the number of customer journeys in the Golden Dataset or other insights we derived during the analysis are not disclosed as they describe proprietary internal data.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

First Design: Semantic Search

4.1 Overview

With a "golden" dataset of customer journeys and a database of text corresponding to each URL in the AWS Support knowledge base, the project could proceed with the model development stage. While researching potential avenues for tackling the business problem discussed in this research, the Sentence Bert Networks paper [13] showed the most promise because the approach described in the publication closely resembled the set-up of this project. Just like in the research paper, there was a text corpus that had to be matched with the most semantically similar text from a set of options. The corpus of text that had to be matched was the customer journey and the set of options were all the possible articles a customer could go to next. The technique used to find the most semantically related item from the set of options is referred to as semantic search. As a result of groundbreaking advancements in the space of Natural Language Processing over the past decade, new methods for semantic search had recently accomplished state of the art performance on a wide set of reference benchmarks. Such technological improvements display the potential to enhance already existing systems and meet business opportunities like the one we focused on. Furthermore, AWS had already explored and implemented some of these methods internally across different areas of the business, leveraging their internal science teams. Given the success of deploying some of these novel frameworks,

we decided to evaluate how well such an approach would fit our business problem requirements. Before describing the implementation steps, we should first get familiar with the technologies we decided to employ. We will first take a closer look at word and sentence embeddings, which are the core concepts underlying semantic search. With this foundational knowledge, we will explore in depth what semantic search is and in what circumstances it is used. Next, we will discuss about two closely related Large Language Models, BERT and SBERT, which provide the infrastructure required for the semantic search approach adopted in this research.

4.2 What are Embeddings?

The field of natural language processing (NLP) is focused on creating a bridge of communication between computers and humans. However, there is a major challenge in facilitating this dialogue. Humans communicate in words and sentences, but computers only understand and process numbers. Therefore, words and sentences need to be translated into numbers in a coherent way. The solution is what we refer to in natural language processing as embeddings. Embeddings are the basic building block of most language models [7]. Embeddings are vector representations of real-world objects that machine learning and artificial intelligence systems use to understand complex knowledge domains like humans do. For instance, one of the most famous examples of word embeddings comes from Word2vec[12], a method designed to efficiently create word embeddings that was first introduced around 2013. This method showed that the vector operation “King” –“Man” + “Woman” “Queen”.

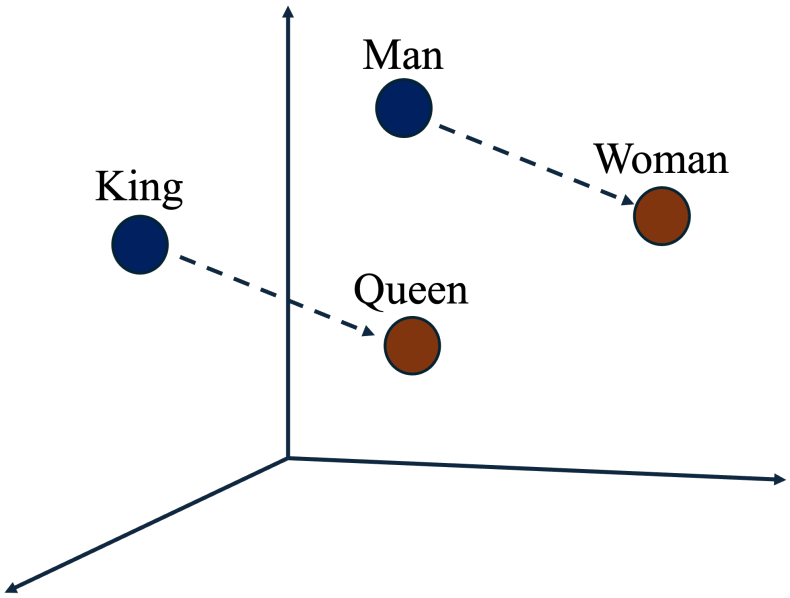

Figure 4-1: Word Embeddings Spatial Representation

This result was remarkable because it showed that vector representation carries the information necessary to capture real world relationship between words. Similarly, the words country and capital show similar properties with the pairs of words Romania – Bucharest, United States - Washington. Embeddings convert real-world objects into complex mathematical representations that capture inherent properties and relationships between real-world data. A sentence embedding is just like a word embedding, except it associates every sentence with a vector of numbers, in a coherent way, where coherence means that it carries the same properties as the word embeddings. For example, similar sentences are assigned to similar vectors, different sentences are assigned to different vectors and each of the coordinates of the vector identifies some (whether clear or obscure) property of the sentence.

4.3 What is Semantic Search?

Semantic search is a search technique focused on understanding the intent of a user's query and trying to find the most relevant documents for this intent. Traditional search techniques based on text similarity, such as TF-IDF[15], used to retrieve documents

that contained word or sentence overlaps with a given search query. Employing semantic search, the information surfaced is not only contextually pertinent but also semantically congruent with the user's needs and goals, paralleling the way a well-crafted customer journey anticipates and addresses the evolving needs of the customer. Semantic search considers synonyms, related topics and terms, and variations of words when processing a search query. This helps the search system better understand what users are searching for, even when their queries are vague or imprecise. This approach allows for a more intuitive and efficient way to navigate through the digital information landscape, tailored for the complexity and intuitiveness of human understanding, much like a well-designed customer journey. The most common technique for semantic search uses vector spaces.

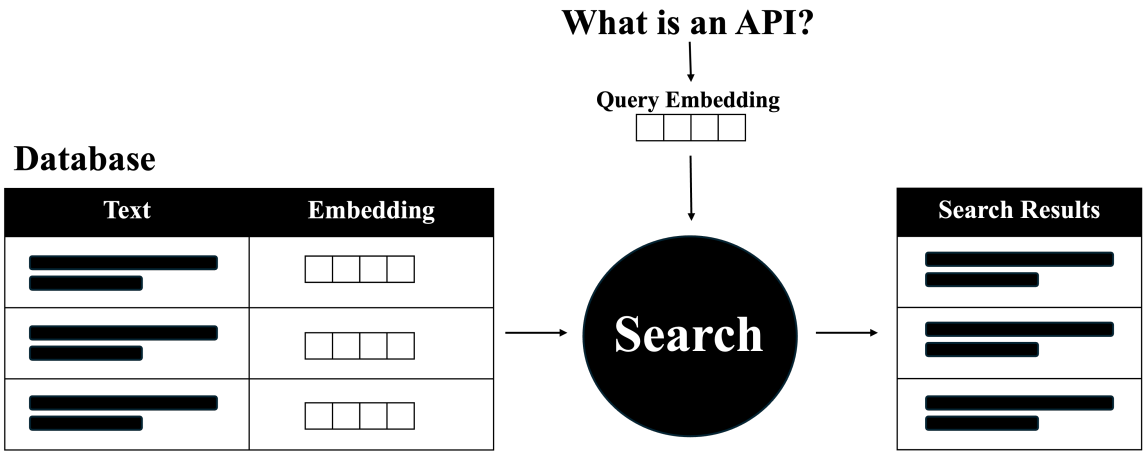

Figure 4-2: Semantic Search

In this technique, documents are mapped to a semantic vector space (embeddings) to represent the knowledge stored in the text. When a query is made, semantic search maps the query to the same vector space, and it can then retrieve the relevant documents. In contrast to keyword search, the focus is on the actual semantic content of documents. For example, if a new AWS customer searches for "application programming interface", semantic search will also find documents talking about "API". Similarly, if a developer looks up "version control system", semantic search will locate documents covering various systems like "Git" or "SVN.", not just the documents

mentioning “version” or “control” or “system”. To sum it up, semantic search works as follows:

- Uses a text embedding to turn words into vectors.
- Uses similarity (cosine difference is commonly used) to find the vector among the responses which has the highest similarity score to the vector corresponding to the query.
- Outputs the response corresponding to this most similar vector.[18]

For the purposes of this project, the query corresponds to the text from the articles visited in a customer journey. The set of responses ranked using a similarity function based on the query corresponds to all the possible articles in the AWS Support knowledge base. Later in this paper it will be shown that the output is a ranked list of most similar articles given a customer journey text query. Given the need for advanced embedding systems that could provide deep contextualized learning, we started looking into BERT.

4.4 What is BERT?

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a Machine Learning model for natural language processing developed in 2018 by a team of Google researchers[6]. The release of this model marked a new era in the field of natural language processing, signaling a paradigm shift towards a more sophisticated and context-aware language comprehension. BERT’s innovation has as its foundation the transformer architecture, a groundbreaking model that powers most of the cutting-edge large language models today, from OpenAI’s GPT-4 to Anthropic’s Claude or Google’s Gemini. Transformers use the attention mechanism to learn contextual relationships between words. The Transformer concept was first proposed in 2017 in the renowned paper “Attention Is All You Need”[23]. This model has been at the core of natural language processing advancements around the world ever since.

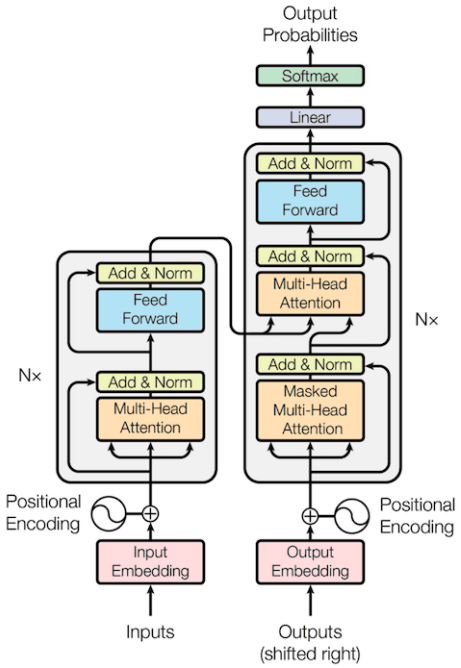

Figure 4-3: Transformer Architecture [23]

BERT is trained on vast amounts of text data using two novel strategies: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). MLM enables bidirectional learning from text by hiding a word in a sentence and training BERT to use the words on either side of the covered word (bidirectional) to predict the masked word[10]. On the other hand, NSP is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not. The two techniques coming together led to a model with a more profound understanding of language nuances.

Despite its groundbreaking capabilities, BERT also has its limitations. While BERT set a new state-of-the-art performance on sentence-pair regression tasks like semantic textual similarity (STS) at the time of its release. However, BERT requires that both sentences are fed into the network, which is very inefficient computationally. Finding the most similar pair in a collection of 10,000 sentences requires about 50 million inference computations ($\tilde{6}5$ hours) with BERT. The architecture of BERT makes it an inappropriate choice for semantic similarity search. Given this set of shortcomings when it comes to BERT semantic search, we shifted our attention to a

different model called SBERT.

4.5 What is SBERT?

Sentence-BERT (SBERT) is an NLP architecture adapted from the pre-trained BERT network[13]. This evolution of the BERT model uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.Cosine similarity is a measure used to determine how similar two vectors are irrespective of their size. Mathematically, it calculates the cosine of the angle between two vectors projected in a multi-dimensional space. This similarity is particularly useful in various applications such as text analysis, where it helps in assessing how similar two documents are based on their content.

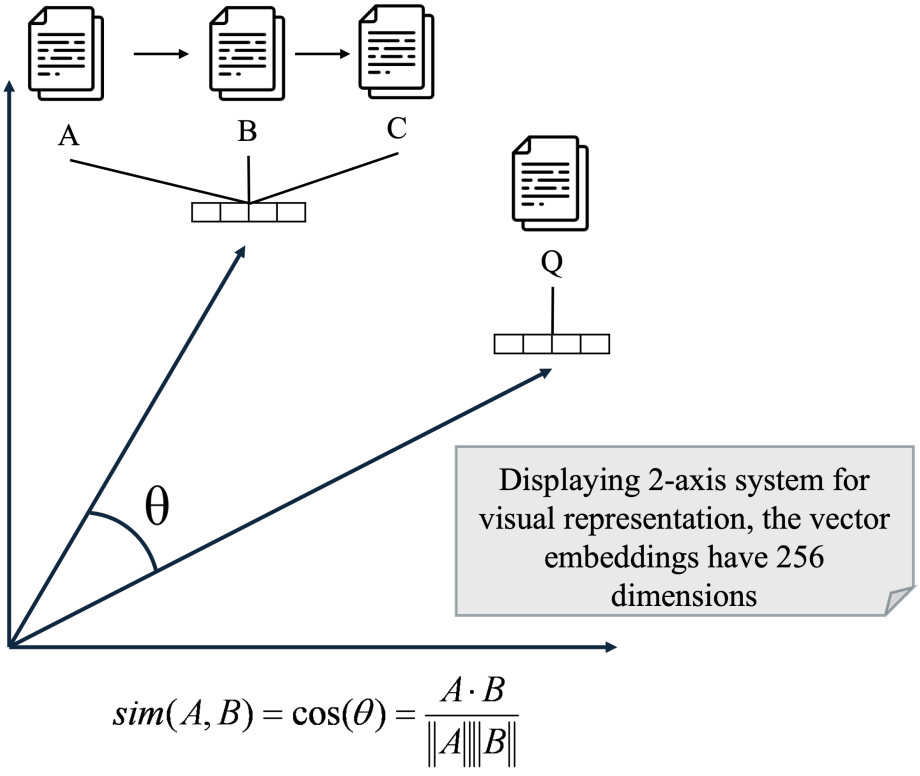

Figure 4-4: Cosine Similarity Search in the context of customer journeys

Given this functionality, BERT can be used for a set of tasks, which were previously not suitable for the model. These tasks include large-scale semantic similarity

comparison, clustering, and information retrieval via semantic search. Under this derived architecture, the effort for finding the most similar pair in a collection of 10,000 sentences is reduced from hours, in the case of BERT, to just a few seconds in the case of SBERT, while maintaining the accuracy from BERT[13]. Given our data-driven insight that the collection of text resources in the AWS Support knowledge base had more than tens of thousands of articles, it was important to find an efficient, scalable model to perform semantic search. The capability of SBERT to semantically compare two distinct text bodies is crucial, aligning perfectly with the research project's vision of parsing through the content repository to identify the most pertinent piece of content in relation to the customer's previous engagements—essentially, comparing the 'two sentences', a historical journey and a potential next step, in our scenario.From an architecture perspective, SBERT adds a pooling operation to the output of BERT to derive a fixed sized sentence embedding. The authors of SBERT experimented with three pooling strategies: using the output of the CLS-token, computing the mean of all output vectors, and computing a max-over-time of the output vectors. Based on these experiments, the mean strategy led to the best performance of the model. To fine-tune BERT, siamese and triplet networks were used to update the weights such that the sentence embeddings became semantically meaningful and could be compared via cosine-similarity. From a training standpoint, the model described in the SBERT paper was trained on a combination of the SNLI[2] and the Multi-Genre NLI. SBERT was fine-tuned with a 3-way SoftMax classifier objective function for one epoch. A batch-size of 16 was used, with the Adam optimizer at a learning rate $2e^5$, and a linear learning rate warm-up over 10% of the training data. The pooling strategy was MEAN.

4.6 Building the Semantic Search model

After identifying the SBERT architecture as the ideal candidate for large scale semantic search, the next priority was to build a model tailored for our internal business application. The following sub-chapters describe these steps in detail:

- A pre-trained base SBERT model was chosen from the set of available open-source models
- A loss function was chosen to fine-tune the base model, strengthening the relationship between successful customer journeys and their target touchpoint found in the historical data
- The fine-tuned SBERT model was used to build a vector database containing embeddings for all the text content found in the AWS Support knowledge base.
- A recommendation system was built that takes as an input the embedding of a customer journey and returns the URL corresponding to the content item that the fine-tuned model determines to be most semantically related

4.6.1 Picking a Pre-Trained Model

The authors of the SBERT paper made available a diverse set of SBERT general purpose models which are differentiated by their architecture size, computation speed, pre-training data and their corresponding accuracy. The first item on our checklist was to select a pre-trained SBERT model as the foundation for additional fine-tuning given our application—a task that was approached by evaluating the models on two key metrics: accuracy and computational efficiency.

Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80 MB
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80 MB
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970 MB
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43 MB
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420 MB
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61 MB
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480 MB
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480 MB

Figure 4-5: SBERT Pre-Trained Models [13]

The **Tall-MniLM-L6-v2** model stood out in the SBERT documentation for its balance of performance and precision. An important constraint we had to keep in mind is the 256 max sequence length of this pre-trained model. Let's first discuss what this parameter means and what the implications are for our project. When a large language model generates embeddings for a given text, it first needs to split the natural language text into tokens through a process called tokenization. Tokenization means splitting the input text into characters, words or subwords which are then converted to ids through a look-up table. Character tokenization simply splits the text into unique characters. While this approach leads to a small vocabulary comprised of each unique character in a text, it does not allow language models to learn the grammar or nuances of a language. On the other hand, splitting a text into words may lead to a very large vocabulary given a large text corpus. Even though splitting a text into words could enable models to build an in depth understanding of languages, such a big vocabulary causes both an increased memory and time complexity. To strike a balance between the pros and cons of character level and word level tokenization, transformers models use a hybrid between word-level and character-level tokenization called subword tokenization. Subword tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords[17]. There are three main types of subword tokenizers used by transformers: Byte-Pair Encoding(BPE), WordPiece and SentencePiece. Given that SBERT in an finetuned extension of the BERT model, they both use the WordPiece tokenization algorithm. The algorithm was published in the "Japanese and Korean Voice Search" paper[16].

While we do not need to break down the inner workings of the WordPiece algorithm, it is important to understand that the max sequence length of the SBERT model we picked means that we can process up to 256 subword tokens from a text. Given that each token is less than a word, that means that our context window for the model is less than 256 words. When analyzing the AWS Support knowledge base, we learned that our articles range in length, from a few words to a few thousands of words. Therefore, for articles longer than 256 words, we were guaranteed to have our text

truncated to the first 256 tokens, meaning that we would lose some of the knowledge from these articles, potentially impacting the performance of our recommendation systems further down the road. We decided to move forward with our pre-trained model choice, keeping in mind this constraint and making note that a language model with a larger context window could be used for further improving the performance of the recommendation system we were going to build. We touch upon this topic in Chapter 7 - Enhancements to Semantic Search Models.

4.6.2 Picking a loss function

The next critical decision in the development pipeline was the selection of a loss function for fine-tuning the open source SBERT model, which fundamentally shaped the semantic search capabilities of the system. To make this decision, two recurring ideas throughout this research were taken under consideration:

- The target touchpoint of a successful customer journeys is semantically similar with the aggregate content of the articles visited by a customer in their journey. Therefore, the embeddings corresponding to an article that is semantically close to the aggregate content of a customer journey have a higher probability to be a good candidate for the next best piece of a content that a customer should visit to accomplish their objective

- Successful historical journeys carry valuable information about the content that customers should visit sequentially to accomplish their objective without human support intervention. Therefore, the embedding of a target touchpoint corresponding to a successful journey should be closer in the vector space to the embedding of that customer journey than the embeddings of other pieces of content from the set of possible touchpoints

Based on these two assumptions and the learnings from the SBERT framework, we had two prominent contenders for a loss function: triplet loss and contrastive loss, each with its unique approach to embedding representation. **Contrastive loss** is a

popular machine learning technique used to optimize a model's ability to capture the similarity or dissimilarity of pairs of inputs by amplifying the contrast between positive and negative samples within a dataset. This function involves two components: an anchor and a positive or a negative sample. Positive samples are examples of items in the dataset that are similar to each other, while negative samples are examples that are dissimilar. Within the context of Siamese networks, contrastive loss is employed in embedding models like SBERT.

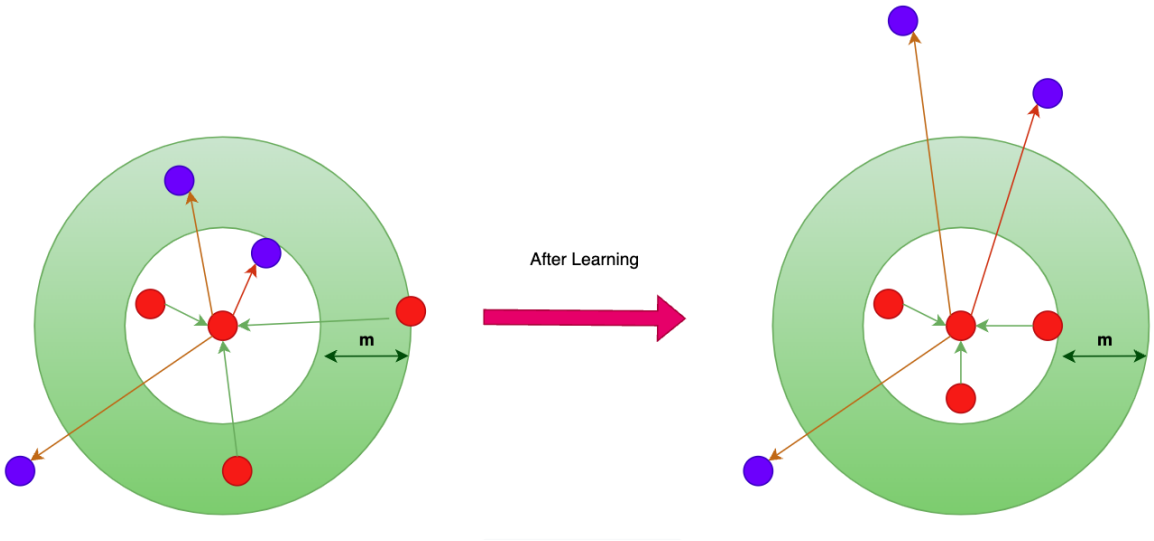

Figure 4-6: Contrastive Loss [22]

The contrastive loss function consists of two terms: an anchor and a positive or negative pair. In the case of positive pairs, the loss function pulls the embeddings of similar examples to be close to the embeddings of the anchor in the vector space. For negative pairs, the loss pushes embeddings of dissimilar examples to be far from the embedding of the anchor in the vector space. Following the core research assumptions described earlier, a positive pair would correspond to the embedding of a successful historical customer journey (anchor) and the embedding of the target touchpoint of that successful customer journey (positive sample). A negative pair would be the embedding a successful historical customer journey (anchor) and the embedding of a touchpoint that did not conclude that successful customer journey (negative sample). In the image above, the red dot at the center corresponds to embedding of a successful

historical customer journey (anchor), the surrounding red dots being pulled closer in the space correspond to the end touchpoints of these successful journeys (positive sample) and the purple dots correspond to touchpoints that were not found at the end of the successful customer journey (negative samples).

Triplet Loss is similar with Contrastive Loss in enhancing a model's ability to capture the similarity or dissimilarity of pairs of inputs. However, this function involves three components—an anchor, a positive example, and a negative example.

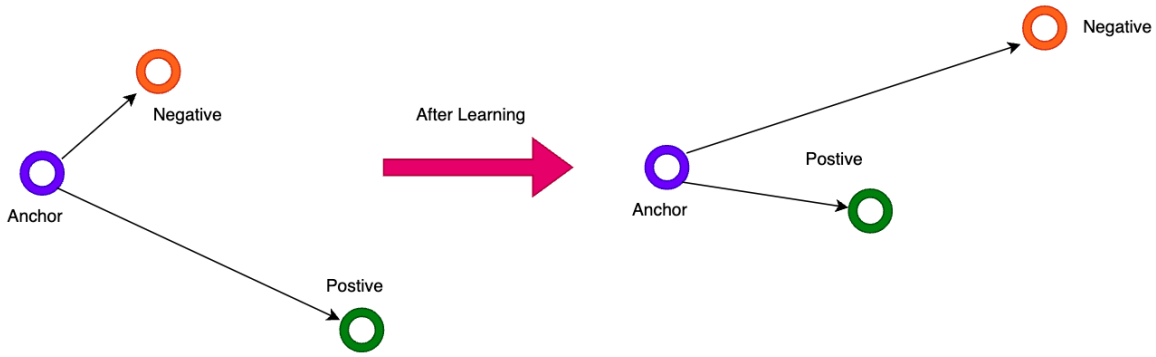

Figure 4-7: Triplet Loss [22]

The function is designed to minimize the distance between the anchor and the positive and maximize the distance between the anchor and the negative. This loss function is particularly suitable in scenarios where the distinction between relevant and irrelevant data points is essential for the model. For the context of this research, it is important to capture the higher probability of a target touchpoint to be a suitable best next content to visit for a customer if successful historical journeys ended with that touchpoint. However, the concept of a distinctive, negative sample that must be differentiated from the positive sample does not apply under these circumstances because AWS Support content does not have the concept of opposite content. For example, customer journeys exploring documentation for services such as developer tools may end frequently on the API description for an application like Amazon CodeWhisperer. But that does not necessarily mean that other content, which is not the Amazon CodeWhisperer API, for instance the CodeWhisperer integration with an IDE, could be considered dissimilar in relationship with the CodeWhisperer API. In other words, successful customer journeys may be paired with positive samples, such

as the target touchpoints that were found at the end of the journey, or with negative samples, such as touchpoints that were not found at the end of those journeys. But the relationship between those positive and negative samples ought not to be one of dissimilarity.

	Triplet loss	Contrastive loss
Use	We use triplet loss to learn different embeddings for positive, negative, and anchor data points.	We use contrastive loss to differentiate between similar and dissimilar data points.
Input	The input of the triplet loss consists of an anchor point, a positive point, and a negative point.	The input in contrastive loss requires a comparison point and an anchor point.
Calculations	It maximizes the distance between an anchor point and a negative point and minimizes the distance between the positive point and an anchor point.	It encourages similar points to close together while dissimilar points to be far apart.
Efficiency	It requires more time in training because of the large number of samples.	It is more efficient because it requires only similar and dissimilar points.
Applications	Its applications are in face recognition and in-person re-identification.	Its applications are in similarity learning, image retrieval, and in recommendations systems

Figure 4-8: Loss Function Characteristics [22]

Once the characteristics of the two loss function options were analyzed, the contrastive loss function was picked as the best option to move forward with. Not only did contrastive loss enable a model to make use of the knowledge carried by successful historical customer journeys, but it was also widely used in recommendation systems applications. These characteristics made it a suitable choice for the objective of this research.

4.6.3 Generating samples for contrastive loss

With contrastive loss chosen as the loss function used for fine tuning, we proceeded with building the actual data pairs to be used in the training process. The initial representation of this pair was a dictionary of touchpoints, mapping the second to last touchpoint from a customer journey chain of touchpoints to the last touchpoint in the

journey, which we refer to in this research as the target touchpoint. It is important to remember that the journeys had a variable size, but we paid more attention to the last two touchpoints because they were indicative of reaching a positive or negative outcome. At training time, the embeddings of the text corresponding to anchor touchpoints were mean pooled, to obtain a single vector embedding. In terms of the pairing, embedding corresponding to the positive or negative touchpoints is used. As a result, the positive or negative pairs are composed of two vector embeddings which can be pulled closer together or pushed further apart in the vector space by the contrastive loss function. First, the positive samples were generated from the historical dataset by creating pairs between the touchpoints of a successful customer journey without the last touchpoint as the anchor, and the last touchpoint as the positive sample. On the other hand, generating negative samples turned out to be more challenging as it involved a series of decisions to be made. In the historical customer journey dataset, there was a significant imbalance between the number of successful customer journeys and the number of unsuccessful customer journeys. As a reminder, the unsuccessful customer journeys correspond to customer engagements with AWS Support properties that resulted in a support case being opened. This number was smaller compared with the number of interactions customers have that do not result in a case being opened by several degrees of magnitude. As a result, the set of choices for negative samples was much smaller than the set of choices for positive samples. The generation of negative examples posed a challenge due to an imbalance in our dataset—successful journeys far outnumbered the unsuccessful ones. To work around this dataset characteristic, we first built a dataset of negative samples with the set of unsuccessful target touchpoints we had available in the historical customer journey dataset. Then, for the remainder of samples that we needed to balance the number of positive cases for, we assigned a target touchpoint at random from the AWS Support knowledge base, making sure that the chosen touchpoint was not a successful one in the existing historical dataset. For example, let's say we had 100 positive samples based on the historical dataset and we wanted to generate 100 negative samples. We first found 10 negative target touchpoints in the historical customer journey dataset. Then, for the remainder of

90, we took successful customer journeys, removed the target touchpoint and then assigned as the target a touchpoint at random from the universe of possible options, checking that it was different from its initial successful target.

4.6.4 Fine Tuning

As discussed in the prior steps, the **Tall-MiniLM-L12-v2** pre-trained SBERT model was used as the base model to be fine-tuned. The sequence length to was set to 256, which was the maximum sequence length for the pre-trained model of choice. The embeddings for the contrast loss touchpoint pairs were constructed in a 384-dimensional space, allowing for rich semantic representations. The embeddings were normalized, facilitating similarity computations using cosine-similarity. Mean pooling was employed to aggregate token embeddings into a fixed-size representation. We conducted a single epoch of training. Training was performed with a batch size of 16. The warm-up number of steps was set to a dynamic 10% of the total training steps to gradually adjust the learning rate, enhancing stability during optimization. We defined the training objective using a Contrastive Loss function. Throughout the training process, we continuously evaluated the model's performance using an evaluator designed for Information Retrieval tasks, ensuring that the model fine-tuning process aligned with the desired outcomes. Please note, the embeddings used at the fine-tuning step were generated with the open-source pre-trained SBERT model of choice. After the fine-tuning process, the embeddings generated by the fine-tuned model were different based on the weight adjustments determined by the contrastive loss function, tailored for our specific objective.

4.6.5 Building a Vector Database

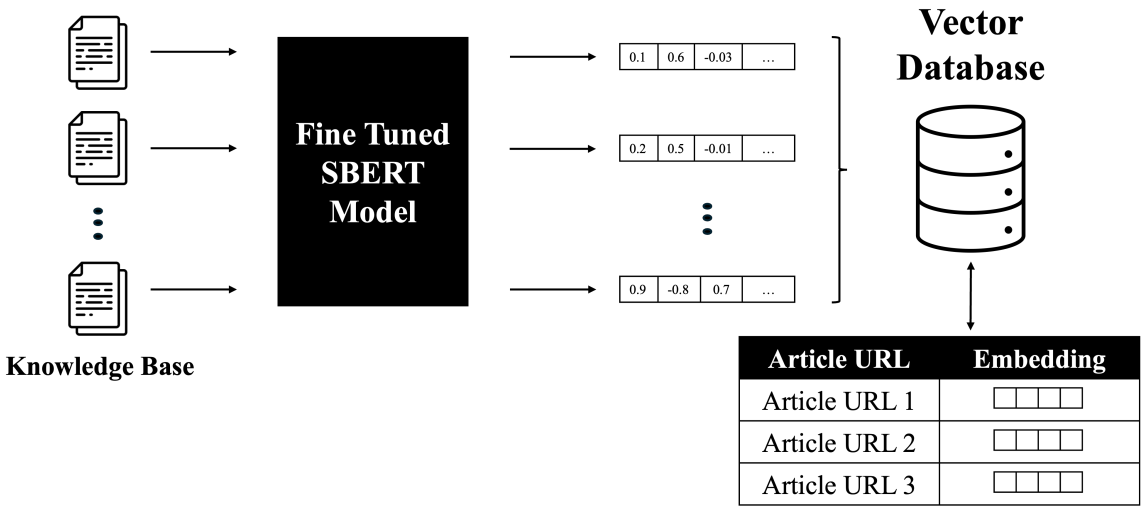

Figure 4-9: Vector Database Architecture

With a fine-tuned SBERT model ready to use, the next stage of the system development was focused on creating a vector database containing embeddings for all the text content articles in the AWS Support knowledge base. For a recommendation system to be able to identify the next best piece of content for a customer to visit, two items need to be compared: the embedding representing a customer journey up to the point of the recommendation and every single possible option that could be recommended. With a knowledge base of more than tens of thousands of articles in AWS support, generating embeddings for all these items at runtime would be computationally inefficient and the time required to execute such a task would make it unfeasible to have recommendation readily available for customers as soon as they land on a new page. The solution for this computational challenge was to store the embeddings for each individual article and have them ready in production. Therefore, the SBERT fine-tuned model was used to generate these embeddings which were then stored in a JSON dictionary which had the page URL as a key and the vector embedding corresponding to the text found at that URL as the value. The creation of these embeddings is a one-time operation, intended to be a preparatory step before the system goes live. In real-time operation, the system would generate an embedding for the current customer journey, which is

dynamic, and it depends on the touchpoints recently visited by a customer, using the same fine-tuned SBERT model. It is still important to consider the dynamic aspect of the AWS Support knowledge base: new articles are published frequently and updates to already existing articles are also made for content enhancement. To account for this, the solution would be to schedule a refresh the vector database, just as a one-time daily operation, having the stored embeddings readily available throughout the day for the live system. Once these embeddings were in place, the foundation of the recommendation engine was ready.

4.6.6 Recommendation System Design

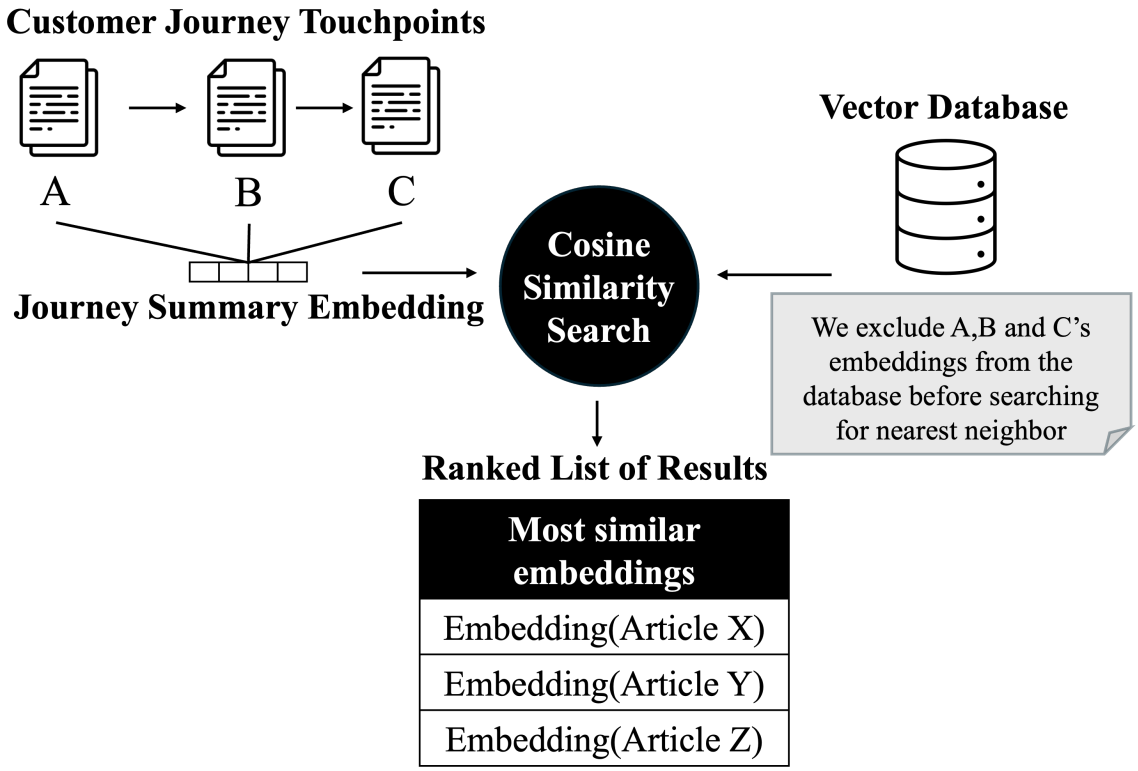

Figure 4-10: Semantic Search Architecture

This stage integrated the fine-tuned SBERT model with the customer journey vector database. The desired final state of the system needs to take the sequence of touchpoints visited by a customer and return a set of recommendations for the next best content to visit that would answer the customer need. These recommendations need to be

delivered in the form of a sorted list of article titles, together with their associated hyperlink. We leveraged the historical journey dataset to simulate this exercise. First, we built a function that filtered down the potential recommendation pool to remove all the touchpoints already visited in a journey. Our assumption is that if a customer has already visited a content page and continued their journey, it means that they have not found the information they were looking for on that page. Therefore, our system should never send a customer to a touchpoint that was visited within the same journey. Recollecting our data pre-processing steps, journeys leading to a target touchpoint that was previously encountered were also systematically removed. Next, we enabled our system to accept a parameter "nTouchpoints" denoting the number of historical touchpoints to be considered when making a recommendation. As a reminder, customer journeys have a variable length: some may be only 2 touchpoints while others may well be over 10 touchpoints. Our initial hypothesis was that each of the touchpoints already visited carry knowledge about the customer needs and intent. However, we wanted to test this hypothesis by analyzing various ranges of touchpoints and determine if analyzing more of the journey would lead to better recommendations. The nTouchpoints parameter was adjustable, allowing us to experiment with the depth of historical interaction data that informs our recommendation. The next step involved the transformation of a collection of embeddings corresponding to the historical nTouchpoints of the journey to a single embedding, which we will refer to as a **summary embedding**, that could be compared with the potential recommendation pool found in the filtered vector database. For example, picture a journey of 10 touchpoints. If we wanted to consider only the 3 most recent pages visited by the customer when making a recommendation, then nTouchpoints = 3 would be passed to our system. As a result, the touchpoint to be aggregated into the summary embedding would be touchpoints with indices 8, 9 and 10. The customer journey touchpoints are sorted by time of visit, so touchpoints with indices 8, 9 and 10 are the 3 most recently visited pages for this example journey. However, if the journey only had 4 touchpoints and nTouchpoints = 5 parameter was passed, then the maximum number of available historical steps would be used, which in this case would be 4. For the aggregation of

these nTouchpoints, we opted for a mean pooling strategy to combine the embeddings of these touchpoints into a singular, coherent vector representation.

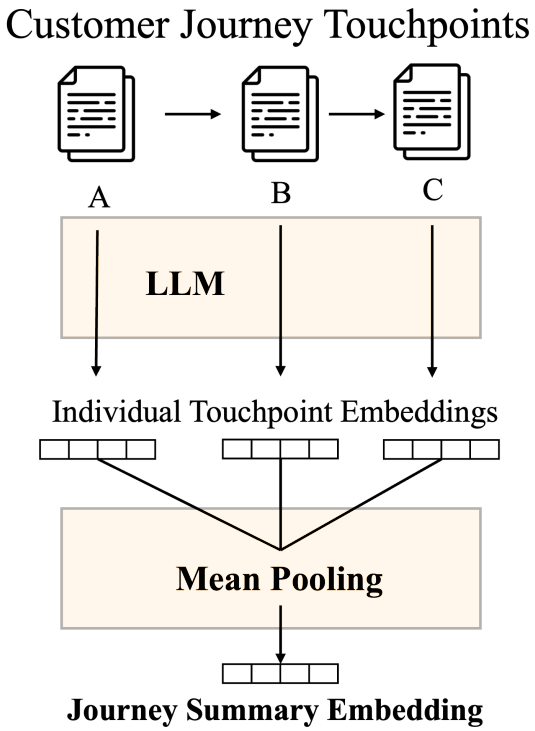

Figure 4-11: Building a Customer Journey Summary Embedding

This strategy was chosen as the most effective through our experiments, aligning with the findings presented in the Sentence-BERT paper. With this aggregated journey embedding, we built used our fine-tuned SBERT model to perform cosine similarity comparisons against the entire corpus of content present in the filtered vector database. This comparison resulted in a score for each pair—comprising the journey embedding and a candidate piece of content—that quantitatively represented their semantic alignment. These scores were then sorted, and the top k embeddings were selected. Using the URL : embedding mapping we had built in a prior step, we obtained a list of the top k most relevant articles for a customer to visit. To summarize the process described above, our system was designed to first filter down the set of potential content recommendations to ensure that already visited pages are not shown again. Then the system aggregates the vector embeddings of the n most recent touchpoints

(content visited by the customer) into a summary embedding, where n is a parameter that I, as the developer, could adjust to optimize performance. Then this customer journey summary embedding was used to identify the n most relevant pieces present in the AWS Support knowledge base for their next step. With this output, our semantic search recommendation architecture was complete are ready for testing.

4.7 Model Evaluation - Semantic Search

In the evaluation phase of the recommender system, the priority was to measure the effectiveness of the recommendations from a qualitative and a quantitative perspective. To do so, we once again centered our focus on our primary objective, which was to deliver recommendations that are likely to resolve the customer's queries, thereby potentially concluding their journey without the need for human assistance. This measure of success aligned with our goals of enhancing customer satisfaction and achieving operational efficiency on a path towards scalable support. Additionally, we wanted to test the hypothesis that the touchpoints already visited in a customer journey carry knowledge about the customer needs and intent. To measure the performance of the semantic search recommendation system, it was essential to understand two key outcomes driven by this system:

- Do the customers visiting the AWS Support knowledge base interact with the recommendations displayed to them along their journey
- When the customers interact with the recommendations, how often do they conclude their journey thanks to our system's guidance

For customers to benefit from improved recommendations and for AWS Support to measure if such recommendations conclude journeys with a high success rate, customers first need to engage and interact with these recommendations. Such customer engagement could only be evaluated in a live production environment. However, the development efforts and resources required to take the system to production were worth pursuing only after the model had showed promising results in a simulated

environment, as a proof of concept. Our approach included an online evaluation method proposal that could measure customer engagements with the recommendations and an offline evaluation method for calculating a set of metrics capturing the effectiveness of our recommendations in concluding customer journeys without the need for further human support. The online evaluation proposal was presented as a future step to be implemented before the recommendation system is taken to production while the offline evaluation was fully implemented. We thoroughly present both evaluation methods in the upcoming sub-chapters.

4.7.1 Online Evaluation through A/B Testing:

We proposed an A/B testing methodology to be implemented within the Support Center framework, one of the AWS Support properties. The primary objective of this test proposal was to compare the effectiveness of the newly developed recommender system against the existing content recommendation mechanisms. In this setup, the semantic search system's recommendations would be displayed on the page where customers have the option to open a ticket. This choice was made because a customer who reaches the page where they can open a support ticket signals intent to request human assistance because of not finding a resource in their customer journey which could enable them to self-service. If a customer picks a recommendation made by our system on this page and does not open a ticket, that is a strong indicator that our system's recommendation was appropriate and helped the customer solve their problem. The test proposal involved randomly assigning users who land on the support page to two different groups: one group would receive recommendations from the newly developed system, while the control group would receive the most frequently visited articles across the Support knowledge base, as we described earlier. Key performance indicators (KPIs) for this experiment would include metrics such as the rate of ticket deflection, customer engagement with the recommended content (measured through click-through rates and time spent on content). These metrics would help us understand not only the system's ability to accurately predict customer needs but also its effectiveness in enhancing the user experience and potentially

reducing the volume of support tickets. While the A/B test proposal was feasible, the infrastructure required to parse customer journey data in a live production environment required resource allocations which could only be justified if the recommendation system demonstrated strong performance in an offline setting. This requirement allowed us to explore alternative methods to assess the potential of our recommender system in a more cost-efficient manner.

4.7.2 Offline Evaluation with Historical Data:

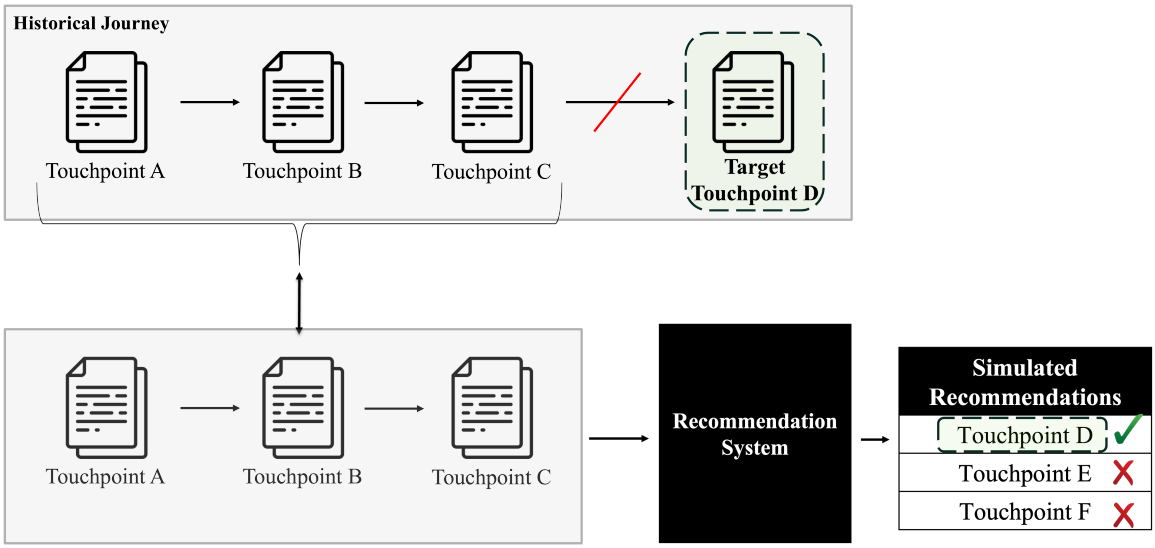

Figure 4-12: Back-testing Approach

Equipped with multiple years' worth of historical customer journey data, simulating the performance of the newly developed recommendation system was the next logical step. The data could already tell us which customer journeys were successful. As previously mentioned, it was assumed that the target touchpoint of a successful customer journey contained the knowledge, insight or answer a customer was looking for along their journey. Then, it was decided that the best way to evaluate performance was to check if we the system would recommend the target touchpoint N of a successful journey by taking the N-1 touchpoints journey as an input. If the system could make that recommendation, then the wisdom of the crowd captured by the historical dataset would confirm that we pointed the customer to the next best step. With this approach

decided on, the missing piece of our experiment was the set of metrics to calculate during the simulation. We opted for an academically recognized set of metrics used to evaluate recommendation systems in an offline environment:

- **Hit Rate**: Measures the frequency with which recommended items match the target items in our test set. Assume that we had a successful journey in the test dataset A – B – C. Then, our system would take the journey A-B as an input and make a set of ranked recommendations. If the list of recommendations made by our system contains the article C, that instance would be counted as a hit. The hit rate is the proportion of hits out of the number of test journeys we ran through the model

- **MRR (Mean Reciprocal Rank)**: Assesses the rank of the first correct recommendation. The MRR is calculated at an individual test level. Assume that we had a successful journey in the test dataset A – B – C. Then, our system would take the journey A-B as an input and make a set of ranked recommendations. The MRR of this test would be 1/rank(C). If our list of ranked recommendations has C at the top, then MRR = 1. If our list has C as the third recommended item, then MRR = 1/3. If C is not recommended, then MRR = 0. To calculate this metric for the entire test dataset comprised of multiple journeys, we average the individual test MRR's to obtain the average MRR.

- **MAP (Mean Average Precision)**: Evaluates the precision of recommendation lists. The precision of a recommendation is the ratio of relevant recommendations returned in the list out of the length of the list. Assume that we had a successful journey in the test dataset A – B – C. At the same time, we also had a successful journey A – B – D. Therefore, both C and D would be relevant recommendations. If the recommendation list has a length of 2, and those recommendations are F and D, then the precision is $\frac{1}{2}$. To calculate this metric for the entire test dataset comprised of multiple journeys, we average the individual test AP's to obtain the MAP.

- **NDCG (Normalized Discounted Cumulative Gain)**: Accounts for the quality of recommendations based on their position in the recommendation list. This metric takes into account the number of recommendations that match a target as well as their position in the ranking.

Having established the set of relevant offline metrics to capture for the semantic search recommendation system, we proceeded with implementing a script that could perform simulations on the historical dataset and calculate each of these metrics at scale. First, before even fine-tuning our model, we split the historical customer journey dataset into three sets: training set, evaluation set and test set. We used a ratio of 80 – 10 – 10, respectively for splitting the golden dataset. We developed a function to evaluate the test set of journeys against their actual outcomes, allowing us to calculate these metrics for each journey. By saving the metrics for each individual historical journey in data structures such as lists, we were able to aggregate them to understand the average performance across our test dataset. This aggregate view was crucial for assessing the overall effectiveness of our recommendation system and it allowed us to re-purpose the infrastructure for measuring the performance of alternative recommendation system implementations which we will discuss in later chapters.

4.7.3 Target of a journey

While experimenting with the recommender system, we discovered that a natural outcome of using a fine-tuned semantic search model to find the best content for a customer to read was that most of the items in the recommendation list were very relevant given the prior touchpoints visited by customers. However, our assumption up to this stage of the research was that only one of those relevant recommendations could be considered a target touchpoint for a customer journey because a journey such as A – B – C – D has only one terminal node, D. Furthermore, when choosing the offline evaluation methodology for our recommender system, we learned that the offline metrics used for such systems allow for multiple correct recommendations. Noticing in historical customer journey data that we could find instances of successful journeys

such as A – B – C – D as well as A – B – C - Q, we re-evaluated the concept of a target touchpoint. The goal of a recommendation is to offer customers content that would answer their needs and, as a positive outcome, help them end their journey as soon as possible, offering a positive experience. Therefore, we concluded that more than one article could provide a customer with the content they need based on their objective and intent. This conclusion led us to create lists of acceptable target touchpoints for given customer journeys, based on the historical data. As discussed in the example before, noticing that the journey A – B – C - ? turned out to be successful for both target touchpoints D and Q, then we generated the list of acceptable target touchpoints [D,Q] for that particular customer journey. At the same time, we wanted to evaluate the performance of our model for both the more restrictive definition of a target touchpoint that only allows for a single correct recommendation as well as the more flexible definition that accepts a list of correct recommendations. We divided our evaluation work based on these two concepts, concisely defined below:

- **Single Target**: When considering a single target for the recommendation system, the focus is on precision. The goal is to predict the exact next touchpoint in the customer journey that we have a high confidence that it could conclude the journey with a positive outcome. For a given sequence like A-B-C-D, the target is D – the actual final touchpoint of a journey part of the test dataset. This approach emphasizes the accuracy of the recommendation at the final step of a customer's journey.

- **Multiple Targets**: In the case of multiple acceptable targets, the system allows for a broader definition of success. For example, if AB is a common sequence in the training dataset[1] leading to both D, H and K in different successful journeys, then all three are considered acceptable targets. This method recognizes the complexity and variability of customer journeys, allowing for multiple ways of ending a customer search succesfully.

[1]In the figure below we only considered the one touchpoint sequence B followed by various different touchpoints.

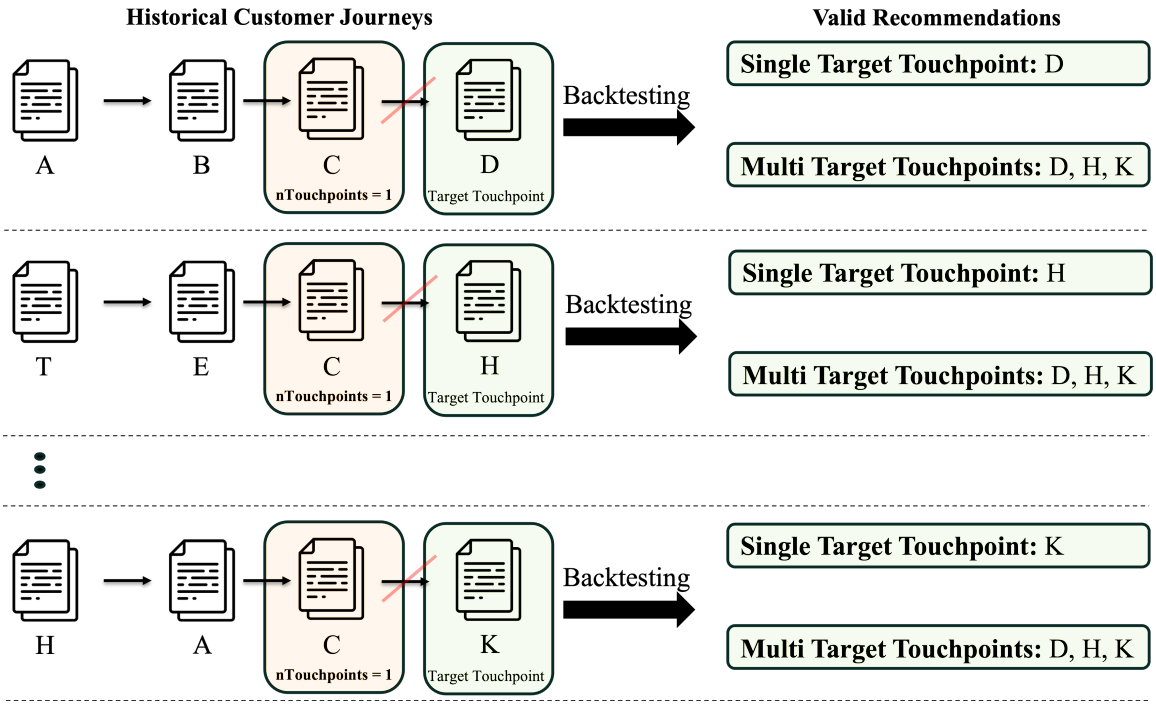

Figure 4-13: Single and Multiple Target Touchpoints

Please note, in scenarios where multiple targets are accepted, it is anticipated that there will be a perceived increase in the performance metrics of the offline recommender system. However, this evaluation method is still valid and has merit because the set of acceptable target journeys is derived from historical patterns of successful customer journeys. It is reasonable to propose that multiple content options could be relevant to the customer's search, reflecting a more holistic understanding of varied customer needs.

4.8 Semantic Search Recommendation Experiments

With both a robust recommendation system and evaluation framework in place, we were ready to conduct a series of experiments. Our experimental design included a variety of tests aimed at understanding the system's performance under different conditions. We designed the experiment infrastructure in Python. To simulate production runtime customer journeys, we leveraged the historical successful customer journey dataset. We remove the last touchpoint of these journeys because that is the

target our system is aiming to recommend. Furthermore, we implemented the ability to pass the following parameters:

- The number of touchpoints from a journey to be aggregated, nTouchpoints, when building the customer journey summary embedding. For example, if a customer journey has a length of L = 5 touchpoints, if we pass the parameter nTouchpoints = 3, then the summary embedding is built by taking the [L – nTouchpoints +1, L] touchpoints from the journey, map them to their corresponding embeddings from the vector database described in Chapter 4.6.5 Building a Vector Database and then mean pool them to obtain a single vector embedding. As it may be noticed, we never include the last touchpoint, or target touchpoint as we have been referring to it in this research, because that is the embedding that our recommendation model is aiming to return

- The number of recommendations our system returns, which we named nRecommendations. As described in Chapter Recommendation System Design, our recommendation system takes the customer journey summary embedding and performs semantic search against all the content embeddings in the AWS Support knowledge base, which we stored in the vector database described in chapter 3.5.5. This process generates a sorted list of recommendations base on the similarity score between the journey summary embedding and the embeddings in the vector database. With the nRecommendations parameter, we tell the function to only return the URL's corresponding to the top nRecommendations items in this sorted list

Please see the simulation results in the tables below.

Simulation	MRR	MAP	NDCG	Hit Rate
1touchpoints_@1	0.077	0.077	0.077	0.077
2touchpoints_@1	0.071	0.071	0.071	0.071
3touchpoints_@1	0.067	0.067	0.067	0.067
4touchpoints_@1	0.066	0.066	0.066	0.066
5touchpoints_@1	0.066	0.066	0.066	0.066
1touchpoints_@2	0.099	0.099	0.105	0.121
2touchpoints_@2	0.092	0.092	0.098	0.113
3touchpoints_@2	0.089	0.089	0.094	0.110
4touchpoints_@2	0.086	0.086	0.091	0.106
5touchpoints_@2	0.086	0.086	0.091	0.105
1touchpoints_@3	0.110	0.110	0.121	0.153
2touchpoints_@3	0.103	0.103	0.113	0.145
3touchpoints_@3	0.100	0.100	0.110	0.142
4touchpoints_@3	0.097	0.097	0.107	0.138
5touchpoints_@3	0.096	0.096	0.107	0.137
1touchpoints_@5	0.119	0.119	0.137	0.193
2touchpoints_@5	0.112	0.112	0.131	0.186
3touchpoints_@5	0.108	0.108	0.126	0.180
4touchpoints_@5	0.106	0.106	0.123	0.176
5touchpoints_@5	0.105	0.105	0.122	0.175
1touchpoints_@10	0.128	0.128	0.160	0.262
2touchpoints_@10	0.122	0.122	0.155	0.260
3touchpoints_@10	0.118	0.118	0.149	0.252
4touchpoints_@10	0.115	0.115	0.145	0.245
5touchpoints_@10	0.114	0.114	0.144	0.243

Simulation	MRR	MAP	NDCG	Hit Rate
1touchpoints_@1	0.244	0.062	0.086	0.244
2touchpoints_@1	0.201	0.053	0.072	0.201
3touchpoints_@1	0.189	0.051	0.070	0.189
4touchpoints_@1	0.183	0.050	0.067	0.183
5touchpoints_@1	0.182	0.049	0.067	0.182
1touchpoints_@2	0.299	0.088	0.126	0.354
2touchpoints_@2	0.250	0.073	0.104	0.300
3touchpoints_@2	0.233	0.070	0.099	0.277
4touchpoints_@2	0.226	0.069	0.097	0.268
5touchpoints_@2	0.224	0.068	0.095	0.265
1touchpoints_@3	0.322	0.101	0.148	0.422
2touchpoints_@3	0.273	0.085	0.125	0.367
3touchpoints_@3	0.253	0.081	0.118	0.337
4touchpoints_@3	0.245	0.079	0.114	0.327
5touchpoints_@3	0.243	0.078	0.113	0.324
1touchpoints_@5	0.338	0.115	0.173	0.491
2touchpoints_@5	0.289	0.096	0.147	0.438
3touchpoints_@5	0.268	0.092	0.138	0.406
4touchpoints_@5	0.261	0.090	0.135	0.396
5touchpoints_@5	0.258	0.088	0.133	0.390
1touchpoints_@10	0.351	0.130	0.207	0.590
2touchpoints_@10	0.303	0.110	0.179	0.543
3touchpoints_@10	0.283	0.104	0.168	0.513
4touchpoints_@10	0.275	0.102	0.163	0.497
5touchpoints_@10	0.272	0.100	0.161	0.492

Figure 4-14: Semantic Search Results

The entries in the table should be interpreted the following way:

- **1touchpoints@5**:
  - **1 touchpoints** means that we only looked at 1 historical step made by the customer. In other words, we only used the most recent article visited by the customer to inform our recommendation.
  - **@5** means that we made 5 recommendations which could include target touchpoints

- **3touchpoints@1**:
  - **3 touchpoints** means that we only looked at 3 historical steps made by the customer. In other words, we only used at most the 3 most recent articles visited by the customer to inform our recommendation. We say at most because if the historical journey used in the simulation only had 2 touchpoints, then the min(nTouchpoints, journey length L) is used, which in this case would be 2.

- **@1** means that we made 1 recommendation which could be a target touchpoint

The key difference between the tables is the way we applied the target touchpoint concept. In the first table, we are displaying the result obtained with a single target touchpoint, where the journey A – B – C – D - ? extracted from the historical dataset has a single possible correct recommendation, the touchpoint that followed D in that journey instance. In the second table, we allow a set of target touchpoints, based on the touchpoints that followed D in the universe of successful historical journeys where D was the second to last touchpoint.

4.9 Semantic Search Key Takeaways

Two patterns are clearly highlighted by the results of our experiments. First, the more recommendations are made, the higher the accuracy of the system becomes. This pattern was expected, and it is intuitive, given that the more recommendations we make, the higher the probability that one of them matches the target touchpoint of the historical journey. Even if the recommendations were made using a function that picked target touchpoints at random from the knowledge base, we would still expect to see a positive correlation between the number of recommendations made (@x) and the accuracy of the system. Secondly, we learned that the more historical touchpoints are considered, the lower the accuracy of the semantic search recommendation system becomes. This result goes against our initial assumption that all the past interaction of a customer are semantically relevant to their intent and need. In other words, the farther back we analyze the resources explored by a customer, the less we can predict what the customer is looking for by leveraging a semantic search approach. Analyzing the results of our experiments, we put together a series of key takeaways:

1. **Key Takeaway 1: Historical Data's Diminishing Returns**
   Our analysis revealed a counterintuitive trend: incorporating a more extensive history of customer interactions into our recommender system led to diminishing

accuracy. This discovery contradicts our initial hypothesis, which stated that a richer historical context would enhance recommendation precision. We infer that historical information, beyond a certain threshold, becomes dilutive rather than accretive in decision-making. This suggests an optimal balance in the quantity of historical data utilized for accurate recommendations.

2. **Key Takeaway 2: Customer Journey Dynamics**

   During the data analysis stage, we learned that a significant proportion of the journeys were short in length, 1-3 touchpoints. When we sampled some of the longer journeys that we back tested our recommendation system against and read the content of the touchpoints, we observed that customer journeys are characterized by rapid shifts in search topics, refuting the notion of a consistent thematic trajectory which our hypothesis relied on. This insight aligns with the prevalence of short-length journeys in our data set, which is indicative of customer shifting their attention to a different topic after a few interactions. This indicates a dynamic, rather than static, customer interest pattern, requiring a flexible and responsive recommender system capable of adapting to these rapid changes.

3. **Key Takeaway 3: Efficacy of Fine-Tuned Semantic Search**

   Despite the dilutive nature of historical touchpoints, our system's fine-tuned semantic search demonstrated robust performance. It efficiently identified recommendations that were not only relevant but also exhibited a high potential to conclusively address the customer's query. This underscores the value of nuanced semantic analysis in recommendation systems, even in the context of variable customer journey patterns.

4. **Key Takeaway 4: System Performance in the Context of Coherent Journeys**

   Upon closer examination of instances where our recommender system excelled, a common factor emerged: these were instances of coherent customer journeys. This pattern indicates that our system is particularly effective when dealing

with customer interactions that follow a consistent, thematic trajectory. In such coherent journeys, the historical touchpoints contribute significantly to the accuracy of our recommendations. This finding suggests a nuanced approach to utilizing historical data: it is highly valuable in cases where customer interests and queries demonstrate thematic consistency. Thus, enhancing our system to identify and cater to these coherent journeys could unlock its full potential. This approach would leverage the strengths of our dataset, transforming it into a more powerful and nuanced tool for personalized recommendation strategies.

4.10 Coherence Study

Considering the counter-intuitive outcomes from our experimental results which showed that historical information becomes dilutive rather than accretive in predicting customer intents, we decided to undergo a coherence study to further understand the intricacies of customer journeys. This investigation aimed to shed light over the nature of the relationship between earlier touchpoints in a customer's journey and the concluding target touchpoint. Specifically, we were interested in determining whether customer learning is a gradual process, where each step brings them incrementally closer to their desired outcome, or if it resembles a leap, with seemingly disparate touchpoints followed by a sudden convergence to a closely related final step. This exploration was pivotal in discerning the role and impact of cosine similarity—the foundation of our recommender system— within the customer journey. To conduct this analysis, we measured the average cosine similarity between past touchpoints and the target touchpoint across various journey lengths. We focused on journeys comprising a minimum of five touchpoints to assess the progressive similarity from the fifth step to the immediate predecessor of the target touchpoint. By systematically evaluating the similarity between each step and the target, and aggregating these data points, we wanted to obtain a nuanced understanding of the learning trajectory. This allowed us to visualize an 'average journey', where each step's distance from the target is correlated with its cosine similarity, providing insights into the typical

learning curve exhibited by customers. This study revealed fundamental patterns in customer behavior and content interaction, informing future work for the refinement of our recommender system's design to better align with the learning characteristics of AWS Support customers. Below, we may see a graphic visualization of the cosine similarity between each step in a five-touchpoint customer journey and the target.

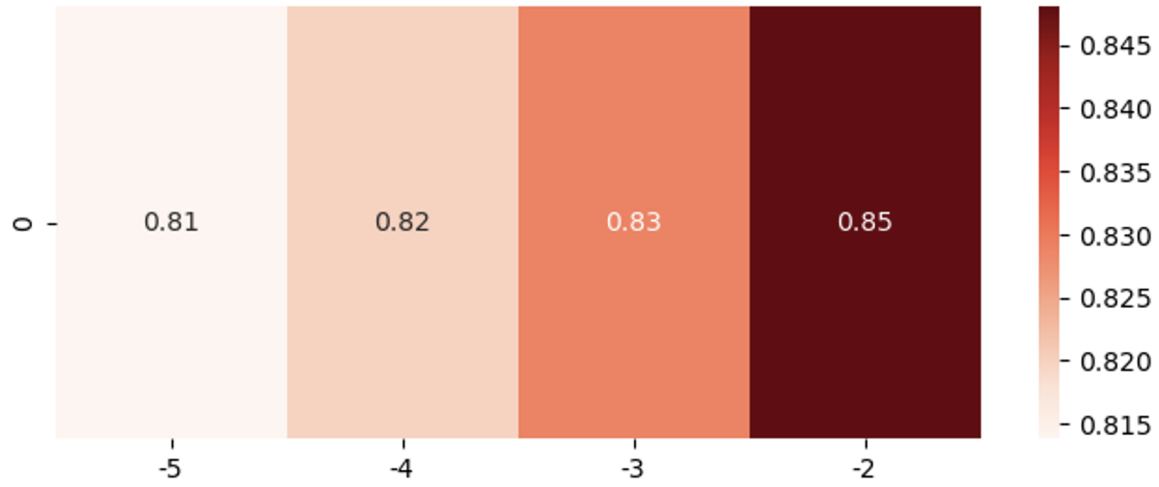

Figure 4-15: Coherence Study Results

The score displayed at index -5 is the average cosine similarity between the touchpoints visited by a customer five steps before the target touchpoint and the target touchpoints. To calculate it, we extracted from the historical customer journey dataset all the journeys of length 5 or more and we created a vector with the similarity score between each of the touchpoints in the journey and the target touchpoint. After this iterative process, we averaged all these vectors and generated the heatmap displayed above. Our coherence study results confirmed our experimentally driven insight that the farther we analyze customer interactions from the target touchpoint, the less semantically similar those touchpoints will be. To translate this customer behavior to a hypothetical scenario, a customer may have started a journey by searching information about the EC2 service, learned what they needed and then shifted their attention to a different problem they were looking to solve. The next item they visited was the documentation of a Large Language Model such as Claude, and they continued their search for another 4 touchpoints, getting closer and closer to their desired answer.

While the customer had two different objectives, because of our customer journey dataset construction methodology, these two different interactions are captured under the same customer journey id.

Chapter 5

Second Design: Frequency N-gram

5.1 Overview

While our semantic search-based system was designed to excel in scenarios where the next step in a customer's journey was contextually linked to their previous interactions, we learned during the evaluation of our first model and during the coherence study that there are instances where the logical next step may not be semantically related to the preceding journey. To address such cases, we investigated an alternative approach that could detect patterns in customer behavior without relying solely on semantic search. Additionally, we wanted to benchmark our semantic search system against a model with lower complexity and higher explainability so we could evaluate whether the increased effort resulted in proportionally better outcomes. This led us to the implementation of a n-gram model.

5.2 What is a N-gram?

The n-gram is the simplest type of language model. An n-gram is a sequence of n words: a 2-gram, also called a bigram, is a two-word sequence of words such as "target touchpoint", "semantic search", or "Masters Thesis", and a 3-gram, also called a trigram, is a three-word sequence of words like "evaluate the model", or "calculate cosine distance". At the same time, the 'n-gram' also means a probabilistic model

that can estimate the probability of a word given the n-1 previous words, and thereby also to assign probabilities to entire sequences. An n-gram model, in the context of our project, refers to a predictive model that utilizes subsequences of n-1 articles (touchpoints) from customer journey historical data to anticipate the next item in the sequence. The N-Gram essentially captures and analyzes the frequencies and patterns of these n-sized customer journey touchpoints, which can be indicative of common customer pathways, even when those pathways do not share obvious semantic links that we could track with the first model we developed. Let's say that for example we chose k to be 2, it means that we just look at the second to last touchpoints and capture the frequency of end of journey touchpoints that follow them. For example, if we had in the historical dataset the successful journeys A – [B – C] , P – [B – A] , L – [B – W], and E – [B – C], then we would build a frequency dictionary for touchpoint B, keeping track of the touchpoints that follow it and their frequency: B: C:2, A: 1, W:1. This dictionary captures the probability distribution of touchpoints following B. The n-Gram model is particularly suited to scenarios where the customer's next action might be informed by habitual behavior or sequential patterns, rather than semantic continuity. This model thrives on identifying the most frequent and therefore likely transitions between touchpoints, regardless of their content.

5.3 Building the N-Gram model

5.3.1 The N-Gram Frequency Dictionary

We created a pre-processing Python function that took the historical customer journey dataset as an input, as well as a parameter N, denoting the length of the N-Gram sub-sequence. This function returned the N-Gram probability distribution dictionary we described earlier, for every single sub-sequence of length N that precedes a target touchpoint in the historical journey dataset. One important modelling decision we made was to only consider the right edge of the journey, constructing frequency vectors only for the N-1 sub-sequences preceding the target touchpoint, instead of a rolling

window. The reasoning was that we only wanted to capture those patterns that had a high correlation with the end of a successful journey, and we wanted to avoid the risk of capturing connections between customer sub-journeys represented under the same unique ID. To illustrate this decision, let's look at the hypothetical successful customer journey A-B-C-D-E. Assume N=3, so we are building frequency dictionaries of touchpoint which follow sub-sequences of 2 touchpoints. For our project, this sample journey only yields 1 N-Gram, the one preceding the target touchpoint: CD: E:1. If we were to choose a rolling window implementation, we would also have a frequency dictionary for AB and BC. However, we learned during the semantic search model analysis that we capture multiple customer explorations within AWS Support under the same customer journey ID, so we felt more confident about associating the CDE sequence with a positive customer experience driven by a successful journey that the ABC or BCD sub-sequences.

5.3.2 Recommendation System Design

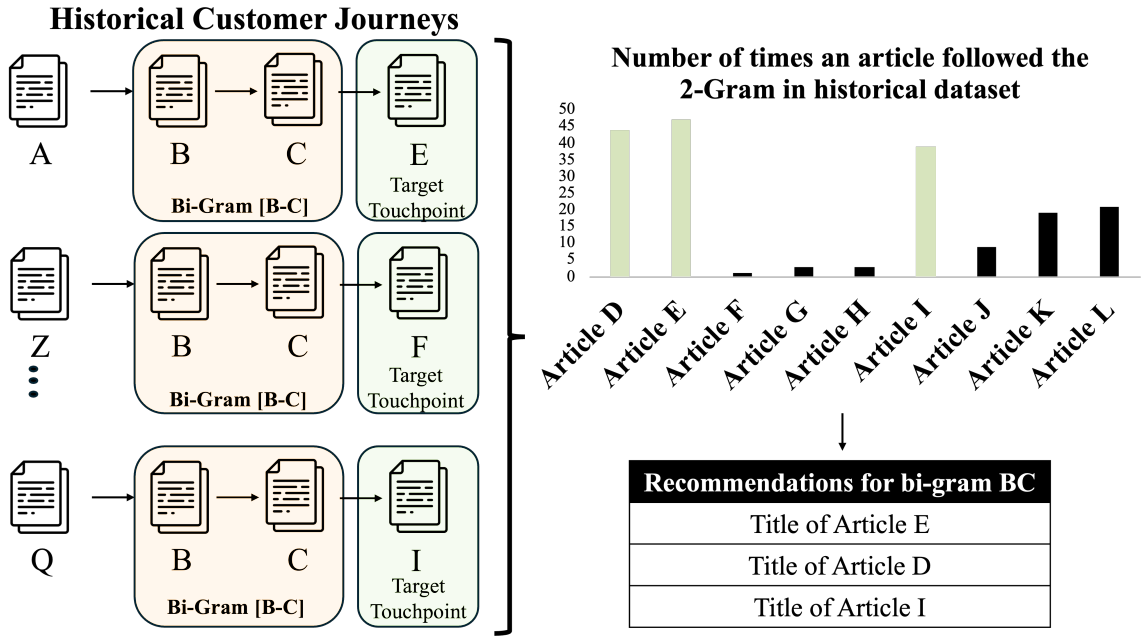

Figure 5-1: N-gram Architecture Design

With our N-Gram frequency data structure in place, we translated the patterns we captured into recommendations. Just like in the case of our semantic search model recommendation system, we built an infrastructure that allowed us to run multiple experiments with two main parameters: nTouchpoints and nRecommendations. In the semantic search model, the nTouchpoints parameter denoted the number of touchpoints prior to the target one to be aggregated via mean pooling to obtain the summary embedding. In this case, the nTouchpoints tells us the size of the n-Gram, so if nTouchpoints is 1, then we are in a bigram scenario (2-Gram). For efficiency purposes, we built the n-Gram frequency dictionaries in advance, leveraging the historical customer journey dataset. If this system were to be brought into production, we would still recommend building these data structures before recommendations have to be made live, for speed and efficiency purposes. Given that the frequencies captured in the n-Grams could change day after day, we would recommend an update of the data structures every morning, by parsing the prior day's journeys. The system recommends the next touchpoint based on the highest probability transitions observed in the historical data. If the n-gram is not present in the data structure, we make a random recommendation from the corpus. For example, if we receive the test journey A- B – Q – M and we want to make a recommendation using a 3 – Gram, then we look up at the frequency dictionary built for the QM subsequence and we sort the next touchpoints in a decreasing way, by frequency. Then, to make k recommendations, we pick the top k touchpoints from the dictionary. Please note, if we have less than k items in the dictionary, we start picking touchpoints at random from the potential recommendation set. Additionally, if there is no N-Gram for QM based on the historical data, we make k recommendations using a random function that picks from the potential recommendation set. We have discussed numerous ways in which these scenarios could be optimized, such as using the semantic search model if a subsequence does not have an N-Gram or if the frequencies are too low. We have decided to pick recommendations at random because we wanted to isolate the performance of the N-Gram approach from other techniques, without adding complexity to the model evaluation and allowing us to compare the model results.

5.4 Model Evaluation – Frequency N-Gram

The n-Gram recommendation model objectives were the same ones as the semantic search-based one – we wanted customers to interact with the recommendations and we wanted customers to find their desired answer in our recommendation list, leading to a positive experience. To ensure a fair comparison with our semantic search-based system, we evaluated the n-gram recommender using the same offline metrics: Hit Rate, MRR, MAP, and NDCG. For more details about each of these metrics, please read Chapter 4.7.2 Offline Evaluation with Historical Data:. This parallel evaluation strategy allowed us to directly contrast the effectiveness of both systems using a consistent benchmark. Just like in the semantic-based model evaluation, we leveraged the 80/10/10 historical dataset split, where 10% of the historical journeys were used for testing purposes. For the evaluation stage, we chose to only construct a bigram. The choice to construct a bigram model (n=2) was directly influenced by our coherence study, which revealed that customer focus tends to shift rapidly. This finding suggested that longer n-gram sequences might not as accurately reflect the immediate interests of customers, thus potentially reducing the relevance of the recommendations. Therefore, the results discussed are based on the bigram approach, considering various numbers of recommendations to gauge the system's effectiveness. In the spirit of academic thoroughness, it is important to note that during the evaluation phase, a data discrepancy was identified within the multi-target configurations of the n-gram model. To maintain the integrity of our findings, these results have been excluded from this presentation, with a more in-depth multi-target analysis earmarked for future exploration. We are presenting only the single-target analysis.

5.5 N-Gram Experiment Results

Simulation	MRR	MAP	NDCG	Hit Rate
kgram_@1	0.0980	0.0980	0.0980	0.0980
kgram_@2	0.1138	0.1138	0.1179	0.1295
kgram_@3	0.1203	0.1203	0.1276	0.1490
kgram_@5	0.1249	0.1249	0.1360	0.1693
kgram_@10	0.1279	0.1279	0.1431	0.1909

Figure 5-2: N-gram Results

The analysis of the recommender systems yielded insights consistent with our initial expectations. As we increased the number of recommendations provided by the n-gram model, we observed a corresponding improvement in accuracy. However, a notable observation was that even when offering up to ten recommendations, the n-gram model did not fully match the performance of the semantic search model when it provided only five suggestions. Specifically, both models hovered around a 19% accuracy rate under these conditions. Despite the close percentages, this minor gap becomes more pronounced when we consider the sheer volume of customer interactions across numerous journeys. In scenarios where only five recommendations are made, the semantic search model surpasses the n-gram model, demonstrating 19.09% accuracy compared to 16.9%. At scale, this difference is not only statistically significant but could also have substantial implications for the customer experience. In a somewhat counterintuitive twist, our analysis revealed that with a single recommendation, the bigram model, at a 9% accuracy rate, slightly outperforms the semantic search model's 7%. It's crucial to interpret these percentages in the right context. While a 7% or 9% hit rate might suggest that the correct recommendation is made only a small fraction of the time, it does not necessarily imply that other recommendations are irrelevant. They may well be contextually appropriate, a hypothesis that merits further investigation through live interactions or A/B testing. What we can infer from these figures is that in 7% to 9% of cases, we are highly confident that the customer's

journey will conclude with the content offered by the recommendation system. By and large, this analysis reveals that implementing a more sophisticated model such as the semantic search based one would show a return on investment, outperforming the results of a more rudimentary system such as the n-Gram. When we consider factors such as the user experience and we assume that people would rarely pay attention to more than 5 recommendations, the difference in performance between the two models is meaningful, particularly given the massive scale at which such models would be deployed as part of the AWS Support knowledge base.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Third Design: Deep Learning Hybrid Model

6.1 Overview

Building upon the foundational concept of semantic search explored in the first model, for our third design we devised a hybrid model that integrates the contextual intelligence of semantic embeddings with the predictive strength of a deep neural network (DNN). This innovative model aimed to address the challenges inherent to the volatility of customer journeys by harnessing the complementary capabilities of semantic search, which filters target touchpoints that are contextually relevant, and of machine learning, which captures customer behavior patterns regardless of semantic relationships within the journey, further narrowing down the set of potential target touchpoints. At the heart of this hybrid approach is the pursuit of a more granular and predictive analysis of customer behavior. The model operates in three principal stages: the first stage leverages semantic search to identify the five nearest touchpoint neighbors of a customer journey's summary embedding. In the second stage, the summary embedding together with its five neighbors are fed into a deep neural network as inputs, and the network is trained to predict the vector embedding of the customer journey target touchpoint. Lastly, in the third stage, semantic search is once again performed using the vector embedding prediction generated by the trained deep neural

network. The embeddings retrieved from the AWS Support vector database we built earlier are the recommendations of this hybrid system.

6.2 Building the Hybrid model

6.2.1 Training the DNN

To start off, the training dataset for the deep neural network was put together. The inputs of the deep neural network consisted of pairs of six embeddings, one representing the summary of the journey and its five nearest neighbors. To compile these pairs, for each customer journey in our training split of the golden dataset derived in Chapter 3.6 The Golden Dataset, a summary embedding was generated. The customer journey summary embedding was derived by mean pooling the three most recently visited touchpoints in a customer journey. Then the five nearest neighbor embeddings were selected using cosine similarity against the entire vector database of articles in the AWS Support knowledge base. Please note, all the embeddings mentioned, as well as cosine similarity searches, were implemented using the fine-tuned SBERT model built for the semantic search model and described in Chapter 4.6Building the Semantic Search model. Oncea set of six embeddings were obtained for each customer journey in the training dataset, we fed them as inputs into a Deep Neural Network. The model developed was a simple Multi-Layer Perceptron (MLP), designed specifically for predicting the next touchpoint in a customer's journey by analyzing sets of semantically related touchpoint embeddings. The first component of the model was a Flatten layer because the input data comes in as sequences of 6 touchpoints, each represented by a 384-dimensional vector. Flattening these inputs prepares them for processing in the dense layers by converting them into a single vector. After flattening, the model uses two dense layers; The first dense layer has 758 neurons and uses the ReLU activation function. This layer is designed to capture and transform the input data into a higher-level representation, which is crucial for learning complex patterns. The second dense layer follows, with 384 neurons, also activated by ReLU. This layer

continues the process of feature extraction and transformation, but with a focus on refining and concentrating the representation prepared by the first layer. The choice of two layers and the ReLU activation function was aimed at creating a network that was capable of learning non-linear relationships without being overly complex. This setup struck a balance between learning efficiency and computational demand. The final layer is another dense layer with 384 units, matching the dimensionality of our touchpoint embeddings, with a linear activation. This layer outputs a vector meant to represent the target touchpoint of the journey. For training, the model was compiled with the Adam optimizer and mean squared error (MSE) as the loss function. This choice was based on Adam's effectiveness in handling sparse gradients and MSE's suitability for regression tasks, where we were predicting continuous values in the form of a target touchpoint vector embedding. The model was trained using batches of 32 examples over 10 epochs. This training strategy was selected to ensure that the model had enough iterations to learn from the data without overfitting, given the complexity of the task at hand.

6.2.2 Recommendation System Design

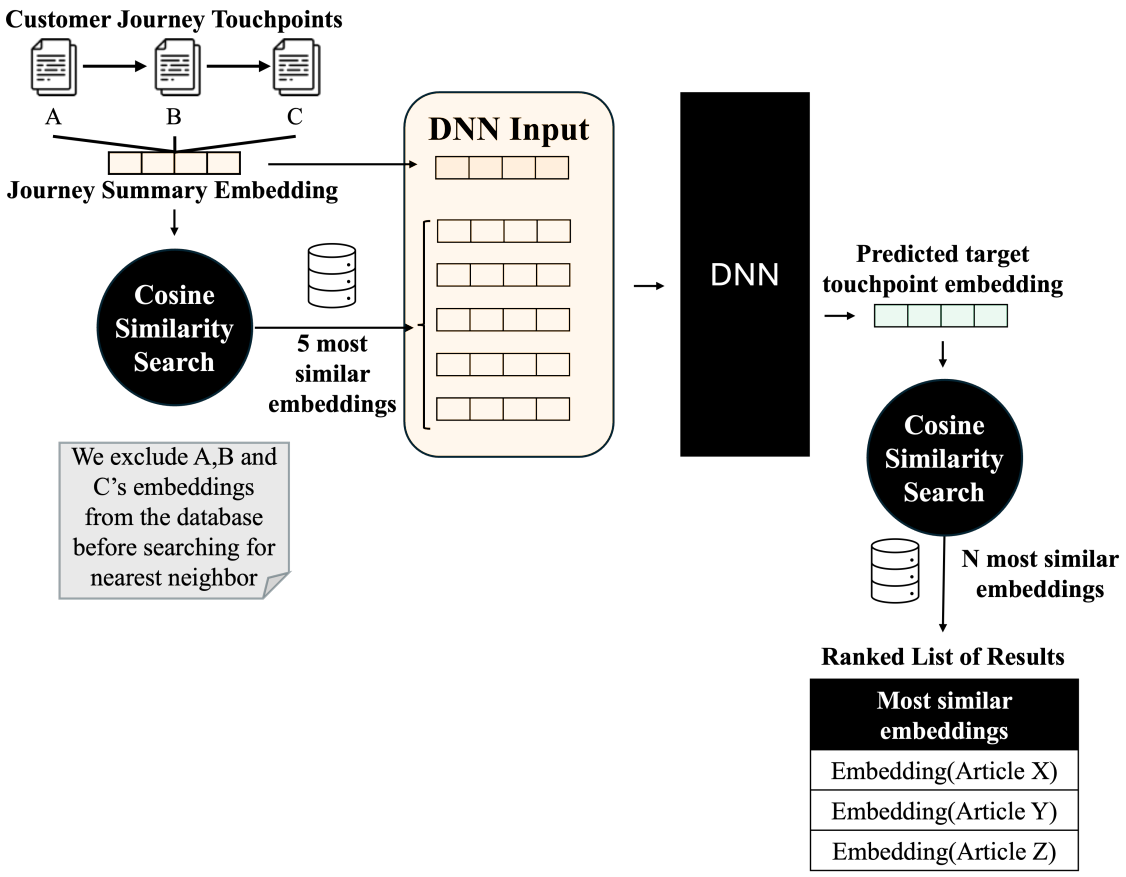

Figure 6-1: DNN Architecture Design

Once the deep neural network was trained to predict the target touchpoint of a customer journey using its summary embedding and the five nearest neighbors of this embedding,this model could be used to build a new recommendation system. Similarly with the training methodology, for out model to make a prediction, the 6 vector embedding inputs are required. At the same time, this recommendation system had to be designed with a set of parameters that would allow the user to run a set of experiments whose results could be compared with the other two models developed during this research project. Therefore, the recommender system was designed to accept the nTouchpoints parameter and the nRecommendations parameter. When a new customer journey is evaluated for the purposes of making target touchpoint recommendations, the most recent nTouchpoints are aggregated via mean pooling.

Then, the five nearest neighbors are found for this summary embedding and the set of six vectors is fed into the trained DNN. Once the DNN generates a predicted target touchpoint embedding, semantic search is performed once again to find the nRecommendations embeddings from the AWS Support vector database that are most semantically similar with the DNN output. As mentioned during the training stage, all the embeddings and vectors search operations are performed by the fine-tuned SBERT model designed for our first recommendation system.

6.3 DNN Experiment Results

Simulation	MRR	MAP	NDCG	Hit Rate
1touchpoints_@1	0.026	0.026	0.026	0.026
2touchpoints_@1	0.023	0.023	0.023	0.023
3touchpoints_@1	0.023	0.023	0.023	0.023
5touchpoints_@1	0.021	0.021	0.021	0.021
1touchpoints_@5	0.043	0.043	0.052	0.077
2touchpoints_@5	0.038	0.038	0.046	0.069
3touchpoints_@5	0.039	0.039	0.046	0.069
5touchpoints_@5	0.037	0.037	0.045	0.069
1touchpoints_@10	0.049	0.049	0.065	0.120
2touchpoints_@10	0.044	0.044	0.061	0.115
3touchpoints_@10	0.045	0.045	0.060	0.113
5touchpoints_@10	0.042	0.042	0.057	0.109

Simulation	MRR	MAP	NDCG	Hit Rate
1touchpoints_@1	0.127	0.016	0.029	0.127
2touchpoints_@1	0.110	0.014	0.025	0.110
3touchpoints_@1	0.106	0.014	0.024	0.106
5touchpoints_@1	0.103	0.013	0.023	0.103
1touchpoints_@5	0.198	0.036	0.070	0.326
2touchpoints_@5	0.171	0.030	0.058	0.279
3touchpoints_@5	0.166	0.029	0.057	0.271
5touchpoints_@5	0.162	0.028	0.055	0.266
1touchpoints_@10	0.211	0.045	0.093	0.423
2touchpoints_@10	0.184	0.037	0.078	0.379
3touchpoints_@10	0.178	0.036	0.076	0.363
5touchpoints_@10	0.173	0.035	0.073	0.355

Figure 6-2: DNN Results

How to interpret the table entries:

- **2touchpoints@5**:
  - **2 touchpoints** means that the two most recent touchpoints in a customer journey were mean pooled to generate the summary embedding which is then used to find its 5 nearest neighbors before being fed into the DNN
  - **@5** means that we made 5 recommendations which could include target touchpoints
- The table on the left reflects metrics when journeys have a single possible target touchpoint while the table on the right reflects metrics when journeys have

multiple possible target touchpoints

When analyzing the performance of the hybrid recommendation system, which was designed to capture the strengths of the fine-tuned SBERT model with the predictive capabilities of a trained deep neural network (DNN), several key observations emerged regarding its efficacy in accurately predicting customer journey touchpoints. Despite the model's innovative design, aimed at leveraging the strengths of both components, the results highlight a notable drop across all metrics, particularly when comparing its performance against the semantic search system. Experiments for both single and multi-target touchpoint definitions were run. As a reminder, single target meant that each customer journey had only one acceptable successful recommendation, the actual target touchpoint of the customer journey within the test dataset. For multi-target, just as the name suggests, multiple targets are accepted for a customer journey. The multiple targets accepted are all the target touchpoints in the test dataset that follow a second to last touchpoint in successful journeys. For example, if the test dataset had the journeys A – B – C , N - M – B – T, S – Q – R – T – B – H, then we would note that B is followed by C, T, H is successful customer journeys where B was the second to last touchpoint. Therefore, for a test journey J – O – U – R – B - ?, we would count any of the C, T, H touchpoints as a successful recommendation. The hybrid model's accuracy, as measured by hit rates, was significantly lower than anticipated for both singular and multi-target scenarios. For the hybrid DNN model, when making 5 recommendations, we obtained a hit rate of at most 7.8% for single target and 32% for multi-target. At the same time, the n-Gram model displayed a hit rate of 17% for single target and 5 recommendations, while the semantic search model displayed hit rates of 20% for single target at five recommendations and 50% for multi-target. This discrepancy suggests a fundamental limitation in the hybrid model's architecture to capture the complex dynamics between customer intent and the subsequent journey touchpoints. Although the integration of a trained DNN was intended to enhance the semantic similarity capabilities instilled in the fine-tuned SBERT model, the synergy expected from this combination did not materialize as effectively as predicted. The analysis suggests that the model struggles to accurately

capture and interpret customer intent. This could be attributed to the complexity and variability of customer behavior, which may not be fully captured by the model's current architecture, or the training data used. Preliminary evaluations hint at possible issues with overfitting, suggesting that the model might not have been adequately trained to generalize from the training data to unseen data. The analysis underscores a critical need for re-examining the underlying assumptions of the hybrid model's design and its operational mechanisms.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 7

Chapter 7: Recommendations & Future Work

7.1 Recommendations

7.1.1 Overview

In summary, our study not only challenges traditional notions about the utility of historical data in recommender systems but also highlights the potential of fine-tuned semantic search techniques. These insights pave the way for more efficient, responsive, and relevant recommendation systems in dynamic customer interaction environments.

7.1.2 Perform Cost-Benefit Analysis

A critical step before scaling our model is conducting a thorough cost-benefit analysis. The aim is to ensure that the projected annual savings from implementing our system outweigh the investment required for its production deployment. If the savings exceed the investment, it would be prudent to proceed with production-scale implementation.

7.1.3 Explore Broader Applications of Semantic Search

Given the success of semantic search in our study, we recommend exploring its applicability in other domains: Internal Information Retrieval for SMEs: Utilize semantic search to swiftly surface relevant internal information for Subject Matter Experts (SMEs) addressing customer tickets. Content Development Aid: When creating new content, employ semantic search to bring forth semantically similar information already available across the division. This can ensure content uniqueness and relevance.

7.2 Future Work

7.2.1 Overview

The conclusion of this thesis is not an end, but a starting point for future scholarly inquiry. Our work here has highligted the potential of semantic search in creating recommendation systems, laying a foundation for new research pathways. As the next steps are charted, a focus on practical application and a deeper understanding of data will guide these endeavors. Each proposed area for future work seeks not only to refine and expand upon the current findings but also to develop tools that better serve the evolving needs of customers. By embracing these future directions, research can continue to strive for excellence in creating intelligent, responsive, and user-focused support systems.The following areas have been identified for further exploration:

7.2.2 Refinement of Customer Journey Data Models

A refined analysis of customer journeys, incorporating the temporal aspects of user interactions, could yield sophisticated sub-journey segmentation. This approach aligns with findings from the coherence study, indicating that customers' focus can shift rapidly. By dissecting journeys into more discrete episodes, we can revisit the hypothesis regarding the value of historical touchpoints and potentially uncover a more accurate prediction mechanism.

7.2.3 Enhancements to Semantic Search Models

The semantic search models, as currently implemented, are subject to a key limitation: the maximum sequence length of the 'all-MiniLM-L6-v2' SBERT model, which is set at 256 tokens. This cap imposes constraints on the quantity of textual information that can be effectively processed. The context window for the model thus becomes less than 256 words, potentially truncating longer articles and leaving out critical information that could impact recommendation performance. To address this issue and further enhance our semantic search model, the following advancements are proposed:

- Integration of Larger Context Window Models: Exploring language models with larger context windows will allow for a more comprehensive analysis of longer documents, ensuring that valuable content is not overlooked in the embedding process.

- Selective Content Summarizing: Developing an intelligent summarizing algorithm that can distill longer texts into their most salient points within the token limit could be a viable interim solution. This approach would retain key information for embedding, even in cases of extensive content.

7.2.4 Development of n-Gram Models

For the n-gram models, investigating sequences beyond bigrams may enhance predictive power. Exploring trigrams or higher n-grams could capture more complex patterns in user behavior, although it's crucial to balance the increased computational demand with performance benefits.

7.2.5 Advancements in Deep Neural Networks

In the realm of deep learning, there is ample scope for experimenting with different architectures and more expansive datasets. Incorporating the timing of touchpoints as a feature in the neural network, for example, could add a layer of temporal awareness to the recommendations.

7.2.6 A/B Testing for Practical Insights

To ensure the effectiveness of semantic search-based recommendation systems in practical, real-world settings, it is imperative to conduct A/B testing. This method allows for direct comparison of the performance of a new system against an existing one, focusing on key metrics that indicate user engagement and satisfaction.

A/B testing is essential to evaluate the real-world efficacy of the system. User engagement metrics such as click-through rates, session durations, and the relevance of the recommendations provided will offer tangible evidence of the system's impact on user behavior and satisfaction.

Assessing the user interface and experience (UI/UX) is also important, as the design and usability significantly influence how users interact with and perceive the recommendations. This phase of testing helps understand the effectiveness of different UI/UX designs in enhancing user interaction and overall experience.

Bibliography

[1] Bonnie Berger, Michael S. Waterman, and Yun William Yu. "Levenshtein Distance, Sequence Comparison and Biological Database Search". In: *IEEE Transactions on Information Theory* 67.6 (June 2021), pp. 3287–3294. ISSN: 0018-9448, 1557-9654. DOI: 10.1109/TIT.2020.2996543. URL: https://ieeexplore.ieee.org/document/9097943/ (visited on 05/02/2024).

[2] Samuel R. Bowman et al. "A large annotated corpus for learning natural language inference". In: (2015). Publisher: [object Object] Version Number: 1. DOI: 10.48550/ARXIV.1508.05326. URL: https://arxiv.org/abs/1508.05326 (visited on 05/02/2024).

[3] Claire Yi Tian Chan and Douglas Petrikat. "Self-Service Technology: Benefits and Challenges". In: *Journal of Computer Science and Technology Studies* 4.2 (Nov. 2022), pp. 118–127. ISSN: 2709-104X. DOI: 10.32996/jcsts.2022.4.2.14. URL: https://al-kindipublisher.com/index.php/jcsts/article/view/4485 (visited on 04/07/2024).

[4] Geoff Colvin. "How Amazon's cloud took the world by storm". English. In: *Fortune* December 2022/January 2023 (Nov. 2022). URL: https://fortune.com/longform/amazon-web-services-ceo-adam-selipsky-cloud-computing/.

[5] Microsoft Corporation. *STATE OF GLOBAL CUSTOMER SERVICE*. English. Tech. rep. Microsoft, 2017, p. 32. URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://info.microsoft.com/rs/157-GQE-382/images/EN-CNTNT-Report-DynService-2017-global-state-customer-service-en-au.pdf.

[6] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (2018). Publisher: [object Object] Version Number: 2. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805 (visited on 04/07/2024).

[7] Serrano Luis. *What Are Word and Sentence Embeddings?* Language. Jan. 2023. URL: https://txt.cohere.com/sentence-word-embeddings/.

[8] Ron Miller. *Tech Crunch*. English. July 2016. URL: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/.

[9] Andras Molnar and Russell Golman. "Impatience for information: Curiosity is here today, gone tomorrow". en. In: *Journal of Behavioral Decision Making* 37.1 (Jan. 2024), e2360. ISSN: 0894-3257, 1099-0771. DOI: 10.1002/bdm.2360. URL: https://onlinelibrary.wiley.com/doi/10.1002/bdm.2360 (visited on 04/26/2024).

[10] Britney Muller. *BERT 101: State Of The Art NLP Model Explained*. Mar. 2022. URL: https://huggingface.co/blog/bert-101.

[11] Globe Newswire. *Self-Service Technology Market to Touch USD 72.51 Billion by 2030, at a CAGR of 11.27% - Report by Market Research Future (MRFR)*. May 2022. URL: https://www.globenewswire.com/en/news-release/2022/05/03/2434187/0/en/Self-Service-Technology-Market-to-Touch-USD-72-51-Billion-by-2030-at-a-CAGR-of-11-27-Report-by-Market-Research-Future-MRFR.html.

[12] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation". en. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://aclweb.org/anthology/D14-1162 (visited on 04/07/2024).

[13] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: (2019). Publisher: [object Object] Version Number: 1. DOI: 10.48550/ARXIV.1908.10084. URL: https://arxiv.org/abs/1908.10084 (visited on 05/01/2024).

[14] Borja Requena et al. "Shopper intent prediction from clickstream e-commerce data with minimal browsing information". en. In: *Scientific Reports* 10.1 (Oct. 2020), p. 16983. ISSN: 2045-2322. DOI: 10.1038/s41598-020-73622-y. URL: https://www.nature.com/articles/s41598-020-73622-y (visited on 08/07/2024).

[15] Stephen Robertson. "Understanding inverse document frequency: on theoretical arguments for IDF". en. In: *Journal of Documentation* 60.5 (Oct. 2004), pp. 503–520. ISSN: 0022-0418. DOI: 10.1108/00220410410560582. URL: https://www.emerald.com/insight/content/doi/10.1108/00220410410560582/full/html (visited on 05/02/2024).

[16] Mike Schuster and Kaisuke Nakajima. "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Kyoto, Japan: IEEE, Mar. 2012, pp. 5149–5152. ISBN: 978-1-4673-0046-9 978-1-4673-0045-2 978-1-4673-0044-5. DOI: 10.1109/ICASSP.2012.6289079. URL: http://ieeexplore.ieee.org/document/6289079/ (visited on 05/02/2024).

[17] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: (2015). Publisher: [object Object] Version Number: 5. DOI: 10.48550/ARXIV.1508.07909. URL: https://arxiv.org/abs/1508.07909 (visited on 05/02/2024).

[18] Luis Serrano. *What Are Word and Sentence Embeddings?* Jan. 2023. URL: https://cohere.com/blog/sentence-word-embeddings.

[19] LGO Staff. *Partner Companies - Amazon*. English. URL: https://lgo.mit.edu/partnercompanies/amazon/.

[20] Statista. *Annual revenue of Amazon Web Services (AWS) from 2013 to 2023*. English. Sept. 2024. URL: https://www.statista.com/statistics/233725/development-of-amazon-web-services-revenue/#:~:text=In%202023%2C%20Amazon%20Web%20Services,dollars%20with%20its%20cloud%20services..

[21] Statista. *Do you expect a brand or organization to have an online self-service support portal?* 2018. URL: https://www.statista.com/statistics/810374/share-of-customers-by-if-they-expect-brands-to-have-a-self-service-portal/.

[22] Muhammad Taqi Raza. *What is the difference between triplet and contrastive loss?* 2024. URL: https://www.educative.io/answers/what-is-the-difference-between-triplet-and-contrastive-loss.

[23] Ashish Vaswani et al. "Attention Is All You Need". In: (2017). Publisher: [object Object] Version Number: 7. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (visited on 04/07/2024).