# THE HIDDEN ROLE OF CANDIDATE-POOL PROVENANCE IN SELECTOR EVALUATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Localise-then-repair systems first retrieve a small set of candidate code locations and then select the most relevant one. Existing evaluations often interpret the resulting localisation accuracy as a property of the selector, implicitly assuming that this accuracy is stable across candidate generators. We show that this assumption does not generally hold. We introduce a controlled evaluation protocol that fixes the selector, candidate-set size, gold element, and evaluation instances, while varying only the generator that supplies the distractors. Across multiple candidate generators and 7 open-weight selectors on 269 SWE-bench Verified issues, changing candidate provenance substantially shifts conditional selection accuracy, with an average gap of 8.2 points between real localisers. Despite these shifts in absolute performance, selector rankings remain largely stable. We further find that generator coverage is strongly associated with selector difficulty, enabling a simple coverage-based adjustment that improves transfer of absolute performance estimates to unseen generators. These results show that selector accuracy should be reported together with the candidate generator on which it is measured, and motivate separating retrieval coverage from conditional selection performance in localisation evaluation.

## 1 INTRODUCTION

Large language models are increasingly used to resolve repository-scale software issues, where the model must identify a small number of relevant code locations before generating a repair. In practice, this process is rarely a single prediction problem. Modern systems typically first narrow a repository containing thousands of functions or methods to a small candidate set, and then ask a language model to reason over those candidates and identify where the issue should be fixed. This decomposition appears explicitly in systems such as Agentless (Xia et al., 2024), and variants of the same pattern are present in AutoCodeRover (Zhang et al., 2024), SWE-agent (Yang et al., 2024), and more recent graph-based localisation methods (Chen et al., 2025; Jiang et al., 2025; Ouyang et al., 2024). As repository-scale software engineering benchmarks become increasingly competitive, localisation accuracy is therefore used not only as an intermediate diagnostic, but also as evidence about how effectively a model can distinguish relevant code from irrelevant repository context.

There is, however, a hidden dependency in this measurement. A localisation selector is never evaluated in isolation: it is evaluated on a candidate pool produced by some generator. Changing that generator changes two things simultaneously. First, it changes *coverage*: whether the correct code location appears in the candidate pool at all. Second, even conditional on the correct location being present, it changes the alternatives against which that location must be distinguished. A weak retriever may surround the answer with obviously unrelated functions, whereas a strong retriever may place it next to highly similar code from the same file, module, or semantic neighbourhood. The latter generator can therefore improve coverage while making the downstream selection problem harder. Consequently, the accuracy measured for a selector may depend substantially on where its distractors came from.

This distinction is obscured by the usual end-to-end evaluation. If a system achieves accuracy $a$ using generator $G$, the reported number reflects both what $G$ retrieved and how well the downstream selector handled the resulting pool. From that number alone, one cannot determine whether an error arose because the answer was never retrieved or because the selector failed to identify it among



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

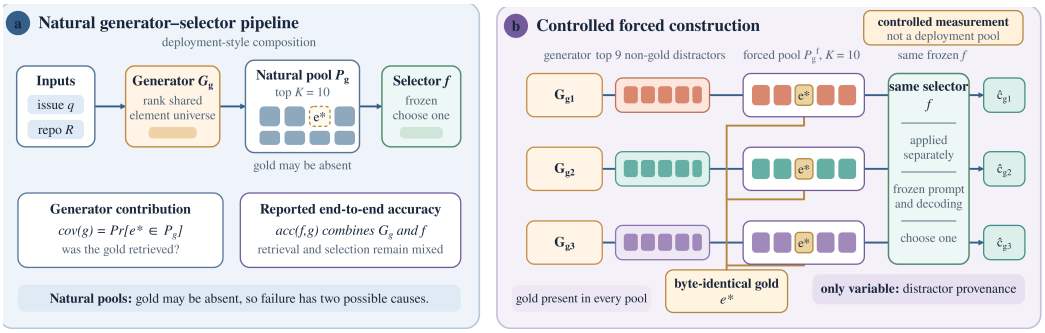



Figure 1: The generator–selector pipeline and the controlled intervention. Coverage is set by $G$ alone; conditional recall is measured with the gold element present by construction, so the two stages become separately attributable.

the retrieved alternatives. More importantly, the selector-side component need not remain constant when $G$ is replaced. This raises a basic measurement question: *to what extent does a selector accuracy measured under one candidate generator remain valid under another?* Existing localisation benchmarks provide little evidence for this assumption because the generator is normally fixed together with the evaluation harness.

We study this question through a controlled intervention on candidate-pool provenance. We hold the downstream decision problem fixed and vary only the source of its distractors. For each issue, we construct candidate pools in which the gold code element is present by construction and is byte-identical across generators; the number of candidates, prompt, label space, decoding procedure, and selector checkpoint are also held fixed. Only the generator that supplies the non-gold candidates is changed. This construction separates the generator's ability to *retrieve* the answer from the selector's ability to *recognise* it once retrieved, yielding a conditional recall that can be compared across generators. Figure 1 contrasts this controlled construction with the natural generator–selector pipeline.

We apply this protocol to repository-scale localisation on SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024). Our main panel contains generators spanning lexical, dense, structural, language-model-based, history-based, and mixed retrieval strategies, together with seven open-weight selectors from multiple model families. Across 269 issues, the resulting experiment contains more than one hundred thousand scored selector decisions. The design deliberately treats the generators as experimental instruments rather than competing localisation methods: they share the same element universe and output interface, allowing candidate provenance to be changed without simultaneously changing the rest of the evaluation.

The effect is substantial. Among the five generators that represent deployable, issue-aware localisers, transferring a selector measurement from one generator to another produces an average absolute gap of 8.2 percentage points in conditional recall. An exact permutation test rejects the hypothesis that generator provenance is irrelevant ($p < 0.001$), and the conclusion survives removing both the most distinctive generator and the repository contributing the largest number of evaluation units. Across the broader generator panel, the effect is even more pronounced: generators that retrieve the gold location more frequently systematically produce candidate pools on which the same selectors perform worse. Thus, stronger retrieval and easier selection are not aligned objectives.

Importantly, this dependence changes the *level* of selector performance much more than its *ordering*. Although absolute conditional recall shifts across generators, selector rankings remain highly stable, with the weakest pairwise Kendall correlation still at $\tau = 0.81$. This distinction matters for how localisation results should be interpreted. Comparative statements such as "selector $A$ outperforms selector $B$" are relatively robust to candidate provenance in our experiments, whereas absolute statements such as "selector $A$ achieves $x\%$ localisation accuracy" are generator-dependent. Candidate provenance is therefore not merely an implementation detail of the benchmark; it determines the scale on which selector accuracy is measured.

The dependence is also surprisingly regular. Generator coverage and conditional recall move in opposite directions, and across our generator panel their relationship is close to linear. This regularity turns the measurement problem into a practical opportunity: coverage, which can be computed without any additional selector calls, predicts the difficulty of an unseen candidate generator substantially better than transferring a selector score unchanged from a single benchmark generator. In leave-one-generator-out evaluation, the coverage-based correction reduces prediction error from 10.2 to 2.2 points over the full generator range. On a held-out retrieval artifact released with SWE-bench and excluded from model fitting, the predicted conditional recall is within 1.5 points of the measured value. We also identify the limit of this result: among the narrower set of realistic localisers with similar coverage, the advantage over direct score transfer is smaller and its uncertainty includes zero.

Taken together, these results suggest that localisation evaluation should distinguish *getting the answer into the pool* from *selecting the answer once it is there*. A selector does not have a single generator-independent localisation accuracy; it has a performance level induced jointly by the selector and the candidate distribution on which it is measured. The observation leaves existing localisation systems and their generators in place and changes what should be reported when evaluating them.

Our main contributions are:

- We formulate candidate-pool provenance as a controlled factor in repository-scale localisation and introduce an evaluation protocol that separates generator coverage from selector conditional recall while holding the gold element and downstream decision problem fixed. Section 3 defines the protocol and Appendix F validates it.
- We provide a large-scale controlled measurement across 269 SWE-bench Verified issues, multiple candidate-generator families, and seven open-weight selectors, showing that changing only candidate provenance induces an average 8.2-point conditional-recall gap between real localisers, despite leaving selector rankings largely stable.
- We characterize the structure of this shift and show that generator coverage provides a simple, zero-selector-call predictor of conditional recall on unseen generators, while explicitly identifying the regime in which the correction's advantage is not statistically resolved. Section 6 tests it (Fig. 5 in Appendix J).

## 2 Related Work

### 2.1 Software Issue Localisation and Repair

Issue resolution on SWE-bench (Jimenez et al., 2024; OpenAI, 2024) is commonly structured around a localisation stage followed by repair. Agentless (Xia et al., 2024) makes this separation explicit, while AutoCodeRover (Zhang et al., 2024) and SWE-agent (Yang et al., 2024) interleave localisation and repair but still rely on a candidate-narrowing process; LocAgent (Chen et al., 2025), CoSIL (Jiang et al., 2025), and RepoGraph (Ouyang et al., 2024) further replace or augment lexical retrieval with repository structure and graph-based signals. This design inherits a much older line of work on retrieval-based bug localisation (Zhou et al., 2012; Wong et al., 2016), where ranking quality was already understood to depend on the interaction between the retrieval model and the corpus it searches. The LLM setting introduces an additional complication: the downstream selector is often a general-purpose model that is not trained for the output distribution of a particular localisation method. Consequently, changing the candidate generator may alter not only whether the correct location is retrieved, but also how difficult it is for the same selector to identify that location once it is present. Existing software-localisation systems typically report selector or localisation accuracy under their own candidate generator, whereas we explicitly measure how a fixed selector changes when the generator underneath it is replaced.

### 2.2 Candidate-Pool Dependence and Evaluation Validity

This generator dependence connects software localisation to a broader class of two-stage retrieval systems. In text retrieval, reranker performance depends on the first-stage retriever and the hardness of the negatives it supplies (Nogueira et al., 2020; Xiong et al., 2021), while in retrieval-augmented generation the downstream reader's accuracy likewise depends on the retrieved context and can vary

non-monotonically with retrieval quality (Cuconasu et al., 2024). Our setting isolates this effect more directly: the selector is frozen and untrained on any specific generator, ruling out ordinary train–test mismatch, and the gold element is byte-identical across generators, allowing coverage and conditional discriminability to be separated. We additionally control for known presentation effects, including option-label preferences (Zheng et al., 2024) and sensitivity to answer position (Liu et al., 2024), by holding presentation fixed and marginalising over candidate orderings. More broadly, this study follows work on evaluation validity and behavioural testing (Ribeiro et al., 2020), which shows that aggregate metrics can obscure the contribution of individual system components. Here, the candidate generator, usually treated as fixed evaluation infrastructure, is itself the controlled factor, while BM25 (Robertson & Zaragoza, 2009), dense retrieval (Karpukhin et al., 2020; Reimers & Gurevych, 2019), and the broader retrieval-augmented generation framework (Lewis et al., 2020) serve as experimental instruments rather than methodological contributions.

# 3 Generator-Shift Protocol

## 3.1 The two-stage pipeline

Repository-scale fault localisation is a two-stage pipeline. A *generator* $G$ reads an issue report $q$ and a repository snapshot $\mathcal{R}$ and returns a pool of candidate code locations $P = G(q, \mathcal{R})$ with $|P| = K$. A *selector* $f$ reads $q$ and a serialisation of $P$ and returns one candidate. The pipeline is correct on an instance when $f(q, \mathrm{render}(P))$ is a location the gold patch edits.

Published evaluations report the composition $f \circ G$ but attribute the number to $f$. The generator is treated as a fixed part of the harness, and is almost always lexical retrieval over the issue text. Our question is what happens to the reported number when that part of the harness is replaced.

## 3.2 Estimands

Fix a unit distribution over issue–repository pairs. Coverage is how often the generator's pool contains a gold element, a property of the generator alone: an oracle selector achieves exactly $\mathrm{cov}(g)$. Accuracy is how often the selector returns gold from that pool, and oracle regret $\mathrm{reg}(f, g)$ is the part of the loss the selector is responsible for. For a selector $f$ and generator $g$:

$$\mathrm{cov}(g) = \Pr[\,\text{gold} \in G_g(q, \mathcal{R})\,] \tag{1}$$

$$\mathrm{acc}(f, g) = \Pr[\,f \text{ returns gold from } G_g\,] \tag{2}$$

$$\mathrm{reg}(f, g) = \mathrm{cov}(g) - \mathrm{acc}(f, g) \tag{3}$$

These three quantities all confound generator quality with selector quality, because a generator that retrieves better also changes what the selector has to discriminate between. We therefore define a fourth quantity on a pool construction that removes the confound.

**Conditional recall.** Let $G_g^{\mathrm{f}}(q, \mathcal{R})$ be the pool containing the gold element(s) plus the top-ranked *non-gold* elements of $G_g$, padded to the same size $K$. Then

$$R(f, g) = \Pr[\,f \text{ returns gold from } G_g^{\mathrm{f}}\,]. \tag{4}$$

Across generators, $G^{\mathrm{f}}$ holds the unit set, the pool size, the gold element and its serialisation byte-identical. The only thing that varies is which distractors accompany the gold element. $R(f, g)$ is therefore a measurement of the selector under a controlled change of candidate provenance, and $\mathrm{acc}(f, g) \le \mathrm{cov}(g)$ decomposes as coverage times a quantity that behaves like $R$.

**Forced pools.** $G^{\mathrm{f}}$ is the only construction under which a cross-generator selector comparison is identified: without it, an accuracy difference mixes two causes that cannot be separated after the fact. Either the answer was absent more often, or it was harder to see. $\mathrm{cov}$ and $\mathrm{acc}$ are reported on natural pools throughout, and Section 6 shows the two arms compose.

**Transport gap.** For a source generator $s$ and a target generator $t$,

$$\Delta(f; s \to t) = R(f, s) - R(f, t) \tag{5}$$

is the error a practitioner makes by reading a number measured on $s$ as a prediction of behaviour on $t$. Generator-transportability is the hypothesis $\Delta \approx 0$ for all $(s, t)$; a material gap falsifies it.

### 3.3 GENERATORS

Every generator is a pair of ranking functions with an identical signature: rank the repository's non-test Python files, keep the top $F$, extract the element universe from those files, rank the elements, keep the top $K$. Seven primary generators span five families.

- **BM25**: Okapi BM25 over the issue text at both levels, the retrieval stage used by Agentless-style pipelines and by essentially every published selector benchmark.
- **Dense**: cosine similarity of mean-pooled sentence embeddings.
- **Struct**: paths, dotted module names and identifiers the issue names inside code spans, matched against file paths and AST definitions and expanded one import-graph hop. It is a deliberately simple structural instrument.
- **LLM-A**, **LLM-B**: a language model names suspicious files from a compressed repository tree, then elements from those files. Two checkpoints instantiate the same family, which separates within-family from between-family shift.
- **History**: commit count and recency over the commits preceding the base commit; it never reads the issue and is a deployment prior, not a competitive localiser.
- **Random**: uniform over the repository's elements, seeded by instance. The signal-free anchor of the generator-strength axis.

Three further generators are derived from those rankings at no extra retrieval cost. **Mix-7** and **Mix-4** interleave their members round-robin, so a pool carries one distractor per member family; Mix-4 excludes all three neural generators. **BM25$_{\text{lm}}$** keeps BM25's ranking but re-picks candidates to carry Dense's candidate-length profile, so a gap that survives it is not a length effect.

### 3.4 WHAT IS HELD FIXED

The intervention is the candidate-generation distribution and nothing else. The unit set, the element universe, the gold definition, the pool size, the gold element's bytes, the prompt template, the label alphabet, the token cap, the decoding rule and the selector checkpoints are all identical across generators. Selection reads the next-token distribution once, restricts it to the label tokens of that rendering, renormalises, and takes the argmax, so there is no sampling, no parse failure, and no temperature confound between generators.

## 4 EXPERIMENTAL SETUP

**Units.** We use SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024), the human-validated 500-instance subset of SWE-bench, and read each repository at the instance's base commit from a bare clone, so no repository state is ever written. An instance is kept when its gold patch touches only Python files outside test paths, when every patch hunk falls inside an extractable function or method of at most 60 lines, and when the resulting gold set has at most four elements. 269 instances over 11 repositories survive; attrition by reason is in Appendix A. The unit set and the gold element set are fixed once, before any generator runs, so every generator is scored on exactly the same issues against exactly the same targets.

**Pools.** Every pool holds $K = 10$ candidates. In the *forced* construction the gold element or elements are present by construction and the remaining slots are the generator's top-ranked non-gold elements; in the *natural* construction the pool is simply the generator's top ten. Candidate code is never truncated. Slot order is a fixed pseudorandom permutation of the pool, so the gold position is not privileged, and each pool is scored again under two further permutations.

**Generators.** The 10 generators of Section 3.3 are configured as follows; Appendix B gives each generator's full settings. BM25 uses $k_1 = 1.2$, $b = 0.75$ (Robertson & Zaragoza, 2009). Dense uses mean-pooled `all-MiniLM-L6-v2` embeddings truncated at 256 tokens (Reimers & Gurevych, 2019). Struct mines paths, dotted module names and identifiers from the issue's fenced and inline code spans only, matches them against file paths and AST definitions, and expands one import-graph hop. LLM-A is Qwen2.5-7B-Instruct (Qwen Team, 2024) and LLM-B is Qwen3-4B-Instruct-2507 (Yang et al., 2025), both decoded greedily at temperature 0 with a fixed seed, each making two calls per instance: files from a compressed repository tree, then elements from those files. History counts commits over the 4000 commits preceding the base commit. Every generator keeps 25 files before ranking elements.

**The held-out generator.** One further generator is external: SWE-bench releases a BM25 retrieval setting alongside the benchmark (`SWE-bench_bm25_13K`); we take the ordered file list that artifact retrieves for each of our units, apply the same element stage and the same pool construction, and hold the result out of the panel, out of every fit, and out of every number except the out-of-sample test in Section 6.

**Selectors.** 7 open-weight checkpoints from 1.7B to 30.5B parameters across 5 release families (Olmo-3, Phi-4, Qwen2.5, Qwen3, Qwen3-VL) (Yang et al., 2025; Qwen Team, 2024; Allen Institute for AI, 2025; Microsoft, 2025; Qwen Team, 2025). 3 share the Qwen3 lineage, which is a real limit on how far the selector-side conclusions generalise. All run through one code path: BF16, `transformers` with SDPA attention, one GPU at a time, locally with `TRANSFORMERS_OFFLINE=1`; every model in this study, as a selector or as a generator, is an open-weight checkpoint run locally. The panel is split across two hosts by which host holds each checkpoint's weights and no checkpoint spans hosts, so every within-checkpoint comparison is made inside one environment (Appendix C). The prompt is wrapped in each model's chat template and terminated with `Answer: [`; we read the next-token distribution once, restrict it to that rendering's label tokens, renormalise and take the argmax. Selection is deterministic and parse-failure-free, and the restriction is not distorting: the label tokens carry 99.5% of the next-token mass on average.

**Scale of the measurement.** The measurement comprises 104,372 scored decisions over 269 units across forced, natural, control, and held-out arms, at 11.1 GPU-hours of measured occupancy. Appendix D gives the complete factorisation, library versions, and per-checkpoint runtime record.

**Statistics.** Every mean and every source-to-target difference carries a repository-clustered bootstrap interval over 5000 resamples, drawn from one shared set of resample indices so differences are paired; headline contrasts add an exact McNemar test, and the omnibus question is answered by an exact permutation test over generator identity. Seeds are repeated measurements of the same unit, not independent units. The full generator $\times$ generator matrix is reported; no pair was selected after inspection.

**Controls.** Three controls bound the measurement, with pre-declared kill criteria checked before and after the runs (Appendix E). *Repeat* re-scores an identical rendering in a different batch, fixing the numerical noise floor. *Issue removed* replaces the issue with a placeholder, bounding the accuracy available without reading it. *BM25$_{lm}$* holds BM25's ranking but imposes Dense's candidate-length profile, separating a content effect from a length effect; Appendix H reports it.

# 5 Results

## 5.1 What the generators produce

Table 1 in Appendix J describes the ten generators before any selector is called: they differ on coverage, on where the first gold element sits in the full ranking, on how many distractors come from the gold file, and on distractor length. Pairwise overlap between the distractor sets of two different families is at most 0.29 by Jaccard index (mean 0.06; 0.22 between the two agentic checkpoints, which share a family), so the pools are different objects rather than reorderings of one another. The gold element is identical in every row by construction.

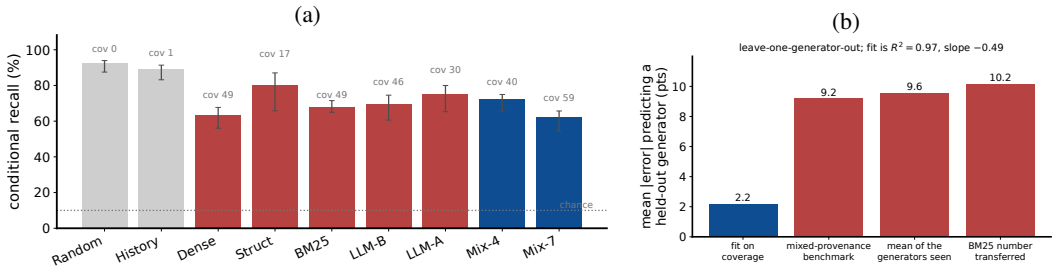



Figure 2: **(a)** Conditional recall by generator, pooled over the selector panel, with repository-clustered bootstrap intervals. The gold element is byte-identical across every bar; only the nine distractors change. Grey annotations give each generator's coverage. History and Random never read the issue and anchor the generator-strength axis. **(b)** Mean absolute error of predicting a held-out generator's conditional recall, over all nine generators. The restricted comparison over the five real localisers is 2.5 against 6.5 points.

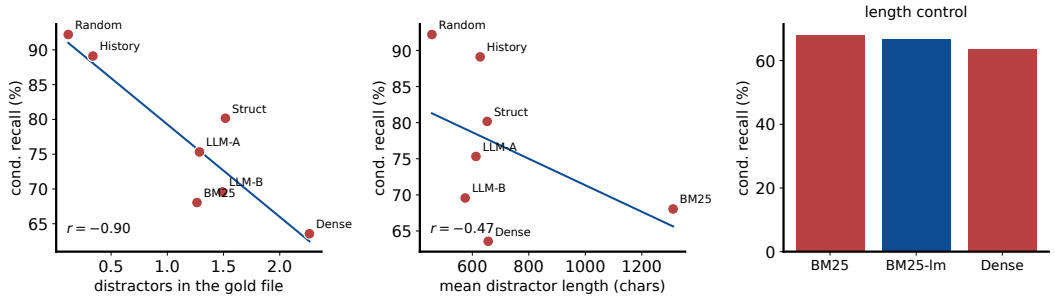



Figure 3: Left and centre: pool-side difficulty against conditional recall, one point per generator. Right: the length control. $\text{BM25}_{\text{lm}}$ keeps BM25's ranking but carries Dense's candidate-length profile, and stays with BM25 rather than moving to Dense.

### 5.2 Conditional recall depends on the generator

Figure 2(a) gives the primary result, and Table 3 in Appendix J lists every cell with its interval. Every cell is the same selector answering the same question about the same issue with the same gold element in the pool; only the nine distractors change. Pooled over the selector panel, conditional recall runs from 62.1% on Mix-7 pools to 92.2% on Random pools, a span of 30.1 points. Five of the ten are not localisers anyone ships: History and Random are issue-agnostic anchors on the generator-strength axis, the two mixtures are our own constructions, and $\text{BM25}_{\text{lm}}$ is the length control. Restricting to the five that are real localisers still leaves a span of 16.6 points; the next subsection asks how much of that span the data actually resolve.

The ordering is not arbitrary. Generators that read the issue and retrieve well place harder code next to the gold element and conditional recall falls accordingly; generators that ignore the issue fill the pool with unrelated code that any selector can eliminate. Figure 3 plots pool-side difficulty against conditional recall. Figure 6 in Appendix J shows the pattern holds for every checkpoint in the panel rather than being carried by one model.

### 5.3 The transport gap

Table 2 and Fig. 5(b), both in Appendix J, read the same data the way a practitioner would: a number measured on the row generator, used as a prediction of behaviour on the column generator. Both report realised minus measured, so a negative cell is a selector that looks better on the benchmark than it turns out to be.

**Omnibus test.** An exact test answers whether generator identity matters among real localisers at all, before any pairwise comparison and independently of pairwise thresholds. Under the null hypothesis that a selector's decision on a unit is unaffected by which generator built the pool, the 5 deployable outcomes within a (selector, unit) cell are exchangeable. Permuting them there and recomputing

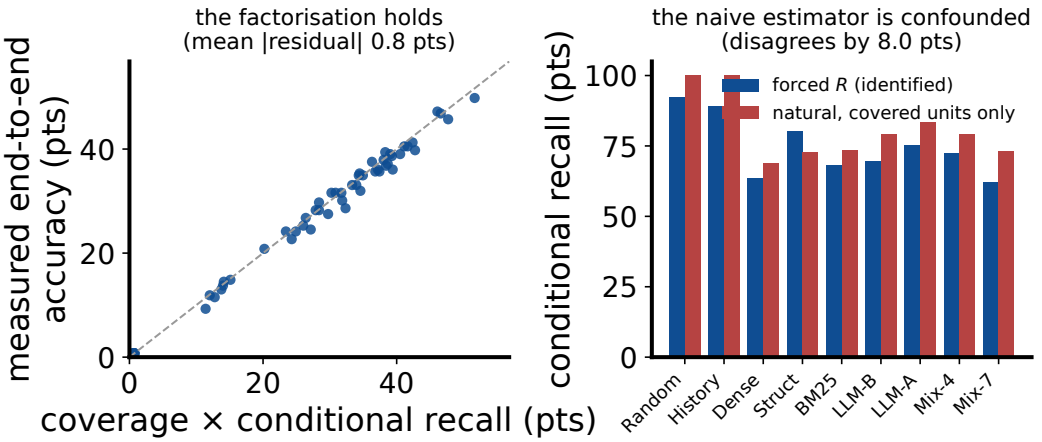



Figure 4: Left: measured end-to-end accuracy against the product of coverage and forced conditional recall, one point per (selector, generator) cell; the dashed line is $y = x$. Right: the forced estimand against the naive alternative of measuring conditional recall on natural pools restricted to covered units, which is confounded by which units each generator covers.

the mean absolute gap over ordered pairs gives an exact reference distribution. The observed value, 8.2 points, is not attained in 5,000 permutations ($p < 0.001$; Figure 9 in Appendix J). The effect survives removing the most distinctive generator, the dominant repository, or both; Appendix K reports all three exact tests. The same permutation construction resolves 9 of 10 deployable pairs under family-wise control. Appendix K gives the pre-registered bootstrap calibration, pairwise thresholds, and repository-sensitivity analysis in full.

### 5.4 Coverage and conditional recall move in opposite directions

Figure 5 in Appendix J plots each generator by its coverage and its conditional recall against curves of constant end-to-end accuracy. The two are negatively related across the 9 generators, and tightly so: Pearson $r = -0.987$, Spearman $\rho = -0.996$. The generators that most often put the answer in the pool are exactly the ones whose pools the selector handles worst. Because end-to-end accuracy is close to their product, the two effects partly cancel, and Table 4 with Figure 7, both in Appendix J, show the consequence: a wide spread in coverage and a wide spread in conditional recall produce a much narrower spread in what the pipeline achieves, with oracle regret absorbing the difference.

This is why a single reported accuracy is uninformative about either stage. Two pipelines with the same end-to-end number can differ by tens of points in coverage, and the stage that accounts for the difference is not recoverable from the number alone. Section 6 turns the same relation into a correction.

### 5.5 The level moves; the ordering does not

A gap this large could break a benchmark in two different ways: by misstating how good a selector is, or by misstating which selector is better. Only the first happens. Figure 8 in Appendix J reports Kendall's $\tau$ between the selector orderings induced by each pair of generators. Agreement is high throughout: the weakest pair is $\tau = 0.81$ (BM25 to History) and the mean over ordered pairs is 0.93. The disagreement that does occur is confined to adjacent, closely matched checkpoints; the best selector is the same one under every generator we built, including the signal-free control. Choosing the best-looking selector on one generator and deploying it on another therefore costs 0.0 points here, and is strictly suboptimal for 0 of 72 ordered pairs.

This is a null result; it is also the practically useful half of the picture, and it follows from the regularity above: a shift that depends on the generator and not on the selector moves the level without permuting the ranking. Comparative claims of the form "selector $A$ beats selector $B$" survive a

change of candidate provenance in our data; absolute claims of the form "our selector reaches $x\%$" do not, and Section 6 gives the correction.

# 6 ANALYSIS

## 6.1 ALTERNATIVE EXPLANATIONS

Controls rule out length, noise, issue-blind selection, order, and repository or issue concentration; Appendix L reports their measured bounds in full.

## 6.2 THE INSTRUMENT COMPOSES BACK INTO THE PIPELINE

The forced-pool estimate composes back into natural end-to-end accuracy at the unit level with 0.8 points of mean absolute residual. Aggregate multiplication and conditioning only on naturally covered units are materially weaker; Appendix F gives both analyses in full. Figure 4 plots the composition and the naive alternative.

## 6.3 AN UNSEEN GENERATOR'S DIFFICULTY IS LARGELY PREDICTABLE FROM ITS COVERAGE

The negative relation in Figure 5 is tight. Regressing pooled conditional recall on coverage across the 9 generators gives $R^2 = 0.97$ with slope -0.49: roughly half a point of conditional recall for every point of coverage. A coverage-only model cannot be exactly right, and the counterexample is in our own data: BM25 and Dense have *identical* coverage (48.7%) and conditional recalls differing by 4.5 points. Coverage is a strong summary of pool difficulty, not a sufficient statistic for it.

Nested-subset and leave-one-generator-out fits show that no anchor, mixture, or single generator carries the line; Appendix I reports the full sensitivity analysis.

**Usefulness of the correction.** Stability of the fit is not the same as usefulness, and the two come apart. Tested leave-one-generator-out per selector over the full coverage range, predicting a held-out generator from its coverage gives 2.2 points of error against 10.2 for transferring the number measured on BM25 — an advantage of 8.0 points, interval [+4.4, +9.2]. Restricted to the five real localisers, the only folds a practitioner actually faces, the errors are 2.5 against 6.5, but the advantage is 3.9 points with an interval of [-1.7, +5.9] that *includes zero*. The correction costs nothing and is not worse in expectation, but inside a band of localisers with similar coverage we cannot demonstrate that it beats simply reusing a BM25 number.

## 6.4 THE CORRECTION PREDICTS A GENERATOR WE DID NOT BUILD

On the held-out `SWE-bench_bm25_13K` generator, coverage-only prediction gives 69.0% against 70.5% measured: 1.5 points of error versus 2.5 for transferring BM25 unchanged. Appendix G gives the held-out construction and result in full.

# 7 CONCLUSION

A localisation selector does not have an accuracy. It has an accuracy *against a candidate generator*: holding the selector, the unit set, the pool size and the gold element byte-identical and changing only which generator supplied the nine distractors moves conditional recall by 8.2 points between two real localisers on average, an effect that survives an exact exchangeability test after the most distinctive generator and the dominant repository are both removed. The mechanism is ordinary — a generator that retrieves better fills the pool with code that is harder to tell apart from the gold element — and its regularity is what makes it usable: the shift is close to a constant offset per generator, so selector ordering survives, and conditional recall is near-linear in coverage, so an absolute number can be corrected for the provenance it will meet. The practical consequence concerns how these numbers are produced: report coverage and conditional recall separately, marginalise over candidate order, and correct absolute numbers for coverage. All three reuse the models, data and compute the evaluation already spends.

### AI USE STATEMENT

In this work, we used generative AI tools for language editing, code debugging, and improving the presentation of figures and tables. We also used them for limited brainstorming. All AI-assisted content was reviewed and verified by the authors, including code, experimental results, references, and scientific claims. The authors made all final decisions regarding the methodology, experiments, and conclusions, and take full responsibility for the final content of this work.

## REFERENCES

Allen Institute for AI. Olmo 3: Charting a path through the model flow to lead open source AI. https://allenai.org/papers/olmo3, 2025. Fully open 7B and 32B language models; weights, data and checkpoints released.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.426. URL https://aclanthology.org/2025.acl-long.426/.

Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. The power of noise: Redefining retrieval for RAG systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 719–729. ACM, 2024.

Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. Issue localization via LLM-driven iterative code graph searching. *arXiv preprint arXiv:2503.22424*, 2025. URL https://arxiv.org/abs/2503.22424. CoSIL; accepted by the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE 2025).

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474, 2020.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Microsoft. Phi-4-Mini technical report: Compact yet powerful multimodal language models via mixture-of-LoRAs. *arXiv preprint arXiv:2503.01743*, 2025. URL https://arxiv.org/abs/2503.01743.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 708–718. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.63. URL https://aclanthology.org/2020.findings-emnlp.63/.

OpenAI. Introducing SWE-bench Verified. https://openai.com/index/introducing-swe-bench-verified/, 2024. Human-validated 500-instance subset of SWE-bench.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI software engineering with repository-level code graph. *arXiv preprint arXiv:2410.14684*, 2024. URL https://arxiv.org/abs/2410.14684.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Qwen Team. Qwen3-VL-30B-A3B-Instruct. https://huggingface.co/Qwen/Qwen3-VL-30B-A3B-Instruct, 2025. Open-weight model release, Apache-2.0. Mixture-of-experts vision–language model, 30B total and about 3B active parameters; used here text-only as a selector.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4902–4912. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016. doi: 10.1109/TSE.2016.2521368.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024. URL https://arxiv.org/abs/2407.01489.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *The Ninth International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=zeFrfgyZln.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. AutoCodeRover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2024. doi: 10.1145/3650212.3680384.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=shr9PXz7T0.

Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pp. 14–24. IEEE, 2012. doi: 10.1109/ICSE.2012.6227210.

# A DATA CONSTRUCTION

**Unit selection.** We start from the 500 instances of SWE-bench Verified. An instance is dropped when its gold patch touches a non-Python file, when the gold file lies on a test path, when a patch hunk falls outside every extractable element, when the gold element exceeds the 60-line cap, or when more than four elements are needed to cover the patch. 269 instances survive across 11 repositories. Per-reason attrition counts are written by `code/build_universe.py` to `data/universe_stats.json` and shipped with the artifact.

**Element universe.** For each instance we read every non-test `.py` file in the repository at the base commit through `git cat-file --batch`, parse it with Python's `ast`, and keep every function or method of at most 60 lines as an element. Gold elements are those whose line span intersects a pre-image span of the gold patch. The universe and the gold set are computed once and are identical for every generator; this is what makes the generator the only thing that varies.

**Reproducibility of the unit set.** The unit set and the gold element sets are fixed once, before any generator runs, by `code/build_universe.py`, and `code/check_pools.py` re-verifies before any selector call that the gold element is byte-identical across all ten generators.

**Benchmark contamination.** SWE-bench Verified is public and predates every checkpoint in the panel, so we assume some of these issues and patches were seen in pretraining. Three features of the design bound what that can do to the conclusions. The estimand is a *difference* between generators measured on the same units with the gold element byte-identical, so a memorised answer raises every generator's recall together and cannot manufacture a transport gap. The issue-removal control puts accuracy at 22.8%, against a 16.1% chance rate, so selectors are not recovering gold locations from the repository alone. And conditional recall is *highest* on the signal-free Random pools (92.2%) and lowest on the strongest retrievers, the opposite of what recall of a memorised answer would produce. What contamination can still distort is the absolute level of every number, and those levels should not be read as localisation quality in an absolute sense.

**Licences and use.** SWE-bench Verified is publicly released by its authors for research use; the open-source repositories it draws on carry permissive licences (BSD-3-Clause, MIT, Apache-2.0, PSF) and are read at pinned base commits through read-only bare clones, so nothing is executed, modified, or redistributed. Issue reports are public GitHub text and can contain contributor names, handles, or email addresses inside quoted tracebacks; we extract, model, and report no personal attribute, and the released artifact contains candidate identifiers and per-decision records rather than fresh copies of issue text.

# B GENERATOR DETAILS

Each generator ranks files, keeps the top 25, extracts elements from those files, ranks the elements, and keeps the top ten.

**BM25.** Okapi BM25, $k_1 = 1.2$, $b = 0.75$, over identifier-split lowercase tokens of the issue against tokens of the file path and content (first 20,000 characters) at the file level, and against path, qualified name and body at the element level.

**Dense.** Mean-pooled `all-MiniLM-L6-v2` embeddings, 256-token truncation, cosine similarity against the embedded issue. File text is the path plus the first 4000 characters; element text is the path, qualified name and first 2000 characters of the body.

**Struct.** From the issue's fenced and inline code spans only, we extract file paths, dotted module names, and identifiers. A file scores on an exact or suffix path match, on a module-name match, and on the size of the intersection between the identifiers and the names it defines. Files that import a seeded module, or are imported by a seed, receive a one-hop score. Elements score on a leaf-name match, on qualified-name components, on the overlap between identifiers in the body and the issue's identifiers, and on their file's rank.

**LLM-A and LLM-B.** Two calls per instance against a locally served OpenAI-compatible endpoint, greedy, temperature 0, fixed seed. The first call shows the issue and a directory-grouped, per-directory-truncated listing of the repository's non-test Python files and asks for up to twelve paths. The second shows the issue and the qualified names defined in the returned files and asks for up to twelve names. Returned strings are resolved against the real path and qualified-name tables; anything unresolved falls back to the History ordering, so a refusal degrades gracefully rather than producing an empty pool. Neither call ever sees the gold patch, the gold path, or the test patch. Exact prompt templates are in `code/propose_llm.py`.

**History.** `git log` over the 4000 commits preceding the base commit; a file scores on its commit count plus a recency term. Elements inherit their file's score with a deterministic per-instance jitter. The generator never reads the issue.

**Random.** A per-instance seeded uniform permutation of the repository's files and elements.

**Derived generators.** Mix-7 and Mix-4 interleave their members' ranked lists round-robin, keeping the first occurrence of each element. $\text{BM25}_{\text{lm}}$ walks Dense's chosen distractors in order and, for each, takes the not-yet-used candidate from BM25's top 60 whose body length is closest, so the pool keeps BM25's ranking and Dense's length profile.

## C SELECTOR DETAILS

All selectors run BF16 with SDPA attention in `transformers`. Prompts are wrapped in each checkpoint's chat template with thinking disabled where the template supports it, and terminated with `Answer: [`. We take one forward pass, restrict the next-token logits to the first token of each label, renormalise, and take the argmax. Where a checkpoint exposes a trunk and an `lm_head` the trunk is run and only the last position projected; a conditional-generation wrapper is called whole instead. Both paths read the same logits.

The panel is split across two hosts by which host holds each checkpoint's weights: an NVIDIA A40 (46 GB) and an NVIDIA H100 (80 GB), one card at a time on each. A checkpoint never spans hosts, so the within-checkpoint comparison across generators, which is the estimand, is always made inside one hardware and library environment. Per-checkpoint host and library versions are in `results/runtime.json`.

Every checkpoint in the panel, both agentic generators, and the sentence embedder are open-weight models run locally with `TRANSFORMERS_OFFLINE=1`.

## D COMPUTE

Pool construction is CPU-only apart from the sentence embedder, which needs under 1 GB. Selector inference used a single card at a time on each of two hosts. Total measured GPU time and the per-checkpoint breakdown are in `results/runtime.json`.

One further open-weight checkpoint, `Qwen3.6-27B-FP8` (Apache-2.0), could not be run through the same numerical path as the rest of the panel and is excluded.

**Measurement factorisation.** The design is 269 units $\times$ 10 generators $\times$ 7 selectors $\times$ three orderings for the forced arm, plus one ordering of the natural arm for every selector, plus the two controls at one ordering each on the 4 selectors that reached them, plus the held-out external generator scored on every selector at the forced and natural arms: 104,372 scored decisions in total, at 11.1 GPU-hours of measured occupancy. Library versions are `transformers` 5.x with SDPA attention on PyTorch 2.10, and `all-MiniLM-L6-v2` for the dense generator; exact per-checkpoint versions and hosts are recorded in the released `runtime.json`.

## E PRE-DECLARED KILL CRITERIA

The frozen protocol lists conditions under which the study would have been abandoned rather than reported. All were checked mechanically, before the confirmatory runs and again after them, and all passed:

- the gold element is byte-identical across all ten generators for every unit;
- every pool holds exactly ten candidates;
- no candidate body is truncated, tested by whether it closes its own Python syntax rather than by inspecting its last characters;
- no rendered prompt exceeds the declared token cap;
- re-scoring an identical rendering reproduces at least 98% of decisions.

The first four passed outright. The fifth passed on the average (99.1%) but not in every cell: 3 of 40 selector–generator cells fell under it, the worst reproducing 97.4% of decisions. We did not treat that as a kill, because the criterion exists to catch a measurement dominated by numerical noise and a 2.6% worst-case noise level is several times smaller than the differences we report. The verification script (`code/check_pools.py`) ships with the artifact and prints each criterion with its measured value.

## F VALIDATING THE FORCED INSTRUMENT

$R$ is measured on forced pools. Two checks on the natural arm, neither used to fit anything, validate it as an instrument.

First, the factorisation. If $R$ really is the selector's behaviour given that the answer is present, then end-to-end accuracy on a generator's own pools should be its coverage times that quantity, *unit by unit*. Across the 70 (selector, generator) cells the residual $\mathrm{acc} - \overline{\mathrm{cov}_u R_u}$ has mean absolute value 0.8 points, largest 3.7, mean signed -0.5. The instrument composes at that level: coverage and conditional recall together predict what the pipeline achieves, without running the pipeline.

The same statement between column *means* is materially weaker. $\mathrm{acc} - \overline{\mathrm{cov}}\,\overline{R}$ has mean absolute value 2.2 points, worst cell 9.5, mean signed +2.0. The difference is $\mathrm{Cov}_u(\mathrm{cov}_u, R_u)$, which is positive throughout: a generator covers the units it finds easy, and those are the units its pools are easiest to select from. Only the unit-level identity licenses the factorisation; the aggregate product is a lower bound in practice, not an equality.

Second, the alternative one might reach for instead. Rather than forcing the gold element in, one could measure conditional recall on natural pools by restricting to the units a generator happened to cover. That estimator disagrees with $R$ by 8.0 points on average and by +10.9 on Mix-7, and its pairwise gaps track the identified ones at only $r = 0.37$ over 140 ordered pairs. The reason is the confound the forced construction exists to remove: the units a generator covers are the units it finds easy, so the naive estimator measures a generator's choice of units as much as a selector's discrimination. Forcing is what makes the comparison identified.

**Composition summary.** $R$ is measured on a construction that is deliberately not a deployment scenario, but it composes back into one at the unit level: across the 70 (selector, generator) cells, the residual $\mathrm{acc} - \overline{\mathrm{cov}_u R_u}$, taking the product *within* each unit before averaging, has mean absolute value 0.8 points. The product of the two aggregate means is materially weaker — 2.2 points on average, 9.5 at worst — because coverage and conditional recall covary positively across units. The alternative of measuring conditional recall on natural pools restricted to covered units disagrees with $R$ by 8.0 points and tracks the identified pairwise gaps at only $r = 0.37$, because the units a generator covers are the units it finds easy. The analyses above give both cases in full.

## G THE HELD-OUT EXTERNAL GENERATOR

The published artifact is `SWE-bench_bm25_13K`, released with the benchmark. For each instance it supplies a retrieval context assembled from whole files up to a 13K-token budget; we recover the

ordered file list from that context and use it as the generator's file stage, keeping our own element stage, pool construction, prompt and decoding rule. All 269 units of our frozen set appear in the artifact, and the median instance contributes three files, so it is a narrower file ranking than the 25 our own generators keep.

Before any selector was called we checked the property the comparison depends on: the gold element sets of the external pools are byte-identical to the main pools' on all 269 units, the unit sets are equal, every pool holds ten candidates, and every candidate body parses. The generator is excluded from the panel, from the coverage fit, and from every reported number other than the out-of-sample test in Section 6.

**Held-out prediction result.** Every generator in the main panel is our own instrument, so a correction fitted on them might describe only generators we build. We therefore tested it on one we did not: the BM25 file ranking released with the benchmark as `SWE-bench_bm25_13K`, scored on the same units, the same gold elements, the same element stage and the same prompt, and held out of the fit and of every number reported so far.

Its coverage, 44.2%, sits inside the band the deployable localisers occupy. Predicting its conditional recall from that coverage alone gives 69.0% against a measured 70.5%: 1.5 points of error across the 7 selectors, against 2.5 for transferring each selector's BM25 number unchanged. The correction survives contact with a candidate distribution produced by someone else, which is the case it was proposed for.

## H THE LENGTH CONTROL AND THE MIXED-PROVENANCE BENCHMARK

Retrieval families disagree about how long a candidate is as well as about which one it is: lexical scoring rewards long functions, an embedding truncated at 256 tokens rewards short ones, and mean distractor length differs by a factor of 2.9× across generators. $\text{BM25}_{\text{lm}}$ tests that alternative explanation directly, keeping BM25's ranking but re-picking candidates to carry Dense's length profile. If length drove the result it would sit at Dense's recall; instead it stays within 1.4 points of BM25 and differs from Dense by 3.1 points (Figure 3, right). The control runs on the BM25–Dense contrast because that pair differs in length profile at identical coverage; it does not by itself rule length out of the larger gaps involving the anchors.

**Mixed-provenance pools.** A natural alternative benchmarks on a pool whose distractors are drawn round-robin from every generator family. It does not help. The four-way mixture predicts held-out generators at 9.2 points, no better than BM25, and the seven-way mixture is worse than either, because interleaving every family produces the highest-coverage and therefore hardest pool in the study rather than a representative one. Diversity of provenance is not the operative variable; where a pool sits on the coverage axis is. Figure 2(b) reports the leave-one-generator-out prediction errors.

## I COVERAGE-CORRECTION SENSITIVITY

**Nested-subset refits.** Refitting on nested subsets says the line is not carried by its two near-zero-coverage anchors or its two derived generators: -0.49 on all 9 generators [-0.68, -0.39], -0.45 with both anchors deleted, -0.49 with both mixtures deleted, and -0.45 on the five deployable localisers alone, whose coverages span only 17.5–48.7%. $R^2$ stays between 0.91 and 0.97 throughout. Deleting any single generator from the pooled fit moves the slope by at most 0.027 (worst case Random) and never drops $R^2$ below 0.97.

The same fit also prices the retrieval stage: at BM25's coverage one extra point of coverage returns 0.43 points of end-to-end accuracy rather than the 0.67 a reader would infer, the rest eaten by the harder pool a better retriever produces. Mixing provenances, the other obvious fix, does not work at all (Appendix H).

| Generator | Cov. (%) | Median rank | Mean same-file | Mean chars | Mean tokens |
|---|---|---|---|---|---|
| BM25 | 48.7 | 12 | 1.26 | 1311 | 3449 |
| Dense | 48.7 | 11 | 2.26 | 656 | 2241 |
| Struct | 17.5 | n.r. | 1.52 | 653 | 2215 |
| LLM-A | 30.5 | 447 | 1.29 | 613 | 2178 |
| LLM-B | 46.1 | 19 | 1.49 | 575 | 2086 |
| History | 0.7 | n.r. | 0.34 | 628 | 2203 |
| Random | 0.4 | n.r. | 0.12 | 457 | 1911 |
| Mix-7 | 58.7 | 5 | 1.31 | 792 | 2497 |
| Mix-4 | 40.1 | 24 | 0.88 | 833 | 2580 |
| $\text{BM25}_{\text{lm}}$ | 0.0 | n.r. | 1.82 | 707 | 2341 |

Table 1: What each generator produces, measured before any selector is called. Cov. is the percentage of issues whose top-10 contains a gold element; Median rank is the median rank of the first gold element (n.r. when the median is outside the ranked list); Mean same-file is the mean number of the nine distractors that sit in the gold file; Mean chars is the mean distractor body length; Mean tokens is the mean rendered prompt length. The gold element is identical across every row.

| From ↓ / to → | BM25 | Dense | Struct | LLM-A | LLM-B | History | Random | Mix-7 | Mix-4 |
|---|---|---|---|---|---|---|---|---|---|
| BM25 | 0.0 | -4.5* | +12.1 | +7.3 | +1.5 | +21.1* | +24.2* | -5.9* | +4.2 |
| Dense | +4.5* | 0.0 | +16.6* | +11.8* | +6.0* | +25.5* | +28.6* | -1.4* | +8.7* |
| Struct | -12.1 | -16.6* | 0.0 | -4.9 | -10.6 | +8.9* | +12.0* | -18.0* | -7.9 |
| LLM-A | -7.3 | -11.8* | +4.9 | 0.0 | -5.8 | +13.8* | +16.9* | -13.2* | -3.0 |
| LLM-B | -1.5 | -6.0* | +10.6 | +5.8 | 0.0 | +19.5* | +22.6* | -7.4* | +2.7 |
| History | -21.1* | -25.5* | -8.9* | -13.8* | -19.5* | 0.0 | +3.1 | -27.0* | -16.8* |
| Random | -24.2* | -28.6* | -12.0* | -16.9* | -22.6* | -3.1 | 0.0 | -30.1* | -19.9* |
| Mix-7 | +5.9* | +1.4* | +18.0* | +13.2* | +7.4* | +27.0* | +30.1* | 0.0 | +10.1* |
| Mix-4 | -4.2 | -8.7* | +7.9 | +3.0 | -2.7 | +16.8* | +19.9* | -10.1* | 0.0 |

Table 2: Transport matrix: change in pooled conditional recall (points) when a selector measured on the row generator is deployed on the column generator. Positive means the deployed value is higher than the measured one. * marks the pre-registered repository-clustered bootstrap interval excluding zero; Section 5 shows that criterion is anti-conservative at this benchmark's 11 unbalanced clusters and replaces it with an exact permutation-calibrated test.

## J FIGURES AND FULL RESULT TABLES

Table 1 describes what each generator produces before any selector is called, and Table 2 gives the transport matrix. Table 3 lists conditional recall for every selector and generator with its interval, and Table 4 decomposes each generator's own-pool accuracy into coverage, conditional recall and oracle regret. Figure 6 shows conditional recall for every checkpoint, Fig. 7 splits end-to-end accuracy into what the selector achieves and the oracle regret above it, and Fig. 8 compares the selector orderings across generators. Figure 5 plots each generator's coverage against its conditional recall and shows the transport matrix.

## K PAIRWISE CALIBRATION AND REPOSITORY SENSITIVITY

The exact permutation test of Section 5 is repeated in three progressively harsher forms: with the most distinctive generator removed, with the largest repository removed, and with both removed. Removing Dense entirely leaves 7.1 points ($p < 0.001$). Removing instead the django/django repository, which alone supplies 54% of the units, leaves 5.8 points ($p < 0.001$). Removing both at once still leaves 4.1 points ($p = 0.0076$). Pool provenance changes what a selector does, and not because of one generator or one codebase. Figure 9 plots each observed gap against its exact null distribution.

**Which individual contrasts are established.** Our pre-registered pairwise criterion was a repository-clustered bootstrap interval, and it turned out to be the wrong instrument at 11 clusters this unbalanced. Measured under the exchangeability null above rather than assumed, its null intervals reject at 11.3%

| Selector | BM25 | Dense | Struct | LLM-A | LLM-B | History | Random | Mix-7 | Mix-4 |
|---|---|---|---|---|---|---|---|---|---|
| Qwen3-1.7B | 62 [58,65] | 59 [51,64] | 81 [64,88] | 74 [64,80] | 68 [59,73] | 90 [86,93] | 94 [89,96] | 59 [52,63] | 68 [65,71] |
| Qwen3-4B | 73 [69,76] | 70 [61,73] | 86 [70,94] | 81 [71,86] | 74 [66,79] | 94 [89,96] | 96 [91,97] | 67 [59,70] | 78 [70,81] |
| Phi-4-mini | 55 [51,59] | 47 [34,53] | 61 [48,67] | 57 [42,64] | 55 [42,61] | 73 [64,77] | 77 [68,81] | 43 [33,48] | 56 [47,61] |
| Qwen2.5-7B | 74 [72,79] | 70 [64,74] | 85 [68,92] | 78 [67,85] | 72 [64,79] | 94 [87,96] | 96 [91,98] | 69 [59,73] | 77 [70,80] |
| Olmo-3-7B | 55 [48,59] | 49 [40,55] | 68 [51,76] | 66 [56,71] | 59 [49,65] | 82 [74,86] | 87 [81,89] | 51 [42,55] | 63 [53,67] |
| Qwen3-8B | 77 [73,83] | 73 [65,79] | 89 [74,96] | 84 [77,87] | 77 [69,81] | 95 [89,97] | 97 [92,99] | 69 [63,73] | 80 [73,82] |
| Qwen3-VL-30B-A3B | 81 [78,88] | 78 [72,84] | 91 [78,97] | 87 [78,91] | 81 [73,86] | 96 [90,99] | 98 [94,99] | 78 [69,82] | 85 [79,87] |
| Mean | 68.0 | 63.6 | 80.2 | 75.3 | 69.6 | 89.1 | 92.2 | 62.1 | 72.3 |

Table 3: Conditional recall (%) at the function level: the pool contains a gold element by construction and holds 10 candidates, the gold element is byte-identical across columns, and only the nine distractors change. Each cell is the mean over three orderings of the same pool and 269 repository-clustered issues. Brackets give the repository-clustered bootstrap interval of Section 4, which Appendix K finds anti-conservative at this benchmark's unbalanced repository clusters.

| Generator | Cov. | Cond. rec. | End-to-end acc. | Oracle regret |
|---|---|---|---|---|
| BM25 | 48.7 | 68.0 | 35.7 | 13.0 |
| Dense | 48.7 | 63.6 | 33.6 | 15.1 |
| Struct | 17.5 | 80.2 | 12.7 | 4.8 |
| LLM-A | 30.5 | 75.3 | 25.4 | 5.1 |
| LLM-B | 46.1 | 69.6 | 36.4 | 9.7 |
| History | 0.7 | 89.1 | 0.7 | 0.0 |
| Random | 0.4 | 92.2 | 0.4 | 0.0 |
| Mix-7 | 58.7 | 62.1 | 42.9 | 15.8 |
| Mix-4 | 40.1 | 72.3 | 31.7 | 8.4 |

Table 4: Decomposition on the generator's own top-10 pools. Coverage is a property of the generator alone; conditional recall is measured on gold-forced pools; end-to-end accuracy and oracle regret are means over the selector panel. All values are percentages; lower oracle regret is better. The factorisation holds unit by unit, not between column means: coverage and conditional recall covary positively across units, so multiplying these two aggregates understates end-to-end accuracy by 2.2 points on average and by 9.5 in the worst cell (Section 6).

against a nominal 5% while still marking only 8 of 20 ordered pairs — anti-conservative and underpowered at once. Table 2 keeps those calls for continuity with the pre-registered plan.

The same permutation construction that answers the omnibus question also calibrates the pairwise one, and is exact at any number of clusters. Taking the null distribution of the per-pair $|\Delta|$ statistic over 2,000 permutations gives a critical value of 2.9 points for a single pair, and 3.5 points for the maximum over all pairs, which controls the family-wise error rate. Against the family-wise threshold, 9 of 10 deployable pairs differ; the one exception is BM25 vs LLM-B, at 2.1 points. The effect is present between almost every pair of real localisers, and the clustered interval's failure to resolve those pairs is a property of the estimator, not of the data.

One caveat keeps this in proportion: a permutation test conditions on the units observed, so it establishes the effect *on this issue set* exactly and says nothing by itself about a new population of repositories. As a partial substitute, deleting any single repository leaves the mean absolute gap between 5.8 and 8.9 points.

Per checkpoint the worst-case gap ranges from 20.2 to 37.8 points, so no checkpoint is exempt.

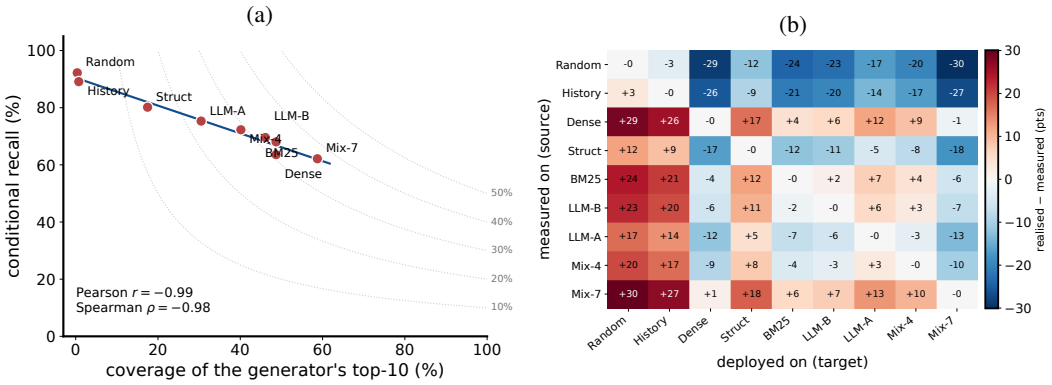



Figure 5: **(a)** Coverage against conditional recall, one point per generator, over curves of constant end-to-end accuracy. Retrieving better puts the answer in the pool more often and makes it harder to pick out. **(b)** Transport matrix. Each cell is realised minus measured: the change in pooled conditional recall when a selector measured on the row generator is deployed on the column generator. Negative means the deployed value is lower than the benchmark reported.

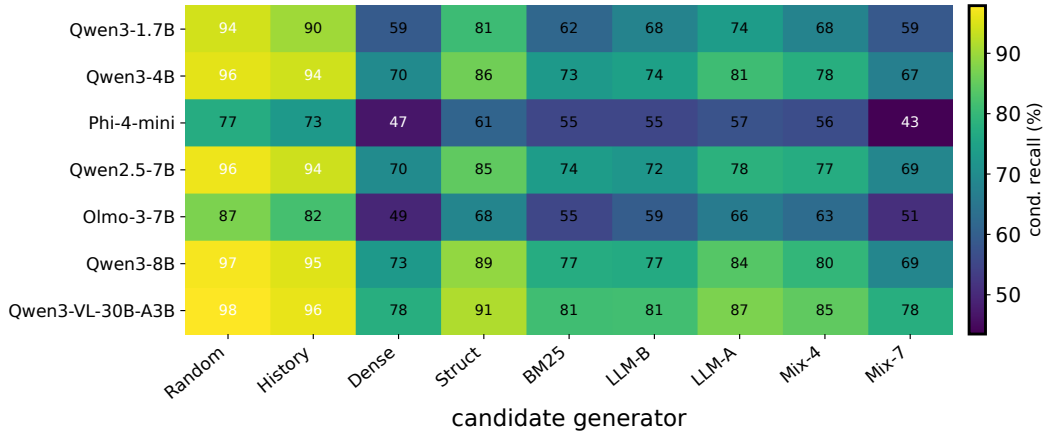



Figure 6: Conditional recall for every selector–generator pair. Rows are checkpoints ordered by size; the generator effect is visible in every row.

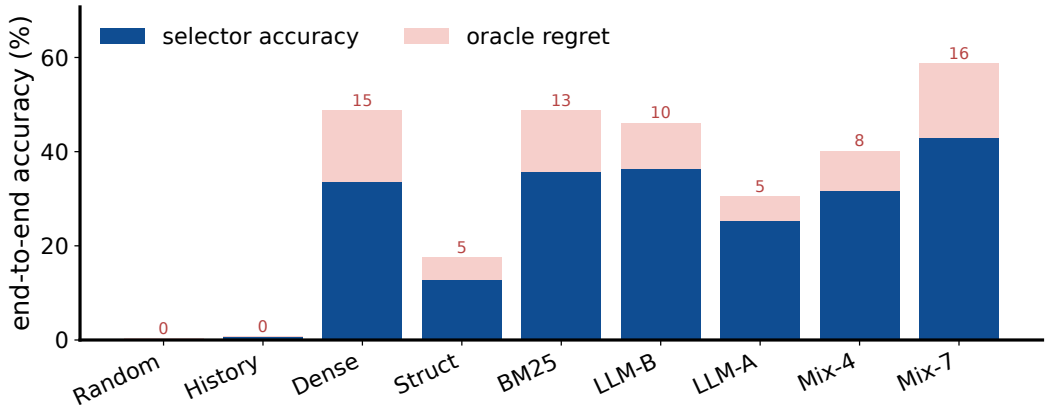



Figure 7: End-to-end accuracy on each generator's own top-10 pools, split into what the selector achieves and the oracle regret above it. Bar height is coverage.

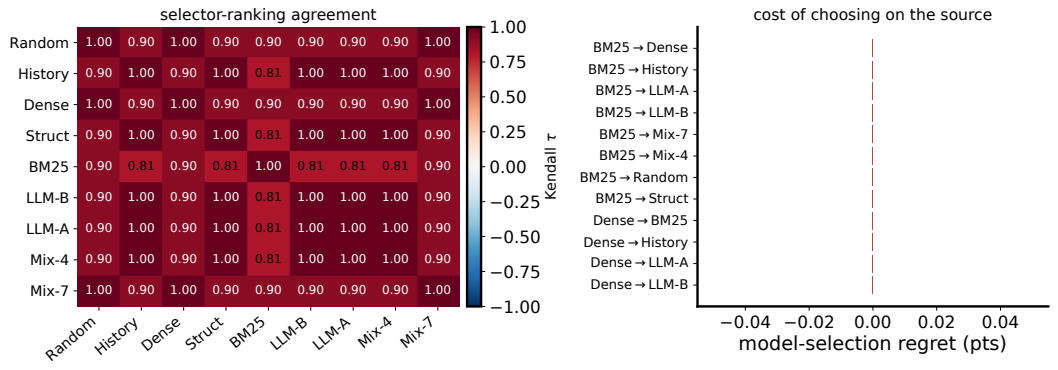



Figure 8: Left: Kendall's $\tau$ between the selector orderings induced by each pair of generators; agreement is high everywhere. Right: the conditional recall lost by choosing the best-looking selector on the source generator and deploying it on the target. Every value in this panel is exactly zero, because the selector picked on the source is also the best on the target for every ordered pair, so every bar has zero length and nothing is drawn inside the panel frame.

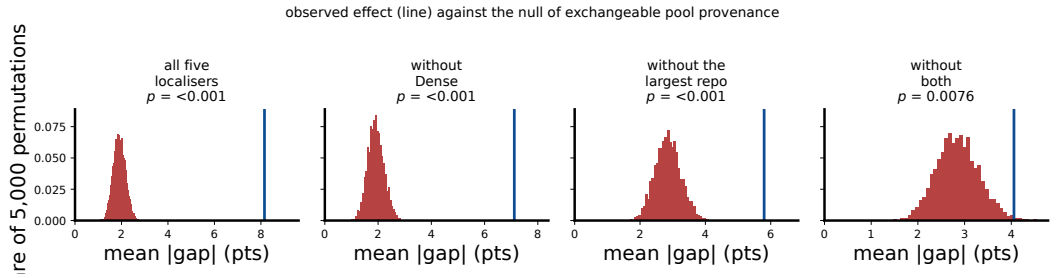



Figure 9: The observed mean absolute transport gap among deployable localisers (vertical line) against the exact null distribution obtained by permuting generator identity within each (selector, unit) cell. The effect survives deleting the most distinctive generator, the repository supplying 54% of the units, and both at once.

## L Control analyses

Five controls bound alternative explanations. *Length.* $\text{BM25}_{\text{lm}}$ — BM25's ranking carrying Dense's length profile — stays within 1.4 points of BM25 and 3.1 from Dense, so the effect is content, not length. *Noise.* Re-scoring an identical rendering in a different batch reproduces 99.1% of decisions (3 of 40 cells fall under the pre-declared 98% floor, the worst at 97.4%); the gaps we report are several times that noise level. *The issue text.* Replacing the issue with a placeholder drops accuracy to 22.8%, against a 16.1% chance rate — the gold set holds 1.61 elements on average, so the right baseline is not $1/K$ — and 45.3 points below the full panel's recall on BM25 pools with the issue present (the issue-blind control ran on 4 of the 7 selectors, so the two averages are over overlapping, not identical, selector sets). Most discrimination comes from reading the issue, and the residual structural signal is shared by every generator. *Candidate order.* Order sensitivity tracks pool hardness ($r = -0.97$), which is why every number here is marginalised over three orderings. Figure 10 summarises the repeat, issue-removed, and ordering controls. *One repository, or one kind of issue.* Within-repository estimates agree with the pooled ordering at mean Spearman $\rho = 0.82$ over 8 repositories, with the headline sign holding in 8 of 8 and across 3 difficulty bands (Figure 11).

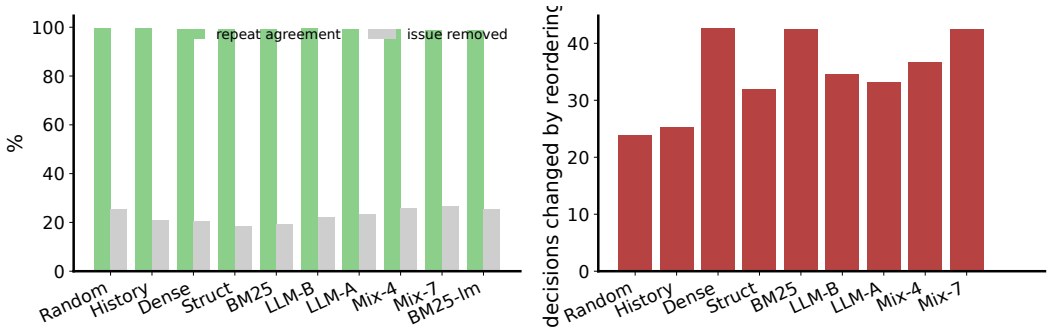



Figure 10: Left: repeat agreement and issue-removed accuracy per generator. Right: the fraction of decisions that change when the same pool is reordered.

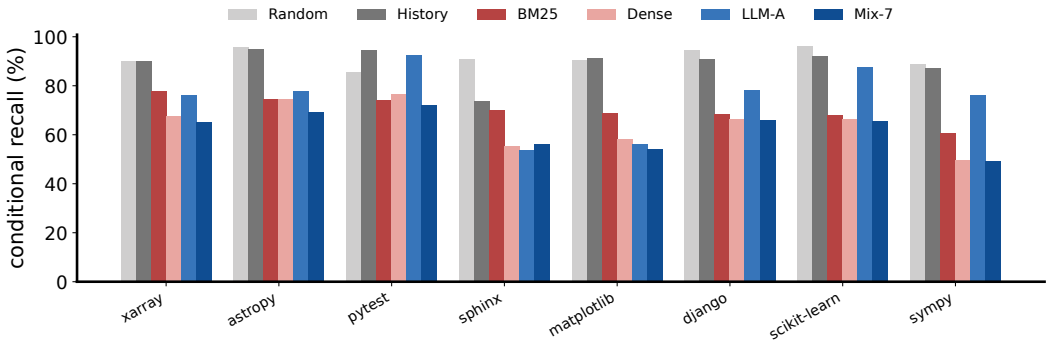



Figure 11: Conditional recall by repository, for the repositories with enough units to support a within-repository estimate. The generator effect is present in each of them rather than being carried by one.