PREDICTING THE EFFECTS OF REFACTORING

CURRENT STATE

OVERVIEW AND GOALS

- Predicting refactoring effects using machine learning
  - Through product metrics before and after a refactoring commit based on history git data
- Goals
  - Create a large dataset containing enough refactoring operations
  - Determine which features are suitable to predict how well an refactoring operation is performing
  - Generate a prediction model based those features and evaluate it
- Main research questions to be answered
  - **Which features of a commit-refactoring combination are suitable to predict the effect of the refactoring method?**
  - **How well does the trained machine learning model perform?**

STEPS

1. Select suitable repositories having an adequate number of commits and refactoring operations
2. Create a dataset with several product metrics and source code measures extracted from the previously selected repositories
3. Explore the dataset and determine which features are strong and suitable to build a prediction model
4. Train the machine learning model
5. Evaluate the model performance
6. Write documentation

STEPS

1. Select suitable repositories having an adequate number of commits and refactoring operations
2. Create a dataset with several product metrics and source code measures extracted from the previously selected repositories
3. Explore the dataset and determine which features are strong and suitable to build a prediction model
4. Train the machine learning model
5. Evaluate the model performance
6. Write documentation

REPOSITORY SELECTION

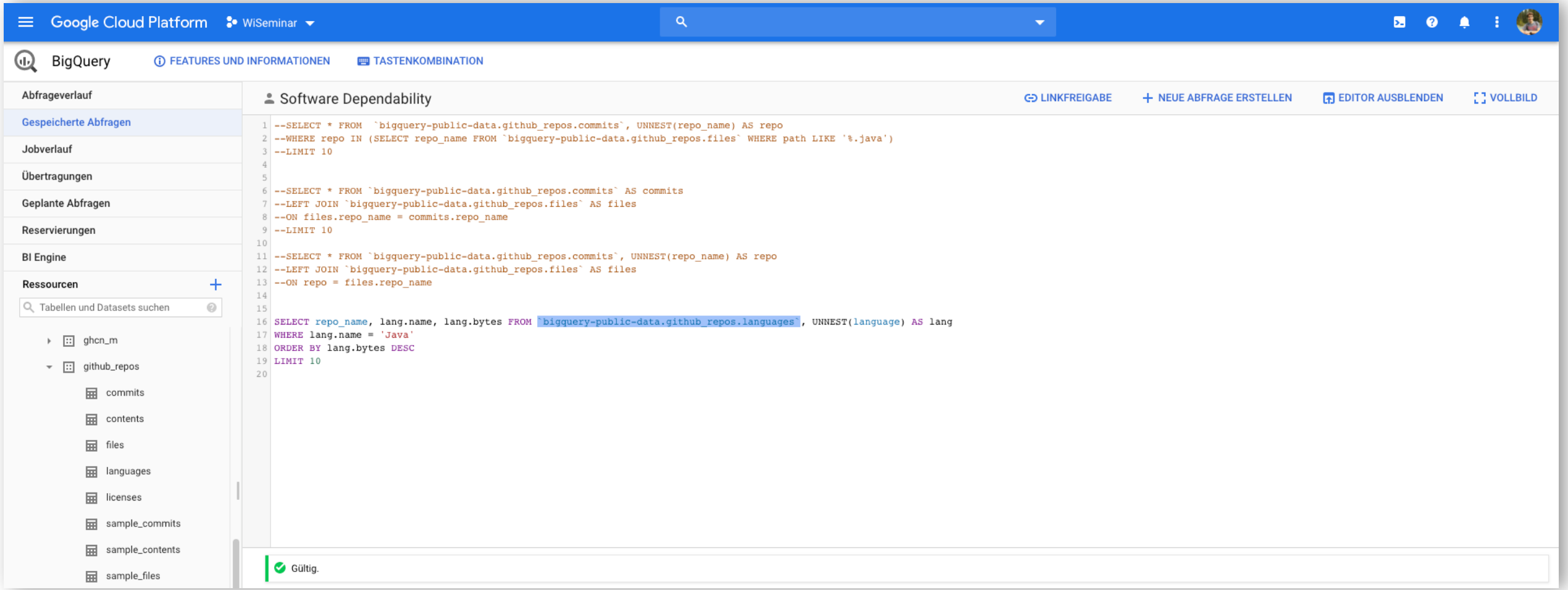
- Requirements
  - Creation of a large dataset
  - Adequate amount of commits and refactoring operations
  - Open-source and public available to sustain the possibility of recreating the research idea and steps
  - Focus on one programming language to avoid any language specific bias and get comparable results

- How can those repositories be found?
  - Github has a huge amount of hosted repositories, but which to choose?
  - Github itself does not provide any datasets
  - **Basic idea: Use existing datasets and research on dataset generation for static source code analysis**

PRACTICAL APPROACH: GOOGLE BIGQUERY

- Google BigQuery is an online SaaS for big data operations using their implementation of the MapReduce pattern
- Lots of public datasets are available to run queries on (in their extended SQL dialect)

- Github repository version dataset is available [1]
  - Containing more than 300.000 indexed Github repositories
  - Repositories are stored in a structured format and contain meta data of the repo, all commits, file metadata, changes and also the file contents
  - Querying is fast and easy (only limited to 1TB per month)


Google Cloud Platform
WiSeminar
BigQuery
FEATURES UND INFORMATIONEN
TASTENKOMBINATION
Abfrageverlauf
Gespeicherte Abfragen
Jobverlauf
Übertragungen
Geplante Abfragen
Reservierungen
BI Engine
Ressourcen
Tabellen und Datasets suchen
ghcn_m
github_repos
commits
contents
files
languages
licenses
sample_commits
sample_contents
sample_files
Software Dependability
LINKFREIGABE
NEUE ABFRAGE ERSTELLEN
EDITOR AUSBLENDEN
VOLLBILD
--SELECT * FROM `bigquery-public-data.github_repos.commits`, UNNEST(repo_name) AS repo
--WHERE repo IN (SELECT repo_name FROM `bigquery-public-data.github_repos.files` WHERE path LIKE '%.java')
--LIMIT 10
--SELECT * FROM `bigquery-public-data.github_repos.commits` AS commits
--LEFT JOIN `bigquery-public-data.github_repos.files` AS files
--ON files.repo_name = commits.repo_name
--LIMIT 10
--SELECT * FROM `bigquery-public-data.github_repos.commits`, UNNEST(repo_name) AS repo
--LEFT JOIN `bigquery-public-data.github_repos.files` AS files
--ON repo = files.repo_name
SELECT repo_name, lang.name, lang.bytes FROM `bigquery-public-data.github_repos.languages`, UNNEST(language) AS lang
WHERE lang.name = 'Java'
ORDER BY lang.bytes DESC
LIMIT 10
Gültig.

DRAWBACKS

- Dataset does not contain important repositories (e.g. missing the Spring Boot framework repositories which could be good candidates for Java source code)
- Many forked repositories not providing any additional information and making the dataset even smaller
  - Shows an additinoal problem: Avoid forked repositories bias

- Next step: More scientific approach!

SCIENTIFIC APPROACH

- „Public Git Archive: a Big Code dataset for all“ by Markovtsev and Long [2]
- Approach to generate a public available dataset for AI research on static source code
- Conducted by the company SourceD located in Madrid

- They proposed a data pipeline to extract and explore the version control history from Github [3]
  - Targeting the datasets which are available from GHTorrent
  - Horizontally scalable to process millions of repositories
  - Introduced a new storage format called „Siva“ which should reduce the used disc space for forked repositories

- Looking good, right?

Public Git Archive: a Big Code dataset for all

Vadim Markovtsev
source{d}
Madrid, Spain
vadim@sourced.tech

Waren Long
source{d}
Madrid, Spain
waren@sourced.tech

arXiv:1803.09144v1 [cs.SE] 20 Mar 2018

ABSTRACT

The number of open source software projects has been growing exponentially. The major online software repository host, GitHub, has accumulated tens of millions of publicly available Git version-controlled repositories. Although the research potential enabled by the available open source code is clearly substantial, no significant large-scale open source code datasets exist. In this paper, we present the Public Git Archive – dataset of 182,014 top-bookmarked Git repositories from GitHub. We describe the novel data retrieval pipeline to reproduce it. We also elaborate on the strategy for performing dataset updates and legal issues. The Public Git Archive occupies 3.0 TB on disk and is an order of magnitude larger than the current source code datasets. The dataset is made available through HTTP and provides the source code of the projects, the related metadata, and development history. The data retrieval pipeline employs an optimized worker queue model and an optimized archive format to efficiently store forked Git repositories, reducing the amount of data to download and persist. Public Git Archive aims to open a myriad of new opportunities for "Big Code" research.

CCS CONCEPTS

• **Human-centered computing** → *Empirical studies in collaborative and social computing*; • **Software and its engineering** → *Soft-*

control accessible, therefore universal. The next stage of the revolution is permitting the automatic analysis of source code at scale, to support data-driven language design, to infer best (and worst) practices, and to provide the raw data to data hungry machine learning techniques that will be the basis of the next generation of development tools [3, 15]. It requires source code archives that are both big and programmatically accessible for analysis.

The GHTorrent project [12] took first steps in this direction, focusing on metadata in order to be scalable. Current source code datasets typically contain tens of thousands of projects at most [3] and are dedicated to particular programming languages such as Java and JavaScript [25], thus lacking diversity and attracting critics [6]. Software Heritage [7] is a recent attempt to archive all the open source code ever written, however no public dataset has been published yet by them.

We present the Public Git Archive, the first big code dataset amenable to programmatic analysis at scale. It is by far the biggest curated archive of top-rated[1] repositories on GitHub, see Table 1 for comparison. The Public Git Archive targets large-scale quantitative research in the areas of source code analysis (SCA) and machine learning on source code (MLoSC). The dataset is made available via HTTP as a separate index file together with files in the Siva format, a novel archive format tailored for storing Git repositories efficiently [29]. Every GitHub repository can be forked; forks

src-d / datasets

Watch 17 | Star 172 | Fork 42

Code | Issues 24 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights

Branch: master ▾ datasets / PublicGitArchive / | Create new file | Upload files | Find file | History

r0mainK Move the poster assets to a separate subdir ··· ✓ Latest commit 37cc34c on 29 Oct 2019

..		
doc	Move the poster assets to a separate subdir	6 months ago
list-pga-heads	Add the PGA heads docs	7 months ago
pga-create	Switch to go mod	7 months ago
pga	Fill TODOs, improve doc	6 months ago
pga2uast	Run git repo enumeration in parallel	8 months ago
poster	Move the poster assets to a separate subdir	6 months ago
web	mention pga on the docs	2 years ago
README.md	Fill TODOs, improve doc	6 months ago

README.md

Public Git Archive size 6.0TB

Paper (accepted to MSR'18). Presentation.


⌥⌘1 ~/Desktop (zsh) ⌘1 ..Code/datasets (z... ⌘2 ..licGitArchive (zsh) ⌘3 ~/sdk/spark (zsh) ⌘4 ~ (zsh) ⌘5
~
8%
11 GB
0.0 kB↓ 1.0 kB↑
~ pga get siva -l java
Error: could not open index file: could not copy to temporary file /Users/martinsteinhauer/.pga/siva/latest.index.csv.gz.tmp: Get "http://pga.sourced.tech/csv/siva/latest.index.csv.gz": dial tcp: lookup pga.sourced.tech: no such host
~

vmarkovtsev commented on 8 Jan Collaborator

Sorry for responding late (winter holidays). source{d} no longer exists, so the public datasets that had to be served from a dedicated server are all down. The tooling that depends on the server is essentially non-functional anymore. The files that are on Google Drive are still there and were additionally backed up, so they should continue to work. We were able to copy all the siva and UAST parquet files to Google Cloud Storage, however, **we accidentally lost the PGA index file (CSV). If somebody has a copy, please send it to me.**

The new company - Athenian - where some of the core devs migrated including me is sponsoring the GCS, however, public serving from there costs too much currently. I'll try to find an alternative way with **@Guillemdb** and **@sergio-hcsoft** who have recently bought the former data processing/ML cluster and have 80TB of free storage.

Now if you need a fraction of PGA, please send me the list of desired siva files, I will fetch them from GCS, package together and upload to Google Drive. Composing that list without an index is impossible though.

GHTORRENT

- Online service that has several Github datasets stored and available to download
- Data formats: Dumps for MongoDB and MySQL

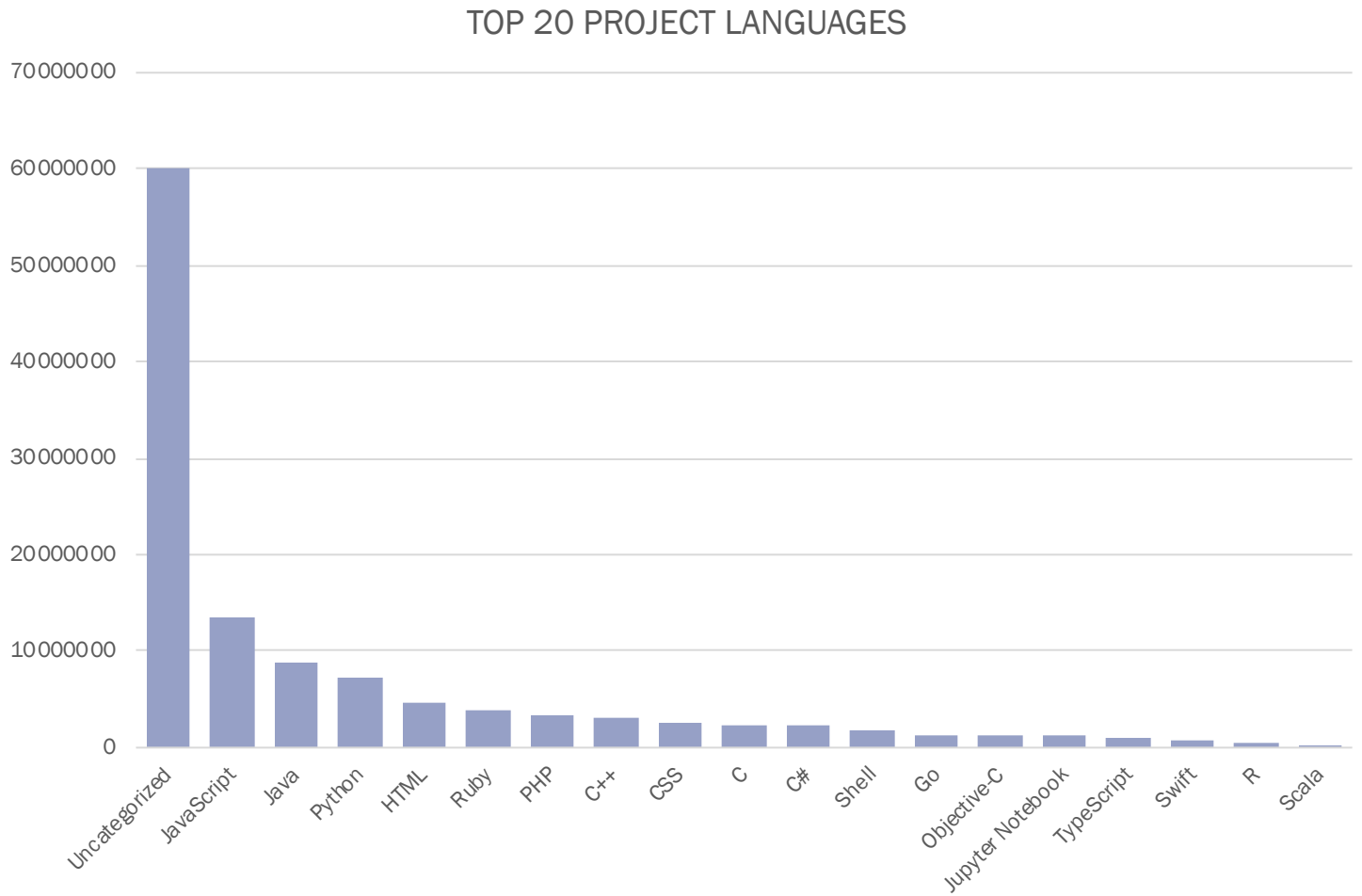
- Latest dataset from June 2019
- tar-archive size: 100GB, extracted about 400GB
- Counted projects: 125.543.333
- Java projects: 8.759.213

BUILDING THE DATASET PIPELINE

- Goal: Get all projects with Java source code, a high number of commits and sorted by their rating (stars)

- GHTorrent data: MySQL dump stored as CSV files and schema (we do not do any MySQL restore!)
- Pipeline environment: **Apache Spark** [4]
- By using the CSV files we can do SQL-like operations directly on the files
- Advantages
  - Benefit of distributed computing and MapReduce patterns, allows big join operations on the source dataset
  - Highly scalable and parallelizable when storing source files in the Hadoop file system and using a Spark cluster on e.g. AWS or Azue
  - Easy to use and reproducable

EXAMPLE: LANGUAGE DISTRIBUTION

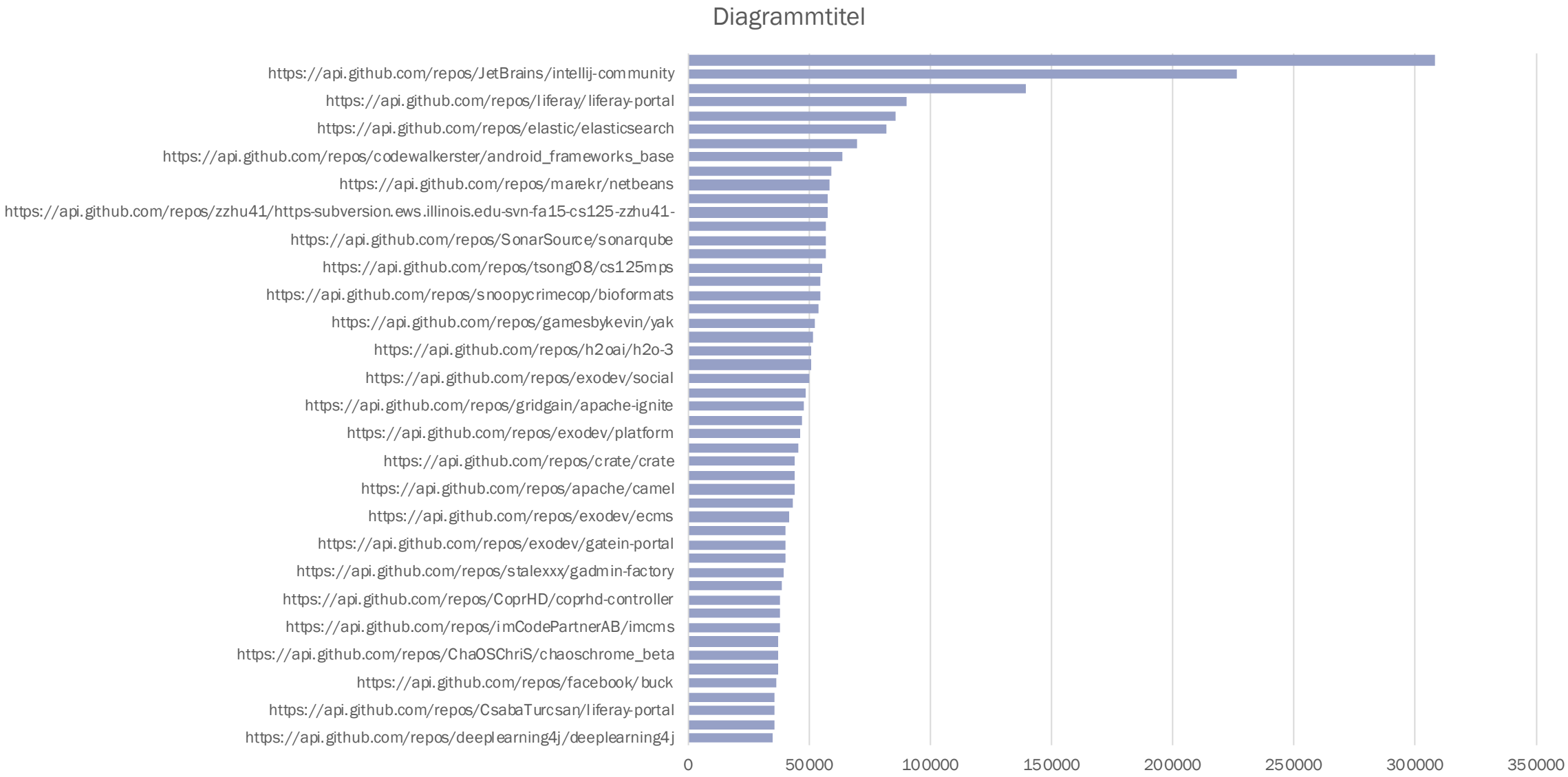


JOINING COMMITS AND PROJECTS

```sql
CREATE TABLE IF NOT EXISTS `ghtorrent`.`projects` (
    `id` INT(11) NOT NULL AUTO_INCREMENT COMMENT '',
    `url` VARCHAR(255) NULL DEFAULT NULL COMMENT '',
    `owner_id` INT(11) NULL DEFAULT NULL COMMENT '',
    `name` VARCHAR(255) NOT NULL COMMENT '',
    `description` VARCHAR(255) NULL DEFAULT NULL COMMENT '',
    `language` VARCHAR(255) NULL DEFAULT NULL COMMENT '',
    `created_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '',
    `forked_from` INT(11) NULL DEFAULT NULL COMMENT '',
    `deleted` TINYINT(1) NOT NULL DEFAULT '0' COMMENT '',
    `updated_at` TIMESTAMP NOT NULL DEFAULT '1970-01-01 00:00:01' COMMENT '',
    PRIMARY KEY (`id`)  COMMENT '',
    CONSTRAINT `projects_ibfk_1`
    FOREIGN KEY (`owner_id`)
    REFERENCES `ghtorrent`.`users` (`id`),
    CONSTRAINT `projects_ibfk_2`
    FOREIGN KEY (`forked_from`)
    REFERENCES `ghtorrent`.`projects` (`id`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8;

CREATE TABLE IF NOT EXISTS `ghtorrent`.`commits` (
    `id` INT(11) NOT NULL AUTO_INCREMENT COMMENT '',
    `sha` VARCHAR(40) NULL DEFAULT NULL COMMENT '',
    `author_id` INT(11) NULL DEFAULT NULL COMMENT '',
    `committer_id` INT(11) NULL DEFAULT NULL COMMENT '',
    `project_id` INT(11) NULL DEFAULT NULL COMMENT '',
    `created_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '',
    PRIMARY KEY (`id`)  COMMENT '',
    CONSTRAINT `commits_ibfk_1`
    FOREIGN KEY (`author_id`)
    REFERENCES `ghtorrent`.`users` (`id`),
    CONSTRAINT `commits_ibfk_2`
    FOREIGN KEY (`committer_id`)
    REFERENCES `ghtorrent`.`users` (`id`),
    CONSTRAINT `commits_ibfk_3`
    FOREIGN KEY (`project_id`)
    REFERENCES `ghtorrent`.`projects` (`id`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8;
```

```java
System.out.println("Starting app '" + APP_NAME + "'");


Dataset<Row> projectDataset = session.read()
        .format("csv")
        .option("header", "false")
        .option("mode", "DROPMALFORMED")
        .schema(DatasetHeader.getProjectHeader())
        .load(projectSourceFile)
        .filter("language = 'Java'");

Dataset<Row> commitDataset = session.read()
        .format("csv")
        .option("header", false)
        .option("mode", "DROPMALFORMED")
        .schema(DatasetHeader.getCommitHeader())
        .load(commitSourceFile);

JavaRDD<Row> results = projectDataset
        .join(commitDataset, projectDataset.col( colName: "id").equalTo(commitDataset.col( colName: "project_id")))
        .select( col: "url")
        .groupBy( col1: "url") RelationalGroupedDataset
        .count() Dataset<Row>
        .toJavaRDD();

results.saveAsTextFile( path: "/Users/martinsteinhauer/Desktop/commitResults");
```

TOP 50 REPOSITORIES WITH MOST COMMITS



NEXT STEPS

Continue creating the dataset

- Select the repositories with most commits
- Extend the pipeline to automatically clone the repositories
- Run Refactoring Miner on the cloned repositories [5][6]
- Generate a set of different product metrics before and after the refactoring applied

REFERENCES

[1] https://console.cloud.google.com/marketplace/details/github/github-repos?filter=solution-type:dataset

[2] V. Markovtsev und W. Long, „Public git archive: A big code dataset for all“, *Proc. - Int. Conf. Softw. Eng.*, S. 34–37, 2018.

[3] https://github.com/src-d/datasets

[4] https://spark.apache.org/

[5] N. Tsantalis, M. Mansouri, L. M. Eshkevari, D. Mazinanian, and D. Dig, “Accurate and efficient refactoring detection in commit history,” pp. 483–494, 2018.

[6] https://github.com/tsantalis/RefactoringMiner