November 2023

Continual Learning System with Sentence Embeddings

Anonymous

Continual Learning System with Sentence Embeddings

Abstract

Recent improvements in computing power and ambient technologies have opened up new perspectives in ambient intelligence and proactive software systems. The need emerges for an ambient and supportive system that deeply understands the user's needs and preferences under the current context. In this work, we propose a recommendation system that aggregates and understands the current situation and continuously learns from the user’s decision history. To leverage a rich pool of contextual data, the system dynamically changes the strategy and scope for contextualization and encodes multimodal data into unified embeddings. A contextual similarity database consisting of these embeddings is leveraged for finding, ranking and comparing scenarios. The proposed recommender is intended to fit into a decentralized service system and addresses challenges in application interaction, user engagement, user privacy and scalability.

Chapter 1: Background Information

With the advancement in computing hardware and ambient technologies, the human-computer interaction model will undergo a fundamental paradigm shift from reactive systems to proactive systems in the near future [1]. For always-on, intelligent devices like XR devices and AR glasses to be further integrated into people's lives, they should be able to leverage the rich context of past and present to infer and fulfill the user's needs in an proactive, low-friction way. In other words, instead of having the user explicitly invoking applications, recommendations should dynamically emerge by monitoring the environment and anticipating what the user wants.

One way that this vision can be achieved is to actively compose recommendations based on the services available in the environment [3]. In this proactive software framework, recommendations are generated by having the services proactively propose to the user. For example, when the user walks into the living room, multiple services including smart light bulbs, smart speaker and smart tv will all propose to perform certain tasks for the user.

As services in the ambient environment are aggregated in a decentralized manner, a recommendation system is needed whenever multiple actions are proposed at the same time. The recommender should remember the user's preference and recommend items that are appropriate to the current context. Due to the high frequency of interaction between the user and device, the recommendation system's ability to learn from user feedback is essential to improving efficiency and reducing friction. This project focuses on building a recommendation system that incorporates contextual information for continual learning.

1.1 Challenges in Recommender Design

As the recommender operates in a new framework with a unique set of requirements and constraints, it needs to contend with a new set of challenges. We will identify and define some of the most important challenges.

1.1.1 Contextualization

The recommender system is designed to model the preference of the user using relevant contexts, such as time, location, user's intention etc. The main challenge lies in figuring out the scope of relevant contexts and the way to encode them.

All types of descriptive information about the user and the environment can be considered as context but not all contexts are relevant. For example, when the user enters the

living room, the ambient brightness is useful in deciding whether to switch on the light, but is irrelevant to the smart speaker in deciding whether to play music.

To efficiently and accurately represent the contexts, the recommender also needs to encode contextual data from an arbitrary set of sensors and services into a unified representation. Furthermore, because new services might be added or dynamically defined by the user at any time, the contextualization model should be able to dynamically comprehend new types of data. The main challenge lies in the diversity of multimodal data, both in terms of data type and semantics.

1.1.2 Privacy

Given the unprecedented level of personally-relevant context provided by head-mounted sensors and wearables, respecting and protecting the user's privacy must be embedded in the low-level system design as services are a fundamental building block of the proactive software system. If not designed with care, a proactive system can be distracting, intrusive or even harmful. The challenge is to define a data schema for training the recommender with significant limitations in the collection, use and analysis of identifiable user data.

1.1.3 Personalization

Recommenders should be able to provide personalized recommendations based on user preferences and history. The system should be able to define behaviors for each item-context-user combination. However, for an always-on device that aims to be a seamless part of daily life, the use of centralized machine learning and cloud-based AI would pose serious danger to user privacy. Therefore, the challenge lies in ensuring the quality of personalization in a privacy-preserving manner.

1.1.4 Cold Start

"Cold start" refers to a situation where the recommender system cannot draw inferences for a new item or a new user. This problem is more prevalent in ambient intelligent systems as the availability of services changes dynamically with the environment.

1.1.5 Explainability

Trustful persuasion and end-user programming are two of the important research perspectives in ambient computing [2]. For a proactive system, applications are invoked implicitly by observing the context rather than explicitly by the user. Therefore, the circumstances under which an application is invoked must be clearly defined and configurable.

Furthermore, the recommender needs to be able to explain to the user why an application is preferred over others, so the AI model must display a high level of explainability.

1.1.6 Scalability

The system should be able to incorporate and coordinate a large number of services and scenarios. If not designed properly, the inference speed, learning speed, and computational complexity can all deteriorate significantly as the number of services increases. Thus, the architecture should put emphasis on decoupling and the API and message format between components should have low complexity.

1.1.7 Developer Workflow

We want to foster an ecosystem that allows developers to design, implement and iterate on their applications as proactive services. The main difference for a proactive system is that the developers need to define the context to invoke the application and design the user interface accordingly.

To promote user engagement, developers might like the application to launch as often as possible, which can be a major source of user friction in a proactive system. We need to design mechanisms to avoid frequent invocation on a system level. In other words, we need to carefully balance developer access/permission, ease of development, personalization and user friction. Ideally, the system should encourage developers to specify strict and accurate conditions for the app to be invoked but also maintain full control over the individual recommendations.

Chapter 2: Research Goals and Outcomes

2.1 Research Goals

The goal of this research is to prototype possible software and AI model design of a recommendation system that learns to predict the user's preference under different situations over time. More specifically, given a set of entities and their status representing the current context, the recommendation system should be able to rank a list of proposed applications from the most likely to be accepted by the user to the least. Apart from the recommender model itself, the communication protocols between the recommender system and the rest of the framework will also be defined. We will also discuss how the new design addresses the challenges in contextualization, privacy, personalization, explainability, scalability and developer workflow.

2.2 Problem Formulation

Due to the novelty of the proactive framework, we will try to formulate the problem with some examples and explanations. We will describe the concept of context and proposals, which are the input for the recommender system.

2.2.1 Context as Entities and States

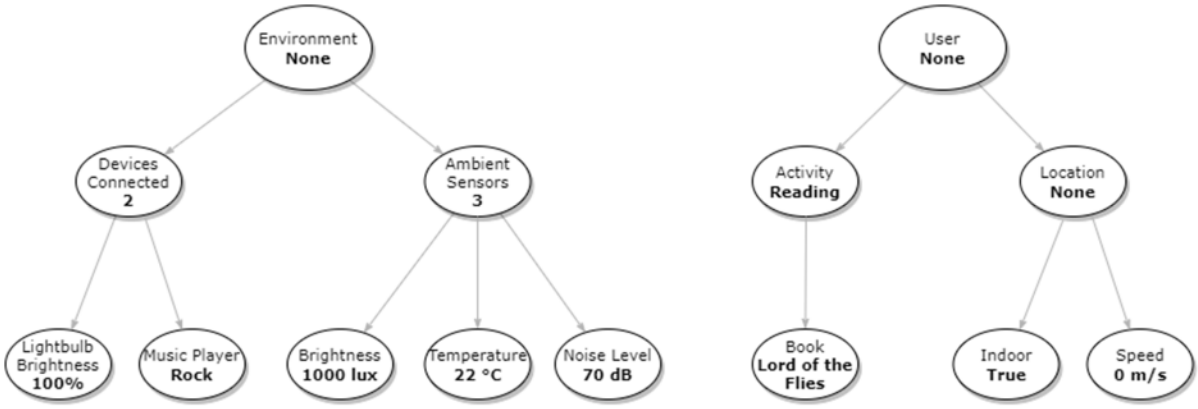

Fig. 1: An illustration of a scenario defined by a set of entities and their status. The entities are represented as a node in a tree to encapsulate structural and hierarchical information.

A context/scenario is defined by a set of entities and their status (Fig 1). An entity can either be one aspect of a tangible thing (light brightness, door open/shut, camera view) or an abstract concept (time of day, season of the year, user goal, user mood, user moving speed). To capture the hierarchical information, entities are arranged as nodes for a tree. At any time, the system will have access to only a subset of all available entities. There are 2 categories of entity: user attribute and environment attribute.

User Attributes are entities that describe the situation/condition/attribute of the user. These attributes are either detected (moving speed, skin temperature, heart rate) or inferred (user indoor/outdoor, user mood, active/inactive), and are merely observed. Notice that some of these attributes may not be instantaneous (activity last night, schedules for tomorrow) to be relevant.

Environment Attributes are entities that describe the status of the surrounding environment. These attributes are either detected (noise level, ambient light luminance, temperature) or reported by smart devices (lightbulb brightness, AC status, current playlist on speaker).

States are the minimum set of properties that adequately describe a specific entity. For a smart light bulb entity, its state can be a boolean value suggesting on/off or an integer value

ranging from 0 to 100 indicating the brightness level. For entity user mood, its property can be a categorical classification from a set of predefined moods (happy, anxious, energetic) or can be a value on the mood spectrum ranging from positive to negative.

2.2.2 Proposals

Services would propose either to invoke a certain application or to perform certain actions in the application. The level of details in the proposal may vary significantly depending on the type and functionality of the service. For example, the illumination service might propose to turn on or off the light, and the food delivery service might ask the user to make multiple selections in the application and check out.

2.3 Research Outcomes

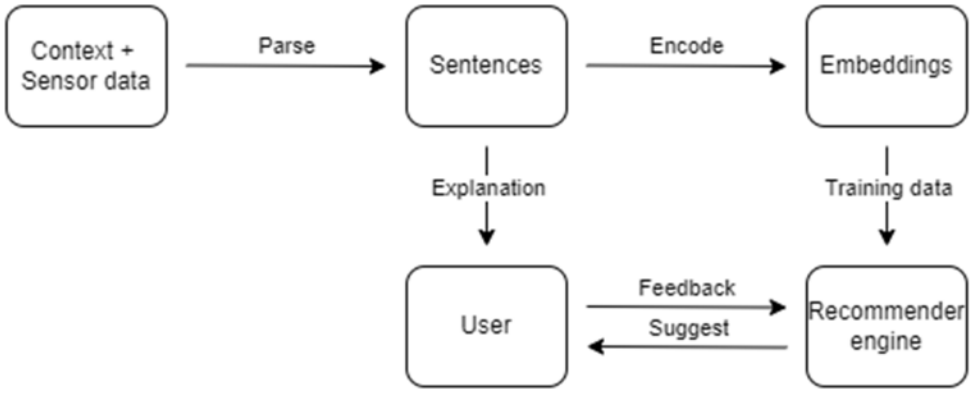

Fig. 2: Data flow of the recommendation system.

We designed and implemented a new recommendation system based on text semantic embedding. The system parses and represents all relevant information into human-readable text and uses these sentences as descriptions for context or proposals. The sentences are then either provided to the user as an explanation or encoded into semantic embeddings to provide data to the recommendation engine. By collecting explicit and implicit feedback from the user, the recommender engine stores a database of context-proposal-feedback combinations, which is used to learn the user's personal preference over time. The recommender engine uses a rule-based algorithm for deciding the score for each proposal and the text encoder is the only component that uses deep-learning models.

In terms of the communication protocol, we propose to let the services describe the exact circumstances under which they propose. These proposal conditions not only provide explicit explanations to the user but also serve as guidance for the scope of contextualization. Adding

texts as an intermediate representation of contextual data greatly improves the robustness and comprehensiveness of the system.

2.4 Previous Work

Ambient computing promotes a new vision of an all-inclusive, context-aware framework of bottom-up computing [2][12], which requires a drastic redesign of how software components are aggregated. [3] proposed to composite applications based on the components available at the time in the ambient environment using a new type of middleware, namely an Opportunistic Composition Engine (OCE). The OCE is a multi-agent system that periodically detects the components and their services presented in the ambient environment. This paper also described a communication protocol and an agent-level learning schema based on user feedback via reinforcement learning. However, our proactive interaction system diverges from the OCE system as it employs a rich pool of contextual data and disallows agent-level learning via deep learning due to privacy and explainability concerns. Nevertheless, [3] provides a great introduction to a proactive ambient system.

Traditionally, recommender systems research [4] aims to predict the users' preferences (ratings) for a set of items, such as movies [6], music[8], or books [7]. Three main types of information filtering techniques have been proposed and widely adopted, namely collaborative filter, content-based filtering and the hybrid approach [5][9-11]. Collaborative filtering is based on the assumption that other users with similar preferences (similar ratings) will rate other items similarly. Therefore, a user profile can be established by finding other users that behave similarly. Content-based filtering utilizes the user's history and tries to find recommendations by searching for items similar to those that the user preferred in the past. The hybrid approach is the combination of these two. However, many of these recommendation techniques cannot be directly used for a recommender system in ambient devices. At any time, the items that an ambient device can choose from are highly restricted by the available components in the ambient environment, so all decisions the user made in the past should be seen as highly conditional. As the device might also be exposed to the same choices repeatedly, the level of personalization and contextual understanding needed is much higher than in a general-purpose recommendation system. Therefore, collaborative filtering won't help other than situations like the "cold start" (new applications) and content-based filtering is highly restricted by the set of items available. An analogy would be the traditional recommender system is a store owner with a huge catalog of

items and can recommend based on customer's taste, while our recommender system is a butler that deeply understands the user and recommends according to context.

With advances in recommender system techniques, context-aware recommender systems [13-17] have gained more traction since the late 2000s. Instead of the traditional view, where the recommender tries to estimate a rating score for each user-item combination, context-aware recommender systems try to estimate a rating score for each user-item-context combination. This view drastically increased the dimensionality of the data. Most of the proposed approaches treat contextual information (knowledge base) as complementary to personalization and try to alter ratings based using context prefiltering, postfiltering or modeling. Such contextual recommender systems have been implemented at a smaller scale. [18] uses context prefiltering in a personal shopping assistant and [19] uses context postfiltering in personalized advertising. For [18], the context consists of the history of a set of well-defined user behaviors. For [19], all contexts are from a set of positive physical attributes like temperature or weather. We drew some inspiration from these designs, but due to the extensive scale and diverse type of our contextual data, our system is very different from most context-aware recommender systems. In our work, rather than treating the context as additional information, we treat it as a deciding factor for item rating and employ a purely context-based filtering technique.

Chapter 3: Methods

For the recommendation engine, we employ a memory-based modeling approach by establishing a personal history database that records the user's decision under different scenarios. When a new set of applications are proposed, the recommender module will actively search the database and look for similar scenarios in the past. The user acceptance score for each proposal is decided by its similarity to the past proposals and the user's feedback to them.

To accurately estimate the similarity between scenarios and applications, we need to extract the corresponding semantic representation from them. For the purpose of explainability and trustful persuasion, we added an intermediate process of encoding multimodal information to text. As shown in Fig 2, doing this not only enables decision explanation but also allows for end-user programming, where the user can use text to specify certain conditions for an application to be invoked.

To address privacy concerns, user history is stored on the device as embeddings of scenarios and proposals. With a new device, we inject the database with embeddings of the most common decision in typical scenarios as “prior knowledge” (i.e. the user is home -> turn the light on; the user is leaving home -> show weather report). When new services are added or new scenarios are encountered, the system can then learn from the collective prior knowledge and recommend the most probable decision. Note that the injection is one-directional thus won’t expose any personal information. As the user engages with the system, it will then gradually learn the user’s personal preference and become truly personalized. Therefore, using the semantic embeddings increased the robustness of the model and mitigated the effect of a “cold start”.

3.1 Recommender Module Architecture

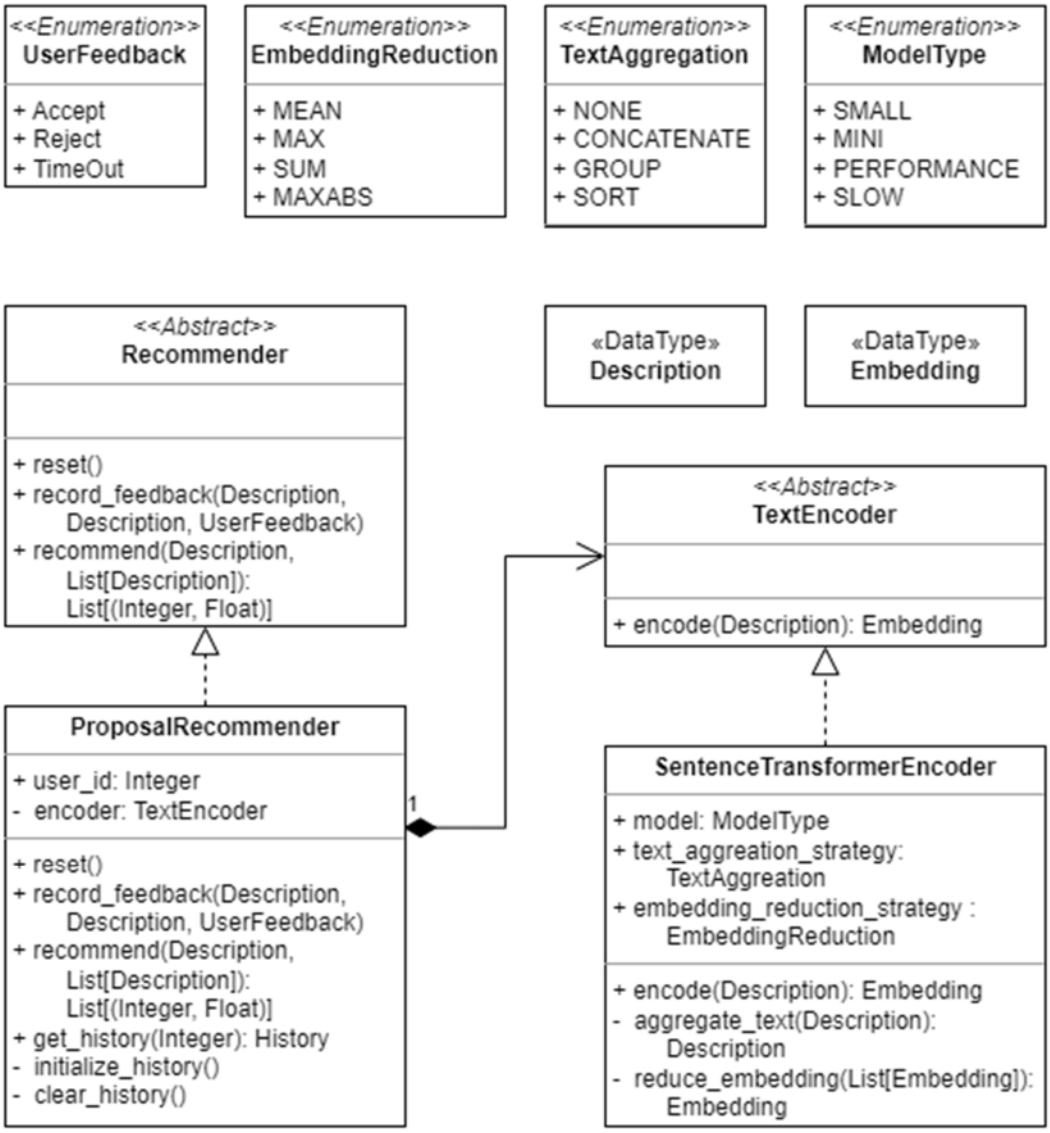

Fig. 3: Software architecture of the recommendation system.

In our design, the *ProposalRecommender* class uses two functions: *record_feedback* and *recommend* to communicate with the rest of the recommendation engine. Whenever applications are proposed, the *recommend* function will be called to get a user acceptance score for each proposal. Whenever explicit (agree, refuse) or implicit (timeout) feedback is received by the system, the *record_history* function is called to insert the record into an on-device database. The *ProposalRecommender* has a *TextEncoder* object, which is responsible for encoding descriptions

(sentences) into embeddings (vectors of floats). Enumeration classes like *UserFeedback* and *EmbeddingReduction* (strategy) are also used to standardize the API.

3.2 Text Encoder

Recent developments in natural language processing techniques have greatly improved the quality of text embeddings [20-23]. Text embedding techniques aim at encoding text into a high-dimensional vector representation that can be used for predictive or exploratory tasks. Popular text embeddings model uses the hidden layer outputs of the transformer text classification models like BERT [20]. SBERT [22] utilizes transfer learning to modify the pre-trained encoder of BERT using siamese and triplet network structures. The SBERT network aims to derive semantically meaningful sentence embeddings that can be compared with cosine-similarity. For this project, we used the publicly available pre-trained SBERT.

Due to the high computation complexity of the transformer architecture, the SBERT architecture limits the total number of words to 256. We need to find a way to aggregate the sentences and preserve the semantic information. Two strategies have been proposed to avoid truncating the sentences. The first method encodes each individual sentence and aggregates the embeddings; the second method splits the array of sentences into partitions of less than 256 words and concatenates each partition into a paragraph for encoding.

3.3 Ranking Algorithm

To rank the proposals from most likely to be accepted to the least, we designed and implemented a ranking algorithm as follows:

Setup: define weight for each action: {Accept: $W_a > 0$, Reject: $W_r < 0$, TimeOut: W_{to}}.

1. The recommender is given the current context and a list of proposed applications $(a_1, a_2, \ldots, a_k)$.
2. Search the history to get N (context, proposal, action) triplets where its context is the most similar to the current scenario.
3. Calculate the similarity between the current context and the N past scenarios, namely $S_{c1}, S_{c2}, \ldots, S_{cN}$. For each proposal, calculate the similarity to the N past application, namely $S_{ai1}, S_{ai2}, \ldots, S_{aiN}$ for $i = 1,2,\ldots,k$.
4. The score for an application i is then calculated as $Score(a_i) = \sum_{j=1}^{N} (S_{cj} \cdot S_{aij} \cdot W_{a|r|to})$.

5. The proposed applications are ranked according to their scores. Proposals with scores under a threshold might be suppressed to reduce friction.

This algorithm ensures that, if the user has repeatedly accepted one application a under scenario S for more than $\frac{N}{2}$ times, under scenarios similar to S, a will always be ranked higher than any other applications. In case of new proposals, we proposed that the installation of a new application would automatically inject the database with at least $\frac{N}{2}$ acceptance records to ensure that the system will propose the new application the next time. To understand how a service specifies the condition to propose, we will describe the communication dynamics in the next section.

3.4 Service-recommender Communication

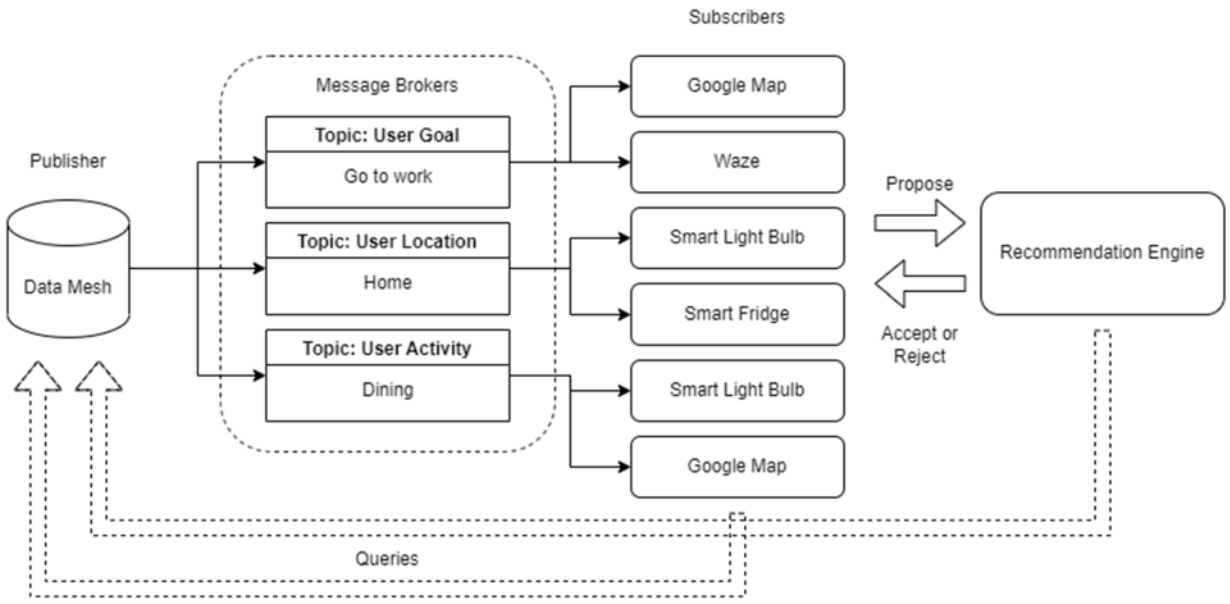

Fig. 4: Example of the publish-subscribe pattern employed by the system.

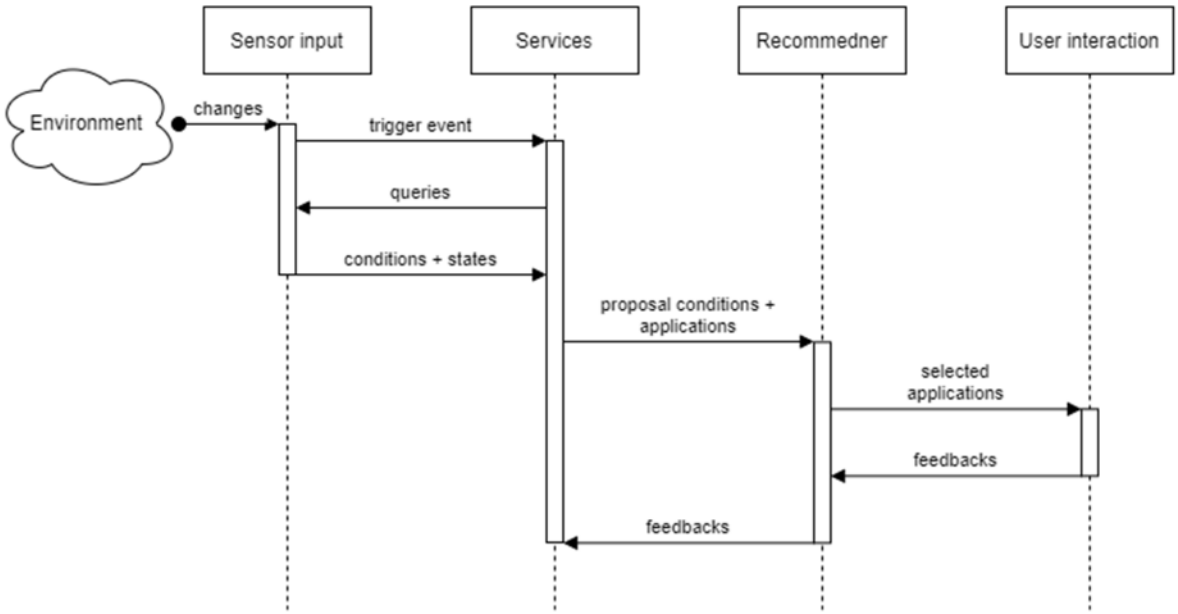

Fig. 5: Proposal life cycle.

To create a proactive system, we propose to define services as a simple rule-based component that provides proposals under a set of conditions. Each service will subscribe to one topic and will be triggered when the trigger event happens (Fig. 4).

The trigger event specifies when an entity changes to a certain state. For example, the Google Map service will be triggered when the user action of “driving” is confirmed. This marks the start of the service’s lifecycle.

The proposal conditions specify all the conditions to be met before an application is proposed. For example, for the smart light bulb service, if the trigger event is “smart light bulb is connected”, the proposal conditions can be [“user is indoor”, “smart light bulb is turned off”, “ambient brightness <= 500 lux”]. The proposal conditions are also the explanation given to the recommender and the user. These conditions are explicitly defined and configurable by the user, therefore enabling end-user programming.

When the recommender receives proposals, it will use the current context to get the ranking for all applications. The context is a union of all the proposal conditions received by the recommender in one cycle which can also be used to explain why a certain application is proposed. When new services are added, the context will expand to include the new conditions. As these conditions have all been checked before the service proposes, the explanatory context forms an accurate description of the relevant context.

Chapter 4: Results and Discussions

Collecting a dataset for the described ambient system would require large-scale user study and data collection. Unfortunately, we are still in the process of collecting this data. Therefore we will present the results by showing experimentation and discussing how the new system addresses the challenges described in section 1.2.

4.1 Prototype and Experimentation

A prototype version of the recommendation system has been implemented in Python using the algorithms described above. For this experiment, {Accept: $W_a = 1$, Reject: $W_r = -1$, TimeOut: $W_{to} = 0$}.

For the demo, we describe a simple use case. The user wants to drink coffee and has installed the services of Starbucks and Tim Hortons. The user encounters the scenario where he is walking and is getting closer to both a Starbucks store and aTim Hortons store. To formulate a final recommendation, a beverage selection service is also installed to propose a list of beverages available at the specified store. We first choose to order a cappuccino from Starbucks and record an acceptance.

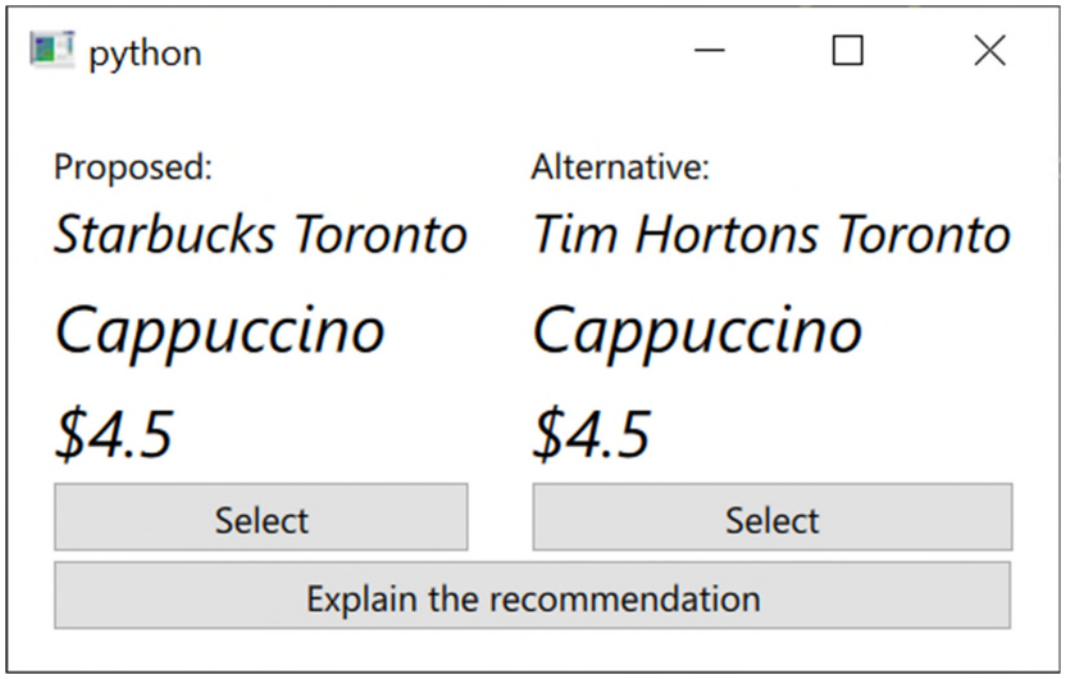

Fig. 6: Window capture after the user accepted the cappuccino from Starbucks.

As shown in Fig 6, after the user has chosen a cappuccino from Starbucks once, the system recognized the past scenario and recommended cappuccino from Starbucks again. Note

that although not proposed, cappuccino is recommended at Tim Hortons. This is because according to user history the closest context to choosing a drink at Tim Hortons is choosing a drink at Starbucks, therefore the same drink is recommended. In an environment where only the Tim Hortons store is available, cappuccino will still be proposed according to the user's past preference. Now let's choose a brewed coffee at Tim Hortons instead.

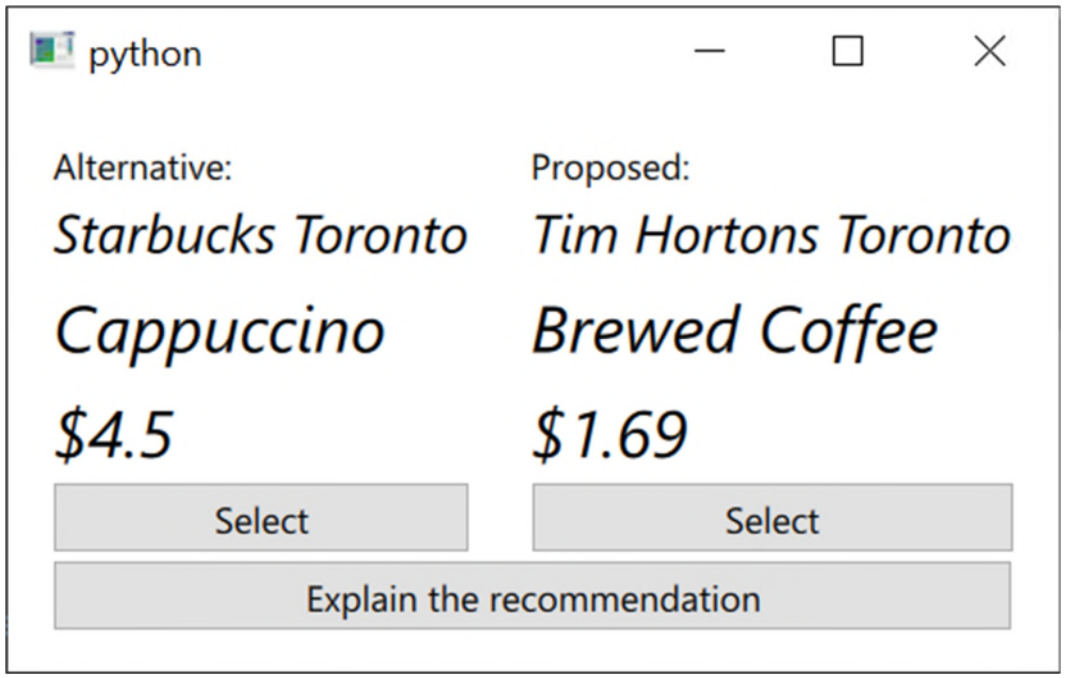

Fig. 7: Window capture after the user accepted the brewed coffee from Tim Hortons.

As shown in Fig 7, after the user has chosen brewed coffee from Tim Hortons, the system recommended the latest choice. Note that cappuccino is still recommended at Starbucks.

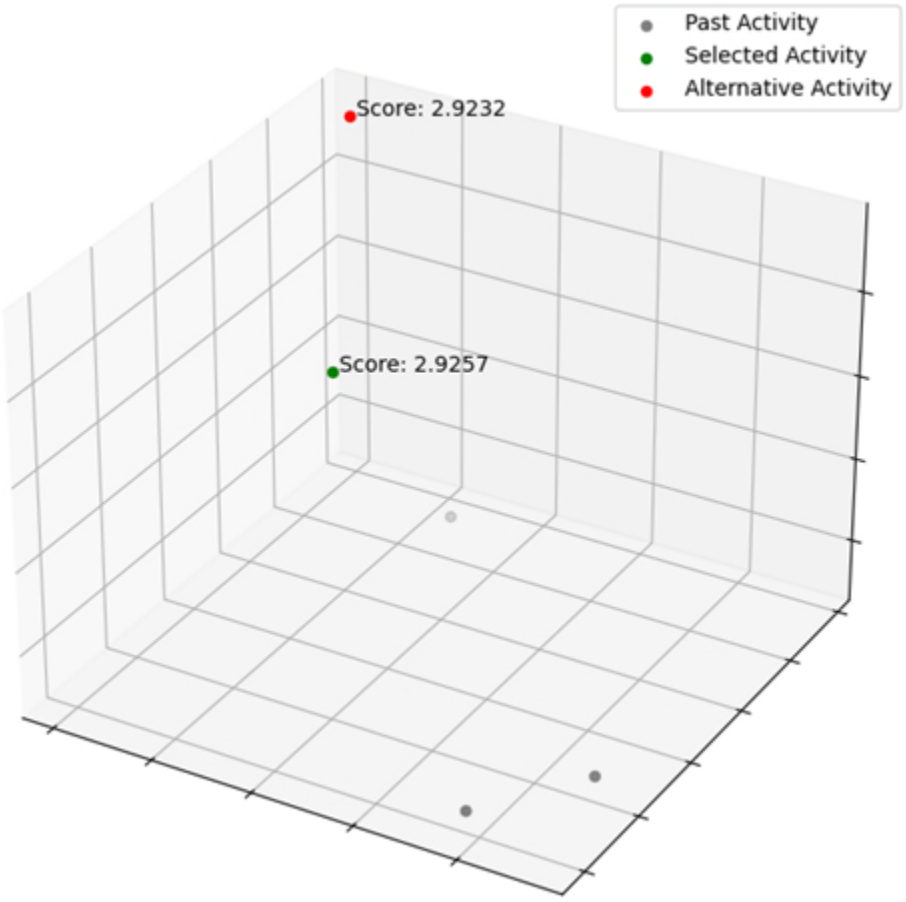

Fig. 8: Explanation provided as embeddings and scores.

In Fig 8, the two proposals are plotted against past activities (dimension reduces with PCA). Intuitively, the application whose semantic is closest to the past records is proposed.

4.2 Addressing the Challenges

4.2.1 Contextualization

All contexts are encoded into text descriptions and then encoded into high-dimensional embeddings. Compared to other types of data formats (video, image, representation), the text is more expressive and thus can be used as an intermediate encoding. The scope of the context is decided dynamically by the services to support exploratory feature selection and end-user programming.

4.2.2 Privacy

All personal data are stored on the device in the format of embeddings. The only way of collaborative filtering is by injecting prior knowledge into the database, which is one-directional and doesn't require any user profiling.

4.2.3 Personalization

As shown by the demo, our system supports a high level of personalization. It will actively adapt to the user's preferences and

4.2.4 Cold Start

As shown by the first recommendation in the experiment, although no prior history exists for Tim Hortons, the system was able to correctly identify the user's preference for a cappuccino. In cases where the device or user is new, collaborative knowledge can be selected and injected into the database.

4.2.5 Explainability

By our design, services are defined in a way that automatically provides an explanation (proposal conditions) whenever it is invoked. As all contexts are encoded in text, these conditions are inherently understandable by the user. However, how text is encoded into embedding is still a black-box process and requires further research to promote trustful persuasion.

4.2.6 Scalability

We try to promote scalability by keeping all processes on the device and truncating old entries in the database. However, both accessing the database and running SBERT are quite expensive. A new type of model or technique is needed to accelerate the inference time for SBERT and reduce energy consumption.

4.2.7 Developer Workflow

Due to how services are defined, developers will need to define the exact proposal conditions and balance between proposing vague scenarios (a small set of conditions) or specific scenarios (many conditions). When the scenario is too vaguely defined, the application is more often proposed but might risk being rejected more often. Furthermore, vague scenarios mean the application will be more likely proposed with other applications, so it is facing more competition. If the scenario is too specific, the user might never encounter such a situation and thus never use the application. By design, developers are encouraged to find specific and well-

defined scenarios for their applications. This not only mitigated the risk of spamming but also opened up new possibilities for ambient application development.

4.3 Text Encoder

To test out the two text embedding reduction strategies, we use the Stanford Natural Language Inference (SNLI) [24] dataset, which consists of sentence pairs (text and hypothesis) and their judgements (contradiction, neutral or entailment). To test the encoder, we randomly picked 20 texts to form a base scenario and replaced a random number of them with either their entailment, contradiction or neutral hypothesis to form alternative scenarios. If the sentence aggregation is valid, the base scenario will be closest to the entailment scenario and furthest to the contradictory scenario. Out of 100 randomly selected scenarios, the first method produces 84 correct predictions and the second method produces 63 correct predictions.

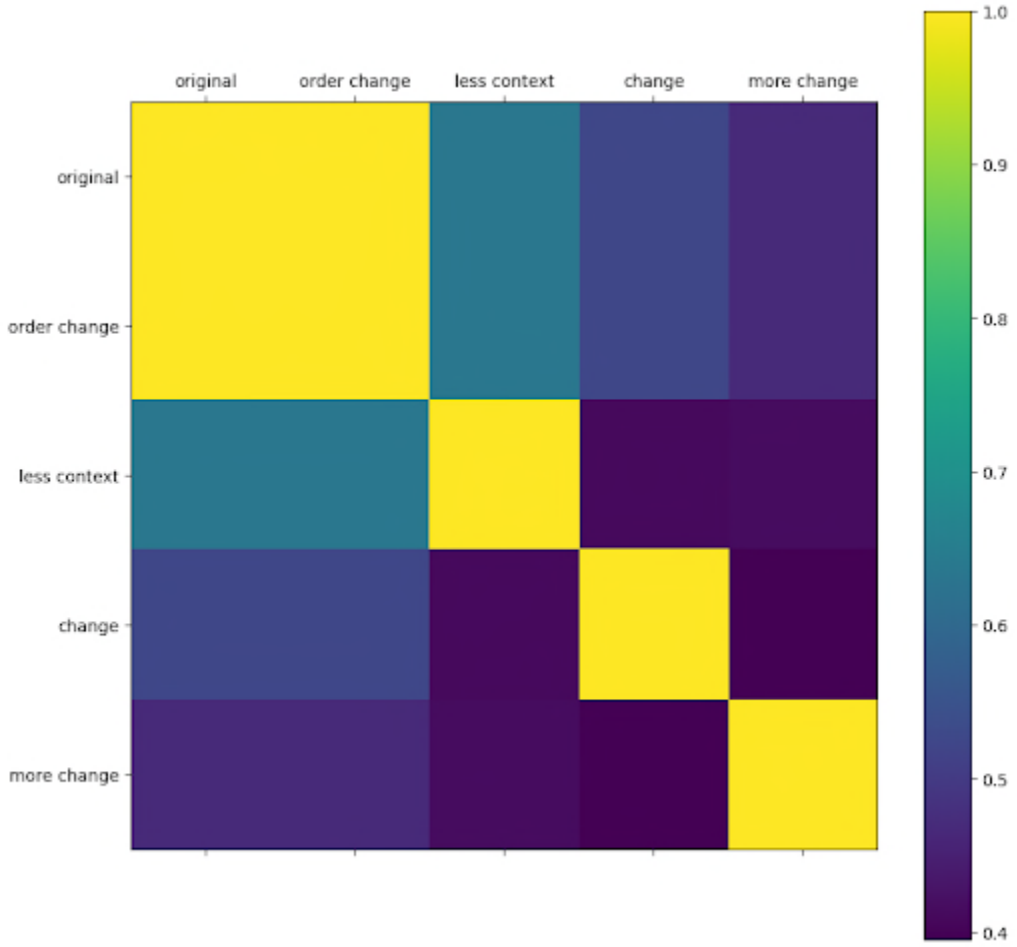

Fig. 9: Embedding similarity matrix for different scenarios.

For demonstration, we plot the similarity matrix for 5 hypothetical scenarios under method 1 (Fig 4). The individual scenarios are listed below:

- Original: ['Ambient temperature is 22 degrees.', 'Ambient brightness is 5000 lux.', 'Ambient noise level is 70 dB.', 'User goal is road trip.', 'User activity is driving.', 'User location is outdoor.']
- Order_change: original context with sentences in random order.
- Less_context: original context without the sentence 'User location is outdoor.'.
- Change: 'Ambient temperature is 22 degrees.' -> 'Ambient temperature is -3 degrees.'
- More_chnage: 'User activity is driving.' -> 'User activity is none.'; 'User location is outdoor.' -> 'User location is indoor.'

As shown above, the original scenario has exactly the same embedding as the scenario with order change, which is intentional since the order of sentences might change each time during contextualization. The similarity in embeddings correlates to the similarity in scenarios.

Chapter 5: Conclusions and Future Research Plans

The project presented the principles of a new solution in a proactive ambient system. To suit the needs of the new proactive software framework, we proposed a contextual recommendation system and a communication protocol. The new recommender system utilizes a text semantic encoder to enable continual learning and tackle challenges in contextualization, personalization and explainability. The new protocol requires proactive services to specify a set of proposal conditions and proposed applications.

In the future, we wish to investigate a faster and more accurate model for the text encoder. As a common technique in the recommendation model, one might also investigate the use of a knowledge base to expand the description of application and context.

References

[1] Emiliani, P.L. & Stephanidis, Constantine. (2005). Universal access to ambient intelligence environments: Opportunities and challenges for people with disabilities. IBM Systems Journal. 44. 605 - 619. 10.1147/sj.443.0605.

[2] Aarts, Emile & Ruyter, Boris. (2009). New research perspectives on Ambient Intelligence. JAISE. 1. 5-14. 10.3233/AIS-2009-0001.

[3] Younes, Walid & Adreit, Francoise & Trouilhet, Sylvie & Arcangeli, Jean-Paul. (2020). Agent-mediated application emergence through reinforcement learning from user feedback. 3-8. 10.1109/WETICE49692.2020.00009.

[4] Bobadilla, Jesus & Ortega, Fernando & Hernando, A. & Gutiérrez, A.. (2013). Recommender systems survey. Knowledge-Based Systems. 46. 109–132. 10.1016/j.knosys.2013.03.012.

[5] Su, Xiaoyuan & Khoshgoftaar, Taghi. (2009). A Survey of Collaborative Filtering Techniques. Adv. Artificial Intelligence. 2009. 10.1155/2009/421425.

[6] Carrer-Neto, Walter & Hernández-Alcaraz, María & Valencia-García, Rafael & Garcia-Sanchez, Francisco. (2012). Social knowledge-based recommender system. Application to the movies domain. Expert Systems with Applications. 39. 10990–11000. 10.1016/j.eswa.2012.03.025.

[7] Núñez Valdez, Edward & Cueva Lovelle, Juan & Sanjuán, Oscar & García Díaz, Vicente & Pablos, Patricia & Marín, Carlos. (2012). Implicit feedback techniques on recommender systems applied to electronic books. Computers in Human Behavior. 28. 1186–1193. 10.1016/j.chb.2012.02.001.

[8] Lee, Seok Kee & Cho, Yoon & Kim, Soung. (2010). Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. Information Sciences. 180. 2142-2155. 10.1016/j.ins.2010.02.004.

[9] Meteren, Robin. (2000). Using Content-Based Filtering for Recommendation.

[10] Basilico, Justin & Hofmann, Thomas. (2004). Unifying Collaborative and Content-Based Filtering. Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004. 10.1145/1015330.1015394.

[11] Pazzani, Michael. (1998). A Framework for Collaborative, Content-Based and Demographic Filtering. Artificial Intelligence Review. 13. 10.1023/A:1006544522159.
[12] Ramos, C. (2007). Ambient Intelligence – A State of the Art from Artificial Intelligence Perspective. In: Neves, J., Santos, M.F., Machado, J.M. (eds) Progress in Artificial Intelligence. EPIA 2007. Lecture Notes in Computer Science(), vol 4874. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-77002-2_24
[13] Adomavicius, Gediminas & Mobasher, Bamshad & Ricci, Francesco & Tuzhilin, Alexander. (2011). Context-Aware Recommender Systems. AI Magazine. 32. 67-80. 10.1609/aimag.v32i3.2364.
[14] Chen, Annie. (2005). Context-aware collaborative filtering system: Predicting the user's preferences in ubiquitous computing. Proceedings of 2005 Computer-Human Interaction (CHI'05). 1110-1111. 10.1145/1056808.1056836.
[15] Adomavicius, Gediminas & Sankaranarayanan, Ramesh & Sen, Shahana & Tuzhilin, Alexander. (2005). Incorporating contextual information in recommender systems using a multidimensional approach. ACM Trans. Inf. Syst.. 23. 103-145. 10.1145/1055709.1055714.
[16] Song, Linqi & Tekin, Cem & Schaar, Mihaela. (2015). Online Learning in Large-Scale Contextual Recommender Systems. IEEE Transactions on Services Computing. 9. 1-1. 10.1109/TSC.2014.2365795.
[17] Livne, Amit & Shem Tov, Eliad & Solomon, Adir & Elyasaf, Achiya & Shapira, Bracha & Rokach, Lior. (2021). Evolving context-aware recommender systems with users in mind. Expert Systems with Applications. 189. 116042. 10.1016/j.eswa.2021.116042.
[18] S. Sae-Ueng, S. Pinyapong, A. Ogino and T. Kato, "Personalized Shopping Assistance Service at Ubiquitous Shop Space," 22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008), 2008, pp. 838-843, doi: 10.1109/WAINA.2008.287.
[19] R. Bulander, M. Decker, G. Schiefer and B. Kolmel, "Comparison of Different Approaches for Mobile Advertising," Second IEEE International Workshop on Mobile Commerce and Services, 2005, pp. 174-182, doi: 10.1109/WMCS.2005.8.

[20] Devlin, Jacob & Chang, Ming-Wei & Lee, Kenton & Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[21] Minaee, Shervin & Kalchbrenner, Nal & Cambria, Erik & Nikzad Khasmakhi, Narjes & Asgari-Chenaghlu, Meysam & Gao, Jianfeng. (2021). Deep Learning--based Text Classification: A Comprehensive Review. ACM Computing Surveys. 54. 1-40. 10.1145/3439726.

[22] Reimers, Nils & Gurevych, Iryna. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. 3973-3983. 10.18653/v1/D19-1410.

[23] Conneau, Alexis & Kiela, Douwe & Schwenk, Holger & Barrault, Loïc & Bordes, Antoine. (2017). Supervised Learning of Universal Sentence Representations from Natural Language Inference Data.

[24] MacCartney, Bill & Manning, Christopher. (2008). Modeling Semantic Containment and Exclusion in Natural Language Inference.. 1. 521-528. 10.3115/1599081.1599147.