WILL TURINGBOTS REPLACE HUMAN SOFTWARE DEVELOPERS?

AN ANALYSIS USING GITHUB DATA

Presented by: Zeel Patel
Big Data & Cloud Computing

December 2024

OVERVIEW

01 Executive Summary

02 Data Preprocessing

03 EDA of Variables

04 Analysis Techniques

05 Conclusion

06 Appendix

EXECUTIVE SUMMARY

OBJECTIVE

This project explores whether AI tools like GitHub CoPilot, ChatGPT, and similar assistants can enhance developer productivity, streamline software development processes, and potentially replace human software engineers and data scientists in specific tasks. The study focuses on understanding trends in programming languages, repository activity, commit patterns, and technological breakthroughs. The analysis is based on GitHub Archive data (~1.36 TiB) stored in Google Cloud, providing a comprehensive view of open-source development dynamics over time.

KEY FINDINGS

- Timeline Insights: Peaks in commits align with major tech events; data gaps pre-2008 and post-2022 identified.
- Programming Languages: Python and JavaScript dominate, while C shows a decline.
- Licensing Patterns: MIT license is the most widely used, linked to modern languages.
- Commit Trends: New technology development and bug fixes are the top commit reasons.
- Text Similarity: ~85% of commit subject and message values are unique.
- Prolific Contributors: Key contributors drive significant impact on open-source projects.

RECOMMENDATIONS

- AI tools like GitHub CoPilot and ChatGPT automate repetitive tasks, enhancing efficiency. However, they lack the creativity and problem-solving capabilities needed for complex system design, keeping developers essential for high-level tasks.
- Big Tech companies drive technological innovation through open-source projects, accelerating adoption of modern tools like CoPilot and fostering collaboration in the software development ecosystem.

METHODOLOGY

1. Data Acquisition

- Source: GitHub Archive data (~1.36 TiB) hosted on Google Cloud Storage.
- Structure: Nested data divided into five folders (Commits, Contents, Files, Languages, Licenses).
- Access: Used PySpark for distributed data processing.

2. Data Preprocessing

- Eliminated irrelevant and erroneous records to reduce data volume.
- Parsed nested structures to extract meaningful variables.
- Handled missing and duplicate data for analysis accuracy.
- Sampled 20% of the data.

3. Exploratory Data Analysis (EDA)

- Timeline analysis: Identified peaks, valleys, and data gaps.
- Trends in programming languages and licensing.
- Understanding License distribution
- Repository growth patterns and influential technologies.
- Influence of AI/Data Science Projects on the commits
- Identified the most frequent and influential committers
- Text Uniqueness

4. Analysis Techniques

- Conducted timeline analysis to identify peaks, valleys, and data gaps.
- Used keyword-based classification to identify AI and data science repositories.
- Categorized commit reasons (e.g., bug fixes, new features) and analyzed commit patterns.
- Applied Cosine similarity for text analysis to detect duplication in commit messages.

SOURCE DATA OVERVIEW

Dataset Description

- Name: GitHub Archive Dataset
- Size: ~1.36 TiB
- Storage: Google Cloud Storage
- Structure: Five subfolders containing Parquet files

Folder Contents

1. Commits: Metadata on repository commits (author, date, message).
2. Contents: Source code and file content within repositories.
3. Files: Metadata about files (path, type, blob ID).
4. Languages: Distribution of code by programming language.
5. Licenses: Licensing information for repositories.

Sampling Approach

- Sample Size: Randomly sampled 20% of the commits dataset.
- Randomly sampled to reduce processing time while retaining data diversity for analysis.
- Filtering the cleaned datasets (files, languages, licenses) to include only rows where repo_name matches the repo_name values in the sampled commits dataset, ensuring consistency across datasets for further analysis.

Key Challenges

- Nested structure required detailed parsing.
- Large data size posed computational constraints.
- Noisy and incomplete records needed filtering.

DATA PREPROCESSING

Data Cleaning

- **Removed:**
- Erroneous or irrelevant records (e.g., null or incomplete values).
- Duplicates to improve data quality.
- **Standardized:**
- Variable formats (e.g., timestamps, text fields).
- Nested fields into flattened structures for easier analysis.

Sampling

Commit Dataset:
Randomly sampled 20% of the commits dataset to reduce processing time while retaining data diversity for analysis.

Handling Nested Structures

- Extracted meaningful fields from deeply nested data.
- Mapped hierarchical relationships for easier querying.

Storage Optimization

Saved intermediate results in Parquet format to improve read/write speeds and reduce storage costs.

Tools & Techniques

- Used PySpark for distributed data processing.
- Employed filtering and transformation techniques to reduce noise.

EXPLORATORY DATA ANALYSIS

Overview
To perform an exploratory data analysis (EDA) to identify usable variables and eliminate poorly populated or duplicate fields in the datasets.

1. commits_sampled (393,172,705 records):
- Cleaned by filtering null values, ensuring consistency between authors and committers, and removing duplicates.
- Randomly sampled 20% of the cleaned dataset for efficient analysis.

2. contents_cleaned (228,154,045 records):
- Rows with null values in critical columns (id, size, content) were dropped.
- Duplicates were removed to retain only valid and unique records.

3. files_filtered (4,834,290 records):
- Cleaned to ensure only relevant file metadata remained, focusing on unique records.
- Filtered to include only records where repo_name matched those from the commits_sampled dataset.

4. languages_filtered (4,834,290 records):
- Flattened nested arrays to extract programming languages and byte usage.
- Duplicates were removed to provide a clean language breakdown per repository.
- Filtered to include only records where repo_name matched those from the commits_sampled dataset.

5. licenses_filtered (2,692,937 records):
- Cleaned by removing null or duplicate rows, ensuring only valid license information was retained for analysis.
- Filtered to include only records where repo_name matched those from the commits_sampled dataset.

	Dataset Name	Record Count
0	commits_sampled	393,172,705
1	contents_cleaned	228,154,045
2	files_filtered	4,834,290
3	languages_filtered	4,834,290
4	licenses_filtered	2,692,937

Fig 1.1: Record Count Table

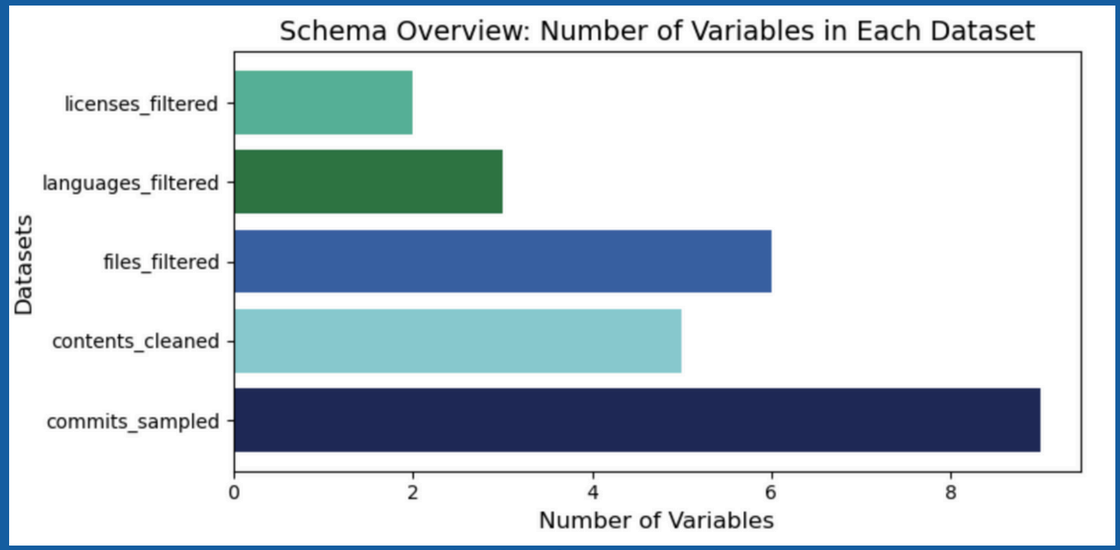


Fig 1.2: Schema Overview

TIMELINE ANALYSIS

Outliers and Data Gaps

1. Pre-2008:
   - Outliers removed as GitHub was founded in 2008, and data before this year is removed.
2. Post-2022:
   - High volume of null or zero values observed, indicating incomplete or poor-quality data collection.
3. Data Gaps:
   - Gaps in activity likely stem from irregular data archiving practices or changes in GitHub's data accessibility policies.

Insights on Spikes and Valleys

- 2011–2015 Spikes:
  - Likely linked to major technology launches such as the rise of mobile app development and open-source frameworks (e.g., React, Angular).
  - Contributions from Big Tech companies (Google, Microsoft, etc.) popularizing open-source development.
- Post-2019 Decline:
  - Could be influenced by global events such as the COVID-19 pandemic, which shifted focus to remote work and reduced new software projects in some sectors.

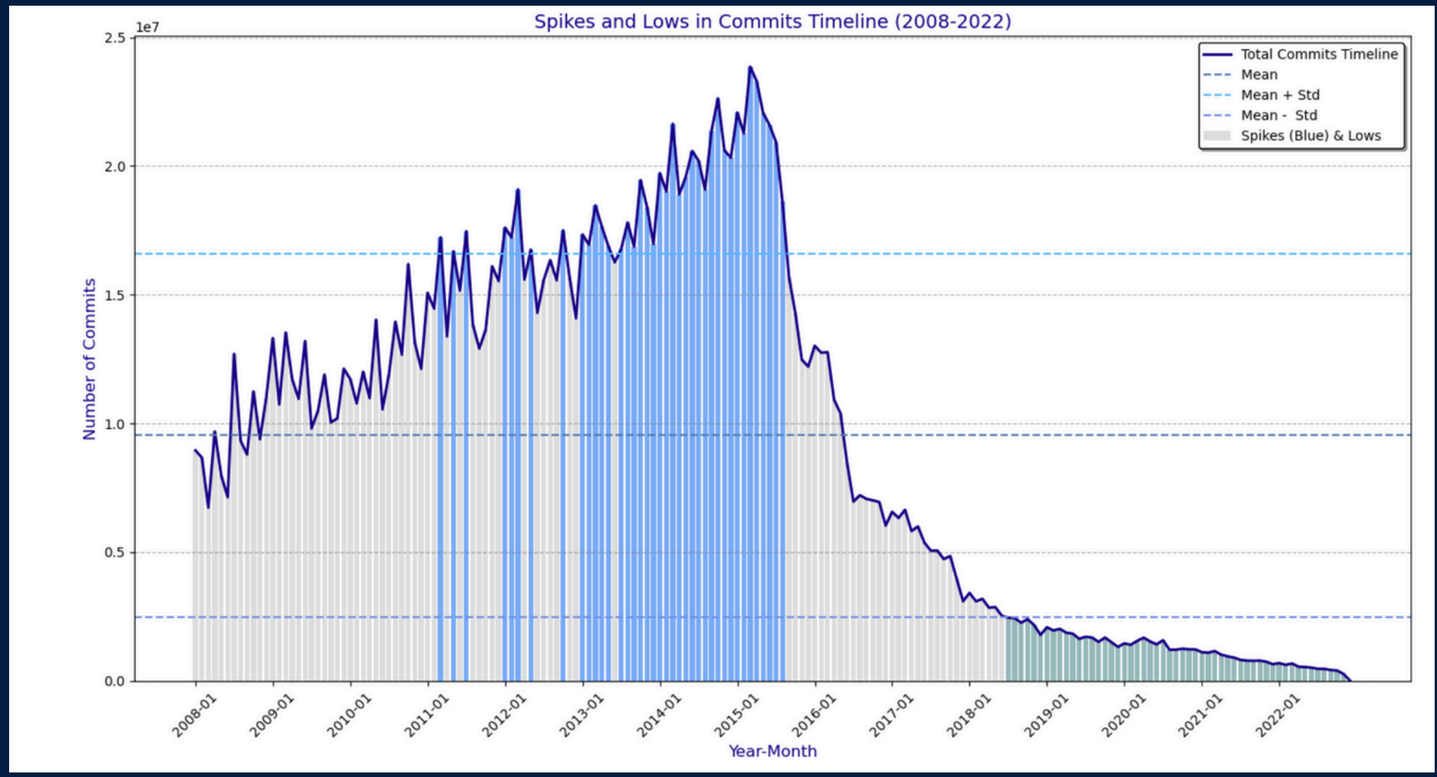


Fig 2.1: Timeline of commit activity (2008–2022), showing peaks, valleys, the mean and standard deviations

Timeline Overview

Time Period: 2008–2022

Key Observations:

- Significant spikes in commit activity observed between 2011 and 2015.
- Gradual decline in commit activity after 2016, with a sharp drop post 2019.

PROGRAMMING LANGUAGE TRENDS

Overview

- Analyzed programming language trends on GitHub from 2008 to 2022.
- Insights derived based on both number of bytes and number of commits.

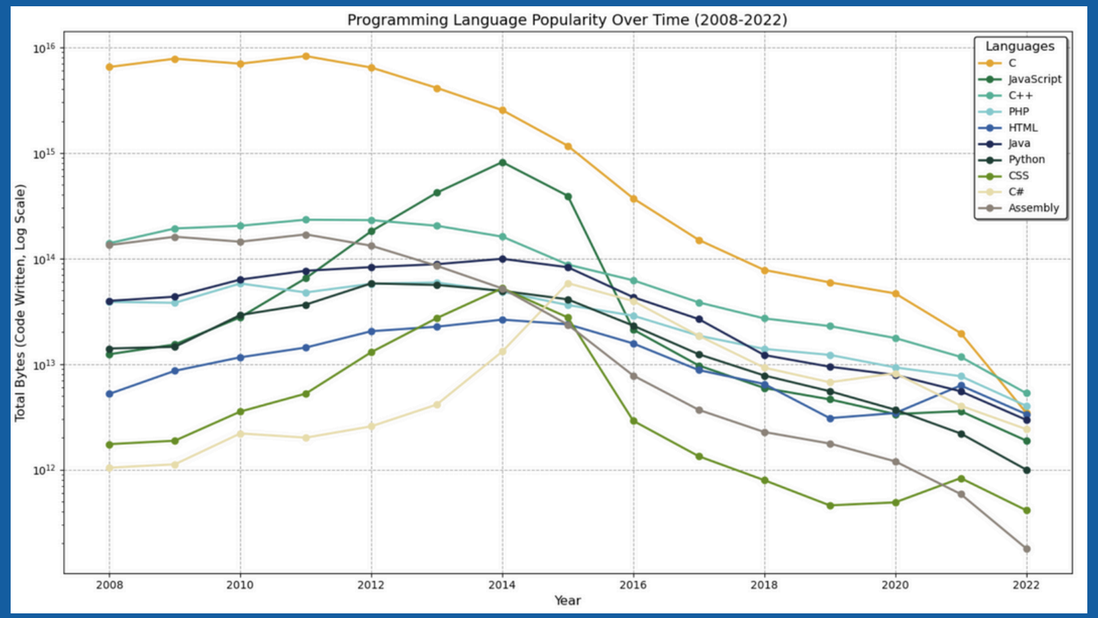


Fig 3.1: Programming Language Popularity Over Time (2008–2022) by Bytes

Objective: Trend analysis for top 10 languages based on bytes of code.
Approach: Applied **logarithmic scale** on the y-axis for better visualization of extreme peaks (e.g., C).
Insights:

- Python, JavaScript, and C dominate by volume, with Python gaining prominence due to its role in data science and AI.
- Fluctuations in C and Java, highlighting their evolving adoption.

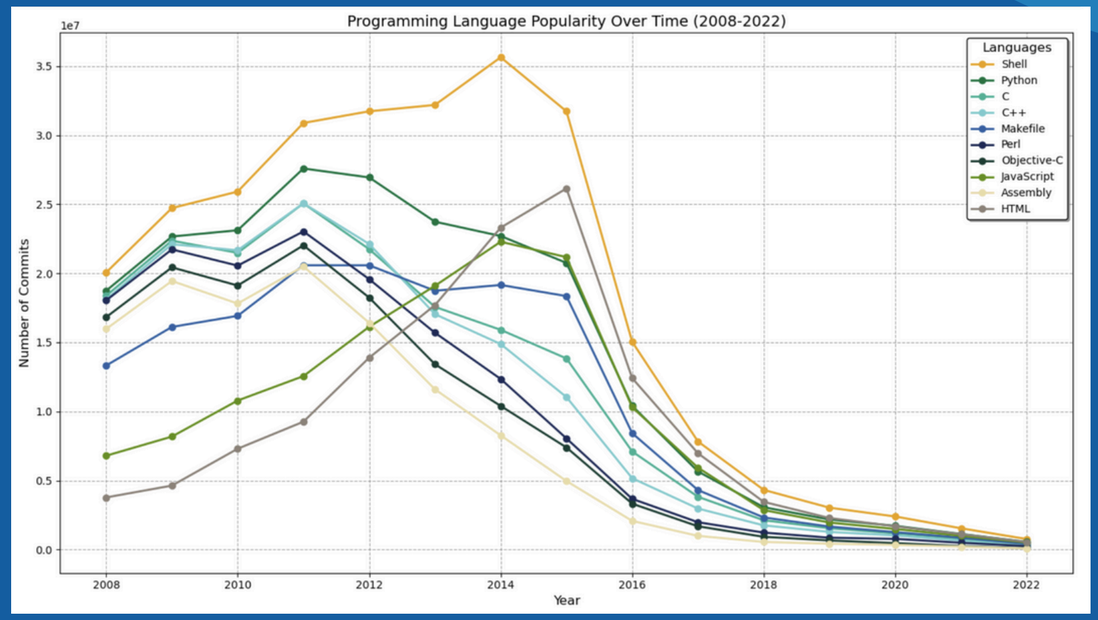


Fig 3.2: Programming Language Popularity Over Time (2008 - 2022) by Commits

Objective: Trend analysis for top 10 languages based on number of commits
Insights:

- Shell, Python, and JavaScript dominate commit activity.
- Shell usage peaked around 2013, followed by a steady decline.
- Python reflects its adoption for data science and AI.
- Popularity trends correlate with the introduction of technologies like TensorFlow (Python) and React (JavaScript).

LICENSE DISTRIBUTION

Overview

- Analyzed the distribution of licenses across GitHub repositories.
- Examined the correlation between programming languages and specific licenses.

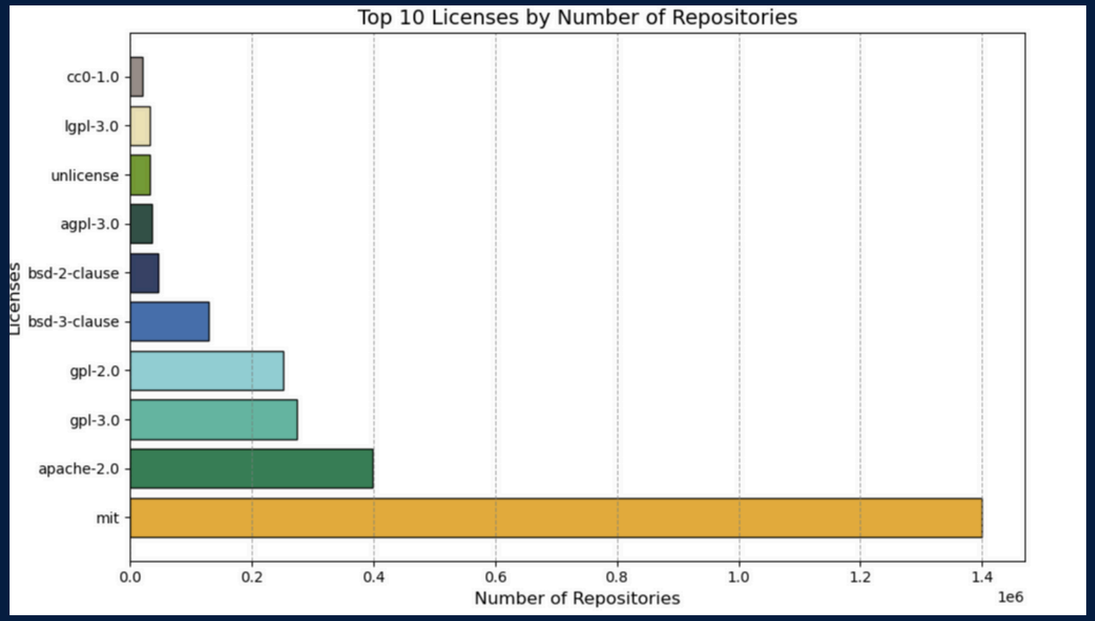


Fig 4.1: Top 10 Licenses by Number of Repositories

Fig 4.2: Top 10 Licenses vs Top 10 Programming Languages

Key Findings

1. Top Licenses by Usage:
   - MIT License is the most widely used, accounting for the majority of repositories.
   - Followed by Apache 2.0, GPL-3.0, and GPL-2.0 licenses.
2. Insights:
   - Permissive licenses (e.g., MIT, Apache) dominate, promoting open collaboration and faster adoption.
   - Stricter licenses (e.g., GPL) are still significant for legacy and enterprise systems.

Key Findings

1. Language-License Correlation:
   - JavaScript, CSS and HTML heavily use the MIT License.
   - C and C++ are strongly associated with GPL licenses, likely due to their historical roots in open-source systems.
2. Insights:
   - Permissive licenses like MIT dominate newer languages like Python and JavaScript.
   - Stricter licenses like GPL remain common for legacy systems written in C and C++.

REPOSITORY POPULARITY AND GROWTH

Overview

- Analyzed the most popular and rapidly growing repositories by total commits.
- Explored the technologies driving growth and Big Tech's influence in open-source contributions.

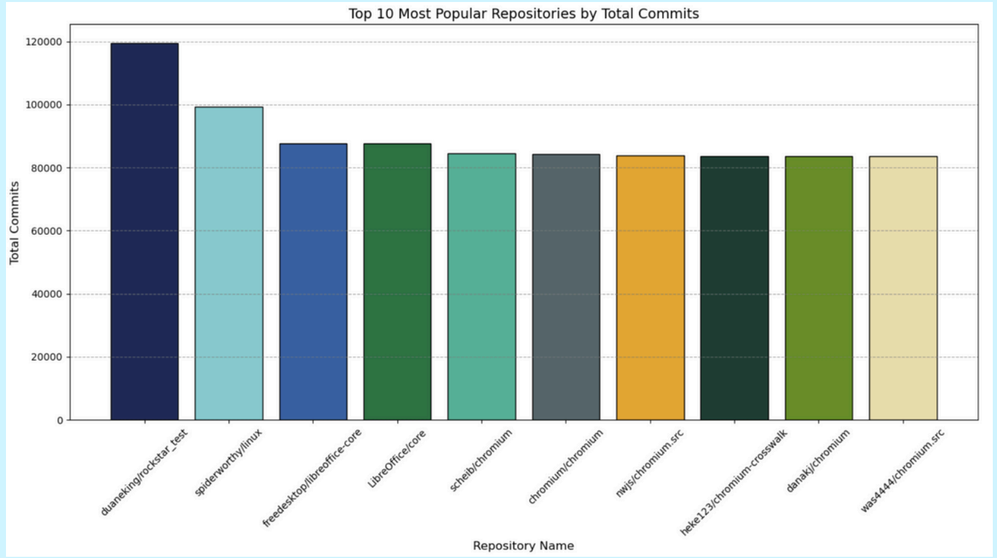


Fig 5.1: Top 10 Most Popular Repositories by Total Commits.

- The repository duanekin/rockstar_test leads with the highest total commits, followed by spiderworthy/linux and freedesktop/libreoffice-core.
- Several repositories associated with popular projects like Chromium (e.g., scheib/chromium, nwjs/chromium.src) feature prominently.
- High activity reflects the demand for cross-platform, scalable, and open-source solutions.

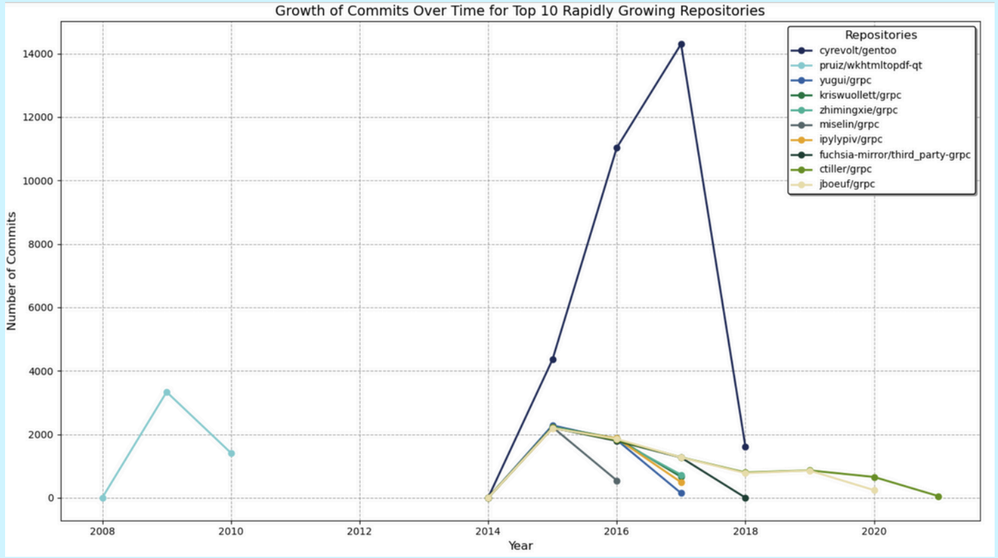


Fig 5.2: Growth of Commits Over Time for Top 10 Rapidly Growing Repositories

Approach:

- Calculating the percentage growth rate for each repository using the previous year's commit count.
- Selecting the top 10 repositories with the highest maximum growth rates.

Insights

- Technology Influence: Many repositories are linked to gRPC, a modern high-performance RPC (Remote Procedure Call) framework. This highlights the rise of microservices architectures and distributed systems.
- Big Tech Influence: Repositories like gRPC are closely tied to contributions from companies like Google, which heavily promote open-source adoption.

TECHNOLOGIES IN DATA SCIENCE/AI PROJECTS

Overview

- **Explored technologies most frequently associated with Data Science and AI projects.**
- **Analyzed trends in these technologies over time to identify changes in adoption and usage.**

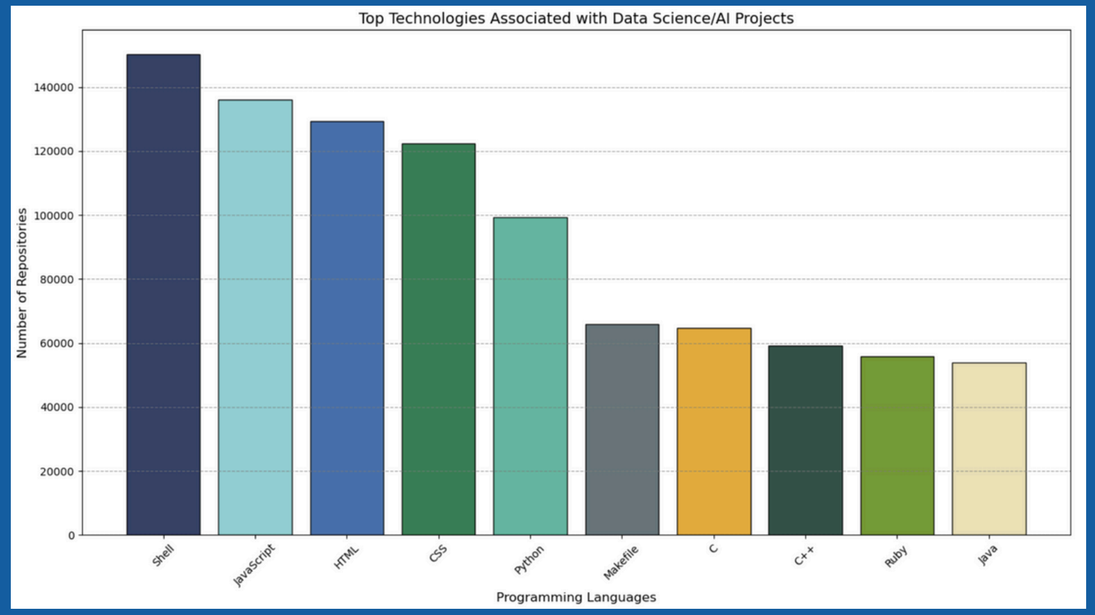


Fig 6.1: Top Technologies Associated with Data Science/AI Projects

- Repositories are being filtered using keywords such as "AI", "ML", "Deep Learning", and related terms found in their subjects or messages.
- keywords = ["AI", "ML", "Deep Learning", "Data Science", "Neural Network", "Machine Learning", "Artificial Intelligence", "Big Data","Data Mining", "Data Analytics", "Analytics", "Data", "Business Intelligence"]
- Python is prominent for its robust libraries (e.g., TensorFlow, PyTorch) and increasing use in machine learning projects.
- Shell and HTML: Early adoption driven by data preprocessing and web interface needs.

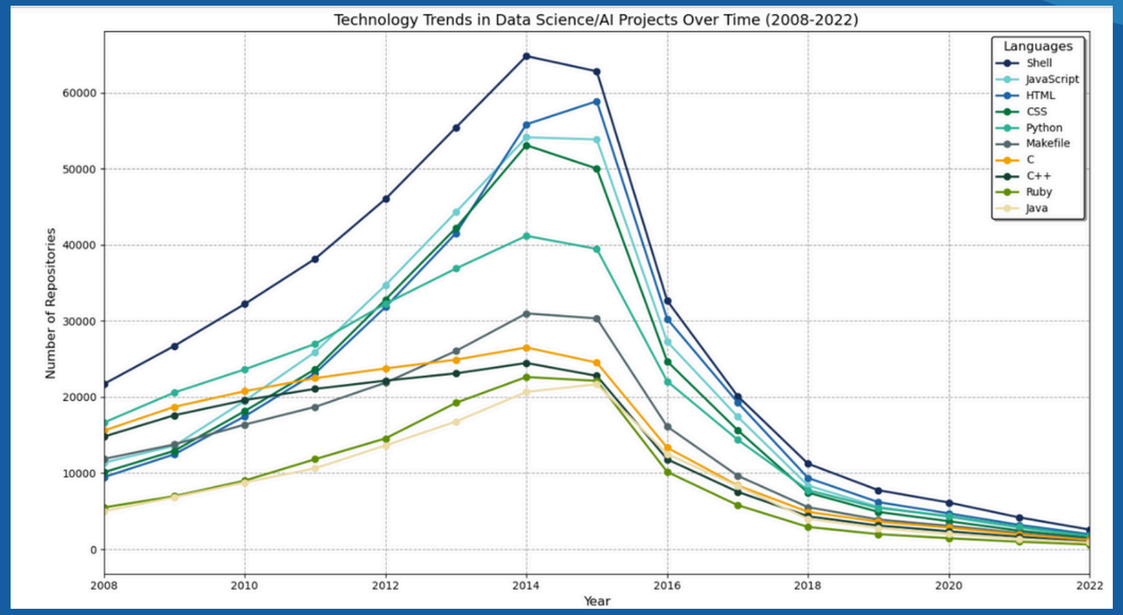


Fig 6.2:Technology Trends in Data Science/AI Projects Over Time (2008–2022)

- Exponential growth in repositories using Python, CSS and JavaScript during 2008–2014, reflecting advancements in AI frameworks and web technologies.
- Significant peaks between 2010–2015 for all technologies.
- Decline after 2016, possibly due to market stabilization and competition with new technologies.

REASONS FOR GITHUB COMMITS

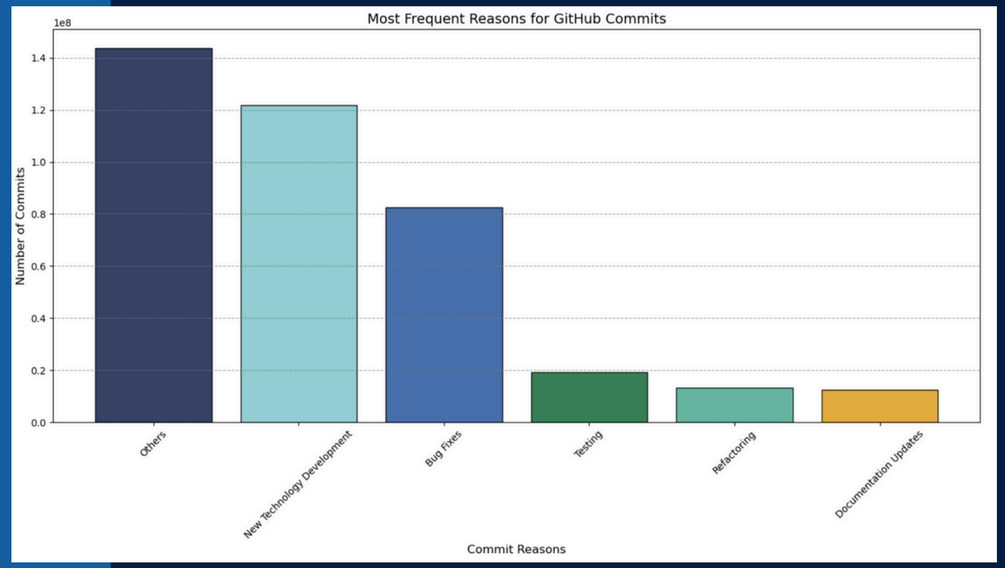


Fig 7.1: Most Frequent Reasons for GitHub Commits (all categories)

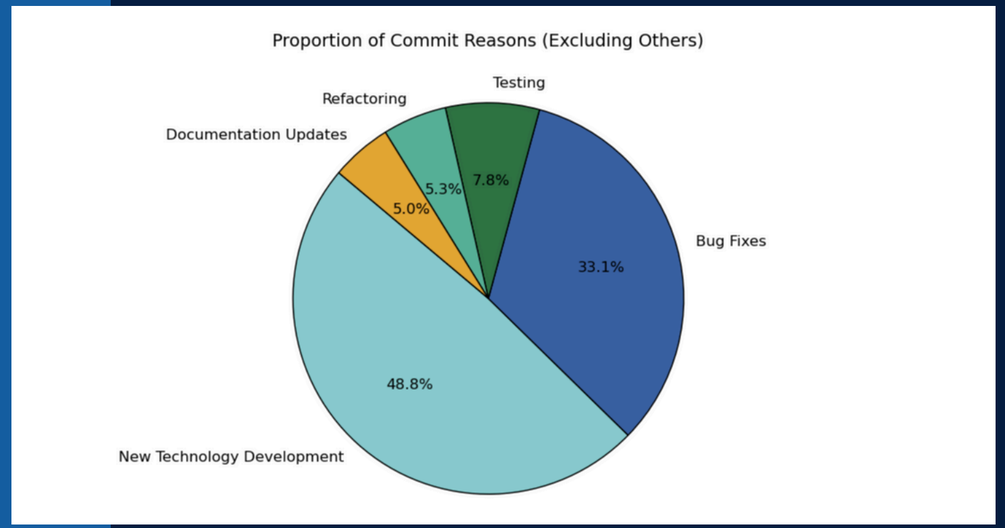


Fig 7.2: Proportion of Commit Reasons (excluding "Others")

Objective

- Identify and classify the most frequent reasons for commits in GitHub repositories.
- Analyze commit messages to determine if the commits focus on new technology development, bug fixes, documentation updates, refactoring, or testing.

Approach

Categorization:

- Commit messages were classified into six categories using keyword matching:
- New Technology Development: Keywords like add, create, develop, initialize, build.
- Bug Fixes: Keywords like fix, resolve, bug, patch.
- Documentation Updates: Keywords like doc, readme, guide, manual.
- Refactoring: Keywords like refactor, improve, restructure, optimize.
- Testing: Keywords like test, pipeline, automate.
- Others: Commits not matching any category.

Insights

1. Key Focus Areas:
   - Nearly half of all commits relate to new technology development, highlighting innovation and feature building as a top priority for GitHub contributors.
   - A significant share (~33%) of commits address bug fixes, showing an ongoing focus on improving software reliability.

2. Smaller Contributions:
   - Activities like testing, refactoring, and documentation updates are important but represent a smaller fraction of the overall commit volume.

3. Development Trends:
   - Commit activities reflect a balance between innovation (new technology) and maintenance (bug fixes, refactoring).

PROLIFIC AND INFLUENTIAL COMMITTERS

Objective

- Identify the most prolific and influential committers on GitHub by analyzing commit volumes.
- Visualize the distribution of commits over time and overall volumes.

Approach

1. Commit Volume Analysis:
   - Ranked committers based on total commit volumes from 2008–2022.
   - Focused on the top 10 committers with the highest activity.
2. Trend Analysis:
   - Tracked yearly commit volumes for the top 10 committers to identify trends over time.

Insights

1. Top Committers:
   - Linus Torvalds leads with a staggering commit volume of over 16 million, reflecting his contributions to foundational projects like Linux.
   - Other significant contributors include David S. Miller and Ingo Molnar, known for their work in open-source kernel and networking projects.
2. Trend Analysis:
   - Most top committers show peak activity between 2008–2014, with a decline in later years as projects matured.
   - Linus Torvalds' activity declined after 2014, possibly due to a shift from direct contributions to project leadership.

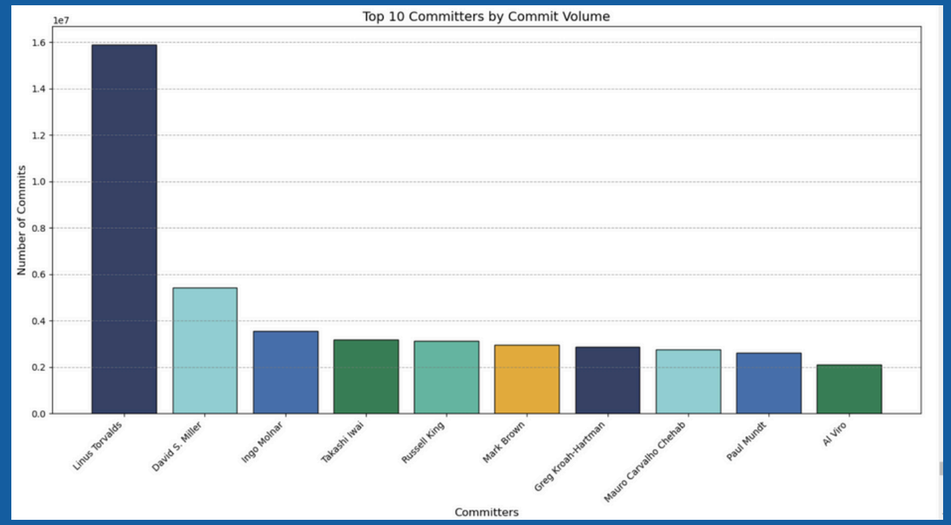


Fig 8.1 : Top 10 Committers by commit volume

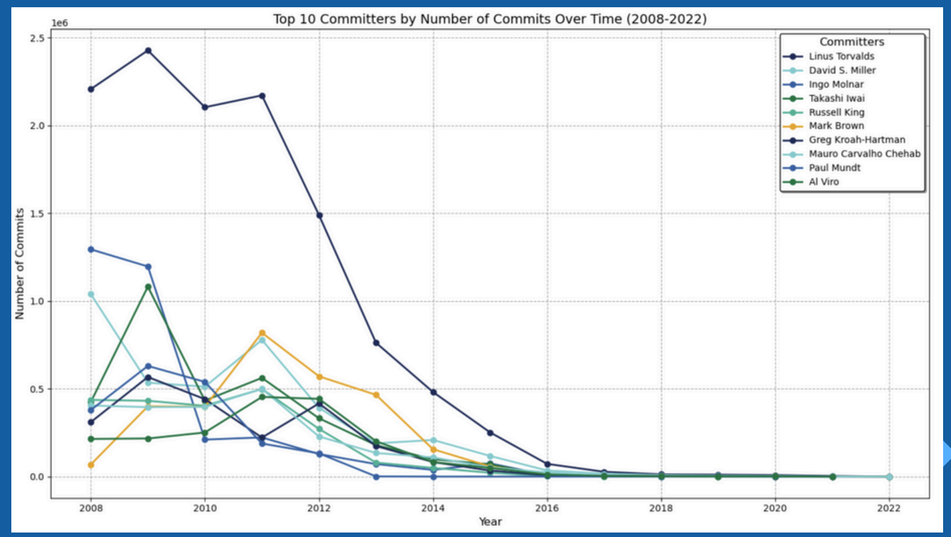


Fig 8.2 : Top 10 Committers by number of commits over time

SUBJECT AND MESSAGE SIMILARITY ANALYSIS

Overview

To determine the similarity between the subject and message fields in commit records, identifying whether values are unique or repetitive.

Graph 1: Similarity Across All Records

Approach:

- Used cosine similarity to classify all records in the dataset based on text similarity across fields.
- Records with a similarity score above 0.7 were labeled as "similar", while the rest were classified as "unique".

Insights

- Approximately 2x more similar records than unique ones were identified, indicating substantial text redundancy across fields.
- The high similarity suggests repetitive workflows or the use of automated commit practices in large-scale collaborative projects.

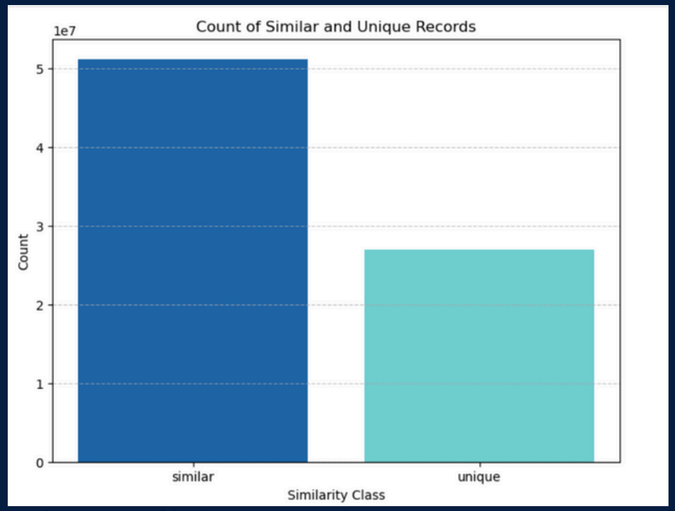


Fig 9.1: Count of Similar and Unique Records Across All Fields.

Graph 2: Duplication Between Subject and Message Fields

Approach

- Focused on comparing the subject and message fields using cosine similarity.
- Applied a stricter threshold of 1.0, where only exact matches between subject and message were classified as "similar".

Insights

- The analysis showed that the majority (~85%) of subject and message pairs are unique, highlighting diverse developer practices.
- The remaining ~15% of exact matches reflect automation or repetitive workflows, such as routine updates, bug fixes, or commit messages generated by tools.

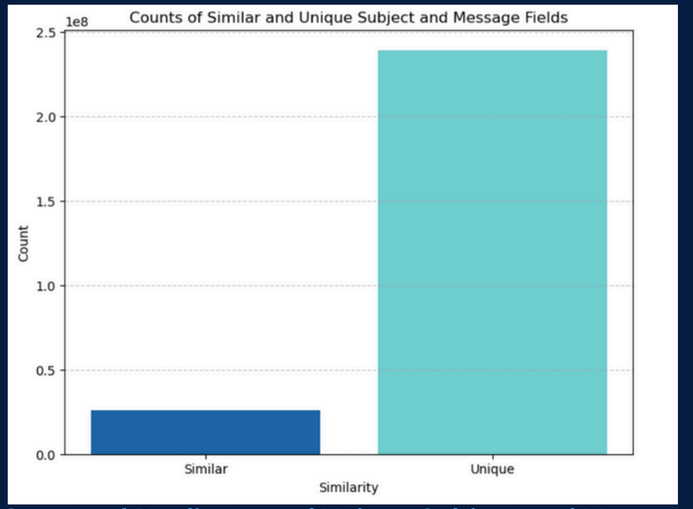


Fig 9.2: Counts of Duplicate and Unique Subject and Message Fields.

SUBJECT AND MESSAGE SIMILARITY ACROSS TOP PROGRAMMING LANGUAGES

Overview

- Analyze the similarity between commit subject and message fields across the top 10 programming languages by byte usage.
- Classify commits as “similar” (cosine similarity > 0.7) or “unique” (cosine similarity ≤ 0.7).

Steps

1. Top 10 Languages: Identified by total byte usage from languages_filtered.
2. Preprocessing: Cleaned, tokenized, and vectorized subject and message fields.
3. Cosine Similarity: Calculated similarity scores between TF-IDF vectors.
4. Classification: Labeled commits as "similar" or "unique" based on thresholds.

Insights

1. Object-Oriented Programming (OOP) Languages

Languages: Java, C++, Python, C#

Observation: Balanced distribution of unique and similar messages.

Reason: Use in diverse tasks and frameworks allows both standardized and descriptive commits.

Front-End and Web Development Languages

Languages: HTML, CSS, JavaScript

Observation: High similarity rates, with HTML showing the lowest uniqueness.

Reason: Templated workflows and automation tools (e.g., Prettier, Git hooks) dominate.

Scripting and Low-Level Languages

Languages: PHP, Assembly

Observation: Assembly has higher uniqueness due to detailed low-level programming, while PHP shows lower uniqueness.

Reason: Assembly requires specific explanations, while PHP is often used in templated web development tasks

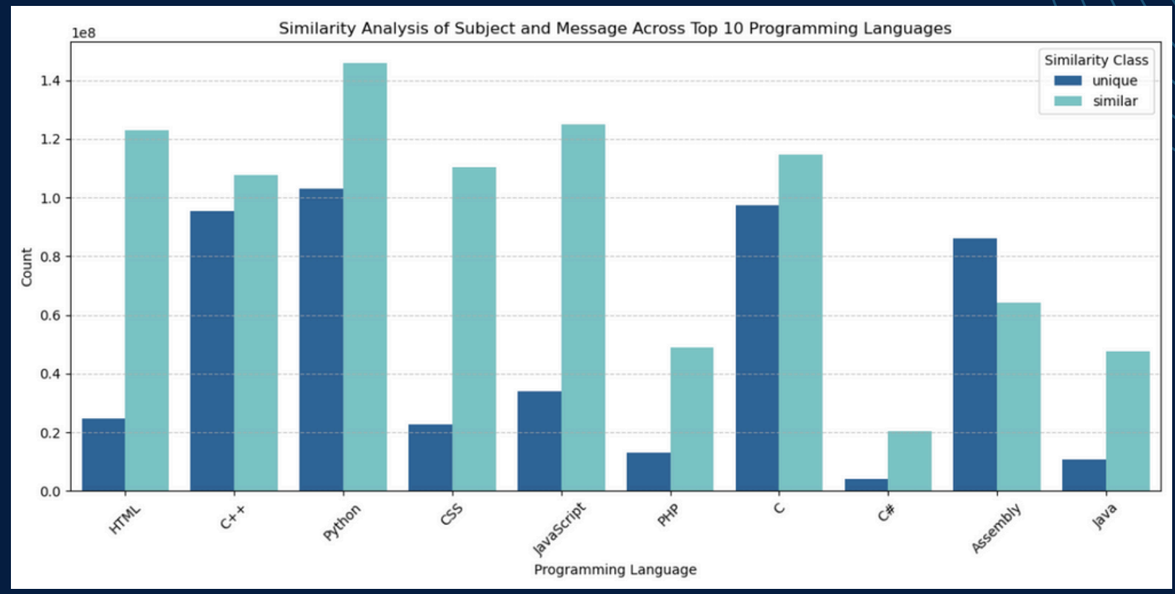


Fig 9.1: Similarity Analysis across Top 10 Programming Languages

CONCLUSIONS

01

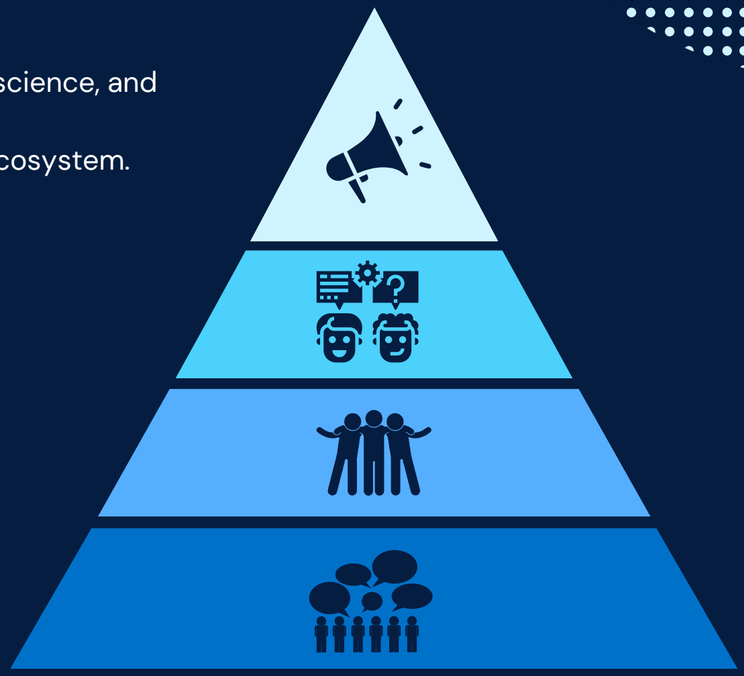
AI in Software Development:

- AI tools like GitHub CoPilot and ChatGPT significantly enhance developer efficiency by automating routine tasks but cannot fully replace human creativity and complex problem-solving.
- Their adoption is growing in areas like code generation, debugging, and documentation, showing their value in augmenting workflows.

02

Programming and Licensing Trends:

- Python, JavaScript, and C remain the most popular programming languages, driven by their utility in AI, data science, and web development.
- The dominance of permissive licenses like MIT highlights a preference for open collaboration in the GitHub ecosystem.



03

Big Tech's Influence:

- Open-source projects from Big Tech companies, such as TensorFlow and Chromium, drive technological breakthroughs and repository growth.
- These contributions foster rapid adoption of modern tools and frameworks.

04

Commit Patterns and Developer Focus:

- New technology development (48.8%) and bug fixes (33.1%) dominate commit reasons, reflecting a balance between innovation and maintenance.

ACTIONABLE RECOMMENDATIONS

FOR DEVELOPERS:

Leverage AI Tools:

- Use AI tools for routine tasks like debugging, code reviews, and documentation to save time and focus on complex problem-solving.

Enhance AI Skills:

- Familiarize yourself with AI frameworks and integrate them into your workflows to stay competitive in the evolving software landscape.

FOR ORGANIZATIONS:

Adopt AI-Augmented Development:

- Invest in AI tools to increase team productivity and streamline software delivery timelines.
- Provide training for developers to maximize AI tool adoption and usage.

Encourage Open Source Contributions:

- Collaborate with open-source communities to drive innovation and benefit from collective expertise.

Balance Human Creativity with AI:

- Use AI tools as a complement, not a replacement, for human ingenuity in software design and architecture.

THANK YOU