The h-index Metric for GitHub

Adapting academic impact measurement for developer influence

Mykhailo Pavlov • Comenius University, Bratislava

Supervisor: Mgr. Marek Šuppa • Comenius University, Bratislava

The h-index: A Standard for Scientific Impact

Proposed by Hirsch in 2005, the h-index has become the standard metric for measuring researcher influence in academia.

Simple Definition

A scientist has index h if h of their papers have at least h citations each

Example

h = 10 means 10 papers with at least 10 citations each

Interpretable

Simple, interpretable, and widely adopted in academia

Why GitHub Lacks a Standard Impact Metric

GitHub hosts millions of developers, but lacks a unified way to measure influence.

Decentralized Platform

Activity is spread across repositories, organizations, and contributions

Current Proxy

Common proxy: follower count, but this measures popularity, not code quality

The Gap

Need a simple, interpretable metric grounded in actual technical impact

Academia

Papers + Citations

GitHub

Repositories + Stars

From Followers to Stars: A Better Metric

The Problem

A user can have 10,000 followers but few starred repositories. Followers reflect social reach, not code quality or reuse

New Mapping

Articles → Public repositories,

Citations → Repository stars

The Solution

Adapt the h-index to GitHub

GitHub h-index

Largest h where a user has h repositories with at least h stars each

The GitHub h-index: Core Definition

Adapted from the academic h-index, the GitHub h-index measures developer impact through repository popularity rather than research citations.

Analogy to Academic h-index: Researcher has h papers with h citations each

GitHub: Developer has h repositories with h stars each

$$h_G = \max\{i \in \{1, \ldots, n\} : s_i \geq i\}$$

Where s_*i* represents stars on the i-th repository (sorted by stars in descending order), repositories replace publications, and stars replace citations.

Example: Calculating h-index Step by Step

Consider a developer with five repositories sorted by star count:

Repository	Stars	Check: ≥ Position?
web-app	12	Yes (12 ≥ 1) ✓
api-service	8	Yes (8 ≥ 2) ✓
data-tool	5	Yes (5 ≥ 3) ✓
cli-utils	3	No (3 < 4) ✗
config-lib	1	No (1 < 5) ✗

Result

h-index = 3

The developer has 3 repositories with at least 3 stars each. The 4th repository has only 3 stars, which is less than 4.

Data Collection at Scale: 3.5 Million GitHub Users

A large-scale data collection effort that efficiently gathered information from millions of GitHub profiles using GraphQL technology.

Why GraphQL API (not REST)

- Fetch multiple resources in a single query
- Reduces total API calls by ~70%
- Essential for collecting repositories, stars, and profiles efficiently

Scale Achieved

- ~3.5 million users collected
- All public repositories and star counts retrieved
- h-index computed for each user

Challenge

GitHub API has strict rate limits per token per hour

Reliable Data Collection Under API Limits

TokenManager – Multi-Token Rotation

- Implemented a class to rotate among ~8 authentication tokens
- Each token checked via HTTP headers before use
- On 403 error, automatically switch to next available token
- Effectively multiplies daily API quota

Checkpoint System – Fault Tolerance

- Save progress to disk after each batch
- If connection fails or program stops, restart from last saved point
- Prevents re-collecting already-processed users

Parallelization – Speed Optimization

- Multiple tokens used simultaneously in parallel threads
- Batch processing: collect users → save → continue
- Reduced total collection time from weeks to days

How We Collected Users and Computed h-index

User Enumeration Strategy

- GraphQL returns max 1000 users per query (10 pages × 100 users)
- To collect broadly, we batched by: Follower count (high to low) and Registration date (chronological batches)
- Prioritized active/popular users; some zero-follower users excluded

Repository Parsing and h-index Calculation

- For each user: fetch all public repositories via GraphQL
- Extract star count for each repository
- Sort repositories by stars descending
- Apply formula: $h_G = \max\{i : s_i \geq i\}$
- Store result with user profile

Error Handling

- Handle deleted accounts gracefully (skip and log)
- Retry failed requests with exponential backoff

Enriching Profiles for Machine Learning

13 Features Collected for ML Model

Repository Metrics

repo count, total stars, avg stars, median stars, max stars, star variance, stars per repo

Social Metrics

followers count, following count, followers per repo, reciprocity ratio

Profile Metadata

organizations count, bio length

Social Network Links Extraction

- Step 1: Direct fields via GraphQL (websiteUrl, twitterUsername)
- Step 2: Regex on bio, company, location to find URLs/emails
- Step 3: Fallback to public HTML profile page if needed
- Classify links: Twitter/X, LinkedIn, Instagram, YouTube, Mastodon, Telegram

Why Random Forest for h-index Prediction?

GitHub h-index follows a heavy-tailed distribution with many low values and few extreme highs. Random Forest handles these outliers and non-linear relationships better than linear models.

Heavy-Tailed Distribution

GitHub h-index follows a heavy-tailed distribution. Many low values, few extreme highs

Handles Non-Linearity

Random Forest handles outliers and non-linear relationships better than linear models

Feature Importance

Provides built-in feature importance. Helps interpret which factors matter most

Robust to Skew

Robust to skewed data without extensive preprocessing

Linear Model

Struggles with extremes

Random Forest

Handles tail values well

13 Features Across Three Dimensions

Repository Metrics

total stars, avg stars, media stars, max stars, star variance, stars per repo, repo count

Social Metrics

followers count, following count, followers per repo, reciprocity ratio

Profile Metadata

organizations count, bio length

- All numeric features log-transformed to stabilize variance where needed.

Feature	Transform	Interpretation
`repo_count`	None	Repository volume
`total_stars`	$\log(1+x)$	Cumulative code impact
`avg_stars`	$\log(1+x)$	Typical repository popularity
`median_stars`	$\log(1+x)$	Robust central tendency
`max_stars`	$\log(1+x)$	Peak impact repository
`star_variance`	$\log(1+x)$	Consistency of impact
`stars_per_repo`	$\log(1+x)$	Normalized engagement
`followers_count`	None	Social reach
`following_count`	None	Community participation
`followers_per_repo`	$\log(1+x)$	Reach per project
`following_to_followers`	$\log(1+x)$	Reciprocity ratio
`organizations_count`	None	Institutional affiliation
`bio_length`	$\log(1+x)$	Profile completeness

Training Setup and Model Performance

Results on Test

Performance of baseline Random Forest model (raw h-index target)

Metric	**Overall**	**Top 10% ($h \geq 3.0$)**
MAE	0.106	0.558
R^2	0.9486	0.8696
Sample size	734,299	87,373 (11.9%)

Performance of Random Forest model with log2-transformed target

Metric	**Overall**	**Top 10% ($h \geq 3.0$)**
MAE	0.109	0.595
R^2	0.9410	0.8459
Sample size	734,299	87,373 (11.9%)

Training Configuration

Dataset: 917,874 users after filtering, removed users with zero stars/repos

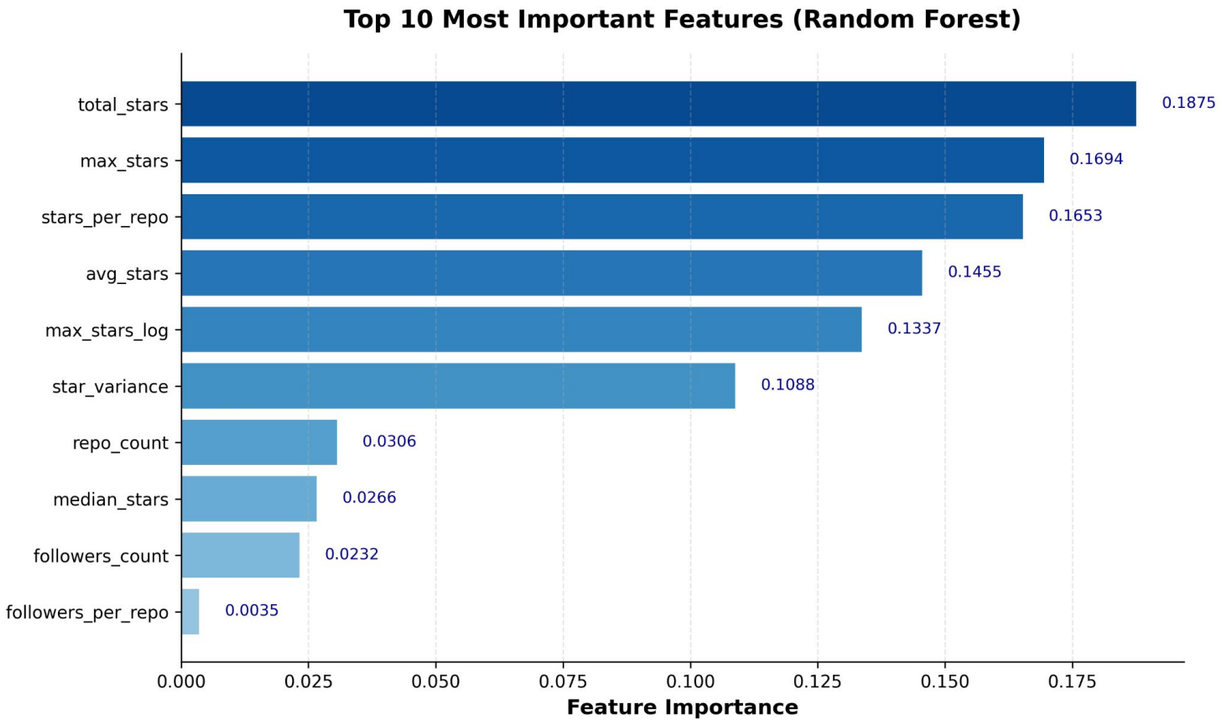
Split: 80% training, 20% testing, fixed random seed

Parallelized training across CPU cores: ~2.5 minutes per model

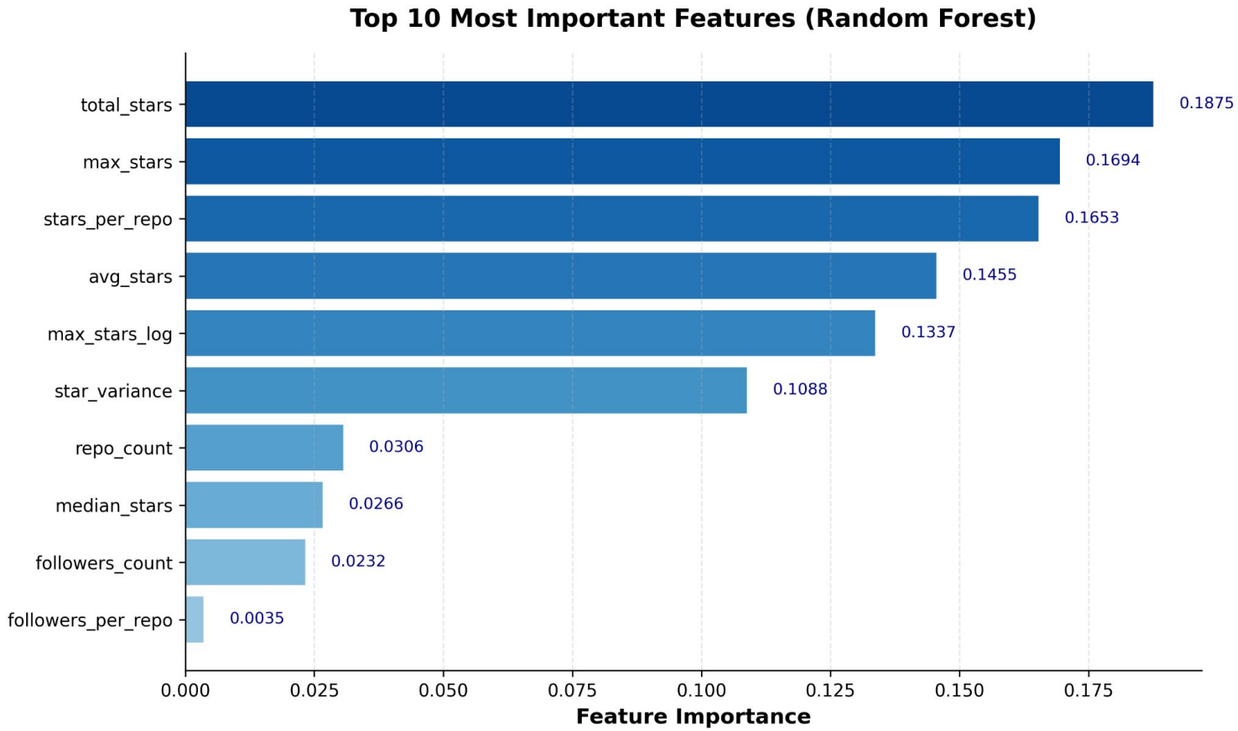
What Drives the GitHub h-index?

Top Predictive Features

Baseline model



Log2-model



Key Takeaway: Code impact drives the GitHub h-index, not social popularity.

What Drives the GitHub h-index?

Key Finding from Feature Importance

- Total stars is the dominant predictor — by far the strongest signal in the model
- Repository-level metrics follow: stars per repo, average stars, median stars
- These metrics reflect consistent code impact across multiple projects

Surprising Result

- Social features (followers, following) have very low predictive power

Case Studies: High-Impact Users Are Under-Predicted

Representative Examples from Test Set

User	True hG	Predicted	Error	Profile Note
A	253	63.7	-189	11k repos, 77k followers
B	151	58.4	-92.6	874 repos, 23k followers
C	136	71.2	-64.8	299 repos, 75k followers

Pattern

- Users with extreme h-index values are consistently under-predicted
- Error grows with true h-index magnitude
- Model is reliable for hG < 30, cautious beyond that

Key Takeaway

For very high-impact developers, use predictions as a lower bound.

Social Network Presence Among GitHub Developers

Understanding how developers connect beyond code

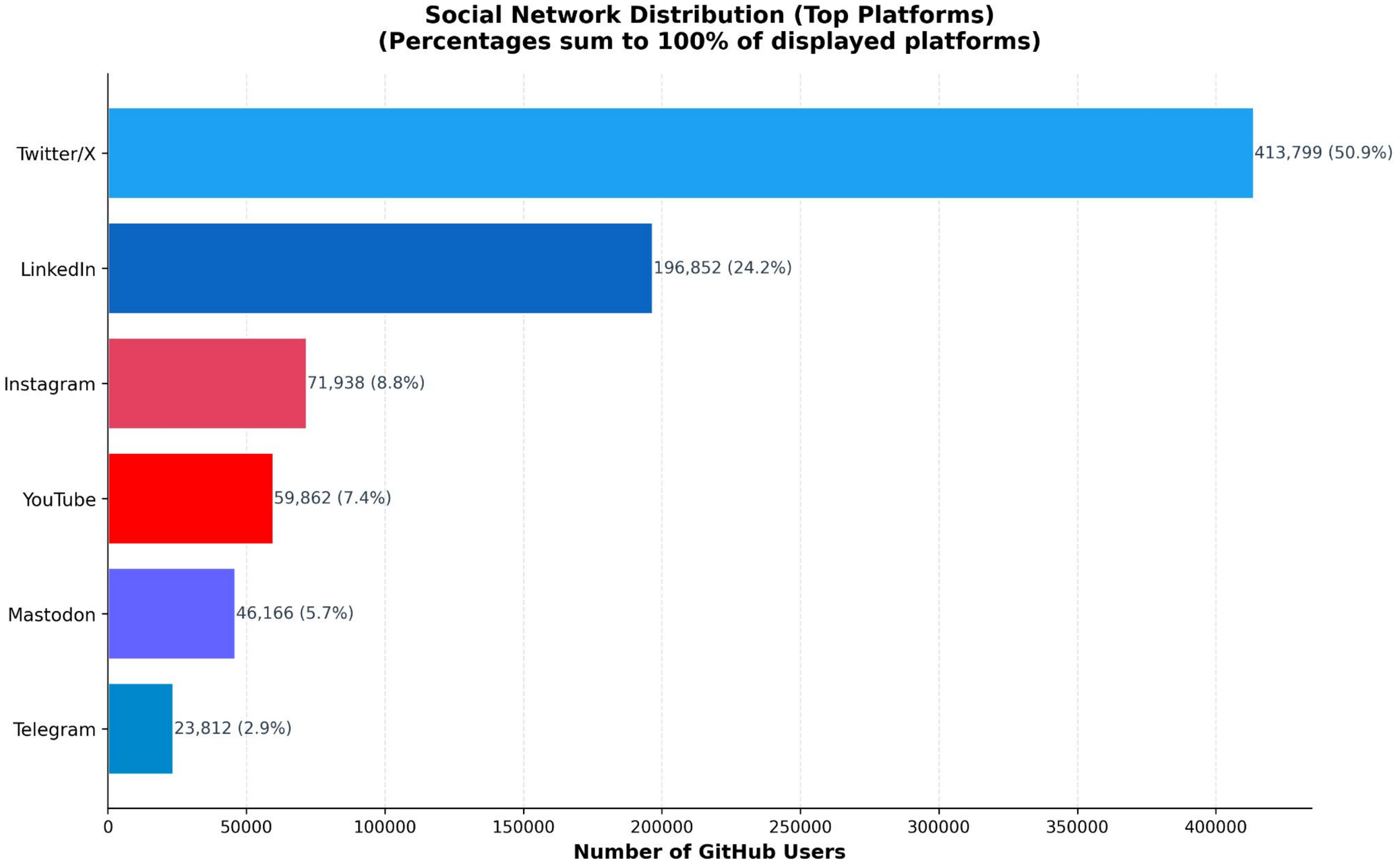
Method

Extracted links from 3.5M user profiles using GraphQL + regex + HTML fallback. Analyzed 812,429 users with at least one linked social platform.

Key Findings

- Twitter/X: 50.9% — most popular platform for developers
- LinkedIn: ~25% — professional networking focus
- Others: YouTube, Mastodon, Telegram — small but present

Takeaway: GitHub developers prioritize professional and tech-focused networks.


Social Network Distribution (Top Platforms)
(Percentages sum to 100% of displayed platforms)
Twitter/X
LinkedIn
Instagram
YouTube
Mastodon
Telegram
413,799 (50.9%)
196,852 (24.2%)
71,938 (8.8%)
59,862 (7.4%)
46,166 (5.7%)
23,812 (2.9%)
0
50000
100000
150000
200000
250000
300000
350000
400000
Number of GitHub Users
Total users on chart: 812,429

Conclusion and Future Directions

Contributions

- Scalable GitHub data collection system (3.5M users)
- Validated GitHub h-index metric: repositories → stars
- Random Forest model: R^2 = 0.9486, MAE = 0.106
- First large-scale social network mapping for GitHub devs

Limitations

- Model underestimates extreme h-index values
- Data collection prioritized popular users

Future Work

- Deep learning / quantile regression for tail values
- Temporal features: contribution velocity, star growth
- Expand to PRs, reviews, contribution graphs

Thank you for your attention