Automated Detection of Typed Links in Issue Trackers

Clara Marie Lüders, Tim Pietz, and Walid Maalej
Universität Hamburg, Hamburg, Germany
E-Mail: clara.marie.lueders@uni-hamburg.de, tim.pietz@uni-hamburg.de, walid.maalej@uni-hamburg.de

***Abstract*—Stakeholders in software projects use issue trackers like JIRA to capture and manage issues including requirements and bugs. To ease issue navigation and structure project knowledge, stakeholders manually connect issues via links of certain types that reflect different dependencies, such as Epic-, Block-, Duplicate-, or Relate- links. Based on a large dataset of 15 JIRA repositories, we study how well state-of-the-art machine learning models can automatically detect common link types. We found that a pure BERT model trained on titles and descriptions of linked issues significantly outperforms other optimized deep learning models, achieving an encouraging average macro F1-score of 0.64 for detecting 9 popular types across all repositories (weighted F1-score of 0.73). For the specific Subtask- and Epic-links, the model achieved top F1-scores of 0.89 and 0.97, respectively. Our model does not simply learn the textual similarity of the issues. In general, shorter issue text seems to improve the prediction accuracy with a strong negative correlation of -0.70. We found that Relate-links often get confused with the other links, which suggests that they are used as default links likely in unclear cases. We also observed significant differences across the repositories, depending on how they are used and by whom.**

***Index Terms*—Issue Management, Issue Tracking Systems, Dependency Management, Duplicate Detection, Mining Issue Tracker, JIRA, BERT.**

I. Introduction

Software development teams use issue trackers to manage and maintain software products and their requirements. Popular issue trackers in practice are Bugzilla[1], GitHub Issues[2], and JIRA[3], with common features like issue creation and tracking, communication around issues, release planning, issue triaging, and issue dependency management. As a central entity, issues have many properties including their type [18] (e.g. requirement, feature request, bug report, or task), priority (e.g. low, high, critical), and assignee. Issue trackers and issue management in general have been the focus of software engineering and requirements engineering research for over a decade, with promising automation approaches, e.g., on issue type prediction (and correction) [12], [13], [21], [27], [36], priority (and escalation) prediction [7], [8], [20], [23], or triaging and assignment [14], [32].

Issues are often interconnected via *links* [16], [18], which enables stakeholders to capture the relationships between the issues and structure the overall project knowledge. Depending on the issue tracker and the project, these links can have different types. Popular link types are *Relate* for capturing a general relation as well as *Subtask* and *Epic* for capturing issue hierarchies. *Duplicate* links are particularly popular in open source projects and issue trackers open to many users. This specific link type has attracted much attention from research in recent years [4], [11], [15], [35]. Finally, *Depend* or *Block* links capture causal or workflow dependencies.

Issue linking is an important part of issue management and software engineering in general. Research has found that linking helps reduce issue resolution time [16] and prevent software defects [28]. Missing or incorrect links can particularly be problematic for requirements analysis and release planning [33]. For instance, missing *Depend* or *Block* links to issues assigned to a specific release might be crucial for that release. Similarly, missing duplicate links might lead to missing additional information [1], which can particularly be relevant for testing.

As issue trackers are getting more central for documenting almost all project activities, particularly in agile projects, and with the evolution of software products over time, a project might easily get thousands of issues. Each new issue might thus have hundreds of thousands of potentially relevant links. Correctly identifying and connecting issues quickly becomes difficult, time consuming, and error-prone [10], [19]. This is even worse in popular issue trackers like those of Apache, RedHat, or Qt, which are open to users and other stakeholders and which may contain hundreds of thousands of issues. The problem becomes also more complex when link types are taken into account: the stakeholder or the issue maintainer does not only have to decide if a link between two issues exist but also what the correct link type is.

To tackle this problem, we report on a novel study, which aims at automatically predicting the issue links including their specific type. Our work is enabled by the advances in deep learning technology (particularly for natural language texts) as well as by the availability of large issue datasets. Our work has two main contributions. First, we compare multiple state-of-the-art end-to-end deep learning models to predict and classify the issue links. Our evaluation shows that a BERT model using the description and title of the two involved issues is by far the most accurate to classify typed links. There are barely any limitations for the applicability of the model, as the issue types are not predefined but learned from the data. The results are very encouraging, particularly as up to 13 link types are predicted. Second, we compare the properties of the various

[1]https://www.bugzilla.org
[2]https://github.com
[3]https://www.atlassian.com/software/jira

repositories and link types and analyse potential correlations with the performance of the model. The results reveal insights on link types confusion as well as on how to design and apply link prediction tools in practice. In addition to the public dataset of 15 JIRA repositories, we share our code and analysis scripts to ease replication.[4]

The remainder of the paper is structured as follows. Section II outlines our research questions, method, data, and the various classification models used. Section III presents the performance results of the BERT typed link prediction model across 15 different repositories. In Section IV we compare the repositories and the link types with the factors that influence prediction performance. In Section V, we discuss our findings, their limitations, possible optimization strategies, and implications for different stakeholders. Finally, we discuss related work in Section VI and conclude the paper with Section VII.

II. Research Setting

We briefly outline our research questions, methods, research data, as well as the machine learning models investigated.

A. Research Questions and Data

Motivated by a) the importance of the typed link prediction in issue management [16], [28], [33], b) recent work on issue duplicate prediction [11], c) recent advances of transformer-based machine learning technique for natural language processing BERT [6], as well as c) the availability of large issue datasets [18], we studied typed links prediction in issue trackers focusing on two main questions:

- **RQ1.** How well can machine learning predict the different link types in issue trackers?
- **RQ2.** What are possible explanations for the performance differences between repositories and link types and what can we learn from the differences?

For answering the research questions, we used a large dataset of 16 different public JIRA repositories [18]. This dataset perfectly match our research goals. Not only is JIRA one of the most popular tools for issue management in practice[5] [6][7]; it is also well-known for its support for various link types, which can also be customised depending on project needs. Other available datasets, e.g. of Bugzilla [15], only focus on the specific link type *Duplicate* and have intensively been used in software engineering literature (particularly mining software repositories) so far [4], [11], [37].

Table I summarizes the analyzed issue repositories. The table shows the number of issues, links, unique link types, coverage, number of projects, and the share of cross-project links. Coverage represents the number of issues having at least one link and the share of cross-project links are the share of links which connect issues of different projects in a repository.

[4]https://github.com/RegenKordel/LYNX-TypedLinkDetection

[5]https://enlyft.com/tech/products/atlassian-jira

[6]https://www.datanyze.com/market-share/project-management--217/jira-market-share

[7]https://www.slintel.com/tech/bug-and-issue-tracking

TABLE I: Studied JIRA repositories in alphabetical order.
Columns: Documented Link Types (#Types); percentage of issues with a link (%Cov.); the percentage of links for issues from two different subprojects. (%CP)

Repo.	#Issues	#Links	#Types	#Pro.	%Cov.	%CP
Apache	**1014926**	**255767**	**16**	**646**	28.5%	5.2%
Hyperledger	28146	16304	8	32	**54.9%**	4.6%
IntelDAOS	9474	2599	11	**2**	30.8%	3.8%
JFrog	15535	3229	10	10	28.6%	8.2%
Jira	274545	99819	**16**	30	46.7%	**43.2%**
JiraEcosystem	41866	11398	14	101	33.0%	6.8%
MariaDB	31229	14618	8	11	44.5%	2.5%
Mindville	**2134**	**44**	**4**	7	**4.0%**	4.6%
Mojang	420819	215527	5	8	53.7%	5.4%
MongoDB	137172	63821	14	27	45.2%	19.1%
Qt	148579	40105	11	21	30.2%	6.9%
RedHat	353000	119669	15	241	39.2%	23.5%
Sakai	50550	19803	8	53	42.4%	**1.4%**
SecondLife	1867	631	6	**2**	39.9%	2.4%
Sonatype	87284	4465	11	5	7.0%	1.5%
Spring	69156	14462	7	80	25.6%	10.0%
Total	2686282	882261	164	1276	-	-
Median	59853	15461	10.5	24	36.1%	5.3%

We bolded the minimum and maximum for each metric across all repositories.

The investigated repositories are heterogeneous in the reported properties. The amount of reported issues ranges from 1,867 to 1,014,926, whilte the number of links ranges from 44 links in Mindville and 255,767 links in Apache. The repositories also vary in terms of link types: Mindville uses 4 unique link types while Jira and Apache use 16 unique link types. As 44 training points is too low for a stratified split, we excluded Mindville from the analysis. On average, the coverage is about 36% meaning that 36% of all issues are part of a link. The coverage ranges from 4.0% in Mindville to 54.9% and 53.7% in Hyperledger and Mojang respectively. Except in Jira, RedHat, and MongoDB links rarely cross project boundaries. The majority (95% average) of links are between issues of the same project. The Jira repository, corresponding to the development of the JIRA issue tracker, shows a high coverage among its projects (46.7%).

When analyzing the data, we observed that a private issue can be part of a link. As we have no further information for private issues, we excluded these links from the analysis. We also removed multi-links, e.g. when two public issues where part of multiple links with different types. This affected 1.1% of all public links. We also simplify the analysis by ignoring link direction.

Out of the 31 unique link types found in the dataset, most appear in less than half of all projects. Uncommon link types only represent a small share of the links. Table II shows the frequencies of common link types across the repositories. The highest variance can be observed for *Relate*, *Duplicate*, *Subtask*, and *Epic*. We focus our study on link types that have a share greater than 1% in the respective repository, e.g.: Qt's *Clone* appeared only 0.1% in all of Qt's links and thus will not appear in the results table below. Additionally, to ensure a minimum amount of comparability and generalizability across the repositories, we exclude link types from the analysis if

TABLE II: Popular link types and their usage shares in percent across the studied JIRA repositories.

Repository	Relate	Duplicate	Subtask	Clone	Block	Depend	Epic	Split	Incorporate	Bonfire Testing	Cause	**Coverage**
Apache	28.3	10.1	32.8	1.7	6.1	5.1	4.9	0.0	4.1	0.0	1.2	94.3
Hyperledger	17.2	3.9	27.6	2.9	8.2	/	39.6	0.5	/	0.01	/	100.0
IntelDAOS	39.3	9.7	10.5	1.5	25.5	/	/	/	/	/	/	86.6
JFrog	27.4	19.9	36.0	0.8	/	7.9	/	/	1.4	/	/	93.5
Jira	63.8	21.7	2.5	2.9	1.0	0.2	/	0.2	2.5	0.2	1.8	96.6
JiraEcosystem	22.9	15.3	20.0	1.8	5.9	1.1	24.2	1.2	1.8	0.9	3.9	99.1
MariaDB	51.1	9.4	6.1	/	13.0	/	6.4	0.2	7.9	/	6.0	100.0
Mindville	43.2	38.6	/	15.9	2.3	/	/	/	/	/	/	100.0
Mojang	9.5	90.0	/	0.3	0.1	/	/	/	/	0.1	/	100.0
MongoDB	39.9	13.5	1.4	0.3	/	22.9	15.9	1.2	/	/	1.7	96.7
Qt	22.4	10.6	24.4	0.1	0.03	15.6	13.5	6.7	/	/	/	93.4
RedHat	25.9	4.9	20.8	15.4	15.2	/	/	0.1	8.9	/	2.6	94.0
Sakai	49.0	9.3	17.0	4.8	0.03	13.0	/	/	6.7	0.01	/	100.0
SecondLife	29.5	/	49.8	7.6	/	4.4	/	/	2.2	/	/	93.5
Sonatype	40.0	7.7	30.1	/	/	3.6	0.2	0.1	/	8.1	5.3	95.0
Spring	47.7	12.1	13.4	0.1	/	12.1	11.3	/	/	/	/	96.7
Mean	**34.8**	**18.4**	**20.9**	4.0	7.0	8.6	14.5	**1.1**	4.4	**1.3**	3.2	96.2
Standard Deviation	**14.3**	**21.5**	**13.8**	5.4	8.1	7.2	**12.5**	2.1	3.0	3.0	1.9	3.7

their total share across all repositories was less than 2% (which leads to excluding *Split* and *Bonfire Testing* from the following analysis). As a result, our analysis focuses on the following common link types: *Relate*, *Duplicate*, *Subtask*, *Depend*, *Epic*, *Clone*, *Incorporate*, *Cause*, and *Block*. A detailed description of the link types can be found in [18].

B. Research Method and Machine Learning Models

For answering RQ1, we built, trained and compared multiple deep learning end-to-end models that classify the link types or the absence of a link for an issue pair. This includes a BERT model which recently attracted much attention in the NLP community. For answering RQ2, we focused on the top performing model looked for properties of repositories, the link types as well as the linked issues which correlate with the prediction performances.

We focused our research on a minimal model input, consisting of the title and description. These are rather universal properties and usually available at issue creation time. This means, the researched models do not use any additional properties, such as the issue type or status for the prediction. These properties might improve the prediction performance but could also contain information about the label which could create a bias in the model (e.g. the resolution for duplicate issues is "duplicate").

Our labels consist of the link types used in the repository. We also added *non-links* as labels to the models by randomly selecting closed issues pairs that do not have the resolution property "duplicate". E.g. the model trained on Qt had 8 labels *Relate*, *Subtask*, *Depend*, *Epic*, *Duplicate*, *Split*, *Replace*, and *non-link*. The amount of *non-links* is equal to the mean counts of the other labels. We then randomly split the data for each repository into a training (80%) and a test set (20%). For the model validation, we took 20% of the training data, resulting in a 64/16/20 train-validate-test set. We stratified the data by link type. The test set was only used for the evaluation.

For selecting the model's architecture we checked the software engineering and the Natural Language Processing literature. In the NLP community, BERT and DistilBERT have recently attracted much attention for various NLP tasks. In Software engineering, different approaches for duplicate detection have been recently suggested, with two main model architectures. Single-Channel architectures [4] take the word representation (word2vec, fasttext, or GLOVE) and encodes each issue separately with a siamese network, consisting of a CNN, LSTM, and a feed-forward neural network. Then the two encoded issues are fed into another forward neural network to determine if one issue is a duplicate of the other issue. Dual-Channel architectures [11] use the word representations of two issues, which are two matrices of word embeddings, and constructs a tensor (of size 2 x word dimension x fixed text length) by putting the two matrices on top of each other. Thus, the two issues are encoded jointly.

We experimented with BERT, DistilBERT, as well as the Single-Channel and Dual-Channel architectures using FastText and Word2Vec. DistilBERT is a smaller transformer model trained to imitate BERT through knowledge distillation. It retains most of BERT's general language understanding capabilities while using 40% fewer model weights. We included DistilBERT in our comparison because we thought a model with fewer parameters might perform better on smaller repositories.

In preliminary experiments, we found that BERT outperformed all other models in all setups. BERT outperformed DistilBERT by an average of 0.05 F1-score, the Single-Channel models by an average of 0.21, and the Dual-Channel models by an average of 0.26. The code and results of the evaluated model architectures are included in the replication package. In the remainder of the paper we focus on discussing the results of the BERT model.

We first tokenized the texts of the combined title and description with the tokenizer of bert-base-uncased / distilbert-base-uncased, which is a trained WordPiece tokenizer. Then, we truncated the text to 192 tokens with a *longest first* strategy, meaning if possible we kept all text and otherwise truncated each issue text to 96 tokens. We concatenated the title and description of both issues into one string and used this as input for the BERT model. The output of the BERT model was

then fed into a dense layer, which then predicts label of the link. For the training, we chose AdamW with a learning rate of $5e^{-5}$ and a weight decay of 0.1. The batch size was 96 with BERT and 128 with DistilBERT. We trained for 30 epochs and validated after each epoch.

We report the F1-score per common link type and the macro and weighted averages of the model per repository. Detailed tables for recall and precision are included in the replication package, without specific results which are worth discussing in the paper. We also present the normalized confusion matrix for each repository. We chose the conservative macro F1-score as our primary metric to evaluate performance. Weighted averages tend to overestimate model performance, as models tend to predict instances of majority classes better since there are more data points to learn from. Furthermore, neither using class weights nor SMOTE as strategies to counter the class imbalance showed any improvements. As we are unaware of any baselines of link type detection in JIRA, we only report on our results.

III. Prediction Performance

A. Overall Prediction Results

Table III shows the common link types and their respective F1-scores for each repository. We also present the mean and standard deviation per link type and the macro and weighted F1-score per repository. Overall, the model macro F1-score achieves 0.64 on a multiclass problem with a median of 7 classes per repository. The weighted F1-score goes up to 0.73. The results are far better for all repositories than a simple random prediction which would achieve 0.14 and a simple majority model that would achieve an average of 0.08 macro F1 on the data.

As expected, the results differ across repositories. JFrog only has a macro F1-score of 0.48, while Mojang has a macro F1-score of 0.88. Some of the variances across the repositories can be explained by the size of the training set. E.g.: JFrog, SecondLife, and Sonatype each contain less than 5k links in total, whereas Mojang contains roughly 200k links. We also observe that the performance of the model differs per link type. The model detects *Subtask* and *Epic* links consistently with a top performance. *Duplicate*, *Depend*, and *Cause* seem not as well accurately predictable. The other link types *Depend*, *Clone*, *Incorporate* show mixed results. *Non-Link* show top performance for all but one repository.

Figure 1 plots the macro F1-score against the standard deviation of the F1-scores across the link types. A higher standard deviation means that the model performs well for some but not for all link types. A low standard deviation means that the model performs similarly for all types. Mojang, with a lot of available training data and only three different predicted link types (*Duplicate*, *Relate*, and *non-link*) performs best (highest macro F1-score with lowest standard deviation). The next cluster consists of Hyperledger, IntelDAOS, Jira, MariaDB, and MongoDB: with macro F1-scores ranging from 0.70 to 0.74 and a standard deviation around 0.19. Then, Qt, RedHat, and Sakai perform slightly worse: their macro

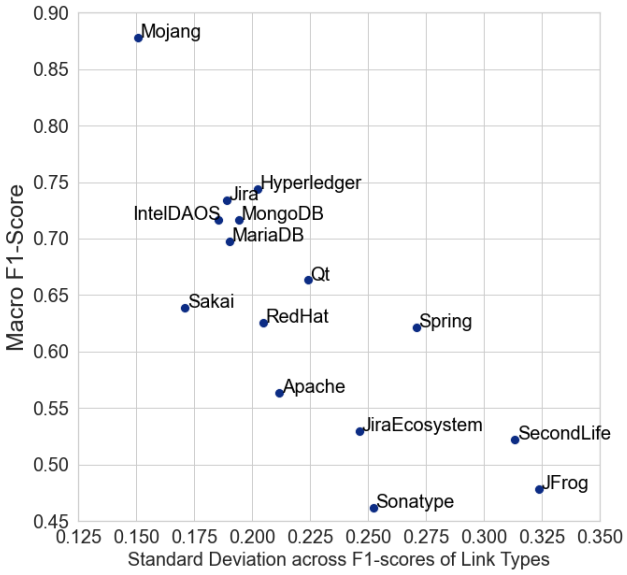

Fig. 1: Performance per repository according to macro F1-score and standard deviation for the link types.

F1-scores lie between 0.63 and 0.66, while their standard deviation is less than 0.225. The last cluster consists of Apache, JFrog, JiraEcosystems, Secondlife, Sonatype, and Spring; these repositories either have a macro F1-score less than 0.60 or a higher standard deviation. These projects all have a lower coverage. The case of Apache is particularly interesting as it has the highest number of links and issues, and one of the highest number of predicted classes. It also contains the most projects (646) which indicates some internal heterogeneity.

Finding 1. A general BERT model applied on issue titles and descriptions predicts the typed links fairly well with a macro F1-score of 0.64 across 15 repositories. Some repositories show a top performance while the model achieved moderate performance for others. The key difference seems to be the link coverage.

B. Individual Link Types and Confusion Analysis

As next step, we took a closer look at the various link types. Figure 2 shows the confusion matrices for each repository, ordered by number of data points per link type in the test set. We identified which link types were confusing to the model and thus consequently mislabeled. Overall, *Relate* is often the majority class and commonly predicted by the model for other link types. This sometimes happens for *Duplicate* but on a much smaller scale. We also observe that *Clone* and *Duplicate* are well distinguished by the model.

***Relate* links.** These links provided in the default JIRA configuration are used in all repositories that we analyzed. They are usually one of the largest classes. The model often

TABLE III: F1-scores for predicting issues links and their types across JIRA repositories.

We color-coded the results: very good: 0.80-1.00; good: 0.60-0.80; moderate: 0.40-0.60; bad: 0.20-0.40; very bad: 0.00-0.20

Repository	Relate	Duplicate	Subtask	Depend	Clone	Incorporate	Epic	Block	Cause	Non-Link	**Macro F1**	**Weight. F1**
Apache	0.64	0.49	0.91	0.46	0.61	0.55	0.97	0.52	0.34	0.76	0.56	0.7
Hyperledger	0.7	0.37	0.89	/	0.8	/	0.97	0.62	/	0.85	0.74	0.85
IntelDAOS	0.69	0.44	0.82	/	0.86	/	/	0.7	/	0.48	0.72	0.68
JFrog	0.57	0.51	0.95	0.45	/	0.0	/	/	/	0.72	0.48	0.66
Jira	0.88	0.71	0.93	/	0.61	0.64	/	/	0.37	0.86	0.73	0.82
JiraEcosystem	0.62	0.57	0.89	0.19	0.59	0.29	0.98	0.41	0.38	0.6	0.53	0.71
MariaDB	0.76	0.37	0.86	/	/	0.68	0.97	0.66	0.52	0.76	0.7	0.72
Mojang	0.7	0.97	/	/	/	/	/	/	/	0.96	0.88	0.95
MongoDB	0.73	0.46	0.85	0.69	/	/	0.97	/	0.36	0.72	0.72	0.72
Qt	0.56	0.43	0.93	0.75	/	/	0.96	/	/	0.81	0.66	0.71
RedHat	0.63	0.29	0.9	/	0.83	0.77	/	0.63	0.4	0.72	0.63	0.7
Sakai	0.72	0.39	0.88	0.48	0.65	0.57	/	/	/	0.78	0.64	0.68
SecondLife	0.73	/	0.87	0.4	0.4	0.0	/	/	/	0.85	0.52	0.74
Sonatype	0.68	0.33	0.91	0.42	/	/	/	/	0.26	0.73	0.46	0.65
Spring	0.73	0.38	0.82	0.48	/	/	0.99	/	/	0.74	0.62	0.69
Mean	0.69	0.48	0.89	0.48	0.67	0.44	0.97	0.59	0.38	0.76	0.64	0.73
Standard Dev.	0.08	0.17	0.04	0.15	0.14	0.28	0.01	0.1	0.07	0.11	0.11	0.08

mistakes other link types as *Relate* links. The F1-score ranges from 0.63 up to 0.91. It is predicted fairly well with an average macro F1-score of 0.73.

***Duplicate* & *Clone* links.** We originally assumed that the model will struggle to distinguish *Clone* links from *Duplicate* and vice versa. Surprisingly, the existence of the class *Clone* did not confuse the model. They were rarely mistaken for each other, which can be observed in the confusion matrices of Apache, Hyperledger, IndelDAOS, Jira, JiraEcosystem, RedHat, and Sakai. This suggests that they exhibit a difference that the model can recognize. *Duplicate* links have a rather mixed F1-scores, ranging from 0.25 to 0.97. Only in Jira and Mojang *Duplicate* links were better classified with 0.67 and 0.97 F1-scores, respectively. The other repositories achieved a macro F1-score up to 0.50. In contrast to *Duplicate* issues that are usually created independently by different stakeholders, *Clone* links are usually intentionally created via the "Clone" feature of JIRA. They seem to be easier detectable by the model with F1-scores ranging from 0.40 to 0.86.

***Subtask* & *Epic* links** The model classified *Subtask* and *Epic* links extremely well, with an average F1-score of 0.89 and 0.97 respectively. *Subtask* links ranging from 0.82 up to 0.95 and *Epic* links ranging from 0.97 up to 0.99, both also have an a low standard deviation. *Subtask* and *Epic* links have an extra section or field and are not grouped under the "Issue Links" Section, they have a special standing in JIRA. Surprisingly, the model was able to differentiate *Subtask* links from *Epic* links.

***Depend*, *Incorporate*, *Block*, & *Cause* links** All of these link types had spread performances or performed not well.

Depend links' performance was very varied, ranging from 0.19 to 0.75. This could be explained by the link type share, Qt, with the best performance, has approx. 16% of its links with *Depend* link types, followed by MongoDB, with approx. 23%. When the performance was bad, usually *Depend* was not used much (absolute counts less than 300) or confused with *Relate* (for Spring).

Incorporate links were also varied in performance with a standard deviation of 0.28, and macro F1-scores ranging from 0.00 up to 0.77. In JFrog and SecondLife, the number of training examples was less than 50 (44 and 14 respectively), explaining the 0.00 in both rows. When the performance was high, there were at least 1000 training examples present.

Block links had a moderate performance with a standard deviation of 0.10, and macro F1-scored from 0.41 up to 0.70. Even though three repositories use this link type often, it has an average of 4.8% share across all repositories. It seems the model only struggles to distinguish this link type from *Relate* but does not struggle to identify it from other link types, such as *Depend* or *Cause*, even though these relationships are more similar to *Block*. We also see that *Block* and *Depend* are used separately: either *Block* is used or *Depend*– except in Apache and JiraEcosystem. We see that performance of *Block* was good once the repository used this link type more than 10%.

Cause links have a very low average share and low performance, ranging from 0.26 up to 0.52. MariaDB had the best performance, but even there the model was unable to discern this link type from *Relate*. The reason for this might simply be that a casual relation is harder to detect while simultaneously not having enough example data.

Non-Link had an encouraging to top performance, up to 0.96 in Mojang, except for IntelDAOS. Overall the model did not struggle identifying links from non-links. The case of Mojang, which only has 3 link types to predict, is an indicator of how well the model can perform as a "linked issues" vs. "non-linked issues" detector.

Finding 2. The model was adept at distinguishing *non-links* from links. The structural links *Subtask* and *Epic* perform very well across all repositories. *Duplicate* does not perform well overall, but the model is able to distinguish *Clone* links from *Duplicate* links. *Depend*, *Incorporate*, and *Block* can be distinguished when they are frequently used. *Cause* had the lowest performance.
Relate seems to cause the most problems for the model and to be a jack-of-all-trades link type used to label a link when a stakeholder is unsure which type fits.

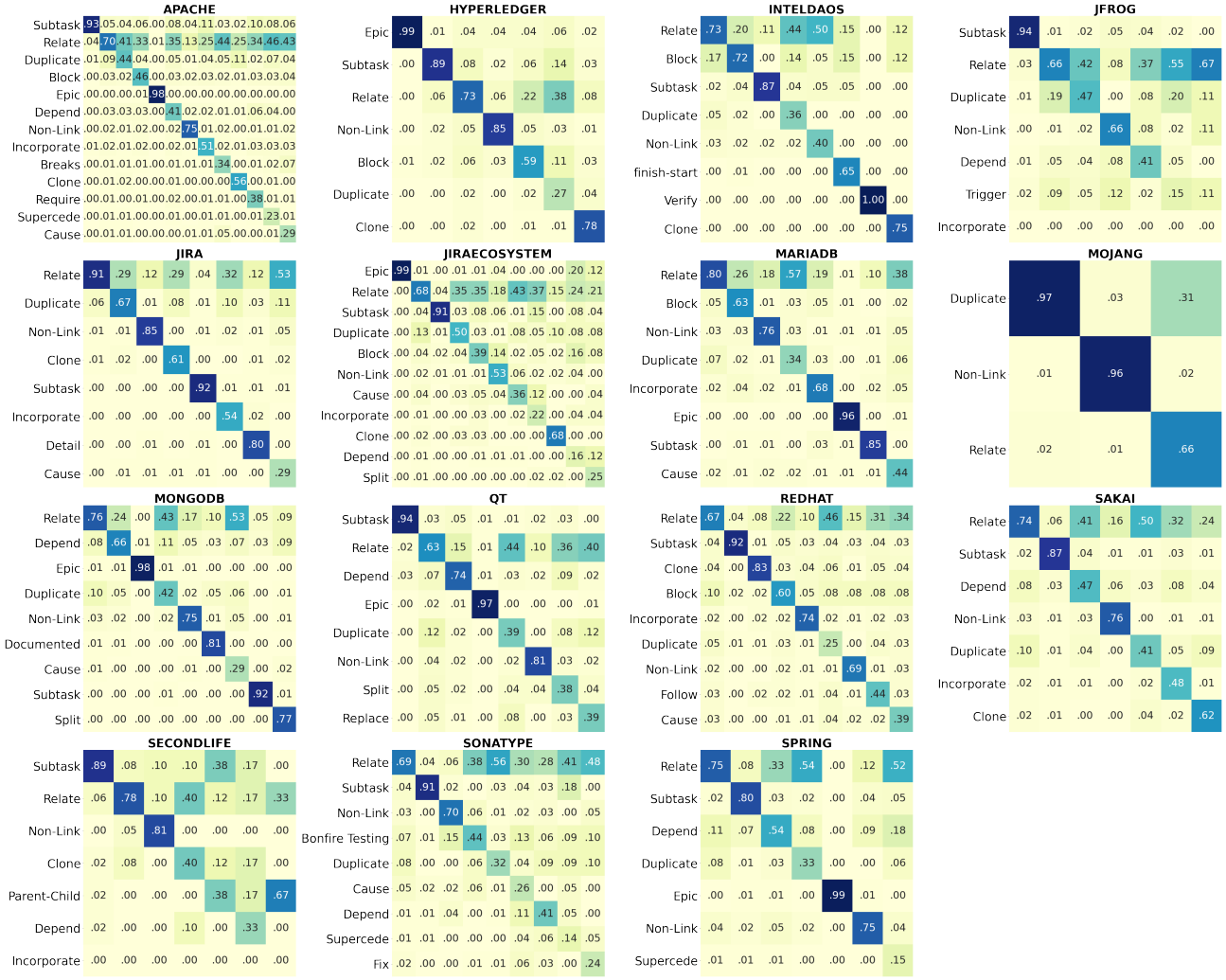

Fig. 2: Normalized confusion matrices for each repository. Rows are sorted by the support of link type, meaning the majority class is always the first row. The columns are in the same order as the rows.

IV. Analysis of Differences

In this section, we explore possible causes of the differences in the model performance across the repositories.

A. Repository Analysis

We calculated the Pearson correlation and p-value of different repository properties listed in Table I to find out what properties correlate with the macro F1-score of the model. We also correlated the number of unique issue creators, reporters, assignees, and total of unique users.

Table IV shows the correlation of the properties with the macro F1-score. The model's performance seems independent of the number of issues in the issue tracker. As expected, we observe a strong positive correlation with the coverage, an average positive correlations with the number of links, and a fairly strong negative correlation with the number of link types. A small negative correlation also exists for the number of projects. Only correlation with coverage is significant.

TABLE IV: Correlation and p-Value of macro F1-scores of the classifier to properties of the repositories

Properties	Correlation	p-Value
#Issues	0.0937	0.7397
#Links	0.3523	0.1978
#Projects	-0.2109	0.4505
#PredictedTypes	-0.5039	0.0554
%Coverage	**0.7500**	**0.0013**
%CrossProject	0.2439	0.3811
#Total Users	0.4419	0.0991
#Assignees	-0.1567	0.5771
#Creators	0.4374	0.1030
#Reporters	0.4435	0.0978
Assignee-Issue-Ratio	**0.5208**	**0.0465**

Moreover, we observe positive correlations with the number of total users, creators, and reporters, and a negative correlation with the number of assignees. Although none of them are significant, it is interesting that the number of assignees correlates negative with the performance. We then calculate the number of issues per assignee in a repository, which turns

to have a significant, fairly strong, positive correlation with the prediction performance. It is worth noting that, given the rather limited statistical power with the 15 studied repository, it is not surprising that most correlations are not significant.

Finding 3. Coverage and number of issues per assignee show strong and statistically significant correlations with macro F1-score to predict typed links. A higher coverage can indicate that stakeholders place more value on linking. The more issues an assignee is responsible for the more homogeneous a repository (and the linking) is likely to become.

B. Link Type Analysis

We calculated the TF-IDF vectors of the title and description of the involved issues and used cosine similarity to calculate the similarity of the issue pairs on a word basis. We also looked at the length of the textual descriptions of linked issues (word count of title and description of issue pairs) as well as the difference in length of two linked issues. We report the results for the common types in the dataset.

TABLE V: Median cosine similarity of the texts of linked issues on text-token-level across the JIRA repositories.

Repo	Rel.	Dup.	Sub.	Dep.	Clo.	Inc.	Epic	Blo.	Cau.	NL.
Apache	0.14	0.29	0.00	0.08	0.90	0.07	0.00	0.03	0.09	0.00
Hyperledger	0.33	0.44	0.21	/	0.95	/	0.06	0.31	/	0.00
IntelDAOS	0.18	0.29	0.07	/	0.99	/	/	0.11	/	0.00
JFrog	0.37	0.38	0.00	0.28	1.00	0.27	/	/	/	0.02
Jira	0.89	0.44	0.06	0.49	0.92	0.40	/	0.29	0.34	0.03
JiraEcosystem	0.25	0.48	0.00	0.15	0.94	0.13	0.00	0.13	0.24	0.00
MariaDB	0.25	0.32	0.06	/	/	0.13	0.00	0.22	0.13	0.04
Mojang	0.24	0.19	/	/	0.37	/	/	0.19	/	0.00
MongoDB	0.17	0.31	0.17	0.24	1.00	/	0.00	/	0.13	0.00
Qt	0.24	0.30	0.21	0.12	0.91	/	0.00	0.20	/	0.00
RedHat	0.21	0.36	0.17	/	1.00	0.00	/	0.10	0.16	0.00
Sakai	0.24	0.39	0.15	0.18	0.89	0.19	/	0.12	/	0.00
SecondLife	0.25	/	0.17	0.15	0.58	0.00	/	/	/	0.03
Sonatype	0.27	0.40	0.00	0.14	/	/	0.34	/	0.21	0.00
Spring	0.20	0.30	0.00	0.16	0.30	/	0.00	/	/	0.00
Mean	0.28	0.35	**0.09**	0.20	**0.83**	0.15	**0.05**	0.17	0.19	0.01

Table V shows the median of the cosine similarity of two issues that make up a link. We see that *Clone* is the most similar type, with an average of 0.83, which makes sense as these links are created by a feature in JIRA that clones the text with only small changes made by stakeholders. It might be interesting to investigate *Clone* links that have dissimilar issues, particularly if this was due to incremental changes over time. *Duplicate* links are the second most similar, with around 0.35 cosine similarity on average and a large gap to the similarity score of the *Clone* links. *Relate* links have an average of 0.28, the third highest. Jira's *Relate* links are used for pretty similar issues. It is also its most used link type and performed the best in the model compared to the other repositories. An investigation over link type changes in the Jira repository might explain this outlier.

Interestingly, *Epic* and *Subtask* links are very dissimilar. Considering that these are hierarchical relationships, one would assume that one of the issue's text is somewhat contained in the corresponding other issues, which would result in a higher similarity. This consistently low similarity together with the consistently high performance of these link types, indicates that the model is not only using textual similarities. Otherwise the performance of *non-links* would have been similarly high. Additionally, the model is able to clearly differentiate *Subtask* from *Epic*.

TABLE VI: Median length of texts of linked issue on text-token-level across the JIRA repositories.

Repo.	Rel.	Dupl.	Sub.	Dep.	Clo.	Inc.	Epic	Blo.	Cau.	NL
Apache	157	159	87	114	98	120	93	109	172	133
Hyperledger	176	159	79	/	92	/	96	131	/	105
IntelDAOS	268	398	90	/	123	/	/	297	/	142
JFrog	152	172	9	70	260	195	/	/	/	131
Jira	192	182	76	158	180	183	/	161	244	168
JiraEcosystem	119	104	36	105	108	93	51	88	142	88
MariaDB	433	432	85	/	/	188	70	341	439	350
Mojang	155	146	/	/	126	/	/	132	/	112
MongoDB	165	154	68	96	100	/	69	/	174	119
Qt	187	181	45	106	134	/	89	216	/	151
RedHat	135	136	74	/	112	92	/	97	153	110
Sakai	174	153	106	150	123	127	/	192	/	143
SecondLife	163	/	79	120	130	102	/	/	/	160
Sonatype	162	185	68	107	/	/	116	/	184	50
Spring	173	179	58	107	124	/	53	/	/	127
Mean	187	196	**69**	113	132	138	**80**	176	**215**	139

Finding 4. *Clone* and *Duplicate* pairs differ a lot in their similarity. *Clone* pairs are the most similar, followed by a wide gap and then by *Duplicate*. Structural links *Epic* and *Subtask* tend to connect very dissimilar issues. As they have the highest prediction performance, we can conclude that the model is not simply learning the text-similarity of issues.

Table VI presents the median issue text length per repository, while Table VII shows the median text length differences per repository. The Table shows the corresponding values for the studied link types. We observe that *Epic* and *Subtask* tend to connect shorter issues than other link types, while issues connected via *Cause*, *Relate*, *Duplicate*, and *Block* are longer. Moreover, inline with the similarity results, we can observe that two issues linked by a *Clone* link are very similar in length, while other link types tend to have a higher difference in the word count.

The link type *Cause*, performing worst, bears the highest difference in text lengths and the longest texts. One possible explanation is that corresponding descriptions might contains much information. The BERT model cuts of the text after a number of words. It likely doesn't see the full text of all links, particular information of long stack traces might get lost.

Table VIII shows the correlation of the analyzed link types' properties with the macro F1-score. We observe that the only significant correlation is the text length. We also see cosine similarity has a small correlation and is not significant, confirming that the model is not a simple similarity comparison. Additionally, we observe that links that are mistaken by the model as *Relate* links often connect lengthier issues than issues of correctly classified links as well as issues of other

TABLE VII: Median difference of text length of two linked issues on text-token-level across the JIRA repositories.

Repo.	Rel.	Dupl.	Sub.	Dep.	Clo.	Inc.	Epic	Blo.	Cau.	NL
Apache	50	43	34	37	3	41	44	36	62	52
Hyperledger	58	50	35	/	2	/	48	46	/	46
IntelDAOS	91	146	30	/	2	/	/	147	/	59
JFrog	44	49	5	18	2	61	/	/	/	60
Jira	0	43	33	39	8	49	/	43	68	54
JiraEcosystem	32	24	12	36	4	31	28	24	51	36
MariaDB	144	124	37	/	/	98	38	122	166	162
Mojang	42	49	/	/	24	/	/	41	/	35
MongoDB	58	47	22	36	1	/	33	/	63	49
Qt	56	51	16	44	9	/	40	143	/	55
RedHat	42	37	27	/	0	44	/	35	52	46
Sakai	52	38	36	48	3	41	/	18	/	48
SecondLife	51	/	31	25	17	70	/	/	/	54
Sonatype	45	42	30	46	/	/	53	/	64	23
Spring	52	50	23	38	44	/	23	/	/	53
Mean	54	57	26	37	**9**	54	38	66	**75**	55

TABLE VIII: Correlation and p-Value of macro F1-scores of the classifier to properties of the link types.

Properties	**Correlation**	p**-Value**
#Counts in Repos	0.2587	0.4704
#Difference	-0.4999	0.1412
#Length	**-0.6994**	**0.0244**
#Cosine Similarity	-0.1989	0.5817

mislabeled links (aside from *Relate*).

Finding 5. Text length correlates negatively (-0.70) with model performance. *Epic* and *Subtask* links have the shortest texts, while *Cause*, *Relate*, and *Duplicate* have longer texts. Links mislabeled as *Relate* links tend to connect lengthier issues than correctly classified links and other mislabeled links.

V. Discussion

We summarize our findings, possible explanations, and their applicability in practice. We then discuss the importance of issue and link quality, and present further research lines. Finally, we outline possible threats to validity.

A. Applicability of Typed Link Prediction in Practice

While much previous work has studied duplication links (details in Section VI), this work is among the first to study reliable detection of general issue links of different types (i.e. typed links). The straight use case is to recommend to stakeholders missing issue links or highlight incorrect link types. Our state of the art BERT model achieved an average macro F1-score of 0.64 and a weighted F1-score of 0.73 across the 15 studied repositories. These are quite promising results considering that the median amount of predicted classes is 7 (as opposed to e.g. the simpler binary classification). Our model is fairly general as it end-to-end (without feature engineering) and as it only uses the title and description of the issues as input features. The model works with mostly untouched data, only with as little basic preprocessing as possible. Stakeholder with more knowledge about the repository might fine-tune features based on their workflow. Our model can further be optimized depending on the individual context.

The model was quite precise at predicting hierarchical links *Epic* and *Subtask* and showed a similarly high accuracy for non-links. However, the model was not as good at identifying *Duplicate* links. Counter-intuitively, it did not struggle to distinguish *Clone* from *Duplicate* links. This is explained by the way *Clone* links are created. JIRA offers a feature to stakeholders to clone an issue. They create a new issue and start with all the properties of the cloned issue as default. This explains the high cosine similarity of two issues linked by clone. In contrast, issues of a *Duplicate* link are usually created by two different contributors, who are often unaware of the other issue. As *Clone* links are the only ones created automatically, a link prediction tool could also ignore them.

Other link types, *Depend*, *Incorporate*, and *Block* need a critical mass to achieve good results. The link type *Cause* has the least accuracy, which might be due to the fact that it is barely used or its length. We truncated the texts, meaning that longer texts were cut, this affected link types which tend to be longer. One reason for long texts, are non-natural language fragments, such as stack traces or code. Non-natural language is also hard for humans to understand. It is thus possible that stakeholders incorrectly link these issues with the *Relate* link type. While manually reviewing the data, we observed that *Relate* links often contain stack traces or other pieces of non-natural language.

There are three strategies which will likely improve the prediction performance and further increase the applicability in practice. First, a tool could first detect the existence of a link and then its type with two different models. The link detection model is trained on the dataset where all linked issues are one class and randomly created *non-links* the other class. The type detection model is trained on all links except the *Relate* links. This alleviates the ensuing prediction problems of likely bad data quality for the *Relate* link type.

Second, a tool might predict categories of link types instead of a specific type. With a median of 10.5 link types per repository, it is likely that some link types are semantically related or partly redundant. Stakeholders in a project are better aware of which link types are similar. As our model is general to various types, no additional changes are needed, except grouping the labels from certain types to a category. Then, a stakeholder with knowledge about the project can choose the correct link type from the predicted category.

The third strategy for a better applicability is a top k prediction, presenting the stakeholder with multiple possible hits. With $k = 3$, we calculated that the average prediction accuracy of our results would increase by almost 18%.

B. Data Quality as Perquisite for Correct Link Prediction

Issue quality, in our case, text quality (missing information, ambiguity), directly impacts prediction quality as we only use the textual descriptions in linked issues. E.g.: The issues BE-213[8] and BE-94[9] in Hyperledger are linked as *Clone* and

[8]https://jira.hyperledger.org/browse/BE-213
[9]https://jira.hyperledger.org/browse/BE-94

mislabeled as *Epic* by our model. Both of them only have a title ("Footer Components" and "Header Components") but only an empty description.

Issue quality might be affected by the stakeholder creating the issue or the link (user, developer, analyst, product manager...). Zimmerman et al. [38] found that most issues reported by users miss important information for the developers, such as steps to reproduce or stack traces. As users are often unaware of what makes a good issue reports, they are likely to create lower quality issues. In contrast, analysts or developers are more aware that certain details are important to implement a requirement, fix a bug, or structure and plan a release.

The link types *Subtask* and *Epic* can be distinguished quite well with the text alone, likely due to the unique way they are treated in JIRA. The main differences to other link types is that they have their own sections. Thus, stakeholders might treat them more carefully. Furthermore, *Epic* and *Subtask* are often used to structure the issue tracking system. It is likely that *Epic* and *Subtask* links are created with intent and care, as they are usually cleared by stakeholders deeply involved with the repository with a planning role. The link are likely created at issue creation time in an analysis and planing task. This all might improve the quality of issue text as well as correctness of the links.

The quality of the issues and the linking (i.e. the correctness of the link and its type) is only as good as the carefulness by the people who create them. This is apparent in the seemingly rather average/low quality of *Relate* links, which is the most popular link type and most confused by the model at the same time. We noticed, that other link types are often mislabeled as *Relate* links. One explanation might be that the typed link prediction might suffer from low quality of *Relate* links. Due to its general nature, it is probable that *Relate* links are misused by stakeholders when they are uncertain whether other specific types are correct or more fitting. Thus, *Relate* might contain a lot of data points that should have another label. For instance, the issue ZOOKEEPER-3920[10] in Apache has a *Relate* link to ZOOKEEPER-3466 and ZOOKEEPER-3828, while the comments discuss that this should be a *Duplicate*.

C. Further Implications for Research

We found attributes of JIRA issue trackers that correlate with the performance of the prediction, the coverage and issue-to-assignee ratio, as well as the text length of a link. Two important factors, which require more extensive research are the link quality and the heterogeneity of a repository.

Coverage, the share of issues that are part of a link, significantly correlates positively with the performance of the model and could be an indicator of higher data quality, making it easier for any model to learn.. A higher coverage implies that stakeholders try to link as many issues as possible or place a higher value on links as tools to manage their knowledge.

The heterogeneity of an issue tracker is an interesting factor in typed link prediction. The issue-to-assignee ratio also correlates positively with the model performance, the more issues are assigned to one person, the better the model. This makes sense as the less people involved, the less heterogeneity is in the repository. We did not find any further significant correlation with the number of the creators, reporters, or overall users.

Another indicator for heterogeneity is the number of projects in a repositories, up to 646 in the case of Apache, as most links usually do not cross boundaries We did not find a significant correlation between the number of cross-project links or number of projects and the performance of the model on a repository. However, it might make sense to evaluate the prediction model per project or cluster of projects instead of per repository.

An interesting aspect of link and typed link prediction is identifying orphans, loners, or phantoms. As coverage seems to be a factor for good typed link prediction, issues without any links to other issues or commits should be investigated. Schermann et al. [30] already examined and created a heuristic model to deal with links between issues and commits.

D. Threats to Validity

The data used for training and testing is largely representative of the way stakeholders use links and their types in practice. The only restriction we placed was the 1% share lower bound, as it is very conservative it should not introduce any bias that overestimates the quality of the model. Practitioners with knowledge of the underlying processes and context can group their link types as they see fit, the same model can still be applied.

As the labels, the link types, are made by humans, they can contain false positives. We discuss likely quality issues and differences between the repositories that influence the quality of the model's prediction. We also evaluated the approach on 15 different repositories and thus the approach is generalizable for repositories that use JIRA. We did not evaluate the approach for Bugzilla and GitHub, which use fewer link types, thus the approach might produce different results for repositories that use other issue trackers.

We added randomly created *non-links* to the dataset, the number of non-links can be varied, we chose the mean of the number of other link types present to not create a majority class. With a higher amount of *non-links* performance can change.

Another threat is the evolution and changes over time in issue trackers. If a repository has been in use for a long time, the implicit definition of links types can have changed. Issues themselves change over time and the links might not be updated accordingly and certain link types might fall out of favor over time. This work did not take the time axis into consideration and results may change.

VI. Related Work

With the rise of agile, requirements are often collected and tracked in issue trackers. This rise also led to a generation of large amounts of data in issue trackers, which are often

[10]https://issues.apache.org/jira/browse/ZOOKEEPER-3920

too much to handle manually. In a case study, Fucci et al. [10] interviewed JIRA users and found that information overload is one of their biggest challenges. Interviewees of the study expressed the need for a requirements dependency identification functionality to reduce the overhead of discovering and documenting dependencies manually. This paper provides a solid model that tackles this requirements dependency identification functionality. Franch et al. [9] tackle the ensuing problems of agile practices in requirements elicitation and management with situational method engineering. The collection of requirements interdependencies (i.e. issue links) also face similar challenges.

Requirements Engineering research has largely studied another kind of dependency between issues/requirements and software artifacts: a topic known as traceability. Deep learning has also been used for traceability. Lin et al. [17] found that the traceability links between issue description to source code can be found with BERT which outperforms the traditional information retrieval methods, achieving an F1-score of 0.612 and 0.729. Typed link prediction has similar application problems as traceability, such as poor quality in issue trackers, found by Merten et al. [22]. Additionally, Seiler et al. [31] conducted an interview study about the problems of feature requests in issue trackers and found that unclear feature descriptions and insufficient traceability are among the major problems in practice. For typed link prediction we also found that heterogeneity of the repository is a problem.

Concerning issue link prediction, *Duplicate* is the most widely researched type, as duplication detection is a tedious task [1] when curating issue trackers. Deshmukh et al. [4] proposed a single-channel siamese network approach with triplet loss which achieves an accuracy close to 90% and a recall rate close to 80%. He et al. [11] proposed a dual-channel approach and achieved an accuracy of up to 97%. Rocha et al. [29] created a model using all "Duplicate" issues as different descriptions of the same issue and report a Recall@25 of 85% for retrieval and an 84% AUROC for classification. All three works [4], [11], [29] use the data set provided by Lazar et al. [15], containing data mined from the four open-source Bugzilla systems: Eclipse, Mozilla, NetBeans, and OpenOffice.

There also exist studies researching other link types between issues and link usage. Thompson et al. [33] studied three open-source systems and analyzed how software developers use work breakdown relationships between issues in JIRA. They observed little consistency in the types and naming of the supported relationships. Li et al. [16] examined the issue linking practices in GitHub and extracted emerging linking patterns. They cateogorized link types into 6 link type categories, namely: "Dependent", "Duplicate", "Relevant", "Referenced", "Fixed", "Enhanced", all the rarer link types were assigned the category "Other". They discovered patterns for automatic classification; "Referenced" links usually refer to historic comments with important knowledge and that "Duplicate" links are usually marked within the day. Tomova et al. [34] studied seven open-source systems and reported that the rationale behind the choice of a specific link type is not always obvious. The authors also found that *Clone* links are indicative of textual similarity, issues linked through a *Relate* link presented varying degrees of textual similarity and thus require further contextual information to be accurately identified. From the varying degrees of textual similarity we hypothesized that *Relate* links might be a jack-of-all-trades type. Furthermore, while we found that some link types have distinct textual similarities, they are not unequivocally identifiable only based on the textual similarity.

"Requires" and "Refines" links were examined by Deshpande et al. [5], who extracted dependencies on two industrial data sets achieving an F1-score of at least 75% in both training sets. *Block* links were examined by Cheng et al. [2] on the projects mined by Lazar et al. [15]. They predicted the *Block* link type with an F1-Score of 81% and AUC of 97.5%.

Most previous works view link types in isolation, our work extends these works as we look at all used link types and achieve promising results in distinguishing them.

Nicholson et al [25], [26] also researched typed link prediction between issues of projects in Apache. The first [25] analyzes the link types and tries to find patterns to predict missing links. The second [26] evaluates several traditional machine learning approaches and achieves a weighted F1-score of about 0.563 up to 0.692 across the three projects HIVE, FLEX, and AMBARI.

Data quality, in this case, requirements and link quality are essential for well-performing tools and smooth-running workflows. We found that data quality affects typed link prediction as well. Montgomery et al [24] recently published a systematic mapping study evaluating the research on requirement quality. They found that most research focuses on improvement techniques rather than finding evidence-based definitions or evaluation of quality attributes. Dalpiaz et al. [3] created an approach to improve the quality of user stories, a type of issue, by removing linguistic defects. Link quality is directly affected by requirements quality. If issue descriptions are vague or contain other defects, links are harder to classify.

VII. Conclusion

Using BERT on the titles and descriptions of issue pairs, we achieved good performances to predict typed issue links on most studied repositories; and consistently excellent performance for predicting *Epic* and *Subtask* links. Our detailed analysis revealed that, by better understanding the data and improving the issue quality (for the training and prediction) and the link quality (for the training) the performance can be improved further – particularly the general purpose *Relate* links. We discussed strategies for increasing the performance and thus the model's applicability, particularly where data quality is limited or heterogeneity in the issue tracker is high. Future work should focus on the underlying data as well as repository-, project-, and stakeholder-specific factors that impact the performance. For this user studies and qualitative research is needed to understand how and why stakeholders use the links and link types.

REFERENCES

[1] Y. C. Cavalcanti, E. S. d. Almeida, C. E. A. d. Cunha, D. Lucrédio, and S. R. d. L. Meira. An initial study on the bug report duplication problem. In *2010 14th European Conference on Software Maintenance and Reengineering*, pages 264–267, 2010.

[2] X. Cheng, N. Liu, L. Guo, Z. Xu, and T. Zhang. Blocking bug prediction based on xgboost with enhanced features. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 902–911, July 2020.

[3] F. Dalpiaz and S. Brinkkemper. Agile requirements engineering: From user stories to software architectures. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 504–505, 2021.

[4] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, and N. Dubash. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 115–124, 2017.

[5] G. Deshpande, Q. Motger, C. Palomares, I. Kamra, K. Biesialska, X. Franch, G. Ruhe, and J. Ho. Requirements dependency extraction by integrating active learning with ontology-based retrieval. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 78–89, Aug 2020.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] S. Fang, Y.-s. Tan, T. Zhang, Z. Xu, and H. Liu. Effective prediction of bug-fixing priority via weighted graph convolutional networks. *IEEE Transactions on Reliability*, 70(2):563–574, 2021.

[8] C. Fitzgerald, E. Letier, and A. Finkelstein. Early failure prediction in feature request management systems. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 229–238, 2011.

[9] X. Franch, A. Henriksson, J. Ralyté, and J. Zdravkovic. Data-driven agile requirements elicitation through the lenses of situational method engineering. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 402–407, 2021.

[10] D. Fucci, C. Palomares, X. Franch, D. Costal, M. Raatikainen, M. Stettinger, Z. Kurtanovic, T. Kojo, L. Koenig, A. Falkner, G. Schenner, F. Brasca, T. Männistö, A. Felfernig, and W. Maalej. Needs and challenges for a platform to support large-scale requirements engineering: A multiple-case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[11] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei. Duplicate bug report detection using dual-channel convolutional neural networks. In *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, page 117–127, New York, NY, USA, 2020. Association for Computing Machinery.

[12] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 392–401, 2013.

[13] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy. Norbert: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 169–179, 2020.

[14] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. ESEC/FSE '09, page 111–120, New York, NY, USA, 2009. Association for Computing Machinery.

[15] A. Lazar, S. Ritchey, and B. Sharif. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories*, pages 392–395, 2014.

[16] L. Li, Z. Ren, X. Li, W. Zou, and H. Jiang. How are issue units linked? empirical study on the linking behavior in github. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 386–395, 2018.

[17] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang. Traceability transformed: Generating more accurate links with pre-trained bert models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 324–335, 2021.

[18] C. L. Lloyd Montgomery and W. Maalej. Jira: An alternative issue tracking dataset, 2022. preprint on https://arxiv.org/abs/2201.08368.

[19] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, S. Brinkkemper, and D. Zowghi. Behavior-driven requirements traceability via automated acceptance tests. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 431–434, 2017.

[20] R. Malhotra, A. Dabas, H. A. S, and M. Pant. A study on machine learning applied to software bug priority prediction. In *2021 11th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pages 965–970, 2021.

[21] T. Merten, M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner, and B. Paech. Software feature request detection in issue tracking systems. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 166–175, 2016.

[22] T. Merten, D. Krämer, B. Mager, P. Schell, S. Bürsner, and B. Paech. Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 45–62. Springer, 2016.

[23] L. Montgomery and D. Damian. What do support analysts know about their customers? on the study and prediction of support ticket escalations in large software organizations. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 362–371, 2017.

[24] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz, and W. Maalej. Empirical research on requirements quality: a systematic mapping study. *Requirements Engineering*, 2022.

[25] A. Nicholson, D. M. Arya, and J. L. Guo. Traceability network analysis: A case study of links in issue tracking systems. In *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 39–47, 2020.

[26] A. Nicholson and G. Jin L.C. Issue link label recovery and prediction for open source software. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 126–135, 2021.

[27] Q. Perez, P.-A. Jean, C. Urtado, and S. Vauttier. Bug or not bug? that is the question. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 47–58, 2021.

[28] P. Rempel and P. Mäder. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, 43(8):777–797, 2017.

[29] T. M. Rocha and A. L. D. C. Carvalho. Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information. *IEEE Access*, 9:44610–44630, 2021.

[30] G. Schermann, M. Brandtner, S. Panichella, P. Leitner, and H. Gall. Discovering loners and phantoms in commit and issue data. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 4–14, 2015.

[31] M. Seiler and B. Paech. Using tags to support feature management across issue tracking systems and version control systems. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 174–180. Springer, 2017.

[32] C. Stanik, L. Montgomery, D. Martens, D. Fucci, and W. Maalej. A simple nlp-based approach to support onboarding and retention in open source communities. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 172–182, 2018.

[33] C. A. Thompson, G. C. Murphy, M. Palyart, and M. Gašparic. How software developers use work breakdown relationships in issue repositories. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 281–285. IEEE, 2016.

[34] M. T. Tomova, M. Rath, and P. Mäder. Use of trace link types in issue tracking systems. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ICSE '18, page 181–182, New York, NY, USA, 2018. Association for Computing Machinery.

[35] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, page 461–470, New York, NY, USA, 2008. Association for Computing Machinery.

[36] J. P. Winkler, J. Grönberg, and A. Vogelsang. Optimizing for recall in automatic requirements classification: An empirical study. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 40–50, 2019.

[37] G. Xiao, X. Du, Y. Sui, and T. Yue. Hindbr: Heterogeneous information network based duplicate bug report prediction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 195–206, Oct 2020.

[38] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36(5):618–643, 2010.