ATHENA: TOWARDS IMPROVING SEMANTIC CODE SEARCH WITH CAUSAL REASONING AND KNOWLEDGE GRAPHS

Nathan Cooper
Advisor: Denys Poshyvanyk
College of William & Mary

April 2, 2021

Abstract

One main idea in software engineering is to reuse existing software instead of building due to the high cost. However, finding existing software that accomplishes a specific use case can be difficult. Code search has attempted to help this by allowing keyword searching for code similar to earlier search engines. Recent work has attempted to leverage the semantics present in the query and code.[6, 10, 17, 19, 20, 24, 34] However, these approaches do not consider the context of methods with the rest of the software. This context can be important when performing a search for determining the relevance of a returned result.[20] To address the current problems in code search we designed ATHENA, a semantic code search engine that leverages software's context through a knowledge graph and a graph neural network. The current implementation of ATHENA is the first step towards this and it currently supports searching for methods using a query method rather than natural language. For our preliminary evaluation, we have selected three open-source Java projects and had three computer science students evaluate the relevancy of 891 retrieve methods that ATHENA returned from 99 queries. We found that in 70% of cases the first method ATHENA retrieved was relevant to the query method.

Introduction

Software has become pervasive in our society, from running on our smartphones to controlling propulsion and control systems in spacecrafts. Without software, none of the current scientific revolutions would be possible. Therefore, it is important to make sure high quality software is available to everyone to ensure the progress of innovation.

The sharing of advances in software has been growing in popularity with big shifts in software companies becoming more open sourced. This has allowed for everyone to use their software as well as helping to contribute their time to improving the software. However, searching for software to reuse is still challenging in today's age.

To tackle this challenge, we introduce the current implementation of ATHENA, a tool for improving the relevancy of retrieved methods from a given query method using a software knowledge graph and graph neural network by capturing the context methods that are being searched. Context is important because similar to natural language with J.R. Firth's famous quote "[y]ou shall know a word by the company it keeps",[12] looking at a method's context, *i.e.,* the file, package, and project it resides in, can give a deeper insight as to its intent and functionality.[20] Specifically, we leveraged a software project's call graph to construct our software knowledge graph and used a content-aware node2vec[23] graph neural network that uses a pretrained Transformer[37] neural network for encoding each method in the knowledge graph.

Our preliminary evaluation involves evaluating how well ATHENA is at finding methods that are relevant to a given query method inside of a software project via a novel coupling metric. This is the first step towards allowing developers to use a natural language query to semantic search for code that accomplishes the goal of the query. We also discuss future work for incorporating natural language queries, instead of solely method queries, causal reasoning, and additional knowledge into our existing knowledge graph in Sec. 7.

In summary, this paper discusses the following:

1. An initial implementation of ATHENA, that helps developers find relevant methods given a query method;

2. The results of our preliminary evaluation that measures ATHENA's ability to retrieve relevant methods;

3. and an analysis of the current limitations of Athena and how future work will be done to address Athena's current limitations.

Implementation

Athena *Overview*

Athena takes advantage of the rich information present in the actual code and structural information present in software projects' call graphs by combining them in a non-linear way using a Graph Neural Network (GNN) algorithm. This combined information is then used for calculating a novel coupling metric, which we named *Neural Coupling*.

Athena has three components, namely *Neural-based Representation*, *Structural Representation*, and *Representation Fusing*, as shown in Figure 1. Each component is explained in the following subsections.

Neural-based Representation

In Athena's *Neural-based Representation* component, each method, M, in a target software project, P, is transformed into a distributed vector representation using some function f_{neural}. For this *Neural-based Representation* function, $f_{neural}(M) = V_M, M \in P$, we use CodeBERT,[10] which is an approach to convert code into a series of tokens, *e.g.,* words, that each have a continuous distributed vector by being trained on two different pretraining objectives. CodeBERT is based on the Transformer[37] architecture, which uses multi-head attention to allow successive layers in the neural network to pay attention to parts of the input, in our case a method, that might be important to the overall meaning. To teach CodeBERT to understand the meaning of a method, the Feng *et al.* trained CodeBERT using two objectives: RoBERTa and ELECTRA.[8,26] The RoBERTa pretraining objective involves masking random tokens of the input, which is a pair of a method and its docstring, and the model is tasked with filling in these missing parts. The ELECTRA pretraining objective is similar to RoBERTa. However, instead of masking tokens of the input, some tokens are replaced with similar ones, in CodeBERT these are determined by an n-gram model.[21] The model is then tasked with determining which of the tokens have been replaced.

After each token in the method is vectorized, we combine them into a single vector in order to easily perform similarity between methods since some methods may be longer than others. To combine the tokens into a method level vector, we used the approach by[33] and their corresponding sentence-transformers[5] library, which allows us to pool the token level vectors with different techniques and add an additional fully connected layer that generated the final method level vector. We discuss all implementation details in Sec. 2.5.

Structural Representation

In order to capture the structural aspect of a software system, we construct a knowledge graph by using the project's method-level call graph to serve as our structural representation. More formally, let $G = (N, E)$ be a target software project graph, where N represents the list of methods in the project as nodes and E represents the list of method calls between the methods in the project.

To construct this method-level call graph, we use *java-callgraph*,[1] which is a tool that allows for generating a call graph between different source code entities.

Representation Fusing

Previous work suggest fusing semantic and structural information by taking a linear combination of the two coupling metrics.[15] However, this has its drawbacks as discussed previously.[22] For example, if one of the metrics performs poorly for a certain type of coupling, *i.e.,* structural coupling when evaluating methods that only have a hidden dependency, then it will drag down the overall performance of the fused metric. To overcome this drawback, we apply a content-aware GNN to combine the two sources of information in a non-linear manner. Specifically, we use a modified node2vec[16] GNN that considered the content inside of the nodes of the graph for non-linearly combining both Neural-based and Structural representations to generate node vectors. This usage of content-aware node2vec has shown promise in the Biomedical domain for embedding Biomedical ontologies.[23] Our novel contribution is through the use of a pretrained Transformer model for generating the semantic embedding of the nodes and application of this approach to the software engineering domain. We hypothesize that having this non-linear interaction, from two different information sources, allows node2vec to learn hidden semantic and structural patterns at method-level granularity. Our preliminary evaluation and its results, which we discuss in Sec. 3 and 4, is the first attempt in empirically refuting or confirming this hypothesis.

In node2vec, each node is given an initial random vector representation called an embedding. These embeddings are iteratively updated similarly to the original word2vec[30] model using a skip-gram objective. However, since graph structures

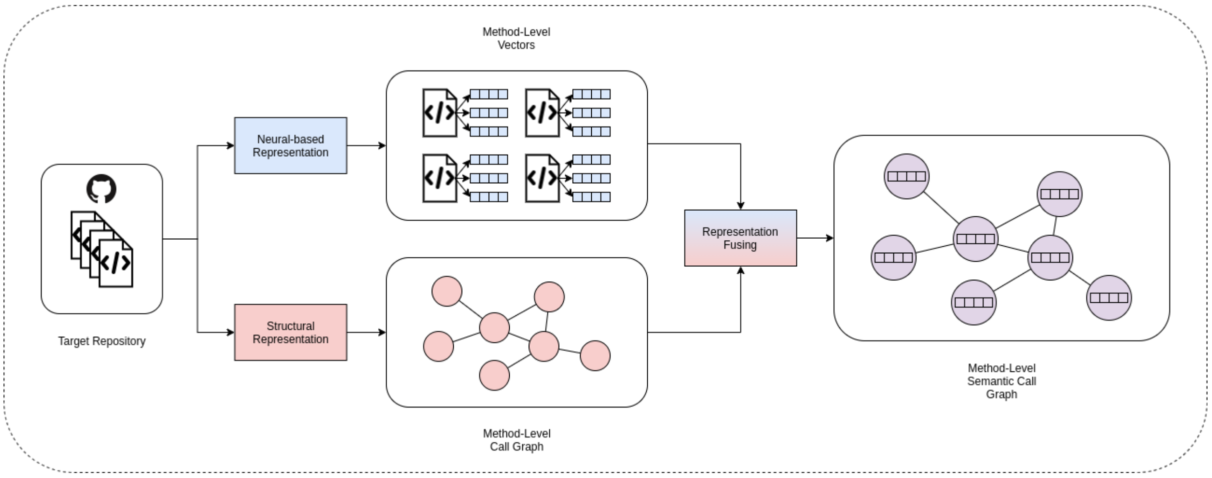

Method-Level Vectors
Neural-based Representation
Target Repository
Structural Representation
Representation Fusing
Method-Level Call Graph
Method-Level Semantic Call Graph

Figure 1: The Athena approach.

have no natural order, the node2vec algorithm randomly selects nodes and performs walks to generate a sequence of nodes that can then have the original word2vec skip-gram algorithm applied for updating the embeddings:

$$max \sum_{u \in N} -logZ_u + \sum_{n_i \in Walk_k(u)} f_{struct}(n_i) \cdot f_{struct}(u). \tag{1}$$

Where $Z_u = \sum_{v \in N} exp(f_{struct}(u) \cdot f_{struct}(v)$ and is approximated via negative sampling[30] due to the high computational cost. Additionally, where $Walk_k(u)$ are k nodes along a random walk starting on the focus node u. We model our edges as unweighted. Therefore, each node's neighbor has an equal probability of being selected along the walk.

This modified skip-gram tries to maximize the log probability of reconstructing a node's neighbors given the embedding representation of the focus node u, where neighbors are approximated by this random walk and do not necessarily need to be close to the starting node. To learn this function, an embedding layer that serves as a node embedding lookup table is used and optimized using stochastic gradient decent.

We modified this node2vec algorithm by replacing the node embedding lookup table with a Transformer model that generates neural vectors captured in the *Neural-based Representation* component instead of with vectors. This allows for the node2vec algorithm to have the information from the source code of each node when incorporating the structural information contained in the call graph of the software project that was extracted in the *Structural Representation* component. Additionally, during our testing we found the usage of neural vectors from the pretrained Transformer architecture to cause issues with training due to the regular dot product performed used in node2vec having the loss gradients explode leading to the model not learning. To fix this issue, we compute the sum of the cosine similarities of the focus node to its neighbors. This prevented the exploding gradients issue. However, more experiments will need to performed to understand the impact of this change.

Table 1: Hyperparameters used for training different ATHENA models.

Hyperparameter	Value
Walk Length	10
Context Size	10
# of Walks per Node	1
# of Negative Samples	1
Optimizer	SGD + ADAM
Learning Rate	$1E-2$
Batch Size	8
Epochs	20

To generate the coupling metric from this fused representation, we compute the cosine distance metric, $cos_d ist(\boldsymbol{m}_1, \boldsymbol{m}_2) = 1 - \frac{\boldsymbol{m}_1 \cdot \boldsymbol{m}_2}{||\boldsymbol{m}_1|| \cdot ||\boldsymbol{m}_2||}$ for every method's vector in a project's knowledge graph and the given query method's vector and use this to represent how coupled two methods are. The coupling metric value for each pair of methods is then returned to the developer in ranked order, higher coupling equates to higher likely the method belongs to the same software component, to help in determining which methods in the project's knowledge graph are part of the same software component as the given query method.

Implementation Details

You can find our selected hyperparameters in Tab. 1. To implement our *Neural Representation* CodeBERT encoder model we used the pretrained model available from Huggingface's *transformers* library.[38] Additionally, to combine the individual token vectors produced by CodeBERT into a singular vector representing an entire method, we used *sentence-transformers* library with max pooling and a fully connected layer with 256 units. For implementing the *Representation Fusing* node2vec model we used Pytorch Geometric[11] and modified their example node2vec implementation [1]. For managing data and training our model used Pytorch Lightning.[9]

All training was done on an Ubuntu 20.04 server with a single A100 NVIDIA GPU with 40GBs of VRAM, 128 CPU cores, and 1TB of RAM.

Evaluation

The end goal of our evaluation is to assess how effective ATHENA is at finding relevant software components to user queries to promote code reuse and reduce code duplication. The first step towards this goal is to find relevant methods since any user query results should all have methods that are relevant to each other. Therefore, our initial evaluation assesses how effective ATHENA is at returning relevant methods given a method query. To help us determine this goal, we formulated the following research question to guide our evaluation:

RQ$_1$: *How well is* ATHENA *able to find relevant methods to a given query method?*

Data Collection

3.1.1 Project Selection

In this study we exploited 313 open-source Java projects from GitHub. To avoid low-quality projects, we only selected projects that meet the following criteria: (1) recently active (has at least one commit within the last 6 months) (2) not a toy project (at least 100 commits and at least 10 contributors), (3) semi-popular (at least 10 stars), (4) used only Java 10 or lower, (5) used Maven[2] for building the project, and finally (6) not a fork and only the default branch.

Relevant Method Recovery

For **RQ$_1$**, we selected a total of 99 query methods from three java projects, namely *commons-beanutils* from apache, *secor* from pinterest, and *pdb* from feedzai. Table 2 has the projects' statistics and description. These query methods came from various commits of the repositories to obtain a diverse set. Once selected, they were given to ATHENA to generate their cosine distance metric to the rest of their corresponding projects using the approximate k-nearest neighbors algorithm from sklearn.[32] Going with too large of a k value will not show the usefulness of ATHENA since most developers will not review large amounts of retrieved methods. Therefore, we used a k value of 9 for our evaluation, which we think borders right on the edge of what most developers will look through. To measure ATHENA's performance, we calculated the mean average precision (mAP) and mean reciprocal rank (mRR) of ATHENA's ability to return relevant methods, where relevance is determined by three computer science students, including myself, and is discussed in more detail below. mAP measures how well an approach is performing by calculating the number of true positives, *i.e.,* methods that are part of the same software component, averaged across multiple windows of the top returned results for all method queries. Specifically:

$$\text{mAP} = \frac{1}{|M|}\sum_{m}^{M}\left(\frac{1}{|R|}\sum_{k}^{|R|}(R_k)\right) \tag{2}$$

Where R_k is the relevancy list of 0s and 1s, 0 being irrelevant and 1 being relevant, up to k results for some method query m.

mRR is used to measure how consistent an approach is at finding at least one relevant method high in the list of results it returns and is defined as:

$$\text{mRR(M)} = \frac{1}{|M|}\sum_{m}^{M}\left(\frac{1}{rank(m)}\right) \tag{3}$$

Where $rank(m)$ is the index of the first relevant method returned from the given method query m.

We used the following guidelines for determining if a retrieved method from ATHENA is relevant to a given query:

1. The retrieved method is a type of clone of the query method. Specifically, a method can be a clone if it meets one of the following clone type criteria from Svajlenko and Roy:[36] (i) type 1 are syntactically identical, except for minor changes in white space, layout, and comments, (ii) type 2 are syntactically identical, except for variable names and literal values, (iii) type 3 are syntactically similar, but contain differences at the statement level such as the addition, modification, or removal of certain parts, and (iv) type 4 are syntactically dissimilar and are only semantically similar, *i.e.,* they perform the same functionality.

2. The retrieved method performs similar or complementary functionality compared to the

[1]https://github.com/rusty1s/pytorch_geometric/blob/master/examples/node2vec.py

Table 2: Statistics of the different libraries used in our evaluation.

Library	# Stars	# Contributors
apache/commons-beanutils	194	29
pinterest/secor	1.7k	108
feedzai/pdb	40	30

query. For example, if the query method deals with getting a user's phone number and the retrieved method deals with getting a user's name, then we would consider this retrieved method as relevant since it performs similar functionality of getting user information.

3. The query or retrieved method call each other since this implies that the retrieved method is part of the same software component as the query method.

4. Only the query and retrieved methods are considered, not the file the method resides in or any other context, when making the relevancy determination.

Results

In this section we discuss the results from our preliminary evaluation and the limitations of Athena.

Relevancy Results Discussion

Tab. 3 show the results of Athena for the three different libraries we included in our preliminary evaluation along with the overall performance. As show, Athena performs best on the library *pinterest/secor*, with an mRR, mAP, mean Rank, and HIT@1 score of 81.3%, 77.6%, 1.16, and 75.8%, respectively. The large difference in performance with respect to the other libraries, $7-8\%$ in terms of HIT@1, may be due to the quality difference in the library since *pinterest/secor* also had the largest amount of GitHub stars and contributors. However, more analysis will be needed to fully understand this difference. For all libraries, Athena performs well with an average HIT@1 score of 70.7%, meaning that a majority of Athena's first retrieved methods are relevant to the given query method. In the next section we discuss some of Athena's limitations and potential remedies.

Results for RQ$_1$: From our preliminary evaluation, Athena is able to achieve an overall mRR, mAP, mean Rank, and HIT@1 score of 76.9%, 73.0%, 1.45, and 70.7%, respectively. This shows that the methods Athena retrieved from our given set of queries are relevant based on our criteria in Sec. 3.2.

Limitations Discussion

In this subsection we will discuss some of the limitations we uncovered during our evaluation of Athena.

From our exploration of the successes and failures of Athena, we found three broad cases for both successes and failures. For successes these cases are:

S$_1$: *Query and retrieved methods are type 1 or 2 clones*

S$_2$: *Retrieved methods are overloaded versions of the query*

S$_3$: *Query and retrieved methods have very similar names*

Fig. 2 show an example of case **S$_2$** by Athena. From this case, you can see Athena identifies a relevant method to the query method since they are performing very similar functions, *i.e.,* extracting an event type. The only difference being the type of object being extracted from.

For failures, these categories are:

F$_1$: *Query method contains very generic information*

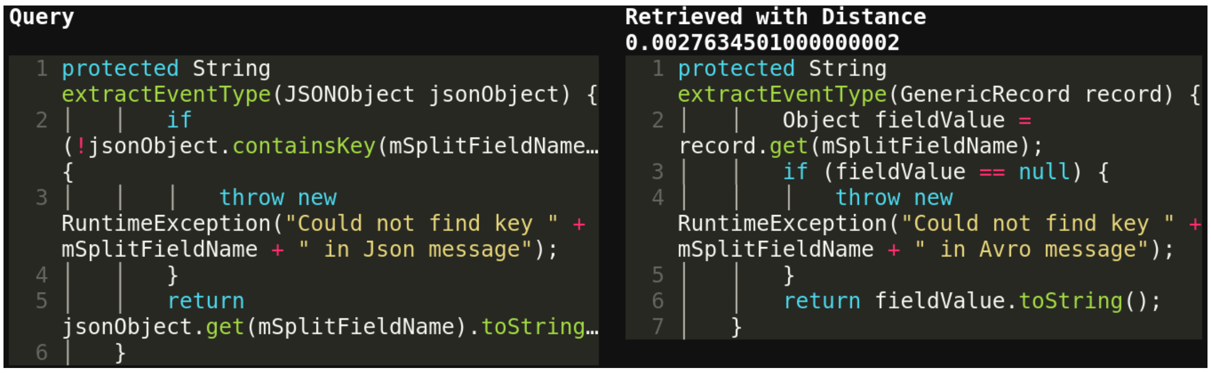
F$_2$: *Query and retrieved methods contain similar structure*

F$_3$: *Query and retrieved methods contain many of the same variable names*

Fig. 3 shows the failure case **F$_1$**. As shown from the failure case, Athena can retrieve a completely irrelevant method since capitalizing a property name has nothing to do with generating a hash code for an object. The function *hashCode* is a common utility function in the Java programming language and generally does not contain a

Table 3: Results of the best performing Athena models for each library and overall.

Library	**mRR**	**mAP**	**μRk**	**HIT@1**
apache/commons-beanutils	73.8%	66.9%	1.19	68.8%
pinterest/secor	81.3%	77.6%	1.61	75.8%
feedzai/pdb	75.7%	74.1%	1.56	67.6%
Overall	76.9%	73.0%	1.45	70.7%



Query

```
protected String
extractEventType(JSONObject jsonObject) {
    if
(!jsonObject.containsKey(mSplitFieldName…
{
        throw new
RuntimeException("Could not find key " +
mSplitFieldName + " in Json message");
    }
    return
jsonObject.get(mSplitFieldName).toString…
}
```

Retrieved with Distance 0.0027634501000000002

```
protected String
extractEventType(GenericRecord record) {
    Object fieldValue =
record.get(mSplitFieldName);
    if (fieldValue == null) {
        throw new
RuntimeException("Could not find key " +
mSplitFieldName + " in Avro message");
    }
    return fieldValue.toString();
}
```

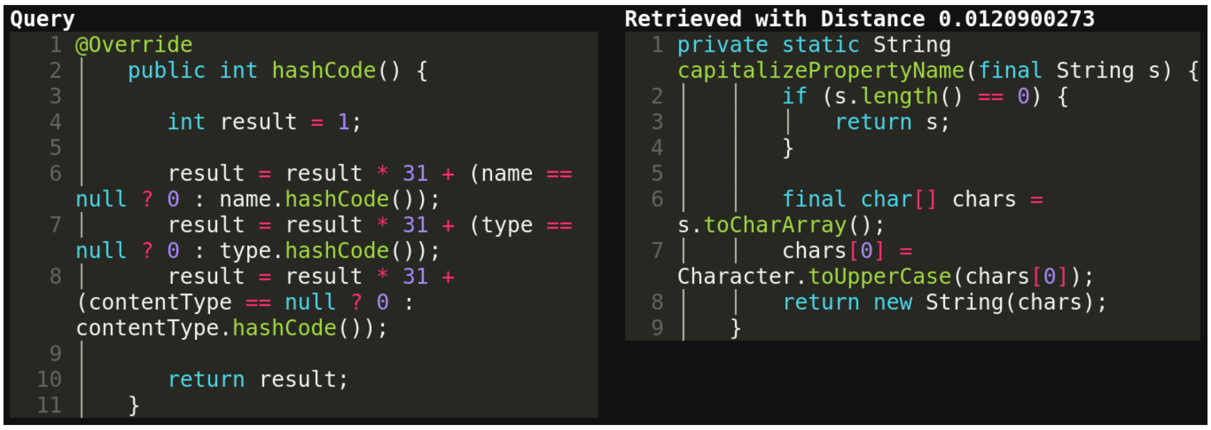
Figure 2: Success case of Athena retrieving a relevant method.

Table 4: Statistics of top 10 successes and failures of Athena across all libraries based on average precision.

Case	**μ Dist**	**σ Dist**	**Median Dist**
Successes	3.08E-3	1.85E-3	3.34E-3
Failures	7.29E-3	3.15E-3	6.00E-3

lot of project specific logic, *i.e.,* they contain very generic information. This makes it extremely difficult for Athena to extract useful information from both the semantic and structural representations of this method. We found this to be common case, mostly due to *getter* functions. However, there were other examples that did not fit into one of our broad set of failure cases. One reason for this could be due to our criteria for classifying a method as relevant. Since the participants did not look into the context surrounding the retrieved method and query, some structural information could be missing that made the retrieved methods actually relevant such as if the query and the retrieved methods were called in an intermediating series of methods. Another reason for these failure cases could be these queries do not contain meaningful relevant methods in the software project and regardless of how accurate Athena was it would be unable to retrieve relevant methods. Lastly, the current implementation of Athena may lack the ability to effectively use the information stored in these query methods to find relevant methods in the rest of the library. Our future evaluation will explore these possibilities to better understand Athena and its limitations as well as potential ways to overcome them.

One interesting finding we discovered, which can be seen in Tab. 4, while exploring these success and failure cases is that successes in terms of high average precision, *i.e.,* multiple relevant methods are higher in the list, also correlates well with the cosine distance between the query method and each retrieved method being low, which meaning more similar to each other based on their fused vector representation. The reverse is true for failure cases, *i.e.,* low average precision correlates well with the cosine distance being high. We calculated the Pearson correlation coefficient[13] and associated p-value and found the average precision and cosine distance to be negatively correlated with an r value of $-4.98E{-}1$ and p-value of $1.55E{-}7$ showing this correlation is statistically significant. This means that Athena's cosine distance metrics can be used as additional information for determining if retrieved methods are relevant to the given query since there is a correlation between how large this distance is and how relevant the returned list of methods are. Therefore, to prevent showing potentially irrelevant methods to a developer, which could cause the developer to stop trusting the tool, a threshold could be used to filter retrieved methods.



Query

```
@Override
    public int hashCode() {

        int result = 1;

        result = result * 31 + (name == null ? 0 : name.hashCode());
        result = result * 31 + (type == null ? 0 : type.hashCode());
        result = result * 31 + (contentType == null ? 0 : contentType.hashCode());

        return result;
    }
```

Retrieved with Distance 0.0120900273

```
private static String capitalizePropertyName(final String s) {
        if (s.length() == 0) {
            return s;
        }

        final char[] chars = s.toCharArray();
        chars[0] = Character.toUpperCase(chars[0]);
        return new String(chars);
    }
```

Figure 3: Failure case of ATHENA retrieving an irrelevant method.

Finding: When the query method tends to be generic, ATHENA has a difficult time retrieving relevant methods. More evaluations need to be performed to explore the true cause of some irrelevant methods being returned even with query methods that are not generic. ATHENA's cosine distance metric correlates relatively well, Pearson correlation coefficient of $-4.98E-1$, with its performance in retrieving relevant methods, in terms of average precision. This correlation may be able to be leveraged to filter out irrelevant methods through a threshold cosine distance value for retrieved methods.

Threats to Validity

Internal Validity. The biggest threat to internal validity comes from our guidelines for deeming if a retrieved method is relevant or not. To reduce this threat we used three students who all have computer science backgrounds, two of which have worked previously in industry. Additionally, for one of our criteria, we relied on previously established definitions for similar methods, *i.e.,*, code clones. However, even with these guidelines, many examples could be misclassified due to not having the full context of the method within the project making it difficult to determine if the methods were a part of the same software component. While this additional context was not possible during this initial evaluation due to time constraints, we will be adding this to our guidelines in our future evaluation. Lastly, our metrics for evaluating ATHENA's performance do not capture any methods ATHENA did not retrieve that may have been relevant, *i.e.,* ATHENA's recall rate. This is because it would be extremely difficult to go through all pairs of methods to determine which methods ATHENA missed, which is why it was disregarded.

External Validity. Due to our small sample size of libraries, only three, it is unclear if these results will generalize to other libraries. This applies similarly to other programming languages since we only chose projects that use Java. However, the CodeBERT[10] model we are using was shown to work across multiple programming languages. Additionally, our future evaluation will involve hundreds of projects to help evaluate ATHENA's generalizability.

Related Work

Existing approaches have attempted to solve code search using classical keyword matching for computing syntactical similarity,[3,4,7,25,27–29,39] *e.g.,* Term-Frequency Inverse Document-Frequency (TFIDF).[35] However, natural language and software have little overlap in their vocabulary and their structure. Therefore, previous techniques such as keyword matching and TFIDF that rely on finding commonalities in the vocabulary of queries and methods resulted in poor performance.[14,18,40,41]

To overcome this limitation a lot of recent techniques compute semantic similarity such as using WordNet for finding synonyms of words as in Li *et al.*[24] Sachdev *et al.*[34] used word2vec,[31] a neural network that uses machine learning to represent words as continuous vectors, to generate word embeddings for natural language and software. Additionally, many recent approaches have used deep learning for code search[6,10,17,19,20] using a variety of different types of architectures and learning methods. The closest to ours is work by Feng *et al.*[10] since we are using their CodeBERT model as the foundation for ATHENA's *Neural Representation* component. However, similar to the other deep learning approaches for code search, CodeBERT does not consider the context methods reside in. We extend their work by including this additional context since software developers consider this important information when considering whether a method is relevant to a search query.[20]

Conclusion and Future Work

In this paper we presented ATHENA, a Graph Neural Network that using a software project knowledge graph to learn the semantic and syntactic information present in the software project's methods. This learned information is then leveraged to retrieve methods relevant to a given query method. We evaluated ATHENA in a preliminary study involving three Java libraries using standard information retrieval metrics and found ATHENA shows potential by achieving an overall mRR, mAP, mean Rank, and HIT@1 score of 76.9%, 73.0%, 1.45, and 70.7, respectively. Additionally we discussed ATHENA's current limitations such as performing poorly on generic methods.

Future work. With the limitations outlined in 4.2, we intend the explore ways of eliminating these limitations through different architectures, training, and thresholding techniques. Additionally for future work, we will implement the features as discussed in the original proposal, namely: (i) natural language queries by adding method docstrings into the knowledge graph so relevancy between natural language and methods can be performed, (ii) causal reasoning to overcome the directionality issue where the model may have a hard time separating converting an int to a string from converting a string to an int by leveraging causal graphs and related techniques, (iii) evaluating our ATHENA on a larger set of libraries for better exploring ATHENA's generalizability, and (iv) constructing a benchmark for evaluating the performance and interpreting different models for code search to help future researchers in this field.

Acknowledgements

I thank my other contributors David Nader, Daniel Rodriguez-Cardenas, and Emad Aghajani for their help on this project with data collection and processing and on the evaluation. I also thank my advisor, Denys Poshyvanyk, for his help and guidance on this project. Finally, I thank the Virginia Space Grant Consortium for helping fund this research.

References

[1] java-callgraph https://github.com/gousiosg/java-callgraph, 2018.

[2] Apache maven project https://maven.apache.org/, 2020.

[3] Krugle code search https://www.krugle.com/, 2020.

[4] Ohloh code search https://code.ohloh.net/, 2020.

[5] Sentence transformers library https://github.com/UKPLab/sentence-transformers, 2021.

[6] CAMBRONERO, J., LI, H., KIM, S., SEN, K., AND CHANDRA, S. When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2019), ESEC/FSE 2019, Association for Computing Machinery, p. 964–974.

[7] CHAN, W.-K., CHENG, H., AND LO, D. Searching connected api subgraph via text phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2012), FSE '12, Association for Computing Machinery.

[8] CLARK, K., LUONG, M.-T., LE, Q. V., AND MANNING, C. D. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR* (2020).

[9] FALCON, W., AND .AL. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning 3* (2019).

[10] FENG, Z., GUO, D., TANG, D., DUAN, N., FENG, X., GONG, M., SHOU, L., QIN, B., LIU, T., JIANG, D., AND ZHOU, M. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (Online, Nov. 2020), Association for Computational Linguistics, pp. 1536–1547.

[11] FEY, M., AND LENSSEN, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[12] FIRTH, J. R., 11 1957.

[13] FREEDMAN, D., PISANI, R., AND PURVES, R. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[14] Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. The Vocabulary Problem in Human-System Communication: an Analysis and a Solution. *Communications of the Acm (cacm) 30*, 11 (1987), 964–971.

[15] Gethers, M., and Poshyvanyk, D. Using relational topic models to capture coupling among classes in object-oriented software systems. In *2010 IEEE International Conference on Software Maintenance* (2010), pp. 1–10.

[16] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).

[17] Gu, X., Zhang, H., and Kim, S. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering* (New York, NY, USA, 2018), ICSE '18, Association for Computing Machinery, p. 933–944.

[18] Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., and Menzies, T. Automatic query reformulations for text retrieval in software engineering. In *2013 35th International Conference on Software Engineering (ICSE)* (May 2013), pp. 842–851. ISSN: 1558-1225.

[19] Husain, H., and Wu, H.-H. Towards natural language semantic code search https://githubengineering.com/towards-natural-language-semantic-code-search/, 2018.

[20] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *arXiv:1909.09436 [cs, stat]* (Sept. 2019). arXiv: 1909.09436.

[21] Jurafsky, D., and Martin, J. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, vol. 2. 02 2008.

[22] Kagdi, H., Gethers, M., Poshyvanyk, D., and Collard, M. L. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *2010 17th Working Conference on Reverse Engineering* (2010), pp. 119–128.

[23] Kotitsas, S., Pappas, D., Androutsopoulos, I., McDonald, R., and Apidianaki, M. Embedding biomedical ontologies by jointly encoding network structure and textual node descriptors. In *Proceedings of the 18th BioNLP Workshop and Shared Task* (Florence, Italy, Aug. 2019), Association for Computational Linguistics, pp. 298–308.

[24] Li, X., Wang, Z., Wang, Q., Yan, S., Xie, T., and Mei, H. Relationship-aware code search for javascript frameworks. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (New York, NY, USA, 2016), FSE 2016, Association for Computing Machinery, p. 690–701.

[25] Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C., and Baldi, P. Sourcerer: Mining and searching internet-scale software repositories. *Data Min. Knowl. Discov. 18*, 2 (Apr. 2009), 300–336.

[26] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach.

[27] Lv, F., Zhang, H., Lou, J., Wang, S., Zhang, D., and Zhao, J. Codehow: Effective code search based on api understanding and extended boolean model (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (2015), pp. 260–270.

[28] McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C., and Xie, Q. Exemplar: A source code search engine for finding highly relevant applications. *IEEE Transactions on Software Engineering 38*, 5 (2012), 1069–1087.

[29] McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C. Portfolio: Finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering* (New York, NY, USA, 2011), ICSE '11, Association for Computing Machinery, p. 111–120.

[30] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Red Hook, NY, USA, 2013), NeurIPS'13, Curran Associates Inc., p. 3111–3119.

[31] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and

K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119.

[32] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[33] Reimers, N., and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing* (11 2019), Association for Computational Linguistics.

[34] Sachdev, S., Li, H., Luan, S., Kim, S., Sen, K., and Chandra, S. Retrieval on source code: A neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (New York, NY, USA, 2018), MAPL 2018, Association for Computing Machinery, p. 31–41.

[35] Salton, G., and McGill, M. J. Introduction to modern information retrieval.

[36] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K., and Mia, M. M. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution* (2014), pp. 476–480.

[37] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, u., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NeurIPS2017'17, Curran Associates Inc., p. 6000–6010.

[38] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (Online, Oct. 2020), Association for Computational Linguistics, pp. 38–45.

[39] Zhang, H., Jain, A., Khandelwal, G., Kaushik, C., Ge, S., and Hu, W. Bing developer assistant: Improving developer productivity by recommending sample code. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (New York, NY, USA, 2016), FSE 2016, Association for Computing Machinery, p. 956–961.

[40] Zhao, L., and Callan, J. Term necessity prediction. In *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM '10* (Toronto, ON, Canada, 2010), ACM Press, p. 259.

[41] Zhao, L., and Callan, J. Automatic term mismatch diagnosis for selective query expansion. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '12* (Portland, Oregon, USA, 2012), ACM Press, p. 515.