Aide-mémoire: Improving a Project's Collective Memory via Pull Request–Issue Links

PROFIR-PETRU PÂRȚACHI, Department of Computer Science, University College London, United Kingdom
DAVID R. WHITE, Department of Computer Science, University of Sheffield, United Kingdom
EARL T. BARR, Department of Computer Science, University College London, United Kingdom

Links between pull-requests and the issues they address document and accelerate the development of a software project, but are often omitted. We present a new tool, Aide-mémoire, to suggest such links when a developer submits a pull-request or closes an issue, smoothly integrating into existing workflows. In contrast to previous state of the art approaches that repair related commit histories, Aide-mémoire is designed for continuous, real-time and long-term use, employing Mondrian Forests to adapt over a project's lifetime and continuously improve traceability. Aide-mémoire is tailored for two specific instances of the general traceability problem — namely, commit to issue and pull-request (PR) to issue links, with a focus on the latter — and exploits data inherent to these two problems to outperform tools for general purpose link recovery. Our approach is online, language-agnostic, and scalable. We evaluate over a corpus of 213 projects and six programming languages, achieving a mean average precision of 0.95. Adopting Aide-mémoire is both efficient and effective: a programmer need only evaluate a single suggested link 94% of the time, and 16% of all discovered links were originally missed by developers.

CCS Concepts: • **Software and its engineering** → *Requirements analysis*; **Software evolution**; **Software version control**; **Maintaining software**.

Additional Key Words and Phrases: Traceability, Link Inference, Missing Link

1 INTRODUCTION

Traceability is the maintenance of relationships between software development artefacts; the most important of these relationships is the link between requirements and their implementation. In the "move fast and break things" era, the addition and maintenance of links are too often neglected. Good traceability practices and tooling can improve all aspects of software development, from requirements elicitation to code maintenance. As such, traceability is a seminal software engineering concern.

In modern development, issues track outstanding work, both reported bugs and feature requests. A Pull-Request (PR) is a sequence of patches submitted for reviewing and merging into a project's mainline, as illustrated in Figure 1. Developers work with issues and PRs, day to day; they are interlinked in a developer's mind. When these traceability links are recorded, they accelerate software development because developers can use them to restore context [28, 50]. They keep teams informed of progress on feature enhancement and prevent commit reversion and issue reopening by connecting the commits within a PR with the issues they address [31, 49]. Developers use them to prioritise work; reviewers use them to learn the context of an issue. They facilitate fault prediction [57], bug localisation [43, 58, 61], and issue triage.

Authors' addresses: Profir-Petru Pârțachi, Department of Computer Science, University College London, London, United Kingdom, profir-petru.partachi.16@ucl.ac.uk; David R. White, Department of Computer Science, University of Sheffield, Sheffield, United Kingdom, d.r.white@sheffield.ac.uk; Earl T. Barr, Department of Computer Science, University College London, London, United Kingdom, e.barr@ucl.ac.uk.

1049-331X/2022/6-ART $15.00
https://doi.org/10.1145/3542937

Despite their importance, these links, like other traceability links, are often not recorded or maintained. Although modern tools, like JIRA and GitHub, provide increased support for linking, developers do not record most links [5]. We confirm this trend in a large-scale analysis of repositories: over half (54%) of PRs are not linked to an issue when submitted, despite the fact that a third of project contribution guidelines recommend linking (Section 7.1). During PR review, missing links are sometimes discovered manually and the PR amended. Around 16% of PRs are linked during this process, leaving 38% unlinked.

We present *Aide-mémoire* (A-M), a tool that suggests PR-issue links to a developer. We call it Aide-mémoire because it aims to help a project partially retain its "collective memory". A-M is *online*: it suggests a link when a developer submits a PR or closes an issue. This is critical for uptake. Offering suggestions to developers when they need not context-switch is critical. In his ICSE 2020 Keynote, Peter O'Hearn remarked that an analysis run as a batched process had 0% developer uptake, which jumped to 70% when the same suggestion is made during code-review [37]. A-M does not require invasive instrumentation, but relies instead on the content that developers already produce: commit logs, PRs, and posts on discussion boards. It needs only tokenise its inputs, so it is language-agnostic and this, coupled with the fact that it builds its online classification model incrementally, allows it to scale to large code bases. To train A-M, we mine GitHub projects and their issue trackers. A hurdle all tools that aim to improve developer workflows face is Agile's "lightweight" requirement [34]: tool adoption and use must quickly pay for itself. A-M's principal goal is to meet this demanding requirement by making the cost of creating and maintaining PR-Issue links easy and seamless. Time will tell how well AM meets this deployability challenge (Section 3.2).

While leaving feature selection to neural architectures is common today, it incurs a higher training cost or requires additional training data. In machine learning approaches other than deep learning, feature selection is a critical step that can make or break a model. With the advent of neural networks in SE, a scan of recent papers will reveal that feature selection has been neglected. We turn to feature selection to speed both A-M's training and, as it is online, its adaptation to new linking regimes. Section 4 carefully details this process and can serve as a primer on feature selection for other software engineers who may find it useful in their work.

When this was written, the state of the art commit-issue prediction tool was RCLinker [23]. RCLinker is offline and handles commits, not pull-requests, and is limited to Java projects, since it requires ChangeScribe [24] to generate natural language descriptions of changesets to Java code. To validate A-M against RCLinker, we therefore had to adapt it (Section 6). We call our variant RCRep; to handle PRs, it replaces ChangeScribe summaries with user-provided PR descriptions, the first post in a PR, which is semi-structured by convention. RCRep suggests a pull-request when its internal RCLinker predictor suggests any commit within that pull-request. Because offline is a degenerate case of online, we run A-M offline to compare it with RCRep; we take care to ensure neither tool sees events from the future. We show that A-M is more precise than RCRep, achieving a mean Precision of 0.76 and F1-Score of 0.46 compared to RCRep's 0.14 and 0.15, with a similar Recall (0.37 vs 0.36) across 47 Java projects (Section 6.2).

Having established A-M's performance against a baseline, we evaluate it in its native setting: online over a multi-lingual corpus of 213 projects, which contains a range of project sizes and programming languages (Section 5.1). We train A-M on project history prefixes of fixed length relative to project size and validate it on the suffix by replaying repository events. It achieves high accuracy: a Mean Average Precision (MAP) of 0.95 (Section 7.2). We also show that A-M generalises well: there is no statistically significant difference with project size or across languages in performance. A-M maintains performance on projects with over a thousand open issues and hundreds of monthly pull-requests; Section 7.2 shows that there is no statistical correlation between any of the performance metrics and project size.

We designed and built A-M to seamlessly integrate with existing developer workflows; A-M must augment, not disrupt them (Section 3.2). Therefore, A-M must be highly precise to avoid distracting developers with useless suggestions they discard. Our finding that A-M achieves 0.95 MAP suggests that it meets this requirement.

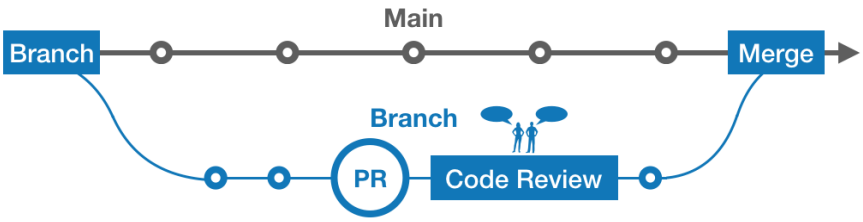


Fig. 1. Simplified pull-request process. Each small circle represents a commit. A developer opens a new branch, makes code changes and submits a pull-request to code review. Further changes may be made on the branch before being accepted and merged.

An offline approach, like RCLinker, must be periodically retrained, while an online model, like A-M, can learn as the project evolves. Thus, A-M does not require a dedicated maintenance task, substantially enhancing its deployability. Finally, installing A-M only requires a lightweight backend that can be installed locally or onto a server for sharing, and a Chrome plugin.

Our main contributions follow:

- We present the design and implementation of A-M, a tool that solves the PR-issue link inference problem via online classification, providing pertinent suggestions;
- We evaluate A-M on a large and diverse corpus, and demonstrate that our approach generalises across languages and scales to large projects containing over a thousand open issues and hundreds of PRs per month;
- We show that A-M can exploit information in PRs to outperform related work that solves the traditional offline *commit-issue* linking problem when applied to PRs.

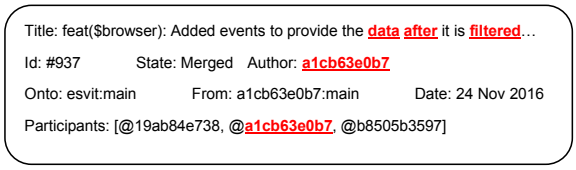
All tools, data and scripts needed to reproduce our work are available at https://github.com/PPPI/a-m.

2 MOTIVATING EXAMPLE

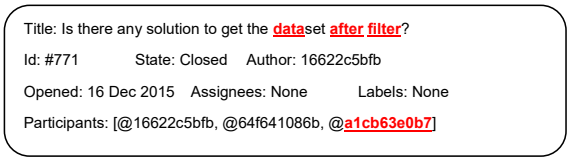
Figure 1 overviews a typical modern development process: a PR consisting of a set of changes to resolve an issue is submitted for code review. If the link between the PR and issue is not recorded, the issue remains open and the record of why a PR was made is lost.

Literature suggests that certain features, such as textual overlap, participants, files touched, or even time between events, can be useful predictors of artefacts being related to each other [23, 36]. While some of these features' usefulness makes intuitive sense, before delving into the details of Aide-mémoire, we first look at a pseudo-anonymised example of a PR-issue pair from GitHub to assess how the features suggested by literature could aid prediction.

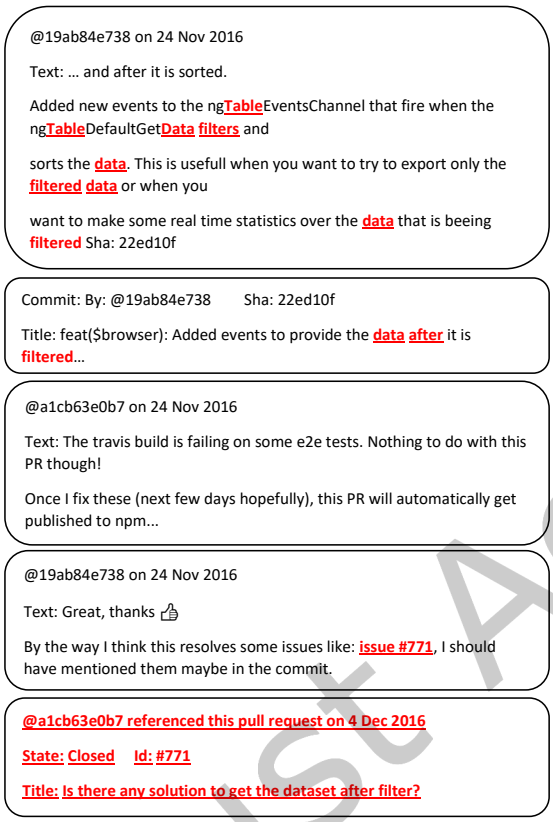
Figure 2 gives an example of an unlinked PR and corresponding issue, where the issue was open but initially missed by the PR submitter. The example is taken from the ng–table project, a table library for AngularJS. On the left of the figure is a pull-request containing code changes that makes it possible for a developer to access the original data of a table after it has been filtered or sorted; this PR addressed the issue in the right of the figure, but was not linked at the time the PR was submitted. Both titles and conversations discuss filtering and mention common terms such as the identifier fragment 'getData'. This causes a high textual similarity between the titles as can be seen in Figure 2a and Figure 2b. Moreover, there is significant textual similarity between the PR description, the first posting in Figure 2c, and the first three postings in Figure 2d. The particular sub-tokens that overlap are highlighted in the title and postings. The PR and issue also share a common participant: namely, the PR submitter (identifier @a1cb63e0b7). The submitter only references the PR and closes the issue, 10 days after



(a) pull-request Metadata.



(b) Issue Metadata.



(c) pull-request Discussion.

(d) Issue Discussion (trimmed for typesetting reasons.)

Fig. 2. Example of a pull-request and a related issue. The PR was not linked to the issue when submitted, but the missing link was subsequently added manually by the submitter after 10 days. The title and discussion share common terminology as well as common participants that can be exploited by Aide-mémoire. Some of the more useful ones are highlighted in red along with the reference message that was created upon closing the issue. Developer names have been pseudo-anonymised.

the PR was submitted. By exploiting these and other features, Aide-mémoire assists developers by suggesting links at PR submission and issue closure and can reduce such delays.

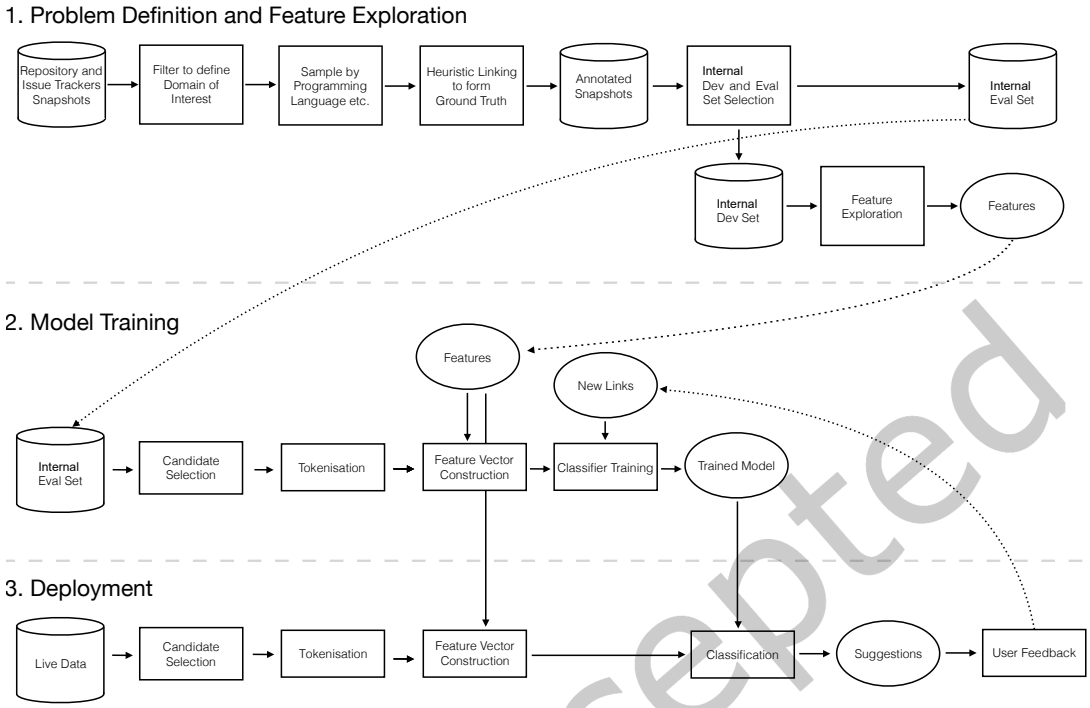


Fig. 3. Methodology used for instantiating Aide-mémoire. In the first stage, a datastore of repositories is identified, filtered and sampled to define the domain of interest. Heuristic linking generates an initial knowledge to learn and evaluate against. A subset of selected repositories is used to identify powerful features. In Stage 2, the annotated repositories are used to learn a predictive model based on the selected features. In Stage 3, live data from the repositories is used to generate feature vectors and suggest candidate links to a developer using the trained model. Feature exploration is not required for the implementation, in which case the Annotated Snapshots will be used directly in Stage 2 and all features will be considered.

3 AIDE-MÉMOIRE

Aide-mémoire is the first online PR-issue linking tool. Figure 3 overviews the methodology and design of Aide-mémoire. We start by obtaining a list of observed links already recorded within the project issue tracker. We then determine the feature space for A-M, which we detail in Section 4. This process is performed once for our evaluation, but may be performed on a per-project basis when deployed. Next, we use the features to learn a model over a set of repository snapshots. We subsequently deploy the learnt model to suggest links when developers submit PRs or close issues.

In solving an online linking problem, we must narrow the set of candidate links we consider to ensure that our system remains responsive. When suggesting issues at PR submission we limit ourselves to *open* issues; one of the main motivations for linking PRs and issues is to ensure the automatic closure of an issue if a PR is merged. For the symmetric problem of suggesting PRs to be linked to a given issue, we use a seven-day window of recently submitted PRs, in line with previous work [36].

3.1 Model Learning

We train a statistical classifier on PR-issue pairs to learn a probability distribution over possible links. For each candidate PR-issue link, we calculate the feature vector as detailed in Section 4 and train the classifier to learn

the probability that the link should exist. In prediction mode, we sort links by this probability when presenting suggestions to a developer. As most candidate pairs represent false links, our data is class-imbalanced; however, we observe empirically on our development set that Mondrian Forests perform well despite this imbalance and hence we do not employ undersampling.

We deviate from the more classical Random Forest used by the state-of-the-art offline tool [23], and instead employ the online method of Mondrian Forests [22]. Mondrian Forests represent a class of Random Forests that employ the Mondrian Process to partition the feature space. This process can be interpreted as a stochastic kd-tree. We hypothesise that their resilience to class imbalance arises from their ability to infer tight bounding boxes around positive examples; further investigation which lies outside the scope of this paper is needed to validate this claim.

Mondrian Forests work well in the online case; training them with sequential examples is equivalent to batch training in the limit [22]. Table 9 in Section 7.5 shows that simply retraining a Random Forest periodically is insufficient in our setting. We speculate that the kd-tree/bounding box nature of Mondrian Forests are a better prior in our setting enabling it to work even on more data-starved projects. Additionally, since probability estimates are obtained by majority voting, this enables us to use the model to obtain finer-grained probability estimates. Once initially trained, we deploy the classifier to provide suggestions to the developers and learn online as further links are created.

We add special 'no_pr' and 'no_issue' entities that represent the absence of any link, with their structures populated with empty strings and null timestamps. These special cases allow us to explicitly learn when no link should be proposed. We truncate suggestion lists at the index of these special entities, using them as a 'tidemark', excluding predictions that are less likely than a link to an empty issue/pr. Learning when these entities apply relies on our use of features that depend on only one side of the link.

Explicitly recording the absence of a link requires the learner to solve two problems at the same time: 'should there be any link?' and 'what should the artefact be linked to?'. Previous work only focused on the latter, suggesting no link only when no suggestion could be made above some threshold, which itself was learnt post-hoc. Our solution allows the model to learn a per-suggestion threshold, providing a natural cut-off point when prompting a developer with a list of suggested links.

3.2 Deployment

We built A-M to vault Agile's lightweight requirement [34] and with deployability in mind from the get go. A-M's use requires only installing a browser plugin and a backend server. Training requires only the URL of a GitHub repository. A-M can search for links to suggest in parallel to other development tasks, so it seamlessly integrates into existing workflows.

A-M's back-end learns, maintains, and stores the project's model; its Chrome plug-in front-end parses issue and PR pages. Although a developer team would benefit from sharing A-M's backend, individual developers can install both the server and the plugin locally. Developers interact with Aide-mémoire via its plug-in when viewing a PR or issue. The plugin suggests links when a developer closes an issue or submits or reviews a PR. A-M only makes high confidence suggestions (controlled by a user-specified threshold) and displays them in rank order. A-M makes no suggestions that are less likely than the special 'no_pr' and 'no_issue' entities, staying silent when its confidence is low.

To learn a model for their project, a developer can install A-M and generate a model locally. To start training, a developer need only enter their GitHub URL into A-M's command line. This instigates the crawling and processing of their repository. Alternatively, project managers can install a central backend. The initial training of our system over a large project such as Google's Guava, which contained 206 PRs and 2563 issues, and a total of 5392 commits when we crawled it, takes less than two hours on an Intel i7-6820HK@2.70 GHz.

Our prototype plug-in subsequently makes suggestions to a developer in, on average, less than 10s of the completion of their PR or issue closing message. The suggestion itself is fast (sub-second); most of the delay is parsing the issue or PR GitHub page and sending the extracted data to the backend. The expensive HTML wrangling and network roundtrip can be overlapped with, and hidden beyond, a developer's other activities. For instance, a developer need only click on the plug-in to start the process. While it's running in the background, they can continue work on their PR submission or issue triaging. Future engineering work could further mask this cost behind continuous integration, by including it as an additional pass in the build file.

To adapt A-M to platforms other than GitHub, one need only change the class that interacts with the GitHub API with one that interacts via the desired platform API. For example, to apply to JIRA, the GitHub API methods [39][1] should be changed to the equivalent JIRA ones [18] in our tool source-code. Afterwards, as the data is converted into a common format, the tool will work as is.

Although deployability is a key design goal of A-M, it has not yet been put to the test. Tackling an industrial problem like PR-Issue linking presents a researcher with a chicken and egg problem: either first they convince an industrial partner to risk working their unproven solution with them, or they first realise their solution, then try to convince an industrial partner to try, and hopefully adopt, it. We chose the latter. This paper and its associated GitHub repository are this work's first chance to reach practitioners.

Full source code, deployment tools, and code and scripts required to recreate our evaluation can be found online [39].

4 EXPLORING THE FEATURE SPACE OF ISSUE-PR LINKS

In this, the neural era, exacerbated by the recent advent of large language models such as GPT-3 [7], feature exploration is unfashionable; the networks are left to sort out features on their own at the cost of training time and data. Most projects are small and all start small, and usually lack sufficient labelled data to train a neural network. Our focus on feature engineering is in the spirit of focusing on good data rather than big data [51]. A-M is not neural because we wanted it to apply to small projects. So, we now describe the traditional feature engineering [21] that underlies A-M. Despite being, or perhaps because it is, out of fashion, software engineering researchers who use ML may find this section useful as a primer. Even neural networks benefit when feature engineering helps architect the network, narrows wide data to the width of a network's input layer, or boosts signal, and hence reduces the amount of training data and training time needed. We first enumerate features drawn from the literature and augment with new features we devised. This collection is comprehensive to the best of our knowledge. Using the full feature set is costly, both in development time and, crucially, when deployed. So, in closing, we show how we reduced our features while retaining synergistic interactions and maintaining A-M's performance by employing Recursive Feature Elimination [21, 47].

4.1 Feature Space Construction

To classify Issues and PRs, we use textual similarity measures. After defining the measures we use, we detail how we construct the full feature space to capture the structure of issues and PRs.

Similarity Measures: In order to compare the subject of an issue and a PR, accompanying text documents such as commit messages, source code changes, and conversations are transformed into a vector representation using tf-idf, a discriminative model operating at the fine-grained level of *term frequencies*. We preprocess all documents to: remove punctuation, split tokens using whitespace and code conventions (as well as retaining unsplit tokens in the case of identifiers), stem [41], remove stopwords, and exclude single-character tokens.

We also use tf-idf to measure document similarity. Among offline approaches, it is state of the art [4, 23, 60] and one can naturally extend it to the open-world setting by dynamically maintaining an idf estimate. We transform

[1] https://github.com/PPPI/a-m/blob/master/Util/github_api_methods.py

documents in our corpus D into term vectors to enable comparison. The value for each term t is its term frequency *tf*, the number of times it occurs in a given document d, weighted by its inverse relative frequency in the corpus:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \log_2 \left(\frac{|D|}{|\{d' \mid t \in d' \in D\}|} \right). \tag{1}$$

We replace terms that are either too rare (occur a single time in our corpus) or are too frequent (present in more than 95% of the documents in the corpus) with the unknown, or out-of-vocabulary, token. We make the non-standard choice for tf-idf parameters to ensure a larger vocabulary for small projects as standard practice would induce too many unknown tokens. We choose to eliminate only those terms that occur a single time to maintain a diverse vocabulary. We use the tf-idf implementation from gensim [45], which was designed to handle large corpora efficiently.

We then use this representation to compute cosine similarity:

$$CS(p_j, i_k) = \frac{p_j i_k}{|p_j||i_k|}, \tag{2}$$

where j may take values from {'title', 'description'} and k from {'title', 'report', 'comment'$_1$, ..., 'comment'$_n$}. Only $\text{CS}_{\text{full-context}}$ makes use of all pairs, the other cosine similarity features restrict j and k as shown in Table 1.

We represent documents as a bag-of-words:

$$\text{bow}(d) = \{(t,\ \text{tf}(t, d)) \mid t \in d\}. \tag{3}$$

As is conventional in the traceability literature, we apply Jaccard and Dice similarity to this representation. Both measures account for multiplicity when computing intersections and unions of bag-of-words representations.

$$J(p_j, i_k) = \frac{|\text{bow}(p_j) \cap \text{bow}(i_k)|}{|\text{bow}(p_j) \cup \text{bow}(i_k)|} \tag{4}$$

$$Dice(p_j, i_k) = \frac{|\text{bow}(p_j) \cap \text{bow}(i_k)|}{\min(|\text{bow}(p_j)|, |\text{bow}(i_k)|)}, \tag{5}$$

where $j \in \{'\text{title}', '\text{description}'\}$ and $k \in \{'\text{title}', '\text{report}', '\text{comment}'_1, \ldots, '\text{comment}'_n\}$. Only $\text{Jaccard}_{\text{full-context}}$ and $Dice_{\text{full-context}}$ make use of all pairs; the other similarity features restrict j and k as shown in Table 1.

The Feature Space: Developers construct issues and PRs over time. As Figure 2 shows, PRs and issues can, and often do, overlap in their topics and even their text. Previous work on retrospectively repairing *commit-issue* links in an offline context exploited similar overlap [23, 36, 52, 60]. We build a feature space by considering features employed by the previous state-of-the-art tool [23], which was created from first principles and adapted to the PR context. Along with Jaccard and Dice, we apply cosine similarity to all PR or Issue fields, and the 'title-title', 'title-report', 'description-title', and 'decription-report' subsets of the PR-Issue pair. We also consider seven new features — 'lack of description', 'size of branch', 'number of files touched', 'report size', 'participants', 'reopens', and 'existing link' from Table 1— that allow the model to learn the new special entities ('no_pr' and 'no_issue') we employ (Section 3.1). A link to either of these new entities is equivalent to identifying PRs or issues that should be classified as *unlinked*. To capture their properties and interrelations, we consider four groups of features: *textual*, *social*, *temporal*, and *structural*.

Textual Features: These features compare issue and PR artefacts with files modified by changes in a PR. They assume that related issues and PRs share common terms. Previous, offline work [23] relied on commit summarisation tools such as ChangeScribe [24], limiting applicability to programming languages supported by the summariser. We exploit a PR's title and description, to avoid reliance on code summarisation tools. We compute these features by converting artefacts to tf-idf vectors and computing the described similarity measures.

Table 1. Features constructed from a document vector representation and metadata. p is a PR object, i is an Issue object, e is a traceability link, and π^0 projects the first component of a traceability link. CS denotes cosine similarity and M denotes metadata-based features. A direct reference to an object indicates the concatenation of all text from its constituent parts. We propose the bolded features; the rest are due to Le *et al.* [23].

Feature	Description
$\text{CS}_{\text{full-content}}$	cs(p, i)
$\mathbf{CS_{title\text{-}title}}$	**cs(p.title,i.title)**
$\mathbf{CS_{title\text{-}report}}$	**cs(p.title,i.report)**
$\mathbf{CS_{description\text{-}title}}$	**cs(p.description,i.title)**
$\mathbf{CS_{description\text{-}report}}$	**cs(p.description,i.report)**
$\mathbf{Jaccard_{full\text{-}content}}$	js(p, i)
$\mathbf{Jaccard_{title\text{-}title}}$	**js(p.title,i.title)**
$\mathbf{Jaccard_{title\text{-}report}}$	**js(p.title,i.report)**
$\mathbf{Jaccard_{description\text{-}title}}$	**js(p.description,i.title)**
$\mathbf{Jaccard_{description\text{-}report}}$	**js(p.description,i.report)**
$\mathbf{Dice_{full\text{-}content}}$	**ds(p, i)**
$\mathbf{Dice_{title\text{-}title}}$	**ds(p.title,i.title)**
$\mathbf{Dice_{title\text{-}report}}$	**ds(p.title,i.report)**
$\mathbf{Dice_{description\text{-}title}}$	**ds(p.description,i.title)**
$\mathbf{Dice_{description\text{-}report}}$	**ds(p.description,i.report)**
files	|{filename|filename ∈ p, filename ∈ i}|
M_{reporter}	1 if the p.submitter = i.reporter, else 0
M_{assignee}	1 if the p.submitter = i.assignee, else 0
M_{comments}	1 if the p.submitter ∈ i.replies, else 0
$M_{\text{top 2}}$	1 if the p.submitter ∈ i.top-2, else 0
$M_{\text{engagement}}$	$\frac{\lvert\{c \mid c \in \text{issue.replies, c.author=p.submitter}\}\rvert}{\lvert\text{issue.replies}\rvert}$
Lag	$\frac{\min(\{\text{abs(p.timestamp}-e\text{.timestamp}) \mid e \in \text{i.events}\})}{\text{developer-fingerprint(p.submitter)}}$
Lag-open	p.timestamp − issue.open.timestamp
Lag-close	issue.close.timestamp − p.timestamp
Lack of description	**1 if the p has no description, else 0**
Size of branch	**|p.commits|**
Number of files touched	**|{file.name|file ∈p.diff}|**
Report size	**len(i)**
Participants	**|{c.author | c ∈ issue.replies|}**
Reopens	**|{t | t ∈ i.transitions · t.to = open}|**
Existing Link	**1 if ∃e ∈ E · π^0(e) = i, else 0**

Social Features: We construct these features from reporters, assignees, and discussion participants. We assume that the developers who solve an issue are likely to discuss it with the reporter, or, under contribution guidelines that require having an issue open to which a PR refers, is both the reporter and the PR author. These features, as seen in Table 1, are almost all binary. $M_{\text{engagement}}$ is the only exception: it measures the proportion of comments that are made by the submitter of a PR and captures the intuition that a more engaged discussion participant is more likely to contribute via a PR.

Temporal Features: These features capture properties of issue state transitions — *e.g.* from 'open' to 'closed' or 'won't-fix' *etc.* We assume transitions, such as closures or reopenings are related to developer activities and thus may correlate with PR events. Starting from Le *et al.*'s [23] temporal features, we adapt them to the PR setting by using the PR fields equivalent to the commit fields used by Le *et al.* Additionally, we model the behaviour of individual developers in terms of the expected time between their most recent interaction with an open issue and their submission of a PR to address that issue. We calculate the mean and variance of a developer's past behaviour, and normalise the elapsed time in terms of standard deviations from that mean. This allows the model to use a notion of expected time until interaction as a feature that is scale normalised, to allow individual variation in development time.

Structural Features: These features capture properties relating to the structure of either an issue or PR. They capture signal that aids classifying them as either linkable or not. For a PR, the first feature considered, presence of a description, checks that the PR provides non-trivial information. The next two proxy PR size in order to learn what constitutes an unfocused (too large) or trivial (too small) PR. For issues, we consider four features: The size of the report, the number of participants, how many times the issue was reopened and if it is already linked to a PR. We assume that issues with higher engagement are more likely to be eventually linked to a PR. Issues that are reopened tend to have multiple links to PRs, unless PRs are later merged. Finally, repositories tend to prefer to merge issues and PRs with other issues and PRs rather than add multiple links. Hence, we consider the presence of a link to be a signal that additional links are unlikely.

To validate including these features, we considered four configurations: using only Le *et al.*'s features [23], including only PR-related structural features, only issue-related structural features, and including features related to both. When we add PR-specific features or both PR and Issue specific features, we found, over our development dataset, an improvement in average precision — from 73% (on a data set that did not differentiate PRs and issues) to 87% (issue) and 93% (PR) — at little cost to recall since all configurations report around 9% recall. While this difference is not statistically significant on our dev-set according to a Mann-Whitney U test, we still included them as sufficiently informative.

4.2 Feature Selection

When selecting features, just using the top k features by importance is tempting. Doing so can, however, miss synergies, redundancies, or antagonistic relationships between features. In our case, using the top k by importance indeed misses redundancies in textual features. We show how we apply the principled approach, Recursive Feature Elimination [47], to enrich A-M's feature set. Feature selection allows us to adapt to a project by selecting existing features. This can eliminate features that confuse the classifier. It does not, however, discover new features. Feature selection's main benefit is that it requires relatively little data and its cost is proportional to the classifier training time. Given sufficient data, an interested developer or researcher could replace A-M's vectors (over a fixed feature set) with an appropriate embedding procedure, such as a pre-trained neural network. We leave this exploration to future work.

To formally analyse the efficacy of matching on text artefacts and other features, we employ a small but representative internal development set of projects separate from our training and testing data. This internal dev set is constructed by bucketing the projects by size into four equal groups and uniformly at random picking a project from each bucket. We analyse the features listed in Table 1. To reduce the number of features, we first consider the linear correlation of the features on our development set. We hierarchically cluster the features using Pearson R^2. We want to preserve our prior that textual features, meta features, social features should be good linear predictors of each other, while being less efficient at predicting across feature type, such as textual features used as a predictor for social features. We manually examine various clusterings over our internal dev set and select $R^2 \geq 0.6$ as our threshold because it maximally aligns with our intuition. Within these groups, we

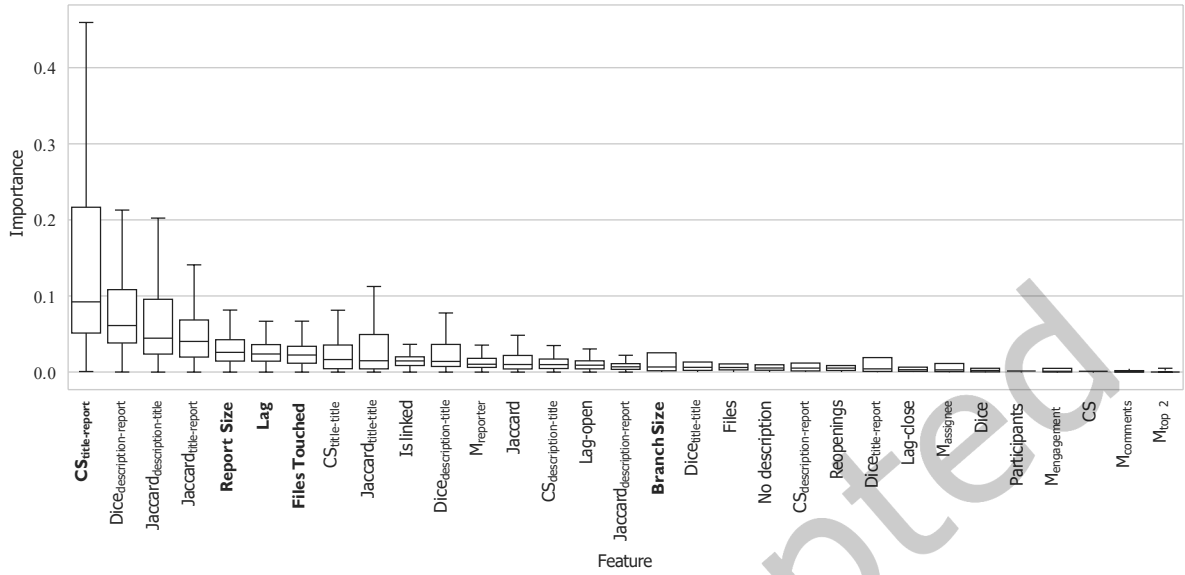


Fig. 4. Feature Importance as computed using a Random Forests Classifier on the development set. We performed recursive feature elimination to select the final feature set while considering synergistic and antagonistic interactions between the features. The final set of features can be seen in bold. We do not show outlier values for presentation reasons.

only employ the features with the highest importance according to a Random Forest model. While such feature pruning ignores higher-order feature interactions, we remark that many features we expect to correlate with each other are also direct substitutes for each other (such as similarity measures). Because they are direct substitutes and, ultimately, proxies for mapping the Issue-PR vector space, we decided that only working with first-order interactions is sufficient. As sanity check, we also perform recursive feature elimination, as described below, using all features. Using the resulting features, A-M's performance does not vary in a statistically significant way.

On this pre-pruned feature set, we then perform recursive feature elimination to further reduce the considered set. The core idea of recursive feature elimination is to ablate features one-by-one as long as the observed performance does not significantly degrade. Figure 4 shows feature importance over the full feature set presented in Table 1. It suggests that Jaccard should be included in the final set of features. However, recursive feature elimination determines that removing it does not impact model performance since Jaccard and $CS_{\text{title-title}}$ are linearly correlated, although below our previous R^2 threshold, and the latter has a higher importance. Thus, we reduce the number of features we use for evaluation to $CS_{\text{title-report}}$, Lag, Report Size, Number of files touched, and Branch Size (bolded in Figure 4). A consequence of modelling negative links explicitly is increased importance of features that depend on the size of the artefacts; this is unsurprising as they represent good proxies for determining unlinkable pull-requests and issues.

In all scenarios, we estimate importance using the standard Random Forest implementation provided by SciPy [19]. When training and validating our system, we only consider pull-requests and issues whose last update is within a certain window of relevancy, which we set to seven days as per the recommendation in Wu *et al.* [60], to limit the number of candidate links considered both for feature selection and model training. For Random Forest hyperparameters, we use Decision Trees and 100 estimators, in line with the recommendation

Table 2. Summary Statistics for a uniformly sampled subset of the Java Generalisation Corpus, ordered by the number of existing links in the corpus. A range of projects sizes is included. In the majority of projects, most PRs are not linked to an issue; this can be seen in the last column, the median linking rate being only 0.46 (0.43 for the sample)

Repository	Links	PRs	Commits	Issues	Links/PR
pinterest/teletraan	11	403	774	41	0.03
mikepenz/MaterialDrawer	18	136	1982	1804	0.13
google/android-classyshark	21	92	597	63	0.23
roughike/BottomBar	44	123	789	687	0.36
facebook/fresco	65	241	1839	1567	0.27
googlei18n/libphonenumber	87	583	1476	1255	0.15
square/leakcanary	124	210	376	580	0.59
square/retrofit	275	771	1562	1623	0.36
ampproject/amphtml	3061	6123	6872	4170	0.5
Sample Total	3706	8682	16267	11790	**0.43**
Corpus Total	19785	43101	174456	67720	**0.46**

from Oshiro *et al.* [30], leaving other settings to SciPy [19] defaults. We evaluate the features over all (p, i) pairs for each repository; Figure 4 shows the results.

5 EXPERIMENTAL SET-UP FOR EVALUATING AIDE-MÉMOIRE

Evaluating A-M required substantial logistical effort, which we now describe. We first present our two corpora, a Java corpus, which enables comparison with RCLinker, the previous state of the art, and a multilingual one, on which A-M can spread its wings. We explain how we weakly label these corpora to obtain training data. We introduce performance measures for predicting lists and our adaptation of accuracy to account for our new 'no_pr' and 'no_issue' predictions.

5.1 A Tale of Two Corpora: Java and Multilingual

We use two corpora in this work. Table 2 presents the detailed statistics for a subsample of our Java corpus, as well as aggregate statistics across all 47 repositories it contains. We use it to compare A-M with RCLinker, the previous state of the art. Table 3 presents detailed statistics for a subsample of our multilingual corpus, as well as its aggregate statistics. The multilingual corpus comprises 213 repositories and six programming languages. We use it to evaluate A-M in detail in its native setting. Our multilingual corpus *contains* the Java corpus as a subset. The SQL queries used to select projects are available online [39].

The repositories contained in our new corpora are sampled from the GitHub GHTorrent dataset [14], which at the time of our sample contained a total of 3704251 repositories. We exclude projects that have fewer than 100 lines of code across all files, ensuring that there is sufficient code in the repository, such that per project vocabularies contain at least 100 terms. We sort by the number of 'watchers' as a proxy for popularity, and select the 50 most popular repositories, a figure limited by the time required to mine relevant data. The popularity bias reduces the inclusion of low quality GitHub projects and increases the number of projects with high issue and PR activity. We exclude projects using natural languages other than English, as we use the Porter Stemmer built for the English language.

The languages for the multilingual corpus were selected first by uniformly sampling five languages from the most popular 15 as reported by GitHub [11]. We applied these two restrictions to constrain the cost of mining the corpus. We selected Scala, TypeScript, C++, C#, and JavaScript. We again exclude small projects and select the 50

Table 3. Summary Statistics for a uniformly sampled subset of the Non-Java Generalisation Corpus, ordered by the number of existing links in the corpus. A range of projects sizes is included. In the majority of projects, most PRs are not linked to an issue; this can be seen in the last column, the median linking rate being only 0.49 (0.39 for the sample)

Repository	Links	PRs	Commits	Issues	Links/PR
SaschaWillems/Vulkan	18	117	1176	246	0.15
OptiKey/OptiKey	53	147	2170	186	0.36
aseprite/aseprite	53	119	5745	1423	0.45
coryhouse/react-slingshot	75	175	600	277	0.43
akveo/ng2-admin	183	558	1144	669	0.33
angular/zone.js	222	403	633	451	0.55
facebook/draft-js	273	457	679	859	0.6
kadirahq/react-storybook	300	607	4694	964	0.49
hakimel/reveal.js	302	617	2095	1335	0.49
citra-emu/citra	468	1771	4972	1071	0.26
Sample Total	1947	4971	23908	7481	**0.39**
Corpus Total	97637	200414	763002	323667	**0.49**

most popular projects for each language. Omitting those we could not successfully crawl from GitHub due to rate limits, we extracted 238 repositories. After additional removing repositories that have too few examples for the results to be meaningful, the final corpus size is 213.

5.2 Weak Labelling

Weak labelling enables the use of large corpora for supervised learning when manual annotation would be prohibitively expensive. While weakly labelled data can be noisy, one need only take care to use it in conjunction with machine learning techniques that can cope with noise. Mondrian Forests are one such technique, as we observe in Section 7.2.

As a weak labeller, we heuristically link PRs to issues when a PR-issue pair is explicitly linked in the GitHub metadata or when they both contain a SHA. We identify SHAs as `r'[0-9a-f]\{5,40\}'` or numerical (`r '[\n\r\s]\#[0-9]+'`) tokens that can be disambiguated to a unique artefact in the project. We apply this heuristic to PRs, issues, commit messages, and diffs for each project in our corpora. We assume that unlinked PR-issue pairs are negative examples, and linked pairs represent positive examples.

Because our corpora are not manually annotated, some negative examples are false negatives, links missed by developers, and some positive examples are false positives. To mitigate this threat, we manually assessed 30 positive links and 30 negative links sampled uniformly from our corpus. On an initial sample, we found significantly many false links to the issue/pull-request with the ID '#1'. We removed these links and, after resampling and re-assessment, found more than 80% of the filtered links recovered by the heuristic to be correct. This is sufficient for A-M, because A-M, by virtue of its use of Mondrian forests, is robust to noise, as our results show in Section 7.4. Crucially, the improvement is larger than the noise in our ground truth labelling: 62% > 20% for precision and 31% > 20% for F1-score (Table 7). The demonstrates sufficient signal to be confident in this result [59].

To further validate the ground truth data we use, we have inspected 100 links (20 links per project) across our development set. We have found that the heuristics employed have a 78% precision with a range of 55–95%. Two authors have performed the annotation and the Cohen's kappa agreement is 0.643 indicating substantial agreement. A single project is below 80% precision at 55%, which is Facebook/react-native. This is due the project

being splintered into multiple GitHub projects each with a separate issue tracker; however, with substantial cross-project referencing. This is not a practice we generally observed among other Open Source Software projects, and hence do not consider it a threat to our data gathering methodology. We have, however, excluded 72 projects from our corpus after heuristic linking, because they have fewer than 25 recorded links, which would lead to a performance metric computed over at most 5 queries due to our 80%/20% train/test split.

These two heuristically-linked corpora are much larger than those previously used; we have made them available to other researchers [40].

5.3 Measuring Performance

Suggesting traceability links to a developer naturally requires producing suggestion lists. Measuring A-M's performance needs performance metrics for lists. We first present Mean Average Precision, a standard metric from Information Retrieval for list prediction tasks. We designed A-M to choose its suggestions with care, to avoid burdening developers with low quality suggestions. To measure how well we succeeded, we introduce hit rate (HR), which counts how often A-M makes a suggestion when it should and how often A-M remains silent when it should.

Mean Average Precision. A-M presents a ranked list of suggestions to a developer containing PRs to be linked (when closing an issue) or else issues to be linked (when submitting a PR). To quantify performance, we use *Mean Average Precision*. We describe a single request for a suggestion list as a query q, where Q is the set of all such queries to A-M from the training data. The list produced in response to an individual query q is r_q; its length is $|f_q|$. For $q \in Q$, $\text{rel}_q(k)$ is an indicator function that returns 1 if the k^{th} item in the response list r_q is relevant and 0 otherwise, including for k values *s.t.* $k > |r_q|$. $P(k)$ is the precision of the first k items returned for a given query: number of correct over k.

In order to consider both the precision of individual suggestions, and, crucially, their positions within the suggestion list, we use Average Precision (AP):

$$AP(q) = \frac{\sum_{k=1}^{k=\infty} P_q(k)\text{rel}_q(k)}{\left|\{k \mid k \in \mathbb{N} \wedge \text{rel}_q(k) = 1\}\right|}. \tag{6}$$

Average Precision is the average of all precision values at each recall value a query q may take as you increase the length of the output list. To measure the quality of suggestions across all queries in Q, we use Mean Average Precision [29]; it measures the quality of a set of ranked lists of suggestions.

Definition 5.1. Mean Average Precision (MAP)

$$\frac{1}{|Q|}\sum_{q \in Q} AP(q). \tag{7}$$

We treat each PR submission or Issue closure event as an individual query q and measure the quality of our suggestions per project. This considers the order in which suggestions are offered, and amortizes outliers such as PRs that either have a high number of links or has missing fields. In other words, MAP measures the preponderance of true links in the responses offered by A-M during the replay of repository events in the event suffix (Section 6.2).

Hit Rate. Our task demands a few high-quality suggestions or no suggestions at all, rather than many low-quality suggestions because suggesting irrelevant links wastes a developer's time. For this reason, A-M truncates its suggestions at 'no_issue' or 'no_pr'. MAP cannot be applied to scenarios with no relevant documents. Indeed, it does not apply to empty lists, because $AP(q)$ is undefined for $|r_q| = 0$. A-M's truncation tactic makes this case

occur fairly often. The set of queries where MAP does not apply is

$$Q' = \{q \mid \forall k \in \mathbb{N} \cdot rel_q(k) = 0\}; \tag{8}$$

that is, the set of all queries for which there are no relevant artefacts to predict. To overcome this limitation, using the above set, we introduce *List Hit Rate*:

Definition 5.2. List Hit Rate

$$\frac{1}{|Q|}\Big(\sum_{q \in Q - Q'} AP(q) > 0 + \sum_{q \in Q'} |r_q| = 0\Big). \tag{9}$$

Hit rate (HR) proxies impact on developer workflow. HR measures not only when A-M correctly predicts a link, but also when it correctly does not. HR penalizes A-M for predicting links when there are none and $AP(q)$ is undefined. HR lifts the notion of accuracy from binary classification to lists. The fact that we predict the absence of links inverts the usual logic. A false positive occurs when A-M returns one or more suggestions when no suggestion should be made, while a false negative is when A-M returns an empty list when at least one link exists in the data. Conversely, a true positive is when at least one correct suggestion is made one at least a suggestion should be made, and a true negative is when A-M outputs an empty list when no suggestions are to be made.

To obtain Precision, Recall and F1 scores for A-M's output, we truncate all lists to length k or the rank of the no_{issue/pr} entity, whichever occurs first, then we concatenate all predictions together, before computing precision p, recall r and F1-score (f1):

$$p = \frac{\text{tp}}{\text{tp} + \text{fp}}, \tag{10}$$

$$r = \frac{\text{tp}}{\text{tp} + \text{fn}}, \tag{11}$$

$$\text{f1} = \frac{2pr}{p + r}. \tag{12}$$

5.4 The Longitudinal Evaluation of Aide-mémoire

A-M is the first commit-issue (where we extract commit-issue links from A-M's PR-issue links) predictor that aims to suggest links to a working developer *as they submit commits or issues*; previous work bulk proposes commit-issue links in project histories. It solves an inherently online problem, which restricts any online predictor's training data. This rules out cross-fold validation, which suffers from data leakage: a predictor might incorrectly train on links from the future relative to the time of a given suggestion.

To present data leakage, we train A-M with longitudinal evaluation [16]. We first flatten each repository into a chronological sequence of events; we consider 'PR index', the position of a PR in the chronological sequence of all project PRs, as a proxy for elapsed development time. We then split each project into a prefix and suffix to obtain a 80%/20% training/test split over the PR indices. We replay the suffix of this event stream, simulating the creation of artefacts such as issues and PRs, and request link predictions from each trained classifier at the time of PR submission or issue closure. In the case of A-M, to simulate developer feedback, if the classifier has a correct prediction in top five, it uses those predictions to update itself. The tf-idf model is continually updated. A-M maps new tokens to a special unknown token. We restrict our evaluation to leave-one-out (1-fold) due to the computational cost of simulating all repository events for large repositories, which in our case encompasses 171 of the 213 projects in our multilingual corpus. We use cross-project performance to assess generalisation, rather than performing multiple longitudinal splits.

To implement our method, we adapted scikit-learn's TimeSeriesSplit [48] to split over PR indices and integrated with A-M. The interested reader can find our implementation online [39][2].

Table 4. The hyperparameters for both classifiers (RandomForestClassifier and MondrianForestClassifier). We used default value for all parameters unless the default value is provided in parenthesis, specifically, we used non-standard values for the number of estimators and class weighting. These decisions were taken using a logarithmic step grid-search on the dev-set only.
*Not included in the table for brevity is the values that class_weight can take: Optional[{balanced, balanced_subsample, custom}]; where custom is provided as a dictionary or list of dictionary from classes to weights.
**Represents 'balanced_subsample'

Parameter	RandomForestClassifier	MondrianForestClassifier
n_estimators: int	128 (default: 100)	16 (default: 10)
bootstrap: bool	True	False
max_depth: Optional[int]	None	None
min_samples: Union[int, float]	2	2
criterion: {gini, entropy}	gini	N/A
min_sample_leaf: Union[int, float]	1	N/A
min_weight_fraction: float	0.0	N/A
max_features: Optional[{sqrt, log2}]	sqrt	N/A
max_leaf_node: Optional[int]	None	N/A
min_impurity_decrease: float	0.0	N/A
min_impurity_split: Optional[float]	None	N/A
oob_score: bool	False	N/A
class_weight*	bal_sub** (default: None)	N/A
ccp_alpha: float≥0.0	0.0	N/A
max_samples: Optional[Union[int, float]]	None	N/A

5.5 Hyperparameters

To choose our hyperparameters for both RandomForestClassifier (RF) and MondrianForestClassifier (MF), we performed a small logarithmic step grid-search from 8 to 256 over number of classifier. We used their default values for all other hyperparameters. 128 (RF) and 16 (MF) performed well on our dev-set. These values are also close to the default values: 100 (RF) and 10 (MF). The former is in line with advice from Oshiro *et al.* [30]. We expected class imbalance in our dataset, so we opted to use a class reweighing scheme (balanced subsamples) to improve the performance of Random Forests. This ensures that, during training, a tree estimator sees a subsample with a balanced number of samples from each class in the training set. MondrianForestClassifier learns splits online via a Mondrian Process, so it does need hyperparameters for splitting, and therefore lacks most of a RandomForestClassifier hyperparameters. Table 4 shows default values as well as our non-standard choices.

6 REPRODUCING RCLINKER, THE STATE OF THE ART IN OFFLINE COMMIT-ISSUE SUGGESTION

As stated in the introduction, A-m tackles a new problem: for pull requests (PR), it suggests PR to issue links at submission. This problem is inherently online. To baseline A-m, we turn to the related offline, commit-issue linking task. Tools solving this problem aim to repair histories for use in other productivity-enhancing tools that consume histories. Here, we considered two: RCLinker [23] and Rath *et al.*'s link classifier [44]. These approaches use different data sets and different feature spaces. Rath *et al.*'s have not made their classifier publicly available. Another blocker for us is that Rath *et al.*'s link classifier relies on features that would be in the future in our

[2]https://github.com/PPPI/a-m/blob/master/Util/CrossValidationSplits.py

online setting. Thus, we turned to RCLinker. To effect this comparison (Section 6.2), we needed to reproduce and adapt RCLinker. Because of its importance, we sanity-check our reproduction of RCLinker, RCRep, against published results, and then adapt it to our problem in two stages: lifting it from commits to PRs and replacing its commit summariser. Section 6.2 details the comparison between A-M and RCRep and demonstrates that, unsurprisingly, an approach, like A-M, which is tailored to the PR-issue linking, outperforms one that is not. Concretely, A-M outperforms by 0.62 in precision, achieving 0.76 at similar recall (0.37 vs 0.36). RCRep's similar recall is a testament to RCLinker's solid design.

6.1 Constructing RCRep

To adapt RCLinker to A-M's PR-issue linking task, we first had to reproduce it on its original commit-issue linking task. We then modified it to solve the PR-issue linking task with a pair of adaptors: an input that translates PR-issue linking into a commit-issue linking problem and an output adaptor that maps RCLinker's solution back to PR-issue links. As a faithful reproduction would discard potentially useful information such as PR descriptions, we also explore using them as a substitute for commit summaries.

RCLinker defines a feature space over commit–issue links. We discussed these features in Section 4; Le *et al.* apply these features to commits, not PRs. As usual, RCLinker first extracts these features from an input project. As commit messages are often short (as best practice encourages [25]), Le *et al.* employ ChangeScribe [24] to automatically summarise commits; RCLinker extracts its textual features from these summaries. It is its reliance on ChangeScribe that restricts RCLinker to Java. They train a Random Forest classifier on the derived feature vectors. RCLinker works over the space of all possible links ($I \times C$); to avoid overwhelming the classifier with negative links, they introduce a novel undersampling algorithm. To form negative links, this algorithm replaces issues or commits in a true link with up to five of their nearest neighbours in the feature space. This undersampling does not scale well to the online case — the time taken to undersample grows linearly with both the total number of links *and* the number of artefacts associated with each link, *i.e.* $O(l(i + c))$. We empirically observed Mondrian Forests to be resilient to the class imbalance problem this undersampling mitigates.

We implemented RCLinker's feature set and classifier; this serves as the internal classifier for our adapted variants. It solves the commit-issue linking problem[3]. Our core classifier reproduction depends on ChangeScribe [24] for commit summaries. Our implementation uses a modified version of the ChangeScribe Eclipse plugin that can be run headlessly. As our RCLinker classifier is the lynchpin of our adaptations, we validate it on Bachmann *et al.*'s original commit-issue problem [4] on two corpora: the Apache Commons [4] on which Le *et al.* evaluated RCLinker and our Java corpus from Section 5.1. Table 6 shows the results on the Apache Commons corpus. Our reproduction obtains similar results with only a slight penalty to Recall. We note that the Random Forest implementations in SciPy [19] and in Weka [15] may use different defaults. Table 5 shows the result on our Java corpus. It demonstrates strong generalisation beyond the Apache Commons corpus used in the original RCLinker paper and first introduced by Bachmann *et al.* [4].

We adapt our RCLinker reproduction to the PR-issue linking task in two stages. In the first stage, we wrap its classifier in two translators. The input translator converts PR or issue events into queries over commit-issue links. The output translator converts the commit-issue links predicted by the classifier into PR-issue links: it links a PR to an issue if *any* of the commits from that PR are linked to the issue. We name this adaptation "RCRepCS", as it uses ChangeScribe.

In the second stage, we further adapt "RCRepCS" to our setting and replace ChangeScribe with PR descriptions. This allows us to assess the impact of commit summarisation on the task. We name this variant "RCRep". While commit messages are usually short — developers are encouraged to keep them within 72 characters — PR

[3]Reproduction is often thankless and hard. We thank Le *et al.* for being extremely helpful and responsive to our requests during our reproduction of their work.

Table 5. Performance values on the Java Corpus for RCRepCS when solving the commit-issue prediction task. This evaluation confirms that our reproduction of RCLinker is effective; indeed, we find it generalises well beyond the corpus used in the original paper.

Repository	F1-Measure	Precision	Recall
Bilibili/ijkplayer	0.39	0.90	0.27
facebook/fresco	0.59	0.88	0.47
googlei18n/libphonenumber	0.56	0.80	0.48
google/android-classyshark	0.76	0.97	0.65
google/gson	0.45	0.75	0.34
iluwatar/java-design-patterns	0.24	0.46	0.18
JakeWharton/butterknife	0.72	0.72	0.75
mikepenz/MaterialDrawer	0.75	0.76	0.75
nostra13/Android-Universal-Image-Loader	0.79	0.85	0.74
ReactiveX/RxAndroid	0.45	0.75	0.35
roughike/BottomBar	0.55	0.80	0.44
square/leakcanary	0.43	0.53	0.38
square/picasso	0.29	0.35	0.27
wequick/Small	0.49	0.50	0.51
Median	**0.52**	**0.76**	**0.46**

Table 6. Performance values on the original Apache Corpus for RCRepCS when solving the commit-issue prediction task. We bias our reproduction towards higher precision to account for the shift to the perspective use-case, and we observe a higher performance of our reproduction relative to the results reported in Le *et al.* [23] for F1-Score and Precision at the cost of Recall.

Repository	F1-Score		Precision		Recall	
	RCRepCS	RCLinker	RCRepCS	RCLinker	RCRepCS	RCLinker
CLI	0.76	0.61	0.70	0.45	0.88	0.91
Collections	0.82	0.59	0.72	0.43	0.95	0.92
CSV	0.86	0.54	0.87	0.39	0.85	0.88
IO	0.66	0.70	0.96	0.59	0.50	0.87
Lang	0.82	0.72	0.82	0.58	0.83	0.94
Math	0.55	0.70	0.84	0.61	0.42	0.83
Overall	0.74	0.64	0.82	0.51	0.74	0.89

descriptions tend to be longer. We create a new variant of our reproduction, RCRep, which further replaces ChangeScribe summaries with PR descriptions. This configuration assess the quality of ChangeScribe as a source of textual information relative to developer Pull-Request discussions. Table 7 shows the impact of this change; in short, PR summaries outperform ChangeScribe summaries. In the remainder of this section, we use RCRep to refer to both RCRepCS and RCRep.

As Section 5.4 details, we longitudinally compare RCRep and A-M. For A-M, we train it on the first 80% of events in a repository and evaluate it on the last 20%. We request a prediction whenever a PR is submitted or issue is closed. For RCRep, we create a repository view that represents the first 80% of the repository. We then provide it the true links and undersample RCRep links using RCLinker's method described above. We then evaluate RCRep by asking it to predict the links that exist within the last 20% of events from a repository.

Aide-mémoire is an online assistive tool that outputs a variable-length ranked list of suggested links. RCLinker outputs unranked commit-issue links. To enable comparison, RCRep's output adaptor collapses all of its core classifier's suggestions for commits in the same PR into a single list. RCRep orders these suggestions by the output probability from its Random Forests classifier; it breaks ties by the distance between PR and issue identifiers, which GitHub assigns jointly and in increasing order to issues and PRs as they are created. Because we compare both variants to A-M, and RCRepCS is Java-only, Section 6.2 considers only Java projects and shows that A-M outperforms both variants.

A-M and RCRep use random forest classifiers, which require hyperparameter settings: we set the number of trees to 10 for RCRep, in line with the original paper, and 128 for A-M when using a RandomForestClassifier or 16 when using a MondrianForestClassifier, as discussed in Section 5.5; the separation criteria was entropy for RCRep, while Mondrian Trees employ a budget to decide when they should refine the partition of the space [22]. Additionally, we use the default random forest probability cut-off of 0.5. Dynamic cut-off due to prediction of 'no_pr' or 'no_issue' occurs on top of the default cut-off when applicable. In tf-idf, the minimum term count was two and the term cut-off set at 95%. For RCRep undersampling, we use the five nearest neighbours, setting this hyperparameter following the RCLinker authors.

6.2 Benchmarking Aide-mémoire Performance with RCRep

We now quantify Aide-mémoire performance relative to RCRep's. We seek to answer the following question:

RQ1 How well does RCRep, our reproduction of the state-of-the-art at predicting commit–issue links, generalise to PR–issue links?

Unsurprisingly, Aide-mémoire outperforms RCRep on the PR-issue linking task. We further improve RCRep's performance by adapting it to directly to use PR descriptions in place of ChangeScribe's commit summaries.

Table 7 shows detailed results. Overall, Aide-mémoire clearly outperforms RCRepCS and RCRep across the Java corpus. In terms of precision, RCRepCS achieves its best result on twitter / distributedlog , which represents a large and well linked project; in terms of F1-Score, it achieves its best result on the similarly well linked google/flexbox–layout. We note that using PRs only as a source of natural language description improves performance across most projects. We speculate that A-M's use of Mondrian Forests also allows it to update the prior on a per-sample basis, while RCRep, which uses Random Forests, relies on a periodic update of predicates on our feature vectors. This enables A-M to refine precise bounding boxes for the data observed, while RCRep is restricted to splitting planes. We conclude that attempting to solve the issue-PR prediction problem as an instance of the commit-issue task is ineffective even when a change summariser is provided. This validates our decision to solve the issue-PR link prediction separately and make use of PR level metadata. In particular, the high false positive rate and the lack of 'no_pr' or 'no_issue' entities impacts the precision of RCRep. Also, of note is that both A-M and RCRep fail on projects that have very little linking at the PR level — there is insufficient data to train an accurate classifier; one such project is Android–Universal–Image–Loader, which is just barely above our training example threshold employed as a filter on our corpus.

RQ1 Finding (Comparison with SOTA)

RPRecp does not generalise well to PR-Issue linking, achieving 0.14 precision.

While RCRep performs poorly on issue-PR link prediction, we emphasise that RCLinker was designed to predict commit-issue links; we include this comparison to motivate the separate handling of issue-PR links. For completeness, we also evaluate the performance of RCRepCS on the original commit-issue link prediction problem; we provide these results in Table 5. Additionally, RCLinker is applicable to scenarios where there is no

Table 7. Mean performance values of predicting issue-PR links across a sample of the Java Corpus for RCRepCS, RCRep, and A-M. The approach taken by RCRepCS to solve the commit-issue link prediction does not generalise to issue-PR links, even when provided with PR descriptions as summaries, whereas A-M performs well on most projects. Extreme (0.0 or 1.0) results can be observed on small projects where there are few queries in the suffix. The projects with an (*) have had their names shortened for presentation purposes.

Repository	**Mean Precision**			**Mean Recall**			**Mean F1**		
	RCRepCS	RCRep	A-M	RCRepCS	RCRep	A-M	RCRepCS	RCRep	A-M
ijkplayer	0.00	0.00	**0.52**	0.00	0.00	**0.06**	0.00	0.00	**0.11**
AndroidSwipeLayout	0.00	0.00	**0.99**	0.00	0.00	**0.10**	0.00	0.00	**0.18**
fresco	0.01	0.01	**0.92**	**0.51**	**0.51**	0.37	0.03	0.02	**0.53**
redex	0.06	0.16	**0.76**	0.47	0.46	**1.00**	0.10	0.20	**0.86**
libphonenumber	0.01	0.00	**0.83**	**0.50**	0.20	0.26	0.03	0.00	**0.39**
android-classyshark	0.00	0.33	**0.99**	0.00	**0.72**	0.54	0.00	0.45	**0.70**
flexbox-layout	0.48	0.29	**0.79**	**0.79**	0.75	0.46	**0.60**	0.42	0.58
gson	0.06	0.07	**0.92**	**0.48**	0.36	0.35	0.10	0.11	**0.51**
EventBus	0.00	0.13	**0.65**	0.00	0.20	**0.45**	0.00	0.15	**0.53**
java-design-patterns	0.06	0.17	**0.89**	0.19	**0.22**	0.06	0.09	**0.19**	0.12
butterknife	0.00	0.08	**0.73**	0.30	**0.64**	0.30	0.00	0.14	**0.42**
RxJava-Android-Samples	0.41	0.29	**0.71**	0.50	0.43	**0.59**	0.39	0.34	**0.64**
aUPTR*	0.00	0.00	**0.99**	0.00	0.00	**0.25**	0.00	0.00	**0.40**
MaterialDrawer	0.04	0.06	**0.93**	0.53	0.58	**0.79**	0.07	0.11	**0.86**
Hystrix	0.04	0.28	**0.71**	**0.43**	0.32	0.29	0.08	0.30	**0.41**
AUIL*	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RxAndroid	0.03	0.20	**0.86**	0.15	**0.80**	0.50	0.05	0.31	**0.63**
BottomBar	0.01	0.07	**0.86**	**0.28**	0.04	0.22	0.02	0.05	**0.35**
leakcanary	0.09	0.19	**0.88**	0.24	0.30	**0.35**	0.13	0.23	**0.50**
picasso	0.00	0.04	**0.78**	0.27	**0.55**	0.35	0.01	0.08	**0.49**
distributedlog	0.95	0.64	**1.00**	0.10	0.10	**1.00**	0.18	0.17	**1.00**
Small	**0.01**	**0.01**	**0.01**	**0.17**	0.16	0.01	0.01	**0.03**	0.01
zxing	0.15	0.11	**0.72**	**0.99**	0.96	0.20	0.25	0.20	**0.31**
Overall	0.10	0.14	**0.76**	0.30	0.36	**0.37**	0.09	0.15	**0.46**

PR information at all, and thus RCLinker and A-M are *complementary*: historical data could be repaired using RCLinker, followed by adopting A-M to improve future linking. In general, offline tools such as RCLinker are hindered by reliance on commit-level information, which makes them less applicable to modern projects that follow a PR-centred development process.

7 EVALUATING AIDE-MÉMOIRE ON THE ONLINE PR-ISSUE LINKING PROBLEM

First, we show that the state of practice in PR linking remains dismal and that tools like A-M are needed. We then evaluate A-M's performance on the PR-issue linking task over a large multilingual corpus. We observe that A-M is indeed language-agnostic and scales to larger projects, benefits from the use of an online classifier and that it shows tolerance to noisy training data. We assess the usefulness of A-M as a developer aid and show that developers could benefit from such a tool.

7.1 The Dismal State of Issue-PR Linking on GitHub

To determine the state of linking practice in the wild on GitHub, we first queried GHTorrent [13] for those projects that provide a `CONTRIBUTING.MD` file or similar within the root of the project (which is a GitHub convention for the location of the file). `CONTRIBUTING.MD` files are not unique to GitHub; they are indeed common to OSS in general. Our focus on GitHub is a pragmatic one, GHTorrent [13] enables answering such question via SQL queries. Further, GitHub encourages projects to have `CONTRIBUTING.MD` files at project creation via prompts and providing templates. Still, we found only around 4.7% of all the projects available via GHTorrent include such a file (167109 out of 3537142 at the time of query).

Restricting ourselves to these projects, we then considered two ways to obtain subsamples for manual investigation: biasing the sample by popularity, *i.e.* we weight the probability of each project being selected by the number of GitHub stars the project has as a proxy for popularity, and performing a uniform sample over this restricted set of projects. In both scenarios we were looking for explicit statements that links have to be recorded. Example statements are 'It is best practice to have your commit message have a summary line that includes the ticket number [...].', 'This is also the place to reference the GitHub issue that the commit closes.', 'All Pull Requests, [...], need to be attached to a issue on GitHub.'. We also included those projects that referenced an external resource that described a good commit message and that recommended linking to affected ticket numbers.

We sampled 200 projects for each method and were able to obtain the file for 128 projects from the uniform sample, while we obtained 155 for the popularity-biased sample. Of note here is that we attempted to obtain the most recent version of `CONTRIBUTING.MD` from each project's GitHub page rather than the GHTorrent blob. Our results are as follows: one third of the randomly sampled projects made explicit reference to linking practice (43 out of 128), versus around 43.5% (68 out of 155) of the popularity-biased sample. This suggests that the majority of projects on GitHub do not require that the project's collective memory in terms of code-issue traceability links be maintained by Pull Request submitters. It is worth mentioning that we have not considered if the community enforces such a requirement even when it is not codified, or, conversely, if when the practice is codified whether it in enforced. Nevertheless, the difference between the uniform and the popularity-biased sample suggests that more popular projects do tend to require such linking. A further observation is that the popular projects that are maintained by a corporate entity — Google, Facebook, and Microsoft in the sample — tend to have a company-wide policy regarding contributions from the community that require that issues and Pull Requests be linked and that a Pull Request references an already open issue in order for the submission to be accepted.

7.2 The Quality of Aide-mémoire's Suggestions

We evaluate A-M on a multilingual corpus containing 213 projects written in six programming languages. Section 5.4 presents the longitudinal evaluation protocol. The corpus contains a variety of project sizes, so we are able to evaluate both the generality and scalability of A-M. Recall that A-M provides a list of suggested issues to be linked to a PR at submission time, and a list of suggested PRs to be linked when an issue is closed. When considering the performance of our system, we sought to answer the following questions:

RQ2 What proportion of our suggestions contains at least one true link in a k-length list ($k = 1, 3, 5$)?
RQ3 What is the mean average precision (MAP) of our suggestions?
RQ4 Does our system suggest links that were later caught by PR reviewers?

A-M uses 'no_pr' and 'no_issue' entities to truncate its suggestions. If these special entities appear in the k suggestions, we truncate the suggestion list at that point to avoid suggesting unlikely links. Through these entities, A-M correctly learns to be silent when appropriate. It offers a correct suggestion or remains appropriately silent in 86% of cases for $k = 1$; we will henceforth refer to this desirable behaviour as *List Hit Rate*, as per Definition 5.2. If we consider only predictions of actual links (rather than no prediction), A-M suggests a correct

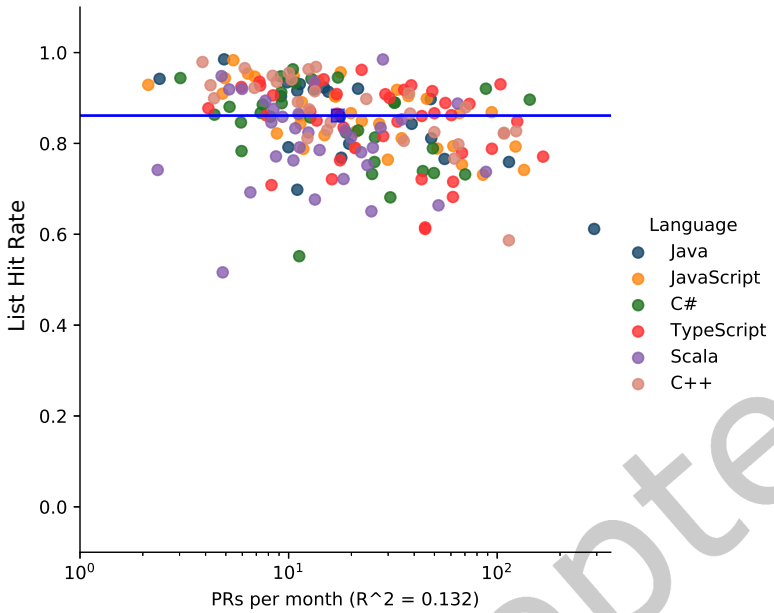


Fig. 5. Performance of A-M quantified by the percentage of queries where A-M is correctly silent when there is no suggestion, or we report at least one correct link when there is a suggestion to be made for list length $k = 5$, median result presented as a blue square. We see that system performance does not degrade severely with the increase of PRs submitted per month (and by proxy) project size; indeed, we find no statistically significant trend associated with the number of PRs per month.

link in 94% of such cases for $k = 1$ (same for $k = 3$ and $k = 5$). Figure 5 shows a more detailed view for $k = 5$. We consider values $k = 1, 3, 5$ to cover the most likely usecases of A-M— $k = 1$ represents trusting the top prediction greedily, while $k = 3, 5$ represent an user glancing over a short list to select the suggestion We now answer RQ2:

RQ2 Finding (Accuracy)

Aide-mémoire suggests true links in a k-length list for $k = 1, 3, 5$ in 94% of the cases.

We now consider full suggestion lists; we use Mean Average Precision (MAP) to measure their quality. Table 8 shows that A-M consistently achieves high MAP. Indeed, only seven projects fall below 0.6 MAP, on three of which A-M fails to learn any model. We observe this failure mode when the training data lacks insufficient true links. However, as results for HR are consistently above 0.5, we have succeeded in making A-M learn to be appropriately silent when it is unsure of its predictions. Overall, A-M produces highly precise suggestions, as our answer to RQ3 shows:

RQ3 Finding (Mean Average Precision)

Aide-mémoire achieves a median MAP of 0.93 (mean 0.89) over our multilingual corpus.

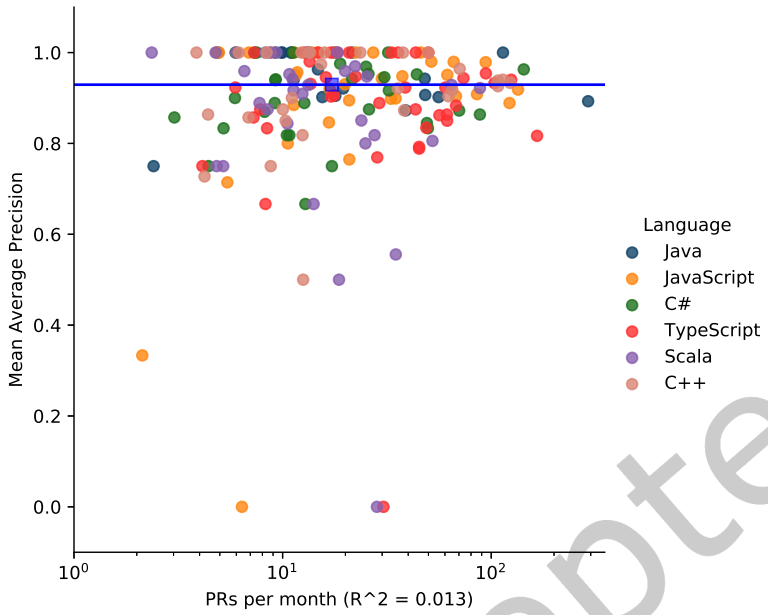


Fig. 6. Performance of A-м reported as Mean Average Precision as a function of the number of PRs per month for a project, median result as a blue square There is no statistically significant trend with scale.

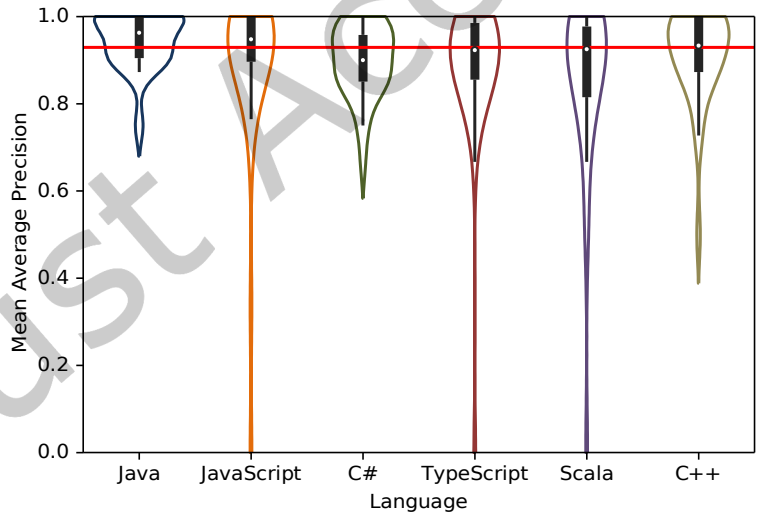


Fig. 7. Performance of A-м reported as Mean Average Precision across languages. There is no statistically significant trend with language (as checked using a two-sample T-test).

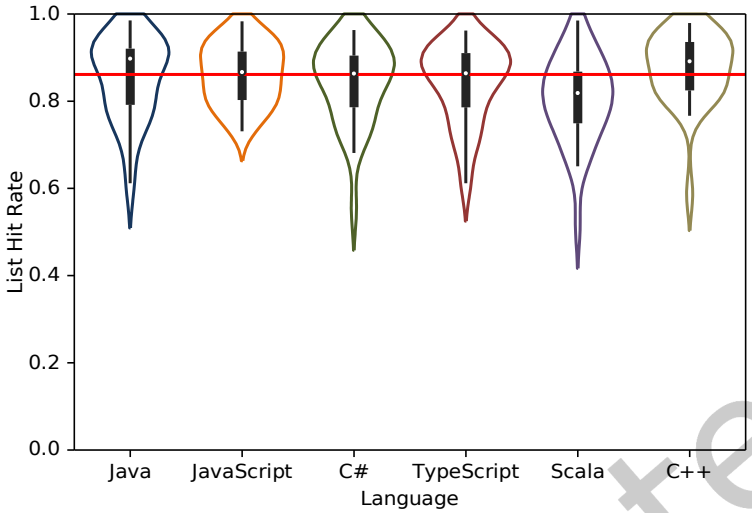
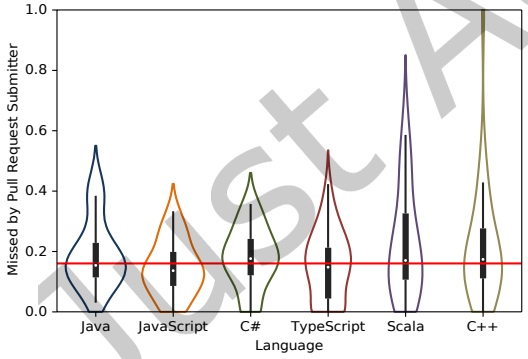
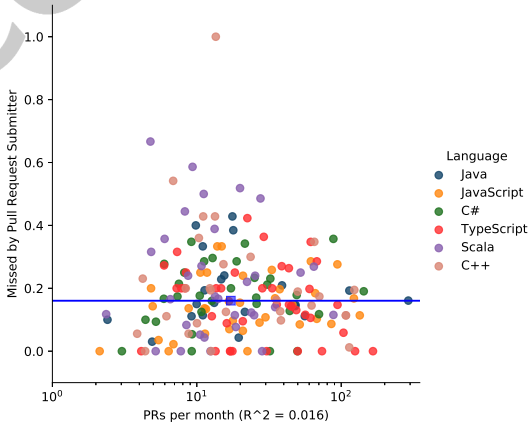


Fig. 8. Performance of A-M quantified by the percentage of queries where we are correctly silent when there is no suggestion, or we report at least one correct link when there is a suggestion to be made for list length $k = 5$. We can see that the project language has negligible impact on performance and the median stays close to 0.86 (the red line); there is no statistically significant difference between languages, except for Scala where we find that there is an effect size of $T = -3.38$ with $p = 0.001$ using a two-sample T-test.



(a) Percentage of links that were originally missed by the PR submitter and predicted by Aide-mémoire for the programming languages considered.



(b) Percentage of links that were originally missed by the PR submitter and predicted by Aide-mémoire as a function of the project's number of PRs per month, median result shown as a blue square.

Fig. 9. Percentage of links missed by the PR submitter and predicted by A-M. Here we can see that our system has the potential to save development time during the PR process by suggesting links originally missed by the PR submitter and later discovered by a reviewer. There is no statistically significant deviation among languages.

Table 8. Performance of A-M using a Mondrian Forest Classifier across a uniform sample of projects from the second corpus, ordered by PRs per month, after filtering projects that have less than 25 links in total.

Repository	Language	PR/m	MAP	HR	P	R
rtyley/bfg-repo-cleaner	Scala	2.36	**1.00**	0.74	**1.00**	0.12
zealdocs/zeal	C++	4.22	0.73	0.93	0.73	0.26
ecomfe/echarts	JavaScript	5.43	0.71	0.98	0.71	0.08
beto-rodriguez/Live-Charts	C#	8.19	0.87	0.86	0.87	0.47
mobile-shell/mosh	C++	12.31	**1.00**	0.81	**1.00**	0.28
sksamuel/elastic4s	Scala	13.28	0.93	0.94	0.93	0.48
square/picasso	Java	14.76	0.96	0.92	0.96	0.35
square/leakcanary	Java	17.64	**1.00**	0.86	**1.00**	0.33
Microsoft/code-push	TypeScript	18.34	**1.00**	0.84	**1.00**	0.25
kriasoft/react-starter-kit	JavaScript	20.84	0.89	0.82	0.89	0.20
AutoMapper/AutoMapper	C#	21.52	0.94	0.83	0.94	0.47
electron-userland/electron-builder	TypeScript	22.39	0.95	0.96	0.95	0.59
NLog/NLog	C#	25.79	0.95	0.76	0.95	0.41
witheve/Eve	TypeScript	29.09	0.89	0.91	0.89	0.44
Microsoft/vscode-react-native	TypeScript	33.26	**1.00**	0.85	**1.00**	0.43
Dogfalo/materialize	JavaScript	37.50	0.95	0.90	0.95	0.36
scala-js/scala-js	Scala	40.63	0.89	0.45	0.89	0.15
spring-projects/spring-boot	Java	48.00	0.94	0.90	0.94	0.23
citra-emu/citra	C++	62.32	0.90	0.77	0.90	0.22
facebook/osquery	C++	65.15	0.92	0.80	0.92	0.43
angular/material2	TypeScript	165.89	0.82	0.77	0.82	0.22
Sample (mean)		31.85	0.92	0.84	0.92	0.32
Sample (median)		21.52	0.94	0.85	0.94	0.33
Overall (mean)		32.13	0.89	0.84	0.89	0.30
Overall (median)		17.64	0.93	0.86	0.93	0.29

A-M offers high-quality suggestions, with the correct link frequently appearing among the first two suggestions. A-M has low recall, managing to recover only 30% of the links removed in the validation suffixes, and requires some initial data to bootstrap the model. Thus, it cannot be deployed at the start of a new project, rather it must be adopted after the project has completed at least one development cycle. A-M successfully learns to predict links that were missed by the original submitter of PRs, *i.e.* those that were suggested by Pull Request reviewers during the code review process: 28% of the recovered links were predicted using only the PR description and its commits before any discussion took place, *i.e.* 18% out of all originally missed links, and as such the tool can help reduce the burden on change reviewers by offering link suggestions to the submitter. A more detailed breakdown of such cases can be seen in Figure 9. In sum, we answer RQ4:

RQ4 Finding (Predicting Missing Links)

Aide-mémoire found 18% of the links that were initially missed by PR reviewers, strong evidence that its adoption would improve PR reviewing.

7.3 Generalising across Languages and Project Sizes

We hope that A-M will change the state of practice in PR-Issue linking. To realise this ambition, it must scale to large projects and language-agnostic, so that large projects and projects using different languages can easily adopt it. To do so, we seek to answer the following question:

RQ5 Does Aide-mémoire's performance degrade with project size or change of project language?

Figures 5, 6, 8, and 7 summarise its performance while Table 8 shows a uniform sample of the results. The MAP is consistently near the median result of 0.95 across languages and there is no statistically significant deviation according to a two-sample t-test. The results for HR show a similar story: no deviation across languages from our general result of 0.86 except for Scala, which has a small ($T = -3.38$) but statistically significant deviation ($p = 0.001$). For results along the project scale dimension, we can see that there is no statistically significant trend with all Pearson R^2 values below 0.1, hence we anticipate no degradation of performance for large projects.

RQ5 Finding (Scalability)

Aide-mémoire scales to larger project sizes with no impact on performance (Pearson's $R^2 < 0.05$ for MAP and Pearson's $R^2 = 0.13$ for HR). A-M also performs consistently across languages — the results show no statistically significant deviation for MAP according to a two-sample t-test, while only Scala has a small performance drop for HR ($T = -3.38$ and $p = 0.001$)

7.4 Resistance to Noise

A-M is online and learns from humans as they accept or reject its suggestions. To err is human, so A-M must contend with, and remain robust in the face of, humans giving it incorrect links. Here, we intentionally mislabel training data and observe its noise tolerance, we seek to answer:

RQ6 To what extent does Aide-mémoire tolerate noise in its training labels?

As weak labelling is inherently noisy, we explore A-M's tolerance to noise by injecting increasing levels of noise and observing the performance profile in terms of P-R Curves. In order to observe and quantify the noise tolerance of A-M, we augment the training data by randomly flipping a proportion of the examples, i.e. adding a false link as true, or marking a true link as false. We control this proportion and explore from no noise (0%) up to 95% noise. We run this experiment over the same internal development corpus as our Feature Selection detailed in Section 4. We perform 10-fold cross validation at each noise ratio, validating performance using the unaltered ground truth. Due to performing cross-fold validation, we additionally take care to elide links and references that may leak information regarding a held-out fold from the training folds. Looking in Precision-Recall space, we observe four main performance regimes with a slight improvement at low-noise (5-10%) for both Precision and Recall followed by a step-wise decline in Precision and a linear decline in Recall. The first regime switch is at 15% noise with Precision going from $0.7 - 0.72$ to 0.66 and Recall from $0.12 - 0.13$ to $0.10 - 0.11$. The second regime switch is at 65% noise with a jump down in Precision to 0.45 and Recall at 0.04. The final regime switch is at 90% noise, with performance fully degrading to 0.2 Precision and 0.01 Recall. Fixing Recall at 8%, we now answer RQ6:

RQ6 Finding (Robustness)

Aide-mémoire tolerate noise of up to 40% of the labels being incorrect if we require precision to exceed 0.66.

Table 9. Performance of A-M using a Random Forest Classifier across a uniform sample of projects from the second corpus, ordered by PRs per month, after filtering projects that have less than 25 links in total.

Repository	Language	PR/m	MAP	HR	P	R
greenrobot/EventBus	Java	2.41	**1.00**	0.77	**1.00**	0.17
wkhtmltopdf/wkhtmltopdf	C++	3.86	**0.00**	0.78	**0.00**	0.00
zealdocs/zeal	C++	4.22	**0.00**	0.50	**0.00**	0.00
feathersjs/feathers	JavaScript	4.82	**0.00**	0.70	**0.00**	0.00
hexojs/hexo	JavaScript	6.89	**1.00**	0.55	**1.00**	0.14
ngrx/store	TypeScript	7.31	**1.00**	0.73	**1.00**	0.30
databricks/spark-csv	Scala	8.47	**0.00**	0.56	**0.00**	0.00
roughike/BottomBar	Java	11.00	**0.00**	0.68	**0.00**	0.00
milessabin/shapeless	Scala	11.17	**1.00**	0.81	**1.00**	0.14
mobile-shell/mosh	C++	12.31	**1.00**	0.56	**1.00**	0.04
AutoMapper/AutoMapper	C#	21.52	**1.00**	0.39	**1.00**	0.15
tildeio/glimmer	TypeScript	28.42	**1.00**	0.83	**1.00**	0.11
chartjs/Chart.js	JavaScript	34.82	0.82	0.46	0.83	0.15
square/okhttp	Java	38.91	0.88	0.61	0.88	0.05
rangle/augury	TypeScript	45.12	**1.00**	0.48	**1.00**	0.02
rangle/batarangle	TypeScript	45.12	**1.00**	0.51	**1.00**	0.07
mxstbr/react-boilerplate	JavaScript	51.62	0.67	0.40	0.67	0.07
realm/realm-java	Java	55.92	0.88	0.56	0.90	0.08
apollostack/apollo-client	TypeScript	60.30	**1.00**	0.71	**1.00**	0.08
ng-bootstrap/core	TypeScript	67.82	0.86	0.52	0.86	0.06
mrdoob/three.js	JavaScript	85.45	0.81	0.66	0.81	0.17
Sample (mean)		28.93	0.71	0.61	0.71	0.09
Sample (median)		21.52	0.88	0.56	0.90	0.07
Overall (mean)		37.09	0.70	0.63	0.70	0.10
Overall (median)		17.70	0.90	0.63	0.91	0.08

We believe this result is strong, because developers using A-M would be familiar with the system they work on and unlikely to mislabel more than 40% of the suggestions. Indeed, we expect mislabellings to be rare, the noise in A-M's data to drop, and A-M's performance to improve over time.

7.5 The Importance of Mondrian Forests

A-M uses Mondrian Forests because they are online. They are much less commonly used than Random Forests. One can, of course, adapt an offline technique, like Random Forests, to operate in a quasi-online mode via periodic batched retraining. However, such retrained models could not quickly benefit from a live developer feedback. Nonetheless, we investigate the feasibility of building such a variant by comparing A-M's default Mondrian Forests implementation to Random Forests trained on the full history in an offline setting, this answers the following ablation question:

RQ7 How much of Aide-mémoire's performance relies on its use of Mondrian Forests over Random Forest?

A-M's Random Forest variant achieves worse results on the same validation systems. While both configurations show a high MAP (0.89 for Mondrian Forest and 0.70 for Random Forest), the Random Forest results fall quite short on HR and recall, showing a recall of only 10%. Detailed results can be seen in Table 9. We hypothesise that

class imbalance, which we observed Mondrian Forests to deal better with, and the lack of a feedback loop during validation impacted the performance of the Random Forest based implementation. Our answer to RQ7 follows:

RQ7 Finding (Mondrian *vs* Random Forest)

Replacing Aide-mémoire's Mondrian Forest classifier with an offline Random Forest classifier penalises its performance: HR falls to 0.63 from 0.84 and recall to 0.10 from 0.30.

As we can see, use of Random Forests, even given the full history, severely degrades Recall, making it doubtful that deploying them in batch mode would be competitive to Mondrian Forests.

7.6 Threats to Validity

A-M faces standard threat to its internal validity, which we mitigate by making our data and experimental harness available for examination and reproduction. Next, we discuss three external threats, of which the last, potential bias from our collection process, is the most important, before turning to the construct validity threat posed by our reliance on weak labelling.

External: Our exclusive use of GitHub and its PRs poses an external threat, because of the extreme class imbalance considering only projects that use PRs, as Kalliamvakou *et al.* explain in their Peril VI [20]. We cannot directly mitigate this threat because we evaluate A-M on PR statistics, like activity. We observe that the bias introduced by our popularity-thresholding reduces the class imbalance in PR usage in GitHub projects.

Another external threat is overfitting, which our feature selection could exacerbate. To counteract this threat, we collected a separate internal dev set. We therefore run the risk of underperforming in the final evaluation if the dev set is unrepresentative of the wider corpus, but we accept this potentially suboptimal performance in order to minimise the risk of overfitting.

When considering predictions, we restrict the candidate set to open issues. We do so because we envision developers using A-M as a triage assistant. This choice, however, means that we may miss valid links in scenarios, such as two PRs referencing a single issue where the first closes it. Section 7.1 shows that CONTRIBUTING.MD's typically recommend maintaining a one-to-one mapping between issues and PRs. GitHub's CONTRIBUTING.MD template and many large corporations, such as FAANG, also recommend this practice. Taken together, these recommendations suggest the risk of such missed links is small.

To collect our project corpus, we first filter by language and size, then use a popularity-threshold over the distribution of projects to avoid low quality or low activity projects (Section 5.1). This collection process introduces three distinct threats to the external validity of our results. We consider only Java, JavaScript, C#, TypeScript, Scala, and C++ to make data collection sufficiently fast and cope with GitHub's bandwidth restrictions. To mitigate this threat, we uniformly sample five languages from an existing ranking of popular languages [11], then added Java to allow comparing A-M with RCLinker, the current state-of-the-art. We filter by size only to remove trivial projects, which abound in GitHub. Our threshold of 100 LOC across all project files is conservative, from first principles, and standard practice. We want to bias our corpus toward maintained and consequential projects. Selecting projects by popularity achieves this. To mitigate the external threat this introduces, we note that our popularity-threshold selection does not directly rely on variables that we use in our evaluation.

Construct Validity: A-M's reliance on weak labelling constitutes the principle threat to its construct validity. Establishing a ground truth over data requires choosing a point on the spectrum between two poles — a golden set of manually labelled and validated data and using unsupervised methods over vast quantities of uniformly sampled raw data occurring in the wild. The experimental goal determines each pole's desirability. Since we designed A-M to be language-agnostic and to scale, we opted to be closer to the unsupervised pole.

We constructed the weak labels of missing links by removing those that had previously been manually recorded by developers. Thus, our training data is representative only of those links included by developers at PR submission or during change review, rather than those they missed completely. Even if actual links and those recognised by developers are statistically different in the feature space, it is still useful to suggest such links; we found that 16% of PRs are subsequently linked and automating this via A-M would save time.

To mitigate this construct threat, we validated our initial knowledge construction by manually classifying links recorded by our heuristics on a sample of 30 positive and 30 negative examples uniformly selected from all projects. We initially found a significant number of links to the issue/pull request with the ID '#1'. Such incorrect links were created due to IDs introduced by a migration tool and detected by our heuristics. We removed these links and reassessed the correctness of the remaining links to find more than 80% of the links detected by our heuristic as correct.

8 QUALITATIVE ANALYSIS OF AIDE-MÉMOIRE'S SUGGESTIONS

In this section, we manually inspect a sample of links suggested by A-M over our development dataset. We detail how and when A-M works, how it fails, and speculate about the root causes of its successes and failures. Based on the type of links we observe, we also speculate about how A-M could impact the state of practice.

8.1 Aide-mémoire's Suggestions

To evaluate the quality of the link suggestions by Aide-mémoire, we uniformly sampled 20 links per project in our development corpus, for a total of 100 links, then manually inspected them. Given A-M'the feature space, we expected links to arise either due to social links (shared contributors/participants) or textual overlap between titles alone or titles and descriptions. From manually inspecting our ground-truth, we expected multiple PRs proposed for the same issue, cross-repository links, and links migrated from external tools to GitHub to confuse A-M.

Our manual analysis confirmed our expectations about A-M's link suggestions, modulated by the reporter's interaction with an issue and the temporal co-occurrence of updates to PRs and issues. None of our concerns about A-M being confused were realised: two did not occur in our sample and A-M was not confused by the one that did. As expected, given its feature space, A-M does tend to suggest links when there is a title or description textual overlap between the PR and issue. This varied by project, but was around 53%. In 20–25% cases, it matched PRs and issues depending on whether the reporter commented on the issue, was assigned to it, or otherwise interacted with it. Further, PR-issue pair suggestions that had updates or comments that are close in time to each other tended to be greatly favoured (85+%). This is a direct consequence of our 14 day filtering window choice. Additionally, we observed A-M suggesting transitive links that have not been *directly* linked in GitHub. These arise when a newer PR is marked as related to an older one that fixed a particular issue. The project does not link this newer PR to the original issue, so, if this newer PR is accepted (the older one rejected as it is subsumed), then the original issue is not automatically closed and remains open. This finding shows that our original concern that multiple PRs per issue did not, in fact, confuse A-M.

A-M achieved the following accuracy under manual analysis: 75% (MPAndroidChart), 80% (RxJava), 35/65% (Guava) , 65 % (React), and 65/75 % (Plottable). Guava rises to 65% when we remove a God-Issue that dominated suggestions. Plottable rises to 75 % if we consider links to a release PR as a true link. A-M's the performance is consistently high if we filter out the God-Issue links. This is reasonable since God-Issues and their links are easy to identify. To obtain Guava's 65%, we simply filter, as God-issues, those issues that link to 10 or more PRs. Migration or cross-tool synchronisation tend to cause God-{Issues,PRs}. Projects, for which either of these cases hold, could use A-M only on their internal tracking system, which would not include God links. In Plottable, we observed developers using PRs as both agile stories and agile epics. We chose 10 as our threshold since, from our

priors, we do not expect an Epic to encompass more than 10 stories, and we wanted our definition of God-issues to be generous and therefore conservative.

A-M exhibits three main failure modes. If a project dedicates an issue czar to manage public issues and patches, spurious links can appear when this issue czar interacts with multiple, unrelated issues and PRs during triage in close temporal proximity. As expected, another failure mode was spurious links caused by migration tools, especially when references or IDs to artefacts from the previous/external system are kept. Further, migration blinds A-M to social features: usernames become text tokens in the body of the issue created by a bot account rather than comments by a GitHub user. Release PRs linked to many issues was the final failure mode. It is arguable whether the release PR or only fixing PRs subsumed by the release PR should be linked to the issues they fix; when they are linked, they become a problematic God PR. In our study, we removed links from issues to a release PR, since a developer cannot use them to find an actual fix, but still needs to find the PR merged into a release PR.

8.2 Aide-mémoire's Potential Impact

Aide-mémoire is a developer assistant: it does not fully automate PR-issue linkage. It aims to speed and improve the accuracy of developer-centered PR-issue linking. Thus, A-M complements a developer ecosystem, such as GitHub. Developers value PR-issue links and do try to create them. To this end, we observed developers copying the issue title into the PR title, or adding links directly in the title (a workaround GitHub does not support), or resorting to overlapping, even if stilted, vocabulary. When interacting with an issue, we observed developers asking questions designed to build or restore the relevant links before proposing a solution as a PR. Thus, we expect A-M to serve as memory aid for developers by suggesting links at issue triage/closing and PR submission. A-M promises to reduce context switching, notably the cost of searching for relevant PRs or issues. Here, A-M already makes time saving suggestions. The transitive relationships that A-M finds will help close issues linked to PRs subsumed by newer PRs, thereby automatically trimming the issue backlog as PRs get merged. In Plottable, developers manually searched for open issues that had already resolved by a merged PR. A-M will speed this search.

9 RELATED WORK

Good PR-issue linking accelerates development: PR-issue links allow developers to more quickly understand why a pull-request was submitted or how an issue was resolved in code; they also permit the use of productivity enhancing techniques like automatic bug localisation [58, 61], or automatic patch generation tools such as R2Fix [26]. PR-issue links are a developer-centric form of software traceable links, as studied in requirements engineering (RE). We compare and contrast A-M to software traceability, we detail how modern development enables and calls for A-M. Finally, we summarise developer-centric work that solves the related commit-issue linking problem in an offline scenario.

9.1 Traceability

Requirements engineering (RE) focuses on stakeholders, decision makers, and their artefacts: requirements, documentation, specification, and design or architectural documents. These artefacts tend to be natural language, text or speech, and often go unrecorded. When they are recorded, they exist in multiple formats, including spreadsheets, email, figures, and printed material. They further encompass developer artefacts, such as source code, pull-requests, commits, and issues, but do not focus on them.

Software traceability seeks to infer *traces* (*i.e.* links) between these heterogeneous artefacts [9]. Missing or hard-to-parse artefacts greatly complicate trace recovery, which is why much work on traceability seeks to provide tooling to decision makers to capture or parse these artefacts [12, 17, 35]. Software artefacts span a multitude

of formats. To extract links between them, traceability tools must contend with the heterogeneity of artefacts. As a result, they often leverage general, abstract features, such as textual features extracted using Information Retrieval techniques [6]. Traceability tools face a deployability challenge. Improving traceability requires capturing developer decisions. To date, researchers have investigated new workflows or invasive instrumentation, which raises privacy concerns, to record these decisions [3].

In contrast, A-M exclusively focuses on a specific software traceability problem — PR-issue link inference. This focus enables A-M to side-step the heterogeneity problem. First, version control and issue tracking are almost ubiquitous in modern software development. PRs and issues are plentiful, well-suited for data hungry machine learning. Second, their format is well-documented. A-M's focus on PR-issue link inference also allows it to exploit the structure in PR meta-data (Section 4). Our preliminary study of "CONTRIBUTING.MD" files (Section 7.1) also found that 33 out of 43 projects specify a PR template. To address deployability, we designed A-M to seamlessly integrate into modern development practice. As Section 3.2 details, A-M suggests links when a developer closes an issue or submits a PR, when this information is pertinent, without intrusive instrumentation and its attendant privacy concerns.

9.2 Modern Development

Modern development increasingly relies on tooling that integrates Version Control, Issue Tracking, Wikis, Continuous Integration and Continuous Deployment under a single system. Notable examples are GitHub and Atlassian's JIRA. This new development paradigm poses new problems and opportunities. Kalliamvakou *et al.* [20] elucidates these, using GitHub as their archetypal example. A particular opportunity Kalliamvakou *et al.* identify is this paradigm's integration of Version Control and Issue Tracking. A-M rests on this integration. A-M also takes a step toward realising a promise they identify: interlinking developers, pull requests, issues and commits to offer a comprehensive view of the software development process.

Lack of PR-issue links is an ongoing problem in modern software development (Section 7.1). So much so, Agile practice specifies spring-cleaning an issue (a user story in Agile terminology) backlog. During backlog refinement, developers remove stale stories and reprioritise and re-estimate remaining stories. When all stories are stale, this practice discards all sprint-related artefacts — issues, feature requests, user stories, as well their links — in favour of starting the next sprint from a clean slate [1]. Projects must resort to this spring-cleaning all too often [56]. Clearly, this practice loses valuable information. The loss of documentation it entails it just one example. By automatically triaging issues, adoption of PR-issue inference tooling, like A-M, promises to reduce the need to resort to this drastic measure.

9.3 Missing Links

The *missing link problem* is the offline prediction problem of inferring missing commit-issue links given a version history and issue tracker archive. Bachman *et al.* were the first to formulate and quantify this problem [4, 5]. Their work aids developers indirectly by helping researchers and tool-smiths avoid the bias introduced by missing links that could undermine their techniques or tools. Specifically, they show that by assuming recorded links to be representative of all links, tools are biased to use code from more experienced developers, thus not learning from mistakes or bugs introduced by less experienced contributors.

Bachman *et al.* together with Apache Commons developers manually supplied missing links and published their Apache Commons corpus. Wo *et al.* are the first to attempt to automatically infer them. They propose ReLink [60], which measures the similarity of change logs and bug reports with cosine similarity on tf-idf vectors, learning a threshold for true links. Nguyen *et al.* exploited commit and issue tracker metadata in MLink [36] to improve recall over ReLink. They evaluated MLink on the Apache Commons corpus, making it the de facto standard. ReLink and MLink first consider only commit data to form an initial set of commit-issue links that

they then filter. Prechlet and Pepper [42] dispense with the initial commit-only stage, and instead consider both commit and issue data from the start. They argue this bi-directional inference is more sound. Their BFLinks proposes two link predictors (based on bug and commit IDs) and a series of filters to reduce the candidate set of links.

Prior to A-M, Li *et al.* was the state of the art. RCLinker [23] further improves recall. Section 6 details RCLinker's realisation. RCLinker relies on ChangeScribe [24] to produce textual descriptions of commits, especially those that lack commit messages. PRs have their own message and aggregate multiple commits and their messages. This fact alleviates the problem of sparse commit descriptions for A-M. Both tools would benefit from better PR summarisation and description. Liu *et al.* [27] propose a tool, based on a bidirectional RNN with a copy network, to tackle this task.

Sun *et al.* [53] used non-source files in commits for commit-issue link inference. They argue that these files are important for capturing developer intent. They use the standard heuristics, such as checking for camelCase or snake_case, to determine the relevancy of a non-source file in a commit. As is conventional, they implement these heuristics as regexes. They use the resulting set of non-source files together with the co-committed source files to compute textual features. They scan a preset and fixed list of the similarity thresholds to find the maximum F1-score where Recall is at least 0.80. The procedure raises two unanswered questions. First, how did the authors determine the threshold granularity? Second, how does training FRLink on F1-score for a task whose performance is measured in terms of F1-score avoid overfitting? They report these choices allow FRLink to improve Recall over previous work while matching or improving F1-score.

Sun *et al.* evaluate FRLink on a new corpus of GitHub projects. This corpus differs from the Apache corpus used in prior work in the conventions governing commit messages: Apache messages tend to be descriptive [2], while FRLink's GitHub sample tends to contain exact matches, because copying issue text is common practice [46]. Sun *et al.* do not investigate the effect of this differing practice on their results. They specify their corpus in sufficient detail to reconstruct it. FRLink, the tool, however, is not available. When we tried to reproduce FRLink, using its description in Sun *et al.*'s paper, we were unable to reproduce the reported results. We contacted the authors for help explaining and correcting our reproduction without response. More recently, Sun *et al.* [52] treat existing links as labels and reformulate the missing link problem into a semi-supervised problem. As Bachmann *et al.* found, existing links are biased; Sun *et al.* do not discuss how they coped with this bias. They report that their solution, PULink, outperforms FRLink on FRLink's corpus. Like FRLink, PULink is not available.

Ruan *et al.* [46] empirically studied the state of commit-issue linking on GitHub Java projects and found only 42.2% to be linked. They propose DeepLink, a neural approach to the missing links problem. DeepLink trains a text embedding for non-source artefacts and a code embedding for source artefacts using the skip-gram model [32, 33], then passes each of these embeddings separately through an LSTM layer to obtain the final vector representations. They use cosine similarity to compare the vectors, choosing the maximum similarity to represent the score of the commit-issue link. They show an improvement over FRLink in terms of F1-score, and further show that pre-processing heuristics similar to previous work, such as ReLink [60] or MLink [36], help DeepLink achieve a higher F1-score. They also spot that the FRLink corpus had commit logs and issue titles that are exact matches, introducing bias in the dataset which Ruan *et al.* handle, while FRLink does not. Since they did not evaluate DeepLink on Apache Commons or against RCLinker and DeepLink is not available, we do not know its performance relative to RCLinker.

Rath *et al.* [44] also tackle the missing link problem, from within the requirement engineering community and without referencing the line of work stemming from Bachmann *et al.*. The missing link inference work maps textural features to vector space model over unigrams. Rath *et al.* opt instead for an n-gram model. They are the first to perform feature selection, using Weka's feature auto-selection. They report promising results on a different dataset than Apache Commons. As Section 6 notes, Rath *et al.*'s work is difficult to compare with A-M, because it uses temporal features, such as a predicate that is true when a commit falls between issue's creation

and its resolution. In A-M's online setting, this predicate will sometimes be defined in terms of an event that is in the future relative to the present query.

A-M is the first tool that solves issue-PR link prediction online. Unlike previous work, we do not rely on a change summariser to produce a natural language description of source code changes; we exploit the PR description instead. As Section 3.2 details, using A-M requires only installing a lightweight Chrome plugin or a precommit script. It can smoothly integrate into a developer's workflow because it intervenes when a developer submits a commit or closes an issue — when link suggestions are pertinent. Since developers approve our suggestions and silence is merely the status quo, we must avoid distracting them with incorrect suggestions. Thus, A-M differs from previous work in valuing precision over recall.

9.4 Community Smells

Another line of work focuses on social debt [55]. This encompasses unforeseen project costs due to a suboptimal development processes. Such organisational issues can be a root cause for missing links, the process artefacts that Aide-mémoire semi-automatically maintains to reduce technical debt. To detect community structures that can lead to social debt, akin to code smells [10], Tamburri *et al.* propose community smells [54]. The impact of community smells is not limited to social debt. Indeed, Palomba *et al.* [38] show how certain community smells, such as lone wolf (committing code without regard to the community opinion) or black cloud (obfuscation by communication overload) strongly predict code smell intensity. Later, Catolino *et al.* [8] study practitioners' perception of community smells. They determine which, if any, community smells that practitioners see as relevant and formulate a field guide that documents which strategies the practitioners employ to "refactor" a community smell. Aide-mémoire, by contending with process technical debt, can help mitigate issues that may arise due to lone wolves or disengaged participants in a project. As A-M focuses on technical debt and traceability, community smells will require dedicated tooling that directly tackles them as Catolino *et al.* [8] proposed.

10 CONCLUSION AND FUTURE WORK

Related work has either treated missing links as an offline problem or has relied on invasive instrumentation across a range of developer activities. We present an alternative in the form of Aide-mémoire, the first tool to solve the problem of missing links in an online setting. Exploiting existing metadata associated with PRs, such as textual descriptions allows us to avoid reliance on more terse commit message or commit summarisation when these are absent. We also generalise across programming languages and project sizes.

We find that Aide-mémoire generalises well across a much larger range of corpora than previously considered, and outperforms a retargeted version of a state-of-the-art offline tool. Crucially, it does not require customisation of toolchains or invasive monitoring of developer activity. As Aide-mémoire incrementally improves linking within a project, it will help to reduce the noise in its own training data; we therefore expect its performance to improve over time as a project uses it.

A future version could interact with Eiffel by Ståhl *et al.* [50] by emitting the appropriately formatted JSON whenever a developer selects to record a link from a PR to an Issue. This would alleviate issues with the current systems, including A-M, where developers enter these manually in a system outside continuous integration frameworks. This manual practice suffers from inconsistent formats, forgetfulness and other human errors that automation can solve.

Acknowledgements. The authors acknowledge the use of the UCL Legion High Performance Computing Facility (Legion@UCL), and associated support services, in the completion of this work. This research is supported by the EPSRC Ref. EP/J017515/1. We would also like to thank Laurie Tratt for sharing his insight into the management of issues and pull requests within Open Source projects.

REFERENCES

[1] Agile Alliance. 2019. Agile Alliance: Backlog refinement. https://www.agilealliance.org/glossary/backlog-grooming/. Accessed: 2019-11-26.

[2] Apache. 2020. Coding and Commit Conventions. https://subversion.apache.org/docs/community-guide/conventions.html. Accessed: 2020-07-09.

[3] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. 2010. Software traceability with topic modeling. *2010 ACM/IEEE 32nd International Conference on Software Engineering* 1 (2010), 95–104. https://doi.org/10.1145/1806799.1806817

[4] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. 2010. The missing links. *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10* (2010), 97. https://doi.org/10.1145/1882291.1882308

[5] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. 2009. Fair and Balanced?: Bias in Bug-Fix Datasets. *Proc. ESEC/FSE* (2009), 121–130. https://doi.org/10.1145/1595696.1595716

[6] Markus Borg, Per Runeson, and Anders Ardö. 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering* 19, 6 (01 Dec 2014), 1565–1616. https://doi.org/10.1007/s10664-013-9255-y

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[8] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2020. Refactoring community smells in the wild: the practitioner's field manual. In *Proceedings of the acm/ieee 42nd international conference on software engineering: Software engineering in society*. 25–34.

[9] Jane Cleland-Huang, Olly Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. 2014. Software Traceability: Trends and Future Directions. *FOSE 2014: Proceedings of the on Future of Software Engineering (36th ICSE 2014)* (2014), 55–69. https://doi.org/10.1145/2593882.2593891

[10] Martin Fowler, K Beck, J Brant, W Opdyke, and D Roberts. 1999. Refactoring: Improving the Design of Existing Code. ISBN 0201485672.

[11] GitHub. 2016. GitHub Octoverse 2016. https://octoverse.github.com/. Accessed: 2017-08-07.

[12] GitHub. 2017. GitHub: Autolinked references and URLs. https://help.github.com/articles/autolinked-references-and-urls/. Accessed: 2017-08-20.

[13] Georgios Gousios and D. Spinellis. 2012. GHTorrent: Github's data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 12–21. https://doi.org/10.1109/MSR.2012.6224294

[14] Georgios Gousios and D. Spinellis. 2017. Google Cloud Public Table of GitHub Projects. https://bigquery.cloud.google.com/dataset/ghtorrent-bq:ght. Contents from 2.9M public, open source licensed repositories on GitHub; Accessed: 2017-08-10.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18. https://doi.org/10.1145/1656274.1656278

[16] Donald R. Hedeker and Robert D. Gibbons. 2006. *Longitudinal Data Analysis*. WileyInterscience.

[17] JIRA. 2017. JIRA: Link JIRA issues to Confluence pages automatically. https://www.atlassian.com/blog/confluence/link-jira-issues-to-confluence-pages-automatically. Accessed: 2017-08-20.

[18] JIRA. 2017. JIRA: Rest API Examples. https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples/. Accessed: 2021-05-14.

[19] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/ [Online; accessed 31.07.2017].

[20] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proc. 11th Work. Conf. Min. Softw. Repos. - MSR 2014*. ACM Press, New York, New York, USA, 92–101. https://doi.org/10.1145/2597073.2597074

[21] Max Kuhn and Johnson Kjell. CRC Press. *Feature Engineering and Selection: a Practical Approach for Predictive Models*. 2019.

[22] Balaji Lakshminarayanan, Daniel M Roy, and Yee Whye Teh. 2014. Mondrian forests: Efficient online random forests. In *Advances in neural information processing systems*. 3140–3148.

[23] Tien Duy B Le, Mario Linares-Vásquez, David Lo, and Denys Poshyvanyk. 2015. RCLinker: Automated Linking of Issue Reports and Commits Leveraging Rich Contextual Information. In *IEEE International Conference on Program Comprehension*, Vol. 2015-Augus. IEEE, 36–47. https://doi.org/10.1109/ICPC.2015.13

[24] Mario Linares-Vasquez, Luis Fernando Cortes-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. ChangeScribe: A Tool for Automatically Generating Commit Messages. *Proceedings - International Conference on Software Engineering* 2 (2015), 709–712. https://doi.org/10.1109/ICSE.2015.229

[25] Linux Kernel. 2020. Linux Kernel Commit Message Practice. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/submitting-patches.rst?id=bc7938deaca7f474918c41a0372a410049bd4e13#n664. Accessed: 2020-06-19.

[26] Chen Liu, Jinqiu Yang, Lin Tan, and Munawar Hafiz. 2013. *R2Fix: Automatically generating bug fixes from bug reports.* Ph. D. Dissertation. University of Waterloo. https://doi.org/10.1109/ICST.2013.24

[27] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic Generation of Pull Request Descriptions. *arXiv preprint arXiv:1909.06987* (2019).

[28] Walid Maalej and Hans-Jörg Happel. 2010. Can Development Work Describe Itself? *7th IEEE Working Conference on Mining Software Repositories (MSR 2010)* (2010), 191–200. https://doi.org/10.1109/MSR.2010.5463344

[29] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA.

[30] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Baranauskas. 2012. How Many Trees in a Random Forest? *Lecture notes in computer science* 7376 (07 2012).

[31] Qing Mi and Jacky Keung. 2016. An empirical analysis of reopened bugs based on open source projects. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering - EASE '16* (2016), 1–10. https://doi.org/10.1145/2915970.2915986

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems.* 3111–3119.

[34] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. 2005. Challenges of migrating to agile methodologies. *Commun. ACM* 48, 5 (2005), 72–78.

[35] C. Neumuller and P. Grunbacher. 2006. Automating Software Traceability in Very Small Companies: A Case Study and Lessons Learne. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06).* 145–156. https://doi.org/10.1109/ASE.2006.25

[36] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2012. Multi-layered approach for recovering links between bug reports and fixes. *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12* (2012), 1. https://doi.org/10.1145/2393596.2393671

[37] Peter O'Hearn. 2020. ICSE 2020 Keynote: Formal Reasoning and the Hacker Way. [Online: https://youtu.be/bb8BnqhY3Ss?t=2599].

[38] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2018. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE transactions on software engineering* 47, 1 (2018), 108–129.

[39] Profir-Petru Pârțachi, David R. White, and Earl T. Barr. 2020. Aide-mémoire: Accurate Issue Links at Pull Request submission. https://github.com/PPPI/a-m/. Accessed: 2020-07-13.

[40] Profir-Petru Pârțachi, David R. White, and Earl T. Barr. 2020. Datasets as pickled python objects. https://figshare.com/s/83c448eb518b3d04651f. Accessed: 2020-02-25.

[41] M.F. Porter. 1980. An algorithm for suffix stripping. , 130–137 pages. https://doi.org/10.1108/eb046814 arXiv:/dx.doi.org/10.1108/BIJ-10-2012-0068 [http:]

[42] Lutz Prechelt and Alexander Pepper. 2014. Bflinks: Reliable Bugfix Links via Bidirectional References and Tuned Heuristics. *International scholarly research notices* 2014 (2014).

[43] Shivani Rao and Avinash Kak. 2011. Retrieval from software libraries for bug localization. In *Proceeding of the 8th working conference on Mining software repositories - MSR '11.* 43. https://doi.org/10.1145/1985441.1985451

[44] Michael Rath, Jacob Rendall, Jin L. C. Guo, Jane Cleland-Huang, and Patrick Maeder. 2018. Traceability in the Wild: Automatically Augmenting Incomplete Trace Links. arXiv:arXiv:1804.02433

[45] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.* ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[46] Hang Ruan, Bihuan Chen, Xin Peng, and Wenyun Zhao. 2019. DeepLink: Recovering issue-commit links based on deep learning. *Journal of Systems and Software* 158 (2019), 110406.

[47] Scikit-learn. 2020. Recursive Feature Elimination: SciKit Implementation. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html. Accessed: 2020-06-17.

[48] Scikit-learn. 2020. Time-Series Split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html. Accessed: 2021-05-14.

[49] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken Ichi Matsumoto. 2013. *Studying re-opened bugs in open source software.* Vol. 18. 1005–1042 pages. https://doi.org/10.1007/s10664-012-9228-6

[50] Daniel Ståhl, Kristofer Hallén, and Jan Bosch. 2017. Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. *Empir. Softw. Eng.* 22, 3 (2017), 967–995. https://doi.org/10.1007/s10664-016-9457-1

[51] Eliza Strickland. 2022. Andrew Ng: Unbiggen AI. https://spectrum.ieee.org/andrew-ng-data-centric-ai. Accessed: 2022-05-26.

[52] Y. Sun, C. Chen, Q. Wang, and B. Boehm. 2017. Improving missing issue-commit link recovery using positive and unlabeled data. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).* 147–152. https://doi.org/10.1109/ASE.2017.8115627

[53] Yan Sun, Qing Wang, and Ye Yang. 2017. FRLink: Improving the recovery of missing issue-commit links by revisiting file relevance. *Information and Software Technology* 84 (2017), 33–47. https://doi.org/10.1016/j.infsof.2016.11.010
[54] Damian A Tamburri, Rick Kazman, and Hamed Fahimi. 2016. The architect's role in community shepherding. *IEEE Software* 33, 6 (2016), 70–79.
[55] Damian A Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2013. What is social debt in software engineering?. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 93–96.
[56] Laurie Tratt. 2018. Personal Communication with Laurie Tratt. Online.
[57] Michele Tufano, Gabriele Bavota, Denys Poshyvanyk, Massimiliano Di Penta, Rocco Oliveto, and Andrea De Lucia. 2016. An empirical study on developer-related factors characterizing fix-inducing commits. *Journal of Software: Evolution and Process* 26, 12 (aug 2016), 1172–1192. https://doi.org/10.1002/smr.1797 arXiv:1408.1293
[58] Ming Wen, Rongxin Wu, and Shing-chi Cheung. 2016. Locus : Locating Bugs from Software Changes. *31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)* (2016), 262–273. https://doi.org/10.1145/2970276.2970359
[59] Renjie Wu and Eamonn J. Keogh. 2020. Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. arXiv:2009.13807 [cs.LG]
[60] Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. 2011. ReLink: Recovering Links Between Bugs and Changes. *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (2011), 15–25. https://doi.org/10.1145/2025113.2025120
[61] Klaus Changsun Youm, June Ahn, and Eunseok Lee. 2017. Improved bug localization based on code change histories and bug reports. *Information and Software Technology* 82 (2017), 177–192. https://doi.org/10.1016/j.infsof.2016.11.002