Hindsight:
A Git Intelligence Layer

Making Large Codebases Easier to Navigate with AI.

University of Massachusetts Amherst | 698DS

Team:
Mustafa Ali
Vanshika Agrawal
Priya Balakrishnan
Sheng Kai Wen

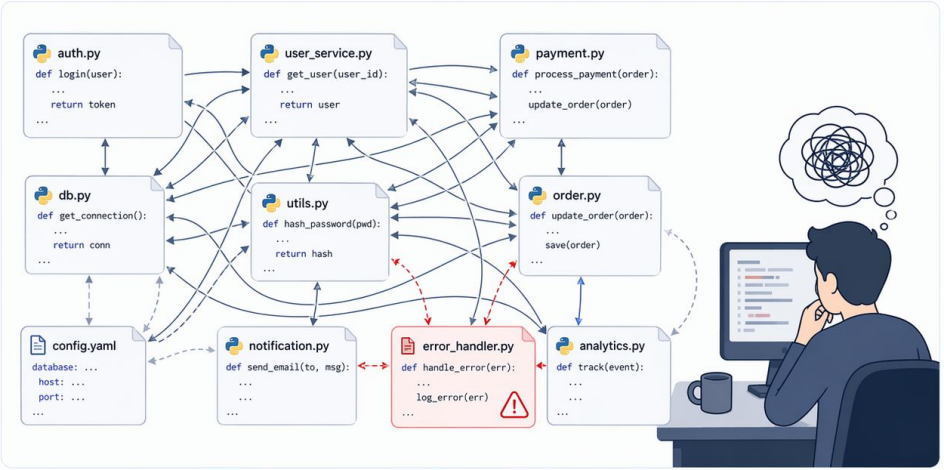
PhD Mentor:
Anshita Gupta

Mentors:
Ryan Lingo
Rajeev Chhajer

Problem Statement

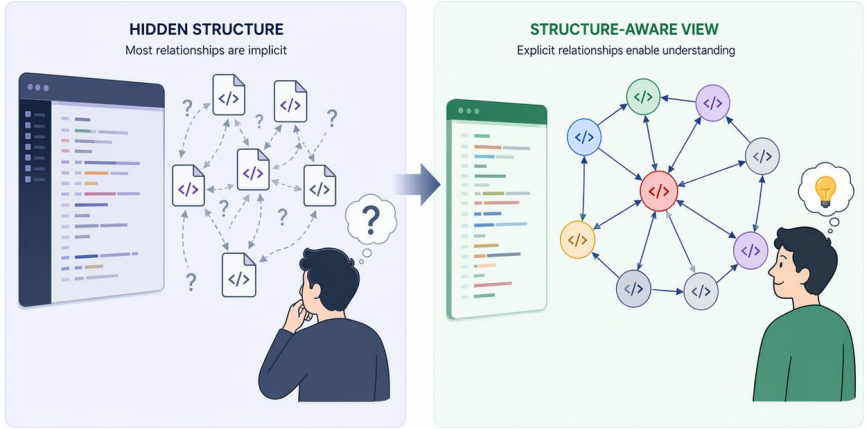
Modern software systems are large, modular, and deeply interconnected, oftenspanning hundreds of files and components. Developers struggle with:

- Understanding cross-file and module interactions
- Tracing dependency chains across the system
- Predicting impact of small changes
- Debugging hidden cross-component failures



Our Motivation

- Software complexity is outpacing understanding
- Structural knowledge is implicit and undocumented
- Misunderstanding leads to:
  - Integration failures
  - Debugging delays
  - Higher engineering cost
- Current AI tools rely on text, not structure



Research Questions

Can graph representations of code repositories improve retrieval-augmented generation (RAG) performance on multi-file code understanding tasks?

Why Graph-Augmented Retrieval over text Only Retrieval (vs BM25 Alone)

What current methods do

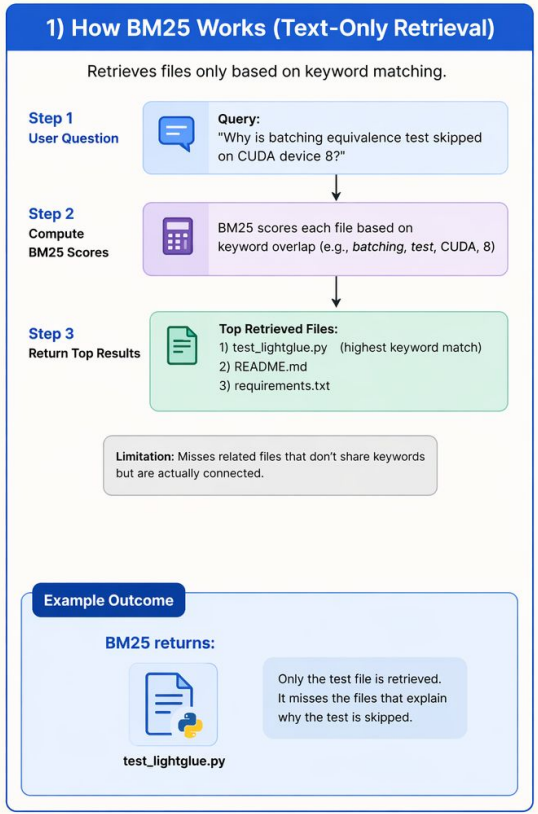
- Retrieves code chunks based on **keyword overlap** (BM25) or **semantic similarity** (embeddings)
- Treat repository as independent **text chunks**

What they miss

- Ignore how code is connected across files
- Fail to capture:
  - Function call chains
  - Cross-file dependencies
  - Shared state / configuration

Why this matters

- Real-world questions require multi-file reasoning
- Critical context may exist in files that are not textually similar



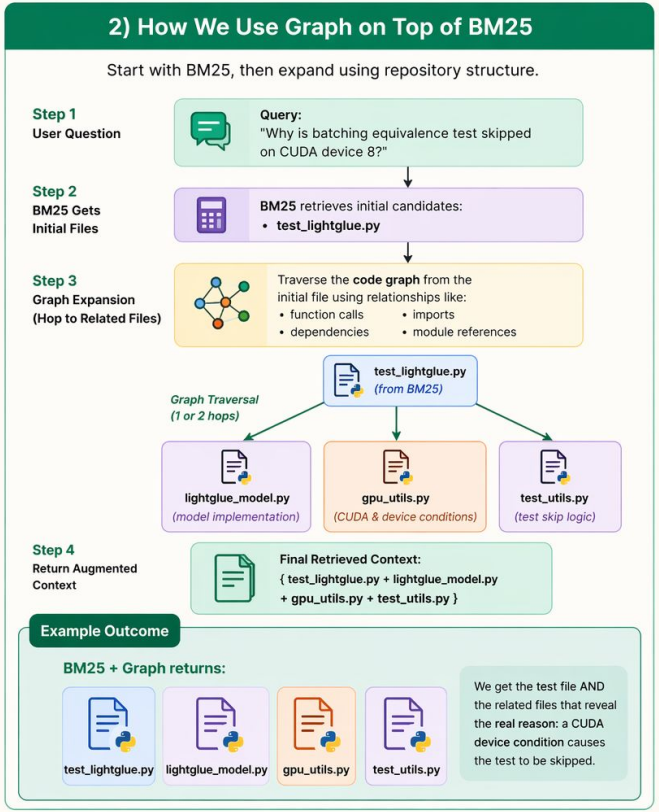
Why Graph-Augmented Retrieval over text Only Retrieval (vs BM25 Alone)

Our Approach

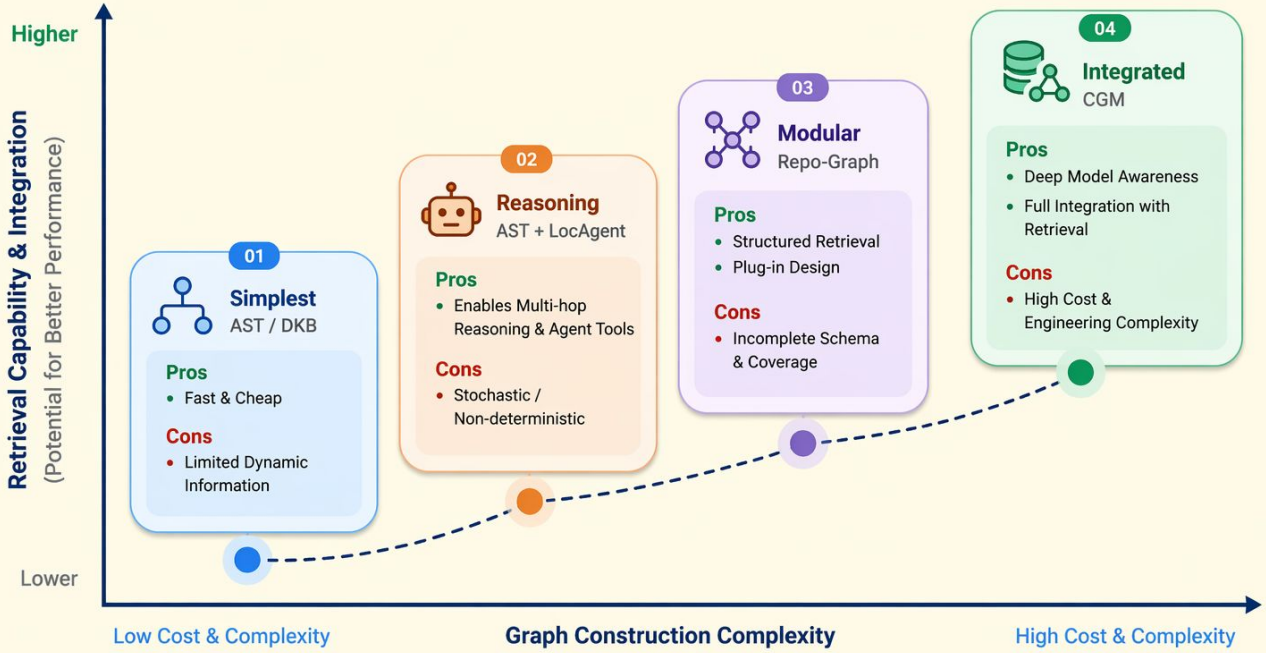
- L1: Use BM25 to get initial relevant files
- L2: Expand using graph over:
  - Function calls
  - Imports / dependencies
  - Module relationships

What this enables

- Retrieve connected files across the repository
- Build multi-hop context for reasoning
- Improve grounding for LLM answers



What graph construction strategies balance performance and cost?



Related Work

Related work converges on two themes: realistic repository-level benchmarks and structure-aware retrieval over codebases.

Repository QA benchmarks

DeepCodeBench

Qodo 2025

PR-grounded multi-file QA benchmark over 8 OSS repositories. Uses fact recall and answer-quality judging to score generated responses.

Why it matters: Directly matches our answer-generation setting and lets us test whether structured retrieval improves grounded reasoning.

SpyderCodeQA

ACL SRW 2024

325 repository-level Q&A pairs covering source-code semantics, dependencies, and repository meta-information.

Why it matters: Shows that repository QA must capture cross-file relations, not just single-snippet understanding.

CoReQA

arXiv 2025

Issue-derived benchmark for repository QA from real GitHub issues and comments, evaluated with LLM-as-a-judge.

Why it matters: Makes repo QA more realistic by tying questions to developer-facing issues and discussions.

Shared signal: *all three benchmarks emphasize multi-file retrieval before answer generation.*

Localization and graph-based methods

LocBench + LocAgent

ACL 2025

LocBench covers bug, feature, performance, and security issue localization. LocAgent uses graph-guided exploration over a heterogeneous code graph.

Why it matters: Better localization narrows the search space before downstream reasoning or answer generation.

RepoGraph

ICLR 2025

Repository-level code graph plug-in that gives AI systems fine-grained repository navigation and context aggregation.

Why it matters: Motivates adding structure-aware exploration instead of relying only on flat chunk retrieval.

Code Graph Model (CGM)

arXiv 2025

Graph-integrated LLM that injects code graph structure into the model and pairs it with graph RAG for repository-level tasks.

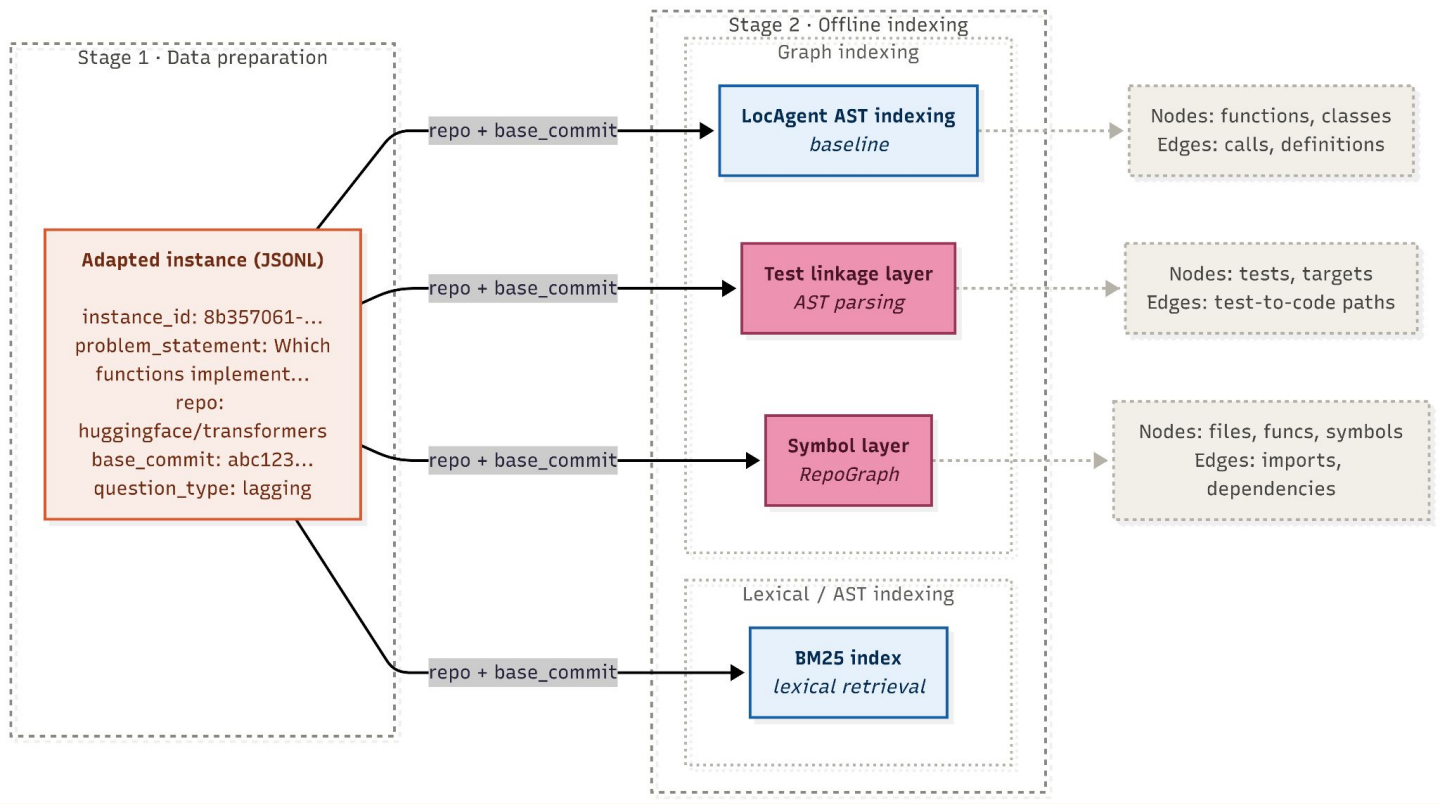
Why it matters: Suggests that structural signals can help both retrieval and the model's internal representation of the repository.

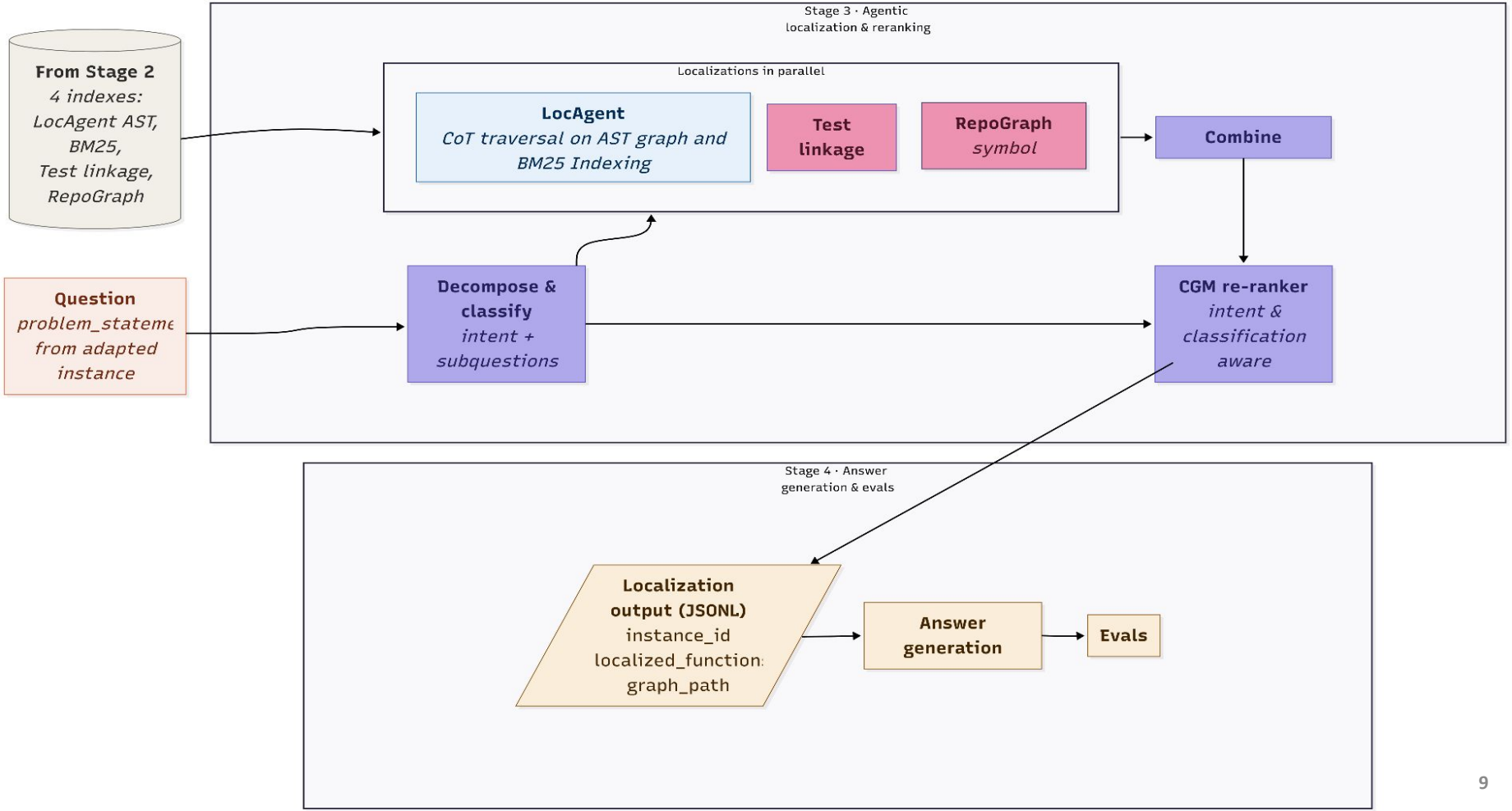
Our positioning *We sit between benchmark-driven QA evaluation and graph-guided repository navigation.*

Takeaway: repository understanding improves when evaluation is realistic and retrieval is structure-aware.

Method

Proposed Solution




From Stage 2
4 indexes:
LocAgent AST,
BM25,
Test linkage,
RepoGraph
Stage 3 · Agentic
localization & reranking
Localizations in parallel
LocAgent
CoT traversal on AST graph and
BM25 Indexing
Test
linkage
RepoGraph
symbol
Combine
Question
problem_stateme
from adapted
instance
Decompose &
classify
intent +
subquestions
CGM re-ranker
intent &
classification
aware
Stage 4 · Answer
generation & evals
Localization
output (JSONL)
instance_id
localized_function
graph_path
Answer
generation
Evals

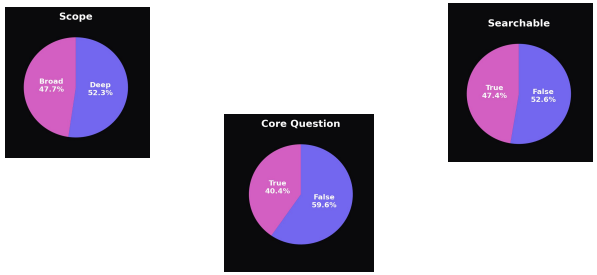
Experiments

Datasets

DeepCodeBench

1,144 total

Q&A benchmark



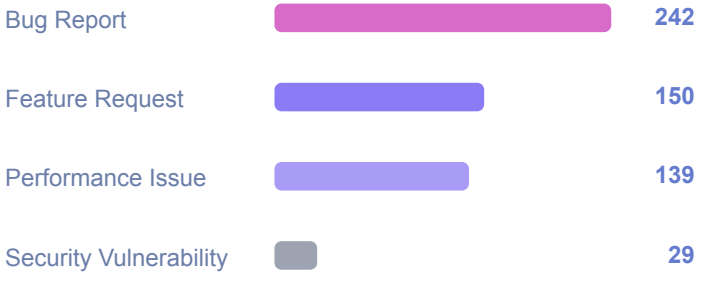
Scope: Question spans single block/multiple files
Type: Question targets core functionality or peripheral details
Searchable: Question includes location hints or not

Measures the quality of the model's final coding answer.

LocBench

560 total

Localization benchmark



Issue descriptions are linked to the relevant edited functions or code regions.

Measures whether the system can find where the problem is.

Intent Categories

→ Leading

When: proactive • pre-change

Focus: impact • consequences • planning

Example cues

- change impact
- downstream effect
- design / implement

! Lagging

When: reactive • post-failure

Focus: debugging • diagnosis • root cause

Example cues

- why broke?
- what caused it?
- unexpected behavior

? Exploratory

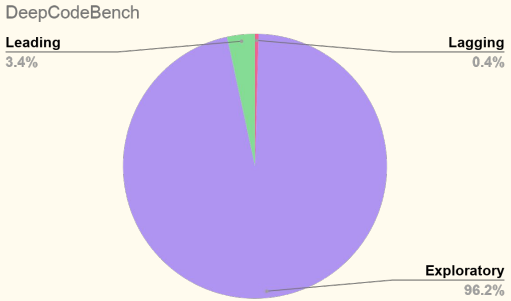
When: neutral • no active failure

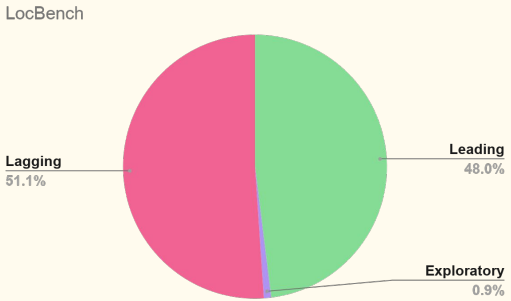
Focus: structure • mechanism • lookup

Example cues

- where is X?
- how does Z work?
- which component?

Dataset Distribution





Evaluation Methods

Q&A Evaluation

Fact Recall

Measures: Coverage of gold reference facts in the generated answer

Formula:
Fact Recall =
(# facts judged present) / (# total gold facts)

Judge Signal:
Fact-level YES / NO

Quality Evaluation

Measures: Overall answer quality relative to the reference answer

Dimensions:
Accuracy • Completeness • Relevance • Clarity

Judge Setup:
1 to 10 scale • repeated calls • average

Localization Evaluation

Acc@k

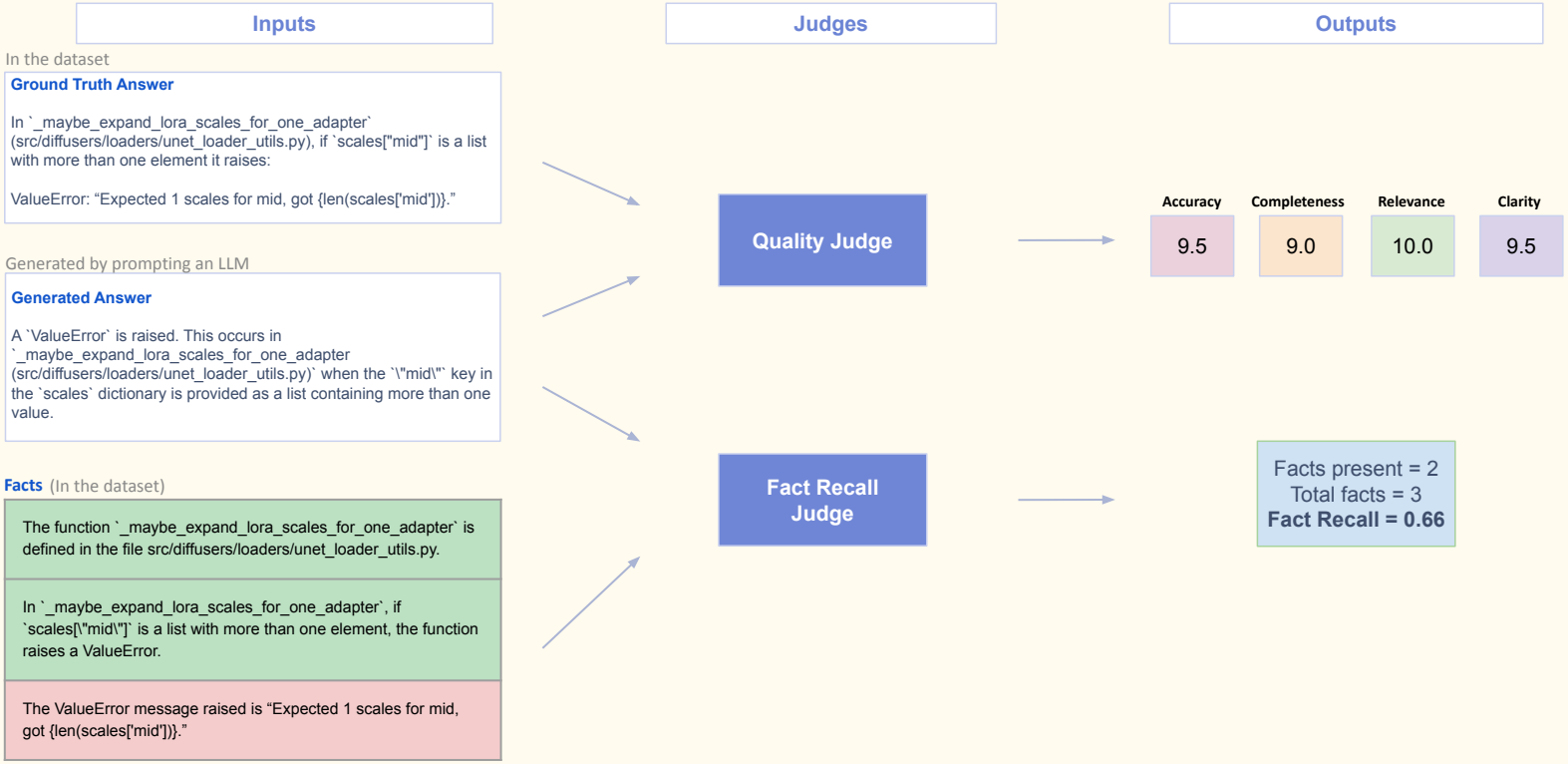
Measures: Overall localization quality i.e, whether the correct output is in the top-k predicted locations

Example

In the dataset

What error is raised when the 'mid' scale is provided as a list of more than one value?

Scope: Deep, Searchable: False, Type: Non-Core



Baselines

Answer Generation

Localization

Limited Context

Input: Repository name only

Tests: Prior-based answering without grounding

Lower bound

Full Context: Issue / PR

Input: PR title, body, and anchored code snippets

Tests: Value of problem statement and local snippets

Problem-statement grounding

BM25

Input: Top-k repo chunks via sparse retrieval

Tests: Lexical retrieval quality

Sparse RAG baseline

CodeRankEmbed

Input: Top-k repo chunks via dense retrieval

Tests: Semantic retrieval quality

Dense RAG baseline

LocAgent

Input: Tool-based exploration over the code graph

Tests: Structured navigation and bug localization

Agentic localization

Increasing access to repository evidence

No grounding

Problem text only

Retrieved repo context

Structured exploration

Implementation Details

Off-the-Shelf Assets

What we reuse directly

Benchmarks
DeepCodeBench • LocBench

Task models
Qwen3-32B • Qwen2.5-7B • gpt-5.3-codex

Judge
Qwen3-32B thinking mode

Retrieval
BM25 • CodeRankEmbed

Benchmarks + models + evaluation

Software Stack

Core implementation layer

Core ML/NLP:
Python • PyTorch • HF transformers • datasets

Indexing:
Bm25s • rank_bm25 • faiss-cpu

Tooling:
OpenAI API • llama-index

Graph Methods:
AST • Dependency • Structural • Knowledge

Retrieval + orchestration

Computational Resources

How runs are executed

Requirements:

CPU for Retrievals
CPU for Graph Constructions
GPU for Localizations and Q&A

Cluster-scale evaluation

Results

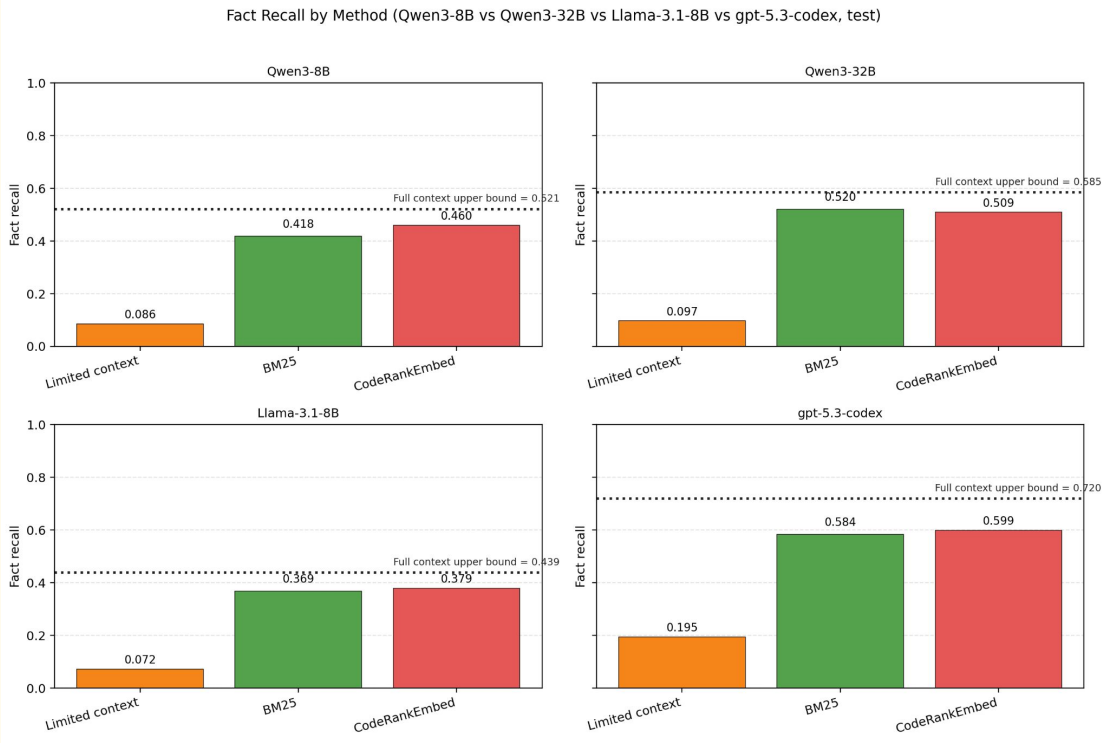
How does performance vary across DeepCodeBench categories?

How reliable is the LLM judge across repeated evaluations and answers from different models?

How do Qwen3-32B and gpt-5.3-codex differ in answer quality?

How do BM25 and CodeRankEmbed compare as retrieval methods?

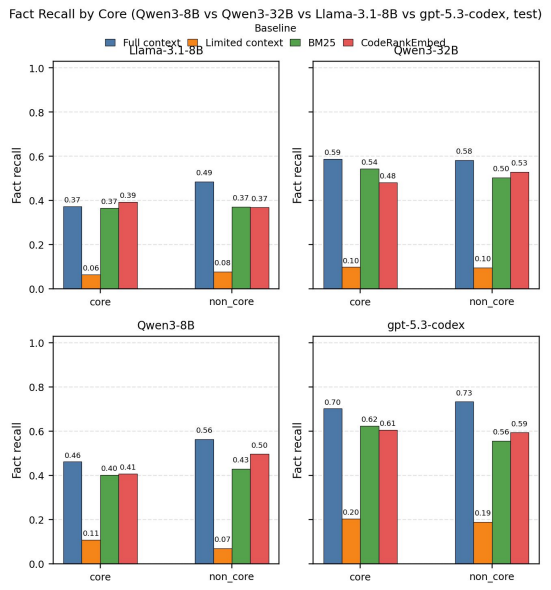
Fact Recall on DeepCodeBench

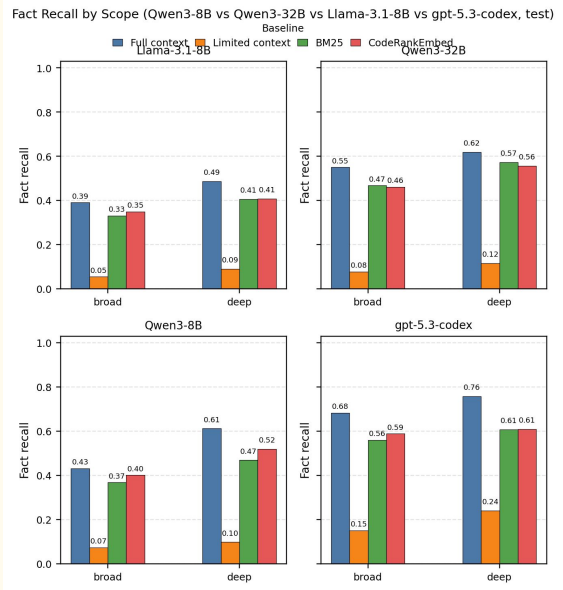


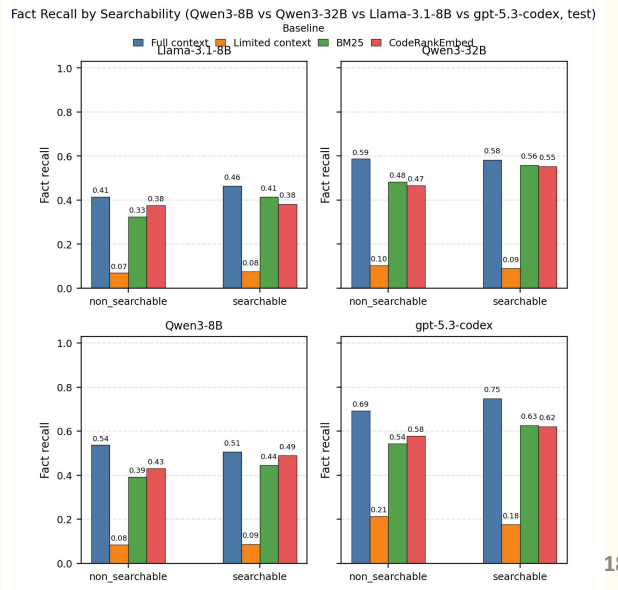
- Full context should be read as an upper bound rather than a competing baseline.
- Among the actual methods, retrieval consistently improves fact recall over Limited context.
- BM25 is the most reliable retrieval method overall, with CodeRankEmbed close behind and occasionally competitive on specific model/question slices.

Fact Recall on DeepCodeBench

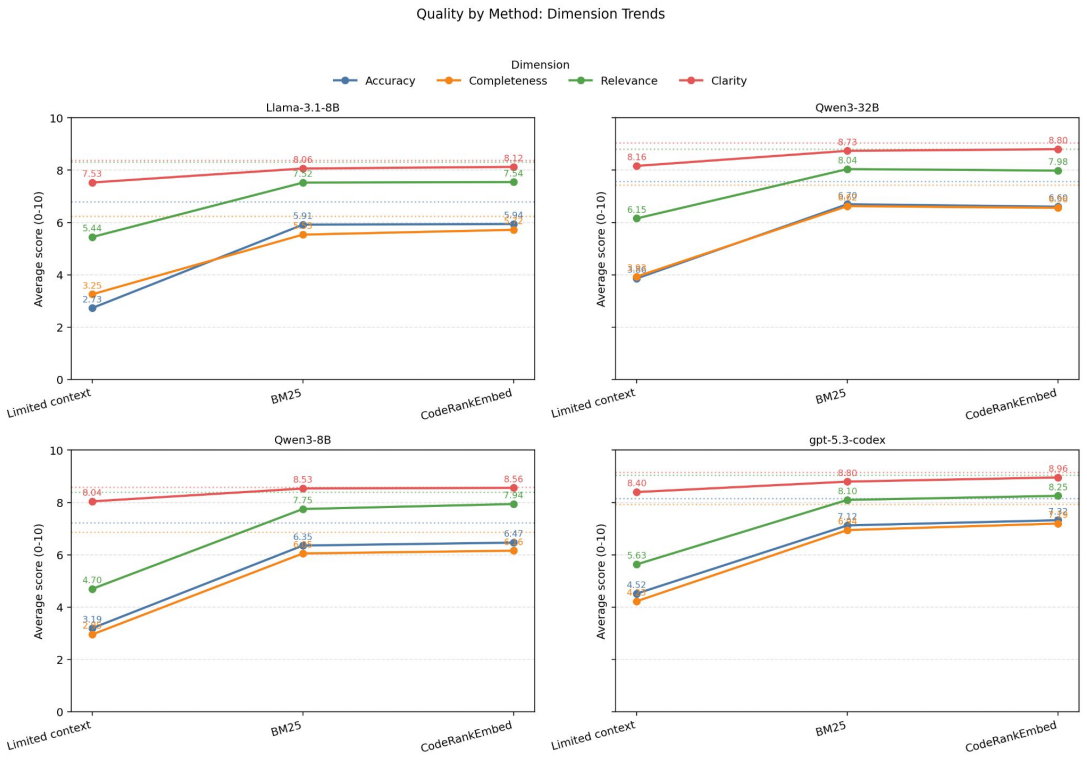
- The retrieval gains are largest on deep questions, where access to the right evidence matters most.
- Retrieval also helps more on searchable questions than on non-searchable ones, suggesting these methods are strongest when relevant facts can be localized.





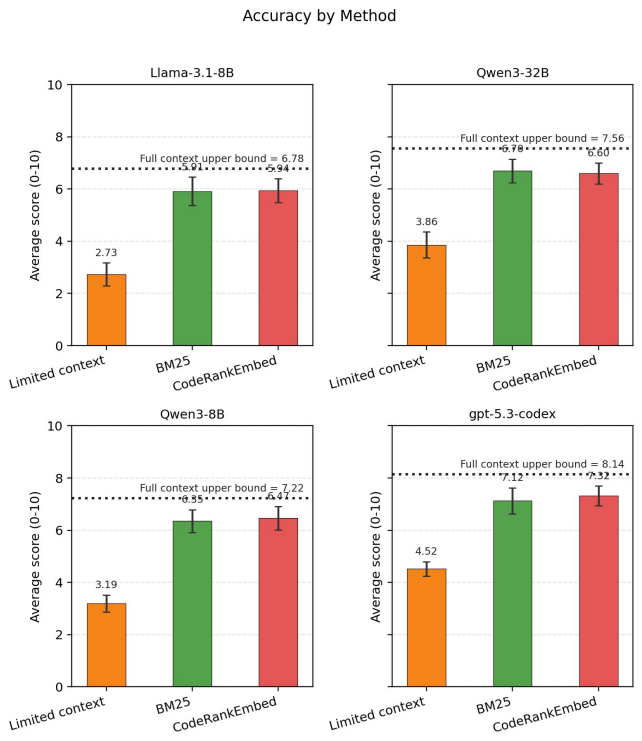


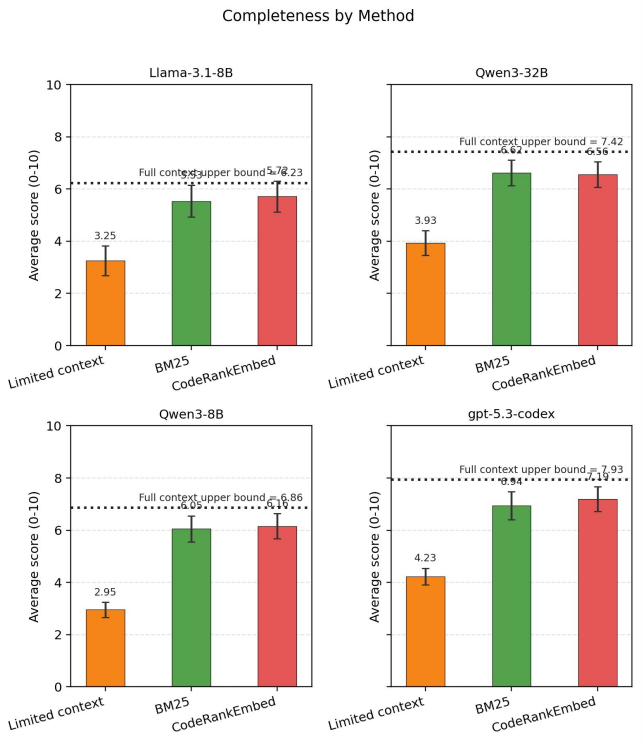
Quality Evaluation on DeepCodeBench



- The same method trend holds for answer quality.
- BM25 and CodeRankEmbed both move quality closer to the Full context upper bound, with relatively small gaps between them.
- Clarity is less sensitive to retrieval method than accuracy and completeness.

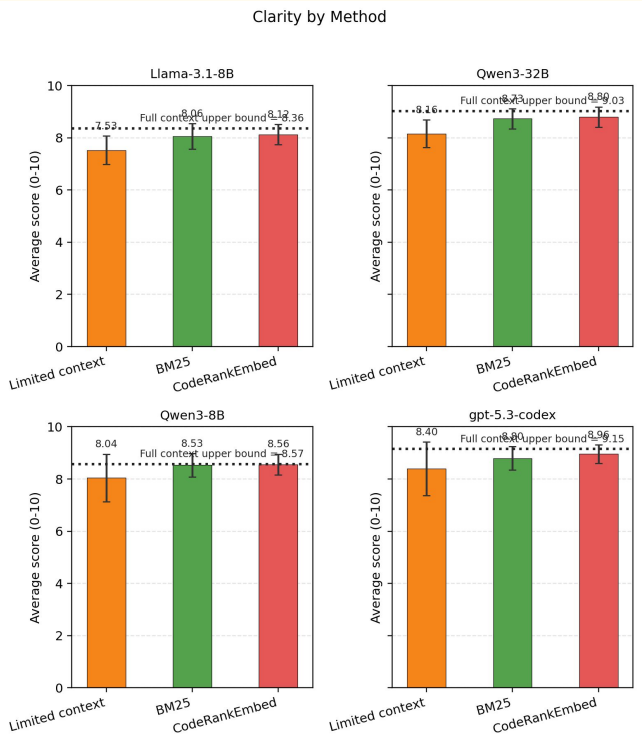
Quality Evaluation on DeepCodeBench

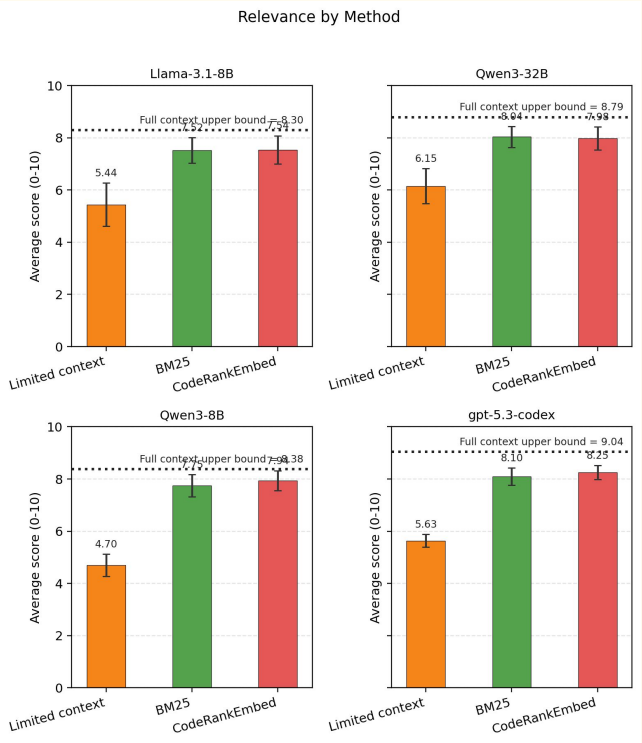




Judge variability is generally small, so the observed method-level quality differences appear stable across repeated evaluations.

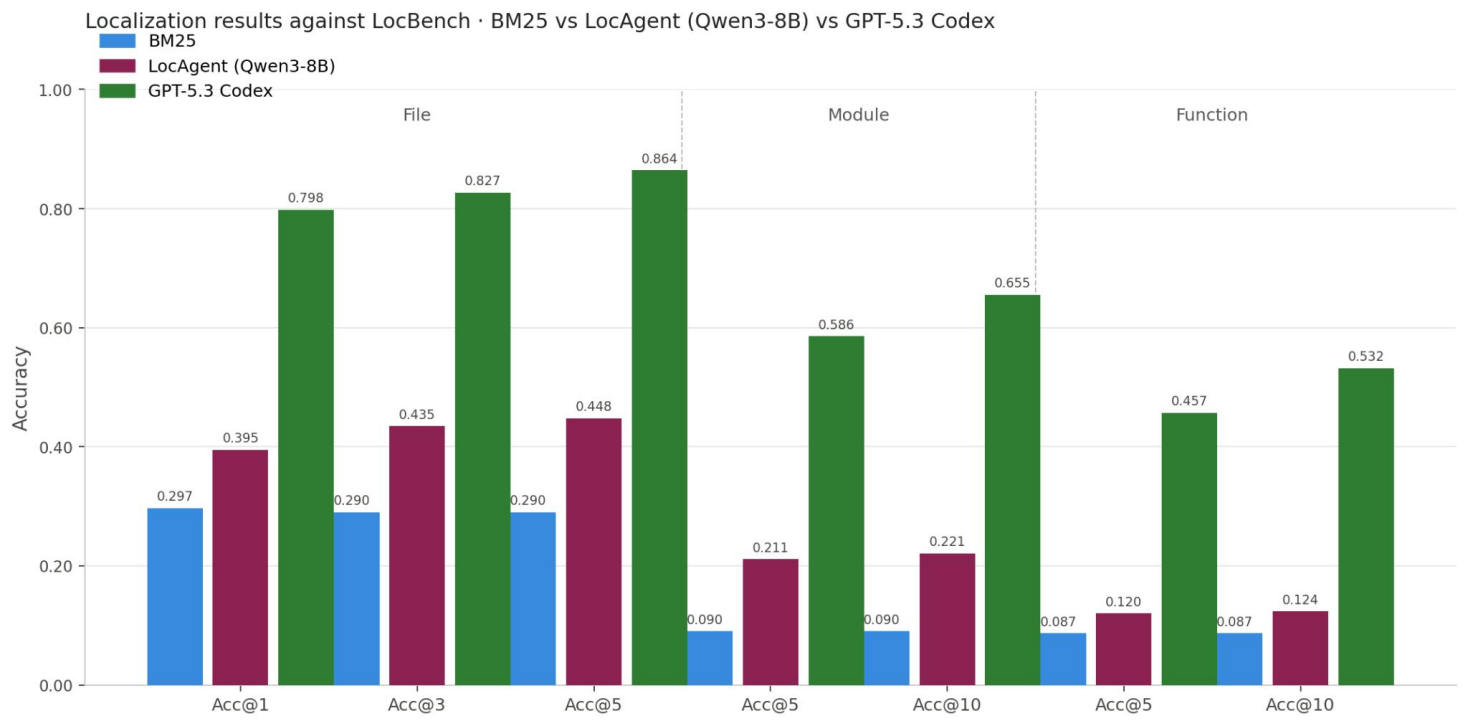
Quality Evaluation on DeepCodeBench



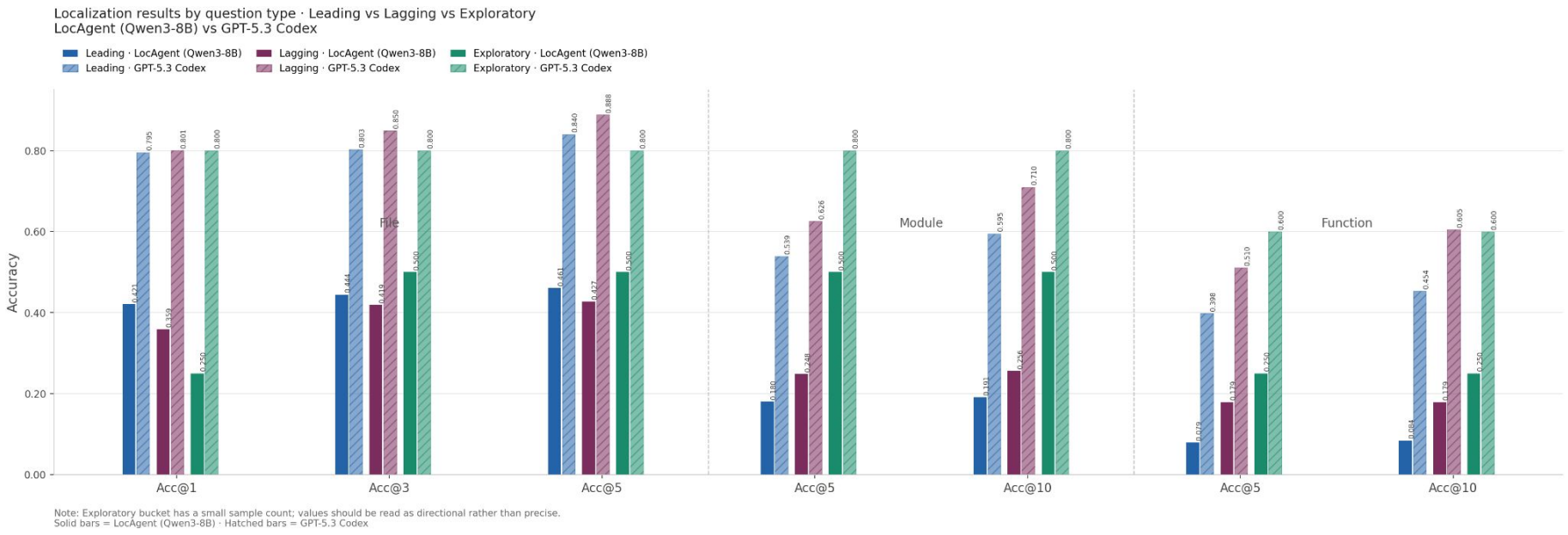


Judge variability is generally small, so the observed method-level quality differences appear stable across repeated evaluations.

Localization on LocBench



Localization on LocBench



Thank You