TIEVS – CAR CLASSIFIED RECOMMENDATION SYSTEM USING NOVEL APPLIANCES

R. A. D. Prathapa

(IT18122060)

BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021

TIEVS – CAR CLASSIFIED RECOMMENDATION SYSTEM USING NOVEL APPLIANCES

R. A. D. Prathapa

(IT18122060)

Dissertation Submitted in Partial Fulfillment of the Requirements for the BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021

DECLARATION

I declare that this is my own work, and this dissertation does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any other university or institute of higher learning, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology the nonexclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
R.A.D Prathapa	IT18122060	[illegible]

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor: Date:

Abstract

This research aimed to develop an optimized recommendation system to provide the customers with some convenience when extravagating through the application for preferences promptly. Existing classified platforms have recommenders induced, though not enhanced meticulously using sophisticated technologies. Previous scientific research has not targeted assessing the recommendation functionality within classified advertising applications though other types of recommenders have been investigated. However, significant issues within their collaborative-filtering systems prevail. Hence this study selected content-based filtering as a basis while eliminating user information consumption. Real-time recommendation with concurrency, which hasn't been thoroughly explored before, was implemented from this analysis by incorporating a Google USE model and a Golang API as a novel concept that is rarely exercised. The pre-trained TensorFlow model handled the classified ad description content text embedding generation while the cosine similarity calculation managed the top five similar item identification. The Golang API handled two main tasks: promoting embedding creation for new ads and the similarity analysis. Thus, the concurrency and real-time recommendation, which was a research gap, was fulfilled. The amalgamation of the USE model and the API was successful with a 926ms response time and an error rate of just 0.43% for 50 concurrent user sessions in load testing. The API and model were finally integrated with the front-end application, and all were deployed in DigitalOcean cloud droplet to facilitate availability and scalability. PostgreSQL database was used for fast database data fetching. In conclusion, the objectives of this research were rigorously attained by instigating an efficient and cost-effective advertisement recommendation engine to support the proposition of amplifying classified suggestions to customers. Creating a hybrid recommender, using the transformer architecture for the Google USE model, and accommodating multiple language classifieds can be indicated as possible future work for this research component.

Key Words – Concurrency, Cosine Similarity, Golang, Google USE, Optimized Classified Advertising, Real-time Car Recommendations

ACKNOWLEDGEMENT

Ms. Manori Gamage (Lateral Entry Coordinator, Senior Lecturer - Faculty of Computing) and co-supervisor Suriyaa Kumari (Lecturer – Faculty of Computing) are sincerely appreciated for their consistent, generous help, guidance, advice, recommendations, and inspiration across the whole research. The comments and suggestions provided were extremely valuable and effective in achieving the research's specified goals and expectations and ensuring a smooth pathway to completion. The supervisors' desire and devotion for both small and large-scale research processes encouraged the implementation of caliber and efficient techniques and the development of research competence on par with that of other competitors, which would have been inconceivable otherwise.

Secondly, Ms. Wasanthi Bandaranayake deserves special thanks for sharing her experienced benefits and challenges with online ad marketplaces, as well as putative advancements that should have been implemented towards a more modern technical element. Her proposals and ideas brought up a new study opportunity that has yet to be examined and analyzed in the current literature in the domains of information technology and business.

Furthermore, humble thanks are extended towards Mr. Austin Reese and the whole Kaggle team for developing and maintaining the necessary analysis-foundational dataset containing the classified advertised used car records from Craiglist, which was openly available for effortless utilization of Natural Language Processing and Machine Learning models that were incorporated in this research.

Additionally, parents are appraised to express emotional gratitude and respect for their perseverance, exertion, and time invested in procuring vital supplies and clearing other unexpected expenses. Subsequently, the research team members' skills and talents in creating a supportive environment at each timeline in this dissertation are indeed cherished.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

Abbreviation	**Description**
API	Application Programming Interface
DAN	Deep Averaging Network
IaaS	Infrastructure as a Service
JAM Stack	JavaScript, API, Markup Stack
JSX	JavaScript XML
ML	Machine Learning
TF-IDF	Term Frequency–Inverse Document Frequency
UI	User Interface
URL	Uniform Resource Locator
USE	Universal Sentence Encoder

1 INTRODUCTION

1.1 Background & Literature Survey

Classified ads are a market that allows users to sell their used/unused items or service by posting advertisements. The market is quickly gaining traction for online classified ads. People have been less eager to utilize traditional classified advertising methodologies like newspapers; hence the recent trend and popularity of online platforms providing the customers the necessary satisfaction, only more improved than the old ways of posting or viewing classifieds of various types of products. Newspaper revenues from advertising classifications are continuously declining as Internet classifications are growing, which is being depicted in Fig. 1.1[1].

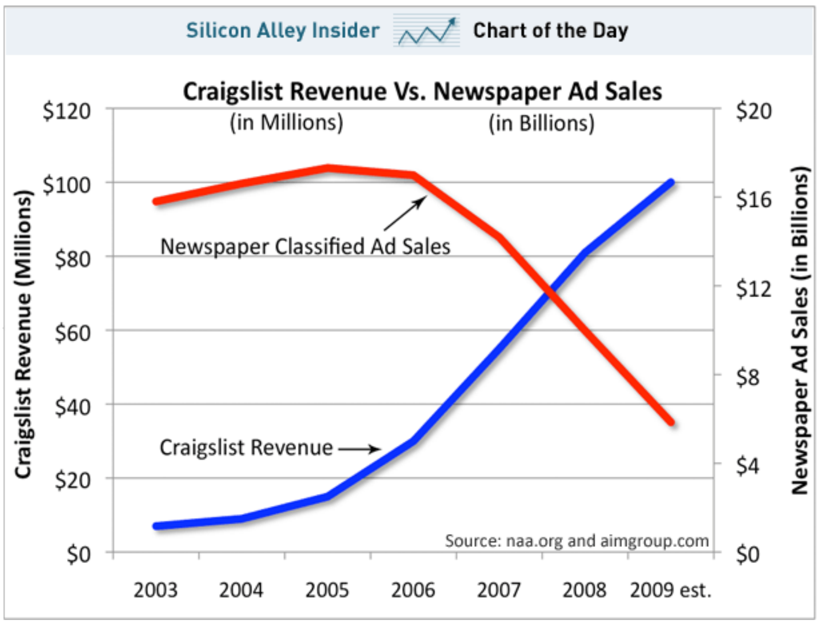


Figure 1.1 – Craigslist Revenue Vs. Newspaper Ad Sales [1]

The benefits of having an online platform for classifieds can see as below, and the customer is the one who seeks out the classifieds when they need a particular product. Potential customers will be able to search for the item or service all the time. It is effortless to search the relevant well-sorted product on the web platform rather than searching in the newspaper. Another benefit is that the user can upload images videos about the item to increase the advertisement's reach [2]. As shown in Fig. 1.2, in the UK, more people prefer online websites rather than going directly to a car dealer when they need to buy a used car. Due to many benefits, as Fig. 1.3 shows, from 2005 to 2009, the use of online classified ads sites has been doubled.

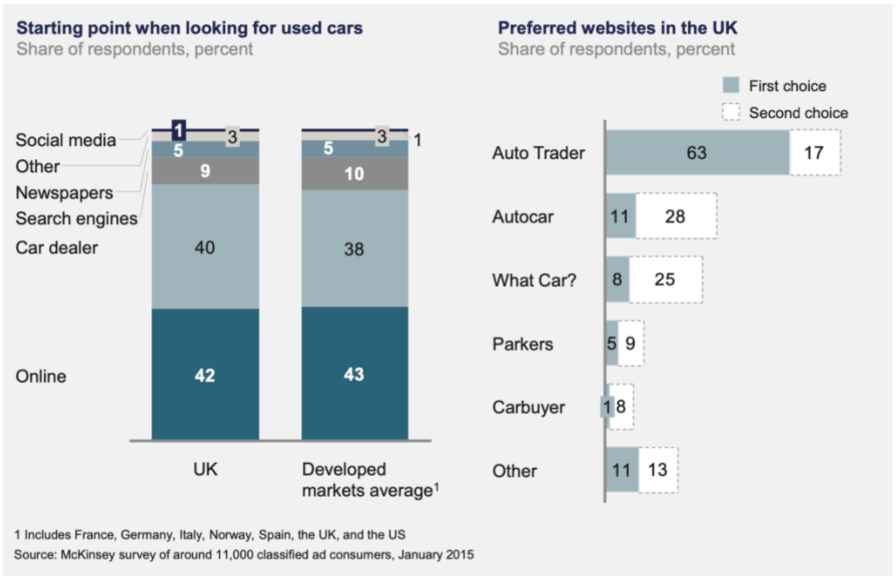


Figure 1.2 – Starting point when looking for used cars [3]

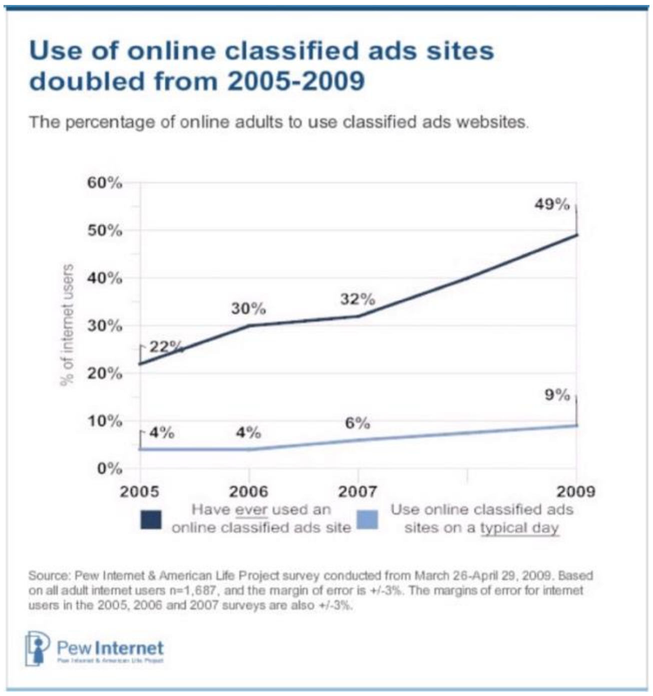


Figure 1.3 – Use of online classified ads sites doubled from 2005 -2009 [4]

With the World Wide Web's growth, the website's complexity and size are also on the rise. Therefore, it is challenging, complicated, and time-consuming to find information, data, or elements on those websites. When the domain is quite large, it is difficult for users to make an appropriate decision. In this regard, a recommender system provides the users with a refined list of alternatives tailored to their preferences. With a recommendation system, the website can recommend the items that the users have an interest in and the items they are searching for. The recommender system can predict whether a particular user would prefer an item or not So that the recommender system can deliver relevant items to the customer. Customers will be more engaged with the site when personalized product recommendations are made, thus boosting communication between them and the system. These exciting items may be any textual information, and they may also be an index on a particular topic that the user is searching for. A website can be recognized as a compilation of the elements involved. Within a recommendation system, we have an item and a user. To recommend the product to the customer, recommend system requires specific parameters like rating or item attributes. Competition is growing day by day among online retailers, and recommendation systems are a way of personalizing a person's needs. Recommendation systems are like a salesperson in a web store [5].

They are mainly five types of recommendation systems, as Fig. 1.4 depicts,

Content-Based: This system recommends items that are similar to other items that the user has liked in the past. The similarity of the items is determined on the basis of the characteristics associated with the items being compared.

Collaborative Filtering: This system is called as "people-to-people correlation." Collaborative filtering is considered to be the most popular and widely implemented technique in the recommendation system. It works on Neighborhood methods, which are focused on relationships between items or between users.

Demographic: This system recommends items based on the demographic profile of the user.

Knowledge-Based: This system recommends items based on specific domain knowledge about how certain item features meet users' needs and preferences of the user. This system will work better than others at the beginning. Nevertheless, if they are not fully equipped with learning components, then they may fail.

Hybrid: Combine one of two systems to a specific industry, for example, content-based and collaborative filtering.

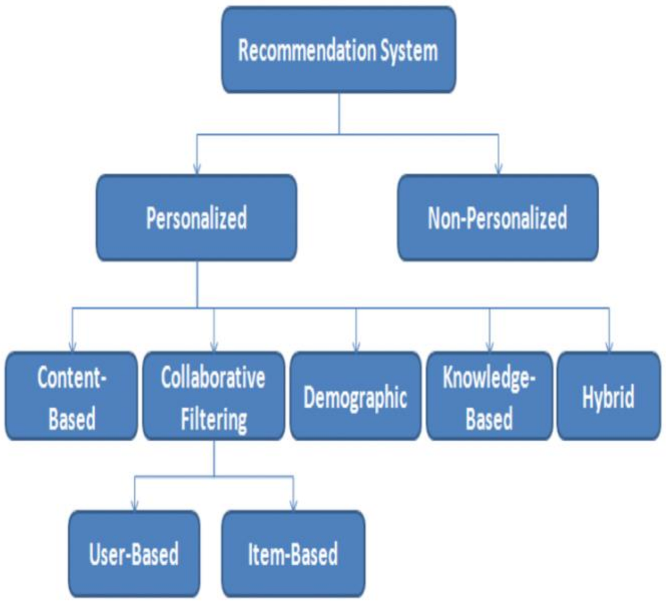


Figure 1.4 – Types of Recommendation System

In recent years, many forms of study have been carried out for recommendation systems. They have used different algorithms and different types of recommendation concepts such as content-based, collaborative, hybrid methods. There are many types of research for a system similar to ones facilitating movies, social networks, music recommenders, e-commerce related services.

R. Singla, S. Gupta et al. have done FLEX, a content-based movie recommender system [6]. They have used Distributed Bag of Word's version of Paragraph Vector (PV-DBOW), Term Frequency algorithms for the recommendation engine. With this, they have analyzed movie plots/ descriptions and generate an inter-movie similarity score. This method is inspired by the 'Paragraph Vectors' doc2vec model approach.

Another research has been done by A. Pal et al. regarding an improved content-based collaborative filtering algorithm for movie recommendations [7]. They used a modified hybrid approach that includes content-based and collaborative-based approaches. Their dataset contains a total number of 10004 ratings, 9125 movies, and 671 users. As in Fig. 1.5, tags and genre take into considerations. First, various users' tags for and movies are used in methodology and translated into a single list. The genres for each movie are appended to the same list of tags. This final list is referred to as the objects for a particular movie. The object set for each active movie is compared to the object set for each other in the dataset, and the number of matching objects is assigned to the set. Then using a formula similar movie list is formed.

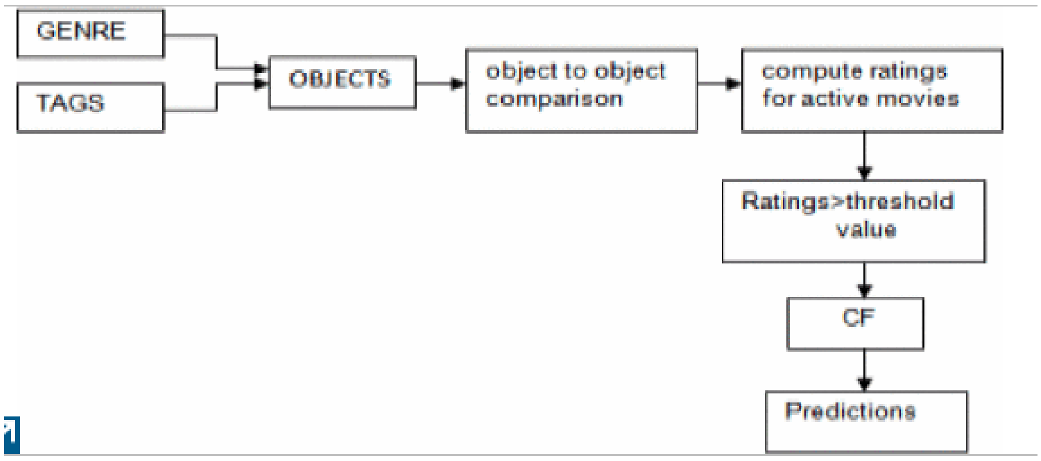

GENRE
TAGS
OBJECTS
object to object comparison
compute ratings for active movies
Ratings>threshold value
CF
Predictions

Figure 1.5 – Flowchart of hybrid collaborative filtering algorithm [7]

According to the author, B. Kostek, the challenge in search solutions lies in their scalability. In his research paper [8], he mentioned music recommendations using collaborative filtering. When one user listens to preferred music, the system will find other users with similar music tastes then the system will recommend the first user with new songs that he/she might like. User, Item, and Rating are the basis of the collaborative filtering. In Fig. 1.6, green lines show the user's choice of songs, and a red arc shows whether two users are interconnected by a given song, with the violet dashed line show the recommendation.

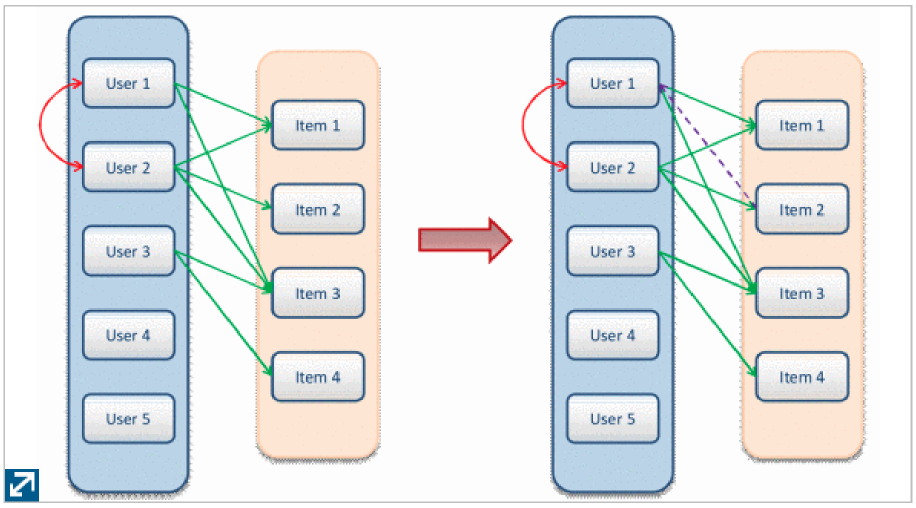


Figure 1.6 – User-based music recommendation [8]

Author S. Shaikh has used a graph-based approach for a recommendation system in e-commerce [9]. This proposed system uses an overlap semantic method to integrate recommendations and semantics. Data will be stored in a graph format, and the Overlap value will be calculated using the overlap formula. Only items that are considered for a recommendation have a correlation value of more than 0.4. The process is shown as an outline in the below Fig. 1.7.

$$\text{Overlap} = \frac{N(p \cap q)}{\min(N(P), N(q))} \quad ($$

▸ View Source

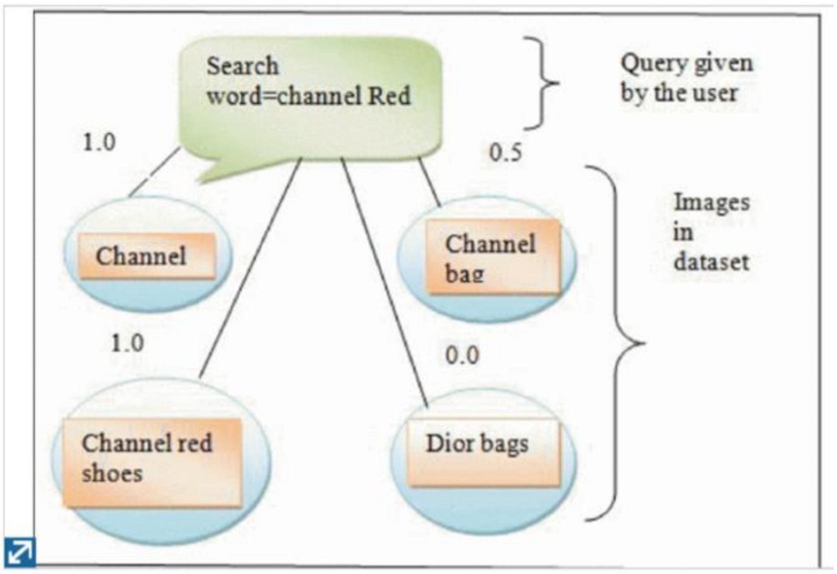


Figure 1.7 – Overlap formula and working model [9]

The authors Prabowol et al. [10] have utilized the collaborative filtering technique to implement their recommendation engine, and as a result of that, they had to use user-based ratings for their recommendation functionality as well.

The same approach has been abducted by the authors' Sun et al. [11]. However, them using customer ratings for model implementation induces the Cold Start problem straight up and naturally. Their model is not able to account for the items which do not have their own ratings being attached. The recommendation is happing without those specific items having little to no ratings. That is something that should not happen from a quality recommendation engine. On the other hand, another study was conducted by Wita et al. [12] by basing and utilizing a graph database called Neo4j to facilitate recommendations using the basic methodology of content-based filtering. Document profiles were used in their method, and two types of keyword extracting mechanisms were used to produce those profiles as well. They were namely term frequency-inverse document frequency (TF-IDF) and Textrank. Those two were responsible for document profile creation and evaluation. The accuracy of their recommendation engine was stated as 0.77. At the end of their study, for the most similar item identification, cosine similarity calculation was used. Content-based filtering was incorporate hence removing the Cold Start problem altogether. They have not analyzed the collaborative filtering aspect; hence user-related details such as selections and feedbacks were not stored for any process executions. Nonetheless, if they were to use a hybrid recommendation engine, which combines both the content-based and collaborative filtering methods, they would gain a more robust and formidable engine. To improve their hybrid model implementation probability, the graph database could be used to act as a provocation. Besides, this study provided an immense amount of inspiration when designing and developing this research’s own system overview and methodology steps due to their practicality of the procedures and the high-quality end results, and simplicity in model implementations and database utilizations.

Another research by Xue et al. [13] attempted to indicate the significance of content-based filtering, along with the incorporation of numerous social networks. Their calculation of the text similarities consisted of both short and long texts procured by a new distance. For that purpose, the word2vec method was investigated further. Their next step was to create a new model that is capable of finding the closest neighbor collection by analyzing the respective users' own social networks swiftly and appropriately. The texts that had been read by the nearest neighbors to the user are then examined and supplied by the model [13] have been built as a recommendation. Although this study's model performance is commendable, the utilization of social networking still impacts towards a Cold Start Problem and a possible customer data and information unauthorized disclosing. Customer data being revealed through an unethical procedure may promote various legal penalties or fines to be bestowed upon whichever application using the recommendation engine by necessary law enforcement.

A fully content-based movie recommendation system was proposed by authors Chen et al. [14]. Their dataset was acquired by the infamous portal called movielens, and it consisted of 27,000 movies and 20 million user ratings, which strongly indicated the strength and depth of the dataset chosen. Their experimentations should therefore be greatly acknowledged and appraised since a highly complex dataset is being used for model implementation purposes. Topic words were extracted for their model, which is named as LDA topic model, from word segmentation results. It is stated that their recommendation model was trained using a neural network model, and the way context words were predicted was using the continuous bag of words (CBOW) technique. That technique is compulsive for its ability to predict content information and account for linear relationships for the specific learning feature. In order to calculate the similarities of the generated word vectors through the word2vec method, cosine similarity calculation was used again. User details such as behaviors, history, bookmarked or liked texts are being considered and stored for recommendation purposes which are not acceptable or convenient for both recommendation models implied application functioning and hardware resource consumption. Regardless, none of the research above [6]-[14] has paid attention to building recommendation engines targeting online classified advertising portals, which is why this study has opted for the task while mainly focusing on novel technology embodiment.

1.2 Research Gap

There are many types of research for a system like a movie, social networks, music recommenders, e-commerce related systems. Still, there is not enough research for classified ad recommendations, let alone car advertisements. According to the information that we can extract from other research, the following attributes to build the recommendation have been considered.

- votes
- likes/dislikes
- ratings
- views
- feedbacks
- comments
- reviews

According to the available research papers [7]-[10],[13], they required user preference attributes as above. Those attributes may not be included in a classified advertising application almost in many scenarios; hence the monitoring for such attributes is in vain. Besides, it creates the Cold Start problem, which is information unavailability for the recommendation engine to process the newly added components. The proposed system from this research would not be accounting for such attributes, except for advertisement content-based features, to avoid the Cold Start problem.

Most of the analyzed research [6]-[13], other than [14], have not considered a sufficient number of records included dataset, having content characteristics, to train and test their relevant models for the recommendation, although existing classified advertising platforms in market monthly accepts more than 200,000 advertisements. Hence the usage of a large dataset is more convenient since the learning range will then be expanded further, which the proposed recommendation system targeted to achieve.

Not to mention, the examined research [6]-[14] has not exercised the idea of a combination of the Google USE model for embedding generation and cosine similarity algorithm types to experiment with the accuracy levels and overall performance. The USE model is a pre-trained model coming from the comprehensive Python library TensorFlow. It is a model that is thoroughly pre-trained for textual data acquired from different websites such as Wikipedia, question and answering forums, review forums, and other verbose sites. Therefore, it is not wrong to address that model as a matured model which needs minimum re-training and preprocessing of raw data when being imported as a model to be used under various Machine Learning (ML) domains. The prevailing research done on behalf of it is scarce as well. Hence a huge gap is formed that needs to be fulfilled promptly through the successful execution of this research.

Moreover, the real-time ad recommendation procedures have not been discussed openly or sophisticated in any of the discussed studies [6]-[14]. Regardless, real-time recommendations are a must, especially when considering classified advertising portals, since every manipulation on an existing advertisement or even a whole new submitted classified ad must be gauged and evaluated to provide the end-users with the best advertising experience when browsing through the classified ad application. Thus, this research intended to provide a solution for the above-mentioned gap expanding issue through a concurrency enabled Application Programming Interface (API) implementation. The programming language which can satisfy the requirement conveniently had to be investigated first of all.

Furthermore, some of the past research [8, 9, 14] did not observe the scalability feature that a recommendation engine should have to accommodate scaling levels of network traffic arising from parallel user sessions. The recommendation model and application deployed cloud environment must be adequately equipped and designed for that matter. A client-side graphical application should always be merged with the implemented recommender model and supplementary API though it was not to be seen from the studies analyzed [6]-[14]. Both technical and business requirements should be dealt with by desirable and worthy research.

Hence, the proposed system will be using the most suitable and optimized content-based recommendation, having a combination of models, for the classification of classified ad preferences for each customer. Table 1.1 shows the summarization of the key comparative features emphasized from the above explanation.

Table 1.1 - Comparison of former research

Research	**Optimized for classifieds ads**	**Higher Scalability**	**Realtime Recommend of Classifieds**	**Avoid Cold Start problem**	**Usage of Google USE & Cosine Similarity**
Research A [6]	✗	✓	✗	✗	✗
Research B [7]	✗	✓	✗	✗	✗
Research C [8]	✗	✗	✗	✗	✗
Research D [9]	✗	✗	✗	✓	✗
Research E [10]	✗	✓	✗	✗	✗
Research F [11]	✗	✓	✗	✓	✗
Research G [12]	✗	✓	✓	✓	✗
Research H [13]	✗	✓	✗	✓	✗
Research I [14]	✗	✗	✗	✓	✗
Proposed System for Classifieds	✓	✓	✓	✓	✓

1.3 Research Problem

Several recommendation algorithms have been developed throughout the recent years regarding movies, videos, images, economic blogs, etc. However, the need for an appropriate recommendation system with a significant level of accuracy targeted towards classified advertising hasn't been conveniently fulfilled as of yet. Moreover, most of the implemented recommendation systems need user browsing history, clicks, likes, dislikes, comments, feedback, reviews, and other user-specific preferences for suggesting the customers with their necessary selections. These attributes are causing an exasperating issue known as the 'Cold Start Problem' in the recommendation engines that uses the above attributes. Basically, what it means by the Cold Start Problem is when a new item or an advertisement is created in the system, it does not have enough data to process within the recommendation engine; therefore, the new item will not be displayed in the recommendation section of the subsequent ad clicking requests by another user from a different session. A sufficient amount of research hasn't been conducted to address the above-stated problem [3]-[11] and is yet to be rectified.

Furthermore, the customers rarely give ratings, comments, or feedback on classified advertisement platforms, hence considering those attributes for recommendation purposes is entirely unavailing and the probability of inaccurate predicted suggestions will simultaneously increase as well. Then the next problem occurs when finding what kind of content attributes should be considered in order to provide those customers with personalized preferences in a more efficient way. In addition, algorithm training, with the use of a dataset having a large number of records, such as 450,000, has never been attempted in previously analyzed research [3]-[7]. In general, approximately 500,000 new classifieds are being submitted into main classified advertising platforms such as Craigslist, Olx, etc. thus, training a recommendation algorithm model based on a limited amount of record set is ineffective and problematic. To improve the accuracy further, more and more information needs to be infused into the models. Besides, it's more challenging to identify the most impactful features for car ad recommendation and for the training algorithm models themselves as well.

Moreover, the utilization of the Google USE pre-trained model found within the TensorFlow python library has not been exercised during most of the major studies done up until now [3]-[11]. It may be due to the fact that the model itself is quite new even for the overall Information Technology domain, despite being incorporated within the recommendation field of classified advertising. USE is responsible for generating high dimension vectors for the purpose of clustering, similarity analysis, text classification, and other various Natural Language Processing related functionalities. Two types of encoder architectures provide the main structure for the model, and since it has been pre-trained by numerous websites, questions/answer forums, Wikipedia, and other similarly heavy verbose websites, the training time and effort required for the model is significantly low. Thereby, it promotes and encourages the main aim of which ever experimentation that it is being used for. Why such as talented model is not being investigated on advertisement recommendation, let alone other e-commerce product/item recommendation is still a mystery that urges to be solved.

Subsequently, comes the question regarding the combination of Google USE model and Cosine Similarity [15], which was never used for the procedure of product recommendation, in the former mentioned research [3]-[11], giving rise to an uncertain hypothesis, that a better performing, scalable and a robust algorithm model is yet to build. However, the feasibility of this combination seemed ambiguous at first glance since the USE model itself hasn’t been thoroughly analyzed and researched by a considerable number of experimenters, researchers, or tech enthusiasts. It was a challenging task to experiment and examine while sacrificing time and dedication towards an entirely novel concept. Nevertheless, it had to be accounted for and meticulously analyzed of the possibility of integration between the two mechanisms, as to provide the existing literature an innovative approach towards better recommendation procedures. Hence addressing the above-stated problematic hypothesis was one rumination of this proposed recommendation system.

Real-time ad recommendation is crucial in nowadays applications, especially when considering classified advertising portals. A sound knowledge or a deeper insight into what real-time recommendation exactly signifies has not to be investigated upon in prevailing prominent research done on implementing recommendation systems [3]-[11]. A real-time recommendation is concerned with providing a user-centered advertisement or any kind of a post suggestion mechanism that accounts for all or any update, delete manipulations done on those advertisements/posts. For example, in a classified advertising portal, if a customer were to edit or delete their own advertisement and save the necessary changes afterward, those changes would not be considered for the recommendation process instantly when another customer with another user session browse through the classifieds. This kind of scenario should never be acceptable for a recommender engine to tolerate under any circumstance, although such recommendation systems still exist. It is because all of those recommender engines are not capable of explicitly identifying and dominating the said aspect using an innovative way of handling. Hence, this is defiantly an issue that needs to be addressed as soon as possible. Thus, this research opted for the challenge in hopes of receiving positive outcomes.

The concurrency characteristic is another criticality that should conventionally be present within a well-composed recommender system, although that is not the case in many significant studies that have been concluded [3]-[11]. Through concurrency, a system will be able to handle multiple tasks simultaneously without its performance or response time being affected. When considering a recommendation system, it will need to handle such tasks and functionalities on a daily basis due to concurrent user sessions being populated. Even when speculating the two processes, which are advertisement submission and all classified ad recommendations, it is at least merely evident that concurrency is mandatory for the new advertisement to be eligible for the subsequent recommendation process after quickly being registered into the database. This feature cannot be implemented simply by utilizing Machine Learning (ML) appliances and would therefore need a different approach technologically, most preferably an API interference. Hence, another problem occurs on how to create simple, fast, and concurrent routines within an API for multi-tasking, and another inevitable matter would be the appropriate selection of the most suitable and proficient programming language for the demanding endeavor.

1.4 Research Objectives

Main Objective

The main objective of this particular proposed system, having an internally driven, optimized item recommendation process, is to visualize the best and most convenient preferred advertisement results for customers who are utilizing the 'Tievs' application. The recommendation engine was proposed to be executed with high precision and speed, so that customer satisfaction is uplifted to a commendable level, giving rise to application remarkability in utilizing new technologies for the purpose of improving usability and user-friendliness.

Without a thoroughly advanced and meticulous recommendation system, the necessary items that the customers might be wanting to explore would not be presented to them, making the customers search those specific needs individually, wasting time from their own busy schedules. As a result, the application as a whole will face a high customer turnover rate, giving rise to the probability of application reputation degradation. Nevertheless, much research has been done to address the above issue for applications containing movies, videos, images, social media content, etc., although only a few of them have targeted classified advertisement platforms. Many have used either content-based filtering types as singular algorithms or collaborative filtering through various versions of algorithms. Collaborative filtering requires customer input attributes such as likes, clicks, comments, feedback, etc., and in classified advertisement applications, the recommendation engine cannot entirely depend on them due to their scarcity, making the prediction process of preferred items for the customers less accurate. Therefore content-based filtering seems much convenient, regardless of not being investigated further in detail as to consider the novel implementations of it, targeting the improvement of the recommendation processes within classified advertisement platforms. Thus, the ulterior aim of this research was to provide the customers with an error-free, scalable, and reliable classified ad recommendation system, all the while utilizing the content-based approach in a novel context.

Specific Objectives

In order to reach the main objectives, the specific objectives that needed to be attained are as follows,

1. Usage of Google USE model from TensorFlow library for the purpose of embedding generation.

2. Implementation and amalgamation of both the USE model and the cosine similarity calculation.

3. Rectifying and finding a solution for the cold start problem in the domain of recommendation systems.

4. Solving the consequences of the cold start problem resolution, which are mainly considered with the speed of model execution.

5. Involving and constructing a concurrency mechanism using the most optimal programming language to manage the latency levels and increase routine speed.

6. Fast and synchronous fetching of the necessary data and information from the selected and most suitable database for the task.

7. Building a real-time recommendation providing mechanism includes an engine so that when ads are deleted or updated, those changes will be accommodated for recommendation instantly.

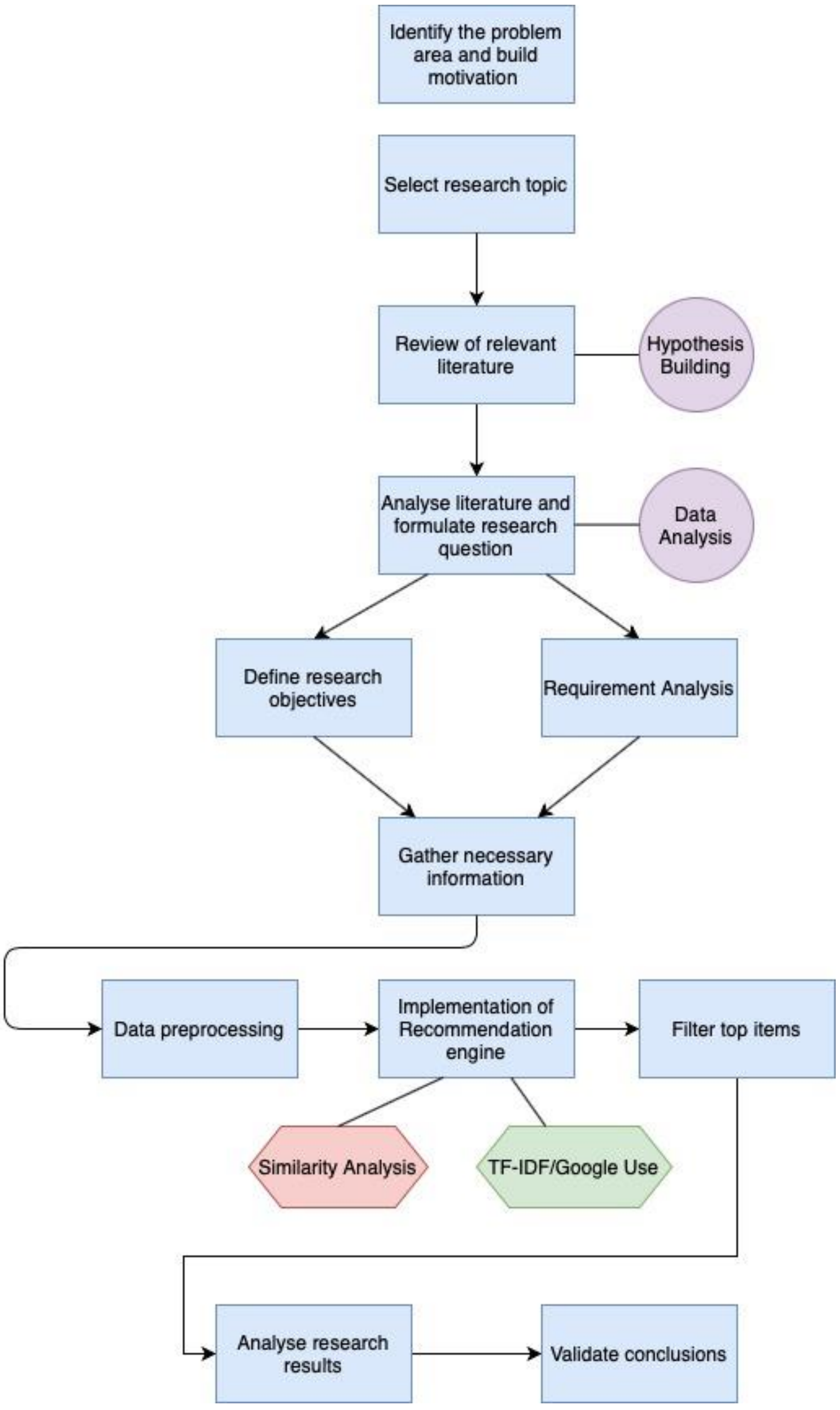
8. Lighting fast API implementation that facilitates both concurrency and real-time recommendation.

9. Cloud server deployment and availability endowment with scalability management of the model included API and the front-end application.

2 METHODOLOGY

2.1 Methodology

2.1.1 Research methodology



Identify the problem area and build motivation
Select research topic
Review of relevant literature
Hypothesis Building
Analyse literature and formulate research question
Data Analysis
Define research objectives
Requirement Analysis
Gather necessary information
Data preprocessing
Implementation of Recommendation engine
Filter top items
Similarity Analysis
TF-IDF/Google Use
Analyse research results
Validate conclusions

Figure 2.1 – Research Methodology

The above-shown Fig. 2.1 is the followed overall research methodology. First of all, the problem according to the research area is identified in order to build the surrounding motivation towards it. After it was decided that the main research should be focused on enhancing the functionalities provided by a classified advertising system, it was the purpose of this research component to investigate how the existing recommendation engine structures can also be further improved. That itself is the chosen research topic for this component. The next was to examined the prevailing literature and to build up a hypothesis. Collaborative, content-based, and hybrid recommendations were the most common recommendation engine structures. During that analysis, issues were found, such as the cold start problem, in which the newly submitted classified ads are not having user-related details like user rating, likes, comments, feedback, and other preferences. If that is the case, collaborative filtering would have insignificances present. Hence, the content-based filtering method showed higher desirability.

Moreover, existing systems were not trained using an ample amount of training records and did not incorporate novel technologies such as Google universal sentence encoder and Go language implemented API and concurrency routines. Most systems were not even high in scalability, flexibility, accuracy, and speed of execution. Thus, a hypothesis was built that if a content referenced recommendation engine was built with the amalgamation of innovative techniques and frameworks, the classified advertising application as a whole would be able to perform better with respect to advertisement suggestion, all the while infusing the most convenient and efficient mechanisms. After the hypothesis was established, further literature and data were reviewed in order to get a thorough understanding of how the implementation procedures and necessary pseudocodes are developed. Henceforth, the research question, which speculates on how to address the prevailing recommendation engine incompetencies, was build. According to the literature review, the main objectives of the research component were defined, and the project requirements were confirmed. Consequently, the data acquiring part eventuated.

Data gathering

The necessary dataset for implementing the proposed recommendation system was acquired from the online dataset available web portal Kaggle. The specific dataset that was used for this research is the ‘Used Cars Dataset’ uploaded as ‘Vehicles.csv’ by Austin Reese, which has been continuously updated monthly, starting from 2020 November. The data included in the dataset was scraped through the infamous online classified advertising web portal known as Craigslist by using a web scraper. The dataset contains about 441,396 records of sold cars with their relevant characteristics on the above-mentioned online platform.

Dataset Online Link - https://www.kaggle.com/austinreese/craigslist-carstrucks-data

One of the main reasons for selecting this particular dataset is a large number of records itself, since in order to achieve a more premium recommendation of car advertisements for the customers, the more amount of information to train the necessary algorithms, is the better. A US-based second-hand car dataset was used because Tievs business idea was initially born targeting the US online classified advertising market, according to the request by our project’s reference person as a requirement. Moreover, the need for manual data collection degrades since it is more time-consuming, considering the proposed scope of this research, and also, a large range of information cannot be attained through that as well.

Continuation of the research methodology

Succeeding the data preprocessing part of the acquired dataset, the recommendation engine implementation process commenced combining both Google USE and cosine similarity calculation from the TensorFlow python library. Through successful embedding creation and similarity calculation, top ads were filtered and visualized from the front-end Angular application. The end research outcomes regarding speed, scalability, and especially hardware resource consumption metrics were then analyzed and evaluated. With respect to those research results, conclusions and future work inspirations were formed as the final resolution.

2.1.2 High-level system overview

Fig. 2.2, shown below, is the overall high-level diagram of the proposed research component. The explanation of the diagram is stated below.

- The novel recommendation algorithm implemented using the TensorFlow imported pre-trained Google USE model is further trained with the incorporated training dataset. The description column is used for this task.

- The 512-vector embeddings are generated with regard to the description column included context and the newly produced results are stores in the database table with a new column called ‘embeddings.’

- The database system utilized for the storage of all tables containing the embeddings is PostgreSQL. It is a fast performing, highly expandable, cross-platform database that is appropriate for the considered complex dataset importing. The assistance for various language support and its open-source feature is commendable.

- The re-trained model is then serialized to be combined with the machine learning (ML) API implemented using the simple, efficient, and reliable open source programming language, which is Go. Go is useful for complex but swift software development, which is exactly what is needed for this research component’s objective attainability.

- The API implemented handles two major procedures, which are the new embedding creation and storage for the newly published advertisements and the concurrent similarity calculation. Both tasks are handled by the API parallelly; hence the infrastructure is built using robust Go language routines.

- The front-end application designed using the Angular framework is integrated with the Go API to provide the customers with the relevant preferred advertisements by filtering them all out quickly.

- When the customer clicks an ad(hyperlink) from the UI, the application calls the API, and it will request the ad details from the content management system, and also it will asynchronously request recommendations from the recommendation API.

- The recommendation API will request other ads in that category from the content management system, and current Ad details will be analyzed using the optimized algorithm for the classifieds ad. The top recommendations will be given to the user as a response to the API request.

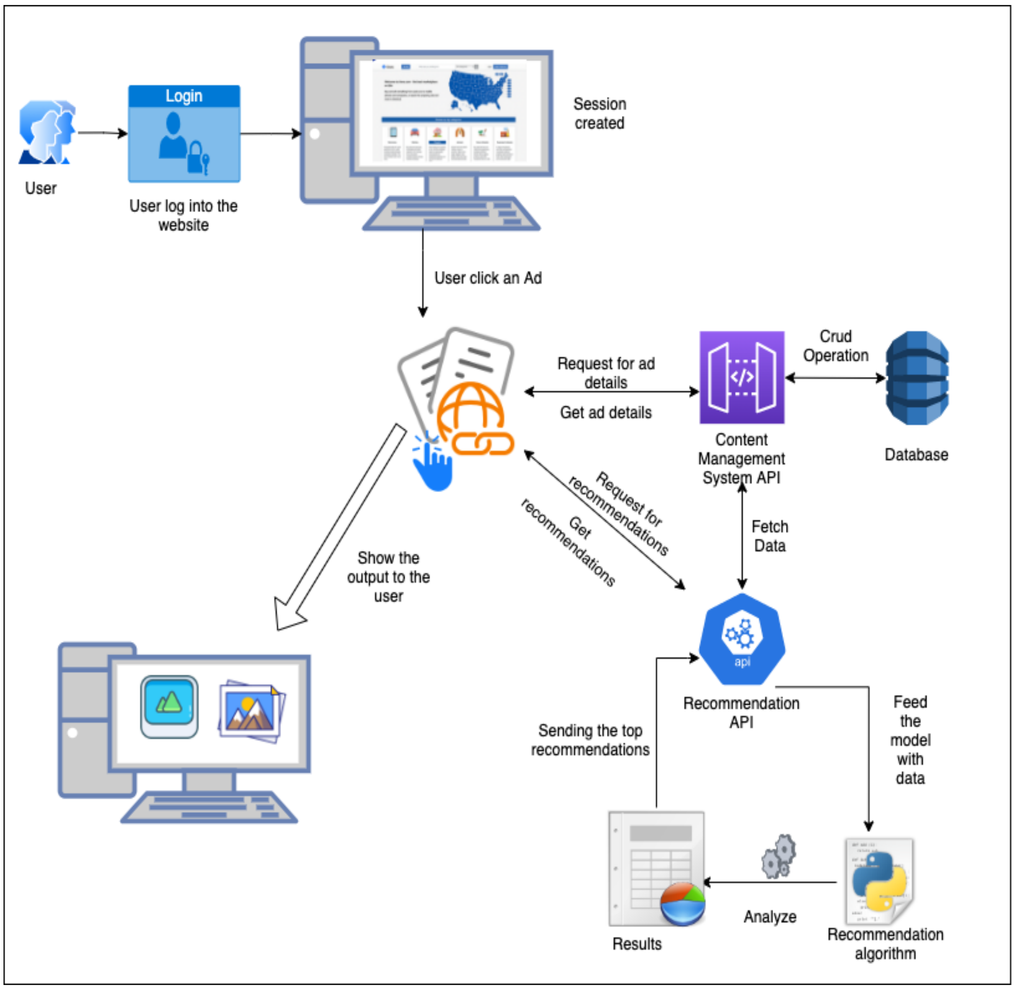


Figure 2.2 – System Overview Diagram

2.1.3 Software development life cycle

The life-cycle development of the software that would be considered was implemented with the agile methodology. In addition, under the vast Agile framework, Scrum methodology was mainly covered. Scrum is a lightweight framework that allows individuals, teams, and institutions to gain value through agile strategies for complex challenges. Product backlog creation, sprint planning and creation of sprint backlog, daily scrum meeting, sprint retrospectives, sprint review, and next sprint planning are the incremental but iterative basic steps that are being followed within a scrum approach followed. For this research component, the same procedures were effectively followed. Many unanticipated events occurred during this research, and the already accepted and established plan was not followed in the exact same way. Alterations were made to the plan when addressing the impediments in the research process. Thus, agile methodology is significant for the conveniency of the continuous and persistent flow of the implementation process. Agile mechanisms gave the research more control, better productivity, better quality, higher customer satisfaction, and higher return on investment as well [16].

2.1.4 Tools & technologies

Tools

- Jupyter Notebook
- PyCharm
- GoLand
- VSCode
- WebStorm
- Loadmill
- DigitalOcean/Supervisorctl
- Cloudflare
- Postman

Technologies

- Python (libraries)
- Golang echo framework
- TensorFlow
- Google USE
- Angular framework
- Typescript
- NodeJS
- PostgreSQL
- HTTP/2

2.1.5 Project Requirements

Functional requirements

- Users should be able to view the classified ads either as recommended or conventional advertisements without significant interruptions.

- Users should be able to access and click the classified advertisements without inconvenience.

- Signals should be sent to the recommendation API about the customer preferred classified from the client system.

- The recommendation system should select the top recommendation list from other overall recommendations.

- Rediscover new classifieds by the recommendation system each time they are added to the system.

- Concurrent embedding creation and similarity calculation without bottlenecks.

Non-Functional requirements

- Scalability
- Concurrency
- Reliability
- Accuracy
- Speed
- Optimized performance
- Load balancing

2.2 Commercialization Aspects of the Research Component

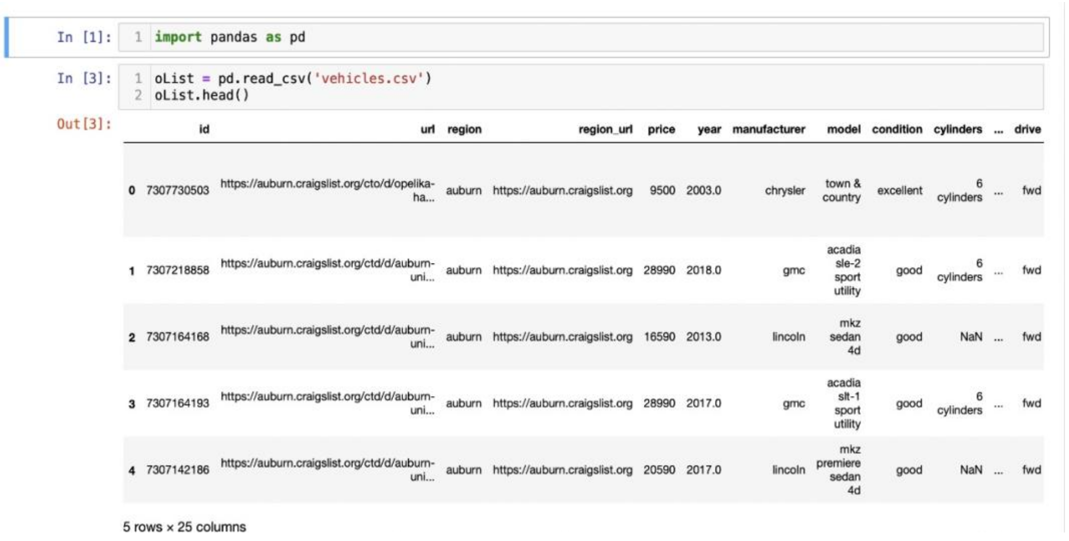
The skeptical speculation that implies the compulsory need for a capable recommendation engine has emerged during the initial research problem identification and topic definition clearing phase. However, certain factors were consequently identified which begged to differ. According to [17], compound annual growth rate (CAGR) has declared that the recommendation growth within the market will be expanded at a 39.8% rate within the year 2026, which can be seen as an influential statistic towards this research's successful development. The value is said to be an astonishing 16.14 billion USD. Therefore, the rudiment drive within e-commerce website developers and other stakeholders towards an efficient recommendation engine advances; hence it influenced this research component production as well.

Moreover, by using this ad-suggesting engine, startup companies who are pursuing e-commerce web app developments would gain benefits since this engine can be used even without comprehensive previously attained user data and information. Additionally, this infers that the cold start problem would be lifted and resolved since user likes, dislikes, comments, feedback, browsing history, ratings, and votes are not accustomed to within the recommendation engine structure. Popular e-commerce giants such as Facebook and Uber have been penalized by various law enforcement for misusing customer personal data for processes such as recommendation itself. In that sense, this constructed recommendation engine would not have similar complications since sensitive consumer data is not being stored for any underlying procedure. Furthermore, real-time recommendation of classifieds will be provided since when a customer submits a new ad, the API executes the new embedding creation related to that, and that ad will instantly be accounted for cosine similarity calculation while paving the way for the overall recommendation process for any other customer who is browsing through the ads. The Golang API handles concurrent tasks and is verified as highly scalable. The ML model and API deployed cloud infrastructure is highly cost-efficient, and the all-inclusive engine is easier to implement and maintain with a fast and scalable architecture.

2.3 Testing & Implementation

2.3.1 Dataset & data preprocessing

Below shown, Fig. 2.3 displays the first five rows of the dataset opened from Jupyter Notebook.



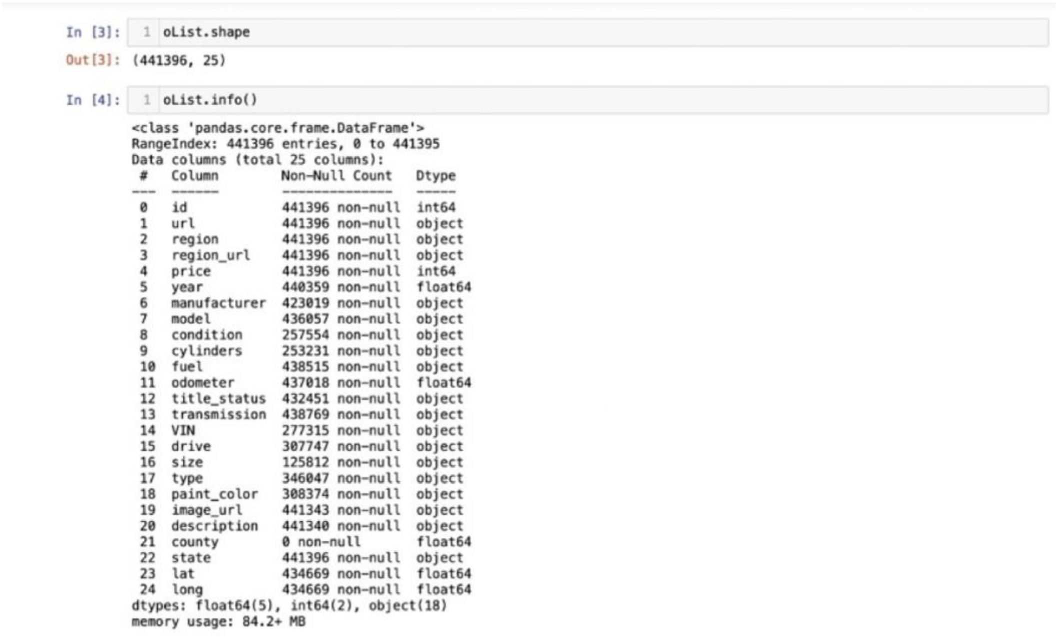
```
In [1]:  import pandas as pd

In [3]:  oList = pd.read_csv('vehicles.csv')
         oList.head()

Out[3]:
```

	id	url	region	region_url	price	year	manufacturer	model	condition	cylinders	...	drive
0	7307730503	https://auburn.craigslist.org/cto/d/opelika-ha...	auburn	https://auburn.craigslist.org	9500	2003.0	chrysler	town & country	excellent	6 cylinders	...	fwd
1	7307218858	https://auburn.craigslist.org/ctd/d/auburn-uni...	auburn	https://auburn.craigslist.org	28990	2018.0	gmc	acadia sle-2 sport utility	good	6 cylinders	...	fwd
2	7307164168	https://auburn.craigslist.org/ctd/d/auburn-uni...	auburn	https://auburn.craigslist.org	16590	2013.0	lincoln	mkz sedan 4d	good	NaN	...	fwd
3	7307164193	https://auburn.craigslist.org/ctd/d/auburn-uni...	auburn	https://auburn.craigslist.org	28990	2017.0	gmc	acadia slt-1 sport utility	good	6 cylinders	...	fwd
4	7307142186	https://auburn.craigslist.org/ctd/d/auburn-uni...	auburn	https://auburn.craigslist.org	20590	2017.0	lincoln	mkz premiere sedan 4d	good	NaN	...	fwd

5 rows × 25 columns

Figure 2.3 – Jupyter Notebook dataset first five-row view

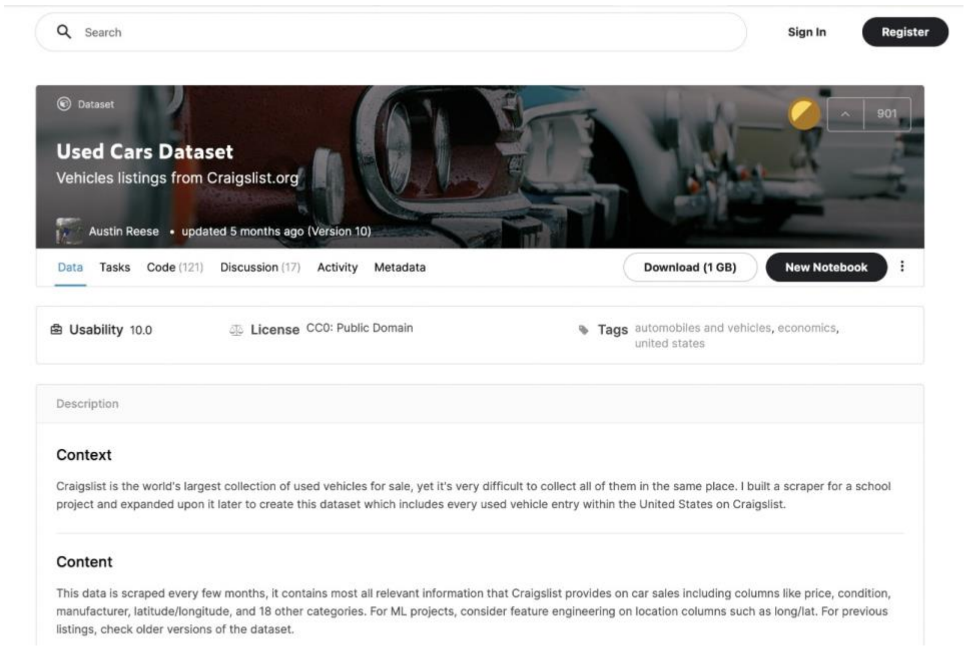
In addition, Figure 2.4 displays the Jupyter Notebook view of the columns and their specific types, which can be found from the dataset. The shape of the dataset is also shown along with it.



```
In [3]:  oList.shape
Out[3]:  (441396, 25)

In [4]:  oList.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 441396 entries, 0 to 441395
Data columns (total 25 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            441396 non-null  int64  
 1   url           441396 non-null  object 
 2   region        441396 non-null  object 
 3   region_url    441396 non-null  object 
 4   price         441396 non-null  int64  
 5   year          440359 non-null  float64
 6   manufacturer  423019 non-null  object 
 7   model         436057 non-null  object 
 8   condition     257554 non-null  object 
 9   cylinders     253231 non-null  object 
 10  fuel          438515 non-null  object 
 11  odometer      437018 non-null  float64
 12  title_status  432451 non-null  object 
 13  transmission  438769 non-null  object 
 14  VIN           277315 non-null  object 
 15  drive         307747 non-null  object 
 16  size          125812 non-null  object 
 17  type          346047 non-null  object 
 18  paint_color   308374 non-null  object 
 19  image_url     441343 non-null  object 
 20  description   441340 non-null  object 
 21  county        0 non-null       float64
 22  state         441396 non-null  object 
 23  lat           434669 non-null  float64
 24  long          434669 non-null  float64
dtypes: float64(5), int64(2), object(18)
memory usage: 84.2+ MB
```

Figure 2.4 – Jupyter Notebook view of columns and data types

Below Fig. 2.5 illustration displays the online Kaggle UI of the ‘Used Car Dataset’ maintained section.



Used Cars Dataset

Vehicles listings from Craigslist.org

Austin Reese • updated 5 months ago (Version 10)

Usability 10.0 License CC0: Public Domain Tags automobiles and vehicles, economics, united states

Description

Context

Craigslist is the world's largest collection of used vehicles for sale, yet it's very difficult to collect all of them in the same place. I built a scraper for a school project and expanded upon it later to create this dataset which includes every used vehicle entry within the United States on Craigslist.

Content

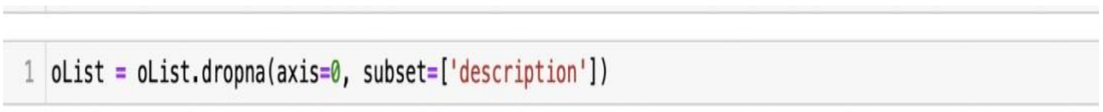
This data is scraped every few months, it contains most all relevant information that Craigslist provides on car sales including columns like price, condition, manufacturer, latitude/longitude, and 18 other categories. For ML projects, consider feature engineering on location columns such as long/lat. For previous listings, check older versions of the dataset.

Figure 2.5 – Kaggle view of the utilized dataset section

Next, the data preprocessing methods were applied to the dataset to make it more suitable and convenient for the concerning algorithm models to perform well. This process is more challenging and should be done with utter most care if model accuracy is highly valued. Some of the simple processes that were executed in this stage upon the dataset would be,

- Deletion of records having null values as a majority.
- Deletion of the columns which are irrelevant for recommendation purposes.
- Processing and converting value types to match with the algorithm types being considered.
- Applying weights for attributes that contribute the most to the recommendation.

Following Fig. 2.6 shows how the null values are removed from the description column, which is the primary column that is considered for the content-based recommendation engine.



```
oList = oList.dropna(axis=0, subset=['description'])
```

Figure 2.6 – Removal of null values from the description column

2.3.2 Google USE embedding creation for existing records

Text embedding plays a vital role in natural language-related tasks. The quality of those depends highly on the dataset basis, which can improve feature caliber. The use of pre-trained models for such text embedding creations provides a variety of advantages such as developed time period being shortened as the model can be used from the point that it was lastly built rather from scratch, only needing a small amount of training and a much better performance being enclosed due to its retraining ability from the starting point it was built. This task shifting and knowledge reallocation can be known as transfer learning. Google universal sentence encoder (USE) is a similar pre-trained model that is capable of encoding textual data formats into high dimensional 512 vectors or otherwise known as embeddings, which are basically numerical representations.

It aims to feed transfer learning to other Natural Language Processing (NLP) tasks such as similarity calculation, relatedness, classification, and clustering. Google USE is publicly available in TensorFlow-hub. This pre-trained model is fundamentally trained on English sentences extracted from various web pages, Wikipedia's, discussion forums, and questions-answer pages. Variable-length input of textual formats can be entered, and the outcome would always be a 512 vector embedding, and semantic textual similarity benchmark is phenomenal regarding Google USE. Getting sentence-level encoded embeddings is made easy as well. When discussing the embeddings, it should be noted that they can be infused upon generic tasks; hence it only captures the most informative features from the sentences and will consequently discard noise. The intuition found here is that with the above-mentioned process, this encoder will be able to address universal NLP matters most effectively.

Below Figure. 2.7 extracted from [18] display how the Google USE convert a particular sentence to a 512-dimensioned vector or an embedding which can then be applied upon multiple tasks for learning.

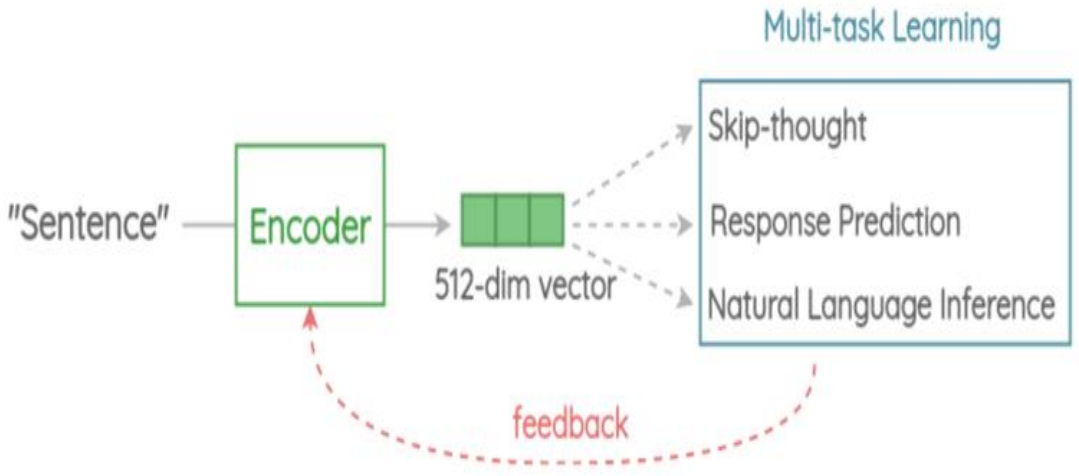


Figure 2.7 – USE embedding creation process [18]

Due to its convenience regarding text embedding creation, this research component has opted to employ it. However, the encoder architectural type utilized for this research is the Deep Averaging Network (DAN), which is a variant of the two types of encoder architectures found in this TensorFlow pretrained model. The other type was the Transformer encoder, and DAN scored points against it with respect to computing resource utilization efficiency and inference time. First of all, in DAN, it averages embeddings of sentences and bi-grams altogether, prior to being passed down towards a four-layer feed-forward deep neural network (DNN) to produce the 512-vector embeddings. During that training, embedding is being learned. Moreover, its compute time is of a linear process in terms of the input sequence sentence length.

Following Fig. 2.8 clearly illustrates how Google USE's DAN encoder architecture performs the sentence embedding/512 vector creation, along with its four-layered model [18].

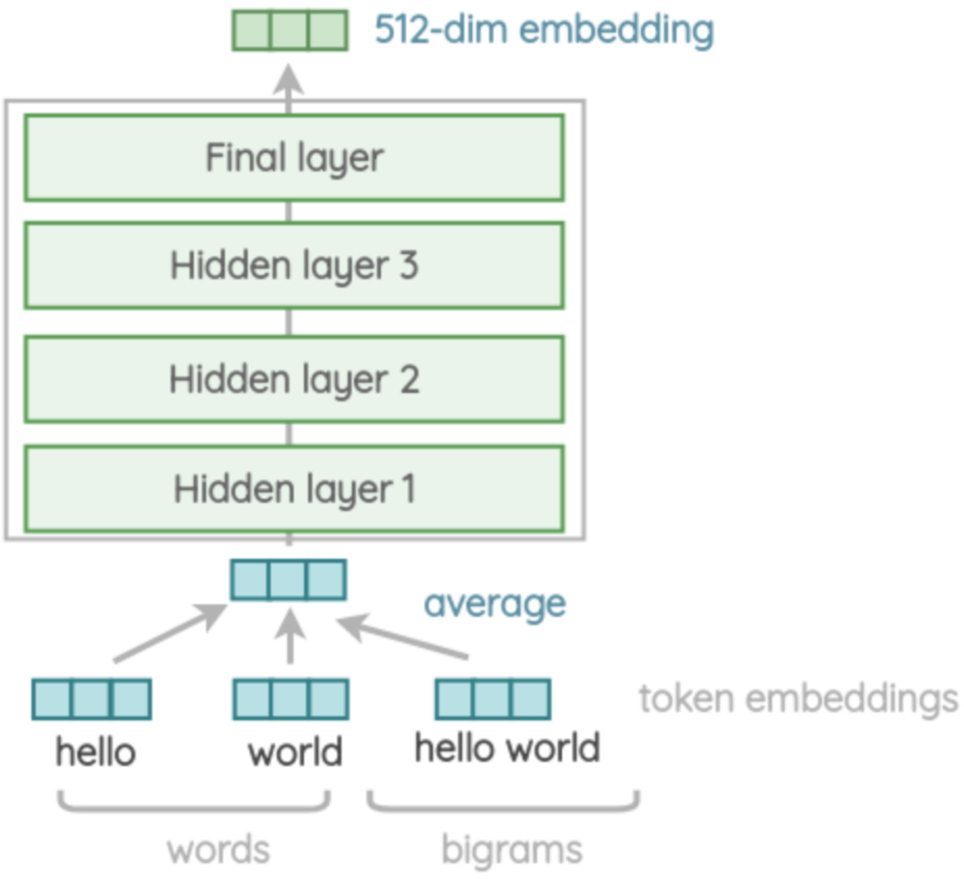
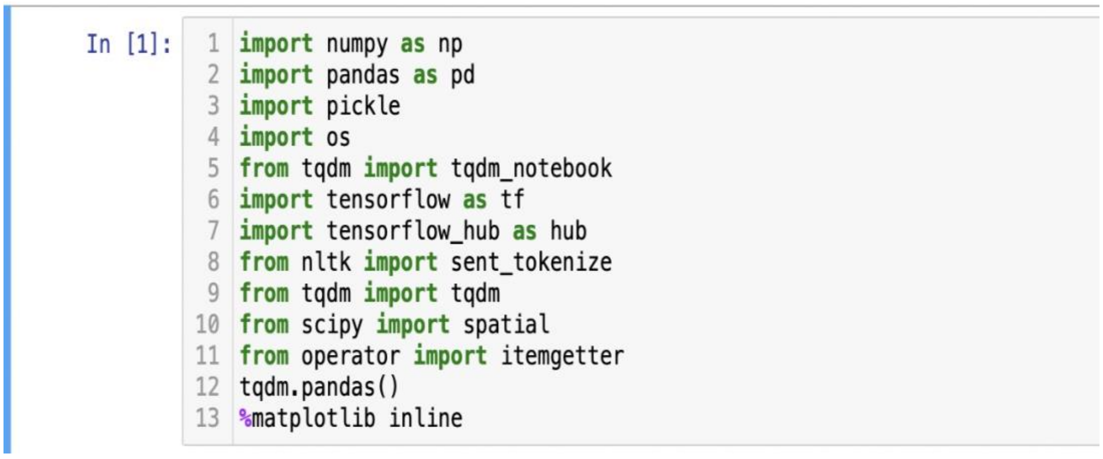


Figure 2.8 – DAN encoder embedding creation process [18]

All the above-stated facts are an introduction to what Google USE is, its architectural designs, and benefits or advantages in using that for a recommendation engine's embedding creation requirement.

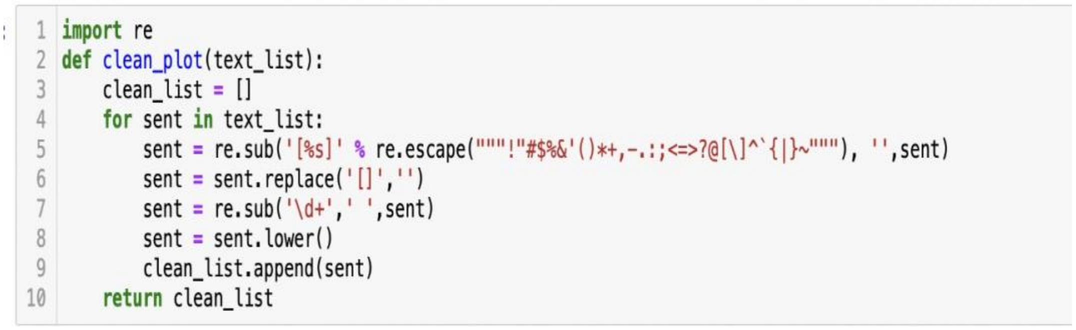
To startup with the process of the model formation, in Jupyter Notebook, all relevant python libraries and dependencies were imported using the necessary commands as shown below Fig. 2.9.



```
In [1]: import numpy as np
        import pandas as pd
        import pickle
        import os
        from tqdm import tqdm_notebook
        import tensorflow as tf
        import tensorflow_hub as hub
        from nltk import sent_tokenize
        from tqdm import tqdm
        from scipy import spatial
        from operator import itemgetter
        tqdm.pandas()
        %matplotlib inline
```

Figure 2.9 – Importing python dependencies

Next, the id and description columns were extracted separately from the dataset, which is 18,035 car records, after handling null and irrelevant values. Then the description column was transferred through a complex regex function for further cleansing. Cleansing tasks include removing unnecessary string characters ($, #, @, *, %, &, =, -, etc.) from the textual content and converting all letters to lowercase. The regex function is shown by the Fig. 2.10 illustration below.



```
import re
def clean_plot(text_list):
    clean_list = []
    for sent in text_list:
        sent = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-.:;<=>?@[\]^`{|}~"""), '',sent)
        sent = sent.replace('[]','')
        sent = re.sub('\d+',' ',sent)
        sent = sent.lower()
        clean_list.append(sent)
    return clean_list
```

Figure 2.10 – Regex function

After that process, the sentences are then tokenized into tokens using Penn Treebank (PTB) tokenizer. Subsequently, the part where the encoder captures these sentences occur. Upon acquiring the sentences, the DAN encoders go through a set of learning processes and share in order to learn the sentence embeddings. The unsupervised tasks are modified skip-thought, conventional input-response prediction, and natural language inference (NLI). Usage of the current sentence to predict the next and previous ones known as a skip-thought concept was used by USE with an encoder basing transformer or DAN, which in this study, DAN was utilized. The USE encoder is trained on this functionality by using Wikipedia and News corpus. Below Fig. 2.11 depicts the basic structure of the training procedure by this task [18].

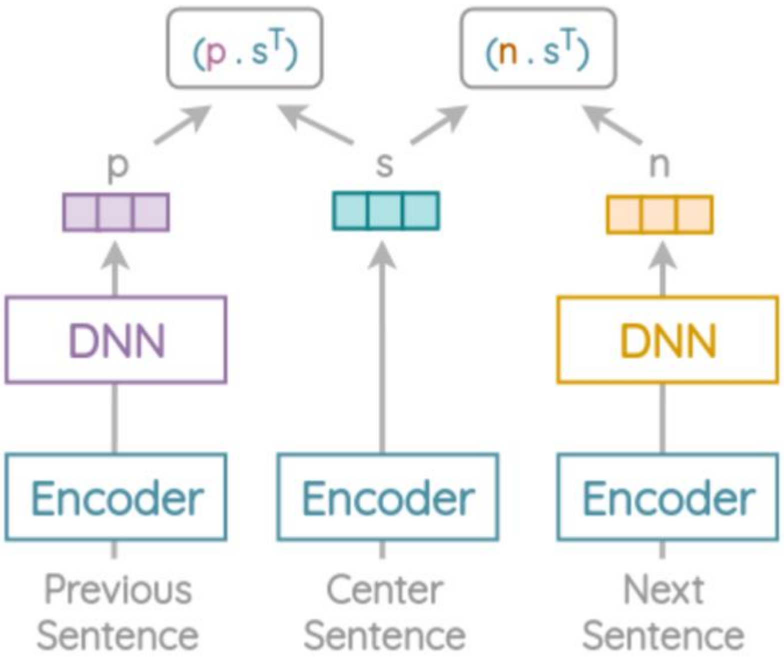


Figure 2.11 – Skip thought task structure [18]

The conversation input-response prediction task is primarily inspired by the Gmail smart reply ability. Corpuses that are scraped from question-answering and discussion forms are used for this type of training process that produces a vector output of both input and response, where response embeddings are passed through a DNN. The dot product of both is taken to calculate the relevancy of the input and response. This ultimately maximized the log-likelihood of each response and its respective inputs. Fig. 2.12 describes the basic architecture of this training method [18].

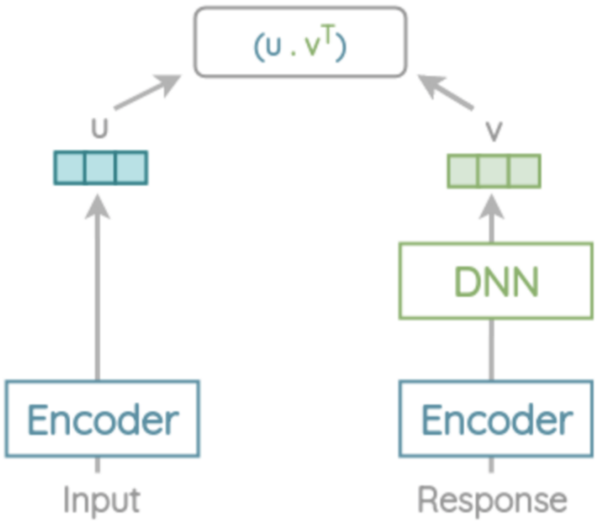


Figure 2.12 – Input-Response prediction [18]

Hypothesis entailments contradict or neutral to a premise is measured and predicted in natural language inferring training task. For that, sentence pairs are used from Stanford Natural Language Inference (SNLI) corpus for USE training. Sentence pairs are afterward encoded using transformer/DAN encoders, and the resulted embedding is taken and then concatenated. This final vector is then passed through a completely connected layer prior to applying the SoftMax classification method to obtain the probability of the judgment type. Fig. 2.13 explains the overall functionality of the learning procedure of NLI [18].

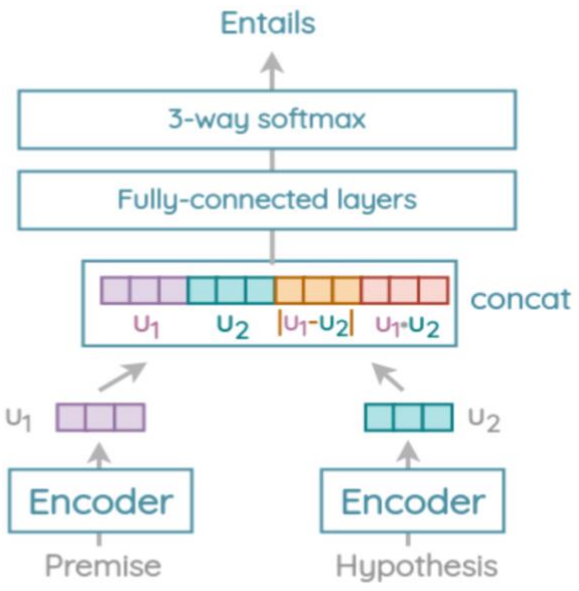
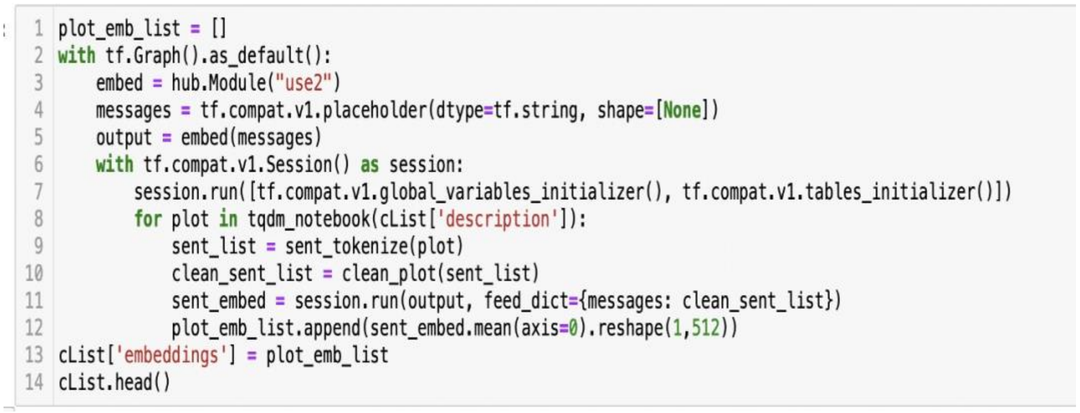


Figure 2.13 – NLI process [18]

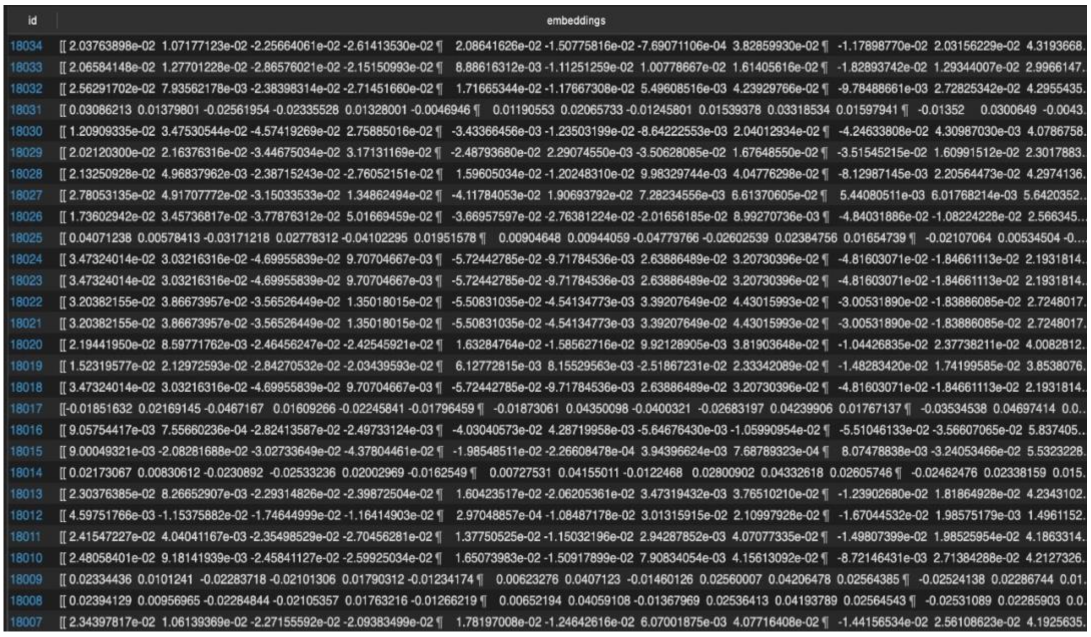
Succeeding the training of the model, it is then capable of generating the appropriate 512-vectors or embeddings for the sentences provided, and this is claimed as the overall inference phase. Following Fig. 2.14 displays the functionality written in python code to achieve the above-mentioned tasks.



```python
plot_emb_list = []
with tf.Graph().as_default():
    embed = hub.Module("use2")
    messages = tf.compat.v1.placeholder(dtype=tf.string, shape=[None])
    output = embed(messages)
    with tf.compat.v1.Session() as session:
        session.run([tf.compat.v1.global_variables_initializer(), tf.compat.v1.tables_initializer()])
        for plot in tqdm_notebook(cList['description']):
            sent_list = sent_tokenize(plot)
            clean_sent_list = clean_plot(sent_list)
            sent_embed = session.run(output, feed_dict={messages: clean_sent_list})
            plot_emb_list.append(sent_embed.mean(axis=0).reshape(1,512))
cList['embeddings'] = plot_emb_list
cList.head()
```

Figure 2.14 – Code for generating embeddings

The new embedding is appended to the existing dataset table as a new column. The generated embeddings of the used car advertisements, based on their description column for recommending purposes, are shown below Fig. 2.15 for abstract referencing.



id	embeddings
18034	[[2.03763898e-02 1.07177123e-02 -2.25664061e-02 -2.61413530e-02 ¶ 2.08641626e-02 -1.50775816e-02 -7.69071106e-04 3.82859930e-02 ¶ -1.17898770e-02 2.03156229e-02 4.3193668...
18033	[[2.06584148e-02 1.27701228e-02 -2.86576021e-02 -2.15150993e-02 ¶ 8.88616312e-03 -1.11251259e-02 1.00778667e-02 1.61405616e-02 ¶ -1.82893742e-02 1.29344007e-02 2.9966147...
18032	[[2.56291702e-02 7.93562178e-03 -2.38398314e-02 -2.71451660e-02 ¶ 1.71665344e-02 -1.17667308e-02 5.49608516e-03 4.23929766e-02 ¶ -9.78488661e-03 2.72825342e-02 4.2955435...
18031	[[0.03086213 0.01379801 -0.02561954 -0.02335528 0.01328001 -0.0046946 ¶ 0.01190553 0.02065733 -0.01245801 0.01539378 0.03318534 0.01597941 ¶ -0.01352 0.0300649 -0.0043...
18030	[[1.20909335e-02 3.47530544e-02 -4.57419269e-02 2.75885016e-02 ¶ -3.43366456e-03 -1.23503199e-02 -8.64222553e-03 2.04012934e-02 ¶ -4.24633808e-02 4.30987030e-03 4.0786758...
18029	[[2.02120300e-02 2.16376316e-02 -3.44675034e-02 3.17131169e-02 ¶ -2.48793680e-02 2.29074550e-03 -3.50628085e-02 1.67648550e-02 ¶ -3.51545215e-02 1.60991512e-02 2.3017883...
18028	[[2.13250928e-02 4.96837962e-03 -2.38715243e-02 -2.76052151e-02 ¶ 1.59605034e-02 -1.20248310e-02 9.98329744e-03 4.04776298e-02 ¶ -8.12987145e-03 2.20564473e-02 4.2974136...
18027	[[2.78053135e-02 4.91707772e-02 -3.15033533e-02 1.34862494e-02 ¶ -4.11784053e-02 1.90693792e-02 7.28234556e-03 6.61370605e-02 ¶ 5.44080511e-03 6.01768214e-03 5.6420352...
18026	[[1.73602942e-02 3.45736817e-02 -3.77876312e-02 5.01669459e-02 ¶ -3.66957597e-02 -2.76381224e-02 -2.01656185e-02 8.99270736e-03 ¶ -4.84031886e-02 -1.08224228e-02 2.566345...
18025	[[0.04071238 0.00578413 -0.03171218 0.02778312 -0.04102295 0.01951578 ¶ 0.00904648 0.00944059 -0.04779766 -0.02602539 0.02384756 0.01654739 ¶ -0.02107064 0.00534504 -0...
18024	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814...
18023	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814...
18022	[[3.20382155e-02 3.86673957e-02 -3.56526449e-02 1.35018015e-02 ¶ -5.50831035e-02 -4.54134773e-03 3.39207649e-02 4.43015993e-02 ¶ -3.00531890e-02 -1.83886085e-02 2.7248017...
18021	[[3.20382155e-02 3.86673957e-02 -3.56526449e-02 1.35018015e-02 ¶ -5.50831035e-02 -4.54134773e-03 3.39207649e-02 4.43015993e-02 ¶ -3.00531890e-02 -1.83886085e-02 2.7248017...
18020	[[2.19441950e-02 8.59771762e-03 -2.46456247e-02 -2.42545921e-02 ¶ 1.63284764e-02 -1.58562716e-02 9.92128905e-03 3.81903648e-02 ¶ -1.04426835e-02 2.37738211e-02 4.0082812...
18019	[[1.52319577e-02 2.12972593e-02 -2.84270532e-02 -2.03439593e-02 ¶ 6.12772815e-03 8.15529563e-03 -2.51867231e-02 2.33342089e-02 ¶ -1.48283420e-02 1.74199585e-02 3.8538076...
18018	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814...
18017	[[-0.01851632 0.02169145 -0.0467167 0.01609266 -0.02245841 -0.01796459 ¶ -0.01873061 0.04350098 -0.0400321 -0.02683197 0.04239906 0.01767137 ¶ -0.03534538 0.04697414 0.0...
18016	[[9.05754417e-03 7.55660236e-04 -2.82413587e-02 -2.49733124e-03 ¶ -4.03040573e-02 4.28719958e-03 -5.64676430e-03 -1.05990954e-02 ¶ -5.51046133e-02 -3.56607065e-02 5.837405...
18015	[[9.00049321e-03 -2.08281688e-02 -3.02733649e-02 -4.37804461e-02 ¶ -1.98548511e-02 -2.26608478e-04 3.94396624e-03 7.68789323e-04 ¶ 8.07478838e-03 -3.24053466e-02 5.5323228...
18014	[[0.02173067 0.00830612 -0.0230892 -0.02533236 0.02002969 -0.0162549 ¶ 0.00727531 0.04155011 -0.0122468 0.02800902 0.04332618 0.02605746 ¶ -0.02462476 0.02338159 0.015...
18013	[[2.30376385e-02 8.26652907e-03 -2.29314826e-02 -2.39872504e-02 ¶ 1.60423517e-02 -2.06205361e-02 3.47319432e-03 3.76510210e-02 ¶ -1.23902680e-02 1.81864928e-02 4.2343102...
18012	[[4.59751766e-03 -1.15375882e-02 -1.74644999e-02 -1.16414903e-02 ¶ 2.97048857e-04 -1.08487178e-02 3.01315915e-02 2.10997928e-02 ¶ -1.67044532e-02 1.98575179e-03 1.4961152...
18011	[[2.41547227e-02 4.04041167e-03 -2.35498529e-02 -2.70456281e-02 ¶ 1.37750525e-02 -1.15032196e-02 2.94287852e-03 4.07077335e-02 ¶ -1.49807399e-02 1.98525954e-02 4.1863314...
18010	[[2.48058401e-02 9.18141939e-03 -2.45841127e-02 -2.59925034e-02 ¶ 1.65073983e-02 -1.50917899e-02 7.90834054e-03 4.15613092e-02 ¶ -8.72146431e-03 2.71384288e-02 4.2127326...
18009	[[0.02334436 0.0101241 -0.02283718 -0.02101306 0.01790312 -0.01234174 ¶ 0.00623276 0.0407123 -0.01460126 0.02560007 0.04206478 0.02564385 ¶ -0.02524138 0.02286744 0.01...
18008	[[0.02394129 0.00956965 -0.02284844 -0.02105357 0.01763216 -0.01266219 ¶ 0.00652194 0.04059108 -0.01367969 0.02536413 0.04193789 0.02564543 ¶ -0.02531089 0.02285903 0.0...
18007	[[2.34397817e-02 1.06139369e-02 -2.27155592e-02 -2.09383499e-02 ¶ 1.78197008e-02 -1.24642616e-02 6.07001875e-03 4.07716408e-02 ¶ -1.44156534e-02 2.56108623e-02 4.1925635...

Figure 2.15 – Generated embeddings

2.3.3 Database selection & migration

Since the embeddings are successfully created as of this stage, the next step is to export these embedding data into a database. The intention of this upload towards a database is to eliminate the cold start problem and to assist in generating a real-time, dynamic database that is able to create and update advertisement-related embeddings repeatedly as required, all the while adding new embedding entries into the database. When deleting a particular advertisement, the specific embedding for that will likewise be deleted as well. Due to that process, which simultaneously updates the embeddings as necessary, each recommendation produced towards the consumers would be the most up-to-date or revised information, hence providing a better user experience altogether. Following Fig. 2.16 depicts the real-time dynamic database transactions.

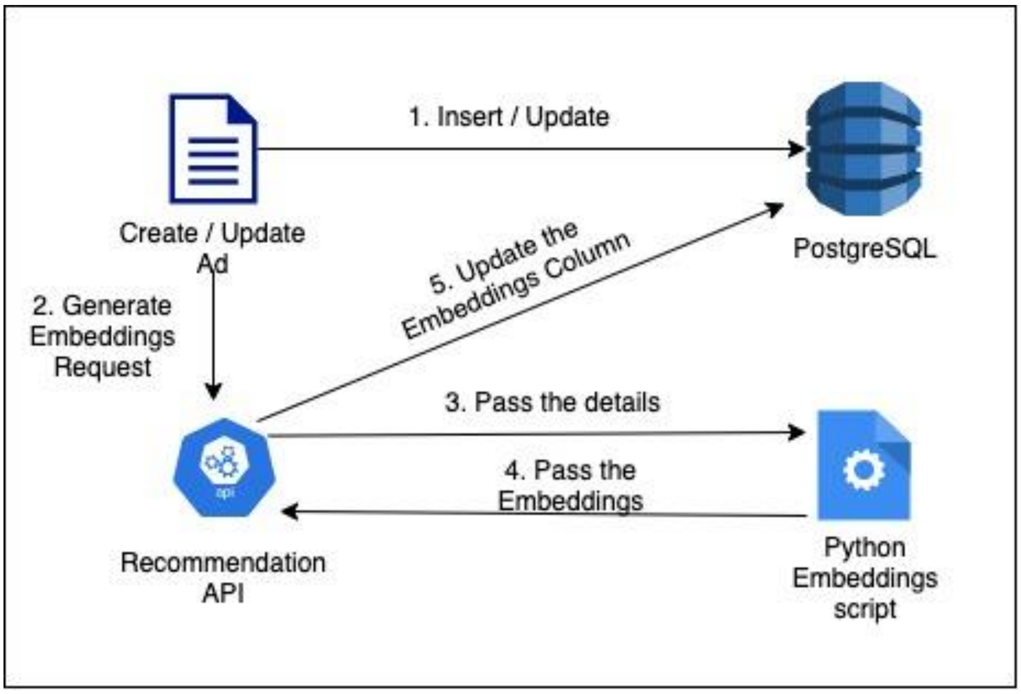


Figure 2.16 – Real-time dynamic database

On that account, there are several eligible database systems prevailing and functioning smoothly within the IT industry. If the Redis database were to be considered, that can be known as a high-speed induced system, although it could not be used for this research component since the scalability of it was not up to a commendable level and RAM usages were high as well.

Therefore, the below-mentioned database benchmark displayed by Fig. 2.17 [19] were reviewed and analyzed for faster data read speed (smaller bar chart length, the better) in order to determine the most optimal system that is appropriate to the existing technology infrastructure.

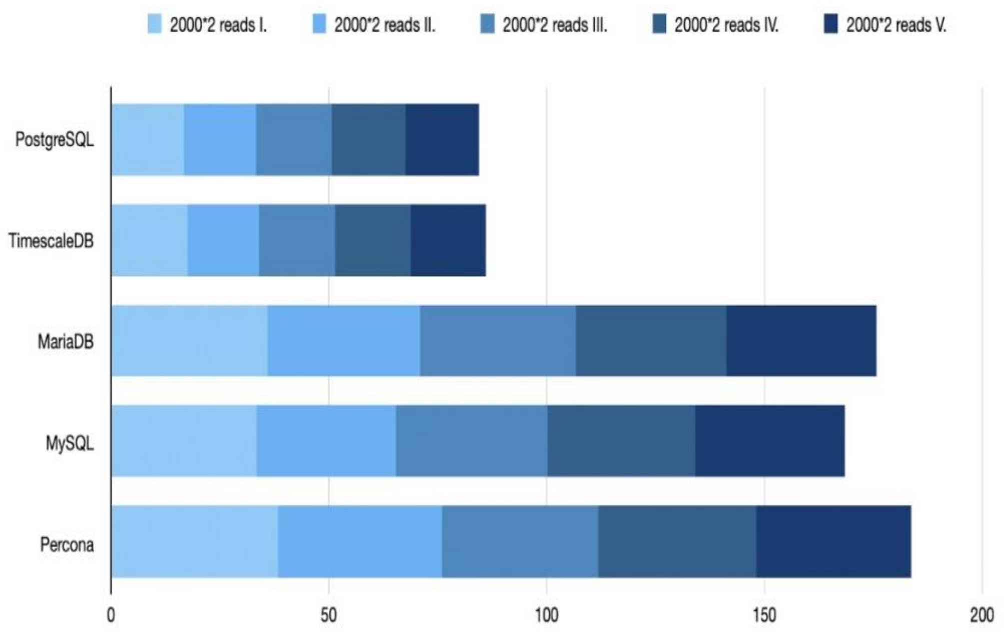


Figure 2.17 – Database system read comparison [19]

Heeding to the above reference material [19] and some other randomly viewed web articles and YouTube videos, a conclusion was made declaring PostgreSQL the supreme winner out of all, by having the smallest CPU utilization, smallest memory usage, smallest size/storage, and the fastest read and write speeds which exceeds the metrics of other database systems.

Fast reads are a must because when each time we need to get real-time or newest ad embeddings for user preferred suggestion that has a 512-dimensioned vector, a quick-reacting database should be in place within the recommendation engine to manage not only read transactions but also write as well and that is exactly why PostgreSQL must be incorporated specifically within this scenario.

Therefore, for this research component's database requirement, PostgreSQL was exploited without any hesitation. The new embeddings generated along with the table itself were then imported into the PostgreSQL database.

2.3.4 Golang API implementation

The main purpose of the API that should be incorporated along with the recommendation process was discovered, and it is divided into two key functions. Those two functions are creating new embeddings for the newly published or submitted classified advertisements and calculation of the cosine similarity using those created and stored advertisement embeddings in order to present the consumers their preferred or otherwise recommended classifieds through the Tievs applications front-end view.

For this component, the rudimental programming language taken into consideration was the python framework since all other functionalities were also developed using the same technology. However, though a trial was conducted using the above-said language to gain the obligated result as an API under the hypothesis that it would work without any impediments, it was quite unsuccessful due to its newly identified and analyzed inability to handle concurrency when trying to calculate cosine similarity metrics using all of the embeddings created within the database table for all advertisements under consideration.

To further improve those API performance results while eliminating impediments and insignificance, sub-research had to be conducted to find other technology frameworks out there which could effortlessly and cost-effectively attain the desired and confirmed objectives. That research appraised how the USE model can be smoothly integrated with the API endpoints and dependency libraries as well.

After the conclusion of that research was done, it was inaugurated that the frameworks of the GO programming language performed much better at a greater level than any other language, especially with regard to concurrency and HTTP server performance metrics. Following Fig. 2.18 displays how python and Go language performed as APIs with their POST, GET and redirect endpoint executions. It clearly displays the python language's inefficiency when compared to the Go language [20].

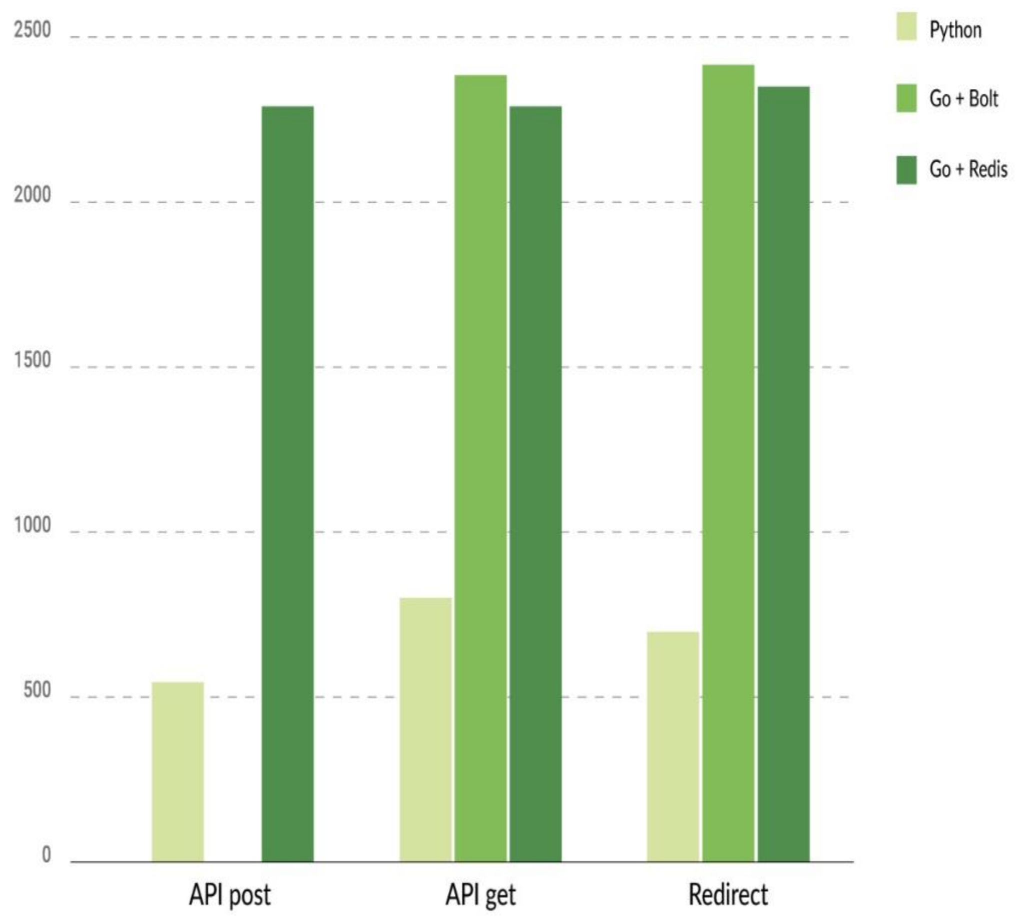


Figure 2 18 – Python Vs. Go [20]

In addition, below bar chart of Fig. 2.19 illustrates the HTTP server performances when handling network traffic requests per second of the programming languages Java Jersey (embedded Grizzly), Go net/HTTP, Node HTTP, and Node Express 4 [21]. The bar chart result that favors Golang is eye-catching itself.

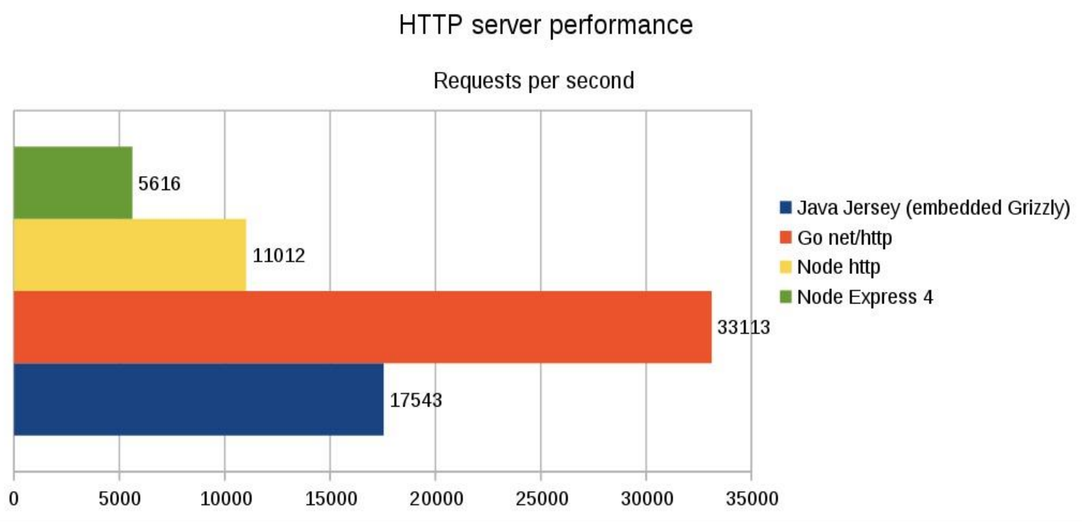


Figure 2.19 – Server performance comparison [21]

Go routines are known as specialists in concurrency by many parties, including [19,20]. Java is also able to write concurrency routines by implementing a runnable interface or extending the Thread class and creating threads using its built-in support. On the other hand, Go lang provides very simple and lightweight independent concurrency routines. It will be completely managed by the Go runtime.

Go lang itself can be known as a concurrent language which further increases the desirability for it to be incorporated into this research component. A go routine is called simple by adding the ‘go’ keyword at the beginning of the function and implicitly enables the concurrency aspect. Plus, Goroutines are much faster than Java threads.

Following shown Fig. 2.20 diagram displays a curve that explains how the Java threads and Go routines collaborate when controlling matrixes, which is quite useful for this research as well since 512-vector embeddings are at work [22].

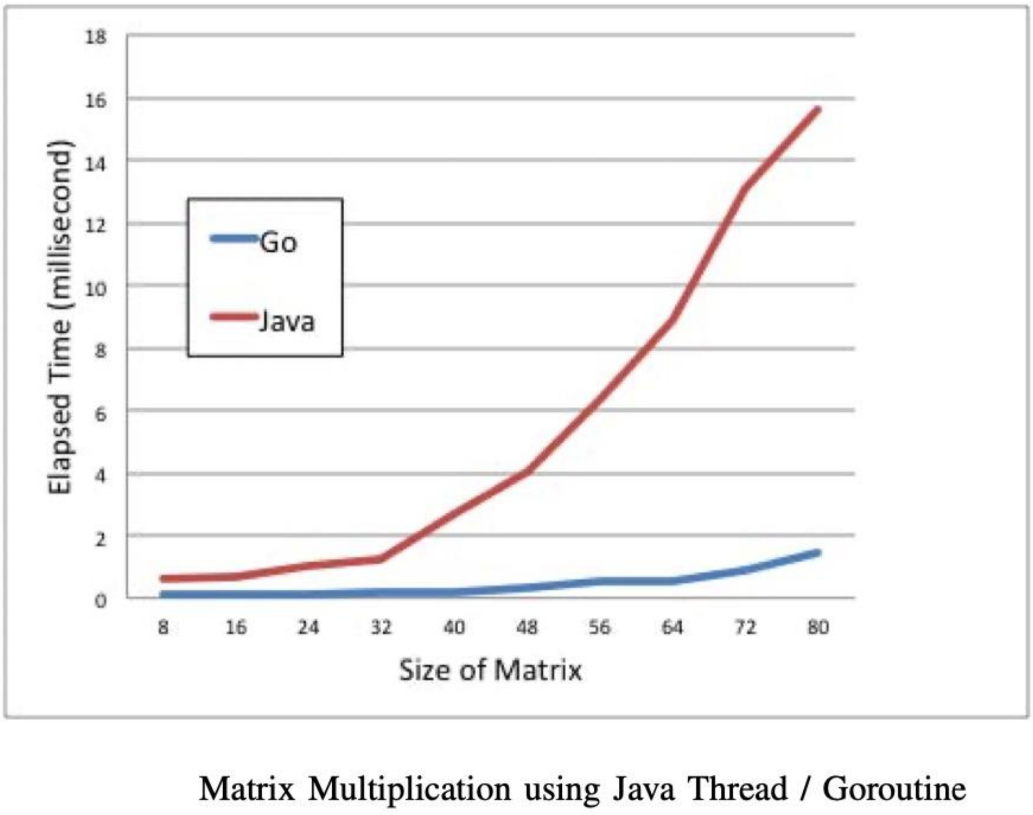


Figure 2.20 – Java thread Vs. Go routine [22]

The working process of a Go scheduler is depicted by the below displayed Fig. 2.21 diagram [23].

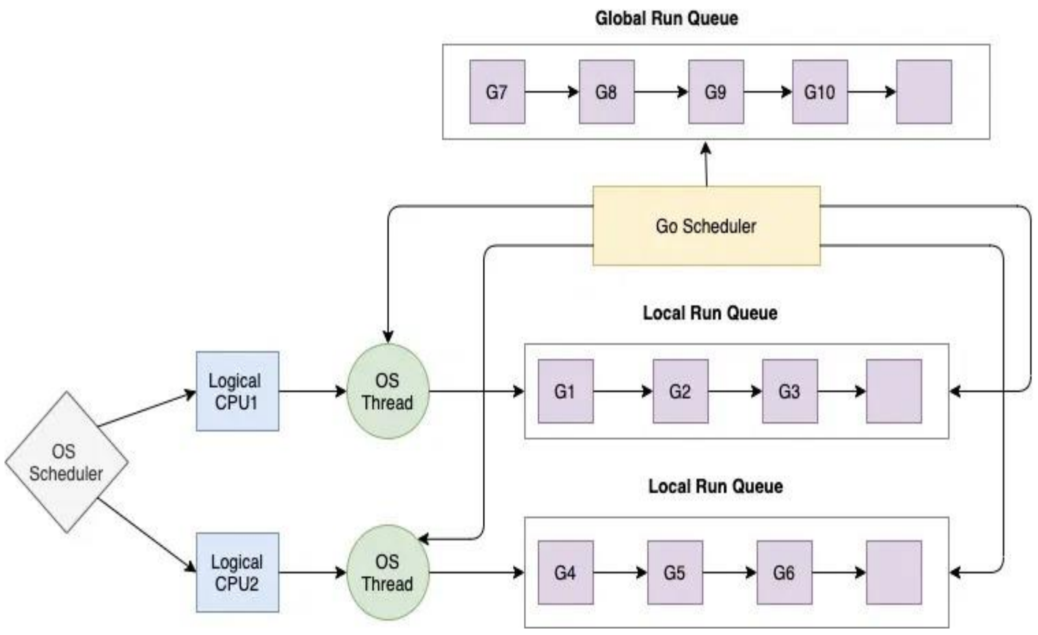


Figure 2.21 – Go language schedular [23]

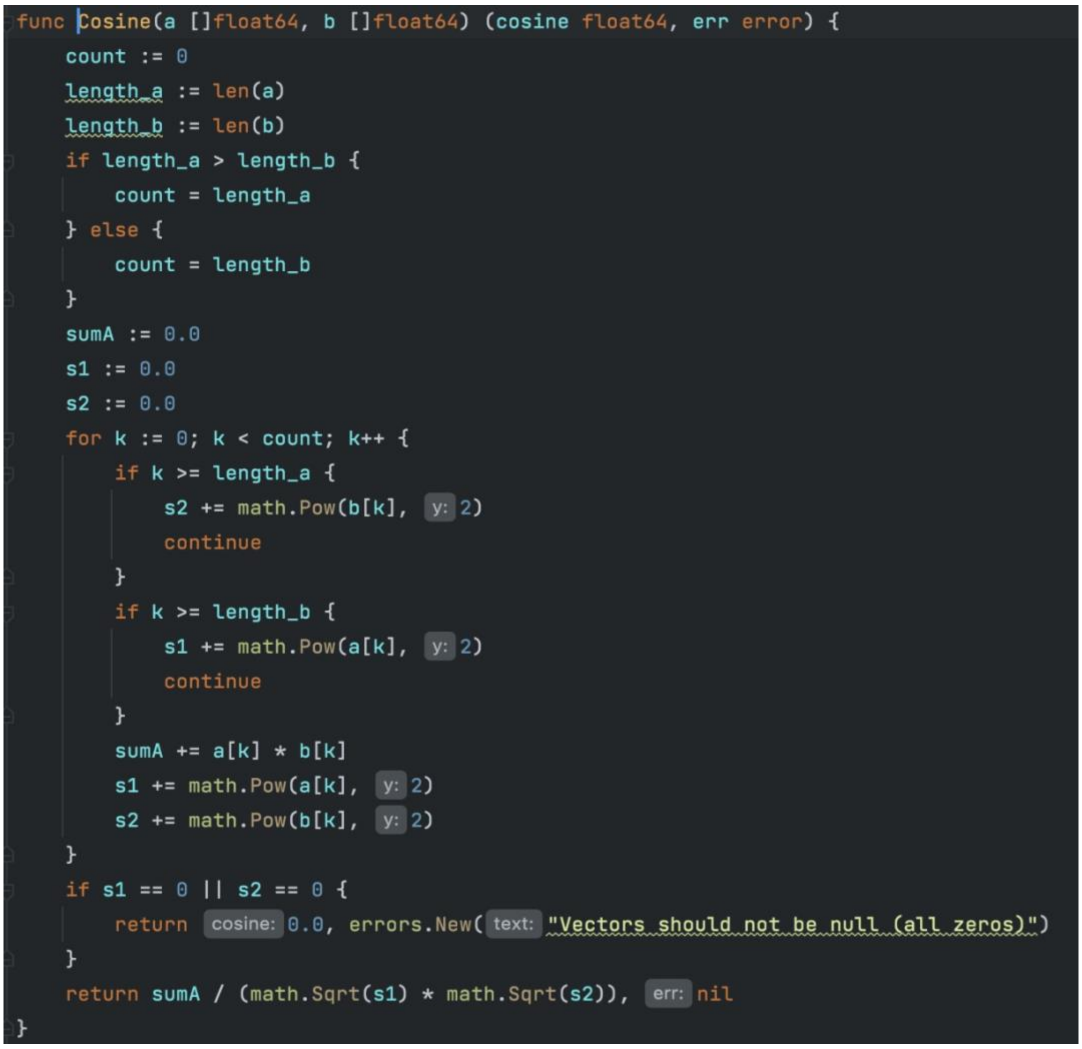
Ultimate the Go language was chosen for the API to be implemented, and, to be more precise, the Golang Echo framework was selected out of other Go language frameworks due to its working ability with HTTP/2 protocol and low latency having a combination of simplicity but robustness. This feature was quite useful when integrating with the front-end application because it was developed basing a JAM stack architecture in Cloudflare.

Real-time similarity calculating function

The basic concept behind what is happening when a consumer clicks on a particular ad and how the consequent recommendation process happens is explained below.

The API receives a GET request of the clicked ad, and it procures the specific id of that classified ad from the URL params. According to the id acquired, the state and embeddings of the specific advertisement were be fetched from the database. The next step is to request and obtain the other advertisement ids and embeddings from the database with respect to the previously clicked advertisement state. Those embeddings gained are in a string format. Therefore, those formatted string embeddings are newly converted into a float array. After that, the major step of cosine similarity calculation was be initiated. In order to do that along with concurrency at a place, a normal Go lang map cannot be used for the task. Nonetheless, as a workaround, a specific map called sync. Map had to be exploited. Below Fig. 2.22 shows the Go language API code for cosine similarity calculation. The equation below (1) depicts the mathematical function which is responsible for calculating the similarity value.

$$similarity = \cos(\theta) = \frac{A\,.B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

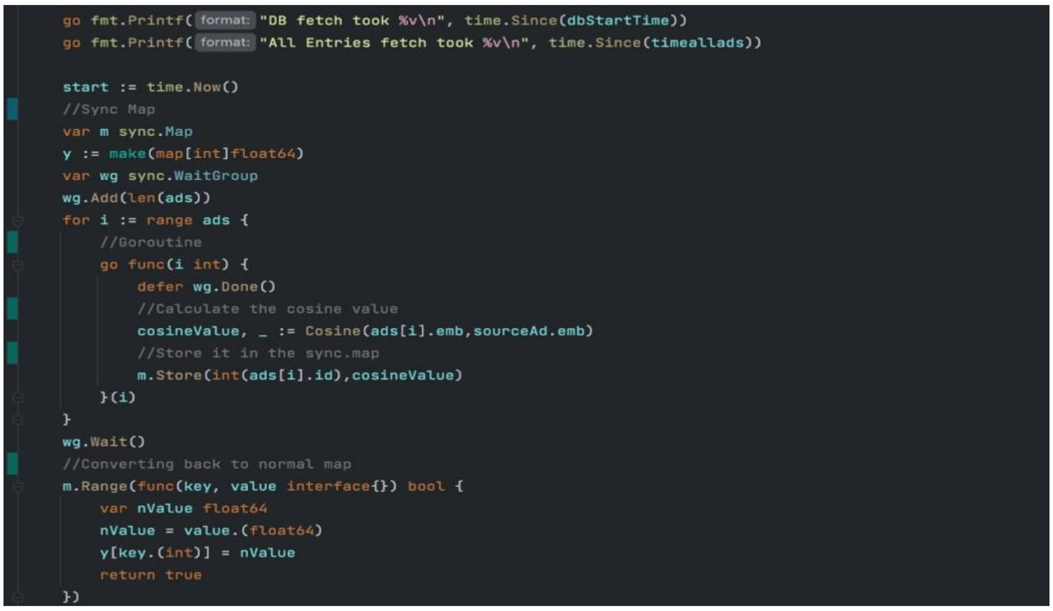


```go
func Cosine(a []float64, b []float64) (cosine float64, err error) {
    count := 0
    length_a := len(a)
    length_b := len(b)
    if length_a > length_b {
        count = length_a
    } else {
        count = length_b
    }
    sumA := 0.0
    s1 := 0.0
    s2 := 0.0
    for k := 0; k < count; k++ {
        if k >= length_a {
            s2 += math.Pow(b[k], y: 2)
            continue
        }
        if k >= length_b {
            s1 += math.Pow(a[k], y: 2)
            continue
        }
        sumA += a[k] * b[k]
        s1 += math.Pow(a[k], y: 2)
        s2 += math.Pow(b[k], y: 2)
    }
    if s1 == 0 || s2 == 0 {
        return cosine: 0.0, errors.New( text: "Vectors should not be null (all zeros)")
    }
    return sumA / (math.Sqrt(s1) * math.Sqrt(s2)), err: nil
}
```

Figure 2.22 – Go lang cosine similarity calculation code

By using the sync.Map, concurrency is applied to a ‘for each’ function to analyze and calculate the similarity value between the selected specific advertisement embedding along with the other embeddings within the array list of the database. The newly calculated similarity value and the id of those ads are then saved in the sync. Map array. At this stage, the concurrency part has been completed successfully. Now, for the further proceedings such as sorting, the sync. The map has to be converted into a conventional map since its primary usage of leveraging the concurrency aspect is done. Simultaneously, the similarity value was converted to a float type value as well. With the help of a specific function (sort. slice), the float converted values are sorted in descending order in the table. The highest or the top most similar records are considered here onwards. There are other details relating to these id’s such as year, price, description, manufacturer, which are essential attributes when building the front-end application. Those are requested from the database at this stage and why it wasn't done previously was due to the unnecessary, redundant traffic and memory utilization it caused by requesting all details of the not-so-matching advertisements. Bottlenecks can arise and make the response time fluctuate in a higher latent level to calculate and process all functions.

The below Fig. 2.23 is the Go language implemented API code that does all the above-mentioned procedures.



```go
go fmt.Printf( format: "DB fetch took %v\n", time.Since(dbStartTime))
go fmt.Printf( format: "All Entries fetch took %v\n", time.Since(timeallads))

start := time.Now()
//Sync Map
var m sync.Map
y := make(map[int]float64)
var wg sync.WaitGroup
wg.Add(len(ads))
for i := range ads {
    //Goroutine
    go func(i int) {
        defer wg.Done()
        //Calculate the cosine value
        cosineValue, _ := Cosine(ads[i].emb,sourceAd.emb)
        //Store it in the sync.map
        m.Store(int(ads[i].id),cosineValue)
    }(i)
}
wg.Wait()
//Converting back to normal map
m.Range(func(key, value interface{}) bool {
    var nValue float64
    nValue = value.(float64)
    y[key.(int)] = nValue
    return true
})
```

Figure 2.23 – Cosine value calculation for all ads

These top similarity scores are then sent as a JSON formatted response to the front-end application through the API.

Realtime embedding creation

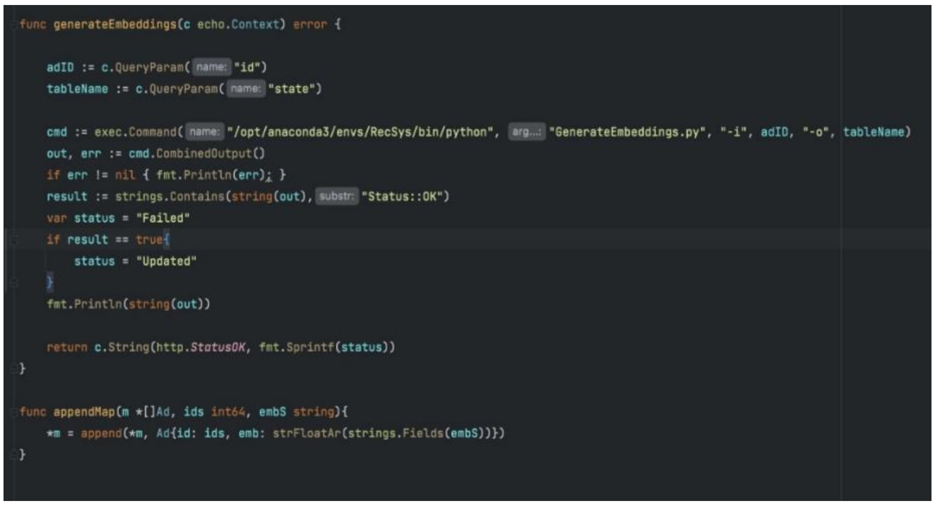
The real-time embedding creation is also a crucial part of the API implemented. Previously, this thesis discussed that the embeddings are being created using the Google USE pretrained model before deployment. After the production deployment, the same is happening thanks to the novel functionality introduced within the API.

When the customer fills all advertisement details from the Tievs UI and submits it by clicking on the submit button near completion, the POST request is sent to the database to store all necessary details. At the end of that process, a unique identification number is allocated for this newly submitted advertisement. Parallelly, another request is sent to the Go lang implemented API. In that request's URL parameters, the newly created ad id is present, and the API acquires that id from the params section. It is then the API's job to initiate the real-time embedding creation. The following procedures are undertaken hereafter.

- The API calls the exec. command towards the local python file to run a script that assists in generating the embeddings from the generate embeddings function (Fig. 2.24 shows the source code of the API executing this part).
- The id with other important details, especially the description column data of the specific ad under consideration, is sent.
- The waiting process starts now for the Go lang API.
- Meanwhile, the python file (GenerateEembeddings.py) containing the generate embeddings function having the complex regex function starts executing the above-mentioned process in the same way by starting to clean the description column, lowercase it, and tokenize it using Penn Treebank tokenizing.
- Then the pretrained TensorFlow model acquired the description sentences to create the embeddings using the DAN architecture.
- After the new embedding is created, they are updated safely in the database table's specific id assigned embeddings column (Fig. 2.25 displays the

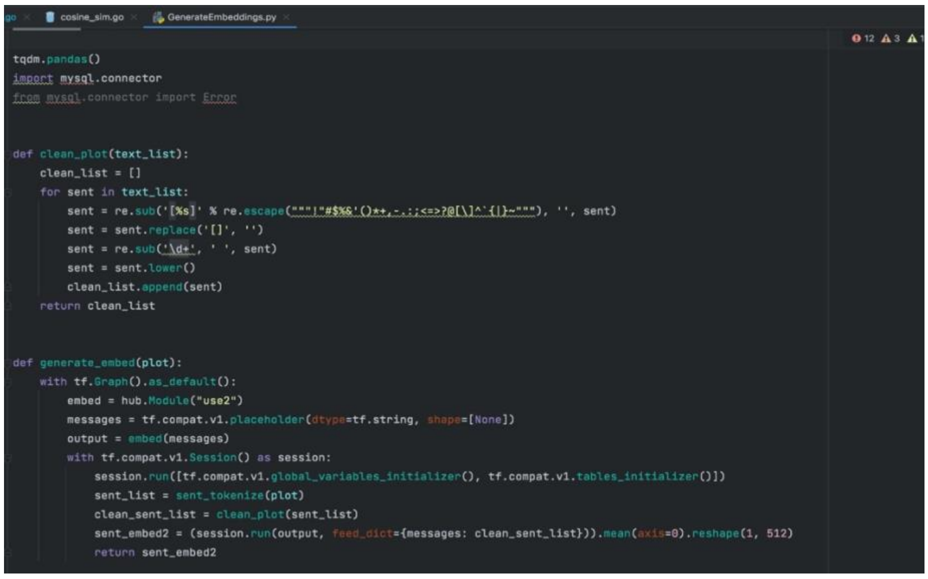
GenerateEembeddings.py file's source code for this part with the function itself).

- The process success or any faults that occurred are subsequently informed to the Go API with a response message.
- Finally, the Go API's response about the success or failure of the embedding creation process is sent to the response expected party as well.



```go
func generateEmbeddings(c echo.Context) error {

    adID := c.QueryParam( name: "id")
    tableName := c.QueryParam( name: "state")

    cmd := exec.Command( name: "/opt/anaconda3/envs/RecSys/bin/python",  arg...: "GenerateEmbeddings.py", "-i", adID, "-o", tableName)
    out, err := cmd.CombinedOutput()
    if err != nil { fmt.Println(err); }
    result := strings.Contains(string(out), substr: "Status::OK")
    var status = "Failed"
    if result == true{
        status = "Updated"
    }
    fmt.Println(string(out))

    return c.String(http.StatusOK, fmt.Sprintf(status))
}

func appendMap(m *[]Ad, ids int64, embS string){
    *m = append(*m, Ad{id: ids, emb: strFloatAr(strings.Fields(embS))})
}
```

Figure 2.24 – Go API function for new embedding creation initiation



```python
tqdm.pandas()
import mysql.connector
from mysql.connector import Error


def clean_plot(text_list):
    clean_list = []
    for sent in text_list:
        sent = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-.:;<=>?@[\]^`{|}~"""), '', sent)
        sent = sent.replace('[]', '')
        sent = re.sub('\d+', ' ', sent)
        sent = sent.lower()
        clean_list.append(sent)
    return clean_list


def generate_embed(plot):
    with tf.Graph().as_default():
        embed = hub.Module("use2")
        messages = tf.compat.v1.placeholder(dtype=tf.string, shape=[None])
        output = embed(messages)
        with tf.compat.v1.Session() as session:
            session.run([tf.compat.v1.global_variables_initializer(), tf.compat.v1.tables_initializer()])
            sent_list = sent_tokenize(plot)
            clean_sent_list = clean_plot(sent_list)
            sent_embed2 = (session.run(output, feed_dict={messages: clean_sent_list})).mean(axis=0).reshape(1, 512)
            return sent_embed2
```

Figure 2.25 – GenerateEmbeddins.py source code

2.3.5 Recommendation engine deployment

The recommendation engine consisting of both the Golang API and the Google USE included machine learning model has deployed DigitalOcean cloud server environment for 24/7 hour available access. DigitalOcean is an infrastructure as a service platform (IaaS) that is capable of swift and complex cloud deployments that run on multiple cloud servers simultaneously.

The API which resides within the server is implemented to obtain fault-tolerant with NGNIX proxy with Supervisorctl so that it would be able to perform load balancing inside the server. This can be known as a special characteristic that has been accomplished in the model deployment phase.

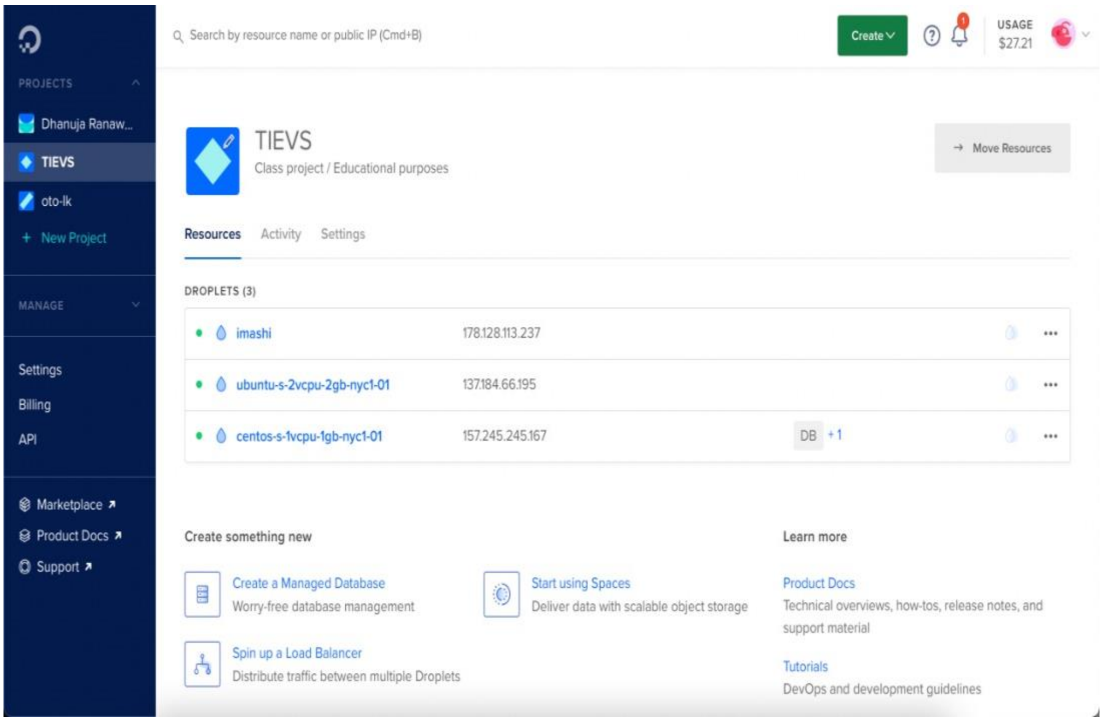
The specifications of the cloud droplet purchased from DigitalOcean are as below.

- RAM – 2 GB
- CPU – 2
- SSD Disk – 60 GB
- Transfer – 3 TB

Inside the droplet, Golang is installed, and the API is compiled while the Supervisorctl manages and monitors all transactions and processes. Not only the recommendation API but also the overall research project's rest of the member component API are stationed within the same droplet in order to support communication channels.

Although that is the current circumstance, the performance of the recommendation system has not been affected even to the slightest when being deployed and run in the same environment with the same specifications.

Below shown Fig. 2.26 displays the DigitalOcean dashboard overview where all the purchased droplets are displayed in the TIEVS folder under the current logged-in credentials.



TIEVS
Class project / Educational purposes

DROPLETS (3)

imashi	178.128.113.237	
ubuntu-s-2vcpu-2gb-nyc1-01	137.184.66.195	
centos-s-1vcpu-1gb-nyc1-01	157.245.245.167	DB +1

Create something new

Create a Managed Database
Worry-free database management

Start using Spaces
Deliver data with scalable object storage

Spin up a Load Balancer
Distribute traffic between multiple Droplets

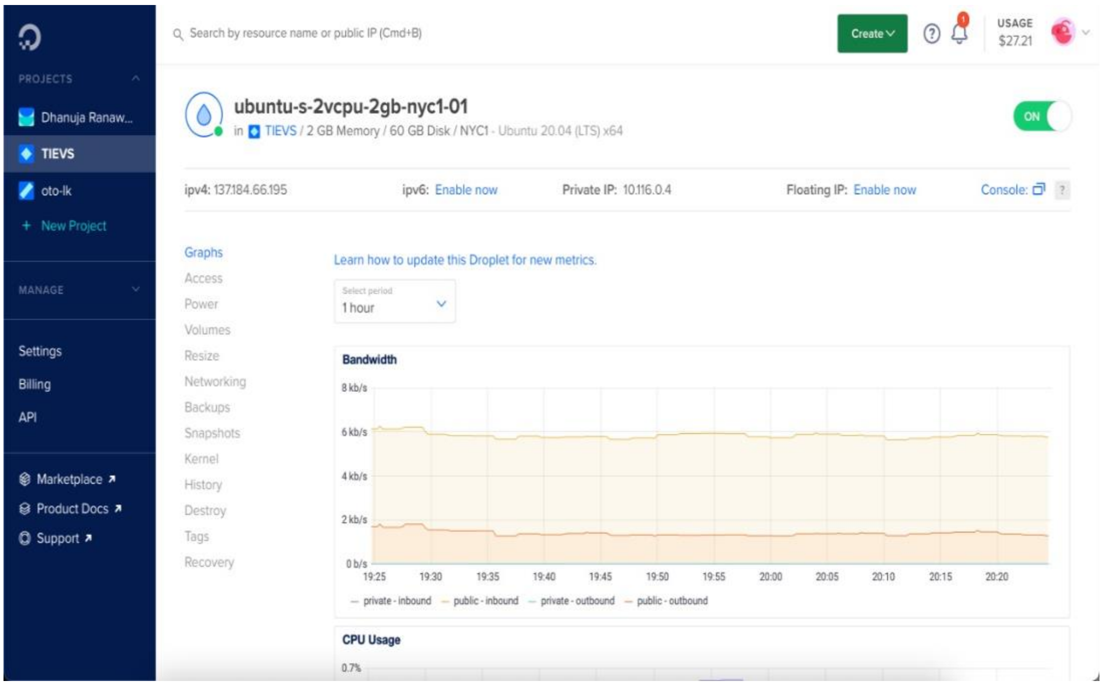
Learn more

Product Docs
Technical overviews, how-tos, release notes, and support material

Tutorials
DevOps and development guidelines

Figure 2.26 – DigitalOcean dashboard

In addition to that, the below Fig. 2.27 illustrates the procured recommendation engine residing droplet’s specific dashboard.



ubuntu-s-2vcpu-2gb-nyc1-01
in TIEVS / 2 GB Memory / 60 GB Disk / NYC1 - Ubuntu 20.04 (LTS) x64

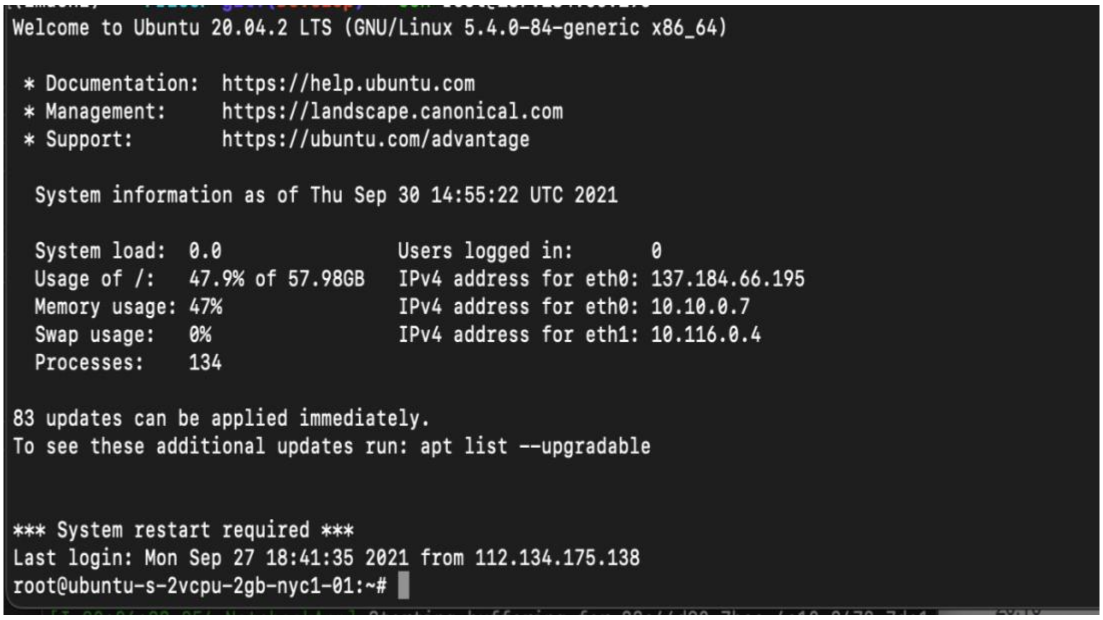
ipv4: 137.184.66.195 ipv6: Enable now Private IP: 10.116.0.4 Floating IP: Enable now Console:

Learn how to update this Droplet for new metrics.

Select period
1 hour

Figure 2.27 – Droplet dashboard

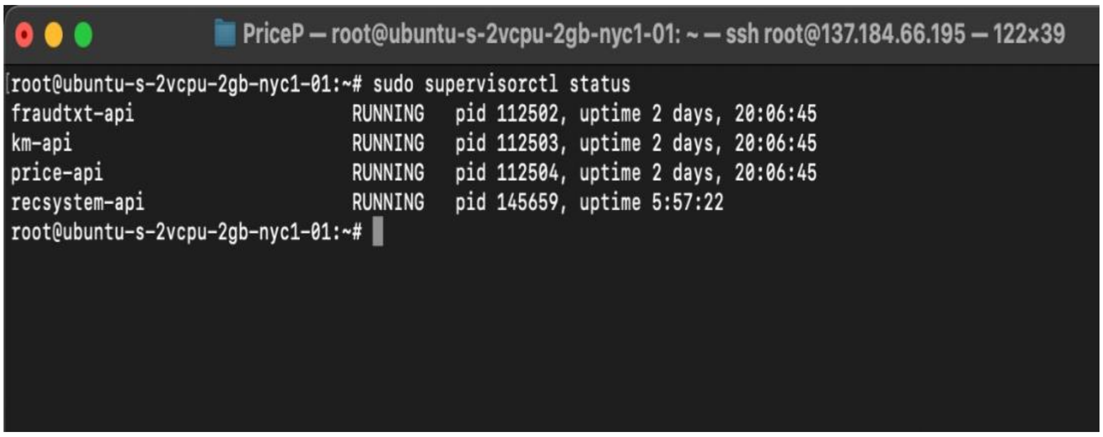
Moreover, the next depicted Fig. 2.28 demonstrate the droplet's command-line interface's Secure Shell (SSH) tunnel.



```
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-84-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Sep 30 14:55:22 UTC 2021

  System load:  0.0               Users logged in:       0
  Usage of /:   47.9% of 57.98GB  IPv4 address for eth0: 137.184.66.195
  Memory usage: 47%               IPv4 address for eth0: 10.10.0.7
  Swap usage:   0%                IPv4 address for eth1: 10.116.0.4
  Processes:    134

83 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


*** System restart required ***
Last login: Mon Sep 27 18:41:35 2021 from 112.134.175.138
root@ubuntu-s-2vcpu-2gb-nyc1-01:~#
```

Figure 2.28 – Droplet SSH tunnel

Finally, the Supervisorctl, which is responsible for handling the API processes execution availability by restarting and reuploading the services if they were to crash or face any failures, is displayed by Fig. 2.29. The rest of the member deployed APIs and models are located in the same location, as seen below.



PriceP — root@ubuntu-s-2vcpu-2gb-nyc1-01: ~ — ssh root@137.184.66.195 — 122×39

```
root@ubuntu-s-2vcpu-2gb-nyc1-01:~# sudo supervisorctl status
fraudtxt-api                     RUNNING   pid 112502, uptime 2 days, 20:06:45
km-api                           RUNNING   pid 112503, uptime 2 days, 20:06:45
price-api                        RUNNING   pid 112504, uptime 2 days, 20:06:45
recsystem-api                    RUNNING   pid 145659, uptime 5:57:22
root@ubuntu-s-2vcpu-2gb-nyc1-01:~#
```

Figure 2.29 – Supervisorctl status

2.3.6 Front-end application development & integration

The front-end application, which is called Tievs, was fundamentally built based on the Angular framework using Typescript as the programming language managing UI components.

It is again an open-source, free web application building platform developed by Google’s Angular team and other individual voluntary communities, especially targeting single-page client-side applications, that is much more convenient for Tievs as an e-commerce web app. Core and optional facilities are imports as typescript libraries during implementation.

Accommodating large application necessities, typescript is ideal for statistical, highly paced front-end development environments since it can build reusable, correct, and maintainable code with mutability and JavaScript XML (JSX) support. Safety, along with the structure of the source code, is a supplementary feature as well.

A specific back-end functionality to implement services and API calling procedures is not necessary when Angular is used since it is not just a simple library but a whole matured framework, and that is the reason this research component had chosen Angular over React, which is still just a library which needs back-end wise support.

Angular Material UI designing dependency libraries were used with CSS and HTML to smooth out the UI features in a more user-friendly and eye-catching manner.

While Angular components handle the UI features and their dynamic, meticulous reactions without the need for repeated page refreshing, services are responsible for the database and API endpoint server function invocations from the front-end itself.

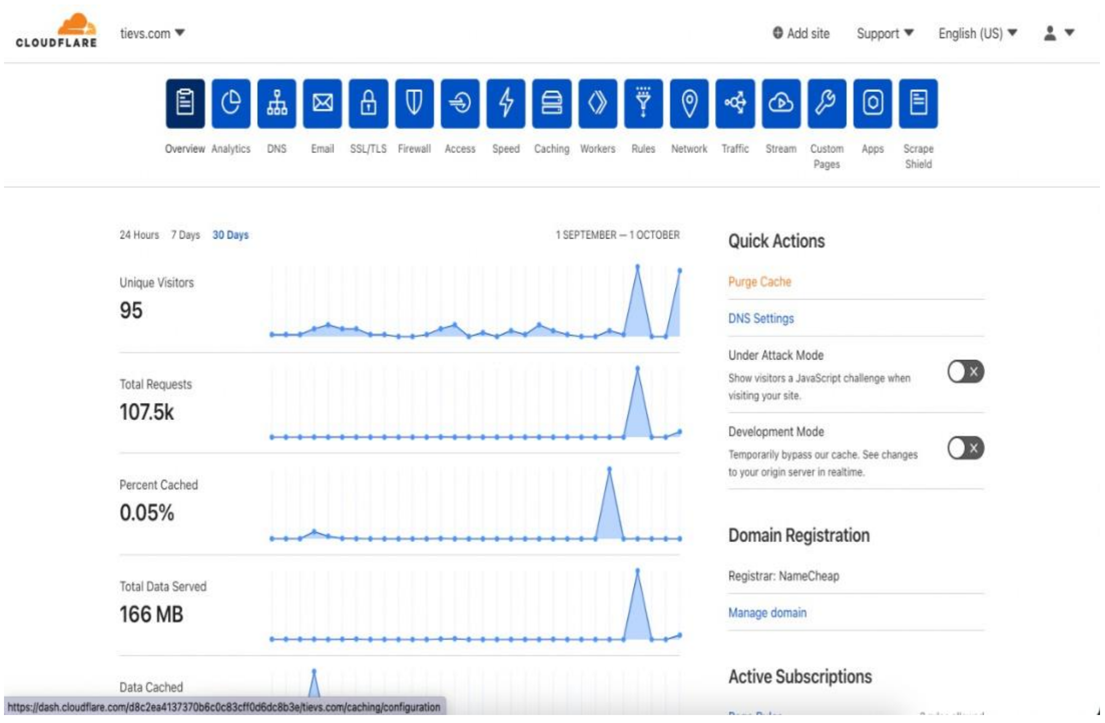
It basically does the job of a back-end service without any troublesome additional programming.

The front-end integrated API and the USE included model is deployed in Cloudflare pages.

Cloudflare pages is a deployment and collaboration JavaScript, reusable API and markup (JAM stack) platform with dynamic and effort-less functionality having Git provided for front-end developers targeting an unmatched performance.

The Cloudflare overview page where the Tievs application resides is shown by the below Fig. 2.30, where metrics such as total requests, precents cached, total data served, data cached, unique visitors are displayed.

Fig. 2.31 depicts the Cloudflare pages' production platform's deployment source and status details.



Unique Visitors 95

Total Requests 107.5k

Percent Cached 0.05%

Total Data Served 166 MB

Data Cached

Quick Actions

Purge Cache

DNS Settings

Under Attack Mode
Show visitors a JavaScript challenge when visiting your site.

Development Mode
Temporarily bypass our cache. See changes to your origin server in realtime.

Domain Registration

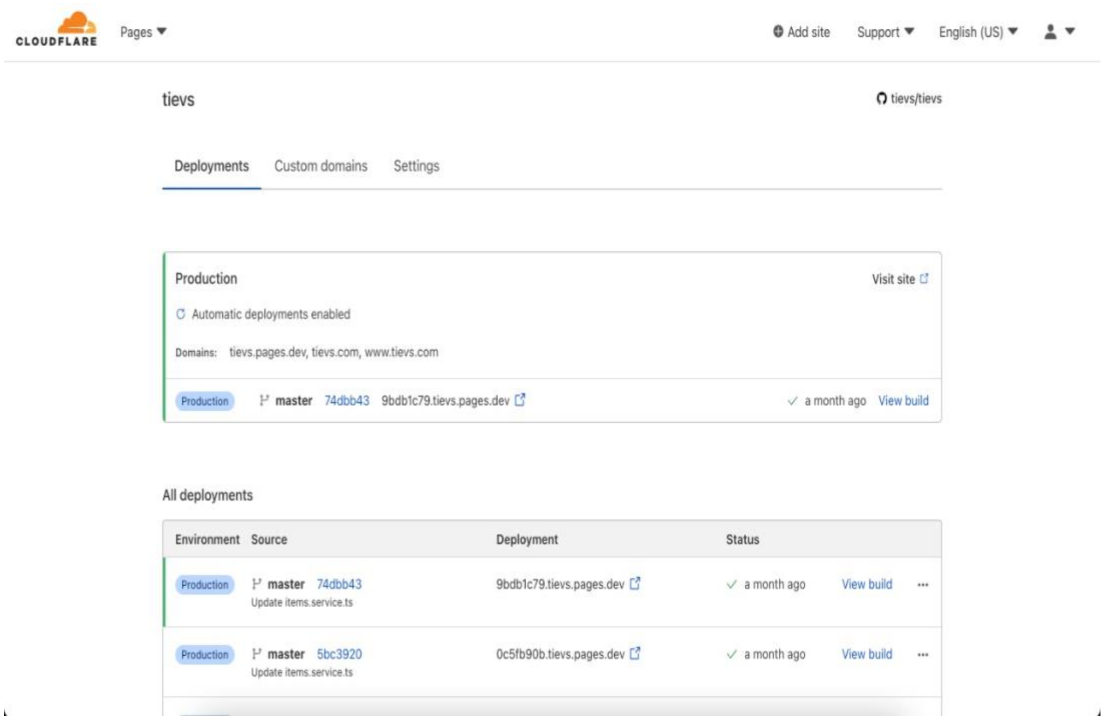
Registrar: NameCheap

Manage domain

Active Subscriptions

https://dash.cloudflare.com/d8c2ea4137370b6c0c83cff0d6dc8b3e/tievs.com/caching/configuration

Figure 2.30 – Cloudflare application deployment dashboard part 1



tievs

tievs/tievs

Deployments   Custom domains   Settings

Production

Visit site

Automatic deployments enabled

Domains: tievs.pages.dev, tievs.com, www.tievs.com

Production   master 74dbb43   9bdb1c79.tievs.pages.dev   ✓ a month ago   View build

All deployments

Environment	Source	Deployment	Status		
Production	master 74dbb43 Update items.service.ts	9bdb1c79.tievs.pages.dev	✓ a month ago	View build	•••
Production	master 5bc3920 Update items.service.ts	0c5fb90b.tievs.pages.dev	✓ a month ago	View build	•••

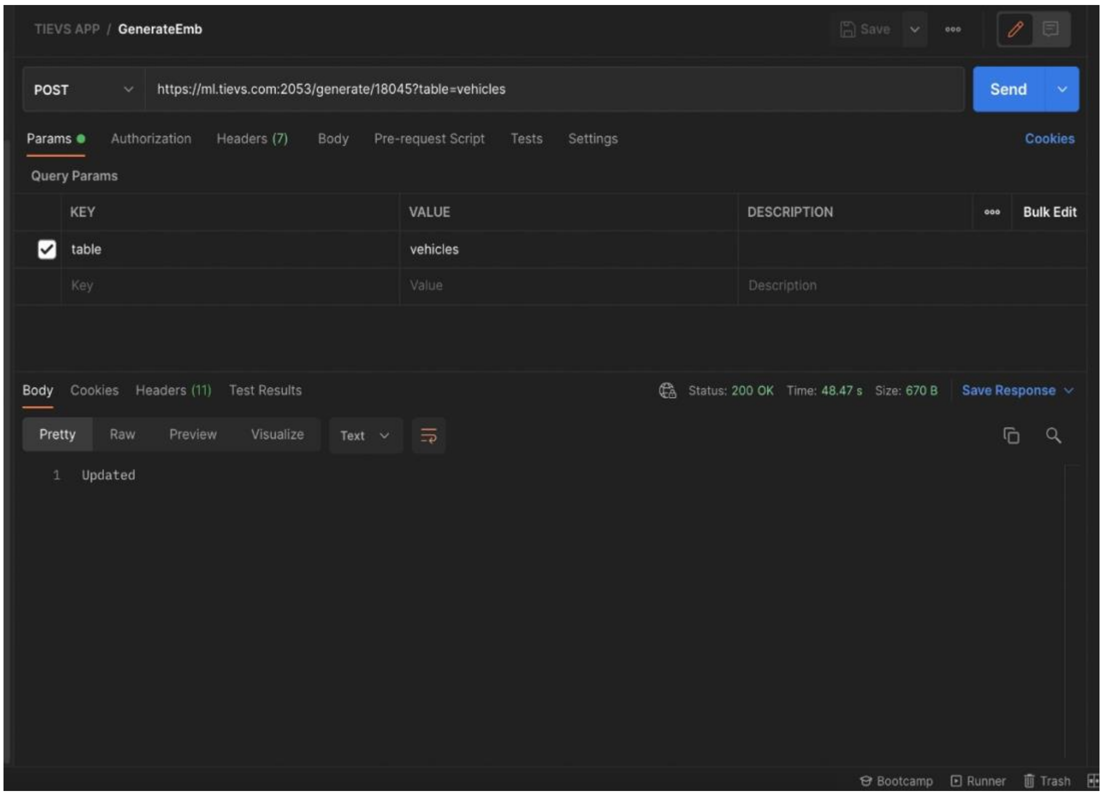
Figure 2.31 – Cloudflare application deployment dashboard part 2

After all application and back-end API and ML model deployment and integration, the user is then able to browse preferred classifieds, click on them and receive the highest correlating and suggestive classifieds as a reference from the bottom panel of the UI. The application's front-end UI result will be displayed in chapter three, 'Results & Discussion' section.

2.3.7 API & model integration testing

Initial testing was required to be executed upon the Golang implemented concurrency enabled API and imported trained Google USE model, which was deserialized and integrated inside the API itself. For that purpose of API testing, Postman was utilized due to its convenient capability to test HTTP/2 (HTTPS) requests through a graphical user interface having all necessary requirements regarding parameter types, request body types, and request types such as POST, PUT, GET, DELETE. Different types of responses can be acquired, which can further be validated as well. Requests can be saved with collection classification, and metrics such as time, size, status responses can be evaluated. Furthermore, by using Postman as an HTTP client, header, authorizations, and pre-request script management was made possible regarding the API to be tested. Fig. 2.32 displays the test done on the POST request of the API, which triggers the calculation and generation of the new embeddings according to a new advertisement submission sent as a table containing the vehicle type classified ad details.

Fig. 2.33 shown below demonstrates how the deployed API server was tested by provoking a GET request for a specific car advertisement. The particular car id, which is assumed to be clicked by a customer in a real scenario, is passed using the URL parameters. The response is displayed as a JSON result indicating all essential attributes of an advertisement that is being recommended, such as price, year, model, manufacturer, and images.



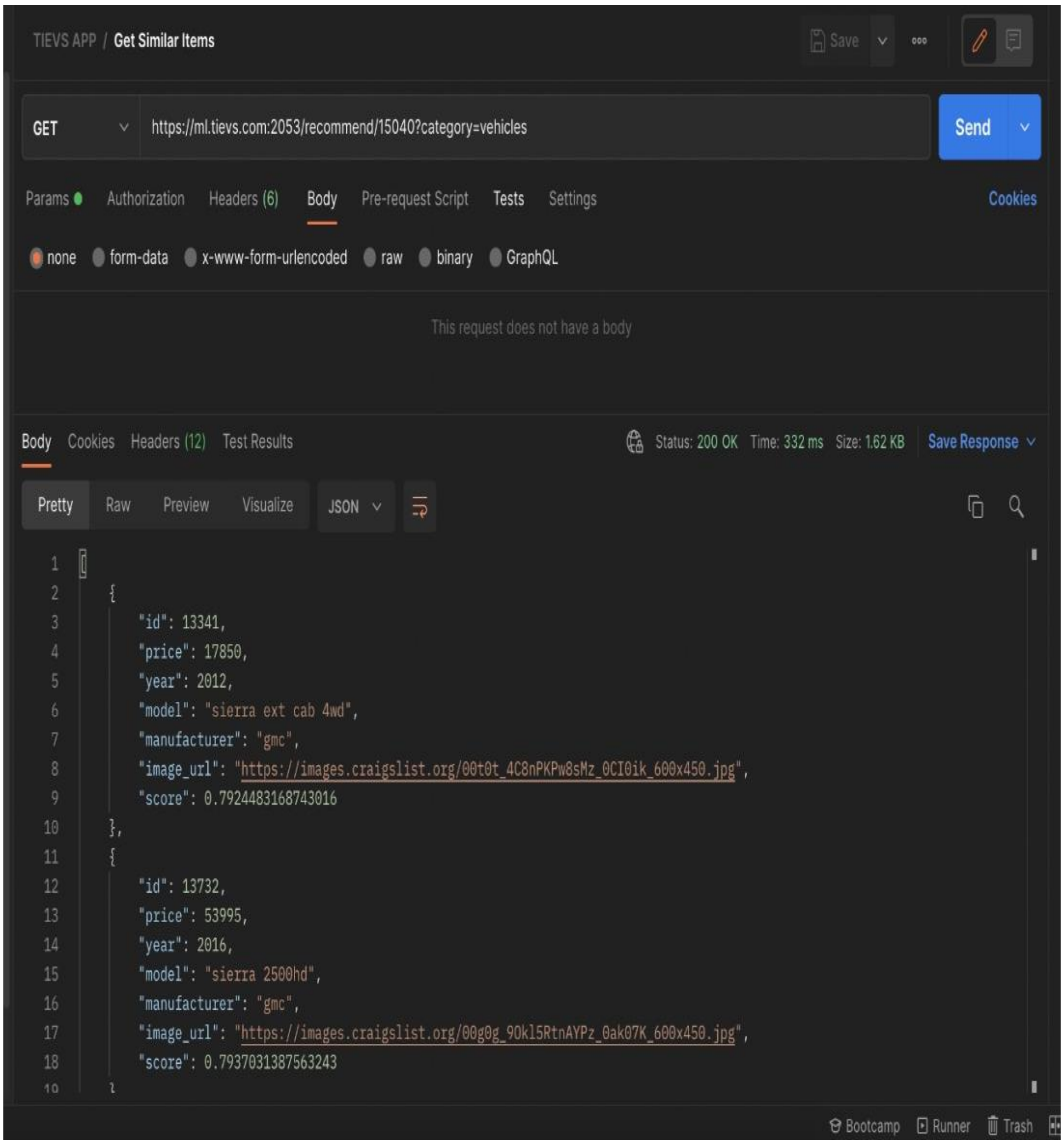
```
POST https://ml.tievs.com:2053/generate/18045?table=vehicles
```

KEY	VALUE	DESCRIPTION
✓ table	vehicles	
Key	Value	Description

Status: 200 OK Time: 48.47 s Size: 670 B

```
Updated
```

Figure 2.32 – Postman generate embeddings testing



GET https://ml.tievs.com:2053/recommend/15040?category=vehicles

This request does not have a body

Status: 200 OK Time: 332 ms Size: 1.62 KB

```
[
    {
        "id": 13341,
        "price": 17850,
        "year": 2012,
        "model": "sierra ext cab 4wd",
        "manufacturer": "gmc",
        "image_url": "https://images.craigslist.org/00t0t_4C8nPKPw8sMz_0CI0ik_600x450.jpg",
        "score": 0.7924483168743016
    },
    {
        "id": 13732,
        "price": 53995,
        "year": 2016,
        "model": "sierra 2500hd",
        "manufacturer": "gmc",
        "image_url": "https://images.craigslist.org/00g0g_9Okl5RtnAYPz_0ak07K_600x450.jpg",
        "score": 0.7937031387563243
    }
```

Figure 2.33 – Postman get similar ads testing

Succeeding the triumphant API implementation with the integration of the developed ML ensemble model, it was necessary for it to be deployed in a cloud server environment for the client application's efficient accessing. For that, the DigitalOcean Infrastructure as a Service (IaaS) cloud service was utilized.

3 RESULTS & DISCUSSION

3.1 Research Results

This chapter is mainly responsible for emphasizing the outcomes of conducting and executing this research component development. A summary is provided regarding the methodology followed, respective results, any newly found statistics, and further future remarks.

The sole aim of this innovative recommendation engine implementation was to provide the customers with the most efficient times advertisement suggestion system without facing any inconveniences or bottlenecks. The market of recommendation systems has been soaring ever since, and that proves the worthiness of giving attention to such an aspect within the domain of information and communication technology. Especially for e-commerce websites such as classified advertising web applications, all platform reputation, productiveness, usefulness for consumers depends on how efficiently the inbuilt recommendation functions. The functionality itself is not the only attribute that should be present, but also the whole UI presentation should be easy to conceive.

It is definitely safe to say that the aimed or targeted goals of conducting this research component have been successfully achieved. The fundamental goal to experiment on a novel and unexplored approach by prevailing researchers within the field of recommendation and classified advertising was fathomed by the satisfying results received from the recommendation engine developed using Google USE pre-trained and re-trained TensorFlow imported model.

In general, a recommendation engine should focus not only on addressing the consumer requirements that should be fulfilled by online classified advertising but also from the application developers' perspective; the system should be easily manageable and maintainable. When discussing that aspect, the metrics such as speed, throughput, response time, memory usage, CPU usage, concurrency, error rates, attempted sessions, and success or failure rates should be taken into consideration. Otherwise, the overall system architecture designed would be difficult to handle when deployed into the production where continuous access is happening from the customers' side, where hardware and software resource consumption could be at a higher multitude. If the immense weight on the infrastructure is to be lifted, the implementing systems must be carefully produced, with a meticulous design having a sound ability to mitigate large traffic requests aimed towards the application, servers, and other networking elements.

Contemplating the above-stated facts, the introduced system has been able straightforwardly to attain the set objectives, making it a far superior recommender system than existing competitor systems. Following revealed Fig. 3.1 exhibit few matrices have astounding values that were used to attest the Tievs recommendation system’s potentiality in achieving goals. The dashboard is dispensed for the API’s registered domain under the load testing software known as ‘Loadmill.

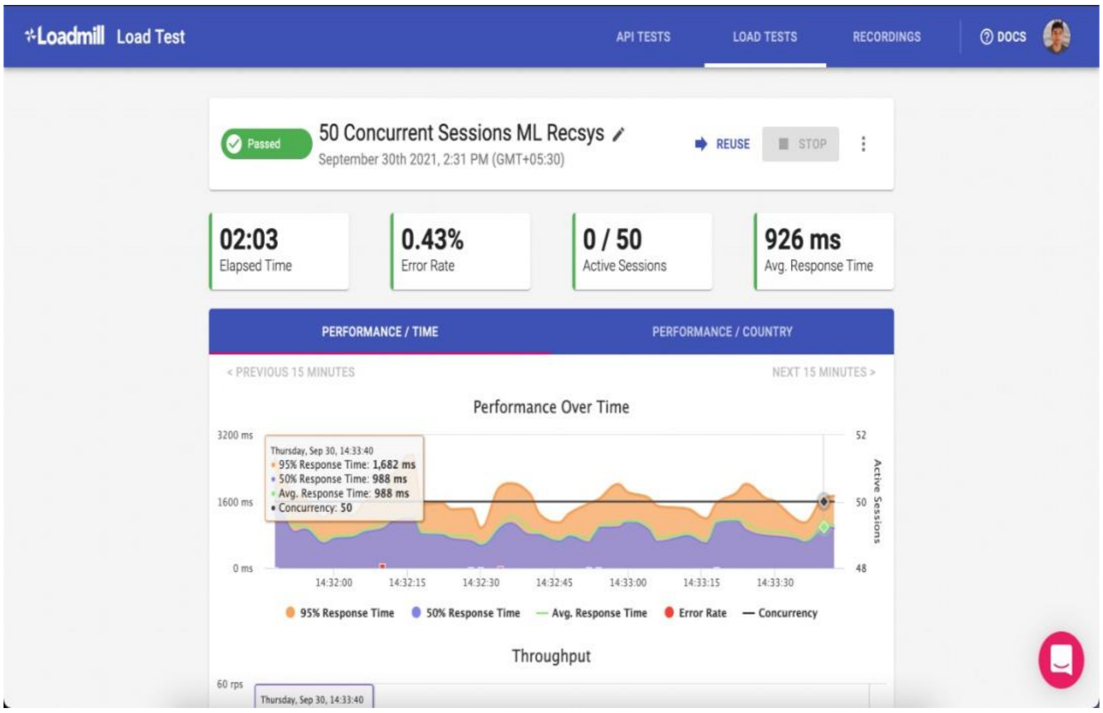


Figure 3.1 – Loadmill load test

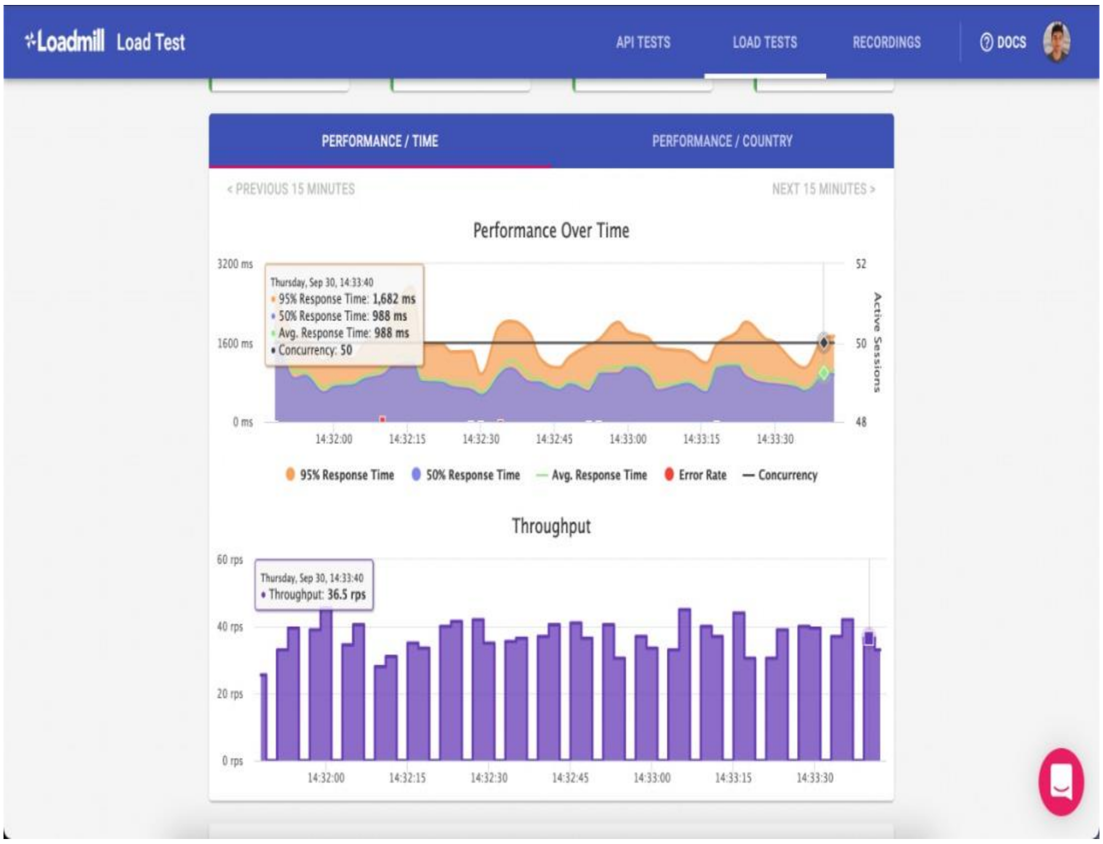
Fig. 3.1 shows that there are 50 sessions concurrently running to make an impact on the server load. Concurrency implemented through Golang is a major aspect that has been researched and analyzed thoroughly through this proposed recommendation system. The Loadmill platform initially had allowed for only five different user sessions to be run within the load testing conducted for the allocated elapsing time.

However, such a low amount of sessions was not adequate to test the overall power of the recommendation engine that could handle the concurrency commendably. Moreover, that is a critical quality that should be maintained by this research component being built, and that was another major characteristic that fills out the gap within the existing research done in the same domain and also what keeps other competitive systems at bay or at a lower level than this system.

To rectify the matter, the Golang API was redefined and reconstructed at a code level to allow for 50 sessions to be established during the load test conducted by Loadmill. By that, the necessary requirements for the concurrency handling capability were regulated.

As a result of the 50 concurrent sessions, those sessions running within an average response time of 926ms was absolutely magnificent. What's astonishing even more is the error rate received. A 0.43% error rate being generated was remarkable considering the concurrency with 50 sessions itself. Error rate itself is not a metric to be ignored and misjudged since that is what defines the accuracy level of the service provided by the recommendation engine.

The accuracy must be precise in order for the customers to receive a more than satisfactory outcome and which should not be forgotten down the road. What makes the error rate so extravagant is the fact that the API being load tested with 50 sessions is handling a crucial and prominent functionality, which is the cosine similarity calculating and filtering the top ads related to a specified advertisement. About 18,000 records scoped with the state of the advertisement are being considered for the calculation, and that takes a toll on the system resources, which implicitly prompts for failures. Nevertheless, this recommender engine handled the assigned tasks without any residual’s and displayed its vibrant performance through the error rate. The elapsed time which the load test was run completely was about 02:03 minutes. The performance over time graph is displayed from the below Fig. 3.2 more clearly about the average response time 988ms, 50% average response time 988ms, and 90% response time 1682ms. The throughput graph illustrates the gradually fluctuating request per second (rps) graph, and the currently pointed throughput value remains at 36.5 rps, which can be reckoned as a similarly outstanding result procured.

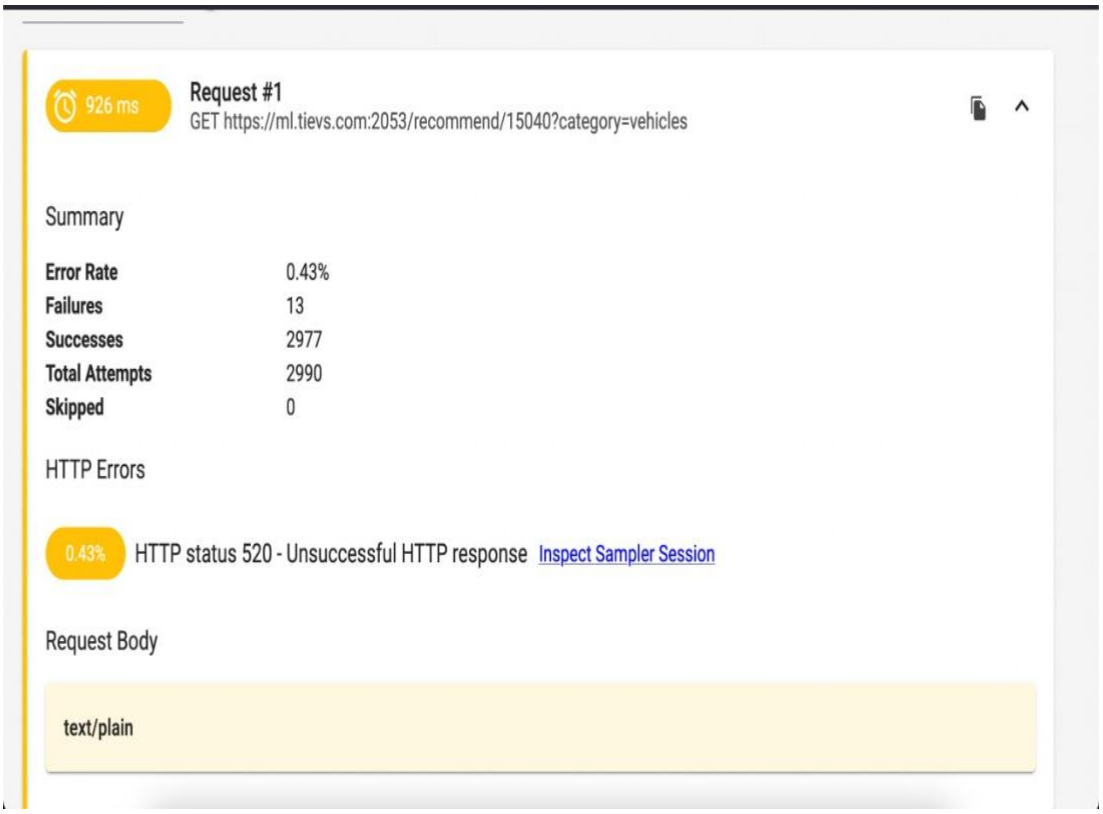


Thursday, Sep 30, 14:33:40
- 95% Response Time: 1,682 ms
- 50% Response Time: 988 ms
- Avg. Response Time: 988 ms
- Concurrency: 50

Thursday, Sep 30, 14:33:40
- Throughput: 36.5 rps

Figure 3.2 – Performance graphs

The above-mentioned and visualized metrics and statistics provide strong solitude towards the sustainability, concurrency, scalability, accuracy, maintainability, performance, and efficiency of the implemented solution and how the incorporated novel technologies work collaboratively and collectively to produce the end results with more than initially expected utilities.


926 ms

Request #1

GET https://ml.tievs.com:2053/recommend/15040?category=vehicles

Summary

Error Rate	0.43%
Failures	13
Successes	2977
Total Attempts	2990
Skipped	0

HTTP Errors

0.43% HTTP status 520 - Unsuccessful HTTP response Inspect Sampler Session

Request Body

text/plain

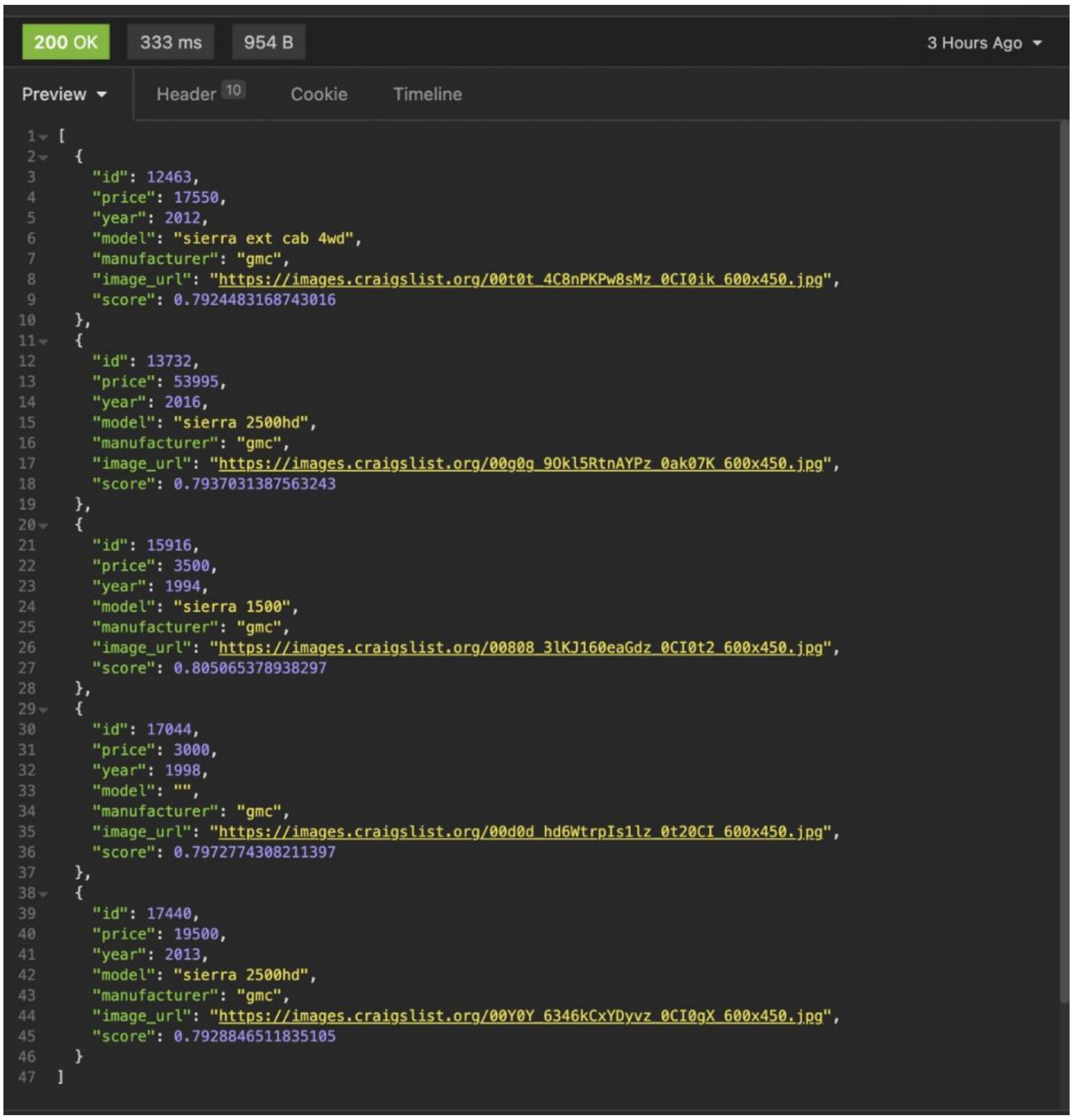
Figure 3.3 – Loadmill test alternative view

Similarly and additionally, the above shown Fig. 3.3 displays a different UI of the Loadmill load testing application, where it shows the total attempts made at the GET request towards the API endpoint shown below for the particular car advertisement id number 15040.

- https://ml.tievs.com:2053/recommend/15040?category=vehicles

The total attempts made is 2990, and from those attempts, 2977 were successful in receiving the desired response. The number of requests that ended up in errors is only 13. Hence, by analyzing those statistics, it is indisputably apparent that the implemented API call and model functionality is executing accordingly without any faults. If an error were to occur, which is extremely rare in this sense, the error would be of the type HTTP 520 status code, which is the code for an unsuccessful HTTP response. The request body is of type plain text. However, the main takeaway here is the fact that the error occurrence probability is vastly depleted.

For more clarification, below Fig. 3.4 shows the Postman result acquired in JSON format of the classified advertisements that have a higher similarity score from the cosine similarity calculation. The request was successfully executed as shown by the 200 status code, and the distinctive quality of it is that the recommendation is provided within a swift time period as only 333 milliseconds. That can be known as another major objective of implementing the discussed recommendation system for the Tievs application.

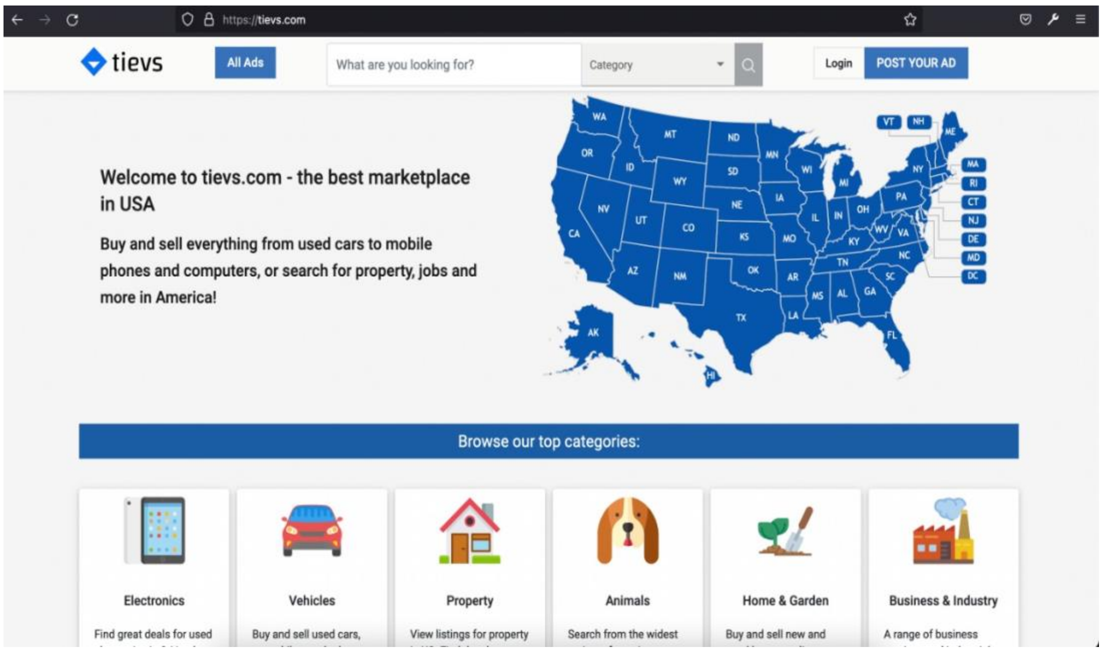


```
[
  {
    "id": 12463,
    "price": 17550,
    "year": 2012,
    "model": "sierra ext cab 4wd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00t0t_4C8nPKPw8sMz_0CI0ik_600x450.jpg",
    "score": 0.7924483168743016
  },
  {
    "id": 13732,
    "price": 53995,
    "year": 2016,
    "model": "sierra 2500hd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00g0g_9Okl5RtnAYPz_0ak07K_600x450.jpg",
    "score": 0.7937031387563243
  },
  {
    "id": 15916,
    "price": 3500,
    "year": 1994,
    "model": "sierra 1500",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00808_3lKJ160eaGdz_0CI0t2_600x450.jpg",
    "score": 0.805065378938297
  },
  {
    "id": 17044,
    "price": 3000,
    "year": 1998,
    "model": "",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00d0d_hd6WtrpIs1lz_0t20CI_600x450.jpg",
    "score": 0.7972774308211397
  },
  {
    "id": 17440,
    "price": 19500,
    "year": 2013,
    "model": "sierra 2500hd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00Y0Y_6346kCxYDyvz_0CI0gX_600x450.jpg",
    "score": 0.7928846511835105
  }
]
```

Figure 3.4 – Postman top similar ad result

After accurately obtaining the necessary results from the developed Google USE statistical model and the Golang implemented API, the next task was to scrupulously design and develop the front-end application using Angular, Typescript, and Material UI feature designing library. However, the development and components being implemented were to be precisely integrated with the ML model and API endpoints without any arising misconceptions. Therefore, as important as the user-friendly UI effect may seem, a correct amalgamation of all technical components was equally essential and a challenge that had to be won over.

All things considered, in the end, this research was able to achieve that aspect of the system as well, although having to face notable difficulties which were overcome due to dedication and conscientiousness. Fig. 3.5 displayed below shows the 'tievs.com' URL navigated home page of the implemented Tievs application. The USA map was built completely by the use of coding and not importing from another third-party source. Although for this research's scope, car classified were selected for analysis expansion, the main target is to build a sophisticated system having the propensity to cater to other types of classifieds such as property, animals, home & garden, electronics, business & industry, etc.



Welcome to tievs.com - the best marketplace in USA

Buy and sell everything from used cars to mobile phones and computers, or search for property, jobs and more in America!

Find great deals for used

Buy and sell used cars,

View listings for property

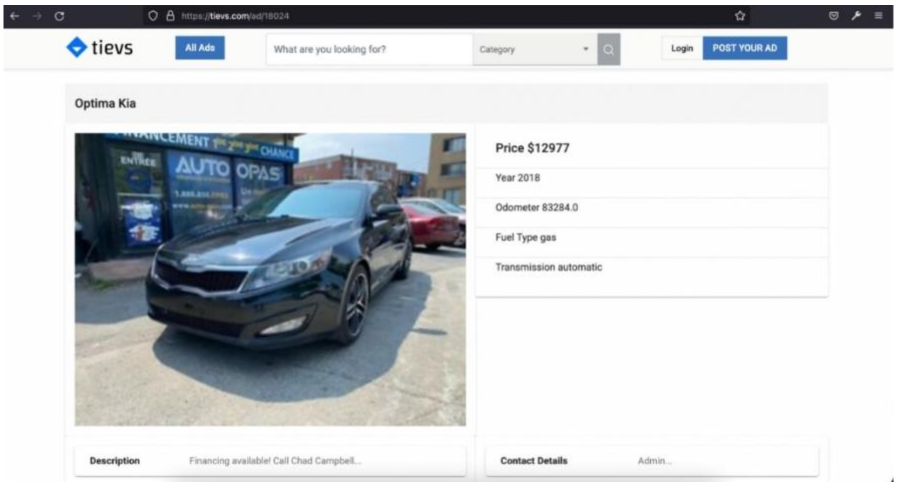
Search from the widest

Buy and sell new and

A range of business

Figure 3.5 – Tievs home page

Next, the UI view designed for an already selected car classified advertisement is being elicited by Fig. 3.6. In that view, only the necessary car characteristics are displayed along with a concise description provided regarding the vehicle from the own words of the person selling it. The seller's contact details are also being unveiled as well. Nonetheless, the UI design is being continuously reconditioned since the production goal is to realistically deploy the web application among its target market. Fig. 3.7 is the exhibit of how similar advertisements, which were recommended to the customers in a user-centric manner through the necessary API function invocations and model executions, are presented.

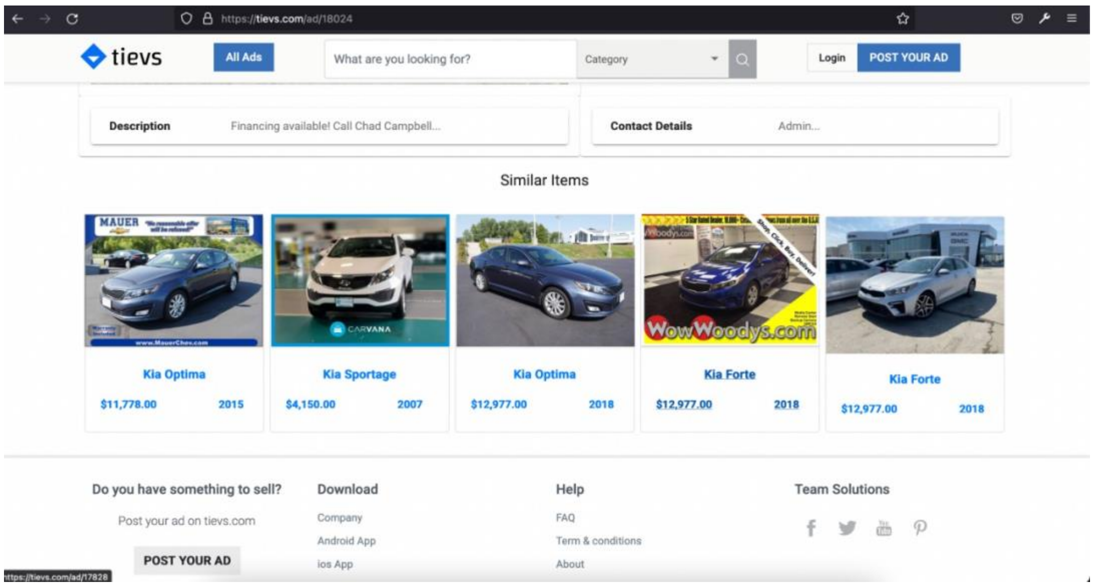


Optima Kia

Price $12977
Year 2018
Odometer 83284.0
Fuel Type gas
Transmission automatic

Description Financing available! Call Chad Campbell...
Contact Details Admin...

Figure 3.6 – Single classified ad view


https://tievs.com/ad/18024
tievs
All Ads
What are you looking for?
Category
Login
POST YOUR AD

Description Financing available! Call Chad Campbell...
Contact Details Admin...

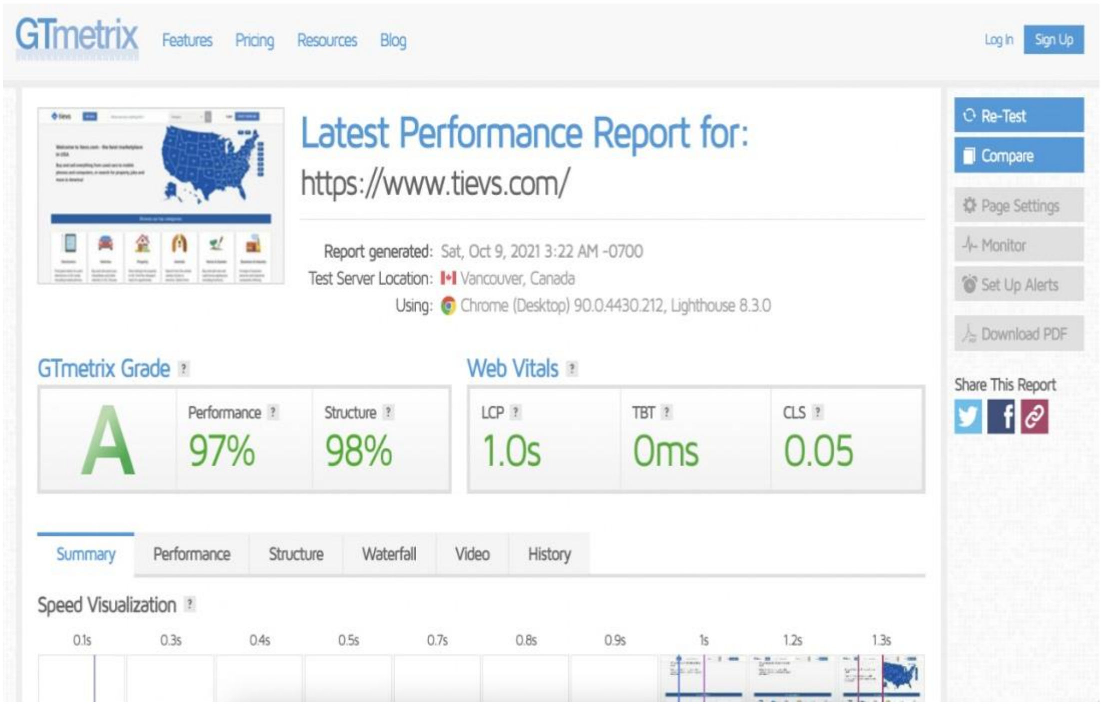
Similar Items

Kia Optima	Kia Sportage	Kia Optima	Kia Forte	Kia Forte
$11,778.00 2015	$4,150.00 2007	$12,977.00 2018	$12,977.00 2018	$12,977.00 2018

Do you have something to sell?
Post your ad on tievs.com
POST YOUR AD
Download
Company
Android App
ios App
Help
FAQ
Term & conditions
About
Team Solutions
https://tievs.com/ad/17828

Figure 3.7 – Top similar classified advertisement view

The following Fig. 3.8 and 3.9 depict the GTmetrix report relating to the final integrated Tievs application, with the recommendation system and all other three components combined. The metrics shown display the high capability and effectiveness of the fully-functional Tievs classified advertising portal.



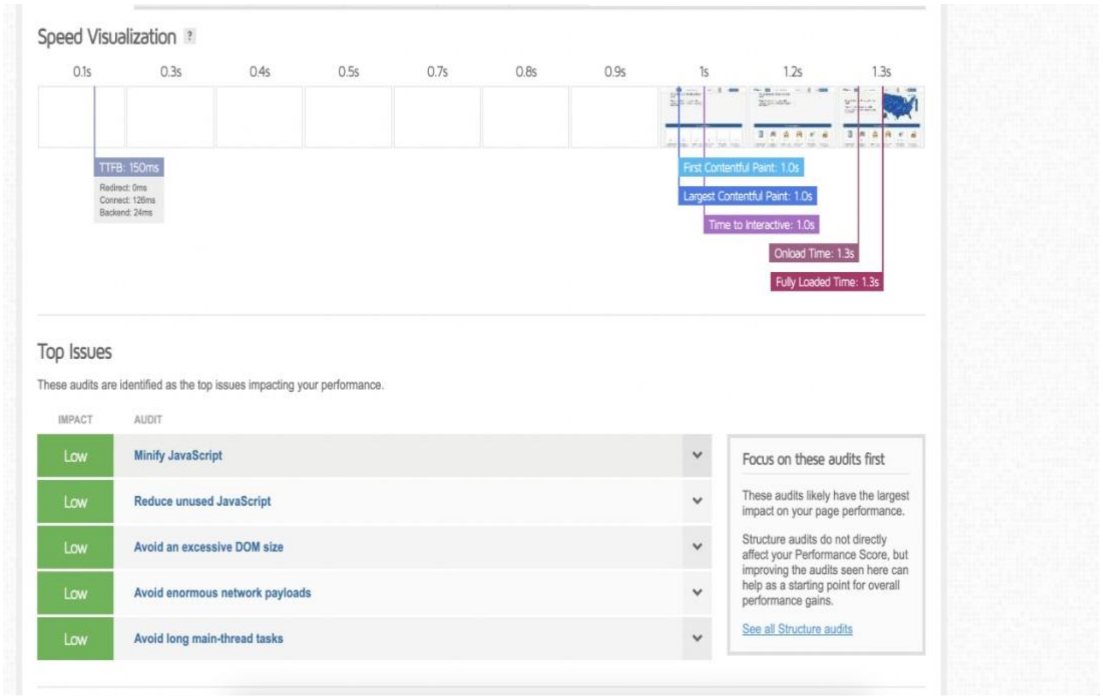
Latest Performance Report for:
https://www.tievs.com/

Report generated: Sat, Oct 9, 2021 3:22 AM -0700
Test Server Location: Vancouver, Canada
Using: Chrome (Desktop) 90.0.4430.212, Lighthouse 8.3.0

GTmetrix Grade: A — Performance 97% — Structure 98%

Web Vitals: LCP 1.0s — TBT 0ms — CLS 0.05

Figure 3.8 – GTmetrix Tievs performance report part 1



Top Issues

These audits are identified as the top issues impacting your performance.

IMPACT	AUDIT
Low	Minify JavaScript
Low	Reduce unused JavaScript
Low	Avoid an excessive DOM size
Low	Avoid enormous network payloads
Low	Avoid long main-thread tasks

Focus on these audits first

These audits likely have the largest impact on your page performance.

Structure audits do not directly affect your Performance Score, but improving the audits seen here can help as a starting point for overall performance gains.

See all Structure audits

Figure 3.9 - GTmetrix Tievs performance report part 2

3.2 Research Findings

This research was fundamentally conducted to identify, speculate, analyze and determine the most optimized recommendation system that is developed using the most innovative technologies and procedures to provide the stakeholders the best services and functionalities without having to face any inconveniences.

The existing systems are built using the most conventional technologies out there within the industry, and they have rarely tried to incorporate any technologies that are rapidly updating day by day. The main aim was to actually perform far better than the existing recommendation systems, either they were built using the content-based, collaborative, or hybrid structure.

For this research's own contentment, the initially stated goals and objectives were acclaimed thoroughly and exceptionally. The significant research findings secured are expressed below in point form for clear and concise interpretation.

- The cold start problem is an important common problem that needed to be addressed within the prevailing systems, and this implemented solution was far more capable of eliminating it since it is built based on content-based filtering but ability-wise enhanced using the collaborative functionality of cosine similarity calculation and intelligent Google USE model.

- Not using user information such as likes, dislikes, comments, feedback, browsing history, etc., enabled new advertisements to be considered for the recommendation process as well. That was attained by this research component and is a notable finding. Else, the newly published advertisements won't be accounted for the recommendation simultaneously required by another customer, which makes the process less accurate and outdated.

- Real-time recommendations could be provided to the consumers without much trouble due to that reason. Parallelly submitted advertisements were examined, and the necessary embeddings were created promptly to facilitate the dynamic, updated suggestive mechanism, which happens continuously according to user requests received. This result gained by this research component, which is similar to an implicit model retraining circumstance, is truly a remarkable found that distinguishes the engine from other competitors.

- Moreover, due to discontinued usage of customer personal information, it was found that the Tievs application incorporating this recommendation engine was free of any possible user privacy-related allegations made by particular law enforcement. It was firmly evident that such penalties would not be able to cause any ructions.

- The implementation of the model was quite effortless due to the use of the pre-trained model of the TensorFlow library was utilized. The USE model is implemented with a DAN and a feed-forward neural network for natural language processing purposes, and it made every process drastically advanced. Many other recommender systems may incorporate complex algorithms to work with, and the end implementations and integrations were found even more complex and tedious for maintenance and fault preventions.

- Flexibility to be improved furthermore was more so cumbersome in previously reviewed work since future developers contributing for the enhancements found the source code analysis quite meddling. That won’t be an issue with this research component since the implementation is transparent in nature and easily comprehensible.

- Both new embedding generation for the freshly published advertisements and cosine similarity calculation, including that ad itself, is happening concurrently within the recommendation system. This concurrency could only be achieved up to a spectacular level with the usage of Golang only. Although on the first try, python language implemented concurrency was utilized, the performance was not up to a satisfactory standard. Hence the decision was taken to proceed with Golang, which in return, bestowed success upon the research with super-fast and efficient Goroutines that can handle concurrent tasks.

- The concurrency itself and the speed it delivers is immensely astonishing and a great find for this research analysis. Even from the start of this experimentation procedure, one of the main focused aspects was the speed of the recommended procedures considering the fact that it handles the calculations of top similar ads from a complex dataset having many records. That high speed was effortlessly explored and encountered at the end.

- Since the solution is deployed in the cloud (DigitalOcean) as droplets, there is a high chance of scalability when required, which can be known as a tempting advantage for such classified advertising applications. Due to this scalability architecture, it was discovered that fault tolerance and disaster recovery mechanisms could also be enacted, which could further improve the recommendation system's stability and continuous availability for the betterment of its consumers.

- Moreover, when regulating this research, it was found that due to the fact that the recommendation engine API and the Google USE model is deployed within an off-site cloud storage environment, the cost of maintaining any essential hardware and utility software is quite low in comparison to an onsite physical server, database, and network management. The cost gap analyzed and calculated between those two types of armature was found at about $230. Hence, the cloud environment was inarguably preferred. The unlimited storage space, competitive edges, backup and data restoring, automatic integration of software, and enhanced reliability were more of the pristine pronouncements through the implemented cloud infrastructure.

3.3 Discussion

The exclusive purpose of this study is to implement an optimized recommendation process to make classified suggestions to the application's customers using a novel technology that has not been previously attempted. Moreover, this research intended to achieve a high scalability and flexibility level for its recommendation engine. Fortunately, after many fastidious implementation procedures and overcoming numerous impediments, the objectives and goals were finally attained. In retrospect, the major results and findings of this research can be interpreted and emphasized.

The presence of the Cold Start problem, which has been prevailing in most collaborative-based recommendation systems, has been omitted through the methodology used in this study since content-based recommendation procedures were undertaken. When the user data such as likes, dislikes, feedback, comments, etc., are not accounted for, the Cold Start problem implicitly moves out of the question. Besides, classified advertising platforms, unlike most other e-commerce applications, do not save customer-related data such as their likes, dislikes, comments, feedback, reviews, and other private preferences, hence the use of collaborative recommendation was not that significant to proceed with this study's major goal. Through the discontinued usage of these user details, some of which could be tagged confidential, Tievs was made immune to possible penalties or fines that would have been effectuated by various law enforcement regarding potential customer data and information disclosing.

Although a complex dataset was utilized for the purpose of training the models, due to computer resource limitations identified during the research scope, the dataset had to be considerably scoped and lessened to 18,040 records. The selection criteria for the dataset filtering was the state of which the used car belonged in its specific classified ad. However, the impact it had when training the model that was being selected was minimal to none. That being said, using an immensely large dataset of about 450,000 records would have led to recommendation faults by the model due to outliers and erroneous description context. Therefore, it was encountered that the declination of the complex data resulted in an appropriate-sized data source to work on without having unnecessary difficulties during model training when creating embeddings of the classifieds.

The implication of the TensorFlow library consisting of Google USE pre-trained model for the task of generating 512-vector embeddings using the DAN architecture and feed-forward neural networks is the most noteworthy objective of this whole research component. This model is quite new within the ML domain, and not much previous research has been conducted as well. Hence, it was incorporated for this research for the description column embedding creation to facilitate the next step of the recommendation engine, which is the cosine similarity calculation. It was found easy to import the necessary libraries and start using the model since it has already been trained by various model training sources such as web pages, wiki pages, forums, question and answer websites, etc., before being acquainted with TensorFlow. After the model had created all embeddings and saved them in a newly constructed column within the dataset table, which was residing in the PostgreSQL database, it was imported into the Golang API for further integration.

The real-time recommendation was designed and developed for the customers of the Tievs application by incorporating concurrency routines that were implemented through the Golang API as Goroutines. In usual recommendation scenarios, when a new advertisement is submitted into the system, and different customers access the application from another session, they would not receive newly published classified included recommendations instantly. It may take a few minutes or hours for all calculations to happen and the respective suggestions to arise. That is why the real-time recommendation is preferred, especially in online classified platforms.

This real-time recommendation characteristic was not present or identified when observing past research done [3]-[11]. Hence, it can be known as a novel and innovative concept that was primarily introduced by this Tievs classified recommendation system itself. Although python language was considered for the implementation of this concurrency enabled API at the very beginning of the research, later on, after thorough analysis was done, it was confirmed that the most suitable programming language would be Go language due to its fast, simple coding structure that enables robust API development. Even Java threads were found less efficient compared to Goroutines, as shown in the ‘Methodology’ chapter above.

Two main processes happen within the API simultaneously hence why the concurrency was considered mandatory. Both new embedding creation when each new advertisement is submitted and cosine similarity calculation of all the classifieds in the database (both new and old) are happening from the duties of the API, and this coding architecture is unprecedented and ingenious. For the new embedding generation process, the API must call another separate python file containing the functionality for the USE model’s embedding creation regarding only a single advertisement. That new embedding must then be stored in the database while the response is directed to the API again. If another customer clicks on an ad, the cosine similarity calculation happens from within the API again as well while considering the new ad submitted from the other end. All of these processes must be handled within a short period of time by the API hence the reason it should be implemented robustly. By analyzing the Loadmill’s load testing result of 926ms response time and 0.43% error rate for 50 concurrent user sessions, it is evident that the API was constructed appropriately through this research analysis.

At first, Loadmill only allowed for five concurrent user sessions to be created. Therefore, in order to activate 50 user sessions, the Golang API had to be redefined using an additional code segment. During the load testing, the response time received is highly satisfactory since this time-lapse is responsible for the execution of the above-mentioned processes of the API. The cosine similarity is calculated under 1 second was absolutely remarkable in the sense of a large number of embeddings to sectionize and contemplate on. The top similar five items are to be given as the recommendation or response from the API. Its error rate is a far more astonishing result as well, being even less than 1%. It means that the classified ads recommended for the customers are highly accurate in the selection, and they are exactly the classifieds that the customers are hoping to view when browsing through all other advertisements. These type of metrics have not been evaluated and compared in other research [3]-[11], and load testing's of any sort were not executed, unlike in this research.

The deployment of the model included API, and all its dependent files within a cloud environment were done at the end of all critical component implementation. The droplet, which was purchased from the IaaS platform DigitalOcean, provided high availability, scalability, and flexibility for the API and the Tievs front-end application as well. Such courses of action were not regulated or organized by existing studies that focused on recommendation engine productions [3]-[11]. Not only about the engine establishing factor, but also the deployment aspects must be addressed without negligence if the recommendation system needs to satisfy its main task of recommending without availability-wise interruptions. Otherwise, the recommendation engine itself would be worthless, and that is why this research hoped to attempt on that notion and consequently succeeded.

As a summarization statement considering all the above-stated facts and information, this scoped research has been able to achieve its predefined goals and objectives in the most convenient and efficient manner. However, room for further improvements still remains. Although collaborative filtering alone was refused from this study, a hybrid version combining both content-based and collaborative recommendations is encouraged since it may prospect a stronger suggestive mechanism. Moreover, the Google USE model's light version could be analyzed for its capability in embedding generation as well. This research used the DAN architecture for the USE model, although the transformer architecture could be exploited for its own efficiency. Apart from that, 926ms response time can further be reduced by using a caching method so that whenever a user clicks on a frequently accessed advertisement, the latency for that ad's specific recommendation received was reduced. The most promising future work of this study would be to implement an additional system procedure to accommodate multiple languages formulated classified ad recommendations, although it will be tremendously laborious to carry through.

3.4 Summary of Student’s Contribution

Table 3.1 – Summary of personal contribution

Member	Component	Tasks
R. A. D. Prathapa	Optimized Classified Advertisement Recommendation System	▪ Investigate suitable Machine Learning and Natural Language Processing algorithm types available with regard to recommendation systems. ▪ Exploring and collection of the most convenient dataset that could be used for model training. ▪ Preprocessing the dataset necessarily prior to model utilization ▪ Finding the most suitable model for recommendation engine development. ▪ Identifying and importing the necessary dependencies to import the model and training of the model. ▪ Successful model serialization and deserialization. ▪ Meticulous implementation of the API using Golang while providing for concurrency and handling of all essential tasks through function invocations. ▪ Build relevant client and server-side components to visualize the analyzed and predicted customer-specific search results and retrain the model using new information acquired from the application database (new embeddings). ▪ Smooth integration between front-end and back-end services.

4 CONCLUSIONS

This section intends to summarize and reminisce all the tasks and implementation procedures followed from the beginning of this research to its very end. First and foremost, the inspiration for this research component derives from the continuously rising fame of online classified advertising systems rather than other physical sources such as newspapers, brochures, leaflets, magazines, booklets, etc., when it comes to marketing ads.

Higher availability, affordability, accessibility, scalability, user-friendly interfaces when compared to physical materials. The usage has simply doubled from what was known 15 years before. Craigslist, Facebook marketplace, Olx, CarDheko, Quikr are only some of the major classified advertising platforms which were often visited by customers to either buy any necessary items, despite that item being used or brand new, to even to sell any items personally. Both the browsing and viewing classifieds and submission of new advertisements are made easy up to a considerable level through these portals.

That being said, the quality recommendation is an essential aspect that a classified advertising portal should contain considering the fact that people are browsing through the platform in order to find some classified which is relevant to them so that ultimately, a purchase will be made through the application while uplifting its reputation among the targeted audience as well. Hence, classified recommendations should be provided to the customers, swiftly as possible. However, there are still a few problems remaining currently, which makes these recommendation systems less effective and summon enhancements.

Moreover, a lesser amount of experimentations were done towards classified recommendation engine improvements. This research was primarily focused on addressing and solving those problems, in addition to analyzing the past research which has been done regarding these recommender systems and employing a different perspective to the existing literature domain. The novelty is effectuated within this research component since none of the past research (and more other studies) that was analyzed [3]-[11] did try to investigate a suitable recommendation engine that will perform quite efficiently when combined with a classified advertising application. Recommendations regarding movies, songs, social media posts, groupware networking posts, and vivid e-commerce sites have been studied most commonly while avoiding the domain of classified advertising, and that is one of the major reasons for the initiation of this dissertation's established research.

There are a few types of methods to implement recommendation systems, and most existing platforms have been built using the collaborative filtering technique. Nonetheless, it still had issues within it, such as the Cold Start problem, meaning that when a new item is being added into the system, and it requires some user-related data such as likes, dislikes, comments, reviews, feedback, etc., it was not eligible to be selected for the overall recommendation procedure due to the lack of user-relevant information, hence why this research utilized the content-based filtering mechanism as the base design and structure.

Moreover, user information such as the ones stated above is not collected or saved through classified advertising systems since they do not have a specific value of necessity towards them than other recommendation incorporated web applications do. In addition, if those customer data were to be disclosed to external sources without regulated permission, penalties or fines would be made through different law enforcements to the respective application, which is responsible for the information reveal. Hence, applying a content-based filtering mechanism for the classified ad recommendation procedure seemed much more practical, appropriate, and safe as well.

TF-IDF was commonly used by almost all research, at least in some aspect, when executing NLP-related tasks. Opportunities were not provided for novel and rarely used models within most scientific research conducted. Thus, the reason this research used the Google USE model, which has never been utilized for classified ad recommendation tasks or any major recommendation system implementations for that matter. In this study, it was mainly attested for its capabilities and effectiveness in embedding creation and similarity metric generation as an NLP venture. The results were remarkable since the model achieved its mission without much difficulty, from data preprocessing, through model training and embedding creation, to model serialization. Since it was a pre-trained model from the Python library TensorFlow, it was a fundamentally mature model to begin with, and the resource consumption was not much invigorating as well.

If it was a complex model that needed a high memory and a CPU capacity to function, it would have never been entitled to be the main model for embedding creation from the description column data of the dataset involved in this research. The dataset was online available through Kaggle with the name ‘Used cars dataset,’ which initially had approximately 450,000 data records though, for the embedding generation purpose, only about 19,000 records were filtered out by the state column of the dataset (advertisement related state).

Another aspect that was not engrossed in the past recommendation investigations was the feasibility of real-time recommendations. That basically means every new, updated, or deleted advertisement is being considered immediately for the consequent recommendation procedures within the application, without facing interruptions or bottlenecks in the smooth approach. In order to implement such a mechanism, this research decided to implement a formidable API having the most vital feature, concurrency.

Through concurrency, it was possible to handle both the embedding creation process and the cosine similarity calculation to provide the requesting end with the top five similar advertisements recommended. Even when a new ad was submitted, the unique embeddings for that needed to be created quickly so that it could be considered for the similarity calculation process. This concept was successfully achieved by using Golang to develop the API since by using Python at the beginning of this research; it was encountered that the concurrency was not up to a complimentary level due to high latency executions.

Golang's lightweight coding architecture and support for fast, robust API development made it conceivable. Having a 926ms response time and an error rate of just 0.43% for 50 concurrent user sessions from the Loadmill tool for load testing regarding the cosine similarity calculation of the embeddings relating to whichever clicked (specified) ad is absolutely impressive. The API with the model was finally deployed in the DigitalOcean cloud environment purchased droplet, enabling for scalability and higher 24/7 availability, along with the front-end Tievs application.

A devastating issue that remains currently in recommendation engines is that they are too complex, and the execution is of heavy load and traffic, which sometimes become too difficult for the deployed server to handle. Hence, additional cloud purchasing may need to happen, ignoring the cost-effectiveness characteristics of such systems since many times, the expenditure is overflowing, and they do not exhibit entirely fair purchasing amounts as well. That issue was thoroughly addressed by this research through the amalgamation of the Google USE model with the cosine similarity calculation and the Golang API. Those designing and development structures made it possible for the purchased server to handle heavy traffic load at a single time without facing any faults or congestions. If the hosting server is functioning healthily, the rest of the API and model executions would also happen uninterrupted, enabling the customer to browse and surf through the classifieds on the client-side application and receive all the relevant recommendations according to their searching advertisement history. That is exactly what this research victoriously accomplished.

Nonetheless, further destined leverage, most likely the paths mentioned in the upper part 'Discussion,' is plausible and permissible regarding this study by any enthusiastic researchers, developer, or experimenter. In condensation, this research has analyzed and investigated all identifiable problems within the domain of classified advertising recommendations and fulfilled the initially signified research objectives through the meticulous methodology followed to implement a technologically novel design and structure infused optimized recommendation engine to cater to the classified ad browsing requirements of the customers more efficiently than currently being served.

REFERENCE LIST

[1] Business Insider. 2021. CHART OF THE DAY: Newspaper Billions Become Craigslist Millions. [online] Available at: https://www.businessinsider.com/chart-of-the-day-craigslist-vs-newspaper-2009-6 [Accessed: 23-Feb-2021].

[2] B. P. P. C. Limited, “Kaidee to charge for some ads,” Bangkok Post.

[3] “Online classified ads: Digital, dynamic, and still evolving,” McKinsey &amp; Company, 03-Apr-2018. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/online-classified-ads. [Accessed: 23-Feb-2021].

[4] S. Jones, “Online classifieds,” Pew Research Center: Internet, Science &amp; Tech, 30-May-2020. [Online]. Available: https://www.pewresearch.org/internet/2009/05/22/online-classifieds/. [Accessed: 06-Oct-2021].

[5] D. Das, L. Sahoo, and S. Datta, “A Survey on Recommendation System,” Int. J. Comput. Appl., vol. 160, pp. 6–10, 2017, DOI: 10.5120/ijca2017913081.

[6] R. Singla, S. Gupta, A. Gupta, and D. K. Vishwakarma, “FLEX: A Content-Based Movie Recommender,” in 2020 International Conference for Emerging Technology (INCET), 2020, pp. 1–4, DOI: 10.1109/INCET49848.2020.9154163.

[7]A. Pal, P. Parhi, and M. Aggarwal, “An improved content-based collaborative filtering algorithm for movie recommendations,” in 2017 Tenth International Conference on Contemporary Computing (IC3), 2017, pp. 1–3, DOI: 10.1109/IC3.2017.8284357.

[8] B. Kostek, “Listening to Live Music: Life Beyond Music Recommendation Systems,” in 2018 Joint Conference - Acoustics, 2018, pp. 1–5, DOI: 10.1109/ACOUSTICS.2018.8502385.

[9] S. Shaikh, S. Rathi, and P. Janrao, "Recommendation System in E-Commerce Websites: A Graph-Based Approached," 2017 IEEE 7th International Advance Computing Conference (IACC), Hyderabad, India, 2017, pp. 931-934, DOI: 10.1109/IACC.2017.0189.

[10] G. Prabowol, M. Nasrun, and R. A. Nugrahaeni, "Recommendations for Car Selection System Using Item-Based Collaborative Filtering (CF)," 2019 IEEE International Conference on Signals and Systems (ICSigSys), Bandung, Indonesia, 2019, pp. 116-119, DOI: 10.1109/ICSIGSYS.2019.8811083.

[11] F. Sun, Y. Shi and W. Wang, "Content-Based Recommendation System Based on Vague Sets," 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, China, 2013, pp. 294-297, DOI: 10.1109/IHMSC.2013.218.

[12] R. Wita, K. Bubphachuen, and J. Chawachat, "Content-Based Filtering Recommendation in Abstract Search Using Neo4j," 2017 21st International Computer Science and Engineering Conference (ICSEC), Bangkok, Thailand, 2017, pp. 1-5, DOI: 10.1109/ICSEC.2017.8443957.

[13] H. Xue and D. Zhang, "A Recommendation Model Based on Content and Social Network," 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2019, pp. 477-481, DOI: 10.1109/ITAIC.2019.8785729.

[14] H. Chen, Y. Wu, M. Hor, and C. Tang, "Fully content-based movie recommender system with feature extraction using neural network," 2017 International Conference

on Machine Learning and Cybernetics (ICMLC), Ningbo, China, 2017, pp. 504-509, DOI: 10.1109/ICMLC.2017.8108968.

[15] M. Alodadi and V. P. Janeja, "Similarity in Patient Support Forums Using TF-IDF and Cosine Similarity Metrics," 2015 International Conference on Healthcare Informatics, Dallas, TX, USA, 2015, pp. 521-522, DOI: 10.1109/ICHI.2015.99.

[16] “Most 5 Valuable Benefits of Agile Methodology | Blog – Denysys Corporation.” https://www.denysys.com/blog/5-benefits-of-agile-methodology/ (Accessed Feb. 25, 2021).

[17] News, E., Recommendation Engine Market Size Expected to Reach USD 16.13 Billion at CAGR of 39.8%, by 2026. [online] EIN News. Available at: <https://www.einnews.com/pr_news/552417708/recommendation-engine-market-size-expected-to-reach-usd-16-13-billion-at-cagr-of-39-8-by-2026> [Accessed 30 September 2021].

[18] Chaudhary, A., *Universal Sentence Encoder Visually Explained*. [online] Amit Chaudhary. 2021. Available at: <https://amitness.com/2020/06/universal-sentence-encoder/> [Accessed 4 October 2021].

[19] Medium. Benchmark databases in Docker: MySQL, PostgreSQL, SQL Server. 2021.[online] Available at: <https://itnext.io/benchmark-databases-in-docker-mysql-postgresql-sql-server-7b129368eed7> [Accessed 4 October 2021].

[20] “My Story with Go, Python, and Benchmarks,” *Django Stars Blog*, 05-Aug-2021. [Online]. Available: https://www.djangostars.com/blog/my-story-with-golang/. [Accessed: 04-Oct-2021].

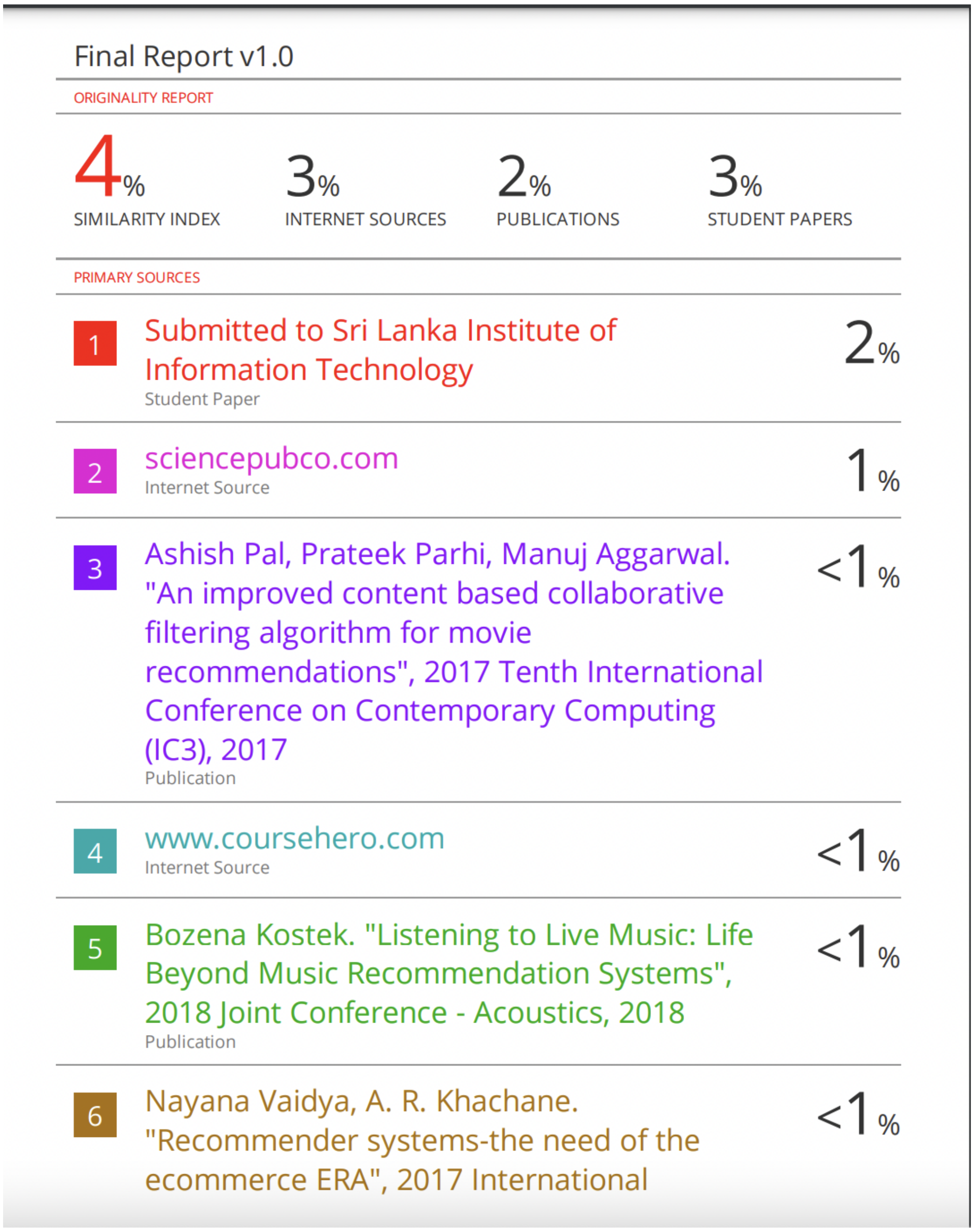
[21] “HTTP Performance Java (Jersey) vs. Go vs. Node.js – Full-Stack Feed,” *Full*. [Online]. Available: https://fullstackfeed.com/http-performance-java-jersey-vs-go-vs-node-js/. [Accessed: 04-Oct-2021].

[22] N. Togashi and V. Klyuev, “Concurrency in Go and Java: Performance analysis,” *2014 4th IEEE International Conference on Information Science and Technology*, 2014.

[23] Admin, “Goroutines in Go (Golang),” *Welcome To Golang By Example*, 25-Nov-2020. [Online]. Available: https://golangbyexample.com/goroutines-golang/. [Accessed: 04-Oct-2021].

APPENDIX

Plagiarism report



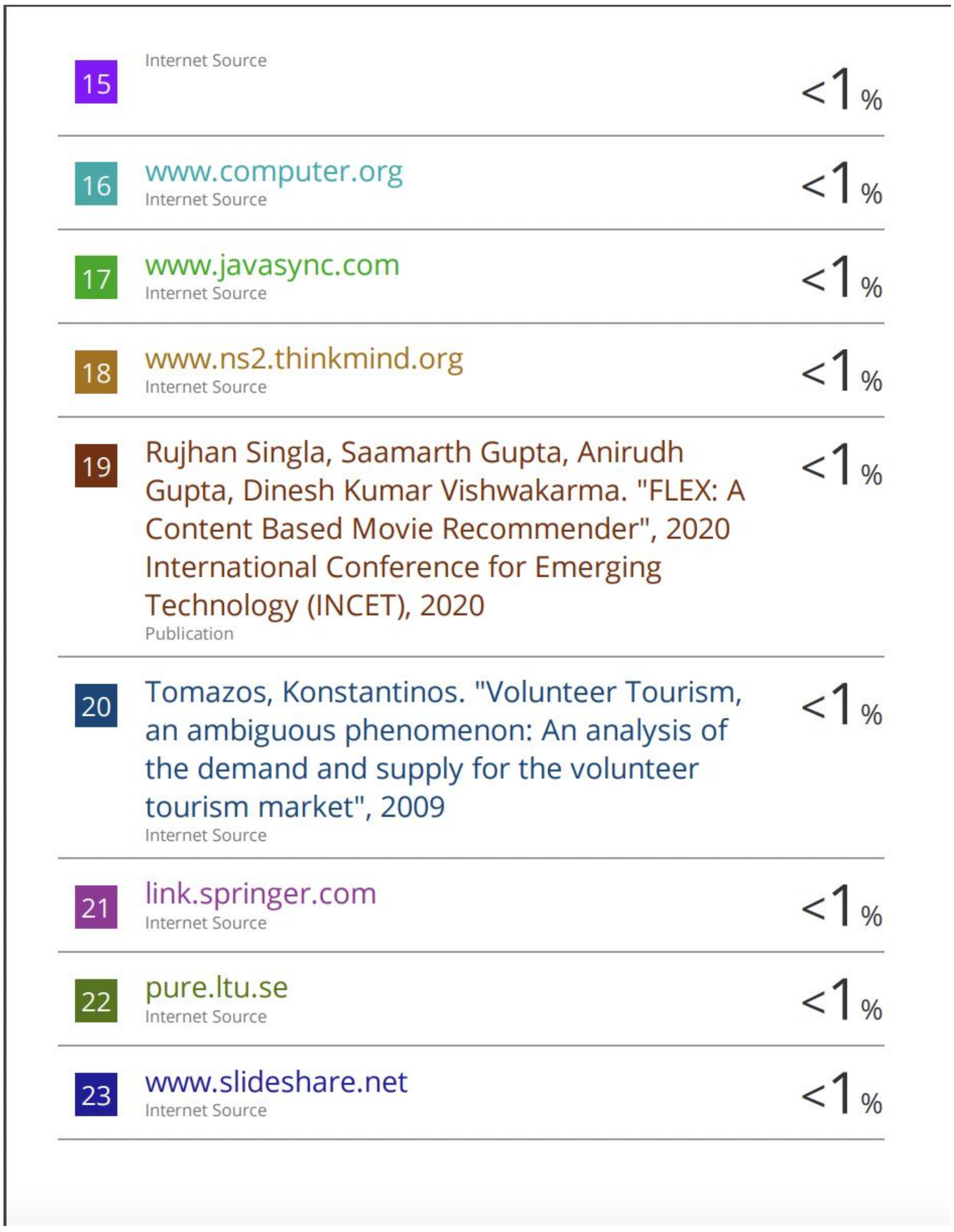
Final Report v1.0

ORIGINALITY REPORT

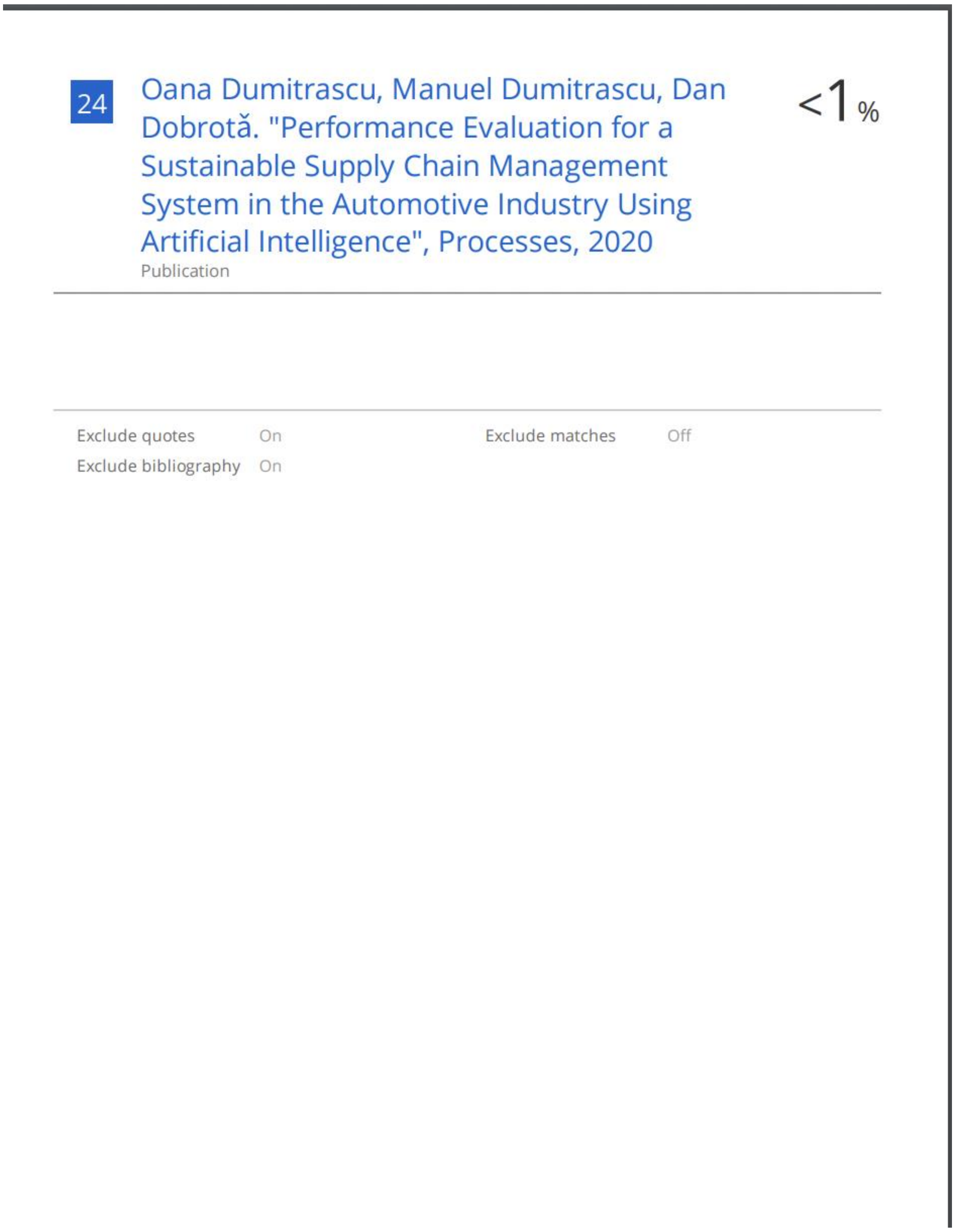
4%	3%	2%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Sri Lanka Institute of Information Technology Student Paper	2%
2	sciencepubco.com Internet Source	1%
3	Ashish Pal, Prateek Parhi, Manuj Aggarwal. "An improved content based collaborative filtering algorithm for movie recommendations", 2017 Tenth International Conference on Contemporary Computing (IC3), 2017 Publication	<1%
4	www.coursehero.com Internet Source	<1%
5	Bozena Kostek. "Listening to Live Music: Life Beyond Music Recommendation Systems", 2018 Joint Conference - Acoustics, 2018 Publication	<1%
6	Nayana Vaidya, A. R. Khachane. "Recommender systems-the need of the ecommerce ERA", 2017 International	<1%



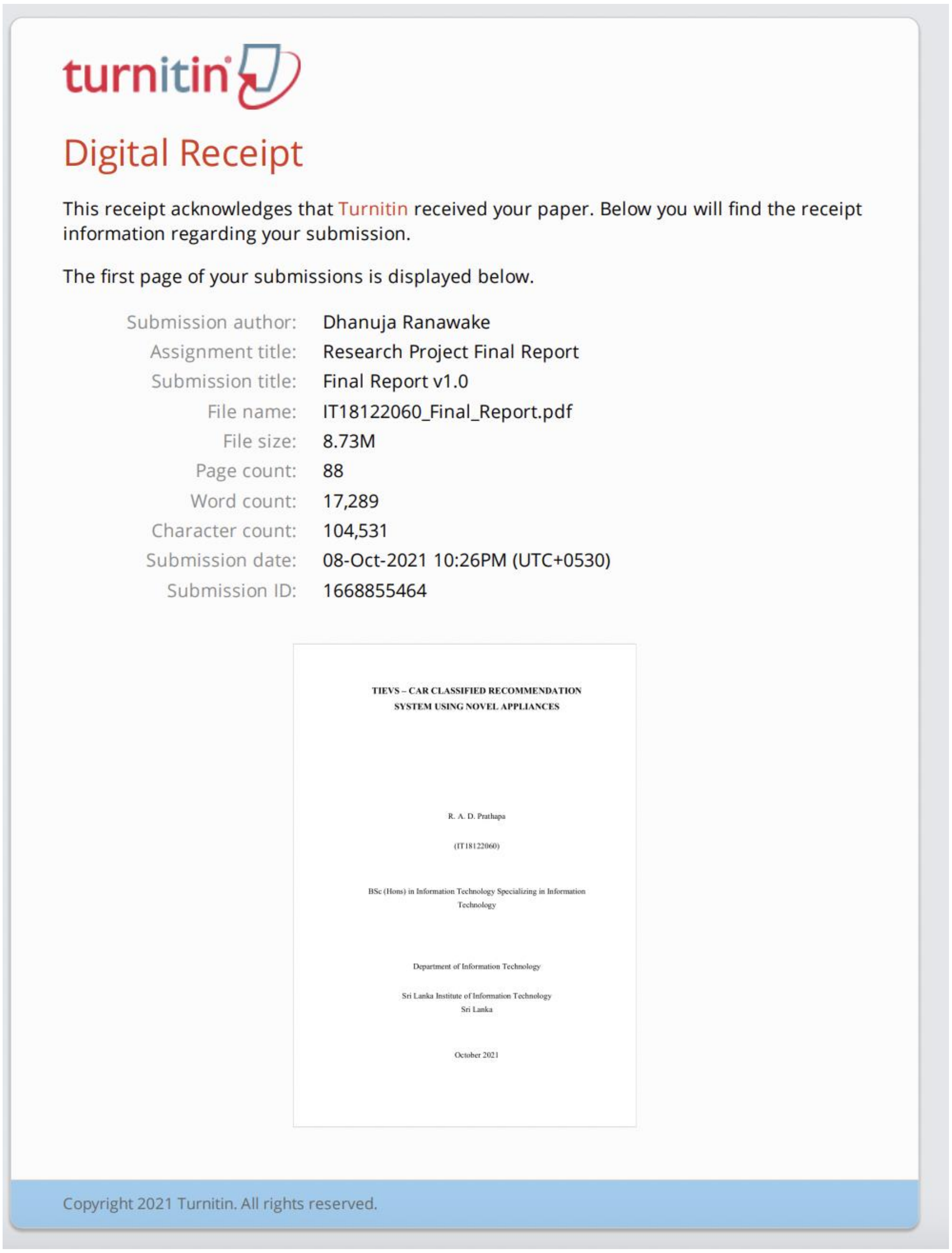
15	Internet Source	<1 %
16	www.computer.org Internet Source	<1 %
17	www.javasync.com Internet Source	<1 %
18	www.ns2.thinkmind.org Internet Source	<1 %
19	Rujhan Singla, Saamarth Gupta, Anirudh Gupta, Dinesh Kumar Vishwakarma. "FLEX: A Content Based Movie Recommender", 2020 International Conference for Emerging Technology (INCET), 2020 Publication	<1 %
20	Tomazos, Konstantinos. "Volunteer Tourism, an ambiguous phenomenon: An analysis of the demand and supply for the volunteer tourism market", 2009 Internet Source	<1 %
21	link.springer.com Internet Source	<1 %
22	pure.ltu.se Internet Source	<1 %
23	www.slideshare.net Internet Source	<1 %



24 Oana Dumitrascu, Manuel Dumitrascu, Dan Dobrotǎ. "Performance Evaluation for a Sustainable Supply Chain Management System in the Automotive Industry Using Artificial Intelligence", Processes, 2020 <1%

Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		



turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Dhanuja Ranawake
Assignment title:	Research Project Final Report
Submission title:	Final Report v1.0
File name:	IT18122060_Final_Report.pdf
File size:	8.73M
Page count:	88
Word count:	17,289
Character count:	104,531
Submission date:	08-Oct-2021 10:26PM (UTC+0530)
Submission ID:	1668855464

TIEVS – CAR CLASSIFIED RECOMMENDATION SYSTEM USING NOVEL APPLIANCES

R. A. D. Prathapa

(IT18122060)

BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021