DOI: https://doi.org/10.15276/aait.09.2026.39
UDC 004.41:004.85

Ranking pull request reviewers by expertise, workload, and expected response time

Oleksandr S. Aratovskyi[1)]
ORCID: https://orcid.org/0009-0006-0854-8582; alex.aratovskiy@stud.op.edu.ua. Scopus Author ID: 60503533100
Vira V. Liubchenko[1), 2)]
ORCID: https://orcid.org/0000-0002-4611-7832; lvv@op.edu.ua. Scopus Author ID: 56667638800
[1)] Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine
[2)] University of Applied Sciences, 20, Ulmenliet. Hamburg, 21033, Germany

ABSTRACT

Relevance. Reviewer rankings are usually judged by whether the suggested person later reviewed the pull request, which says little about how widely recommendations spread or how fast that person responds. **Purpose of the article.** We ask how accurately reviewers can be ranked when a pull request is created, and what accounting for concentration and expected response time costs in accuracy. **Objectives.** We build candidate pools from prior reviews, compare a learned ranker with simple rules, test it on unseen repositories, and compare two ways of balancing other criteria. **Methods.** We use merged pull requests from twelve open-source repositories. Chronological splits limit each recommendation to earlier information. We train a gradient-boosted ranker, measure whether an actual reviewer appears among the first three candidates, and calculate bootstrap confidence intervals. **Scientific novelty.** We compare a weighted penalty with a rule that reorders only candidates near the top relevance score, reporting accuracy, the spread of first choices, and estimated response time together. **Practical significance.** A model trained elsewhere can serve a new project once it has its own review history, and a tolerance parameter bounds departures from the relevance-only ranking. **Results.** On the test set, the ranker's first three choices included an actual reviewer for about two-thirds of pull requests, against under half for the most-active rule; ranking the author's past reviewers already captures most of that gain. In leave-one-repository-out tests, accuracy fell by under one percentage point. The tolerance rule put a third more people in first place or nearly halved the first choice's median estimated response time, each for under one percentage point of accuracy. No nonzero weighted penalty met the validation accuracy threshold. **Conclusions.** Balancing workload or expected response time costs little within a tolerance of the best relevance score and much when applied to every candidate. Whether this shortens actual waiting time remains to be tested in a live pilot.

Keywords: Code review; reviewer recommendation; pull request; ranking model; reviewer workload; cross-project transfer

For citation: **Aratovskyi O. S., Liubchenko V. V. "Ranking pull request reviewers by expertise, workload, and expected response time". *Applied Aspects of Information Technology*. 2026; Vol. 9 No. 4: 601–614. DOI: DOI: https://doi.org/10.15276/aait.09.2026.39**

INTRODUCTION

Code review in its modern, tool-based form catches defects, spreads knowledge of the code base, and keeps a team's conventions alive [1], [2], [3]. On GitHub, the unit of review is the pull request (PR): a change that someone has to inspect, perhaps send back, and approve before merging [4].

In earlier work, we defined the efficiency of code review through three characteristics derived from a formal process model: the time until the first reviewer reaction, the number of review iterations, and the risk that a PR goes stale [5]. Each of them depends on who reviews. A reviewer who knows the touched subsystem is more likely to find the real problems in one pass; one buried under a dozen open reviews is likely to answer late and the waiting PR may go stale. Choosing the reviewer is one of the few levers a project can pull at submission time.

Authors ask the colleague they asked last time, or the most visible maintainer. Based on data from this study, the reviewer most active in the previous ninety days was the first-ranked guess in only a quarter of the evaluated pull requests (25.3 %), and that share ranged from 1.4 % in Grafana to 66.4 % in Django. Where it is high, the project has one reviewer in all but name. Where it is low, history alone does not say whom to ask. A recommender that merely reproduces the habit inherits its cost: the same few people absorb most of the load, and their response time becomes the projects.

Reviewer recommendations have a decade of literature behind them. The early methods rank candidates by expertise on the changed files [6], [7], [8]; later ones add workload and availability as competing objectives and search for a compromise per pull request [9], [10], [11], [12], [13]. Industrial deployments report that back-tested accuracy does not translate directly into a faster review, and that assigning a named individual rather than a team shortens review time [14], [15]. Three questions stay open, and the problem statement below spells them out: whether the reported accuracy rests only on what existed when the PR was created, what

© Aratovskyi O., Liubchenko V., 2026

This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/deed.uk)

balancing the load or the response time costs in accuracy, and whether a ranker learned in some projects carries over to others.

This paper ranks reviewers for a pull request at the moment of its creation, using only what was known then, and measures the accuracy cost of balancing workload and expected response time. To that end, the article does three things. It formulates reviewer ranking as a learning-to-rank problem over features computed strictly from history before the PR creation time, and it measures with an ablation how much accuracy comes from PR content that is only approximately known at that time. It compares two ways to steer the ranking away from concentrated and slow choices: a weighted penalty on workload and expected response time versus a re-ranking rule that acts only within a relevance tolerance, and reports the resulting trade-off between reproduction accuracy, concentration, and expected latency with bootstrap confidence intervals on twelve open-source repositories. It tests whether a ranker learned on eleven repositories transfers to a twelfth it has never seen.

The paper is structured as follows: the related works section reviews reviewer recommendation and ends in the problem statement, the aim, and the objectives; the research methodology section defines the task, data, features, models, baselines, and validation protocol; the results section reports accuracy, transfer, and the trade-off; the discussion interprets them against prior work and lists the threats to validity; the last section concludes.

RELATED WORKS

The first generation of reviewer recommenders equated the right reviewer with the most experienced one. RevFinder ranks candidates by how similar the file paths of the new change are to those of changes they reviewed before [6]; cHRev profiles a reviewer by past review comments, weighted by recency and breadth [7]; CORRECT infers competence from work with the same libraries or technologies in other repositories [8]. Later variants combined file similarity with the comment network of GitHub projects [16], showed that expertise measured from past reviews recommends better than expertise measured from past modifications of the same files [17], or kept a profile per reviewer built from the file path segments of the changes that person had reviewed [18].

All of them share one definition of success: coincidence with the person who actually reviewed. The label costs nothing, but it rewards whatever assignment habit the project already has, so a model that learns to name the usual maintainer scores well and changes nothing.

Workload and availability as competing objectives. The second generation treats reviewer selection as an optimization over several objectives. Ouni and colleagues maximize expertise and prior collaboration with the author [9]; Rebai and colleagues add availability, measured through current load, as a third objective [10]. Sofia balances expertise, workload, and the distribution of knowledge so that the project survives turnover [11]. Al-Zubaidi and colleagues add an estimated chance of participation to a workload-aware search [19]; WhoReview trades expertise and collaboration against workload with an evolutionary algorithm [12]; a recent study looks at the risk of defects and the spread of knowledge together [20].

Response time entered the objectives with MOC2R, which ranks candidates by experience, by the probability of a response within a given time window, and by recent activity, and reports Top-1 accuracy between 41.7 and 61.5 % on six open-source projects under two-, four-, and eight-hour windows [13]. So the idea that a recommender should care how fast a reviewer answers is not new, even if availability, the chance of participation, and the probability of a fast response are different quantities.

Industrial evidence and the gap between back-testing and effect. Kovalenko and colleagues found at two companies that recommended reviewers largely coincided with the people authors would have chosen anyway [15]; Strand and colleagues evaluated an industrial context-aware recommender with practitioners rather than against history alone [21]. At Meta, Rigby and colleagues ran three A/B tests. A recommender built from features in the literature was more accurate than its predecessor; a workload-balanced variant produced mixed results; assigning a recommended individual instead of a whole team cut the time until review [14]. Back testing and live experiments can disagree, then, and the latency Meta measured is the recommender's computation time, not the reviewer's response time.

Responsiveness and completion time. Responsiveness has mostly been studied as an outcome, not as an input to recommendation. Thongtanunam and colleagues linked low review participation to properties of the change and of its author [22]; Ruangwan and colleagues found that whether an invited reviewer responds at all is associated with that reviewer's own history, the past participation rate, the current review load, and the familiarity with the author [23]; Chouchen and colleagues predict the completion time of an entire

review [24]. Our own prior work predicts at submission time whether a PR will need two or more review iterations and whether it will go stale; it derives these targets from a process model in which the reviewer is a separate lane, reaches an F1 score of 0.80 and an AUC-ROC of 0.93 for the iteration target, and transfers under leave-one-repo-out validation [5]; reviewer ranking was left there as an open direction. The datasets it rests on were described in [5], [25]. Surveys confirm that recommendation, participation, and outcome prediction are usually studied apart from each other [26], [27].

PROBLEM STATEMENT

Three things remain open. A leak-controlled evaluation first: a recommender that runs when a PR is created must be judged on what exists at that moment, whereas labels, the final diff size and file list, and the description are retrospective quantities whose contribution should be measured on its own. Then the trade-off itself. The literature does not report how much reproduction accuracy one gives up for a given reduction in concentration or expected waiting time, across many projects and with uncertainty. And transfer: every method above is trained and tested within the same projects, and whether a learned ranker carries over to a different review culture decides whether it is usable without a long history.

This article addresses the three data gaps underlying our process model. The contribution is the protocol, the ablation, and the measured trade-off; the idea of including responsiveness is not ours; earlier work introduced it.

The **aim of the study** is to rank the historical reviewers of a pull request at the moment of its creation, using only what was known at that moment, and to measure the trade-off between three properties of the resulting recommendation: how often it reproduces the reviewers who actually took part, how concentrated its recommendations are across the team, and how fast the recommended reviewers are expected to respond, judging by their own history. A secondary aim is to find out whether a ranker learned on one set of projects transfers to a project it has not seen.

The study does not set out to show that the recommended assignment would have shortened the review. Historical data hold the outcome only for the reviewers who were actually chosen; a causal effect needs a live pilot, and that lies outside this article.

To achieve the aim, the following objectives are set:

1. Profile the twelve repositories as a reviewer ranking task: the number of rankable pull requests after warm-up, the size of the candidate pool, the share of actual reviewers covered by the pool, and the concentration of review activity.
2. Establish baseline accuracy for rules that require no learning, including the most active, the most recent, the workload-corrected, and the fastest-responding candidate, with bootstrap confidence intervals per repository.
3. Train a learning-to-rank model on features computed strictly before the creation time, compare it with the baselines, and measure with an ablation how much accuracy depends on retrospective content features.
4. Introduce a re-ranking rule bounded by a relevance tolerance, select its tolerance and secondary criterion on validation data, and measure the trade-off between reproduction accuracy, concentration, and expected latency on held-out data.
5. Test cross-project transfer by training on eleven repositories and evaluating on the twelfth.

RESEARCH METHODOLOGY

Let p be a pull request created at time $t_0(p)$ by author $a(p)$ in repository q. Let $C(p)$ be the candidate reviewers active in q shortly before $t_0(p)$, $A(p)$ the people who actually took part in the review of p, and $R(p) = A(p) \cap C(p)$ the rankable actual reviewers. The ranking method is a scoring function $f(p, c)$, $c \in C(p)$, computed from information available at $t_0(p)$; the recommendation is the pool sorted by decreasing score, judged by the positions at which members of $R(p)$ appear. $R(p)$ records what the team did, not who the best reviewer would have been.

The learned component produces a relevance score $\hat{r}(p, c)$. The weighted-sum rule penalizes two further quantities available at $t_0(p)$, the current workload $w(p, c)$ and the expected response time $\tau(p, c)$ estimated from the candidate's own history, each replaced by its percentile rank $\rho(\cdot)$ within the pool of the same pull request:

$$s(p, c) = \rho(\hat{r}) - \lambda_w \cdot \rho(w) - \lambda_t \cdot \rho(\tau), \quad (1)$$

where $\hat{r}$, w, and τ refer to the pair (p, c), and $\lambda w \geq 0$ and $\lambda t \geq 0$ are weights chosen on validation data; with $\lambda w = \lambda t = 0$, the rule reduces to the learned ranker. A larger λt necessarily moves candidates with a smaller estimated τ upward, so a lower estimated latency of the top recommendation follows by construction; the metrics include a check that does not reuse the estimate.

The study uses the merged pull requests of the twelve open-source GitHub repositories listed in

Table 1, on which our process model and prediction framework were built [5]. We chose them for their review activity, native GitHub review workflow, and diversity of language, domain, and community structure; [5] describes the selection criteria and the dataset construction; the collection pipeline builds on [25].

The dump was collected through the GitHub GraphQL API in April 2026 and contains 72 571 merged pull requests, of which 67 427 have at least one review by a human account. For every PR, it holds the creation and merge times, the author, the change size and draft flag, the current labels, the reviews with author, state, and time, the review threads, and the reviewers requested at collection time. The dumps start at different dates, from 2014 for angular to November 2025 for vscode, because collection retrieved the most recent PRs first up to a per-repository budget (Table 1).

The dump has limitations that shape the design. It holds the number of changed files but not their paths, so file-path expertise in the sense of RevFinder cannot be computed. It lacks the time at which a review was requested, so the interval from PR creation to a candidate's first review includes the time before anyone asked. Labels, the description, and the draft flag are stored as of collection time. Closed but unmerged pull requests were not collected, so workload is underestimated. At most fifty reviews and fifty review threads per pull request were kept, which truncates a few unusually long reviews.

We count a person as an actual reviewer of p if that person submitted at least one review of any state or wrote at least one comment in a review thread of p, is not the author of p, and is not a bot. Bots are recognized by login markers such as the [bot] suffix and by the names of known automation and AI review services; a PR can have several actual reviewers, all relevant.

The candidate pool $C(p)$ holds every person who acted as a reviewer in the same repository at least once in the 180 days before $t_0(p)$, the author excepted. The window keeps the pool manageable and excludes contributors who have left. We report the shares of pull requests with at least one, and with all, actual reviewers inside the pool as the coverage of candidate generation; pull requests with none are excluded from training and evaluation and counted.

Because the features are computed from history, the first 90 days of every dump only accumulate it and are excluded from evaluation. For short dumps, this removes a visible share of the data (see Table 1).

Every feature is computed for a pair (p, c) from events strictly before $t_0(p)$. The strict set uses only such history and the creation time itself; the content set adds PR properties known only approximately at creation.

The strict set has fourteen features in four groups. Activity: reviews in the last seven, thirty, and ninety days (act_7d, act_30d, act_90d), reviews over the whole history (act_total), days since the last review (days_since_last), and the candidate's share of the repository's reviews in the last thirty days (act_30d_share). Workload: pull requests the candidate engaged with in the last 180 days that are still open at $t_0(p)$ (workload), the quantity w in (1). Responsiveness: the median interval between PR creation and the candidate's first reaction over the reviews submitted in the last ninety days (lat_med_h), the quantity τ in (1), and the share of those reactions within twenty-four hours (fast_24h).

Table 1. **Repositories, pull requests after warm-up and candidate pools (merged PRs; the first 90 days of each dump are warm-up; Test PRs are counted before the pool filter)**

Repository (language)	Merged PRs	After warm-up	Test PRs	Median pool	Reviewers	Top-10 share (%)	Recall any / all (%)	First review, median (h)
woocommerce (PHP)	8 850	7 767	1 554	127	176	30.9	98.8 / 96.7	18.3
react (JavaScript)	8 755	8 341	1 668	45	109	70.5	98.2 / 93.5	2.1
vscode (TypeScript)	6 342	3 057	612	82	54	52.1	99.9 / 99.0	0.7
next.js (TypeScript)	8 850	7 370	1 474	96	67	72.1	99.6 / 97.4	5.0
angular (TypeScript)	6 017	5 920	1 184	42	94	62.9	97.3 / 93.5	2.7
supabase (TypeScript)	6 002	5 149	1 030	103	104	69.9	99.0 / 96.6	3.7
strapi (TypeScript)	2 955	2 409	482	30	40	81.4	98.7 / 96.0	5.3
grafana (Go)	6 550	3 977	796	240	192	23.0	99.4 / 98.7	2.1
mattermost (Go)	5 250	4 694	939	63	75	51.2	99.9 / 98.8	2.8
node (JavaScript/C++)	1 950	1 437	288	137	98	55.8	100.0 / 94.1	3.8
discourse (Ruby)	9 550	8 347	1 670	57	60	72.9	99.7 / 99.4	1.1
django (Python)	1 500	1 331	266	57	48	84.2	99.7 / 92.1	11.6
Total	72 571	59 799	11 963					

Source: **compiled by the authors**

Familiarity and context: the author's previous pull requests the candidate reviewed (familiarity), the author's number of previous pull requests (author_prior_prs), and the median time they waited for a first review (author_med_first_review_h), and the hour and weekday of $t_0(p)$ (hour, weekday).

The content set adds four fields of the PR as stored in the dump: additions, deletions, changed_files, and is_draft. They describe the final state of the change rather than its state at creation, which makes the content, set a retrospective approximation.

The learned ranker is a gradient-boosted tree ensemble with the lambdarank objective [28] in the LightGBM implementation [29]. Each pull request is one query group; actual reviewers are labeled as one, and training stops once the validation NDCG has not improved for one hundred rounds. We train three configurations: per-repository models, a pooled model on the training splits of all repositories, and leave-one-repo-out models trained on eleven repositories and evaluated on the twelfth, with only the model parameters carried over. Feature importance is the total split gain attributed to each feature by the LightGBM implementation [29], aggregated by group.

Six rules without learning serve as baselines: random order; the most active rule, which orders the pool by the number of reviews in the last ninety days; the most familiar rule, which orders it by how many of the author's earlier pull requests the candidate reviewed and breaks ties by the same activity; the most recent rule, by the time since the last review; the workload-corrected rule, which adds the percentile ranks of activity and of low workload so that among comparably active candidates the least loaded comes first; and the fastest rule, by the median historical response time. The learned ranker with $\lambda w = \lambda t = 0$ is the relevance-only configuration, and the same ranker under (1) with weights selected on validation data is the weighted-sum variant.

The proposed rule is a constrained re-ranking. The relevance scores of a pool are min-max normalized, so that the best candidate scores one and the worst zero, and the near-top set holds the candidates whose normalized score is at least $1 - \varepsilon$. Inside the near-top set, the candidates are ordered by a secondary criterion; outside it, they keep their relevance order below the set. Two secondary criteria are studied, separately and together: the expected latency $\tau(p, c)$, and the accumulated exposure of the candidate, defined as $(r + 1) / (e + 2)$, where r is the number of earlier pull requests in the evaluation stream for which the candidate was the first choice and e is the number for which it was in the pool; the smoothing gives an unseen candidate a value of one half. The counters advance in creation order while the history stays unchanged, so the rule replays the exposure a recommender would have accumulated, not the team's reaction to it. Removing the tolerance ($\varepsilon = 1$) leaves the secondary criterion on its own; that ablation shows what the tolerance contributes.

Two further rules translate the objectives of the closest predecessors into the features of this study. They are reference points, not reimplementations. The MOC2R-inspired rule sums up the percentile ranks of the candidate's total experience, its share of past participations within twenty-four hours of pull request creation, and its recent activity. The Rebai-inspired rule sums up the ranks of experience, low workload, and prior collaboration with the author.

Within each repository, the pull requests that survive warm-up are split in creation order: 60 % for training, 20 % for validation, 20 % for testing. We choose hyperparameters, early stopping, the weights λw and λt, and the tolerance ε on the validation split. The test split is touched once. We report results per repository and as a micro-average over all test pull requests.

We measure reproduction accuracy by the mean reciprocal rank (MRR), by the hit rate at one, three, and five positions (Hit@k, the share of pull requests with at least one actual reviewer among the top k), and by the normalized discounted cumulative gain at five positions (NDCG@5) with binary gains [30]. *MRR* uses the rank of the first actual reviewer:

$$MRR = \frac{1}{|P|}\sum_{p \in P}\frac{1}{rank_p}, \qquad (2)$$

where P is the set of evaluated pull requests and $rank_p$ is the position of the first actual reviewer in the recommendation for p.

Uncertainty comes from the bootstrap. We resample the pull requests in a test split 1 000 times (500 times for the configurations in Table 4), and the 2.5th and 97.5th percentiles of each metric form a 95% confidence interval. Comparisons between methods are paired on the same pull requests, and the spread of per-repository results is reported next to the micro-average.

The trade-off analysis scores the weighted sum on a grid of weights (λw, $\lambda t \in \{0, 0.1, 0.2, 0.3, 0.5, 1\}$) and the constrained rule on a grid of tolerances ($\varepsilon \in \{0.02, 0.05, 0.1, 0.2, 0.3\}$) with each combination of the two secondary criteria, on the validation split. A configuration is admissible if its Hit@3 is at least 95 % of the relevance-only ranker's; the ranker it counts as admissible. Among admissible configurations, we select the one with the

lowest Gini of first choices, and separately the one with the lowest median expected latency of the first choice. Only the selected configurations, the untuned rules, and the ablations are read off the test split; the full test grid is a sensitivity check. Concentration is measured as the Gini coefficient of the number of first choices over every candidate that appeared in any pool of the split, zeros included, and the same coefficient is computed for actual review participation as a reference. Expected latency is the median, over test pull requests, of the first choice's lat_med_h, an estimate produced by the same rule that penalizes it. The realized latency on hits, the observed time to the first reaction of the first choice on the pull requests where it did take part, is a descriptive check that does not reuse the estimate. It is biased by selection but not circular.

RESULTS OF STUDY

Table 1 summarizes the twelve repositories after the warm-up rule. Of the 72 571 merged pull requests in the dump, 12 772 fall into the first ninety days of their repository's dump and serve only to accumulate history. The remaining 59 799 pull requests are eligible for evaluation, and 55 521 of them have at least one human reviewer. The warm-up rule removes half of the VS Code dump and two-fifths of the Grafana dump, which start in late 2025, and under 2 % of the Angular dump, which starts in 2014. The temporal split leaves 11 963 test pull requests (Table 1), of which 11 363 have at least one actual reviewer inside the candidate pool and enter the ranking evaluation (Table 2 and Table 4).

For between 97.3 % and 100 % of the evaluated pull requests, at least one actual reviewer is in the pool, and for between 92.1 % and 99.4 %, all of them are. The real process is concentrated to very different degrees: the ten most frequent reviewers account for 23.0 % of review participations in Grafana and for 84.2 % in Django.

Table 2 reports the micro-averaged ranking quality on the test split with 95 % bootstrap intervals; Table 3 breaks the main comparison down by repository. The most active rule reaches MRR 0.401 and Hit@3 0.447. Its spread across repositories is the first finding: Hit@3 runs from 0.071 in Grafana to 0.840 in Django, so the same rule describes a project with a single dominant reviewer and a project where activity alone says almost nothing. Across the twelve test periods, it names only 35 distinct people as its first choice, between one and eight per repository.

The most recent rule is weaker (Hit@3 0.377) but spreads its first choices over 683 people. The two rules that ignore relevance show that neither workload nor speed replaces it. Re-ranking the most active candidates by low workload drops Hit@3 to 0.174; recommending the candidate with the fastest expected response gives 0.064, close to the random rule (0.054). On their own the quantities that reduce concentration and expected latency poorly predict who will review.

The pooled ranker on the strict feature set lifts every metric in Table 2 well beyond the most active rule: MRR 0.569 [0.562; 0.576] against 0.401, Hit@3 0.673 [0.664; 0.682] against 0.447, and their intervals do not overlap. Per repository (Table 3), the ranker beats the most active rule on Hit@3 in 11 of twelve repositories, with a paired bootstrap

Table 2. **Ranking quality on the test split of the twelve repositories (11 363 pull requests, micro-average; 95 % bootstrap intervals over PRs)**

Method	MRR	Hit@1	Hit@3	Hit@5	NDCG@5
Random	0.077 [0.074; 0.079]	0.017 [0.014; 0.019]	0.054 [0.050; 0.058]	0.089	0.043
Most recent	0.330 [0.325; 0.337]	0.173 [0.166; 0.180]	0.377 [0.369; 0.386]	0.501	0.310
Most active (90 d)	0.401 [0.394; 0.408]	0.253 [0.245; 0.261]	0.447 [0.438; 0.457]	0.567	0.378
Most familiar	0.502 [0.494; 0.509]	0.324 [0.315; 0.333]	0.611 [0.602; 0.621]	0.734	0.504
Most active, workload re-ranked	0.172 [0.167; 0.177]	0.084 [0.078; 0.088]	0.174 [0.167; 0.181]	0.234	0.143
Fastest expected response	0.096 [0.093; 0.099]	0.026 [0.023; 0.030]	0.064 [0.060; 0.069]	0.090	0.052
Ranker, per-repository, STRICT	0.546 [0.539; 0.553]	0.376 [0.367; 0.385]	0.652 [0.643; 0.661]	0.765	0.547
Ranker, pooled, STRICT	0.569 [0.562; 0.576]	0.407 [0.397; 0.415]	0.673 [0.664; 0.682]	0.780	0.570
Ranker, pooled, STRICT + CONTENT	0.570 [0.563; 0.577]	0.408 [0.399; 0.417]	0.672 [0.663; 0.681]	0.781	0.571
Ranker, leave-one-repo-out, STRICT	0.562 [0.555; 0.569]	0.398 [0.390; 0.407]	0.664 [0.655; 0.672]	0.776	0.563

Source: **compiled by the authors**

interval that excludes zero in each of them. The exception is Django, where the most active rule already stands at 0.840: the Hit@3 difference is inside the noise (paired difference of −0.020 with the 95% interval from −0.055 to 0.012), and on MRR the most active rule keeps a small but real advantage (−0.031, from −0.058 to −0.004). All rankers in Table 3 use the STRICT features: Per-repo, Pooled, and Held-out denote the ranker trained on the repository itself, on all repositories, and on the other eleven. The last column is the paired MRR difference between the pooled ranker and the most active rule with its 95 % bootstrap interval, and PRs counts the test pull requests with an actual reviewer in the pool.

Two ablations narrow down what the ranker uses. Adding the content set, which here means the four metadata fields stored as of collection time (additions, deletions, number of changed files, draft flag) and not the title or description of the pull request, changes nothing: MRR 0.570 against 0.569. The result therefore rests on features genuinely available when the pull request was created. And training one ranker per repository is worse than training one on all of them (MRR 0.546 against 0.569), even though the per-repository models see exactly the projects they are tested on.

The split gain of the pooled ranker is distributed as follows. A single feature, the numbers of the author's previous pull requests the candidate reviewed, accounts for 63.6 % of it; the candidate's recent activity and its share of the repository's reviews add a further 23.4 %, and author history adds 11.5 %. Workload and responsiveness together contribute 1.4 %. The per-repository rankers vary more, and in some repositories activity outweighs familiarity, but in none of them does workload or responsiveness rise above 10 %. That is the second finding, and it has to be read for what it is. Split gain measures how much a feature helps reproduce the historical choice, not how much it matters for a good assignment. The most familiar rule in Table 2, which orders candidates by this feature and breaks ties by activity, reaches Hit@3 0.611 against 0.447 for the most active rule and 0.673 for the ranker. In a paired comparison, the ranker adds 0.062 [0.055; 0.069] to Hit@3 and 0.067 [0.062; 0.072] to MRR over it, and its MRR is higher in all twelve repositories. Controlling workload and expected response time is a separate goal; the trade-off analysis answers whether it costs accuracy.

Cross-project transfer. The leave-one-repo-out ranker, trained on eleven repositories and applied to the twelfth, loses little: MRR 0.562 [0.555; 0.569] against 0.569 for the pooled model, Hit@3 0.664 against 0.673. Per repository (Table 3), the Hit@3 difference between the held-out and the pooled model stays within 0.025. What transfers is the model, not the data. A project can apply a ranker trained elsewhere once it has accumulated the warm-up history, without labeled assignments of its own to train on, although the features still need its own history. The held-out model beats the most active rule on Hit@3 in 11 of twelve repositories.

Trade-off between accuracy, concentration, and expected response time. Here we ask what a project pays in accuracy for spreading first choices or preferring faster responders; Fig. 1 plots the full test grid of both rules as a sensitivity check.

Table 3. **Per-repository Hit@3 and MRR on the test split**

Repository	PRs	Most active Hit@3	Per-repo Hit@3	Pooled Hit@3	Held-out Hit@3	Most active MRR	Pooled MRR	Held-out MRR	Δ MRR pooled vs most active
woocommerce	1 464	0.255	0.605	0.640	0.654	0.253	0.525	0.536	+0.272 [+0.251; +0.291]
react	1 347	0.499	0.728	0.745	0.729	0.444	0.604	0.616	+0.159 [+0.140; +0.179]
vscode	611	0.247	0.375	0.386	0.373	0.222	0.354	0.356	+0.133 [+0.104; +0.158]
next.js	1 415	0.425	0.686	0.700	0.683	0.363	0.575	0.564	+0.212 [+0.191; +0.233]
angular	1 119	0.650	0.706	0.744	0.734	0.570	0.628	0.624	+0.058 [+0.039; +0.076]
supabase	1 014	0.416	0.626	0.622	0.619	0.369	0.530	0.525	+0.161 [+0.135; +0.187]
strapi	468	0.575	0.703	0.741	0.735	0.454	0.637	0.629	+0.183 [+0.148; +0.218]
grafana	786	0.071	0.579	0.587	0.579	0.098	0.511	0.495	+0.413 [+0.388; +0.439]
mattermost	928	0.516	0.671	0.696	0.689	0.485	0.621	0.614	+0.136 [+0.112; +0.160]
node	288	0.733	0.740	0.799	0.809	0.612	0.702	0.674	+0.090 [+0.057; +0.122]
discourse	1 667	0.543	0.663	0.677	0.652	0.491	0.569	0.536	+0.078 [+0.059; +0.097]
django	256	0.840	0.773	0.820	0.809	0.765	0.734	0.717	-0.031 [-0.058; -0.004]

Source: **compiled by the authors**

No weighted-sum configuration with a non-zero penalty passes the admissibility rule. On validation, even the mildest penalty of 0.1 lowers Hit@3 from 0.674 to 0.630 for latency and to 0.578 for workload, more than the 5% allowed, so the selection returns the relevance-only ranker for both objectives. The constrained rule with $\varepsilon = 0.1$ stays within the threshold for both secondary criteria, a tolerance of 0.2 does not, and we select $\varepsilon = 0.1$.

Table 4 reports the test split as micro-averages over 11 363 pull requests. Its latencies are medians in hours, the Gini coefficient runs over the 1 351 (repository, candidate) pairs that appeared in any pool, zeros included, and the distinct count of the reference row covers all actual participants. The exposure configuration loses 0.008 of Hit@3 and puts 479 people in first place instead of 356, and its concentration falls in every repository. The latency configuration loses 0.003 and halves the median expected latency of the first choice. Because the estimate enters both the penalty and the metric, this reduction is a property of the rule, not evidence about real waiting time. The check that avoids this circularity is realized latency on hits. On the 3 578 pull requests where the first choices of both configurations reviewed, its median is 2.04 hours for the relevance-only ranker and 1.90 hours for the latency configuration. The two name the same person on 3 422 of them; on the other 156, the latency configuration's choice answered faster in 68 % of pairs, with a median paired difference of −2.0 hours [−5.5; −0.4]. Selected on reviewed pull requests and resting on 156 pairs, this check is indicative, not confirmatory.

The ablations show where the accuracy comes from. Without the tolerance, the secondary criterion orders the whole pool and Hit@3 collapses to 0.054 for exposure and 0.064 for expected latency, because neither criterion knows who will review. The rules built from the objectives of the closest predecessors reach Hit@3 0.388 (MOC2R-inspired) and 0.224 (Rebai-inspired) and buy their balance at the price the weighted sum pays. In Fig. 1 (the dashed line in (a) marks actual review participation), the constrained rule dominates the weighted sum on the concentration axis: each weighted-sum configuration has a constrained one that is both less concentrated and more accurate. On the latency axis, the frontiers cross at about two hours; below that point a latency weight of 0.1 is more accurate.

Actual review participation over the same candidates and pull requests has a Gini of 0.847 with 679 distinct participants, so even the most balanced admissible rule is more concentrated than the real process: the first choice is one person per pull request, while participation counts every reviewer, and the rule reproduces the habits it was trained on.

Table 4. **Accuracy, concentration, and response time of the recommendation rules**

Rule	MRR	Hit@1	Hit@3 [95% CI]	Gini of top-1	Distinct top-1	Expected latency (h)	Realized latency on hits (h)	Hits (n)
Most active (90 d)	0.401	0.253	0.447 [0.438; 0.457]	0.989	35	3.03	2.93	2 872
MOC2R-inspired	0.344	0.186	0.388 [0.380; 0.398]	0.987	53	1.02	1.63	2 114
Rebai-inspired	0.233	0.100	0.224 [0.216; 0.233]	0.950	302	3.46	2.68	1 133
Most active, workload re-ranked	0.172	0.084	0.174 [0.167; 0.180]	0.969	118	3.42	3.58	949
Relevance only (ranker)	0.569	0.407	0.673 [0.664; 0.682]	0.930	356	4.07	2.67	4 620
Constrained, exposure ($\varepsilon = 0.1$)	0.544	0.364	0.665 [0.656; 0.674]	0.886	479	4.44	2.47	4 137
Constrained, latency ($\varepsilon = 0.1$)	0.558	0.387	0.670 [0.661; 0.680]	0.935	359	2.12	2.11	4 397
Exposure without tolerance	0.079	0.018	0.054 [0.050; 0.058]	0.443	1 292	9.20	4.80	200
Latency without tolerance	0.096	0.026	0.064 [0.060; 0.069]	0.983	71	0.03	0.04	301
Actual reviewers (reference)				0.847	679			

Source: **compiled by the authors**

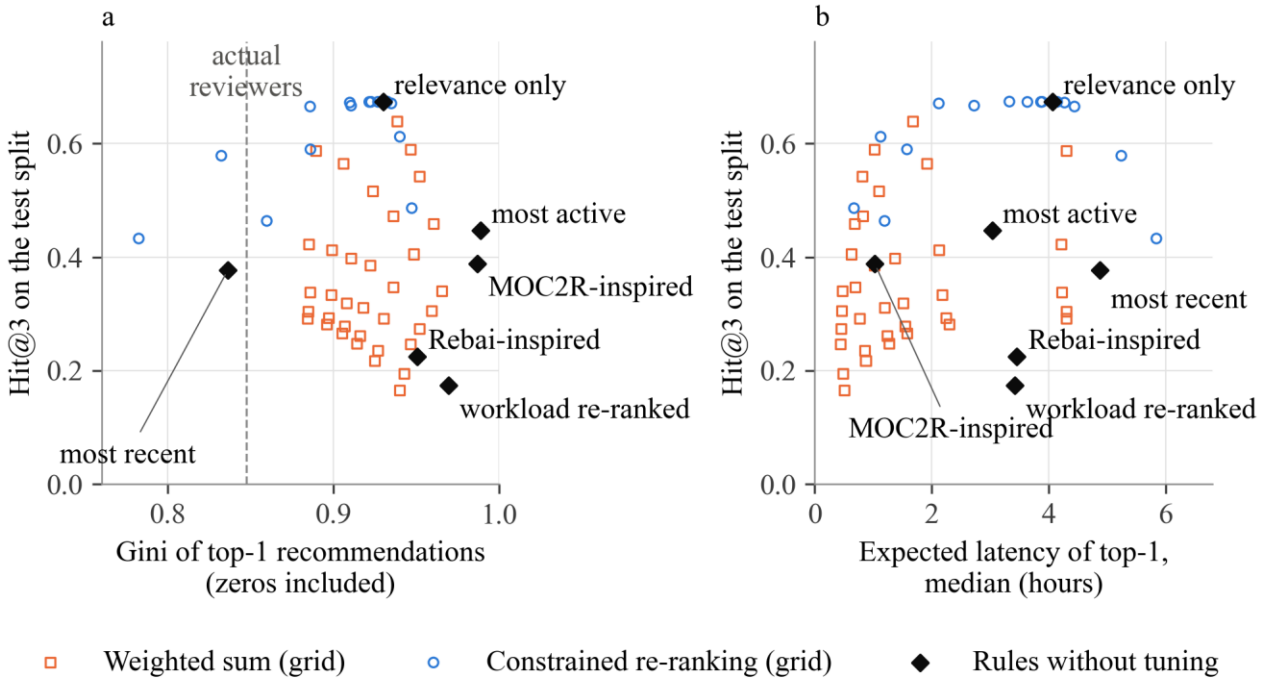


Fig. 1. **The weighted-sum and constrained re-ranking on the test split (excluding the no-tolerance ablations) and the unturned rules: a – Hit@3 against the Gini of first choices; b – Hit@3 against the median expected latency of the first choice**

Source: **compiled by the authors**

DISCUSSION

The ranker's largest gain comes from prior collaboration between reviewer and author, a pattern that also holds in unseen repositories. Workload and expected response time contribute little to predicting historical reviewers, yet they are the quantities a project may want to control. Restricting these criteria to near-top candidates preserves more historical agreement than penalizing every candidate. The constrained rule performs better across the concentration grid; on latency, the frontiers cross at about two hours.

Reproduced habit. Most of what the ranker learns is the existing assignment practice. Ranking candidates by how often they reviewed the same author already reaches Hit@3 0.611, and the ranker adds 0.062 on top. The habit it reproduces belongs to individual authors rather than to the project: the most active rule names 35 people as its first choice with a Gini of 0.989, the most familiar rule names 308 with 0.939, and the ranker 356 with 0.930. The first choice therefore stays highly concentrated, more so than actual participation (0.847). Historical labels reward this practice, so a more even load or a faster response cannot be learned from them. It has to be imposed by a separate rule, and the tolerance keeps that rule from undoing what the ranker learned.

Relation to prior work. The closest multi-objective recommenders, Rebai et al. [10], Al-Zubaidi et al. [19], WhoReview [12], and MOC2R [13], optimize expertise per pull request together with availability, the chance of participation, workload, or the probability of a fast response. This study adds to that list the median historical latency of the candidate's own reactions. Such joint objectives are hard to tune because none of the availability quantities predicts participation: every unit of weight given to them displaces the actual reviewer. The constrained rule re-orders only candidates the relevance model already treats as interchangeable; its advantage on the concentration axis is measured in Fig. 1, not assumed. Kazemi et al. [31] tune the weight of a risk objective statically or dynamically; the tolerance here plays a similar role.

Rigby et al. [14] found that a workload-balanced recommender at Meta produced mixed results, and Kovalenko et al. [15] found that recommendations rarely add information in a team that already knows whom to ask. Table 3 shows where they do: least in Django and Node, most in Grafana and WooCommerce. Gauthier et al. [32] showed that most reviewers labeled as wrong

recommendations would have been comfortable with the change, so historical participation understates suitability, which makes the near-top set plausible; whether candidates within a tolerance of 0.1 are acceptable remains a hypothesis for a pilot.

Threats to validity. The evaluation is retrospective: the label is participation, not suitability, and the concentration and latency figures describe what the rule would have recommended, not what would have happened. The expected latency is estimated from past reactions measured from pull request creation, includes the time before anyone asked, and appears in both the penalty and the metric; only realized latency on hits is observed; it is selection-biased, and the paired comparison rests on 156 pull requests. The exposure counters replay recommendations over an unchanged history and ignore the effects of adoption. The dump lacks changed-file lists, review-request timestamps, and closed-unmerged pull requests, so file-path expertise could not be tested, workload is undercounted, and the responsiveness estimate is coarse. The warm-up rule removes two-fifths to a half of the two shortest dumps; the bot filter is a login heuristic. All twelve projects are open-source GitHub repositories. Industrial teams often assign reviewers by code-owner rules or mandatory approvers, so both the accuracy and the transfer result may differ there.

Use in the review process. In the process model of [5], the recommendation enters at reviewer assignment. The relevance-only ranker is the default; the tolerance can favor broader exposure or shorter expected response time. Validation limits the Hit@3 cost to five percent; at the selected tolerance, it was about one percent. A pilot should measure actual first-reaction time and the distribution of reviewer assignments, with Hit@3 as an accuracy guardrail.

CONCLUSIONS

We rank pull request reviewers using information available when the pull request is created. Earlier multi-objective recommenders jointly search over expertise, availability, or workload [10], [12], [19]. Our method first trains a relevance model, then reorders near-top candidates by accumulated exposure or expected response time within a tolerance selected on validation data. We evaluate agreement with historical assignments, concentration of first choices, expected latency, and transfer to unseen repositories, with bootstrap intervals by repository.

The most-active rule reaches Hit@3 0.447, compared with 0.673 for the ranker; paired intervals exclude zero in eleven of twelve repositories. A rule that ranks the author's earlier reviewers already reaches 0.611, so most of this gain reproduces individual habits, and the ranker adds 0.062 on top. Leaving each repository out in turn reduces Hit@3 by less than 0.01. No nonzero weighted penalty meets the five percent validation threshold, while the selected tolerance remains within it. On the test split, it gives a third more distinct first choices or nearly halves estimated first-choice latency. It dominates the weighted sum on concentration; on latency, the frontiers cross at about two hours. A project can apply the transferred model after collecting local review history, although transfer was shown only among open-source repositories and may differ where reviewers are assigned by fixed rules.

The study stops short of the question that matters most in practice: whether the recommended assignment actually shortens the wait and evens out the load. Its labels are historical participation, and its latency is an estimate, so that question needs a live pilot in which the recommender assigns reviewers and the observed time to the first reaction is the outcome. Future work will also explore richer data: review-request timestamps would separate the time before anyone asked from the reviewer's own delay, changed-file lists would allow file-path expertise, and a larger corpus with industrial repositories would show where the transfer result holds. A further direction is to embed the ranker in the review-process model underlying this work, where reviewer assignment is one decision in the reviewer lane.

Conflicts of Interest: The author Oleksandr Aratovskyi declares no conflicts of interest regarding this study, including financial, personal, authorship, or other interests that could influence the research and the results presented in this article. Author Vira Liubchenko is a member of the Editorial Board of this journal. This role had no influence on the peer-review process or the editorial decision regarding this manuscript

Financing: Research was carried out without financial support

Data availability: The analysis is based on data collected from the public GitHub API. The source code of the data collection and modeling pipeline is published at https://github.com/lancasterSano/pr-review-research (version 1.1.0, DOI: https://doi.org/10.5281/zenodo.23534422). The derived dataset is being prepared for release under the FAIR principles (findable, accessible, interoperable, reusable) with a persistent identifier; until that release, it is available from the authors upon reasonable request.

Using artificial intelligence tools: During manuscript preparation, the artificial intelligence (AI) tool Claude was used solely for grammatical and stylistic proofreading. The authors subsequently reviewed and edited all text fragments processed by the AI tool. AI was not used to generate text, formulas, or tables, to formulate scientific statements, to obtain or interpret results, or to draw conclusions. All scientific statements, results, and conclusions are the original intellectual contributions of the authors

REFERENCES

1. Bacchelli, A. & Bird, C. “Expectations, outcomes, and challenges of modern code review”. *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. San Francisco, CA, USA. 2013. p. 712–721, https://www.scopus.com/pages/publications/84886394450. DOI: https://doi.org/10.1109/ICSE.2013.6606617.

2. Rigby, P. C., German, D. M., Cowen, L. & Storey, M.-A. “Peer review on open-source software projects: parameters, statistical models, and theory”. *ACM Transactions on Software Engineering and Methodology*. 2014; 23 (4): 1–33, https://www.scopus.com/pages/publications/84907033452. DOI: https://doi.org/10.1145/2594458.

3. Sadowski, C., Söderberg, E., Church, L., Sipko, M. & Bacchelli, A. “Modern code review: a case study at Google”. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Gothenburg, Sweden. 2018. p. 181–190, https://www.scopus.com/pages/publications/85049691666. DOI: https://doi.org/10.1145/3183519.3183525.

4. Gousios, G., Pinzger, M. & van Deursen, A. “An exploratory study of the pull-based software development model”. *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. Hyderabad, India. 2014. p. 345–355, https://www.scopus.com/pages/publications/84994130040. DOI: https://doi.org/10.1145/2568225.2568260.

5. Aratovskyi, O. S. & Liubchenko, V. V. “A process-grounded framework for submission-time prediction of code review outcomes”. *Herald of Advanced Information Technology*. 2026; 9 (4) (In press).

6. Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H. & Matsumoto, K. “Who should review my code? A file location-based code-reviewer recommendation approach for modern code review”. *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Montreal, QC, Canada. 2015. p. 141–150, https://www.scopus.com/pages/publications/84928683559.
DOI: https://doi.org/10.1109/SANER.2015.7081824.

7. Zanjani, M. B., Kagdi, H. & Bird, C. “Automatically recommending peer reviewers in modern code review”. *IEEE Transactions on Software Engineering*. 2016; 42 (6): 530–543, https://www.scopus.com/pages/publications/84976536052. DOI: https://doi.org/10.1109/TSE.2015.2500238.

8. Rahman, M. M., Roy, C. K. & Collins, J. A. “CORRECT: code reviewer recommendation in GitHub based on cross-project and technology experience”. *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE-C)*. Austin, TX, USA. 2016. p. 222–231, https://www.scopus.com/pages/publications/84989164677. DOI: https://doi.org/10.1145/2889160.2889244.

9. Ouni, A., Kula, R. G. & Inoue, K. “Search-based peer reviewers recommendation in modern code review”. *Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Raleigh, NC, USA. 2016. p. 367–377, https://www.scopus.com/pages/publications/85013130532. DOI: https://doi.org/10.1109/ICSME.2016.65.

10. Rebai, S., Amich, A., Molaei, S., Kessentini, M. & Kazman, R. “Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations”. *Automated Software Engineering*. 2020; 27 (3–4): 301–328, https://www.scopus.com/pages/publications/85090314020. DOI: https://doi.org/10.1007/s10515-020-00275-6.

11. Mirsaeedi, E. & Rigby, P. C. “Mitigating turnover with code review recommendation: balancing expertise, workload, and knowledge distribution”. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE)*. Seoul, Republic of Korea. 2020. p. 1183–1195, https://www.scopus.com/pages/publications/85094314024. DOI: https://doi.org/10.1145/3377811.3380335.

12. Chouchen, M., Ouni, A., Mkaouer, M. W., Kula, R. G. & Inoue, K. “WhoReview: a multi-objective search-based approach for code reviewers recommendation in modern code review”. *Applied Soft Computing*. 2021; 100: 106908, https://www.scopus.com/pages/publications/85097585515. DOI: https://doi.org/10.1016/j.asoc.2020.106908.

13. Hu, Y.-Z., Wang, J.-J., Li, S.-B., Hu, J. & Wang, Q. “Response time constrained code reviewer recommendation”. *Journal of Software*. 2021; 32 (11): 3372–3387, https://www.scopus.com/pages/publications/85118921592. DOI: https://doi.org/10.13328/j.cnki.jos.006079 (in Chinese).

14. Rigby, P. C., Rogers, S., Saleem, S., Suresh, P., Suskin, D., Riggs, P., Maddila, C., Nagappan, N. & Mockus, A. “Improving code reviewer recommendation: accuracy, latency, workload, and bystanders”. *ACM Transactions on Software Engineering and Methodology*. 2026; 35 (1): 1–23, https://www.scopus.com/pages/publications/105028039436. DOI: https://doi.org/10.1145/3736405.

15. Kovalenko, V., Tintarev, N., Pasynkov, E., Bird, C. & Bacchelli, A. “Does reviewer recommendation help developers?”. *IEEE Transactions on Software Engineering*. 2020; 46 (7): 710–731, https://www.scopus.com/pages/publications/85052792793. DOI: https://doi.org/10.1109/TSE.2018.2868367.

16. Yu, Y., Wang, H., Yin, G. & Wang, T. “Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment?”. *Information and Software Technology*. 2016; 74: 204–218. DOI: https://doi.org/10.1016/j.infsof.2016.01.004.

17. Hannebauer, C., Patalas, M., Stünkel, S. & Gruhn, V. “Automatically recommending code reviewers based on their expertise: an empirical comparison”. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). Singapore*. 2016. p. 99–110, https://www.scopus.com/pages/publications/84989205031. DOI: https://doi.org/10.1145/2970276.2970306.

18. Fejzer, M., Przymus, P. & Stencel, K. “Profile based recommendation of code reviewers”. *Journal of Intelligent Information Systems.* 2018; 50 (3): 597–619, https://www.scopus.com/pages/publications/85028565846. DOI: https://doi.org/10.1007/s10844-017-0484-1.

19. Al-Zubaidi, W. H. A., Thongtanunam, P., Dam, H. K., Tantithamthavorn, C. & Ghose, A. “Workload-aware reviewer recommendation using a multi-objective search-based approach”. *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE). Virtual Event, USA*. 2020. p. 21–30, https://www.scopus.com/pages/publications/85097300890. DOI: https://doi.org/10.1145/3416508.3417115.

20. Sefidi Esfahani, M. & Rigby, P. C. “Mitigating the risk of defects and improving knowledge distribution with code reviewer recommenders”. *Proceedings of the ACM on Software Engineering*. 2026; 3 (FSE): 160–181. DOI: https://doi.org/10.1145/3797070.

21. Strand, A., Gunnarson, M., Britto, R. & Usman, M. “Using a context-aware approach to recommend code reviewers: findings from an industrial case study”. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Seoul, Republic of Korea. 2020. p. 1–10, https://www.scopus.com/pages/publications/85092597198. DOI: https://doi.org/10.1145/3377813.3381365.

22. Thongtanunam, P., McIntosh, S., Hassan, A. E. & Iida, H. “Review participation in modern code review: an empirical study of the Android, Qt, and OpenStack projects”. *Empirical Software Engineering*. 2017; 22 (2): 768–817, https://www.scopus.com/pages/publications/84991786769. DOI: https://doi.org/10.1007/s10664-016-9452-6.

23. Ruangwan, S., Thongtanunam, P., Ihara, A. & Matsumoto, K. “The impact of human factors on the participation decision of reviewers in modern code review”. *Empirical Software Engineering*. 2019; 24 (2): 973–1016, https://www.scopus.com/pages/publications/85053471841. DOI: https://doi.org/10.1007/s10664-018-9646-1.

24. Chouchen, M., Ouni, A., Olongo, J. & Mkaouer, M. W. “Learning to predict code review completion time in modern code review”. *Empirical Software Engineering*. 2023; 28 (4): 82, https://www.scopus.com/pages/publications/85160020919. DOI: https://doi.org/10.1007/s10664-023-10300-3.

25. Aratovskyi, O. S. & Liubchenko, V. V. “Constructing a GitHub-based dataset for machine learning driven software effort estimation”. *Proceedings of the 2025 IEEE 13th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Gliwice, Poland. 2025. p. 1–7, https://www.scopus.com/pages/publications/105032752133. DOI: https://doi.org/10.1109/IDAACS68557.2025.11322078.

26. Badampudi, D., Unterkalmsteiner, M. & Britto, R. “Modern code reviews: survey of literature and practice”. *ACM Transactions on Software Engineering and Methodology*. 2023; 32 (4): 1–61, https://www.scopus.com/pages/publications/85163852501. DOI: https://doi.org/10.1145/3585004.

27. Yang, Z., Gao, C., Guo, Z., Li, Z., Liu, K., Xia, X. & Zhou, Y. “A roadmap for modern code review: challenges and opportunities”. *ACM Transactions on Software Engineering and Methodology*. 2026; 35 (9): 1–42. DOI: https://doi.org/10.1145/3800963.

28. Burges, C. J. C. “From RankNet to LambdaRank to LambdaMART: an overview”. Technical Report MSR-TR-2010-82. *Microsoft Research*. 2010. – Available from: https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview.

29. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. & Liu, T.-Y. “LightGBM: a highly efficient gradient boosting decision tree”. *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Long Beach, CA, USA. 2017. – Available from: https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

30. Järvelin, K. & Kekäläinen, J. “Cumulated gain-based evaluation of IR techniques”. *ACM Transactions on Information Systems*. 2002; 20 (4): 422–446, https://www.scopus.com/pages/publications/1842637192. DOI: https://doi.org/10.1145/582415.582418.

31. Kazemi, F., Lamothe, M. & McIntosh, S. “Exploring the notion of risk in code reviewer recommendation”. Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). Limassol, Cyprus. 2022. p. 139–150, https://www.scopus.com/pages/publications/85146239340.
DOI: https://doi.org/10.1109/ICSME55016.2022.00021.

32. Gauthier, I. X., Lamothe, M., Mussbacher, G. & McIntosh, S. “Is historical data an appropriate benchmark for reviewer recommendation systems? A case study of the Gerrit community”. *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Melbourne, Australia. 2021. p. 30–41, https://www.scopus.com/pages/publications/85125482462. DOI: https://doi.org/10.1109/ASE51524.2021.9678640.

Received 03.09.2026
Received after revision 26.10.2026
Accepted 30.10.2026

DOI: https://doi.org/10.15276/aait.09.2026.39
УДК 004.41:004.85

Ранжування рецензентів запитів на злиття за експертизою, навантаженням та очікуваним часом відгуку

Аратовський Олександр Сергійович[1)]
ORCID: https://orcid.org/0009-0006-0854-8582; alex.aratovskiy@stud.op.edu.ua. Scopus Author ID: 60503533100
Любченко Віра Вікторівна[1), 2)]
ORCID: https://orcid.org/0000-0002-4611-7832; lvv@op.edu.ua. Scopus Author ID: 56667638800
[1)] Національний університет “Одеська політехніка”, просп. Шевченка, 1. Одеса, 65044, Україна
[2)] Університет прикладних наук, Ульменліт 20. Гамбург, 21033, Німеччина

АНОТАЦІЯ

Актуальність. Якість ранжування рецензентів зазвичай оцінюють за тим, чи запропонована людина згодом рецензувала запит на злиття, але це мало каже про ширину розподілу рекомендацій і швидкість відгуку цієї людини. **Мета статті.** Ми з'ясовуємо, наскільки точно можна ранжувати рецензентів у момент створення запиту на злиття і чого коштує в точності врахування концентрації рекомендацій та очікуваного часу відгуку. **Завдання.** Ми формуємо списки кандидатів з попередньої історії рецензування, порівнюємо навчену модель із простими правилами, перевіряємо її на незнайомих репозиторіях і зіставляємо два способи врахування додаткових критеріїв. **Методи.** Ми використовуємо злиті запити дванадцяти репозиторіїв з відкритим кодом. Хронологічне розбиття обмежує кожну рекомендацію попередньою інформацією. Ми навчаємо модель ранжування на основі градієнтного бустингу, вимірюємо, чи потрапляє фактичний рецензент до першої трійки кандидатів, та обчислюємо довірчі інтервали методом бутстрепу. **Наукова новизна.** Ми порівнюємо зважений штраф із правилом, яке змінює порядок лише кандидатів, близьких до найвищої оцінки релевантності, і разом оцінюємо точність, розподіл перших рекомендацій та оцінений час відгуку. **Практична значущість.**

Модель, навчену деінде, придатна для нового проєкту, щойно в ньому накопичиться власна історія рецензувань, а параметр допуску обмежує відхилення від ранжування лише за релевантністю. **Результати.** На тестовій вибірці перша трійка кандидатів моделі містила фактичного рецензента приблизно для двох третин запитів, а в правила найактивнішого рецензента менше ніж для половини; більшу частину цієї переваги вже дає ранжування попередніх рецензентів автора. Під час перевірки з вилученням одного репозиторію точність знизилася менше ніж на один відсотковий пункт. Правило з допуском поставило на перше місце на третину більше різних людей або майже вдвічі зменшило медіанний оцінений час відгуку першого кандидата, щоразу ціною менше ніж одного відсоткового пункту точності. Жоден ненульовий зважений штраф не задовольнив поріг точності на валідаційних даних. **Висновки.** Урахування навантаження або очікуваного часу відгуку мало коштує в межах допуску від найкращої оцінки релевантності і дорого, якщо застосовувати його до кожного кандидата. Чи скоротить це фактичний час очікування, слід перевірити під час пілотного застосування.

Ключові слова: перевірка коду; рекомендація рецензента; запит на злиття; модель ранжування; навантаження рецензента; крос-проєктне перенесення

ABOUT THE AUTHORS



Oleksandr S. Aratovskyi - PhD student, Department of Software Engineering, Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine.
ORCID: https://orcid.org/0009-0006-0854-8582; alex.aratovskiy@stud.op.edu.ua. Scopus Author ID: 60503533100
Research field: Machine learning; code review analytics; mining software repositories

Аратовський Олександр Сергійович - аспірант кафедри Інженерії програмного забезпечення. Національний університет "Одеська політехніка", просп. Шевченка, 1. Одеса, 65044, Україна
***Галузь досліджень*:** машинне навчання; аналітика перевірки коду; аналіз репозиторіїв програмного забезпечення



Vira V. Liubchenko - Doctor of Engineering Sciences, Professor, Department of Software Engineering, Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine;
Lecturer, Fakultät Life Sciences, Hochschule für Angewandte Wissenschaften Hamburg, Ulmenliet 20, Hamburg, 21033, Germany
ORCID: https://orcid.org/0000-0002-4611-7832; lvv@op.edu.ua. Scopus Author ID: 56667638800
Scientific field: Data science, software engineering, project management

Любченко Віра Вікторівна - доктор технічних наук, професор кафедри Інженерії програмного забезпечення Національного університету "Одеська політехніка", пр. Шевченка, 1. Одеса, 65044, Україна;
факультет Наук про життя. Гамбурзький університет прикладних наук, Ульменліт 20. Гамбург, 21033, Німеччина
***Галузь досліджень*:** наука про дані, інженерія програмного забезпечення, управління проєктами