Vectorizing your logs supports failure triage in many ways

Abstract

Engineers triage failures mainly rely on reviewing the failure logs. Lacking tools to aggregate or search logs efficiently, engineers suffer from triaging failures, especially for those duplicates.

In this paper, we propose a new approach called "vectorizing" including processing, saving, and using logs to support failure triage. Logs will be transformed into semantic vectors by a state-of-the-art language model, saved into a vector database, and utilized in many ways. We describe the design, the solution, the use cases, and we evaluate the performance. At last, we share the plan for future works which can benefit most of our R&D engineers.

1. Introduction

As a high-productivity CI/CD-style company, engineers run large quantities of tests every day. In the past year, engineers met #{0} test failures on Nimbus[1] (the largest test platform in VMware). More specifically, we observed 25k test failures from a closely collaborating Test team.

Duplicate Test Failures and PRs

Triaging a large amount of failures takes great effort. Some efforts are necessary as engineers need to deep-dive the root cause. But in the meantime, some of the efforts can be avoided: duplicate failures. As multiple reasons can lead to duplicate failures, it's hard to avoid that. One of the reasons is that our product is complex and has cross-component interconnection, a bug can be revealed in multiple components. For example, a vSAN DOM bug may lead to vSAN CLOM and vSAN MGMT failures. What's more, a cross-GEO test team may also meet duplicate failures -- even though they might be testing the same component, as their test cases vary, they could reveal the same bug with different test cases.

Bugzilla[2] is a company-wise bug tracking and communication platform. In the past year, we observed 210k PRs that were raised in Bugzilla, and specifically 2.5k PRs were from ESXi. Among those ESXi PRs, 2k were labeled as fixed and resolved while 0.5k were labeled as duplicated. More specifically, an average 18% duplicate PR rate could be summarized from one of the closely collaborating teams with us(under ESXi) in 2023.

Inconvenience when querying logs

The scale of the logs is another problem that slows down the bug-triaging process.

Support Bundle, the collection of the logs from the target VM, is widely used for ESXi, vSAN, or vCenter Teams to deep-dive the root cause of test failures. By averaging the size of thousands of ESXi Support Bundles, we observe that it takes 1.3G for each ESXi Support Bundle, which includes tens of millions of logs. Engineers need to search for failures by using keywords from Support Bundles. Even with the help of tools like Humbug[3], it usually takes minutes to finish each query(4.2.2). What's more, vectorizing logs enables engineers to search logs by semantic similarity, which can help engineers to get close to the root-cause more efficiently.

Goal

The paper describes the new failure triage info-retrieval system that aims to deduplicate failures by vectorizing and leveraging vectorized logs. It can also offer semantic similarity search which outperforms plain-text search in both semantic and computational sight. We orchestrated the paper in these chapters. We will explore the fundamental tools we use for the failure-triage info-retrieval system in the Related Works Chapter. We will take a deep look at the system from the Design and several applications we have built or plan to build that support multiple failure triage scenarios. We will display the experimental performance results in the Performance Chapter. Finally, we will share the plan that can benefit most of R&D engineers.

2. Related Works

In this chapter, we will go through the fundamental concepts and components of the newly proposed failure-retrieval system. More detail will be described in Chapter 4.

2.1 Vectorizing

Vectorizing, as known as word/sentence embedding, transforms the source word/sentence to a real-valued multi-dimensional vector. Each dimension(bit) of the vector can be considered as a feature that describes the source in a specific domain. It is similar to Eigen-Decomposition in concept -- an event A (film scores for example) can be transformed as $A = Q \wedge Q^{-1}$, where Q is the weight matrix applied on the diagonal matrix $\wedge$ which represents topics. After further squeezing the matrix Q to be a single-dimension vector, it becomes the embedding of the event and each bit of this vector presents the weight on a specific topic. The difference is that, by leveraging deep learning on neural networks, the model learns both topics and weights automatically.

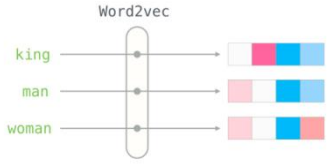

Figure 1. The Illustrate Word2vec [4]

A well-known case of vectorizing is word2vec[5], which embeds word-level targets to vectors -- each word has its own vector. After unsupervised learning, the vectors that represent words can be close to others in the multi-dimensional vector space if the words have similar meanings. The word2vec is usually used in training language models -- compared to a word-to-id vocabulary, using the semantic embedding can help the language model learn quicker and perform better[6]. What's more, as there are many models can do sentence-level embedding, this can also happen to sentences.

Vectorizing has two main benefits. As the vector represents the source to some extent, we can aggregate them based on their semantic similarity. What's more, we can do semantic similarity search -- finding the top similar targets compared to the source. As evaluated, using vectors outperforms using plain text in both semantic and computational performance on both tasks[7] (4.1 and 4.2).

In this paper, we propose log vectorizing -- leveraging the two benefits to cover the two pain points for our engineer during failure triage.

2.2 Vectorizing Tools

Many models that are capable for sentence-level embedding.

LSTM encoder[8] uses Long-Short Team Memory Recurrent Neural Network to generate sentence embedding; InferSent[9] a siamese BiLSTM network with max-pooling over the

output; Universal Sentence Encoder[10] trains a transformer network and augments unsupervised learning; GloVe[11] uses unsupervised learning algorithm for obtaining vector representations for words; Doc2Vec[12] estimates the distributed representations of documents, BERT[13] uses final hidden states for sentence embedding; Sentence-BERT[14] uses siamese network to further train BERT, etc. Among all available models, we choose the Sentence-BERT which achieved state-of-the-art in sentence level embedding to be our model(3.1.1).

2.3 Database of Vectors

Vector database, as known as the Vector Database Management System(VDBMS), is the database that can store vectors(fixed-length lists of numbers) along with other data item. Vector database is capable to do vector-wise semantic similarity search efficiently. We choose PGVector to be our vector database(3.1.2).

Name	License
Apache Cassandra, MilvusQdrant, Chroma,Vespa	Apache License 2.0
LlamaIndex	MIT License
MongoDB Atlas	N/A (Managed service)
Couchbase	Unknown (Preview)
pgvector	PostgreSQL License
Weaviate	BSD 3-Clause
Elasticsearch	Server Side Public License Elastic License

Table 1. Available Vector Database

3. Design and Solution

3.1 Design

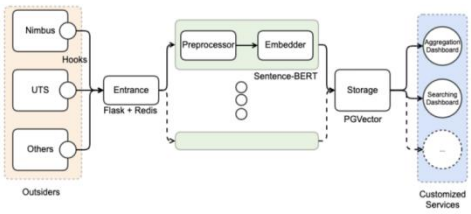

Figure 2. Design and Components

As displayed in the diagram, the architecture of the system can be defined as Entrance, Preprocessor, Encoder, Datastore, and ConsumerServices.

Entrance is a component to get logs from outsiders, for example, Nimbus/UTS/Bugzilla. Logs will be pre-processed in the Preprocessor to meet the format needs of the Encoder. The Encoder is the model that does vectorizing -- taking processed sentences(logs) and embedding them into vectors. The Datastore is the place to save vectors. The ConsumerServices are a set of services we designed to offer aggregation/search functions for customers(R&D engineers). As the Encoder and the Datastore are the core parts of this system, we will focus more on that.

3.1.1 Encoder

The Encoder is the component that transforms logs into vectors. It takes well-formatted logs from the Preprocessor and generates it's semantic embeddings by the language model. The model is usually built in docker container -- it can either be ran on GPU or CPU(NVIDIA-CUDA[15]) depends on the computational requirement of workloads(4.2).

Sentence-BERT

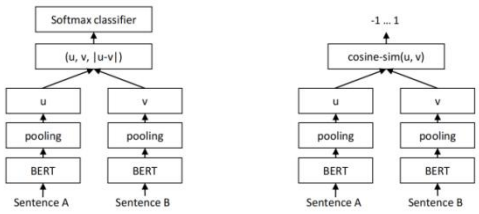

Figure 3. Sentence-BERT Architecture[14]

We choose Sentence-BERT to be our Encoder. Sentence-BERT[14] is a kind of model on which some specific training tasks are applied from the base pre-trained BERT model[13].

Concretely, the model is modified, and added the Siamese and the Triplet network structures on top of the basic BERT architecture with specific tasks that can help to derive sentimental embedding(Figure 3). The SNLI[16] (Bowman et al., 2015) and the Multi-Genre NLI[17] (Williams et al., 2018) are datasets that contain 1M sentence pairs annotated with "contradiction", "entailment" and "neutral" and a range of genres of spoken and written texts. They transform the original labels from the datasets to the cosine labels -- {"contradiction","entailment","neutral"} to {-1, 1, 0}, and fit them in the Siamese Network(Figure 3b). The datasets can also be transformed to corpus like {"source sentence", "entailment sentence", "contradiction sentence"} and used to train the Triplet Loss Network.

After the training, the Sentence-BERT can generate better embedding for sentences compared with the basic pre-trained BERT or any other language models in sentence-pair regression tasks.

Model	MR	CR	SUBJ	MPQA	TREC	MRPC	Avg.
Avg. GloVe embeddings	77.25	78.3	91.17	87.85	83	72.87	81.52
Avg. BERT embeddings	78.66	86.25	94.37	88.66	92.8	69.45	84.94
BERT CLS-vector	78.68	84.85	94.21	88.23	91.4	71.13	84.66
InferSent - GloVe	81.57	86.54	92.5	90.38	88.2	75.77	85.59
Universal Sentence Encoder	80.09	85.19	93.98	86.7	93.2	70.14	85.1
SBERT-NLI-base	**83.64**	**89.43**	**94.39**	**89.86**	**89.6**	**76**	**87.41**

Table 2. Sentence-BERT performance on semantic tasks[20]

Sentence-BERT pre-trained on Distilled BERT

As Sentence-BERT is an extension of the architecture and fitted with new pre-training tasks and datasets, it needs a basic BERT-like model. We choose the " distilbert-base-uncased"[18]. Compared to the BERT model, the Distilled BERT[19] only contains half of the parameters(~55M) and has only a slight performance drop over 14 sentence similarity tasks [20]. The model can be run on a 6GB graphic card with a batch size set to 16, which reduces the GPU requirements of inference. Currently, we have 2 NVIDIA A2 GPU[21] setup and 4 models running on it to support log vectorizing.

3.1.2 Datastore

We save sentence vectors with sentences in the datastore. As a vector database, it has additional semantic similarity functions with common SQL functions.

PG Vector

We use PG Vector[22] (Postgres-Based Vector Database) to be our datastore. PG Vector can supports around 500 QPS(Query Per Second) when set the probes to 40[23] and it takes 1 seconds for a top 100/1,000,000 vectors similarity query(4.2). Although PG Vector is not the best Vector Database among all open-source Vector Databases[24], we still choose it for two main reasons. First, it is a plugin of

Postgres -- we can reuse our pre-setup Postgres Database without any migration. Second, we tested it with the scale of our current workloads and it's capable. Once we increase our workloads, we will re-evaluate the Vector Databases[24].

3.2 Solution

In this chapter, we propose two solutions with several use cases that cover most of R&D failure triage scenarios. When implementing the failure-retrieval system, we keep the fundamental Encoder and the Datastore to be general and adaptable and the Preprocessors and CustomizedServices to be variable and specific for different scenarios.

3.2.1 Aggregation

Aggregation can be very helpful for failure triage. It can cluster/detect duplicates and let engineers focus on the root cause.

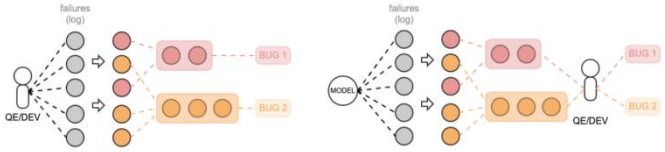

Figure 4. Aggregation supports failure triage

In this chapter, I will demonstrate three use cases that benefited from semantic aggregation. I will describe Usecase I in detail and will focus on the differences between the rest two to it.

DBSCAN

We use DBSCAN[25] as the basic algorithm for aggregation. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering non-parametric algorithm, which means it can cluster samples without defining the cluster quantity.

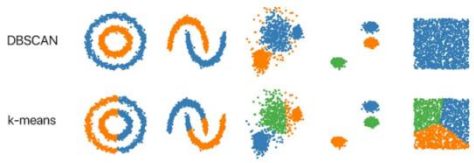

Figure 5. Illustrate DBSCAN(compared to K-Means)[26]

Given a set of points(samples) in some space and a minimal "distance" threshold, it groups points that are closely packed together (points with many nearby neighbors) -- if a sample can find any sample that is from a specific cluster and meanwhile their distance is smaller than the threshold, the sample will join this cluster. In our case, as the encoder model is trained by using cosine similarity, we set the cosine distance {-1.0, 1.0} to be the "distance" function. With a proper threshold set, we can aggregate logs by aggregating their log vectors.

Usecase I: Guest-OS Certification Test Logs

The Guest-OS(Operation System in VM) Certification Team is responsible for certifying Guest OS and Guest OS Partner on vSphere products. Concretely, they test vSphere features like hot-adding disk, adjusting memory on Guest OS including Windows, RHEL, SLE, OracleLinux, and Windows.

2022-05-09 10:48:06,009 | Failed at Play [gosc_cloudinit_staticip] ******************
2022-05-09 10:48:06,009 | TASK [GOS customization failed] **************************
task path: /home/worker/workspace/Ansible_Regression_RHEL_9.x/ansible-vsphere-gos-validation/linux/guest_customization/linux_gosc_verify.yml:108
fatal: [localhost]: FAILED! => ["VM static IPv4 address is '192.168.192.167', expected IPv4 address is 192.168.192.101", "VM static IPv4 gateway is '192.168.1.1', expected IPv4 gateway is 192.168.192.1"]

Table 3. GOSC test failure log paragraph

In most cases, as the size of their test failure logs is under 512 token restriction for BERT-like model[13], we merged their logs as a single sentence and use the model to vectorize it.

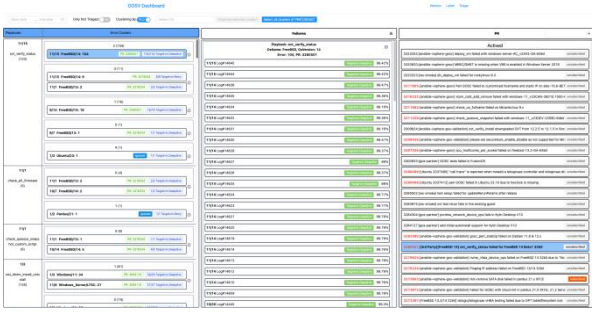

Figure 6. GuestOS Certification Failure Triage Dashboard

As displayed in the diagram, all their test failure logs are posted to our system and aggregated automatically. Sorted by date and quantity, engineers can focus on top-priority test failures first. Once the PR is created in Bugzilla and registered to an existing cluster, any new test failure that aggregated to this cluster will not show up in the UI. Once the PR is marked as resolved, the registered log clusters together with those logs will also be marked as resolved.

Usecase II: vSAN ST Test Logs

The vSAN ST(System Test) has a different test log format from the GOSV Team. They orchestrate tests will a set of operations and each operation is composed of many validations[27]. As each validation has its own log, the test logs are complex and big in scale -- we cannot simply merge them as a single passage and vectorize it by model just like Usecase I.

We use TF-IDF[28](Term Frequency–Inverse Document Frequency), the function that measures the importance of words by giving a collection of documents(test logs), to calculate sentence importance from all previous test results. Simply speaking, the more rare the words occur across documents, the more important the sentences containing those words are.

By ranking the sentence importance from test logs, we can extract, merge, and vectorize them. The rest same -- they will be aggregated and bound with PR.

Usecase III: Nimbus Deployment Logs

Nimbus Deployment logs are the logs from Nimbus when deploying a VM or a set of VMs (Testbed). Similar to Usecase II, Nimbus logs are large in scale and a filtering function is required. The difference is that we can collect both successful deployment logs and failed deployment logs -- which leads to another filtering function.

We feed the pre-trained Distilled-BERT[18] with the MSELoss[29](Mean Squared Error Loss Function) with Nimbus deployment logs. We collect both succeeded and failed deployment logs and balance them with a 1:1 ratio to be the fine-tuning dataset. Concretely, For each line from the deployment logs, if it comes from the succeeded deployment, we label it as 1.0; if it comes from the failed deployment, we label it as 0.0. Because of this, All sentences will have a {0.0, 1.0} ranking. We set the threshold to be 0.8 -- the sentence that gets a lower point will be kept[30].

After the fine-tuning, we deploy it for real-time inference. For all newly happened failed Nimbus deployments, all sentences will be sent to get the points first. Sentences that get passed the threshold will be extracted, merged, vectorized, and aggregated.

3.2.2 Searching

The Searching strategy has two main benefits. It's efficient when the scale of the logs is very large(millions of lines, for example); the search is based on the semantic similarity of the logs, which accelerates the process of finding the root cause.

In this chapter, we will describe two Usecases. In Usecase I, I will share a common scenario for Searching. I will focus more on describing Usecase II as it leverages the two benefits of Searching.

PSOD Core Dumps

As mentioned in Chapter 3.2.1, we use aggregation strategy for some vSAN QE teams -- all their failure logs are vectorized and saved. There is still one problem -- sometimes there are common failures occur from both teams, for example, some PSOD problems. As we build service instances for them separately, they cannot realize and leverage others' knowledge -- until many efforts are taken and finally they are notified from DEV teams.

Figure 7. Backtrace and Core Dump Search Dashboard

Instead of designing a system that can support both teams' requirements, we choose to build the searching service. The searching service is simple: as we have had vectorized and saved all logs, they can compare their current logs with all of them(3.1.2). This does require some additional efforts from our engineers -- they need to make some queries. But comparing with the efforts of designing and maintaining a common aggregation service that meets all specific requirements for all teams, it's still much easier.

Support Bundles

Support Bundle is a bunch of logs raised by core services in ESXi or vCenter, for example, the VPXD logs. Reviewing logs in Support Bundles is one of the major methods for engineers to deep-dive root causes. What's more, the Support Bundle is usually extremely big -- it's common to see tens of millions of logs in a Support Bundle.

The advantage of vectorizing can be fully leveraged in searching logs in Support Bundles. First, it takes much less time when querying logs in the format of vectors than in plain texts(The detail will be displayed in chapter 4). Second, finding semantic similar logs in the Support Bundle is helpful when engineers trying to understand the problem as engineers might not know the exact keywords for searching. Last but not least, engineers can register the root cause log pieces they figured to the failure triaging system.

This can help to find all Support Bundles in all test failures that had or will have the same root cause and notify all teams that met or will meet the same problem -- even without aggregation, it can still help to reduce the efforts of triaging duplicate test failures.

4. Performance Evaluation

4.1 Semantic Performance

Aggregation

	Test Failures	Clusters	PRs	Efficiency
Product Issue	11717	602	182	19x
Intermittent Issue	14086	477	\	29x
Untriaged	0	0	0	\
All	25803	1079		24x

Table 4. PRs and Aggregations from one partner team 2023

During 2023, the team had 25803 test failures. We helped them to aggregate failures into 1079 clusters which could be categorized into 182 PRs. This helped improve the failure triage efficiency by 24x(25803/1079) as they only needed one triage effort for each cluster.

Searching

In semantic search experiment, we focus on testing the ability of finding duplicate PR by locating root cause logs from their test Bundles. This simulates the performance and accuracy of the failure-retriveal system when the root cause is reveal.

	PRs	RCs	Dups
ALL	110	38	72
RC in Bundle Logs	48	18	30

Table 5. Experiment PRs and Bundles

We reviewed 110 ESXi Relevant PRs that had Duplicate PRs with available ESXi Support Bundles. We pick one Support Bundle for each PR. We choose 48 of them to be our experiment Dataset as their root cause could be identified in logs(time and number insensitive). We vectorized all the logs of them(4 lines of logs embedded as a vector) and we performed two experiments.

First, for each RC with Dups, we used the identified root cause log passage(revealed in Bugzilla) to query the Support Bundles. We embedded the log passage and made top-5 and top-50 cosine similarity comparison ranking. What's more, we recorded the median and average position the root cause log passage appeared. As a result, for 91% of the Bundles, the root cause log passage occured at top-5; for 100% of the Bundles, the root cause log passage occured at top-20.

ALL	Median(Position)	Average(Position)	Top-5	top-20
48	1	3.3	44(0.88~0.85)	48(0.88~0.74)

Table 6. Using RC log passage to query Bundles

Second, we set different thresholds of cosine similarity to test the accuracy of identifying duplicate PRs. If there exists any log from the Bundle that the cosine similarity pass the threshold, we labeled them as duplicate. We mixed in 2x irrelevent bundles into the datasets. As a result, around 83% duplicate PRs can be identified by threshold 0.88, 93% duplicate PRs can be identified by threshold 0.8. On the other side, 14% Irrelevant PRs were found for threshold 0.88 and 19% Irrelevant PRs were found for threshold 0.8.

PR Samples	Relevant(Bundles)	Irrelevant(Bundles)	Cosine-0.88		Cosine-0.8	
3306093	4	8	3	0	4	1
3312174	4	8	4	1	4	3
ALL	\	\	83%	14%	93%	19%

Table 7. Accuracy when using thresholds

4.2 Computational Performance

Vector/Text Search Comparison

Environment	Specification	Dataset	Function	Average Time
PG Vector DB	4x CPU, 16G memory, VM-Based	2,000,000 lines of logs	cosine (top 100)	0.31s
Elastic Search			full text	0.77s
Text Search			regex	2.06s
PostresDB			like	23.12s

Table 8. Vector Search vs. Text Search comparison

As displayed in Table. ,we made a comparison of search between PG Vector, Elastic Search, FileSystem Text Search and Postgres text search. We created 4 different VMs with same specification. We used 2 millions of lines of log to be

the dataset and repeated the query for 10 times. One thing to be noted is, as the input limitation for the Encoder is 512 tokens, 2M lines of logs will be embedded into 500k vectors.

GPU/CPU Embedding Comparison

Environment	Specification	Dataset	Average Time
GPU	NVIDIA V100 16G, 1.25GHz	100 sentences x 100 times	1.51s
CPU	Intel Xeon Gold 5230R, 2.10GHz		200.08s

Table 9. GPU vs. CPU Embedding Comparison Environment

As listed in the table, We vectorized 100 sentences(not more than 512 tokens) on both GPU and CPU environment and recorded the execution time. InstaML offers CUDA environment with GPU attached in docker container. We repeated the experiment for 100 times(different sentences each time) and the result is displayed at Figure 10.(red points are recorded time of CPU vectorizing, blue points are GPU vectorizing)

5. Future Work

We are evaluating a new architecture of Sentence-BERT, which has a potential ability to restore logs from the vectors. This means we can drop original logs after the vectorization, which can help to save a lot of storage.

What's more, we plan to integrate the system with Nimbus -- we vectorize all Support Bundles from all test failures happened on Nimbus, and we connect that with Bugzilla. Once a PR in Bugzilla is closed with a clear log passage pasted, all teams will be notified if the Support Bundle from their test failure has very similar logs.

6. Conclusion

In this paper, we proposed "vectorizing", a deep-learning-based failure-retrieval solution that transforms plain text logs to semantic vectors which can be further used for aggregation and searching to reduce triaging efforts on duplicate failures and increase deep-dive efficiency for failure root causes. We further described the solution in 5 scenarios that cover most of R&D failure triaging cases. We evaluated both the semantic and computational performance and displayed that the vectorizing method took advantage of plain text method.

7. References

[1] Nimbus: https://confluence.eng.vmware.com/display/DevToolsDocKB/Nimbus+Manual

[2] Bugzilla: https://bugzilla.eng.vmware.com/

[3] Humbug: https://humbug.eng.vmware.com/ui/home/req

[4] The Illustrated Word2vec, Jay Alammar – Visualizing machine learning one concept at a time. https://jalammar.github.io/illustrated-word2vec/

[5] Word2vec: Efficient Estimation of Word Representations in Vector Space. Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. https://arxiv.org/pdf/1301.3781.pdf

[6] Distributed representations of words and phrases and their compositionality. Mikolov et al. https://arxiv.org/abs/1310.4546

[7] Classification and Clustering of Arguments with Contextualized Word Embeddings. Nils Reimers, et al. at 2019. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.

[8] LSTM: Long Short-Term Memory-Networks for Machine Reading. Jianpeng Cheng, Li Dong, and Mirella Lapata. https://arxiv.org/pdf/1601.06733.pdf

[9] Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, Antoine Bordes. https://arxiv.org/abs/1705.02364

[10] USE: Universal Sentence Encoder. Daniel Cer, Yinfei Yang, Sheng-yi Kong, et al.,https://arxiv.org/abs/1803.11175

[11] GloVe: Global Vectors for Word Representation. Jeffrey Pennington, Richard Socher, Christopher Manning. https://aclanthology.org/D14-1162.pdf

[12] Doc2Vec: Distributed Representations of Sentences and Documents. Quoc V. Le, Tomas Mikolov. https://arxiv.org/pdf/1405.4053.pdf

[13] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. https://arxiv.org/pdf/1810.04805.pdf

[14] Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Nils Reimers and Iryna Gurevych. https://arxiv.org/pdf/1908.10084.pdf

[15] NVIDIA CUDA: Compute Unified Device Architecture. https://developer.nvidia.com/cuda-toolkit

[16] SNLI: A large annotated corpus for learning natural language inference. Samuel R. Bowman, Gabor Angeli, Christopher Potts and Christopher D. Manning. https://arxiv.org/pdf/1508.05326

[17] MNLI: A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. Adina Williams, Nikita Nangia and Samuel R. Bowman. https://arxiv.org/pdf/1704.05426.pdf

[18] Pre-trained DistillBERT Model: distilbert-base-uncased. https://huggingface.co/distilbert-base-uncased

[19] DistilBERT: a distilled version of BERT: smaller, faster, cheaper and lighter. Victor SANH, Lysandre DEBUT, Julien CHAUMOND and Thomas WOLF. https://arxiv.org/pdf/1910.01108.pdf

[20] Sentence-BERT Model Performances over 14 datasets: https://www.sbert.net/docs/training/overview.html

[21] NVIDIA A2 GPU with 16GB memory. https://www.nvidia.com/en-us/data-center/products/a2/

[22] PGVector: Open-source vector similarity search for Postgres. https://github.com/pgvector/pgvector

[23] PGVector Benchmark: https://supabase.com/blog/pgvector-performance

[24] TOP 15 vector databases. https://lakefs.io/blog/12-vector-databases-2023/

[25] DBSCAN: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Martin et al. https://cdn.aaai.org/KDD/1996/KDD96-037.pdf

[26] Implementation of DBSCAN from scratch. https://towardsdatascience.com/understanding-dbscan-algorithm-and-implementation-from-scratch-c256289479c5

[27] vSAN System Team test failure logs example: https://confluence.eng.vmware.com/display/~jinyuj/Vectorizing+your+logs+supports+failure+triage+in+many+ways

[28] TF-IDF: Term Frequency-Inverse Document Frequency, https://en.wikipedia.org/wiki/Tf-idf

[29] MSELoss: Mean Squared Error Loss Function. https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html

[30] Nimbus deployment logs filter example: https://confluence.eng.vmware.com/display/~jinyuj/Vectorizing+your+logs+supports+failure+triage+in+many+ways