ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Query Tool for Analyzing Privacy Changes of GitHub-Hosted OSS Projects

Master Thesis

Anıl Yarış

August 14, 2022

Advisors: Karel Kubicek, Aileen Nielsen, Prof. Dr. David Basin
Department of Computer Science, ETH Zürich

Abstract

With the volume of online activity growing each day, concerns about protection of online user data and user privacy have been in the limelight in recent years. Privacy laws such as *General Data Protection Regulation (GDPR)* and *California Consumer Privacy Act (CCPA)* have been introduced to address this concerns by governing what kind of information businesses are allowed to collect from their clients. However, compliance with such regulations prove problematic among software developers due to lack of knowledge, interest or benefit in implementing privacy changes in the code.

We try to analyze the extent of this problem by developing a query tool that can be used to inspect open-source software development data. Using the activity data from GitHub, an open-source development platform, we collect information such as commits, issues, pull requests, comments, user interactions, timestamps and project metadata. We then load this data into a relational database on which the queries can be executed.

In order to detect activity related with privacy compliance, we search code and text data for the existence of several keywords such as regulation names, privacy-related concepts and legal terms. We work on data with size exceeding 500 GB, therefore in order to increase code efficiency, we pre-compute labels that indicate keyword matches from all programming language and natural language text fields using an efficient string search algorithm. We then load this pre-processed data into the database of the query tool and connect it with the existing GitHub data.

We test our data by performing a wide range of queries and compare their results with the findings of a previous empirical study. We observe effects similar to the ones reported in the study regarding accumulation of privacy-related activities around dates when new statutes enter into force. We also show some novel results regarding the nature of GitHub users making privacy contributions to many projects and likeliness of compliance-related code changes being integrated into the code base. Our tool can be helpful to legislators and software researchers in identifying trends, patterns and common behaviors regarding compliance attempts of programmers.

Contents

Chapter 1

Introduction

In today's highly globalized society, user information is a very important commercial asset for organizations and companies which allows them to analyze the profiles and needs of their clients. However, due to this huge monetary potential, such data collection is not always aligned with customers' benefits and interests. The most common example for this is targeted online advertisements where age, location, gender, online browsing history and a wide range of other identifiers are all used to display advertisements that is predicted to pique the users' interest. For instance, Terkki et. al. [48] were able to show that these ads can even be theoretically used to spy on users with mobile devices.

In order to protect their clients from being the target of such ill-intended data collection and respect their anonymity and privacy, businesses are required to take into account what kind of user data they keep a record of. However if these data protection practices were presented to companies as merely advice, there would be a good chance that many clients' sensitive information would be put into jeopardy due to conflicts with monetary benefits of businesses. Therefore, users' privacy rights need to be enforced by official legislation. Many laws have been introduced throughout the world over the last 50 years that aim to govern the guidelines for how organizations should behave with regard to privacy compliance [6].

In the realm of online services and software, specialized directives such as the *ePrivacy Regulation* [12] have been proposed to regulate online data privacy. However, due to the ever-changing nature of technology, such specialized laws tend to become outdated quickly. Therefore, older and more restricted laws have been superseded by or linked to more general privacy laws such as the General Data Protection Regulation (*GDPR*) [34] and the California Consumer Privacy Act (*CCPA*) [8].

Since software developers are the people who implement data protection

in online services, they need to be aware of the latest privacy regulations. However, there seems to be disconnect between developers and the law in the sense that they lack empirical and formal knowledge on information privacy, are confused about how to interpret official legislation and sometimes are simply not aware enough of the importance of user data protection [36]. Another problem with identifying privacy problems in software is that proprietary code cannot be freely investigated and with the majority of software development in the past being done closed-source, it is difficult to track whether they have kept track with updated privacy regulations over the years.

In other fields, understanding whether the service obeys privacy laws can be done through automated compliance checking where the privacy policy of the service is often analyzed using a semantic or logical model [3, 22, 30]. However, in software development, while similar methods can be applied on the privacy policy of the *service* provided by the software, how this is achieved in the *code* is an entirely different question. This distinction is often referred to as *'code is law'* [28].

With the rise of open-source software, there is now a possibility of connecting the gap between software development and data protection by studying how programmers aim to implement data protection methods during the development process. Through analyzing the behavior of developers in attempting privacy compliance, it is possible to obtain results that would benefit both lawmakers in proposing online data protection statutes, developers whose needs can be better reflected in future legislation, and users who would be better protected by the enhancement of their online privacy.

Our contributions

Open-source development can be tracked from online repository hosting services. GitHub is a major open-source development platform which enjoys the highest popularity among developers compared to its alternatives [46]. As of August 2022, according to the search tool on GitHub's website, it contains more than 333 million repositories, 96 million users and 553 million issues.

The research question proposed above requires a system within which software development data is retrieved, processed, stored and analyzed in terms of privacy compliance. This data is not limited to only code and modifications done to the code base, but also includes natural language interactions between users (issues, pull requests, commit messages and comments) and project metadata such as programming languages, tags and milestones.

For this master's thesis we developed a query tool that stores data collected from GitHub in a relational PostgreSQL database. We make use of data

shared by previous studies that aimed to collect GitHub data and process this in order to retrieve any useful information that was missing or needed to be modified.

Checking for privacy compliance follows the code-is-law approach where we expect to see names of laws and regulations or other keywords related to privacy and data protection within code or other text fields from all GitHub entities. This is achieved by identifying these keywords to search for and generating labels that signify that a match is or is not found.

Once all data is processed, we test the functionality of the tool by performing some simple and some complicated queries. The relational nature of our database allows execution of semantically challenging queries that can look at technical areas such as code contributions, code reviews, issues and pull requests; social aspects of the development process such as users' followers and watched repositories; and metadata like programming languages of projects, issue labels, milestones, etc.

We present our findings in terms of the results of the executed queries, which are mostly in the form of filtered or grouped records, number or percentage of entities satisfying some condition and time series data. Using the results from a previous empirical study on GDPR- and CCPA-related actions from GitHub projects, we comparatively validate our tool and collected data as well as comment on the results of the comparison.

Our results show patterns of design and behavior regarding how open-source developers implement data protection in software. For example, we observe that frequency of privacy-related activities see a significant increase around the dates of privacy laws coming into effect, and merge requests mentioning such regulations are more likely to be accepted by project administrators.

As far as literature is concerned, we believe that this is the first study on privacy compliance of open-source project in this wide scale. Previous studies about data protection in open-source software have mostly explored design methods to make open-source development more aware of privacy issues [23, 26] or analyzed particular non-compliant open-source software as case studies [17]. The aim of this thesis to reveal patterns about privacy-related implementations and interaction from open-source projects rather than suggesting how they can implement data protection better. This presents a novel direction of research and we believe our query tool in the hands of law researchers can help uncover valuable insight on how to improve privacy of end-users.

Report structure

This report is further organized as follows: In Chapter 2 we provide general background information about open-source development, privacy and data protection laws, and how these two can be related to each other. In Chapter 3 we outline our options for collecting the GitHub data and justify our choice. In Chapter 4 we describe in steps how we processed the collected data to form our relational database. In Chapter 5 we evaluate our tool, report our findings and discuss the meaning and reason behind our observations. Finally, in Chapter 6, the report is concluded.

Chapter 2

Background

2.1 Open-Source Software Development

Open-source is a model of software development whose aim is to make computer programs free-to-use and free-to-access. This is maintained by free distribution and redistribution of software as a whole or in parts, public availability of program source code and freedom of modifying or deriving other work from existing software [25]. All open-source software needs to be distributed under licenses such as GNU, GPL, BSD that comply with these definitions [37].

The main driving force behind developing non-proprietary software is the gratification coming from collaboration, innovation as well as guiding the direction of the development by collective reviewing of what changes or improvements the software needs [7]. Naturally, all these aspects of open-source development cause the formation of a community of programmers which presents a self-organized, decentralized network clustered around contributed projects connected by developers' collaborative interactions [31].

Collaborative software development goes hand-in-hand with version control systems which stores the code's different editions over its development history represented by incremental modifications between each node of the version tree [45]. This version tree may contain *branches* that can represent a separate phase of development deviated from the main line or different contributors' work in progress which allows for implementation of multiple features in parallel by each contributor [47]. Such branches can be *merged* back into the main line (often called the master branch) once their development is complete and are deemed suitable by the project administrator(s) to be included in the main code base.

Usage of version control systems is not exclusive to open-source development; organizations can install a version control software in their servers

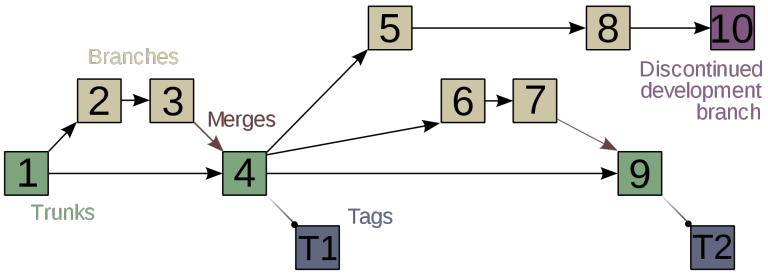

Figure 2.1: An example history tree structure [51].

to privately host proprietary code in which case the employed software developers would use the tools mostly the same way as collaborators of a open-source project would. In fact, both are driven by collaborators and developers of the project identifying issues about the program's functionality. These issues are submitted to an issue tracking system where anyone from the development team can address them.

Modern repository hosting services, in addition to storing the code base and the version history, offer its users the means to view, submit, manage and discuss these issues. Combining this with the ability to associate issues with commits and merge requests, web-based repository hosting services such as GitHub, Bitbucket and SourceForge have become almost indispensable to open-source development [15].

Such online platforms further extend the network structure by allowing users to follow repositories and other developers or add those to their favorites list. It is safe to assume that users with common interests or purposes should interact with similar users or projects, therefore such purely social interactions could also be made a part of any analysis on open-source development data, especially if the concern is to observe trends of intentions.

This social network structure in the form of an interaction graph, as well as any time series data constructed by aggregating entities based on creation timestamps, can be explored to extract trends, patterns and common behavior among collaborators [5, 10, 35]. Generalized toolsets and databases such as [16] allow research to be extended to arbitrary purposes by allowing the users to perform custom queries on some sort of pre-collected data containing the entities that are part of open-source development such as contributors, projects, code edits, discussions, etc.

2.2 Privacy and Data Protection Laws

Data privacy is the concept of a service or its provider protecting any piece of sensitive personal information belonging to the recipients of the offered

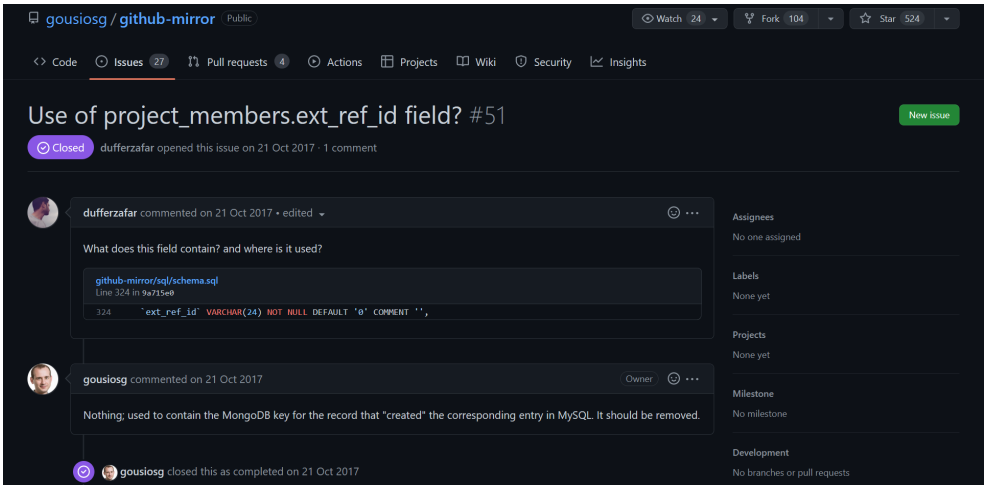

Figure 2.2: An example of a GitHub issue, along with some other tabs related to entities of a GitHub project. Taken from `gousiosg/github-mirror`.

service. What type of data can be safely made public depends on the classification of identifiers and attributes of the user and whether the release of data would allow a third-party to pigeonhole or semantically infer the identity of the user(s) to which the released data belongs [14].

There has been an increase in the number of regulations and statutes over the past decades that are concerned with enforcing data protection on a national basis [21]. Prior to such comprehensive laws it was also common for smaller scale acts to be enacted per specific fields and industries for protecting consumer data, such as healthcare and insurance [39], credit reporting [42], education [1], finance and banking [41], communications [43] and audio-visual content [38]. Another sensitive field that has seen attention is protecting the privacy of minors, especially regarding their online activities [40].

When it comes to protection of online data, the two most prominent laws in governance are General Data Protection Regulation (*GDPR*) [34] in the EU and California Consumer Privacy Act (*CCPA*) [8] with its recent amendment California Privacy Rights Act (*CPRA*) [9] in the US (although a state law, many established technology firms being based in California and abiding by its impositions makes this highly important in online privacy).

GDPR and CCPA have led many organizations to change their online privacy policies to be more protective. However, there are still issues with regard to compliance to these laws mostly due to misinterpretation or misapplication of their requirements [33, 50]. Non-corporate businesses can also struggle with implementing guidelines on privacy-related design architectures and prioritizing compliance amidst a range of functionalities that may

be worth more of their resources [29].

These observations could beg many questions including but not limited to the below:

- Do software developers have enough knowledge about privacy and data protection regulations?
- Do they choose or are required to consider the implications of these regulations when identifying the features of the program.
- How much of non-compliance results from neglect and how much of it is intentional?
- What motivates developers to implement or improve compliance?

All of these questions seem to have one thing in common: the problems they depict arise from compliance being viewed as a *choice* rather than a *necessity*. A way to combat this paradigm is to promote Privacy by Design (*PbD*) schemes which, in general engineering context, aim to make privacy-proactive thinking embedded into the design process by default [11]. There have been many attempts at making the adoption of PbD principles for the developers easier and more efficient by creating tools and design patterns that comply with GDPR, CCPA and other privacy laws [4, 27, 44].

It might indeed be possible for software companies concerned with privacy compliance to have PbD as a company policy. In such an organization developers could be educated about data protection laws and related concepts in general, code reviews and quality assurance could involve checking for whether PbD principles were adhered to, etc. This would indeed cause a rise in the consciousness of the employed developers towards keeping user privacy in mind while coding.

However, expecting such policies to be a part of an open-source project would be less realistic considering its decentralized nature and diversity of the collaborators' backgrounds. Therefore, implementing compliance with data protection laws in open-source development is primarily up to the objectives of contributors.

Fortunately, such intentions shown by particular programmers or project administrators are not implicit and can be extracted from commit messages, issue titles and other text or code fields of the repository. For example, if someone is making an edit in the code for compliance with a certain regulation, it would be expected that somewhere in the code patch or the commit message there would be a mention of the regulation name or at least some privacy-related word in the most general case.

The public nature of open-source development offers many ways of creating software that respects data protection practices. Firstly, when a user is

concerned with the privacy compliance of an open-source project, they can create issues requesting it to be implemented if it has not been already, or can directly make the necessary modifications themselves if they have the technical and legal knowledge and submit their changes for inclusion in the main branch. Looking for such patterns in the development process using the analysis methods described in the previous section could provide insight into the awareness of open-source community for privacy features of the projects they are contributing to.

Secondly, by analyzing code contributions or users' interaction with projects or each other in privacy-related contexts, it is possible to observe social, behavioral or software design patterns for how open-source developers attempt compliance with regulation. Such research can provide insight to legal scholars about the dynamics of open-source community and could even guide future laws and regulations on how to govern open-source development differently.

Chapter 3

Data Collection

As our choice of open-source development platform, our query tool needs to have access to some part of relevant GitHub data and we chose to collect this data as a whole locally in advance. This allows us to pre-process the data during its collection in order to obtain useful aggregate information which would extend our querying capabilities both in terms of runtime and what we can query for.

Such an offline approach has the drawback of the collected data requiring to be updated by running the collection process incrementally on the older version. If the primary aim is offering the tool as a service that can be used any time to get most up-to-date results, a different approach such as storing no data but rather fetching data at the time of query would be necessary.

However, online methods of accessing such a large scale data suffers from bottlenecks such as requiring a very high-speed internet connection or bandwidth/quota limitations imposed by the server that distributes the data. Thus, such an approach is rarely worth its infeasibilities.

For the scope of this project, demonstrating how open-source projects attempt compliance for data protection laws has been set as the main goal, therefore if data was to be collected in advance to a comprehensive level with some acceptable degree of liberty on completeness and up-to-dateness, this demonstration would still be considered successful.

It is worth mentioning again that by our labelling of offline and online methods, the requirement of internet usage or lack thereof is not part of the definition, but rather the query working on pre-collected data versus no pre-collection is. A perfectly viable offline method could be to implement a scraper by retrieving the data using an application programming interface (API).

3.1 Using the GitHub API

GitHub has an API service[1] for accessing public data as well as some private information through authorized requests. Being based on the representational state transfer *(REST)* architecture [18], it operates through the client sending formatted HTTP requests to the server which returns the data in a documented schema. In the case of GitHub, separate *endpoints* for each GitHub entity exist and have different request and response format. Using this API, the first option we considered for data collection was to go through repositories, users, etc. and retrieve the types of data that would be relevant to us.

An initial demo attempt which can be found in Appendix B.1 was implemented using `PyGithub`[2], a Python library that serves as a wrapper around the HTTP requests necessary to interact with the REST API, that checks the current day's trending GitHub repositories and retrieves their commits (message, comments, modified files, diff patches), issues (title, body, comments) and other repository metadata (used programming languages, GitHub topics, tags, contributing users, list of branches), all of which can then be queried for keywords related to data protection and privacy laws. The trending repository list was obtained using another Python library named `gtrending`[3].

This approach proved to be not feasible to be scaled up since the GitHub API has a rate limit of 5,000 requests per hour[4], which was only enough to completely retrieve one repository among that particular day's trending repositories. Using multiple authorization tokens to circumvent this rate limitation is also prohibited by GitHub's terms of service[5], therefore we decided to contact GitHub to ask them about our options.

The maximal increase in the rate limit that was offered to us was 12,500 per hour, which was still too low considering the scale of the data to be collected. As an alternative, they pointed us to previous projects that aimed to collect GitHub data suggesting that we could use a similar approach or use their data directly.

[1] `https://docs.github.com/en/rest`

[2] `https://github.com/PyGithub/PyGithub`

[3] `https://github.com/hedyhli/gtrending`

[4] `https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting`

[5] `https://docs.github.com/en/site-policy/github-terms/github-terms-of-service#h-api-terms`

3.2 Looking at Similar Projects

There were three main candidate projects that we considered making use of, namely GH Archive[6], GHTorrent[7] and Google's own dataset on BigQuery[8].

GH Archive offers its data in the form of event archives, which represent a collection of actions such as "new commits and fork events, opening new tickets, commenting, and adding members to a project". These are available to download as hourly archives in its finest divided form, although it is possible to fetch them for broader ranges of dates and times at once. Each archive file is essentially a JSON file listing the events of the requested time range with all of their subfields as returned by the GitHub Events API[9].

GHTorrent was created by Georgios Gousios of TU Delft [20] and differs from GH Archive by offering fully constructed GitHub entities (commits, repositories, issues, users, etc.), rather than only the event data. The data also comes stored in a relational manner, being offered as MySQL table dumps. Additionally, it is possible to retrieve raw API responses that were used to create the MySQL tables which are downloadable as daily dumps of MongoDB tables.

Google's own BigQuery dataset that contains all public GitHub data was released in June 2016 and it also offers the data in a relational manner that can be queried on its BigQuery service. It is also worth noting that GH Archive's data is also accessible through BigQuery, however, they are stored there in the same event archive format.

Using BigQuery for either of the two datasets would allow us to skip the data collection part altogether, which would be quite advantageous. However, BigQuery only allows 1 TB of monthly processed query data on its free-to-use tier, which would be easily exceeded on queries made on data belonging to a wide date range (GH Archive's BigQuery data is 16.5 TB in size, Google's public dataset is 3.9 TB). Therefore, despite the advantage of no data collection requirement, we decided it would be infeasible to work with BigQuery and considered the other options.

Comparing GH Archive to GHTorrent, we had to make our choice based on a trade-off between two differing aspects of these projects. Since GH Archive stores only GitHub events, it is always kept up-to-date whenever new events are added to the archive. However, in order to construct a queryable dataset containing individual repositories, commits, issues, etc. on their own, one

[6] `https://www.gharchive.org/`

[7] `https://ghtorrent.org/`

[8] `https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code`

[9] https://docs.github.com/en/rest/activity/events

would need to look back at all events related to the entity in question, accumulate the information presented by the events over time and build up the most recent version of that entity.

In contrast, GHTorrent stands on the other end of 'up-to-dateness versus post-processing' trade-off, where the downloadable MySQL dumps represent a frozen instance of the entire GitHub space with everything there to be readily queried. The up-to-dateness of GHTorrent would not be a problem if it was actively maintained like GH Archive is; however, the most recent MySQL dump was released on March 7, 2021[10] and the daily MongoDB dumps have not been updated since May 25, 2021[11].

Despite this limitation, we decided to go with GHTorrent since the database already having been constructed seemed more attractive and more in-line with what we have envisioned beforehand, along with the fact that the dates the laws to be queried for came into effect were still contained within the time range of GHTorrent's data. It is also technically possible in the future to manually update GHTorrent's database by either using the API to fetch the missing dates' data or retrieving such data from another actively maintained source like GH Archive or BigQuery.

[10] `http://ghtorrent-downloads.ewi.tudelft.nl/mysql/`
[11] `http://ghtorrent-downloads.ewi.tudelft.nl/mongo-daily/`

Chapter 4

Database Construction

After GitHub data is obtained, the implementation of the query tool can begin by the database construction procedure. The collected data coming from GHTorrent needs to be loaded into some sort of database system on which the queries can be executed. However, using the data in its initially obtained form would restrict our querying capabilities to only the information contained by the data fields. Therefore, after GHTorrent data is loaded, the database is examined to reveal any piece of missing information. While retrieving the missing data from a different source, some processing can be applied in order to generate aggregate information that could serve as pre-computation to future queries which would allow to skip repetitive steps. Lastly, the missing content and results of pre-processing should be integrated into the database and connected with the original GHTorrent data.

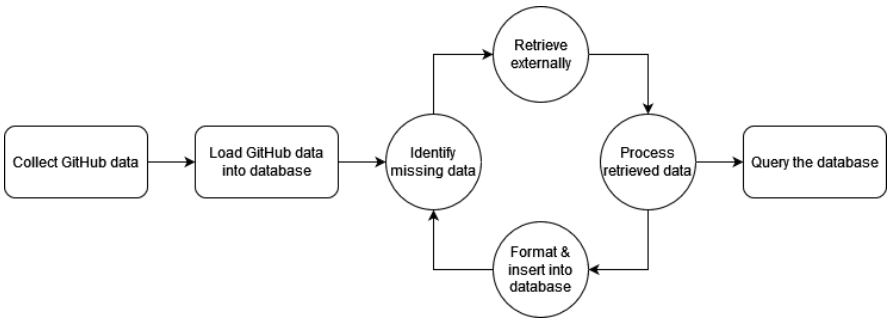

Figure 4.1: Diagram representing the development stept of the query tool.

4.1 Setting Up GHTorrent

As mentioned in the previous chapter, GHTorrent is presented as a MySQL dump containing the GitHub entities in relational schemes shown in Fig-

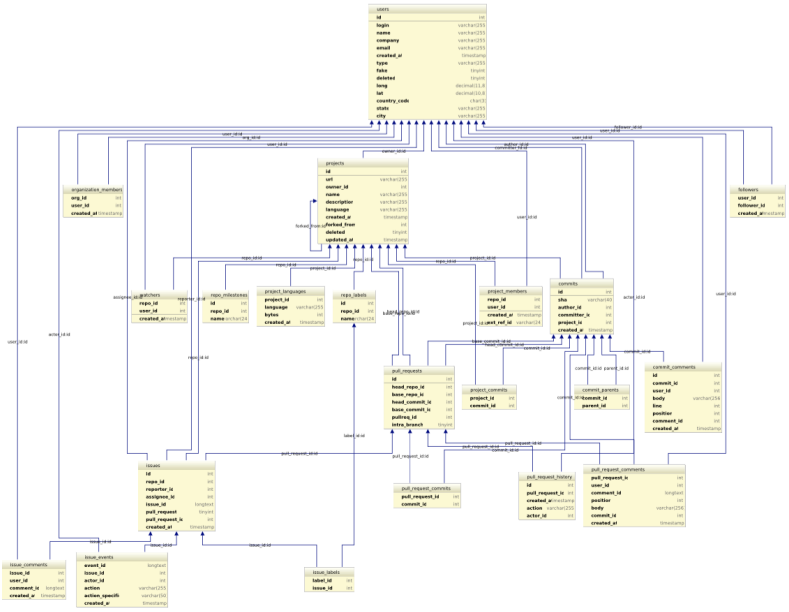

Figure 4.2: Entities and their relations in the GHTorrent database scheme.

ure 4.2. This dump can be loaded into a local environment running an SQL-compatible database management system.

Considering the size of the data (156.7 GB for the most recent compressed archive, 506 GB after decompressing, 770 GB for the size of the database directory after loading the data), the machine that would host the database server would need to be a remote one with large storage and a very high-speed internet connection. For these purposes, the most accessible choice available to us was the ETH Euler cluster[1].

GHTorrent offers scripts for loading the data into MySQL and PostgeSQL. Among these two options only PostgreSQL (version 13.2) was natively available in the Euler cluster, therefore the database management system of choice was PostgreSQL.

Upon loading the data and performing some simple queries which can be found on Appendix C.3, it was evident that aside from pull request comments, none of the text bodies that the keyword querying could be performed on were available as fields in the database tables. GHTorrent also offers the raw API responses that were used to construct the PostgreSQL data in the form of daily dumps of MongoDB files and the required text fields such as commit patches, issue comments, etc. as well as any other field we would observe to be missing in the PostgreSQL dump could be found there. Unlike PostgreSQL, an existing installation of MongoDB was

[1] `https://scicomp.ethz.ch/wiki/Getting_started_with_clusters`

not available within the Euler cluster, therefore a binary executable for the server (version 5.0.7) was set up.

In order to transfer the missing data from MongoDB into PostgreSQL, some programmatic solution that connects these two databases needed to be implemented. This was again done in Python, using the libraries `Psycopg2`[2] for PostgreSQL and `PyMongo`[3] for MongoDB. The initial idea was to extend the PostgreSQL tables with new columns representing the missing text fields, fetching that data from the MongoDB dumps with the `PyMongo` connection and inserting them into the newly created fields of the corresponding PostgreSQL records using `Psycopg2`.

4.2 Batch Processing

This section describes the procedure of iteratively processing *batches* of MongoDB dumps and how this procedure evolved step-by-step along the development of our query tool.

4.2.1 Preparation & data structuring

Before starting to collect data from MongoDB over the whole available collection of daily dumps, it needed to be known what exact fields from each table should be extracted. This was done by importing the earliest daily dump — the reason being its size being the smallest which would reduce download and import times for quick testing — into MongoDB using the `mongorestore` utility that comes with the downloaded binary and comparing the data with the corresponding PostgreSQL tables side-by-side. The full list of fields that were extracted from MongoDB can be found in Appendix B.2.

Since the format of data coming from MongoDB is different from how the PostgreSQL tables are structured, one step before trying to transfer the data between the two was to fully understand the data types of the fields of extracted MongoDB records. This was done by accumulating the records extracted from processing multiple daily dumps into a separate MongoDB database. Since MongoDB records are in a JSON-like form, which fields are present and their data types can change between records of the same type. For example, a record representing a pull request which has been merged would have fields representing the user confirming the merge and when the merge was done, whereas for a pull request which was not merged these fields would not be present in its JSON-like structure. This inspection also helps with identifying which fields can have null values.

[2] `https://github.com/psycopg/psycopg2`
[3] `https://github.com/mongodb/mongo-python-driver`

By iterating over the accumulated records, a schema-like nested structure (Figure 4.3) was obtained for each table representing the list of different fields it could have along with what data types these fields can be. Using this information it was possible to write functions for each table that map data correctly between MongoDB and PosgreSQL by making the necessary type conversions and handling cases with missing fields and null values.

Figure 4.3: An example of the field type nested structure obtained from the `repos` tables of MongoDB dumps.

The re-formatted data was attempted to be inserted into the original GHTorrent tables through `UPDATE` queries which would set the missing text body fields. However, this approach was observed to run slowly as it took multiple hours to process only the first few days' worth of MongoDB dumps, implying a lower-bound total runtime estimate of around 36.5 days. This is possibly due to each query going through the whole table to check if each record satisfies the conditions specified in the `UPDATE` query and this scan needs to happen with every query. Considering the size of these dumps grow as they are from more recent dates — meaning it would take even longer to process them compared to dumps from earlier days — a change in the insertion method was necessary.

Upon research on efficient methods of inserting large amounts of data into PostgreSQL, one common solution that came up was using separate tables for data insertion instead of updating existing tables. This was applied to our case by creating new tables that represent the missing data that was fetched from MongoDB and changing the previously written `UPDATE` queries on pre-existing tables into `INSERT` queries on these new tables. Then, there seemed to be three options for how to relate these new tables to the original GHTorrent tables:

- An extra ID field would be added to the new tables that points to the ID of the original GHTorrent record of the entity in question
- The original tables would be extended with the missing fields all at once with a single `UPDATE` query done directly on PostgreSQL command line shell
- No pre-linkage would be performed, any query that would be performed on the original tables would need to include the new tables as `JOIN`s or part of the `FROM` clause in the query statement

Taking the third approach, if the queries are constructed as multiple subquery statements with pairwise `JOIN`s rather than one very large statement that leaps straight to the desired result but contains too many tables in the `FROM` clause, we observed that very complicated queries could be executed within 10-30 minutes. Since this was within reasonable limits for us, we decided to go with the no pre-linkage option.

Aside from this, there was still room for improvement regarding insertion and data processing. This, along with other modifications to how the MongoDB data was processed different from this initial version, will be discussed in the next section.

4.2.2 Further performance enhancements

The new tables were named in the format `mongo_<entity>` where `<entity>` represents the name of the corresponding table in GHTorrent's database. For example, for the original `issue_comments` table, the new table that holds the missing issue comment data fetched from MongoDB is called `mongo_issue_comments`.

After setting up the schemas for the new tables according to the field and type structure obtained previously, it came to our attention that the approach of storing new data in new tables could allow us to optimize many things at once by doing a further modification. We decided that instead of storing the data in PostgreSQL right away they can be written into CSV files, one for each new table. This would help us in two different ways:

- Running the `INSERT` query after processing each daily dump or alternatively a batch of daily dumps is still sub-optimal due to the multiplied overhead caused by connecting to the database each time through Python and the inner mechanics of running the query. This is supported by the PostgreSQL documentation about populating databases[4]. From our observations and online research about the matter, running a single query to insert, update, load, etc. all the data at once seemed the most time-efficient way of working with large databases. Writing the data into CSV files and loading them into the new tables with a single `COPY...FROM` query allows us to reach this efficiency.
- Since the goal of this project is also to distribute this data to other people that want to run their own queries, we would need to prepare CSV dumps of the newly fetched data anyway for other users to download and load into their own systems. This way, the dumps would be already available once the data collection is complete.

Optimizing keyword matching

Another improvement that was implemented afterwards is related to how the keyword queries would be performed. PostgreSQL has pattern matching capabilities using regular expressions[5] but it is likely that it would take a long time if this matching was to be performed on-the-go when the queries are executed. Avoiding this requires generation of boolean keyword match labels through pre-processing each body of text. This was first intended to be done only on commit patches in order to omit storing them wholly to reduce processing times and storage but later extended to include all other text fields due to its potential help in simplifying and speeding up queries.

Obviously, the main caveat of this method is that it fixes the list of queryable keywords in advance and a user who needs to query for a keyword not in our pre-processing list needs to resort back to native PostgreSQL pattern matching methods. However, even accounting for that case, it is still a benefit to generate these labels for the keywords that would be queried the most. In the worst case, if other important keywords come to mind after data collection, a new run of batch processing could be done with only these new keywords, and then the new labels could be added to the already existing ones either in the CSVs or in the database tables.

The runtime cost of this is the time it takes to process every text body, but this processing is very likely to be faster if implemented programmatically compared to within PostgreSQL since it is possible to find faster, more flexible and more optimal methods. It is also trivial to say that the results of

[4] `https://www.postgresql.org/docs/current/populate.html`
[5] `https://www.postgresql.org/docs/current/functions-matching.html`

this one-time processing can be used by multiple queries whereas if no such early processing is performed, each query would need to execute the same keyword matching over and over again.

We came up with a comprehensive list of keywords including names of laws, acts and other concepts related to privacy and data protection which can be found in Figure 4.4. Then, while collecting MongoDB data with the `Pymongo` connection, each record would go through a keyword matching function and a label for each keyword would be appended to the record structure (a dictionary in the case of Python). The value of the label would be set to `true` if the keyword occurs in the processed text and `false` if it does not.

```
keywords = ["gdpr", "rgpd", "dsgvo", "ccpa",
            "cpra", "privacy", "data_protection",
            "compliance", "legal", "consent",
            "law", "statute", "personal_data",
            "comply", "hipaa", "fcra", "ferpa",
            "glba", "ecpa", "coppa", "vppa"]
```

Figure 4.4: List of keywords used by the keyword matching script to generate labels.

For a better explanation, consider a hypothetical issue comment like *'We have already implemented GDPR compliance, but what about CCPA?'*, then among the list of keywords the labels for `gdpr`, `ccpa` and `compliance` would be set to `true` and the rest would be `false`.

Commit patches are treated differently in the way that they have two labels for each keyword, namely `<keyword>_added` and `<keyword>_removed`, corresponding to whether the keyword occurs in an added or removed line in the commit patch. An example to how the commits are encoded internally by Git can be found in Figure 4.5.

Note that it can be the case that both of these labels for one keyword could have a `true` value within a particular commit if the keyword occurs in both an added line and a removed line in the commit patch (either in the same patch or across separate patches belonging to different files from the same commit).

In order to reduce the amount of these labels, for shorter text fields (issue titles, commit filenames, etc.) that do not carry as much information compared to larger bodies of text, labels coming from all keywords were aggregated into one single flag. As an example, a `true` value in the `title_flag` field of the new `issues` table indicates whether *any* of the listed keywords occur in the title of the issue. While a good way of simplifying the schemes of the new tables, this turned out to be an oversight which will be discussed in Section 4.2.3.

As for how keywords are matched, our first implementation involved us-

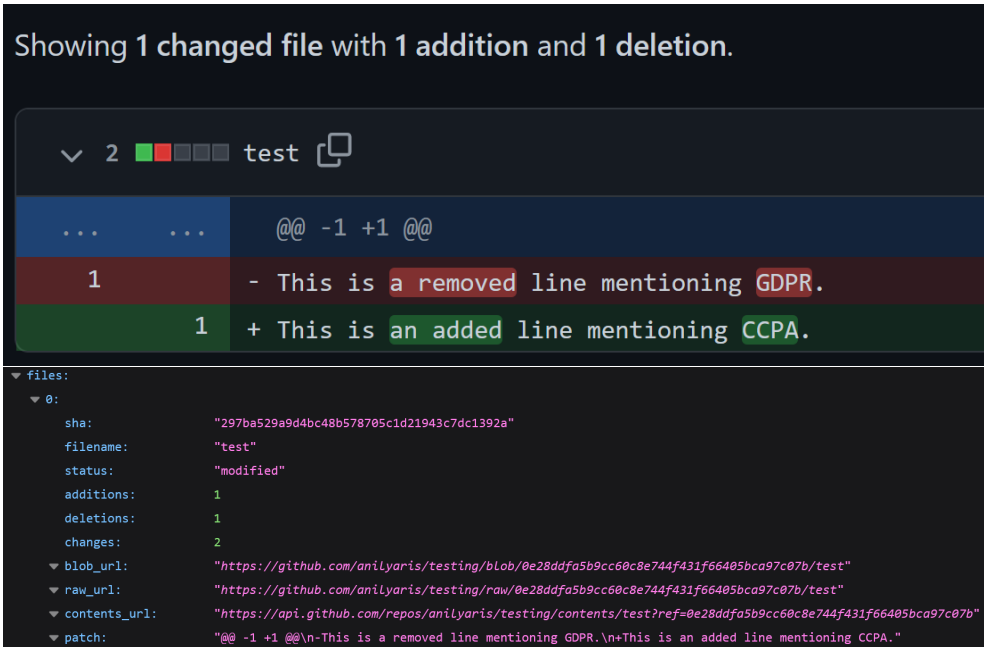

Figure 4.5: Top: A commit patch viewed on GitHub website. Bottom: Same commit accessed through GitHub API. In the encoding of patches (+) and (-) at the beginning of a line represent addition and removal, respectively. For this commit, keyword matching should set the `gdpr_removed` and `ccpa_added` labels to `true`.

ing Python's standard `re` module for regular expressions. Keywords were searched one-by-one case insensitively and the keyword label values were set to whether the regular expression reported a match. We accepted a match if the keyword exists anywhere in the text regardless of where it appears within a word in order to not miss situations like a variable in some code named `button_privacyPopup`.

This freedom results in false positives where the match does not have a contextual relation with the keyword (e.g. the pull request #314 in the repository `PagerDuty/terraform-provider-pagerduty` reports a match for CCPA due to the mention of some `AccPager`) but this was the only way that we would have absolutely no false negatives since not missing matches is more important. Moreover, results of queries can be examined to filter only the desired types of keyword matches (exact words, case sensitive, etc.), therefore the over-inclusive strategy is still a good precursor of discarding irrelevant content.

Using regular expressions instead of Python's native '`in`' operator on strings for substring search was necessary due to the how keywords including multiple words were chosen to be handled. We decided that for such cases, we consider a match if there are no more than one character between each word. To make it clear, considering the keyword `data protection`, we wanted

to match for variable names in the code such as `enableDataProtection` or `data_protection_consent`, which was the reasoning behind our choice. Details of the regular expression implementation can be found in Appendix B.3.

After running the batch processing script with this keyword matching strategy for a while, it was obvious that there were issues with performance. Commit patches were especially taking too long due to their size being larger than most other text fields in consideration. Thus, we had to look for alternative implementations to replace naive regular expression matching. Three other methods were found that could increase processing speed: one still using regular expressions and the other two entirely different string search algorithms/data structures. We compared these three with our initial implementation by benchmarking the time it takes to process the same data using each method.

In our first matching approach with regular expressions we were using a separate pattern for each keyword and iterating over the keywords to set the labels. The modification we made to this was using a single pattern containing all of the keywords, performing the regular expression match once and iterating over the found matches to see which keywords have been found. We called this approach 'combined regex' in contrast to our initial 'separate regex' method.

Another strategy we tried was making use of the *trie* data structure, also called a *prefix tree* [19]. This data structure is constructed from a given set of strings such that each node serves as the common prefix for all the connected nodes in deeper levels. The initial strings are located at the leaf nodes, therefore one could search for these strings by depth-first traversing the trie starting from the root and following the prefix connections. For strings that do not appear in the set of initial strings, searching for it in the constructed trie would terminate at some point where none of the children nodes point to a matching prefix.

In particular, a modified version of the trie called *radix tree* removes the parent node's content from the prefix of the children nodes as well as merges any node with its child if that child is its only one. This results in a space- and time-optimized implementation of the structure [32].

Using the `pygtrie`[6] library, the trie was constructed once from the keyword list that we wanted to match for; then the same trie can be used for all keyword search instances, preventing the overhead from the regular expression approach to construct the regular expression pattern for each keyword search. The library by itself does not offer a method for string matching so the trie traversal was implemented manually with helper methods from the

[6] `https://github.com/google/pygtrie`

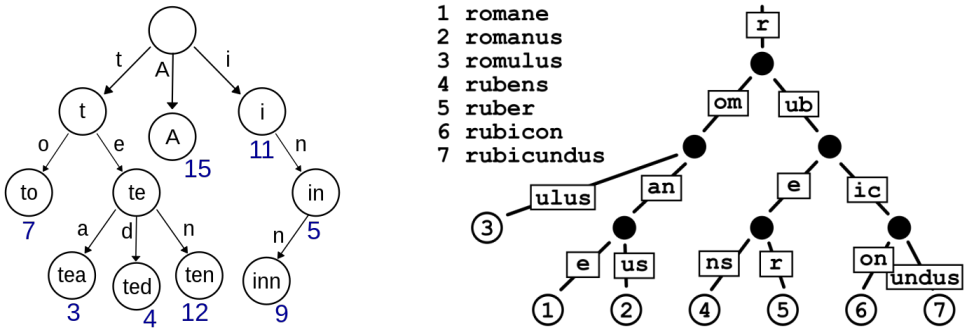

Figure 4.6: Left: example of a trie constructed from strings 'A', 'to', 'tea', 'ted', 'ten', 'i', 'in', and 'inn' [24]. Right: example of a radix tree [52].

library that allows access to the inner node structure. This was also necessary to account for keywords with multiple words since there needs to be a separate handling of such cases in the traversal due to our decision to allow zero or one character between each word for considering a match.

The third keyword matching option came as a result of searching for commonly used string search algorithms that could be applicable to our particular case, i.e. pre-processing the patterns to be searched to look through unprocessed text. We wanted an algorithm that supports multiple keywords and no pre-processing of texts on which the search is performed.

Watson [49] compared two multiple-keyword algorithms and their variants in his complexity analysis, namely Commentz-Walter [13] and Aho-Corasick [2] algorithms. These speed of these algorithms are shown to be affected by the keyword set size and minimum keyword length. While Commentz-Walter is faster on average, Watson shows that for both our set size (21) and minimum length (3 for 'law') Aho-Corasick outperforms Commentz-Walter. Therefore, we explored the Aho-Corasick algorithm as our third keyword matching method.

Aho-Corasick algorithm essentially extends the trie structure with further inner links between nodes for a more optimized search. As an added benefit, the `pyahocorasick`[7] library that we picked was implemented in C which could offer faster execution times compared to `pygtrie` library which was implemented in native Python.

After implementing these alternative keyword matching strategies, we benchmarked the separate regex, combined regex, trie and Aho-Corasick methods by measuring their processing runtime on the same 100,000 commits coming from MongoDB. Based on the benchmark results in Table 4.1 we decided to use the Aho-Corasick algorithm as our keyword matching implementation.

[7] `https://github.com/WojciechMula/pyahocorasick`

Separate regex	Combined regex	Trie	Aho-Corasick
656.5	475.8	1413.4	164.2

Table 4.1: Runtimes of different keyword matching methods in seconds.

Parallel processing

Despite being able to speed up the keyword search itself, we still wanted to improve the overall runtime in order to reduce our estimations on how long it would take to process all daily dump files. The way of achieving this was making use of multi-process programming.

This first came up as an idea to speed up processing of commits as they were taking longer compared to other tables. We divided the keyword matching process among multiple workers with the help of Python's standard `multiprocessing` library and gained some improvement in speed. Then, we realized this approach could easily be extended to processing each table in parallel such that the runtime of processing would be reduced to $max(t_{commits}, t_{issues}, t_{projects}, ...)$ from $t_{commits} + t_{issues} + t_{projects} + ...$ Since each table was being dumped to a different CSV file, there were no concurrent access violations with this approach.

Later, we implemented another strategy in a similar manner where the download and extraction of a daily dump archive file can take place while the previous daily dump was still being processed. The main process waits for the processing to be done, then loads the already extracted files of the next day's dump into MongoDB.

The final use of parallel execution was support for the script to be executed twice, one execution starting from the earliest available daily dump and moving forwards in processing and the other starting from the most recent dump and moving backwards. This effectively cuts the runtime in half, however, unlike the previous modifications, the two executions need to work on the same CSV files concurrently. The safety of this was assured by the `filelock`[8] library which provides tools for locking a file for one process while another is accessing it.

With all the performance improvements, we decided to get an estimate on the total runtime to process all MongoDB dumps. Since the size of the dump files grow as they are from more recent dates, in order to get an accurate estimate we measured two runtimes for processing from two ends of the date range similar to the bidirectional processing described above. Calculating the average time it takes to process one dump from both ends, averaging these two figures and multiplying this with the number of dumps gave us an estimate of $\sim$11 days which was very manageable for us.

[8] `https://github.com/tox-dev/py-filelock`

4.2.3 Observations on collected data & applied fixes

After the first round of batch processing was complete, the produced CSVs were loaded into the database with the `COPY...FROM` command. In order to test for any errors or general issues with the tables a few rudimentary queries were tried. The full SQL statement for these queries and their results can be found on Appendix C.4.

First such query was to group issues that a mention of CCPA was recorded by their creation month. This query involved the new `mongo_issues` table which has the keyword match labels. A very early alert while writing the query was that these labels were generated from only the *body* of the issues. The flag for whether the *title* contains CCPA was an aggregated one, mentioned previously in Section 4.2.2. It was evident that the flag aggregation strategy for some text fields would prevent querying for specific keywords, an issue that was overlooked at the time of its implementation.

Second test query was to do the same grouping on commits instead of issues. The flag aggregation problem came up again regarding the commit message this time, but we were able to notice another different issue after the execution of the query. There were commits with impossible creation timestamps that predate the foundation of Git.

Upon some research, we found out that this problem seems to arise when the local clock of the committer's computer is set to a different date and Git sets the commit timestamp using the system clock. For such commits, even when the commit is inspected either directly in the GitHub website or by querying it with the GitHub API, the creation date is still mislabeled which implies that there is no way of retrieving the correct timestamp. An extreme example to this problem can be found on Figure 4.7.

Despite this, we still wanted to re-obtain the creation dates for all entities that we fetched in the previous round of batch processing just in case GHTorrent itself also has mislabeled dates due to a bug or a wrong implementation. This, along with the need to extend the aggregated labels to include separate labels for all keywords, called for a further round of batch processing.

Regarding implementation of this new round, few needed to be changed. The list of fields to be retrieved from MongoDB was modified, label aggregation was removed and the names of the CSV files to output data were appended with the `_extended` suffix. After this round was complete, the CSVs were imported into newly created tables whose names also had the same prefix as the CSV files, e.g. creation dates and label expansions of the `mongo_commits` table would be recorded in the `mongo_commits_extended` table.

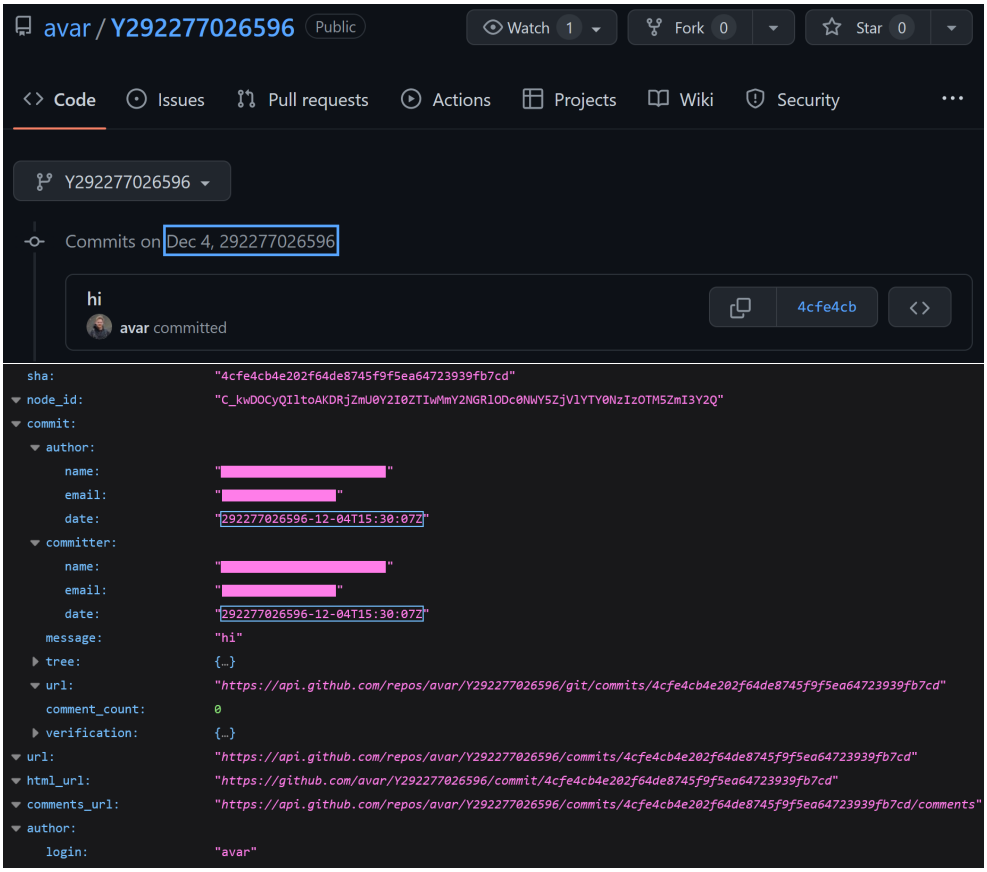

Figure 4.7: Top: Example of a commit with an impossible date inspected on GitHub. Bottom: API response data (click for access) of the same commit. Taken from `avar/Y292277026596`.

Using the extended tables, the queries discussed above were re-formatted and executed. The results show that more issues were found to mention CCPA as the updated query is now more inclusive, but the commit creation dates coming from the `mongo_commits_extended` table had the same problem as the original `commits` table as seen in Appendix C.5. After checking multiple such commits again using the website and the API, we concluded that there is no way to get around this problem and given that the proportion of commits with absurd timestamps is only 0.5% as seen in Code 4.1 we assumed these would not affect our results significantly.

```
SELECT COUNT(*) FROM commits
WHERE created_at < '2005-01-01' -- prior to Git's release year
OR created_at >= '2023-01-01';

  count
----------
 10067989
(1 row)
```

```
SELECT COUNT(*) FROM commits;
   count
------------
 2046785487
(1 row)
```

Code 4.1: Query for number of commits with inaccurate date labels versus number of all commits in the GHTorrent `commits` table.

4.2.4 Extracting raw text data

During the execution of the previous runs, we came up with some semantically complex queries to see how difficult it is to write the SQL statements for such queries as well as to test the upper bounds of query execution times. One such query involved querying for HTML files, which could be useful to check which websites hosting their code on GitHub attempt privacy compliance.

For querying file information, there was no capability within any of the original GHTorrent tables, first MongoDB tables and the extended tables. Parsing the file contents from commits to see whether a file has HTML code was a difficult option, so we needed an alternative. We decided to make use of the file extensions for such types of queries, but the file name data was also missing from the current set of tables.

Meanwhile, we also noticed that including full text data in our dataset for any field containing natural language text is necessary in case we need to analyze matches further, for example to possibly filter out false positives. Text data could also help with other researchers who want to perform semantic analysis on such fields to obtain meaningful results related or unrelated to data protection and privacy. We had originally avoided storing these fields in full due to its storage cost, but for these purposes we decided that it is worth keeping.

These two new data were already pointing to the necessity of a third batch processing run so we started looking for any other fields to retrieve from MongoDB dumps that would increase our querying capabilities. The one thing we thought that could be of help is to get the repository name and repository owner for commits. This data is already available in the GHTorrent `commits` table but can only be accessed by running the commit query in connection with the `projects` and `users` tables which was making spot checking commits on the website or API a convoluted process. This could also theoretically help speed up some queries involving commit-repository relations by skipping the just mentioned table crossing problem but only if such a query does not need to make use of any other table involving original GHTorrent tables since there are mismatches between GHTorrent and

MongoDB data (same repositories / users being named differently due to name changes, data from either source missing from the other altogether).

Before the third round of processing, the list of relevant fields to be retrieved was updated in the batch processing script and the creation of a new `commit_filenames` table was implemented that would store commit hash-file name pairs for each file affected by a commit. In addition, the two-directional (start-to-end, end-to-start) support for parallel processing was doubled such that four executions would run from start to mid-point, from mid-point to start, from mid-point to end and from end to mid-point, respectively. Besides `commit_filenames`, new tables with the `_text` suffix (labelled after retrieval of text fields) were created for the output CSV files to be imported into, similar to the new tables of round 2.

While this third round was running, it was evident that we would likely not be able to import these new tables into PostgreSQL. This is due to the size of the text data being too large to fit into the storage quota of the Euler cluster. At around the halfway point of round 3, we noticed that the total size of the output CSV tables has already surpassed 900 gigabytes, which would imply around 2 TB of data after completion. The size of the PostgreSQL directory containing original GHTorrent data and the tables from the previous two batch processing runs was around the 1.6 TB mark, therefore given our storage limit of 2.7 TB on Euler the import would probably not succeed.

CSV files from this run, along with the files from the previous rounds, are stored on a separate disk as highly compressed archives. One problem is that access to this remote disk is controlled via SSH/SCP using a key, which means public access is not possible at the time of writing. Once we have the means of hosting this large amount of data publicly, the repository[9] of the project will be updated with the according instructions.

[9] `https://github.com/anilyaris/codelaw`

Chapter 5

Evaluation

In this chapter, we discuss the results of our analysis that we obtained using our GitHub querying tool. Some of these results are novel based on some intricate queries that we have performed, and the rest will be presented in a comparative manner.

This comparison will be based on the findings of an unpublished preliminary study by Nielsen on open-source privacy compliance. Nielsen uses the GitHub API to collect data which might be on a more limited scale than ours due to API quotas, therefore while comparing the results the focus will be on proportions, ratios or trends rather than exact amounts, which will be clearer later.

5.1 General Statistics & Novel Findings

In order to help convey the scale of the data and lay some foundations, some general statistics about the data can be presented first. It is worth checking the number of issues that mention privacy regulation names within the issue title, body or the comments made on the issue. This is done using the keyword match labels that we have pre-processed.

Among the 123.4 million issues from the `mongo_issues` table, the number of issues matching the pre-processed privacy regulation names are displayed in Table 5.1. Note that pull requests on GitHub are also issues, therefore the numbers include issues that are and are not pull requests altogether.

The first thing to notice is the number of issues reporting a match for CPRA, which seems to be unreasonably high. Upon manual inspection of a particular match (pull request #9 from `mff-uk/open-dataset-inspector`), this pull request seems to be generated by `dependabot`[1] which is a GitHub feature for

[1] `https://docs.github.com/en/code-security/dependabot`

CPRA	GDPR	HIPAA	ECPA	CCPA	FERPA
400,426*	56,208	18,651	13,399	4,929	3,685
FCRA	GLBA	RGPD	VPPA	DSGVO	COPPA
3,104	1,862	1,776	1,337	960	933

Table 5.1: Number of issues that contain matches for the regulation names in their title, body or comments. (*): Figure for CPRA was affected by a bot-generated pull request as explaned in Paragraph 3 of this section.

developers to keep their projects updated whenever any other library that the project depends on publishes a new release. Dependabot is often used by repository owners to keep track of patches from dependencies published for code security and vulnerability, which would encompass any modifications to such libraries made for privacy compliance.

For this specific case, `dependabot` has generated the pull request due to a new release of the `axios`[2] library, and CPRA actually appears under the 'Changelog' section of the issue body within one of the e-mail addresses listed for giving credit to contributors. This is an example of a false positive described in Section 4.2.2. The sheer number of issues reporting a match for CPRA is due to `dependabot` generating a similar pull request for many repositories that uses the same `axios` library, which apparently has over 6.9 million dependent repositories[3].

If the tables that include text data described in Section 4.2.4 are loaded into the database, it is possible for the above case to filter out matches that came from the e-mail address that caused the false positive. Unfortunately, as also mentioned in the same section, importing these tables to our PostgreSQL instance would cause our account storage limit to be exceeded, therefore we did not have any means of filtering out such false positive matches.

Based on the observation regarding Dependabot, it is a good idea to see the distribution of issues mentioning the keywords regarding whether the issue has been created by a bot or a real user. Issue reporters can be queried using GHTorrent's `issues` table, however a quick query on this table reveals that the `reporter_id` field for most issues representing pull request is `null`. In order to avoid this issue in future queries, empty `reporter_id` fields are filled by the `author_id` of the commit represented by `head_commit_id` in the `pull_requests` table.

Table 5.2 represents how many of the matching issues were reported by real users versus bots. The exceptional case regarding CPRA is now made clear and for other keywords, a significant majority of issues appear to be created

[2] `https://github.com/axios/axios`

[3] `https://github.com/axios/axios/network/dependents`

CPRA	GDPR	HIPAA	ECPA
2,322/269,776	22,058/18,267	13,289/3,888	9,222/1,603
CCPA	FERPA	FCRA	GLBA
3,750/491	2,474/1,067	2,389/261	1,430/172
RGPD	VPPA	DSGVO	COPPA
1,433/172	950/146	824/40	773/42

Table 5.2: Number of issues that contain matches for the regulation names, created by real users versus bots.

by real users. This is a valuable observation which implies developers do not depend on automated methods of code security and take matters into their own hands when implementing privacy compliance.

The reason why the values for real user and bot commits do not add up to the figures in Table 5.1 is that some user and repository names recorded in the GHTorrent data have changed since the latest PostgreSQL dump. Since GHTorrent uses its own IDs for all entities instead of GitHub's inner IDs, connection to newly added MongoDB tables is forced to take place through matching user and repository names. If these names are different between PostgreSQL and MongoDB data then such entities cannot be related between the two data sources, causing them to be lost during table `JOIN`s.

Therefore, while the figures in Table 5.1 were obtained on pure MongoDB data, the queries executed for Table 5.1 needed to use GHTorrent's own `users` table to check whether the user is a bot, meaning that only the issues belonging to projects whose `owner/repo` naming is the same between GHTorrent and MongoDB data.

One other thing that we wanted to analyze is the rate at which an issue mentioning a privacy regulation name results in a commit being incorporated into the main branch as compared to the rate of issues not mentioning any such terms. This is a very complicated query that we wanted to try out in order to test our querying capabilities and runtime of more complicated queries. Executing this for issues mentioning GDPR took about an hour with some subqueries running in the matter of a couple of minutes and others taking up most of the processing time.

Results of this query which can be viewed in Appendix C.6 show that while 1.45% of issues mentioning GDPR result in a merged commit, this rate is 1% for issues not mentioning GDPR. A 45 percent increase in the merging rate is quite a significant increase which shows that privacy-related concerns are more likely to be addressed by the developers compared to issues that request or implement other functionality.

It is worth mentioning that since pull requests appear in GitHub as a sub-

type of issues, this query also includes any merge requests that implement privacy compliance. Perhaps it is the case that when some developer who is aware and knowledgeable of data protection laws takes time and effort to make a privacy-related contribution to the code base, project administrators who are not always as educated about this find it worth to accept the merge out of respect, necessity or other factors.

5.2 Time Series Data for Compliance Attempts

Nielsen's study reports most of its findings in the time series format in which the query results are grouped daily, monthly or yearly and the number of items in each group are plotted against the temporal x-axis. Nielsen investigated open-source compliance for GDPR and CCPA, and comparing our results for these two regulations against hers could help us see the validity of our tool and data.

Note that the time range of our data starts earlier than Nielsen's since we observe matches in earlier years but it also ends a few months earlier due to GHTorrent's latest update being published in March 2021.

Our results align with Nielsen's results in terms of general trends. The most prominent pattern when the time series for both GDPR and CCPA are inspected is that the daily frequency of created issues mentioning both keywords get considerably higher around the date when these regulations came into effect, but the same is not observed for the dates when these laws have been introduced first.

This trend shows that programmers often delay implementing compliance with privacy statutes or at least place it lower among their priorities until they are legally obliged to do so. This supports the evidence that developers are reluctant to adhere to PbD principles where behaving proactively is encouraged, and still obey data protection rules only out of necessity.

Regarding the activity spike observed in our data around February 2016, which is before GDPR passing, these mostly seem to come from a repository called `ImagicalMine/ImagicalMine`. Unfortunately, this project has been discontinued and issues reported to match GDPR are no longer accessible.

Interestingly, our data reveals that on the same month, we can see issue #843 of `openfoodfoundation/openfoodnetwork` mentioning EU cookie compliance and GDPR. This project deserves praise for acting on upcoming data protection laws even before the passing of the law, let alone its coming into force.

Upon later inspection of GHTorrent's dumps, we realized that MongoDB archives between July 2019 and late January 2020 are completely missing.

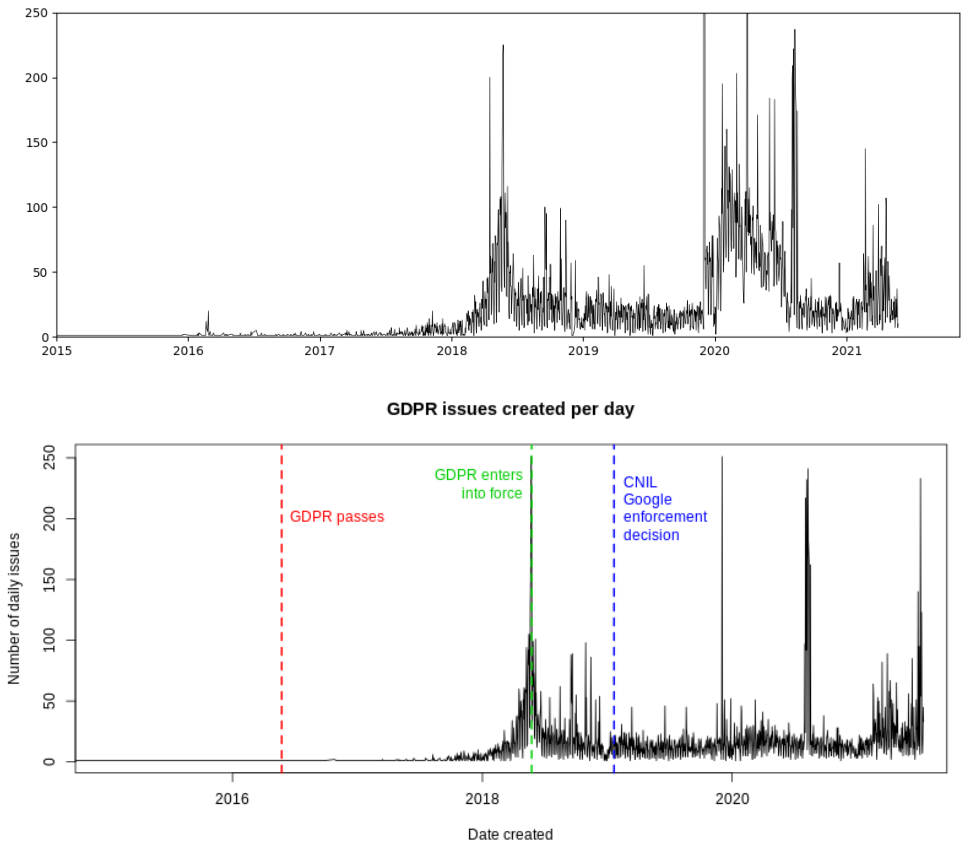

Figure 5.1: Top: Daily number of issues mentioning GDPR plotted from our data. Bottom: Nielsen's result on the same time series data.

We had to retrieve this data from an external source, therefore we downloaded the event stream data for the missing days from GH Archive described in Section 3.2. Then we filtered these events for entities created within the desired date range, generated keyword matching labels from them and formatted that data so that it can be imported into our tables. The plots obtained in Figure 5.1 and 5.2 contain the data from GH Archive events.

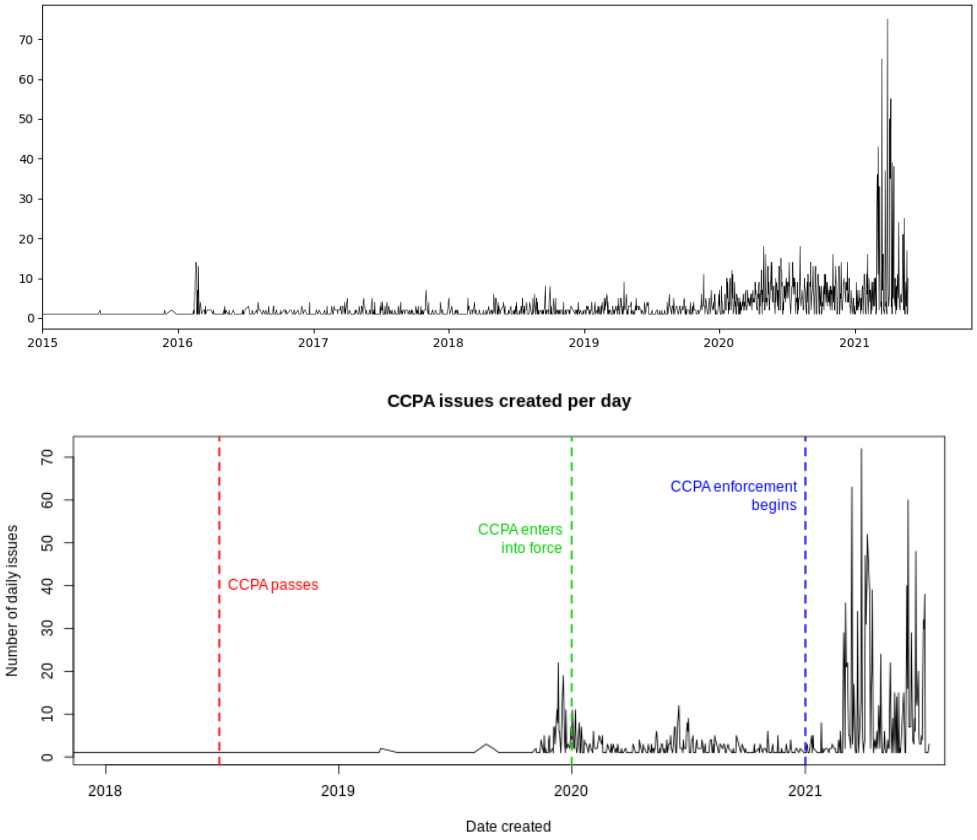

Figure 5.2: Top: Daily number of issues mentioning CCPA plotted from our data. Bottom: Nielsen's result on the same time series data.

Chapter 6

Conclusion

In this master thesis, we described a tool for querying open-source development data in the context of compliance with privacy and data protection regulations. We demonstrated the functionality of this tool by presenting our analysis results obtained through querying our data.

Despite being able to make some important comments on the ways how developers make contributions to open-source projects related to user privacy, we are aware that there is a large room for improvement regarding our data quality. Choosing to use GHTorrent as our basis seemed practical due to its data already being in the relational format that we desired; however, we failed to account for its questionable aspects such as incompleteness of the available data dumps and missing or out-of-date values for some fields in existing data.

The success and accuracy of this tool depends highly on data quality, therefore future effort to improve our project needs to involve re-collection of GitHub data using a better maintained source such as GHArchive. If the ability to process event stream data can be fully implemented, we can both make sure that the collected data is complete and any updates in GitHub data is reflected in our database (e.g. user and repository names being changed).

Regardless, if this report is viewed in a proof-of-concept manner, we believe our query tool still serves as a valuable contribution in the field of privacy compliance in software development. Even with data quality being lower than ideal, some of the findings that we were able to showcase have not been reported by previous analysis, such as the effect of privacy compliance on contribution acceptance in the form of merged pull requests. We hope that in the future, with the help of better and more complete data, this tool will allow researchers to obtain such novel insight on software privacy and open-source development in general.

Bibliography

[1] 20 U.S.C. § 1232g. Family educational and privacy rights. https://www.govinfo.gov/content/pkg/USCODE-2020-title20/pdf/USCODE-2020-title20-chap31-subchapIII-part4-sec1232g.pdf, 1974.

[2] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[3] Anas Al Bassit, Katsiaryna Krasnashchok, Sabri Skhiri, and Majd Mustapha. Policy-based automated compliance checking. In *International Joint Conference on Rules and Reasoning*, pages 3–17. Springer, 2021.

[4] Atheer Aljeraisy, Masoud Barati, Omer Rana, and Charith Perera. Privacy laws and privacy by design schemes for the internet of things: A developer's perspective. *ACM Computing Surveys (Csur)*, 54(5):1–38, 2021.

[5] Usman Ashraf, Christoph Mayr-Dorn, Atif Mashkoor, Alexander Egyed, and Sebastiano Panichella. Do communities in developer interaction networks align with subsystem developer teams? An empirical study of open source systems. In *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, pages 61–71. IEEE, 2021.

[6] David Banisar and Simon Davies. Global trends in privacy protection: An international survey of privacy, data protection, and surveillance laws and developments. *J. Marshall J. Computer & Info. L.*, 18:1, 1999.

[7] Andrea Bonaccorsi and Cristina Rossi. Why open source software can succeed. *Research policy*, 32(7):1243–1258, 2003.

[8] Cal. S. B. 1121 (2017-2018), Chapter 735. California Consumer Privacy Act of 2018. https://www.govinfo.gov/content/pkg/STATUTE-112/pdf/STATUTE-112-Pg2681.pdf#page=729, 2018.

[9] California Privacy Rights Act of 2020. *Text of Proposed Laws*, chapter Proposition 24, pages 42–75. California Secretary of State, 2020. https://vig.cdn.sos.ca.gov/2020/general/pdf/topl-prop24.pdf.

[10] Javier Luis Cánovas Izquierdo and Jordi Cabot. On the analysis of non-coding roles in open source development. *Empirical Software Engineering*, 27(1):1–32, 2022.

[11] Ann Cavoukian. Privacy by design. https://privacysecurityacademy.com/wp-content/uploads/2020/08/PbD-Principles-and-Mapping.pdf, 2009. [Online; accessed 10 August 2022].

[12] COM/2017/010 final - 2017/03 (COD). Proposal for a regulation of the European Parliament and of the Council concerning the respect for private life and the protection of personal data in electronic communications and repealing Directive 2002/58/EC (Regulation on Privacy and Electronic Communications) (Text with EEA relevance). https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52017PC0010&from=EN, 2017.

[13] Beate Commentz-Walter. A string matching algorithm fast on the average. In *International Colloquium on Automata, Languages, and Programming*, pages 118–132. Springer, 1979.

[14] Sabrina De Capitani Di Vimercati, Sara Foresti, Giovanni Livraga, and Pierangela Samarati. Data privacy: definitions and techniques. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 20(06):793–817, 2012.

[15] Luiz Felipe Dias, Igor Steinmacher, Gustavo Pinto, Daniel Alencar Da Costa, and Marco Gerosa. How does the shift to github impact project collaboration? In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 473–477. IEEE, 2016.

[16] Santiago Dueñas, Valerio Cosentino, Jesus M Gonzalez-Barahona, Alvaro del Castillo San Felix, Daniel Izquierdo-Cortazar, Luis Cañas-Díaz, and Alberto Pérez García-Plaza. Grimoirelab: A toolset for software development analytics. *PeerJ Computer Science*, 7:e601, 2021.

[17] Emre Erturk. A case study in open source software security and privacy: Android adware. In *World Congress on Internet Security (WorldCIS-2012)*, pages 189–191. IEEE, 2012.

[18] Roy Fielding. Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85, 2000.

[19] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[20] Georgios Gousios. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[21] Graham Greenleaf. Global data privacy laws: 89 countries, and accelerating. *privacy laws & business international report*, (115), 2012.

[22] Rajaa El Hamdani, Majd Mustapha, David Restrepo Amariles, Aurore Troussel, Sébastien Meeùs, and Katsiaryna Krasnashchok. A combined rule-based and machine learning approach for automated GDPR compliance checking. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, pages 40–49, 2021.

[23] Marit Hansen, Kristian Köhntopp, and Andreas Pfitzmann. The open source approach—opportunities and limitations with respect to security and privacy. *Computers & Security*, 21(5):461–471, 2002.

[24] User:Gohu1er / Wikimedia Commons / Public Domain. Example of a trie. `https://commons.wikimedia.org/wiki/File:Trie_example.svg`, 2021. [Online; accessed August 7, 2022].

[25] Open Source Initiative. The open source definition. `https://opensource.org/docs/osd`. [Online; accessed 5 August 2022].

[26] Bert-Jaap Koops, Jaap-Henk Hoepman, and Ronald Leenes. Open-source intelligence and privacy by design. *Computer Law & Security Review*, 29(6):676–688, 2013.

[27] Christian Kurtz, Martin Semmann, and Tilo Böhmann. Privacy by design to comply with GDPR: a review on third-party data processors. In *AMCIS 2018 Proceedings*, 36. AIS, 2018.

[28] Lawrence Lessig. Code is law. *Harvard magazine*, 1:2000, 2000.

[29] Ze Shi Li, Colin Werner, Neil Ernst, and Daniela Damian. Towards privacy compliance: A design science study in a small organization. *Information and Software Technology*, 146:106868, 2022.

[30] Tomer Libal. Towards automated GDPR compliance checking. In *International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning*, pages 3–19. Springer, 2020.

[31] Gregory Madey, Vincent Freeh, and Renee Tynan. The open source software development phenomenon: An analysis based on social network theory. In *AMCIS 2002 Proceedings*, pages 1806–1813. AIS, 2002.

[32] Donald R Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM (JACM)*, 15(4):514–534, 1968.

[33] Trong Nguyen. An empirical evaluation of the implementation of the california consumer privacy act (ccpa). *arXiv preprint arXiv:2205.09897*, 2022.

[34] OJ L119/1. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). `https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN`, 2016.

[35] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo. "Won't we fix this issue?" Qualitative characterization and automated identification of wontfix issues on GitHub. *Information and Software Technology*, 139:106665, 2021.

[36] Mariana Peixoto, Dayse Ferreira, Mateus Cavalcanti, Carla Silva, Jéssyka Vilela, João Araújo, and Tony Gorschek. On understanding how developers perceive and interpret privacy requirements research preview. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 116–123. Springer, 2020.

[37] Bruce Perens et al. The open source definition. *Open sources: voices from the open source revolution*, 1:171–188, 1999.

[38] Pub. L. No. 100-618, 102 Stat. 3195. Video Privacy Protection Act of 1988. `https://www.govinfo.gov/content/pkg/STATUTE-102/pdf/STATUTE-102-Pg3195.pdf`, 1988.

[39] Pub. L. No. 104-191, 110 Stat. 1936. Health Insurance Portability and Accountability Act of 1996. `https://www.govinfo.gov/content/pkg/STATUTE-110/pdf/STATUTE-110-Pg1936.pdf`, 1996.

[40] Pub. L. No. 105-277, 112 Stat. 2681-728. Children's Online Privacy Protection Act of 1998. `https://www.govinfo.gov/content/pkg/STATUTE-112/pdf/STATUTE-112-Pg2681.pdf#page=729`, 1998.

[41] Pub. L. No. 106-102, 113 Stat. 1338. Gramm–Leach–Bliley Act. https://www.govinfo.gov/content/pkg/STATUTE-113/pdf/STATUTE-113-Pg1338.pdf, 1999.

[42] Pub. L. No. 91-508, 84 Stat. 1114. An Act to amend the Federal Deposit Insurance Act to require insured banks to maintain certain records, to require that certain transactions in U.S. currency be reported to the Department of the Treasury, and for other purposes. https://www.govinfo.gov/content/pkg/STATUTE-84/pdf/STATUTE-84-Pg1114-2.pdf, 1970.

[43] Pub. L. No. 99-508, 100 Stat. 1848. Electronic Communications Privacy Act of 1986. https://www.govinfo.gov/content/pkg/STATUTE-100/pdf/STATUTE-100-Pg1848.pdf, 1986.

[44] Jeroen van Rest, Daniel Boonstra, Maarten Everts, Martin van Rijn, and Ron van Paassen. Designing privacy-by-design. In *Annual Privacy Forum*, pages 55–72. Springer, 2012.

[45] Nayan B Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[46] Similarweb. Top 10 github.com competitors & alternatives. https://www.similarweb.com/website/github.com/competitors. [Online; accessed 13 August 2022].

[47] Diomidis Spinellis. Version control systems. *IEEE software*, 22(5):108–109, 2005.

[48] Eeva Terkki, Ashwin Rao, and Sasu Tarkoma. Spying on android users through targeted ads. In *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*, pages 87–94. IEEE, 2017.

[49] Bruce William Watson. The performance of single-keyword and multiple-keyword pattern matching algorithms. *Computing Science Report*, 94(19), 1994.

[50] Razieh Nokhbeh Zaeem and K Suzanne Barber. The effect of the GDPR on privacy policies: Recent progress and future promise. *ACM Transactions on Management Information Systems (TMIS)*, 12(1):1–20, 2020.

[51] © User:Echion2 / Wikimedia Commons / CC-BY-SA-3.0 / GFDL. Visualization of the "history tree" of a revision controlled project, showing branching, merging, tagging, etc. https://commons.wikimedia.org/wiki/File:Revision_controlled_project_visualization-2010-24-02.svg, 2010. [Online; accessed August 7, 2022; modified].

[52] © Claudio Rocchini / Wikimedia Commons / CC-BY-2.5 / CC-BY-SA-3.0 / GFDL. Patricia trie (radix tree) sample: latinus word. https://commons.wikimedia.org/wiki/File:Patricia_trie.svg, 2007. [Online; accessed August 7, 2022].

Appendix A

Distribution of Data

The data that we have collected and processed are stored in a remote server in CSV format. Due to the size of the compressed archives (exceeding 400 GB even with highest compression setting) it is not possible for us to publicly host this data.

If you would like to try our query tool on your system, please contact Anıl Yarış (anilyaris@gmail.com), Karel Kubicek (karel.kubicek@inf.ethz.ch) and Aileen Nielsen (aileen.nielsen@gess.ethz.ch) in order to request access to our data.

After setting up the necessary SSH keys for connecting to the server that the data is stored in, we will provide scripts that will handle downloading the data and importing them into your existing PostgreSQL server, as well as an explanation of the tables and their schema for referencing while writing queries.

Appendix B

Code and Scripts

In this appendix, we list several scripts and code blocks that was implemented for various purposes during the development of our query tool.

B.1 Initial data collection demo using GitHub API

```
from github import *
from github.Repository import Repository
import gtrending
import pickle

tokens = [LIST_OF_GITHUB_AUTH_TOKENS]
token_index = 0
g = Github(tokens[token_index])
token_index += 1

class Repo:

  def __init__(self, repo):
    print("Repository:", repo.full_name)
    self.repo = repo
    self.buildMetadata()
    self.buildCommits()
    self.buildIssues()
    #self.contents = self.RepoTree(repo)
    #self.contents.buildRepoTree()

  def __repr__(self):
    return repr(self.contents)

  def buildMetadata(self):
    print("Collecting metadata...")
    self.repo.languages = [lang for lang in self.repo.get_languages()]
    self.repo.topics = [topic for topic in self.repo.get_topics()]
    self.repo.tags = [tag for tag in self.repo.get_tags()]
    self.repo.contributors = [contrib for contrib in self.repo.get_contributors()]
```

```
    self.repo.branches = [branch for branch in self.repo.get_branches()]

  def buildCommits(self):
    self.commits = []
    for branch in self.repo.branches:
      print("Collecting commits for branch", branch.name, '...')
      for commit in self.repo.get_commits(sha=branch.name):
        print(commit.sha, commit.commit.message)
        commit.branch = branch
        commit.comment_list = [comment for comment in commit.get_comments()]
        self.commits.append(commit)

  def buildIssues(self):
    print("Collecting issues & pull requests...")
    self.issues = []
    for issue in self.repo.get_issues(state='all', direction='asc'):
      print(issue.number, issue.title, "(PR)" if issue.pull_request else "")
      issue.comment_list = [comment for comment in issue.get_comments()]

      if issue.pull_request:
        reviews = [review for review in issue.as_pull_request().get_reviews()]
        if reviews:
          comment_index = 0
          comment_len = len(issue.comment_list)
          review_index = 0
          review_len = len(reviews)

          combined = []
          while comment_index < comment_len or review_index < review_len:
            comment = None
            if comment_index < comment_len:
              comment = issue.comment_list[comment_index]
            review = reviews[review_index] if review_index < review_len else None
            if not comment:
              combined.append(review)
              review_index += 1
            elif not review:
              combined.append(comment)
              comment_index += 1
            elif comment.created_at < review.submitted_at:
              combined.append(comment)
              comment_index += 1
            else:
              combined.append(review)
              review_index += 1

          issue.comment_list = combined

      self.issues.append(issue)

  class RepoTree:
    def __init__(self, obj=None):
      self.data = obj
      self.children = []
```

```
      self.parent = None

    def __repr__(self):
      return repr(self.data)

    def add(self, tree):
      self.children.append(tree)
      tree.parent = self

    def buildRepoTree(self):
      is_root = isinstance(self.data, Repository)
      if is_root or self.data.type == "dir":
        for content_file in (self.data if is_root else self.data.repository \
        ).get_contents("" if is_root else self.data.path):
          child = Repo.RepoTree(content_file)
          child.buildRepoTree()
          self.add(child)

def query(text):
  query_words = ['GDPR', 'CCPA', 'privacy',
  'consent', '3rd party', 'third party', 'third-party',
  'opt-in', 'opt-out', 'data protection']
  for word in query_words:
    if word in text:
      return True
  return False

def fetch_trending(dump=False):
  global g
  trending = gtrending.fetch_repos()
  for repo in trending:
    success = 0
    while not success:
      try:
        repo = g.get_repo(repo['fullname'])
        repo = Repo(repo)
        success = 1
      except GithubException as e:
        if token_index < len(tokens):
          g = Github(tokens[token_index])
          token_index += 1
        else:
          success = 2
    if success == 2:
      break

    if dump:
      pickle.dump(repo, open(repo.repo.name + ".pickle", "wb"), \
      protocol=pickle.HIGHEST_PROTOCOL)


fetch_trending(True)
repo = pickle.load(open("windowjs.pickle", "rb"))
results = []
```

```
for commit in repo.commits:
  print('Querying commit', commit.sha, commit.commit.message)

  if query(commit.commit.message):
    results.append(commit.commit)

  for comment in commit.comment_list:
    if query(comment.body):
      results.append(comment)

  if commit.files:
    for file in commit.files:
      if file.patch is not None and query(file.patch):
        results.append(file)

for issue in repo.issues:
  print('Querying issue', issue.number, issue.title, \
  "(PR)" if issue.pull_request else "")

  if query(issue.title) or (issue.body is not None and query(issue.body)):
    results.append(issue)

  for comment in issue.comment_list:
    if query(comment.body):
      results.append(comment)

print(results)
```

B.2 Fields retrieved from MongoDB

These are the MongoDB projection pipelines that return only the desired fields from MongoDB data.

```
pipelines = {
"issue_comments": [ {"$project": {
  "id": 1,
  "updated_at": 1,
  "body": 1, "owner": 1,
  "repo": 1,
  "issue_id": 1
} }],
"commits": [ {"$project": {
  "sha": 1,
  "message": "$commit.message",
  "files.filename": 1,
  "files.patch": 1
} }],
"issues": [ {"$project": {
  "number": 1,
  "title": 1,
  "state": 1,
  "updated_at": 1,
```

```
  "closed_at": 1,
  "body": 1,
  "closed_by.login": 1,
  "repo": 1,
  "owner": 1
} }],
"pull_requests": [ {"$project": {
  "number": 1,
  "merged_at": 1,
  "merged": 1,
  "merged_by.login": 1,
  "repo": 1,
  "owner": 1
} }],
"repos": [ {"$project": {
  "name": 1,
  "owner.login": 1,
  "private": 1,
  "pushed_at": 1,
  "homepage": 1,
  "has_wiki": 1,
  "has_pages": 1
} }],
}
```

B.3 Initial regular expression implementation

```
def match_keyword(text, keyword):
    pattern = ''
    for subword in keyword.split('_'):
        pattern += subword
        pattern += '.?'
    pattern = pattern[:-2]
    match = re.search(pattern, text, re.IGNORECASE)
    return bool(match)
```

Appendix C

Queries and SQL statements

In this appendix, we display some SQL statements that constitutes the executed queries or other parts of the database.

C.1 Creation queries for new MongoDB tables

```
CREATE TABLE mongo_issue_comments (
repo character varying(255),
owner character varying(255),
issue_id INTEGER, comment_id TEXT,
updated_at timestamp without time zone,
gdpr BOOLEAN, rgpd BOOLEAN,
dsgvo BOOLEAN, ccpa BOOLEAN,
cpra BOOLEAN, privacy BOOLEAN,
data_protection BOOLEAN, compliance BOOLEAN,
legal BOOLEAN, consent BOOLEAN,
law BOOLEAN, statute BOOLEAN,
personal_data BOOLEAN,
comply BOOLEAN, hipaa BOOLEAN,
fcra BOOLEAN, ferpa BOOLEAN,
glba BOOLEAN, ecpa BOOLEAN,
coppa BOOLEAN, vppa BOOLEAN
) WITHOUT OIDS;

CREATE TABLE mongo_commits (
"sha" character varying(40),
message_flag BOOLEAN, filename_flag BOOLEAN,
gdpr_added BOOLEAN, gdpr_removed BOOLEAN,
rgpd_added BOOLEAN, rgpd_removed BOOLEAN,
dsgvo_added BOOLEAN, dsgvo_removed BOOLEAN,
ccpa_added BOOLEAN, ccpa_removed BOOLEAN,
cpra_added BOOLEAN, cpra_removed BOOLEAN,
privacy_added BOOLEAN, privacy_removed BOOLEAN,
data_protection_added BOOLEAN, data_protection_removed BOOLEAN,
compliance_added BOOLEAN, compliance_removed BOOLEAN,
```

```
legal_added BOOLEAN, legal_removed BOOLEAN,
consent_added BOOLEAN, consent_removed BOOLEAN,
law_added BOOLEAN, law_removed BOOLEAN,
statute_added BOOLEAN, statute_removed BOOLEAN,
personal_data_added BOOLEAN, personal_data_removed BOOLEAN,
comply_added BOOLEAN, comply_removed BOOLEAN,
hipaa_added BOOLEAN, hipaa_removed BOOLEAN,
fcra_added BOOLEAN, fcra_removed BOOLEAN,
ferpa_added BOOLEAN, ferpa_removed BOOLEAN,
glba_added BOOLEAN, glba_removed BOOLEAN,
ecpa_added BOOLEAN, ecpa_removed BOOLEAN,
coppa_added BOOLEAN, coppa_removed BOOLEAN,
vppa_added BOOLEAN, vppa_removed BOOLEAN
) WITHOUT OIDS;

CREATE TABLE mongo_issues (
repo character varying(255),
owner character varying(255),
issue_id INTEGER, open BOOLEAN,
updated_at timestamp without time zone,
closed_at timestamp without time zone,
closed_by character varying(255), title_flag BOOLEAN,
gdpr BOOLEAN, rgpd BOOLEAN,
dsgvo BOOLEAN, ccpa BOOLEAN,
cpra BOOLEAN, privacy BOOLEAN,
data_protection BOOLEAN, compliance BOOLEAN,
legal BOOLEAN, consent BOOLEAN,
law BOOLEAN, statute BOOLEAN,
personal_data BOOLEAN,
comply BOOLEAN, hipaa BOOLEAN,
fcra BOOLEAN, ferpa BOOLEAN,
glba BOOLEAN, ecpa BOOLEAN,
coppa BOOLEAN, vppa BOOLEAN
) WITHOUT OIDS;

CREATE TABLE mongo_pull_requests (
repo character varying(255), owner character varying(255), pullreq_id INTEGER,
merged_at timestamp without time zone, merged BOOLEAN,
merged_by character varying(255)
) WITHOUT OIDS;

CREATE TABLE mongo_projects (
name character varying(255), owner character varying(255),
private BOOLEAN, pushed_at timestamp without time zone,
has_homepage BOOLEAN, has_wiki BOOLEAN, has_pages BOOLEAN
) WITHOUT OIDS;
```

C.2 Creation queries for extended MongoDB tables

```
CREATE TABLE mongo_issue_comments_extended (
repo character varying(255),
owner character varying(255),
```

```
issue_id INTEGER, comment_id TEXT,
created_at timestamp without time zone
) WITHOUT OIDS;

CREATE TABLE mongo_commits_extended (
"sha" character varying(40),
created_at timestamp without time zone,
message_gdpr BOOLEAN, message_rgpd BOOLEAN,
message_dsgvo BOOLEAN, message_ccpa BOOLEAN,
message_cpra BOOLEAN, message_privacy BOOLEAN,
message_data_protection BOOLEAN, message_compliance BOOLEAN,
message_legal BOOLEAN, message_consent BOOLEAN,
message_law BOOLEAN, message_statute BOOLEAN,
message_personal_data BOOLEAN,
message_comply BOOLEAN, message_hipaa BOOLEAN,
message_fcra BOOLEAN, message_ferpa BOOLEAN,
message_glba BOOLEAN, message_ecpa BOOLEAN,
message_coppa BOOLEAN, message_vppa BOOLEAN,
filename_gdpr BOOLEAN, filename_rgpd BOOLEAN,
filename_dsgvo BOOLEAN, filename_ccpa BOOLEAN,
filename_cpra BOOLEAN, filename_privacy BOOLEAN,
filename_data_protection BOOLEAN, filename_compliance BOOLEAN,
filename_legal BOOLEAN, filename_consent BOOLEAN,
filename_law BOOLEAN, filename_statute BOOLEAN,
filename_personal_data BOOLEAN,
filename_comply BOOLEAN, filename_hipaa BOOLEAN,
filename_fcra BOOLEAN, filename_ferpa BOOLEAN,
filename_glba BOOLEAN, filename_ecpa BOOLEAN,
filename_coppa BOOLEAN, filename_vppa BOOLEAN
) WITHOUT OIDS;

CREATE TABLE mongo_issues_extended (
repo character varying(255),
owner character varying(255), issue_id INTEGER,
created_at timestamp without time zone,
title_gdpr BOOLEAN, title_rgpd BOOLEAN,
title_dsgvo BOOLEAN, title_ccpa BOOLEAN,
title_cpra BOOLEAN, title_privacy BOOLEAN,
title_data_protection BOOLEAN, title_compliance BOOLEAN,
title_legal BOOLEAN, title_consent BOOLEAN,
title_law BOOLEAN, title_statute BOOLEAN,
title_personal_data BOOLEAN,
title_comply BOOLEAN, title_hipaa BOOLEAN,
title_fcra BOOLEAN, title_ferpa BOOLEAN,
title_glba BOOLEAN, title_ecpa BOOLEAN,
title_coppa BOOLEAN, title_vppa BOOLEAN
) WITHOUT OIDS;

CREATE TABLE mongo_projects_extended (
name character varying(255), owner character varying(255),
created_at timestamp without time zone
) WITHOUT OIDS;
```

C.3 Keyword queries on unprocessed GHTorrent data

```
SELECT COUNT(*) FROM commit_comments WHERE body ILIKE '%gdpr%';
 count
-------
   118
(1 row)

SELECT COUNT(*) FROM pull_request_comments WHERE body ILIKE '%gdpr%';
 count
-------
  1430
(1 row)

SELECT COUNT(*) FROM commit_comments WHERE body ILIKE '%ccpa%';
 count
-------
    38
(1 row)

SELECT COUNT(*) FROM pull_request_comments WHERE body ILIKE '%ccpa%';
 count
-------
   290
(1 row)
```

C.4 Keyword queries using keyword match labels

```
# Group CCPA issues by month

SELECT to_char(issues.created_at, 'YYYY-MM') as month, COUNT(*)
FROM issues, projects, users,
(SELECT repo, owner, issue_id FROM mongo_issues WHERE ccpa = TRUE) as ccpa_true
WHERE users.login = ccpa_true.owner AND
users.id = projects.owner_id AND projects.name = ccpa_true.repo AND
issues.repo_id = projects.id AND issues.issue_id = ccpa_true.issue_id
GROUP BY month ORDER BY month ASC;

  month   | count
---------+-------
 2012-07 |     1
 2012-10 |     2
 2015-03 |     1
 2015-06 |     1
 2015-11 |     3
 2016-01 |     4
 2016-02 |    48
 2016-03 |    10
 2016-04 |     8
 2016-05 |     7
 2016-06 |    12
```

```
2016-07 |      4
2016-08 |     13
2016-09 |      6
2016-10 |      4
2016-11 |      6
2016-12 |      8
2017-01 |      5
2017-02 |     10
2017-03 |     13
2017-04 |      9
2017-05 |     18
2017-06 |     18
2017-07 |     17
2017-09 |     15
2017-10 |     14
2017-11 |     11
2017-12 |      7
2018-01 |     14
2018-02 |      7
2018-03 |     11
2018-04 |     13
2018-05 |     28
2018-06 |     17
2018-07 |     23
2018-08 |     26
2018-09 |      8
2018-10 |     18
2018-11 |      9
2018-12 |     16
2019-01 |     21
2019-02 |     13
2019-03 |     17
2019-04 |     29
2019-05 |     16
2019-06 |      1
2019-08 |      4
2019-09 |      4
2019-10 |      4
2019-11 |      6
2019-12 |      9
2020-01 |     35
2020-02 |     34
2020-03 |     61
2020-04 |     70
2020-05 |     75
2020-06 |    119
2020-07 |    103
2020-08 |     68
2020-09 |     75
2020-10 |     38
2020-11 |     60
2020-12 |     79
2021-01 |     34
2021-02 |     57
```

```
 2021-03 |   119

# Group CCPA commits by month

SELECT to_char(commits.created_at, 'YYYY-MM') as month, COUNT(*) FROM commits,
(SELECT sha FROM mongo_commits
WHERE ccpa_added = TRUE OR
ccpa_removed = TRUE) as ccpa_true
WHERE commits.sha = ccpa_true.sha
GROUP BY month ORDER BY month ASC;

  month   | count
----------+-------
 1970-01 |     1
 1985-07 |     2
 1986-04 |     2
 1991-11 |     1
 1992-04 |     1
 1993-01 |     2
 1993-03 |     4
 1993-04 |     2
 1993-05 |     4
 1993-06 |     2
 1993-07 |     3
 1993-08 |     5
 1993-09 |     1
 1993-10 |     4
 1993-11 |     7
 1993-12 |     3
 1994-01 |     4
 1994-03 |     1
 1994-04 |     2
 1994-05 |     4
 1994-06 |     1
 1994-08 |     1
 1994-09 |     5
 1994-10 |     2
 1994-11 |     1
 1994-12 |     2
 1995-02 |     2
 1995-03 |     1
 1995-04 |     5
 1995-05 |     4
 1995-06 |     6
 1995-07 |     3
 1995-08 |     7
 1995-09 |     2
 1995-10 |    33
 1995-11 |     3
 1995-12 |     3
 1996-01 |     5
 1996-02 |     2
 1996-03 |     3
 1996-04 |     1
```

```
1996-05 |    14
1996-06 |     5
1996-07 |     5
1996-08 |     3
1996-09 |     9
1996-10 |     7
1996-11 |     4
1996-12 |   352
1997-01 |   519
1997-02 |     6
1997-03 |   260
1997-04 |     3
1997-05 |     8
1997-06 |     3
1997-07 |     9
1997-08 |     1
1997-09 |   146
1997-10 |     9
1997-11 |    10
1997-12 |     4
1998-01 |     8
1998-02 |    14
1998-03 |    24
1998-04 |     2
1998-05 |    12
1998-06 |     5
1998-07 |    14
1998-08 |     5
1998-09 |     6
1998-10 |    13
1998-11 |     6
1998-12 |     4
1999-01 |     2
1999-02 |     9
1999-03 |     7
1999-04 |     6
1999-05 |     4
1999-06 |     7
1999-07 |    11
1999-08 |    13
1999-09 |     9
1999-10 |   190
1999-11 |    10
1999-12 |     8
2000-01 |    18
2000-02 |     9
2000-03 |     8
2000-04 |     8
2000-05 |    13
2000-06 |     8
2000-07 |    17
2000-08 |    11
2000-09 |    12
2000-10 |    15
```

```
2000-11 |    13
2000-12 |     4
2001-01 |    10
2001-02 |    14
2001-03 |    44
2001-04 |    10
2001-05 |   185
2001-06 |    29
2001-07 |    21
2001-08 |    16
2001-09 |    16
2001-10 |    23
2001-11 |    18
2001-12 |    11
2002-01 |    10
2002-02 |    32
2002-03 |    17
2002-04 |    11
2002-05 |    19
2002-06 |    17
2002-07 |    13
2002-08 |    12
2002-09 |    13
2002-10 |    14
2002-11 |     7
2002-12 |    10
2003-01 |    17
2003-02 |    18
2003-03 |    20
2003-04 |    15
2003-05 |    33
2003-06 |    18
2003-07 |    29
2003-08 |    21
2003-09 |    13
2003-10 |    11
2003-11 |    15
2003-12 |    17
2004-01 |    24
2004-02 |    11
2004-03 |    18
2004-04 |    14
2004-05 |    16
2004-06 |    19
2004-07 |     8
2004-08 |    21
2004-09 |    18
2004-10 |     4
2004-11 |     9
2004-12 |    13
2005-01 |     9
2005-02 |    10
2005-03 |    13
2005-04 |     9
```

```
2005-05 |    20
2005-06 |   269
2005-07 |    12
2005-08 |    32
2005-09 |    14
2005-10 |    40
2005-11 |    44
2005-12 |    24
2006-01 |     7
2006-02 |    26
2006-03 |    43
2006-04 |    27
2006-05 |    15
2006-06 |    12
2006-07 |    15
2006-08 |    38
2006-09 |    30
2006-10 |    16
2006-11 |    34
2006-12 |    32
2007-01 |    20
2007-02 |    25
2007-03 |    31
2007-04 |    61
2007-05 |    21
2007-06 |    12
2007-07 |    23
2007-08 |    37
2007-09 |    15
2007-10 |    31
2007-11 |    25
2007-12 |    31
2008-01 |    36
2008-02 |    38
2008-03 |    62
2008-04 |    23
2008-05 |    50
2008-06 |    39
2008-07 |    44
2008-08 |    29
2008-09 |    52
2008-10 |    36
2008-11 |    31
2008-12 |    30
2009-01 |    31
2009-02 |    25
2009-03 |    52
2009-04 |    39
2009-05 |    57
2009-06 |    50
2009-07 |    50
2009-08 |    63
2009-09 |    61
2009-10 |    63
```

```
2009-11 |    23
2009-12 |    54
2010-01 |    60
2010-02 |    79
2010-03 |    75
2010-04 |    93
2010-05 |    94
2010-06 |    74
2010-07 |    50
2010-08 |    82
2010-09 |    80
2010-10 |    60
2010-11 |    54
2010-12 |    82
2011-01 |    70
2011-02 |    60
2011-03 |    64
2011-04 |    57
2011-05 |    92
2011-06 |    90
2011-07 |   146
2011-08 |    82
2011-09 |    89
2011-10 |    76
2011-11 |    74
2011-12 |    88
2012-01 |    85
2012-02 |   120
2012-03 |   137
2012-04 |    93
2012-05 |   101
2012-06 |   147
2012-07 |   104
2012-08 |   167
2012-09 |   116
2012-10 |   103
2012-11 |   136
2012-12 |   111
2013-01 |   123
2013-02 |    90
2013-03 |   139
2013-04 |   106
2013-05 |    72
2013-06 |   164
2013-07 |   193
2013-08 |    73
2013-09 |    98
2013-10 |   136
2013-11 |    95
2013-12 |    72
2014-01 |    87
2014-02 |    73
2014-03 |    91
2014-04 |   111
```

```
2014-05 |   152
2014-06 |   104
2014-07 |    85
2014-08 |   143
2014-09 |   133
2014-10 |   134
2014-11 |   172
2014-12 |   100
2015-01 |   101
2015-02 |   141
2015-03 |   174
2015-04 |   195
2015-05 |   126
2015-06 |   282
2015-07 |   202
2015-08 |   182
2015-09 |   257
2015-10 |   280
2015-11 |   962
2015-12 |  1777
2016-01 |  3307
2016-02 |  4142
2016-03 |  5076
2016-04 |  5081
2016-05 |  5906
2016-06 |  5999
2016-07 |  4406
2016-08 |  5828
2016-09 |  5941
2016-10 |  7642
2016-11 |  7454
2016-12 |  6549
2017-01 |  7162
2017-02 |  6771
2017-03 |  8765
2017-04 |  9251
2017-05 |  8245
2017-06 |  7970
2017-07 |  8321
2017-08 |  8548
2017-09 | 10323
2017-10 |  9595
2017-11 | 14482
2017-12 | 10432
2018-01 | 14556
2018-02 | 10692
2018-03 | 19073
2018-04 | 18740
2018-05 | 25012
2018-06 | 17391
2018-07 | 10930
2018-08 | 11169
2018-09 | 10899
2018-10 | 13749
```

```
 2018-11 | 11479
 2018-12 |  9751
 2019-01 | 11667
 2019-02 | 10181
 2019-03 |  9943
 2019-04 | 12773
 2019-05 | 12695
 2019-06 |   940
 2019-07 |   534
 2019-08 |   877
 2019-09 |   903
 2019-10 |  1096
 2019-11 |  1524
 2019-12 |  3590
 2020-01 | 10840
 2020-02 | 21214
 2020-03 | 26069
 2020-04 | 23811
 2020-05 | 27504
 2020-06 | 26805
 2020-07 | 25915
 2020-08 | 42063
 2020-09 | 34679
 2020-10 | 28270
 2020-11 | 22562
 2020-12 | 11216
 2021-01 | 15493
 2021-02 | 32893
 2021-03 |  6042
```

C.5 Keyword queries with updated creation timestamps

```
CREATE TABLE mongo_issue_comments_ccpa AS
SELECT * FROM mongo_issue_comments
WHERE mongo_issue_comments.ccpa = TRUE;

CREATE TABLE mongo_issues_ccpa AS
SELECT * FROM mongo_issues
WHERE mongo_issues.ccpa = TRUE;

CREATE TABLE mongo_issues_extended_ccpa AS
SELECT * FROM mongo_issues_extended
WHERE mongo_issues_extended.title_ccpa = TRUE;

CREATE TABLE issues_ccpa AS
SELECT repo, owner, issue_id FROM mongo_issues_ccpa
UNION
SELECT repo, owner, issue_id FROM mongo_issues_extended_ccpa
UNION
SELECT repo, owner, issue_id FROM mongo_issue_comments_ccpa;

CREATE TABLE issues_ccpa_with_dates AS
```

```
SELECT issues_ccpa.*, mongo_issues_extended.created_at
FROM issues_ccpa LEFT JOIN mongo_issues_extended
ON issues_ccpa.repo = mongo_issues_extended.repo AND
issues_ccpa.owner = mongo_issues_extended.owner AND
issues_ccpa.issue_id = mongo_issues_extended.issue_id
ORDER BY created_at ASC;

DROP TABLE issues_ccpa;

SELECT to_char(created_at, 'YYYY-MM') as month, COUNT(*)
FROM issues_ccpa_with_dates
WHERE created_at IS NOT NULL
GROUP BY month ORDER BY month ASC;
```

month	count
2010-12	1
2011-12	1
2012-07	1
2012-10	3
2014-06	1
2015-03	1
2015-06	6
2015-08	2
2015-11	6
2015-12	3
2016-01	9
2016-02	58
2016-03	18
2016-04	13
2016-05	20
2016-06	18
2016-07	13
2016-08	23
2016-09	24
2016-10	14
2016-11	17
2016-12	23
2017-01	14
2017-02	25
2017-03	33
2017-04	20
2017-05	38
2017-06	38
2017-07	46
2017-08	29
2017-09	29
2017-10	37
2017-11	31
2017-12	20
2018-01	31
2018-02	23
2018-03	31
2018-04	30

```
 2018-05 |    56
 2018-06 |    24
 2018-07 |    45
 2018-08 |    56
 2018-09 |    35
 2018-10 |    47
 2018-11 |    29
 2018-12 |    30
 2019-01 |    41
 2019-02 |    44
 2019-03 |    49
 2019-04 |    63
 2019-05 |    51
 2019-06 |    30
 2019-07 |     5
 2019-08 |    16
 2019-09 |    16
 2019-10 |    16
 2019-11 |    31
 2019-12 |    37
 2020-01 |    72
 2020-02 |   111
 2020-03 |   115
 2020-04 |   169
 2020-05 |   197
 2020-06 |   199
 2020-07 |   198
 2020-08 |   173
 2020-09 |   193
 2020-10 |   172
 2020-11 |   186
 2020-12 |   167
 2021-01 |   122
 2021-02 |   170
 2021-03 |   540
 2021-04 |   441
 2021-05 |   155
```

```
CREATE TABLE mongo_commits_ccpa AS
SELECT * FROM mongo_commits
WHERE ccpa_added = TRUE OR ccpa_removed = TRUE;

CREATE TABLE mongo_commits_extended_ccpa AS
SELECT * FROM mongo_commits_extended
WHERE message_ccpa = TRUE OR filename_ccpa = TRUE;

CREATE TABLE commits_ccpa AS
SELECT sha FROM mongo_commits_ccpa
UNION
SELECT sha FROM mongo_commits_extended_ccpa;

CREATE TABLE commits_ccpa_with_dates AS
SELECT mongo_commits_extended.sha, mongo_commits_extended.created_at
FROM commits_ccpa LEFT JOIN mongo_commits_extended
```

```
ON commits_ccpa.sha = mongo_commits_extended.sha
ORDER BY created_at ASC;

DROP TABLE commits_ccpa;

SELECT to_char(created_at, 'YYYY-MM') as month, COUNT(*)
FROM commits_ccpa_with_dates
GROUP BY month ORDER BY month ASC;
```

month	count
1970-01	1
1985-07	2
1986-04	2
1992-04	1
1993-01	2
1993-03	3
1993-04	2
1993-05	5
1993-06	2
1993-07	2
1993-08	5
1993-09	1
1993-10	4
1993-11	8
1993-12	3
1994-01	5
1994-03	1
1994-04	3
1994-05	4
1994-06	1
1994-08	3
1994-09	5
1994-10	2
1994-11	1
1994-12	2
1995-02	2
1995-03	1
1995-04	6
1995-05	4
1995-06	3
1995-07	3
1995-08	7
1995-09	4
1995-10	34
1995-11	3
1995-12	3
1996-01	5
1996-02	2
1996-03	2
1996-04	1
1996-05	16
1996-06	5
1996-07	5

```
1996-08 |      3
1996-09 |     10
1996-10 |      8
1996-11 |      6
1996-12 |    540
1997-01 |    537
1997-02 |      6
1997-03 |    272
1997-04 |      2
1997-05 |      8
1997-06 |      4
1997-07 |      9
1997-08 |      1
1997-09 |    317
1997-10 |      9
1997-11 |     10
1997-12 |      4
1998-01 |      7
1998-02 |     14
1998-03 |     23
1998-04 |      1
1998-05 |     12
1998-06 |      5
1998-07 |     15
1998-08 |      5
1998-09 |      6
1998-10 |     13
1998-11 |      6
1998-12 |      6
1999-01 |      5
1999-02 |      8
1999-03 |      9
1999-04 |      6
1999-05 |    174
1999-06 |      7
1999-07 |     11
1999-08 |     12
1999-09 |      9
1999-10 |    202
1999-11 |     13
1999-12 |      9
2000-01 |     22
2000-02 |      9
2000-03 |     11
2000-04 |      8
2000-05 |     14
2000-06 |     10
2000-07 |     22
2000-08 |     11
2000-09 |     14
2000-10 |     15
2000-11 |     13
2000-12 |      5
2001-01 |     10
```

```
2001-02 |    14
2001-03 |    54
2001-04 |    10
2001-05 |   544
2001-06 |    33
2001-07 |    22
2001-08 |    16
2001-09 |    19
2001-10 |    26
2001-11 |    17
2001-12 |    12
2002-01 |    11
2002-02 |    34
2002-03 |    17
2002-04 |    13
2002-05 |    21
2002-06 |    17
2002-07 |    14
2002-08 |    12
2002-09 |    15
2002-10 |    15
2002-11 |     7
2002-12 |    11
2003-01 |    17
2003-02 |    24
2003-03 |    20
2003-04 |    16
2003-05 |    37
2003-06 |    23
2003-07 |    37
2003-08 |    32
2003-09 |    16
2003-10 |    21
2003-11 |    17
2003-12 |    25
2004-01 |    24
2004-02 |    16
2004-03 |    19
2004-04 |    16
2004-05 |    17
2004-06 |    19
2004-07 |    10
2004-08 |    24
2004-09 |    30
2004-10 |     7
2004-11 |    10
2004-12 |    19
2005-01 |    11
2005-02 |    10
2005-03 |    16
2005-04 |    20
2005-05 |    21
2005-06 |   281
2005-07 |    20
```

```
2005-08 |    38
2005-09 |    18
2005-10 |    44
2005-11 |    46
2005-12 |    26
2006-01 |    11
2006-02 |    28
2006-03 |    49
2006-04 |    30
2006-05 |    16
2006-06 |    15
2006-07 |    21
2006-08 |    43
2006-09 |    32
2006-10 |    21
2006-11 |    50
2006-12 |    42
2007-01 |    27
2007-02 |    30
2007-03 |    36
2007-04 |    71
2007-05 |    26
2007-06 |    14
2007-07 |    33
2007-08 |    46
2007-09 |    26
2007-10 |    42
2007-11 |    33
2007-12 |    33
2008-01 |    45
2008-02 |    45
2008-03 |    67
2008-04 |    28
2008-05 |    52
2008-06 |    45
2008-07 |    64
2008-08 |    37
2008-09 |    72
2008-10 |    41
2008-11 |    41
2008-12 |    44
2009-01 |    50
2009-02 |    37
2009-03 |    63
2009-04 |    55
2009-05 |    68
2009-06 |    57
2009-07 |    71
2009-08 |    83
2009-09 |    81
2009-10 |    94
2009-11 |    60
2009-12 |    62
2010-01 |    76
```

```
2010-02 |    90
2010-03 |    95
2010-04 |   124
2010-05 |   125
2010-06 |   105
2010-07 |    86
2010-08 |   137
2010-09 |   139
2010-10 |    81
2010-11 |    92
2010-12 |   105
2011-01 |   100
2011-02 |    79
2011-03 |    82
2011-04 |    85
2011-05 |   130
2011-06 |   165
2011-07 |   188
2011-08 |   111
2011-09 |   135
2011-10 |   133
2011-11 |   127
2011-12 |   138
2012-01 |   131
2012-02 |   162
2012-03 |   220
2012-04 |   126
2012-05 |   142
2012-06 |   176
2012-07 |   123
2012-08 |   191
2012-09 |   162
2012-10 |   146
2012-11 |   190
2012-12 |   181
2013-01 |   174
2013-02 |   115
2013-03 |   201
2013-04 |   177
2013-05 |    93
2013-06 |   221
2013-07 |   286
2013-08 |   120
2013-09 |   158
2013-10 |   177
2013-11 |   188
2013-12 |   209
2014-01 |   156
2014-02 |   134
2014-03 |   164
2014-04 |   169
2014-05 |   252
2014-06 |   133
2014-07 |   126
```

```
2014-08 |   193
2014-09 |   179
2014-10 |   161
2014-11 |   240
2014-12 |   129
2015-01 |   151
2015-02 |   183
2015-03 |   237
2015-04 |   247
2015-05 |   201
2015-06 |   230
2015-07 |   280
2015-08 |   225
2015-09 |   243
2015-10 |   348
2015-11 |  1070
2015-12 |  1992
2016-01 |  3464
2016-02 |  4502
2016-03 |  5357
2016-04 |  5328
2016-05 |  6238
2016-06 |  6512
2016-07 |  4986
2016-08 |  6252
2016-09 |  6220
2016-10 |  8713
2016-11 |  8191
2016-12 |  6875
2017-01 |  7626
2017-02 |  7006
2017-03 |  9237
2017-04 |  8818
2017-05 |  8699
2017-06 |  8358
2017-07 |  8691
2017-08 |  8854
2017-09 | 10719
2017-10 |  9980
2017-11 | 14892
2017-12 | 10987
2018-01 | 14949
2018-02 | 10981
2018-03 | 20542
2018-04 | 19269
2018-05 | 25620
2018-06 | 17805
2018-07 | 11445
2018-08 | 11638
2018-09 | 11333
2018-10 | 14256
2018-11 | 11929
2018-12 | 10037
2019-01 | 12274
```

```
 2019-02 | 10784
 2019-03 | 10903
 2019-04 | 13472
 2019-05 | 13577
 2019-06 | 12380
 2019-07 |   758
 2019-08 |  1052
 2019-09 |  1133
 2019-10 |  1329
 2019-11 |  1763
 2019-12 |  3927
 2020-01 | 11198
 2020-02 | 21768
 2020-03 | 26535
 2020-04 | 24411
 2020-05 | 28368
 2020-06 | 27384
 2020-07 | 25794
 2020-08 | 41707
 2020-09 | 36039
 2020-10 | 29790
 2020-11 | 23841
 2020-12 | 12087
 2021-01 | 16413
 2021-02 | 34758
 2021-03 | 38842
 2021-04 | 33967
 2021-05 | 10491
```

C.6 Querying pull request merge rates

```
-- Filter out issue comments mentioning GDPR
CREATE TABLE mongo_issue_comments_gdpr AS
SELECT * FROM mongo_issue_comments
WHERE mongo_issue_comments.gdpr = TRUE;

-- Filter out issues mentioning GDPR in the body
CREATE TABLE mongo_issues_gdpr AS
SELECT * FROM mongo_issues
WHERE mongo_issues.gdpr = TRUE;

-- Filter out issues mentioning GDPR in the title
CREATE TABLE mongo_issues_extended_gdpr AS
SELECT * FROM mongo_issues_extended
WHERE mongo_issues_extended.title_gdpr = TRUE;

-- Merge the filtered tables which should represent all issues that mention GDPR
CREATE TABLE issues_gdpr AS
SELECT repo, owner, issue_id FROM mongo_issues_gdpr
UNION
SELECT repo, owner, issue_id FROM mongo_issues_extended_gdpr
UNION
```

```
SELECT repo, owner, issue_id FROM mongo_issue_comments_gdpr;

-- Replace repo-owner-issue number with issue IDs
CREATE TABLE issues_gdpr_with_ids AS
SELECT issues.id as issue_id
FROM issues_gdpr, issues, projects, users
WHERE issues_gdpr.repo = projects.name AND
issues_gdpr.owner = users.login AND
projects.owner_id = users.id AND
issues_gdpr.issue_id = issues.issue_id AND
issues.repo_id = projects.id;

DROP TABLE issues_gdpr;

-- List all issues that reference commits
CREATE TABLE issues_referencing_commits AS
SELECT DISTINCT issue_id, action_specific as sha
FROM issue_events
WHERE action = 'referenced' AND action_specific IS NOT NULL;

-- Intersect above table with issues mentioning GDPR
CREATE TABLE issues_referencing_commits_gdpr AS
SELECT issues_referencing_commits.*
FROM issues_referencing_commits INNER JOIN issues_gdpr_with_ids
ON issues_referencing_commits.issue_id = issues_gdpr_with_ids.issue_id;

-- Subtract intersection
CREATE TABLE issues_referencing_commits_not_gdpr AS
SELECT * FROM issues_referencing_commits
EXCEPT
SELECT * FROM issues_referencing_commits_gdpr;

DROP TABLE issues_referencing_commits;

-- List all merged PRs
CREATE TABLE merged_pull_requests AS
SELECT *
FROM mongo_pull_requests
WHERE mongo_pull_requests.merged = TRUE;

-- Replace repo-owner-PR number with PR IDs
CREATE TABLE merged_pull_requests_with_ids AS
SELECT pull_requests.id as pull_request_id
FROM merged_pull_requests, pull_requests, projects, users
WHERE merged_pull_requests.repo = projects.name AND
merged_pull_requests.owner = users.login AND
projects.owner_id = users.id AND
merged_pull_requests.pullreq_id = pull_requests.pullreq_id AND
pull_requests.base_repo_id = projects.id;

DROP TABLE merged_pull_requests;

-- Get list of commits that are part of a merged PR
CREATE TABLE merged_commits AS
```

```
SELECT pull_request_commits.commit_id
FROM merged_pull_requests_with_ids INNER JOIN pull_request_commits
ON merged_pull_requests_with_ids.pull_request_id
= pull_request_commits.pull_request_id;

DROP TABLE merged_pull_requests_with_id;

-- Replace commit ID with SHA for above table
CREATE TABLE merged_commits_sha AS
SELECT commits.sha
FROM merged_commits INNER JOIN commits
ON merged_commits.commit_id = commits.id;

DROP TABLE merged_commits;

-- Get issues mentioning GDPR referencing a merged commit
CREATE TABLE query2_gdpr AS
SELECT DISTINCT issues_referencing_commits_gdpr.issue_id
FROM issues_referencing_commits_gdpr INNER JOIN merged_commits_sha
ON issues_referencing_commits_gdpr.sha = merged_commits_sha.sha;

-- Get issues not mentioning GDPR referencing a merged commit
CREATE TABLE query2_not_gdpr AS
SELECT DISTINCT issues_referencing_commits_not_gdpr.issue_id
FROM issues_referencing_commits_not_gdpr INNER JOIN merged_commits_sha
ON issues_referencing_commits_not_gdpr.sha = merged_commits_sha.sha;

DROP TABLE merged_commits_sha;

-- Get what ratio of issues (not) mentioning GDPR refer to a merged commit
SELECT (SELECT COUNT(*) FROM query2_gdpr)::float /
(SELECT COUNT(*) FROM issues_gdpr_with_ids);

       ?column?
----------------------
 0.014534739761303423
(1 row)

SELECT (SELECT COUNT(*) FROM query2_not_gdpr)::float /
(SELECT COUNT(issues_not_gdpr_with_ids.*) FROM
        (SELECT id as issue_id FROM issues
        EXCEPT
        SELECT * FROM issues_gdpr_with_ids) as issues_not_gdpr_with_ids
);

       ?column?
----------------------
 0.010090053040497688
(1 row)
```

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

QUERY TOOL FOR ANALYZING PRIVACY CHANGES OF GITHUB-HOSTED OSS PROJECTS

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
YARIS	ANIL

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Ankara, 14.08.2022	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.