Automation of Selected Processes in IT Project Management with Natural Language Processing

Aneta Poniszewska-Maranda
Institute of Information Technology/Lodz University of Technology
Lodz, Poland *aneta.poniszewska-maranda@p.lodz.pl*

Piotr Wawrzynkiewicz
Institute of Information Technology/Lodz University of Technology
Lodz, Poland *piotrwawrzynkiewicz00@gmail.com*

Joanna Ochelska-Mierzejewska
Institute of Information Technology/Lodz University of Technology
Lodz, Poland *joanna.ochelska-mierzejewska@p.lodz.pl*

Abstract

The paper presents the use of deep learning models to support the automation of management in IT projects on the example of task assignment. Managing IT projects is a complex process that requires the coordination of multiple tasks, resources, and individuals involved in the project. For this purpose, datasets were created to simulate various project environments, and models based on the GraphSAGE architecture were trained, enabling efficient modeling of relationships between tasks and programmers. It was observed that improving data quality could significantly enhance the performance of the models, suggesting the potential for further development and improvements in this area.

Keywords: Process automation, project management, Natural language processing, deep learning, graph neural networks.

1. Introduction

IT project management is a complex process that requires coordination of many tasks, resources and people involved. Contemporary challenges resulting from the growing complexity of projects, dynamically changing requirements and the need to quickly adapt to new technologies pose increasingly complex tasks for project managers. The simultaneous need to effectively manage teams and resources with limited budgets and lead times leads to the development of tools and methods that allow for more effective planning, monitoring and implementation of projects. One of the key areas that is gaining importance in this field is the automation of project management processes using artificial intelligence techniques, in particular deep learning methods. The implementation of modern algorithms allows for better human resource management, risk analysis and optimization of project processes.

Various methodologies supporting project implementation processes have been used in IT project management for years. The most commonly used include the classic waterfall approach and agile methodologies, the most widely used of which is SCRUM. Modern IT project management is often supported by various tools that facilitate organization, communication, and progress monitoring. Tools such as Jira, GitHub, and Microsoft Teams are an integral part of modern project management practices [22]. The use of these project management tools and methods allows for effective management of IT projects, process optimization and increased efficiency of project teams. However, although these tools offer significant support in automating various aspects of work, full automation of task assignment and project management still requires active supervision and intervention from project managers who must actively track the competences of people in a team that often changes dynamically [14].

The paper presents an analysis of various deep learning models (e.g. neural networks, transformers, graph networks) supporting selected IT project management processes. After analyzing current solutions, the best method for use in the work area was selected and a system for automatic task assignment to programmers based on their competences was prepared in the context of extracting tags/keywords from task descriptions and programmers' CVs. An appropriate data set was selected on which the tested model was tested and an analysis of the implemented model was made on the selected data set and the effectiveness of different models on the test set was compared.

2. NLP methods for keyword extraction in management tasks

The use of natural language processing (NLP) methods in project management enables automatic analysis and extraction of key information, which significantly supports the processes of task allocation and resource management in dynamic project environments. Managing software development teams is a challenging task, especially in the context of assigning tasks and managing dynamic changes in the team composition. One of the main problems is the appropriate matching of tasks to the competences of individual programmers. Managers use various methods and tools to effectively assign tasks and monitor progress. One of such tools is RACI matrix (Responsible, Accountable, Consulted, Informed) [5], MoSCoW method (Must have, Should have, Could have, Won't have) [3], Skill-based Task Allocation method [7], [11]. Although these methods and tools significantly support the task assignment process, full automation of this process still faces challenges. It requires constant adaptation to changing conditions and team competences. Machine learning uses various algorithms to analyze data and build models that can predict outcomes or classify data, such as decision trees [17], linear regression [16], Support Vector Machines (SVM) [4], [8], K-means algorithm [20], neural networks, and deep networks such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and transformers [19], [21], [23]. In recent years, transformers have become a key element in the field of NLP. Models based on transformer architecture, such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), have revolutionized the way natural language processing is done through Attention Mechanism [26].

Graph Neural Networks (GNNs) are another advanced technique in the field of deep learning that has gained importance in processing structured data. GNNs are designed to work with data that can be represented as graphs, where nodes represent entities and edges represent relationships between them. As a tool for analyzing natural language, GNNs can be used to analyze relationships between text entities, such as identifying words, or extracting information from documents where the relationships between different pieces of text are crucial [27, 28]. GraphSAGE (Graph Sample and Aggregate) is an innovative graph neural network architecture designed for scalability and generalization to new, previously unseen nodes. In traditional GNNs such as GCN (Graph Convolutional Networks), the full adjacency matrix must be read into memory, which can be problematic for very large graphs. GraphSAGE addresses this problem by introducing a sampling approach that randomly selects subsets of neighbors for each node during training. This allows the model to efficiently process huge graphs by processing only a fraction of the data at a time [6], [15].

3. Related works

The following two main research questions were defined in order to specify the existing works in the mentioned area: (1) RQ1: "To what extent does the use of graph neural networks (GNN) in IT project management systems improve the efficiency of decision-making processes and the optimization of resource allocation compared to traditional methods and algorithms?". (2) RQ2: "How effective is the integration of natural language processing (NLP) and graph neural networks (GNN) techniques in the automation and improvement of IT project management processes, taking into account the use of TF-IDF, Word2Vec and FastText algorithms?".

The use of GNNs in text processing allows for effective modeling of relationships between words and sentences, which significantly improves text classification results. The works [24], [29, 30] emphasize the potential of GNNs in solving complex problems, especially in document analysis and text structure tasks. The authors of [24] analyze the application of graph neural networks (GNN) in communication networks, especially emphasizing their ability to generate precise data-driven models, which is crucial in the optimization and management of modern networks. The paper [29] presents a review that includes methods, architectures and applications of graph neural networks. The authors discuss in detail the guidelines for building GNNs, analyzing their potential in various fields, from pattern recognition to processing graphical data.

The authors of [30] present a text classification method using graph neural networks (GNN), where each document is represented as an individual graph. The main challenge that the study addresses is the effective representation of the document structure in the form of a graph, which allows for better capturing the complexity of text data and improving the classification results. The use of GNNs allows for the analysis of each document in its original context, which distinguishes this method from classical approaches to text classification. The work [29] presents various ways of applying graph neural networks in NLP tasks. The article addresses the problems related to automatic graph creation from text data and discusses the challenges facing these models, such as efficiency and interpretability. The authors also analyze current trends and future research directions, emphasizing the potential of GNNs in solving NLP problems. The study of word embeddings in text using methods such as Word2Vec and FastText [1], [25] has proven to be extremely effective in text analysis, especially in the analysis of specialized language. Thanks to them, it is possible to model semantic relations between words more precisely, which leads to better results in the tasks of extracting key information from text. The TF-IDF algorithm is one of the basic tools in text analysis, enabling efficient identification of the most important words in documents [13], [18].

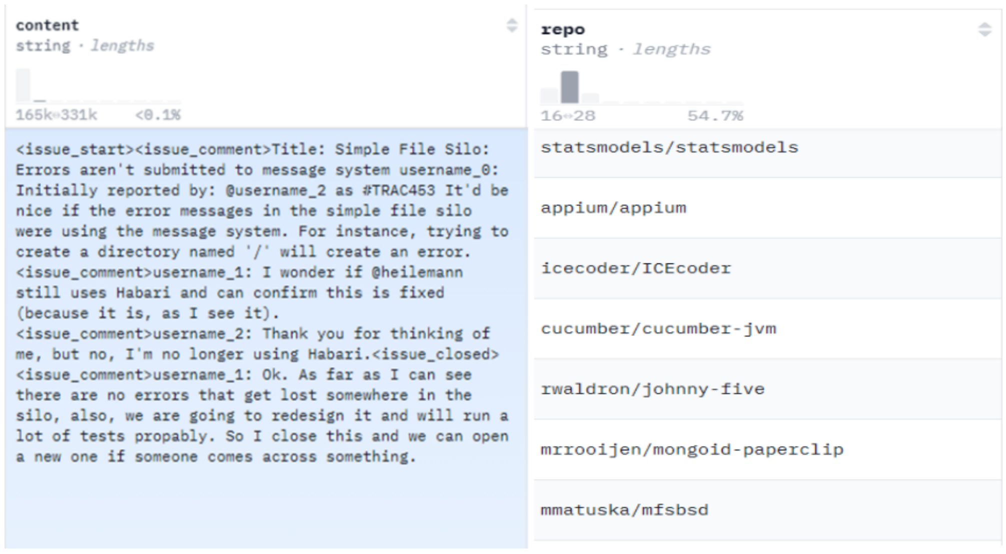
Graph networks enable processing of more information than traditional methods, because they can capture dependencies between data elements with more complex structure. In the task of managing IT projects, this allows for a better understanding of the dependencies between tasks, resources and team members, which supports task allocation and progress monitoring.

4. Problem research solution

This section discusses the process of evaluating the effectiveness of graph neural networks in assigning new problems in a software project to programmers who have the appropriate competences to deal with them. It describes the data used to train the deep learning model, the architecture of neural network, and the evaluation methods.

Dataset description and its preprocessing. The *bigcode/the-stack-github-issues* dataset, available on the *Hugging Face* platform [9], was used in this work. It is a rich collection of issue conversations and pull requests from the GitHub platform. It includes events such as opening issues, comments, and closing issues, and also contains information about usernames, text content, actions, and identifiers. The dataset contains 30.9 million rows and its total size is 54 GB. The data is mainly available in English [2]. This dataset was chosen mainly because of its size and the wide range of domains from which the records come. It contains issues related to JavaScript libraries as well as low-level drivers written in C. The *bigcode/the-stack-github-issues* dataset has been processed: automated texts have been removed, bot comments have been filtered, high-quality conversations have been ensured, and personal data has been anonymized.

The most important column in this data set is the *content* column, which contains the content of the issue along with the entire conversation associated with it, saved in markdown format (Fig. 1). To divide the text into content and comments, the issue has been separated by tags, e.g. <*issue_comment*>, which indicates where a new comment begins. This gives many possibilities for further processing of this collection.



content string · lengths 165k↔331k <0.1%	repo string · lengths 16↔28 54.7%
<issue_start><issue_comment>Title: Simple File Silo: Errors aren't submitted to message system username_0: Initially reported by: @username_2 as #TRAC453 It'd be nice if the error messages in the simple file silo were using the message system. For instance, trying to create a directory named '/' will create an error. <issue_comment>username_1: I wonder if @heilemann still uses Habari and can confirm this is fixed (because it is, as I see it). <issue_comment>username_2: Thank you for thinking of me, but no, I'm no longer using Habari.<issue_closed> <issue_comment>username_1: Ok. As far as I can see there are no errors that get lost somewhere in the silo, also, we are going to redesign it and will run a lot of tests propably. So I close this and we can open a new one if someone comes across something.	statsmodels/statsmodels
	appium/appium
	icecoder/ICEcoder
	cucumber/cucumber-jvm
	rwaldron/johnny-five
	mrrooijen/mongoid-paperclip
	mmatuska/mfsbsd

Fig. 1. Example content column and repo column element in bigcode/the-stack-githubissues.

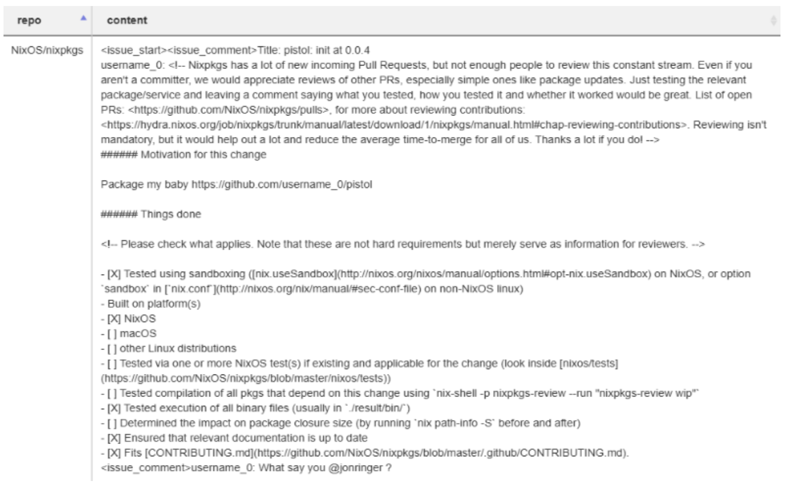
Due to the size of the set (54GB), only a fragment of this set (10GB) was used in the experiments, from which 4 data sets were then generated simulating different test cases differing in size and data diversity. The data sets were created based on the *repo* column and each of them is built from several to a dozen or so repositories of different sizes (Fig. 1). Unfortunately, the *bigcode/the-stack-github-issues* collection does not provide information on who solved a given issue from a repository, so some simplification was necessary. In this work, it was assumed that each repository containing issues was created by a single team/person. This is a necessary assumption so that the datasets can be used to train a model that assigns tasks to developers based on their competencies.

Filtering a data set. After loading the entire dataset (10GB), the first stage of filtration is to find repositories that meet the conditions of the number of issues contained in the repository. The *the-stack-github-issues* dataset is mainly composed of very small repositories containing from 1 to 10 issues, on which it would be difficult to train the model due to the lack of characteristic features. Therefore, only repositories meeting specific parameters of quantity were used to construct datasets simulating different test cases, e.g. the number of issues from 128 to 256. Each test set differs in composition and number of repositories, which allows to test the model in different situations.

The next stage was transformations on the already limited test set, which contains up to 32 repositories depending on the scenario. Unnecessary columns are removed from the set: issue id, issue number, pull request, events, text size, usernames. These fields do not contain important information in the context of analyzing competencies needed to complete the problem (Fig. 2).

After cleaning and extracting the relevant content, the data contains only the most important information: the name of the repository, the processed content, and the extracted code. This data structure allows for more efficient and targeted processing in the subsequent stages of analysis.

Keyword extraction. Extracting keywords from the processed text is a key step in the analysis, which allows for the identification of the most important concepts and topics in the dataset. This process begins with converting the text to lowercase, which aims to avoid the



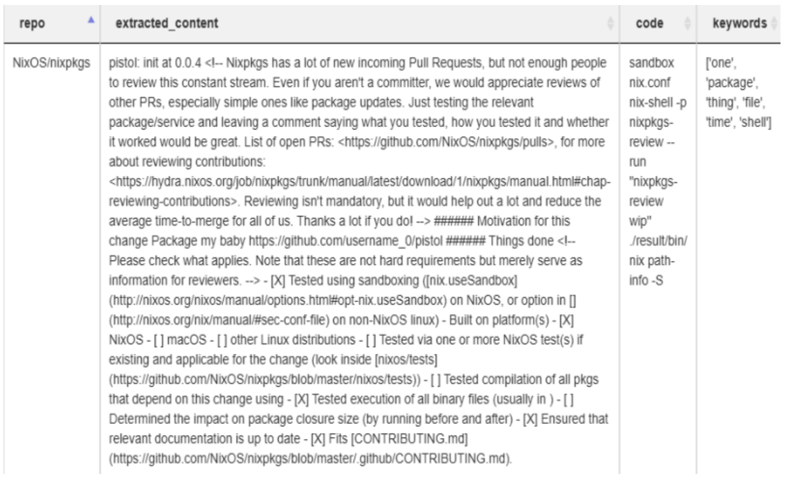
repo	content
NixOS/nixpkgs	<issue_start><issue_comment>Title: pistol: init at 0.0.4 username_0: <!-- Nixpkgs has a lot of new incoming Pull Requests, but not enough people to review this constant stream. Even if you aren't a committer, we would appreciate reviews of other PRs, especially simple ones like package updates. Just testing the relevant package/service and leaving a comment saying what you tested, how you tested it and whether it worked would be great. List of open PRs: <https://github.com/NixOS/nixpkgs/pulls>, for more about reviewing contributions: <https://hydra.nixos.org/job/nixpkgs/trunk/manual/latest/download/1/nixpkgs/manual.html#chap-reviewing-contributions>. Reviewing isn't mandatory, but it would help out a lot and reduce the average time-to-merge for all of us. Thanks a lot if you do! --> ###### Motivation for this change Package my baby https://github.com/username_0/pistol ###### Things done <!-- Please check what applies. Note that these are not hard requirements but merely serve as information for reviewers. --> - [X] Tested using sandboxing ([nix.useSandbox](http://nixos.org/nixos/manual/options.html#opt-nix.useSandbox) on NixOS, or option `sandbox` in [`nix.conf`](http://nixos.org/nix/manual/#sec-conf-file) on non-NixOS linux) - Built on platform(s) - [X] NixOS - [ ] macOS - [ ] other Linux distributions - [ ] Tested via one or more NixOS test(s) if existing and applicable for the change (look inside [nixos/tests](https://github.com/NixOS/nixpkgs/blob/master/nixos/tests)) - [ ] Tested compilation of all pkgs that depend on this change using `nix-shell -p nixpkgs-review --run "nixpkgs-review wip"` - [X] Tested execution of all binary files (usually in `./result/bin/`) - [ ] Determined the impact on package closure size (by running `nix path-info -S` before and after) - [X] Ensured that relevant documentation is up to date - [X] Fits [CONTRIBUTING.md](https://github.com/NixOS/nixpkgs/blob/master/.github/CONTRIBUTING.md). <issue_comment>username_0: What say you @jonringer ?

Fig. 2. Initial structure of the dataset after removing unnecessary columns.

repetition of tokens with the same meaning that could occur in different spellings, e.g. in the case of proper names. Unifying the text to lowercase allows for a more consistent analysis, reducing the risk of treating the same word written in different letters as different tokens.

Next, linguistic analysis is performed, which consists of extracting nouns and proper names. These nouns are important because they most often represent important concepts and topics in the analyzed text. To further unify the dataset, all extracted nouns are transformed to the singular form using an appropriate linguistic tool, which unifies concepts that are plural and singular.

The next step is to calculate the centroid, which is the average vector of all words, which represents the "center" of the meaning of the entire set of words. Then, the cosine similarity between the vector of each word and the centroid is calculated. Based on the calculated similarity values, the words are sorted in descending order. The most important words, with the highest similarity to the centroid, are extracted as keywords. Thanks to this process, it is possible to effectively identify the main topics and concepts that dominate the analyzed content of problem. Result of this transformation is dataset containing content, code, keywords (Fig. 3).



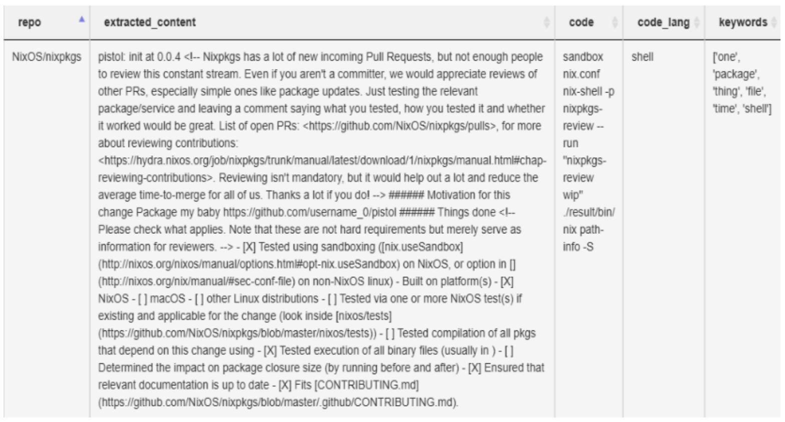
repo	extracted_content	code	keywords
NixOS/nixpkgs	pistol: init at 0.0.4 <!-- Nixpkgs has a lot of new incoming Pull Requests, but not enough people to review this constant stream. Even if you aren't a committer, we would appreciate reviews of other PRs, especially simple ones like package updates. Just testing the relevant package/service and leaving a comment saying what you tested, how you tested it and whether it worked would be great. List of open PRs: <https://github.com/NixOS/nixpkgs/pulls>, for more about reviewing contributions: <https://hydra.nixos.org/job/nixpkgs/trunk/manual/latest/download/1/nixpkgs/manual.html#chap-reviewing-contributions>. Reviewing isn't mandatory, but it would help out a lot and reduce the average time-to-merge for all of us. Thanks a lot if you do! --> ###### Motivation for this change Package my baby https://github.com/username_0/pistol ###### Things done <!-- Please check what applies. Note that these are not hard requirements but merely serve as information for reviewers. --> - [X] Tested using sandboxing ([nix.useSandbox](http://nixos.org/nixos/manual/options.html#opt-nix.useSandbox) on NixOS, or option in [](http://nixos.org/nix/manual/#sec-conf-file) on non-NixOS linux) - Built on platform(s) - [X] NixOS - [ ] macOS - [ ] other Linux distributions - [ ] Tested via one or more NixOS test(s) if existing and applicable for the change (look inside [nixos/tests](https://github.com/NixOS/nixpkgs/blob/master/nixos/tests)) - [ ] Tested compilation of all pkgs that depend on this change using - [X] Tested execution of all binary files (usually in) - [ ] Determined the impact on package closure size (by running before and after) - [X] Ensured that relevant documentation is up to date - [X] Fits [CONTRIBUTING.md](https://github.com/NixOS/nixpkgs/blob/master/.github/CONTRIBUTING.md).	sandbox nix.conf nix-shell -p nixpkgs-review --run "nixpkgs-review wip" ./result/bin/ nix path-info -S	['one', 'package', 'thing', 'file', 'time', 'shell']

Fig. 3. Dataset structure after keyword extraction.

Programming language detection. If there is a code fragment in the content column, it is extracted and analyzed separately from the rest of the problem content. If the code and content were analyzed using a FastText-based method, the specific code syntax would have a very large impact on the calculation of the text centroid. Separating these two analyses allows for more reliable results.

The open-source Guesslang tool was used to analyze the text. Guesslang uses advanced machine learning techniques and TensorFlow tools to build, train, and evaluate a text classification model that recognizes programming languages [12].

One of the problems with this solution is the length of the code in the analyzed problems. In many languages, the semantics can be very similar, e.g. in languages related to C. This results in incorrect classifications of short code fragments that constitute a large part of the set. Quite often, only a code fragment is pasted into the problem description, after which it is not easy to determine what programming languages it is. In such situations, the result of programming language detection is not taken into account in subsequent stages of problem analysis (Fig. 4).



repo	extracted_content	code	code_lang	keywords
NixOS/nixpkgs	pistol: init at 0.0.4 <!-- Nixpkgs has a lot of new incoming Pull Requests, but not enough people to review this constant stream. Even if you aren't a committer, we would appreciate reviews of other PRs, especially simple ones like package updates. Just testing the relevant package/service and leaving a comment saying what you tested, how you tested it and whether it worked would be great. List of open PRs: <https://github.com/NixOS/nixpkgs/pulls>, for more about reviewing contributions: <https://hydra.nixos.org/job/nixpkgs/trunk/manual/latest/download/1/nixpkgs/manual.html#chap-reviewing-contributions>. Reviewing isn't mandatory, but it would help out a lot and reduce the average time-to-merge for all of us. Thanks a lot if you do! --> ###### Motivation for this change Package my baby https://github.com/username_0/pistol ###### Things done <!-- Please check what applies. Note that these are not hard requirements but merely serve as information for reviewers. --> - [X] Tested using sandboxing ([nix.useSandbox] (http://nixos.org/nixos/manual/options.html#opt-nix.useSandbox) on NixOS, or option in [] (http://nixos.org/nix/manual/#sec-conf-file) on non-NixOS linux) - Built on platform(s) - [X] NixOS - [ ] macOS - [ ] other Linux distributions - [ ] Tested via one or more NixOS test(s) if existing and applicable for the change (look inside [nixos/tests] (https://github.com/NixOS/nixpkgs/blob/master/nixos/tests)) - [ ] Tested compilation of all pkgs that depend on this change using - [X] Tested execution of all binary files (usually in) - [ ] Determined the impact on package closure size (by running before and after) - [X] Ensured that relevant documentation is up to date - [X] Fits [CONTRIBUTING.md] (https://github.com/NixOS/nixpkgs/blob/master/.github/CONTRIBUTING.md).	sandbox nix.conf nix-shell -p nixpkgs-review --run "nixpkgs-review wip" ./result/bin/ nix path-info -S	shell	['one', 'package', 'thing', 'file', 'time', 'shell']

Fig. 4. Dataset structure after programming language detection.

Selection of the most relevant keywords. Keyword extraction aims to find words that best define the problem. Due to the specific language in which problem descriptions are written, many extracted keywords do not carry information about the technology or the archetype of the problem. For this purpose, an alternative approach was developed, which is based on the TF-IDF method. The TF-IDF value is calculated as the product of two factors: the frequency of a term in a document (TF) and the inverse frequency of documents in which the term appears (IDF) [10].

Using this method, we can automatically assess the importance of individual keywords in the context of the entire dataset. Terms that appear frequently in few documents receive higher TF-IDF values, which allows them to be identified as more relevant and domain-specific. In this way, it is possible to filter keywords effectively, without the need for manual supervision, favoring terms that relate to IT problems. These relevant keywords will be the basis for the construction of a problem graph, in which connections between nodes will occur when problems share the same relevant keyword (class).

Graph creation. The next stage of data set transformation is the construction of the graph itself, on which the model will be trained. The most important assumption of graph construction is that the edges of the graph are created between nodes that have the same important keywords. The graph nodes represent individual problems from which keywords were extracted. Each node contains a full set of keywords, both important and those of lesser importance, which allows for maintaining the full context of each problem description. The edges, on the other hand, are created solely on the basis of important keywords, which connect nodes representing problems with common, key features such as programming language, problem archetype or technology used.

After the complete transformation of the datasets into graphs, the proprietary model based on the *GraphSAGE* architecture was trained on each of the training sets to learn an efficient representation of the nodes based on their local neighborhoods. During the training process, the model iteratively updates its weights to improve its ability to classify nodes based on their attributes and graph structure. The ultimate goal is to obtain a model that can accurately classify unknown nodes based on the information available in the graph, using both the features of the nodes and their connections to other nodes.

The architecture of graph neural network used in our work was based on *GraphSAGE* (*Graph Sample and Aggregation*) model. This model was chosen for its ability to scale to large graphs and efficiently learn node representations. GraphSAGE is particularly effective in tasks where it is critical to predict properties for nodes that have not been seen before by the model, which is common in real-world applications. The proposed model based on the GraphSAGE architecture, has three convolutional layers (*SAGEConv*) as (Fig. 5):

1. The first layer takes as input vectors representing keywords (x) and graph edge indexes (*edge_index*). This layer uses 64 dimensions of embedding vectors that transform input features by aggregating information from neighboring nodes. This allows the model to capture the local context of each node.

2. The second convolutional layer consists of 128 dimensions of embedding vectors, which allows the model to capture more complex patterns in the data.

3. The third layer, acting as the output, transforms the data to a dimension corresponding to the number of classes predicted by the model.

Each convolution operation is supported by a normalization layer (*BatchNorm*), and the results from the convolutional layers are activated using the *ReLU* (*Rectified Linear Unit*) function. The ReLU function introduces nonlinearity to the model, which is crucial for its ability to model complex dependencies in the data. Batch normalization helps to speed up the learning process of the network and stabilizes its course by reducing the problem of changing input distributions, known as *internal covariate shift*.

Finally, after passing through successive convolutional layers, the model generates output representations of the nodes, which are then processed by the *log_softmax* function. This function transforms the resulting vectors into probabilities of the nodes belonging to the classes, which is necessary for classifying the nodes in the graph. The model optimization process is carried out using the *AdamW* (*Adam with Weight Decay*) optimizer, which was chosen for its effectiveness in training deep neural networks. In this case, a learning rate of 0.005 and *weight decay* factor of 0.001 were used, which helps control overfitting of the model to the training data. Additionally, a learning rate decay schedule (*ReduceLROnPlateau*) was used, which allows automatic adjustment of the learning rate in response to a plateau in the improvement of model results.

Graph neural network training. Graph neural network training enables the GraphSAGE model to efficiently learn the representation of nodes in the graph. This process starts with dividing the data into training, validation, and test sets, which allows the model's performance to be assessed at different stages of training. The data is divided into these three sets randomly, with 80% of the data allocated to the training set and remaining 20% divided equally between validation and testing. Each epoch of the GraphSAGE model training consists of several steps.

5. Analysis of results

Model accuracy is one of the most important metrics for assessing its effectiveness in assigning problems to developers. In the context of this study, accuracy is defined as the ratio of the number of correctly assigned problems to the total number of test cases. To increase the precision of the model, different optimization approaches were used, including the selection of appropriate

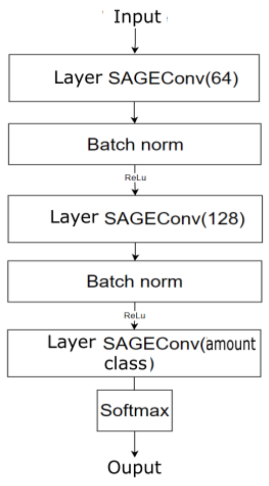


Fig. 5. Architecture of proposed model based on GraphSAGE.

cost functions and advanced optimization algorithms. The experiment included two different methods for assessing accuracy. The first is traditional approach, in which accuracy is measured by directly assigning a problem to 1 person. The 2nd approach takes into account the margin of error, assuming that the model works correctly if the person selected by the model is among the three highest-rated candidates for a given problem. This is possible thanks to the use of softmax function, that returns percentage probability of assigning a problem to individual developers.

The experiment created four datasets, differing in the structure and content of the data they contained. The repositories included in these datasets were randomly selected, with the main selection criterion being the number of issues they contained. This selection method ensures that the repositories are diverse in terms of their scale and allows for capturing the diversity of issues that development teams may encounter. The classification of repositories by size was performed as follows: (1) Large repository: from 1024 to 2048 issues; (2) Medium repository: from 512 to 1024 issues; (3) Small repository: from 256 to 512 issues.

The random selection of repositories with different numbers of issues allows for maintaining diversity in key areas, such as programming languages, technologies used in the repositories, and types and complexity of issues. This allows for more representative results that will be applicable to a wide range of real-world development scenarios. Additionally, important keywords were extracted for each set. On these sets, 4 original models were tested (one model for each set) based on the GraphSAGE architecture described in the previous section.

The Z1 dataset is designed to simulate a team of about 10 developers. They have 3 small projects and 2 medium-sized projects, each of which is a different technology. This dataset is particularly diverse in terms of the technologies used and the types of problems, which allows for thorough testing of the model in a diverse environment. The list of repositories used to create this dataset includes: department-of-veterans-affairs/vets-website (1023 problems), pixijs/pixi.js (1016 problems), wasmerio/wasmer (256), airbnb/react-dates (242), TypeStrong/typedoc (225). Keywords for Z1 set: sql, information, yaml, text, team, link, typescript, javascript, update, testing, groovy, user, page, version, behavior, pixi, add, gov, bug, definition, documentation, new, texture, acceptance, typedoc, batchfile, pii, sprite, va, department.

The Z2 collection represents a more diverse environment, where teams work on projects of varying sizes, with a predominance of medium to large projects. It contains a variety of projects that allow testing the model in conditions requiring handling large amounts of data and complex problems. The list of repositories used to create this collection includes: prestodb/presto (1356 problems), ValveSoftware/Proton (877 problems), Microsoft/vcpkg (755), chapmanb/bcbio-nextgen (242), pytorch/audio (207), opensearch-project/OpenSearch-Dashboards (154), ruma/ruma (129), peeringdb/peeringdb (128). Keywords for the Z2 set are: proton, driver, com, appid, vcpkg, version, log, release, shell, information, presto, link, groovy, update, steam, csv, markdown, report, compatibility, yaml, cmake, sql, build, prestodb, support, java, install, query, add, gist, makefile, batchfile, game.

The Z3 dataset represents a complex environment dominated by large projects with many problems. This dataset is ideal for testing the model in high-scale settings where the diversity of problems and technologies poses challenges to project management. The list of repositories used to create this collection includes: facebook/jest (1845 problems), prestodb/presto (1356 problems), ValveSoftware/Proton (877), gohugoio/hugo (866), Microsoft/vcpkg (755), deeplearning4j/deeplearning4j (754), wenzhixin/bootstrap-table (722), google/site-kit-wp (684), commons-app/apps-android-commons (610), mdn/sprints (604), pydanny/cookiecutter-django (242), redux-saga/redux-saga (237), tilt-dev/tilt (225), magda-io/magda (222), StylishThemes/GitHub-Dark (207), enactjs/enact (204), agershun/alasql (177), slackapi/node-slack-sdk (172), derekparker/delve (171), apache/tinkerpop (160), WebAssembly/design (154), linvi/tweetinvi (141), Simulated-GREG/ (139), scylladb/scylla-manager (129), sphinx-gallery/sphinx-gallery (128), actor-frame/actor-framework (128), Keywords for the Z3 set are: report, feature, update, dockerfile, markdown, sql, com, steam, groovy, javascript, pull, table, go, api, kotlin, add, presto, yaml, html, org, batchfile, typescript, run, version, information, jest, using, request, game, makefile, csv, coffeescript, support, bug, ruby, shell, css, page, new, proton, cmake, problem.

The last test set of Z4, here called the "Balanced Set", is used as a tool to analyze potential model overfitting. The projects in this set have a very similar number of 46 problems, which allows us to see how the model performs under conditions where the size of the projects is the same. This allows us to determine whether the model overfits the data. The list of repositories used to create this set includes: udacity/create-your-own-adventure (380 problems), openMSX/openMSX (367 problems), google/filament (337), nodejs/node-gyp (334), PHPOffice/PhpSpreadsheet (297), GoogleChrome/workbox (294), getferdi/ferdi (275), opentargets/platform (264)). Keywords for the Z4 set are: create, groovy, reviewable, filament, javascript, story, commit, npm, bug, request, yaml, com, io, md, workbox, phpspreadsheet, version, err, gyp, behavior, review, node, build, src, ferdi, batchfile, add, adventure, php.

The entire implementation of the task allocation automation solution was done in Python, using version 3.10. For efficient data processing, programs were run in an environment with access to GPU, which significantly accelerated the graph model training. The pip tool was used to manage libraries and dependencies, and the project was developed in a virtual environment.

The main way of evaluating the models was to analyze the accuracy measures such as F1 Score, precision (Precision) and sensitivity (Recall). Below are presented the results obtained for the models trained on different data sets: small company (set Z1), medium company (set Z2), large company (set Z3) and sustainable company (set Z4). These results allow to evaluate the effectiveness of the proprietary models based on the GraphSAGE architecture in different contexts and with different data complexity. The proposed model, based on the GraphSAGE architecture accordingly achieved the results presented in table 1: M1 trained on a dataset of small company (set Z1), M2 trained on a dataset of medium-sized company (set Z2), M3 trained on a dataset of large company (set Z3), M4 trained on a dataset of company with repositories balanced in the number of problems (set Z4).

Table 1. Evaluation results of models M1, M2, M3 and M4.

	M1	M2	M3	M4–Exact	M4–Top3
F1 Score	0.6313	0.5561	0.3448	0.5172	0.7668
Precision	0.6358	0.5712	0.3379	0.5356	0.7810
Recall	0.6269	0.5424	0.3519	0.5000	0.7810

The results for the Z1 data set seem promising, the M1 model obtained almost 64% accuracy for the test data and learned 80 epochs. Around the 30th epoch, the model achieved the best results, which later decreased. The model strongly favors the first person (we assume that each repository was handled by one person), i.e. the one who completed 1023 problems.

The results for the Z2 data set are slightly worse than for the Z1 data set. The M2 model obtained 57% accuracy despite the larger number of people to whom tasks could be assigned. Similarly to the M1 model, the model's effectiveness was high around the 30th epoch and then decreased. Similarly, the M2 model also tends to favor people who completed more problems. This is logically justified, because the more problems a person has done, the more potential connections they have with new problems that appear in the graph. However, favoritism is "balanced" in the M2 model, each of the 3 people who did more problems than the other 5 generates false predictions to a similar extent. This cannot be said about the M1 model, where the greater part of false predictions was generated by 1 person who had a comparable number of problems to person number 2. Therefore, despite having lower efficiency, the M2 model seems to be more balanced than the M1 model.

The M3 model trained on the Z3 dataset aims to test how well the model copes with a large number of people to assign tasks to. The model assigns tasks to M3 between 24 people, which is significantly more than the M1 model (5 people) and the M2 model (8 people). In addition, the Z3 dataset contains 11,909 problems, while the Z2 dataset contains only 3,848. This translates into much worse results than the M1 and M2 models. The M3 model achieved only 34% accuracy. Despite the obvious deficiencies in the M3 model's accuracy, its learning process was much more stable than the other models. For some people who had fewer problems, there are no visible good problem matches, while others have practically only correct matches. Considering the stability of learning and the trends, it can be assumed that the biggest problem of the M3 model is the poor quality of the dataset, where some people have high problem matching efficiency, and others close to zero.

The last of the M4 models was trained on the Z4 set. This set contains people who have a very similar number of problems. The pool of people to be assigned was 8 and the model achieved an efficiency of 54%. Additionally, a test was conducted for this model, simulating real problem assignment in which many people have the competence to perform a given problem, so it was assumed that the assignment is true if the real label is among the 3 most probable labels. For this test, the efficiency was 78%. The model learns less in steps in the case of a larger amount of data. Accuracy values, where the margin of error is taken into account, are more stable than in the case of simple accuracy. The M4 model does not tend to assign tasks to person number 1, despite having 116 more tasks than person number 8 and is generally balanced for the low quality of the data.

6. Conclusions

The research involved the analysis and implementation of deep learning models supporting IT project management processes, in particular the automatic assignment of tasks to programmers based on their competences. To enable experiments, the data sets were created that reflect different types of organizations differing in structure and content. These sets were designed to simulate real working conditions in diverse teams, both in terms of the number of people and the number of problems that these people solved. Models based on the proprietary GraphSAGE architecture were trained on their basis. The tests carried out showed that despite achieving different results, the models tend to favor people who solved a larger number of problems. This is understandable, considering the fact that a larger number of completed tasks provides more data for analysis and drawing conclusions, which leads to more frequent assignment of tasks to such people. This is particularly visible in the results of models, which tend to assign tasks to the most active people, which leads to a certain unbalanced distribution of tasks. However, despite this tendency, the models showed potential and stability in the learning process. The analysis of the objective function graphs and accuracies shows that the models learn in a controlled manner and their performance is satisfactory in many cases. In the case of models, which were tested on more complex and larger data sets, it was noticed that despite the decrease in accuracy, the stability of the learning process was clearly better, which suggests that the GraphSAGE architecture copes well with larger data sets and more complex structures.

References

[1] Agibetov, A., Blagec, K., Xu, H., Samwald, M.: Fast and scalable neural embedding models for biomedical sentence classification. In: BMC Bioinformatics, Vol. 19, pp. 541 (2018)

[2] BigCode Project. bigcode/the-stack-github-issues. [Online] https://huggingface.co/datasets/bigcode/the-stack-github-issues, 2023. Accessed: 10.2024.

[3] Borhan, N., Zulzalil, H., Hassan, S., Mohd Ali, N.: Requirements Prioritization Techniques Focusing on Agile Software Development: A Systematic Literature Review. In: International Journal of Scientific & Technology Research, Vol. 8(11), pp. 2118–2125 (2019)

[4] Boswell, D.: Introduction to support vector machines. In: Department of Computer Science and Engineering University of California San Diego, Vol. 11, pp. 16–17. (2002)

[5] Brower, H.H., Nicklas, B.J., Nader, M.A., Trost, L.M., Miller, D.P.: Creating effective academic research teams: Two tools borrowed from business practice. In: Journal of Clinical and Translational Science, Vol. 5(1), pp. e74 (2020)

[6] Chen, Z., Wang, Z., Yang, Y., Gao, J.: Resgraphnet: Graphsage with embedded residual module for prediction of global monthly mean temperature. In: Artificial Intelligence in Geosciences, Vol. 3, pp. 148–156 (2022)

[7] Ciupe, A., Orza, B., Florea, C., Vlaicu, A.: Skill-oriented priority scheduling for solving the resource constrained project scheduling problem. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), Romania, pp. 85–92 (2015)

[8] Fletcher, T.: Support vector machines explained. Tutorial Paper (2009)

[9] Hugging Face. Hugging face platform. [Online] https://huggingface.co, 2023. Accessed: 11.2024.

[10] Karabiber. Tf-idf—term frequency-inverse document frequency. [Online] https://www.learndatasci.com/glossary/tf-idf-term, 2020. Accessed: 11.2024.

[11] Korytkowski, P., Malachowski, B.: Competence-based estimation of activity duration in it projects. In: European Journal of Operational Research, Vol. 275(2), pp. 708–720 (2019)

[12] Lefever, Y.: Guesslang: Programming language detection. [Online] https://github.com/yoeo/guesslang, 2023. Accessed: 11.2024.

[13] Lubis, A.R., Nasution, M., Sitompul, O.S., Zamzami, F.M.: The effect of the tf-idf algorithm in time series forecasting for word relevance on social media. In: Indonesian Journal of Electrical Engineering and Computer Science, Vol. 22(2), pp. 976–984 (2021)

[14] Milojevic, D., Macuzic, I., Dordevic, A., Savkovic, M., Dapan, M.: Comparative analysis of software tools for agile project management. In: Quality Festival 2023, ISBN 978-86-6335-104-2 (2023)

[15] Momanyi, B.M., Zhou, Y.W., Grace-Mercure, B.K., Temesgen, S.A., Basharat, A., Ning, L., Tang, L., Gao, H., Lin, H., Tang, H.: SAGESDA: Multi-GraphSAGE networks for predicting SnoRNA-disease associations. In: Current Research in Biomedical Sciences, Vol. 7, pp. 100122 (2024)

[16] Montgomery, D.C., Peck, E.A., Vining, G.G.: Introduction to linear regression analysis. In: Wiley (2021)

[17] Priyam, A., Abhijeeta, G.R., Rathee, A., Srivastava, S.: Comparative analysis of decision tree classification algorithms. In: International Journal of Computer Applications, Vol. 3(2), pp. 334–337 (2013)

[18] Qaiser, S., Ali, R.: Text mining: use of TF-IDF to examine the relevance of words to documents. In: Intern. Journal of Computer Applications, Vol. 181(1), pp. 25–29 (2018)

[19] Qureshi, H.A., Shah, Y.A.R., Qureshi, S.M., Shah, S.U.R., Shiwlani, A., Ahmad, A.: The promising role of artificial intelligence in navigating lung cancer prognosis. In: International Journal For Multidisciplinary Research, Vol. 6(4), pp. 1–21 (2023)

[20] Riandini, M., Zarlis, M., Situmorang, Z.: Determination of internship location for outstanding students of smk singosari using the k-means clustering algorithm. IN: AIP Conference Proceedings, Vol. 3065(1), pp. 030016 (2024)

[21] Salehinejad, H., Sankar, S., Barfett, J., Colak, E.: Recent advances in recurrent neural networks. In: arXiv preprint arXiv:1801.01078 (2017)

[22] Sarhadi, P., Naeem, W., Fraser, K., Wilson, D.: On the application of agile project management techniques, v-model and recent software tools in postgraduate theses supervision. In: IFAC-PapersOnLine, Vol. 55(17), pp. 109–114 (2022)

[23] Soroka-Potrzebna, H.: Barriers of knowledge management in virtual project teams: a TISM model. In: Procedia Computer Science, Vol. 207, pp. 800–809 (2022)

[24] Suarez-Varela, J., Almasan, P., Ferriol-Galmes, M., Rusek, K., Geyer, F., Cheng, X.: Graph neural networks for communication networks: Context, use cases and opportunities. In: IEEE Network, Vol. 37(3), pp. 146–153 (2022)

[25] Unnikrishnan, R., Kamath, S., Ananthanarayana, V.S.: Benchmarking shallow and deep neural networks for contextual representation of social data. In: Proceedings of 18th India Council International Conference (INDICON), India, pp. 1–8 (2021)

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proceedings of 31st International Conference on Neural Information Processing Systems, pp. 6000–6010 (2017)

[27] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: Proc. of International Conference on Learning Representations (2017)

[28] Vrahatis, A.G., Lazaros, K., Kotsiantis, S: Graph attention networks: A comprehensive review of methods and applications. In: Future Internet, Vol. 16(9), pp. 318 (2024)

[29] Wu, L., Chen, Y., Shen, K., Guo, X., and Gao, H., Li, S.: Graph neural networks for natural language processing: A survey. In: IEEE (2023)

[30] Zhang, Y., Yu, X., Cui, Z., Wu, S., Wen, Z., Wang, L.: Every document owns its structure: Inductive text classification via graph neural networks. In: arXiv:2004.13826 (2020)