Ask KG Agent: A Multi-Agent Framework for Code Localization Using Code and Knowledge Graphs

Chanyoung Chung[1], Jeemin Kim[1], and Joyce Jiyoung Whang[2*]
[1] School of Computing, KAIST, [2] Department of AI Computing, KAIST
[*] Corresponding Author

The 40th Annual Conference on
Neural Information Processing Systems (NeurIPS 2026)

KAIST
BDILab
BIG DATA INTELLIGENCE

01 Code Localization

- The task of **identifying code segments relevant to software issues**
- Applicable to various software engineering tasks, including bug fixes and feature additions



Add sparse matrix output to SplineTransformer. As B-splines naturally have a sparse structure, I'd like to have the option that `SplineTransformer` returns a sparse matrix instead of always an ndarray.

```
class SplineTransformer(TransformerMixin, BaseEstimator):
  def __init__(
    order="C",
    sparse_output=False
  ):
    self.order = order
    self.sparse_output = sparse_output

  def fit(self, X, y=None, sample_weight=None):
    elif not np.all(np.diff(base_knots, axis=0) > 0):
      raise ValueError
    if self.sparse_output:
      raise ValueError
    n_knots = base_knots.shape[0]
```

```
def transform(self, X):
  if X.dtype in FLOAT_DTYPES:
    dtype = X.dtype
  else:
    dtype = np.float64
  XBS = np.zeros((n_samples, n_out), dtype=dtype)
  if use_sparse:
    output_list = []
  else:
    XBS = np.zeros((n_samples, n_out), dtype=dtype)

  for i in range(n_features):
    spl = self.bsplines_[i]
```

01 Code Localization

- Code localization is a **primary bottleneck** in automated software engineering
  - Understanding code for identifying localizations to fix consumes up to 66% of developers' debugging time [1]
  - On SWE-Bench Lite, coding agents failed to identify target functions for approximately half of the issues [2]

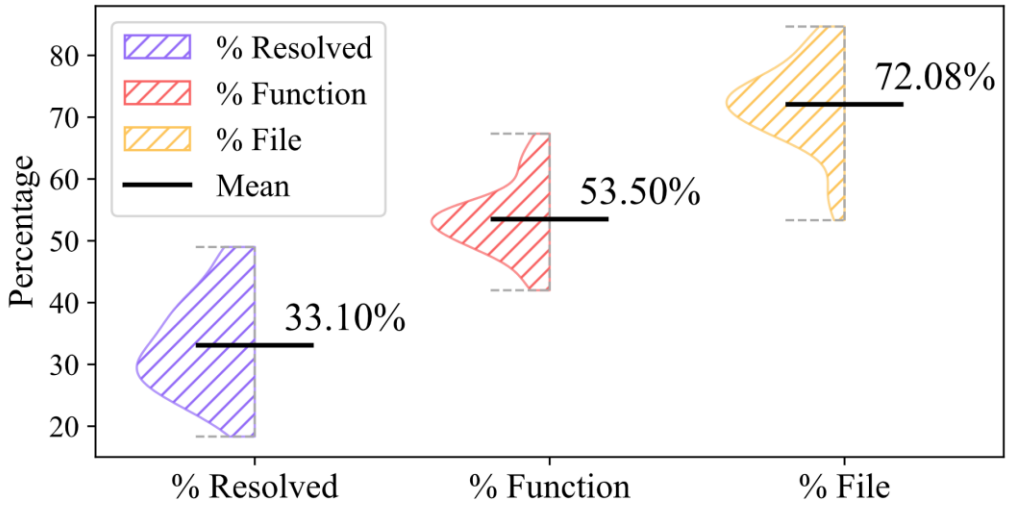
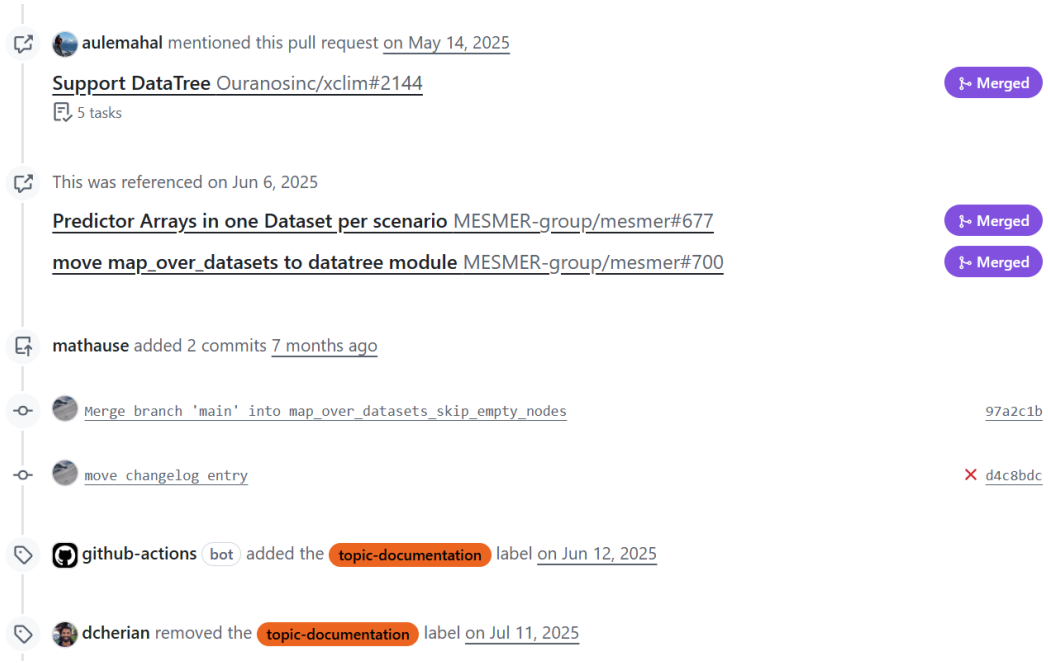


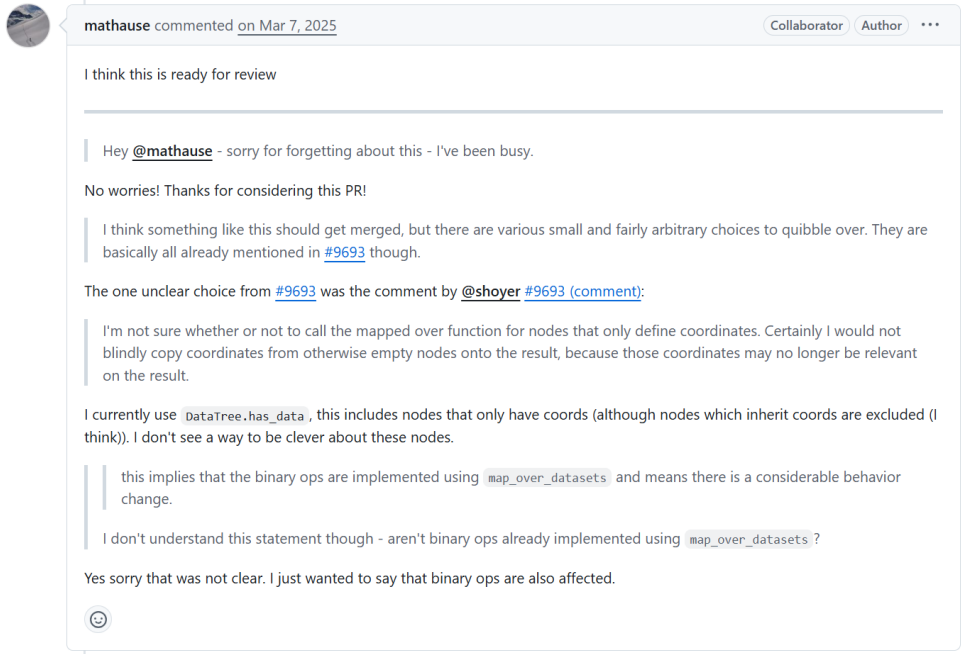
Figure 1. Distribution and average of file / function match rate and resolved rate on SWE-Bench Lite LeaderBoard.

[1] Chen et al., LocAgent: Graph-Guided LLM Agents for Code Localization, ACL 2025.
[2, Image] Yu et al., OrcaLoca: An LLM Agent Framework for Software Issue Localization, ICML 2025.

01 Development Traces

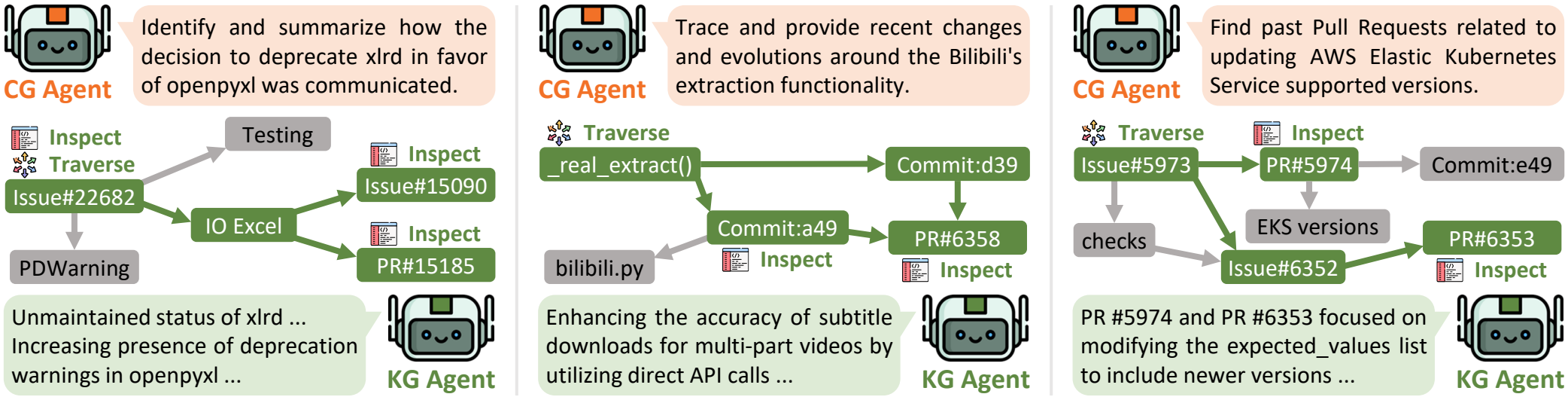
- Source code alone often fails to capture the **developer's intent and rationale behind code changes**
- **Development traces** generated during software maintenance, such as issues, pull requests, and commits, provide **valuable insights from past modifications and bug fixes**



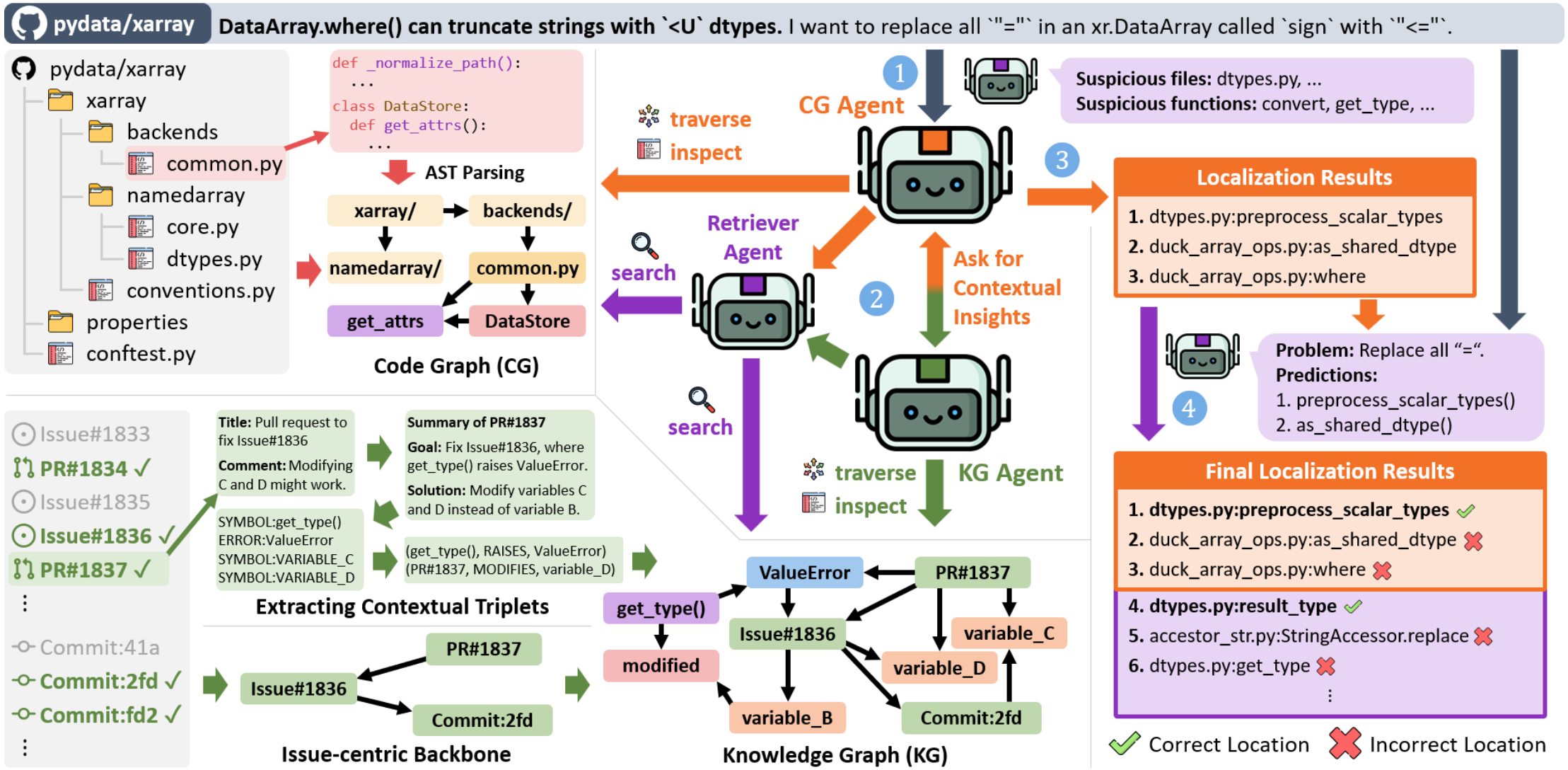


01 Utilizing Development Traces

- Several recent studies have attempted to integrate historical data in a naive manner
  - They are limited by treating development traces as **isolated snippets for retrieval or simple inspection**
  - They **obscure the broader design rationales across multiple maintenance logs**, resulting a shallow inspection
- Our solution: **Codebase Knowledge Graph**
  - Captures **interconnections between historical records**, such as issues, PRs, and commits
  - Augmented with LLM-extracted triplets to capture **design rationales and developer intents underlying records**



02 Overview of Co-Locator

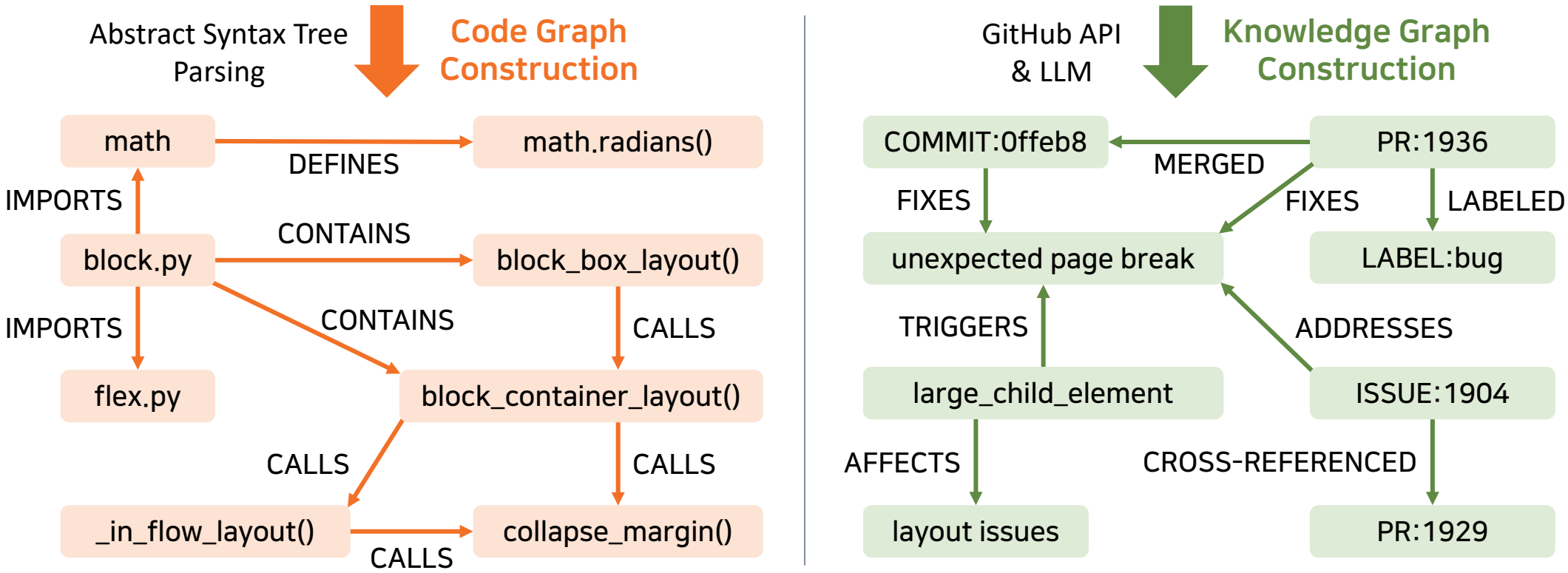


02 Obtain Code and Knowledge Graphs

- Construct a **code graph** and extract a **knowledge graph** specific to the given query

Kozea/WeasyPrint **Title: v63.0 flexbox issue overflow hidden**

With v63.0 the following code breaks: <code snippet>
When overflow: hidden is added to .cell, all cells appear on top of each other instead of next to each other.



02 Initial Retrieval

- Retriever agent generates **initial prediction by retrieving relevant locations** using issue descriptions
  - Retriever agent collects relevant locations in both file- and function-level
- The retrieved locations, along with the issue description, are **provided to the CG Agent**

Kozea/WeasyPrint **Title: v63.0 flexbox issue overflow hidden**

With v63.0 the following code breaks: <code snippet>
When overflow: hidden is added to .cell, all cells appear on top of each other instead of next to each other.



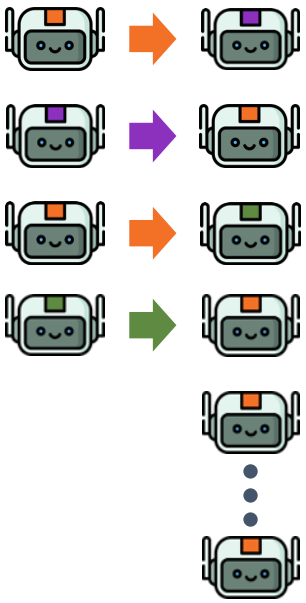
Retriever Agent

Top relevant files
- /weasyprint/formatting_structure/build.py
- /weasyprint/layout/percent.py

Top relevant functions
- /weasyprint/formatting_structure/build.py:set_viewport_overflow
- /weasyprint/layout/percent.py:resolve_percentages

02 Collaborative Code Localization

- Building upon the inputs, the agents collaborative narrow down the target locations
  - The CG (Code Graph) agent identifies potential fault locations by tracing logical flows within the code graph
  - The KG (Knowledge Graph) agent provides historical and intent-based context to the CG agent
  - The Retriever agent assists the CG and KG agents by retrieving relevant entities from their respective graphs



Search functions that are responsible for adjusting the layout of elements.

In the current codebase, `flex_layout` function adjusts the layout of elements.

Summarize how the `flex_layout` function was previously modified in relation to overflow handling.

There isn't a direct modification for `flex_layout` function itself regarding overflow, but several issues highlight pitfalls associated with flex layouts and overflow. Changing the child's display to `block` mitigates unexpected overflow.

Let's now inspect the `block_level_layout` function in `/weasyprint/layout/block.py` to see if there are any changes or considerations related to `overflow` that might propagate to flex layouts.

Final Predictions:
- weasyprint/layout/flex.py:flex_layout
- weasyprint/layout/block.py:block_box_layout

02 Post Retrieval

- After collaborative code localization, the **CG agent predicts a set of locations to be modified**
- Retriever agent uses the issue description and predicted locations to **retrieve additional locations**
  - Post-retrieval ensures a comprehensive localization result

Kozea/WeasyPrint **Title: v63.0 flexbox issue overflow hidden**

With v63.0 the following code breaks: <code snippet>
When overflow: hidden is added to .cell, all cells appear on top of each other instead of next to each other.

Functions to be modified: weasyprint/layout/flex.py:flex_layout, weasyprint/layout/block.py:block_box_layout



Additional functions to be modified:
weasyprint/stacking.py:_dispatch, weasyprint/layout/block.py:block_container_layout, ...

03 Experimental Settings

- Datasets
  - SWE-Bench-Live Lite (NeurIPS 2025) and LocBench (ACL 2025)

- Baselines
  - **Traditional Retrieval-based Approaches**
    - BM25, E5-base-v2, CodeRankEmbed, Jina-Code-v2, GTE, Codesage-large-v2, SFR-Embedding-2, SWERank
  - **Procedural Approaches**
    - Agentless
  - **Agent-based Approaches**
    - OrcaLoca, LocAgent, CoSIL

- Backbone LLMs
  - **GPT-4o-2024-08-06** (GPT-4o), **Qwen2.5-Coder-32B-Instruct** (Qwen2.5)

03 Metrics

- Code localization performance is measured at **three granularities**.
  - **File-level** Accuracy, **Module-level** Accuracy, **Function-level** Accuracy

- Evaluation Metric: **Accuracy@K**
  - Localization is considered successful only if **all ground-truth locations are contained within top K predictions**.

- For file-level localization, we use **Accuracy@5** and **Accuracy@10**
  - Ground-truth file counts range from 1 to 5.
- For module- and function-level localization, we use **Accuracy@10** and **Accuracy@15**
  - Ground-truth function counts from 1 to 10.

03 Code Localization Results

- Evaluate two variants of Co-Locator: Co-Locator⁻ and Co-Locator⁺
  - Co-Locator⁺ adopts the **reranking strategy from SWERank** whereas Co-Locator⁻ excludes reranking

		SWE-Bench-Live						LocBench					
		File		Module		Function		File		Module		Function	
		5	10	10	15	10	15	5	10	10	15	10	15
BM25		0.553	0.656	0.467	0.508	0.295	0.332	0.538	0.645	0.452	0.516	0.302	0.336
E5-base-v2		0.680	0.754	0.603	0.652	0.406	0.443	0.716	0.796	0.607	0.652	0.432	0.468
CodeRankEmbed		0.586	0.684	0.533	0.594	0.357	0.393	0.616	0.698	0.529	0.568	0.320	0.370
Jina-Code-v2		0.664	0.758	0.631	0.684	0.443	0.480	0.705	0.775	0.609	0.655	0.430	0.489
GTE		0.557	0.697	0.484	0.541	0.271	0.307	0.596	0.696	0.500	0.545	0.305	0.348
Codesage-large-v2		0.656	0.771	0.607	0.660	0.398	0.447	0.671	0.734	0.579	0.623	0.402	0.430
SFR-Embedding-2		0.742	0.812	0.697	0.750	0.541	0.574	0.771	0.855	0.696	0.743	0.509	0.564
SWERank⁻		0.754	0.824	0.709	0.746	0.541	0.574	0.752	0.823	0.675	0.711	0.507	0.557
SWERank⁺		*0.787*	0.840	*0.754*	*0.791*	*0.586*	*0.623*	*0.811*	0.855	*0.729*	0.759	*0.616*	*0.646*
GPT-4o	Agentless	0.611	0.705	0.496	0.496	0.324	0.328	0.780	0.838	0.632	0.636	0.434	0.439
	OrcaLoca	0.332	0.332	0.254	0.254	0.201	0.201	0.521	0.521	0.364	0.364	0.280	0.280
	LocAgent	0.557	0.557	0.381	0.381	0.258	0.258	0.702	0.707	0.518	0.518	0.404	0.405
	CoSIL	0.635	0.697	0.582	0.590	0.422	0.434	0.779	0.811	0.641	0.666	0.523	0.550
	Co-Locator⁻	0.775	0.832	0.746	0.787	0.553	0.607	0.802	0.848	0.714	0.746	0.589	0.632
	Co-Locator⁺	**0.811**	0.857	0.770	**0.811**	**0.623**	0.635	0.830	0.871	0.752	0.786	**0.657**	0.686
Qwen2.5	Agentless	0.607	0.680	0.422	0.422	0.254	0.254	0.748	0.816	0.521	0.530	0.357	0.363
	OrcaLoca	0.303	0.303	0.234	0.234	0.172	0.172	0.429	0.429	0.309	0.309	0.229	0.229
	LocAgent	0.480	0.480	0.299	0.299	0.217	0.217	0.613	0.613	0.445	0.452	0.346	0.350
	CoSIL	0.611	0.689	0.504	0.512	0.377	0.393	0.741	0.775	0.618	0.632	0.484	0.502
	Co-Locator⁻	0.770	*0.844*	0.750	0.799	0.561	0.607	0.808	*0.866*	0.728	*0.760*	0.591	0.629
	Co-Locator⁺	0.807	**0.865**	**0.783**	**0.811**	0.611	**0.643**	**0.848**	**0.880**	**0.757**	**0.787**	0.654	**0.689**

03 Benefits of Knowledge Graphs

- Co-Locator+ successfully identifies target locations while the variant without KG fails

Kozea/WeasyPrint **Title: v63.0 flexbox issue overflow hidden**

With v63.0 the following code breaks: <code snippet>
When overflow: hidden is added to .cell, all cells appear on top of each other instead of next to each other.

w/o KG

[CG] Let's begin by inspecting functions involved in Flexbox layout calculations and Overflow property.

[CG] Analyze further by looking at the **`overflow` property validation function** located at `properties.py`.

[CG] Since the problem lies in the interaction of `overflow: hidden` with Flexbox layouts, we need to investigate deeper into **how `overflow` property affects the layout in Flexbox.**

[CG] Let's use `traverse_cg` to trace the usage of the **`overflow` property within `flex_layout`**.

[CG] In the context of Flexbox, this means that when `overflow: hidden` is applied, `min_width` and `min_height` default to `0` for flex items with `'auto'` minimum dimensions. This could cause the flex items to overlap because their minimum size constraints are effectively removed.

-------- Final Predictions --------

- weasyprint/layout/flex.py:flex_layout ✗
- weasyprint/layout/percent.py:resolve_one_percentage ✗

w/ KG

[CG -> KG] Summarize how the `flex_layout` function was previously modified in relation to overflow handling.

[KG -> CG] There isn't a direct modification for `flex_layout` function itself regarding overflow, but **several issues** highlight pitfalls associated with flex layouts and overflow. **Changing the child's display to `block` mitigates unexpected overflow.**

[CG] Let's now inspect the `block_level_layout` function in **`/weasyprint/layout/block.py`** to see if there are any changes or considerations related to `overflow` that might propagate to flex layouts.

[CG] Next, let's inspect **`block_box_layout` from `block.py`** to see if it contains any logic related to `overflow`.

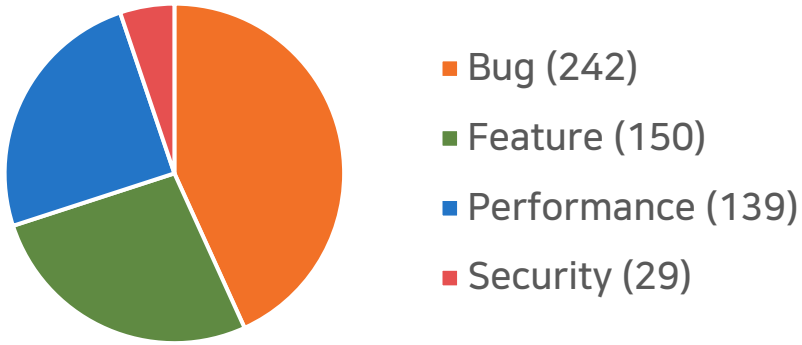
[CG] **`block_box_layout` orchestrates the layout process for block-level boxes.**

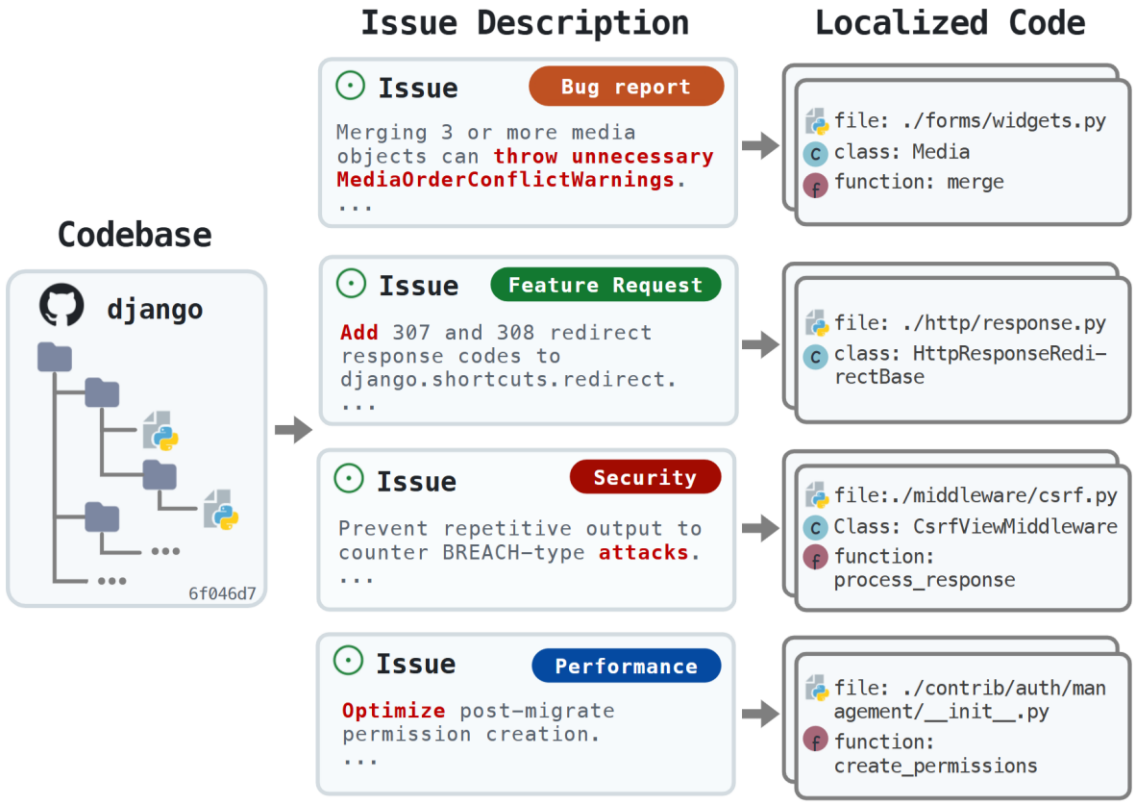
-------- Final Predictions --------

- weasyprint/layout/flex.py:flex_layout ✗
- weasyprint/layout/block.py:block_box_layout ✓

03 Per-Category Analysis

- Queries in LocBench are classified into four categories:
  - **Bug Report**
    - Issues requesting fixes for specific bugs
  - **Feature Request**
    - Issues requesting the addition of new features
  - **Performance Issue**
    - Issues requesting performance improvements
  - **Security Vulnerability**
    - Issues reporting security problems

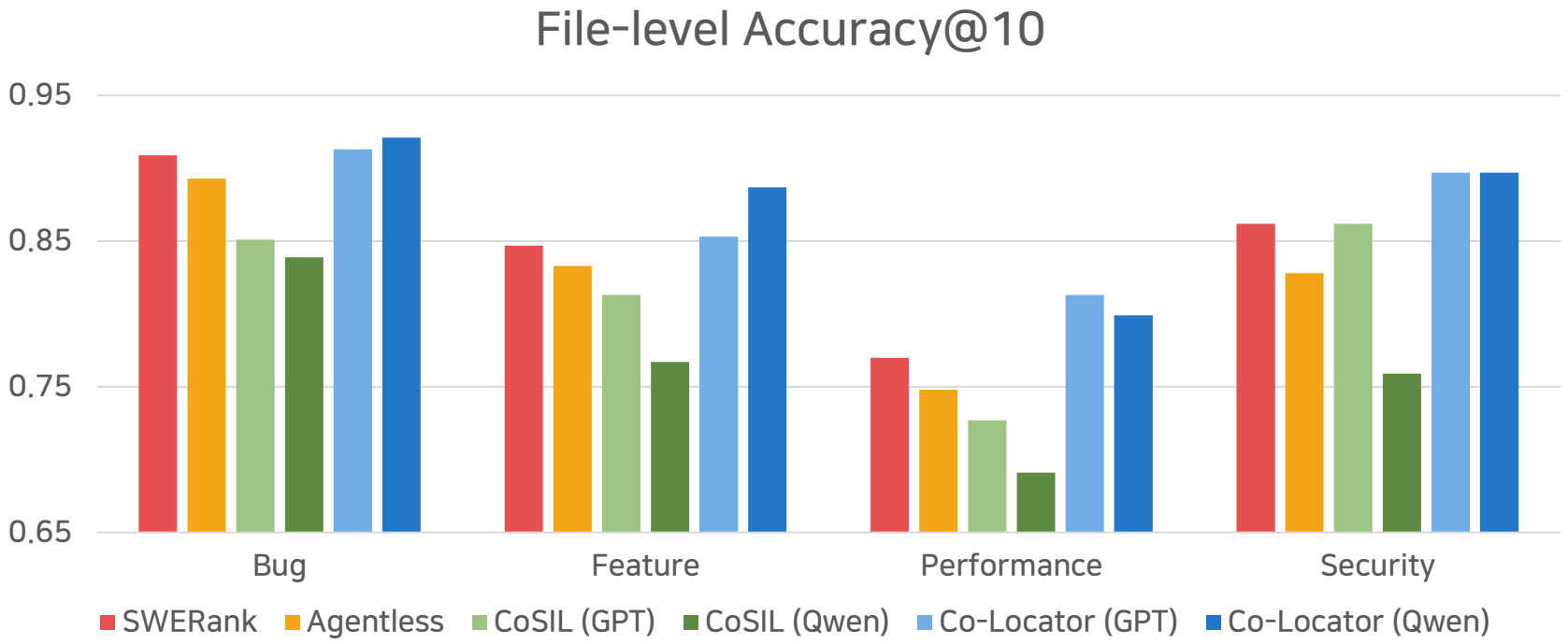




Image(right): Chen et al., LocAgent: Graph-Guided LLM Agents for Code Localization, ACL 2025.

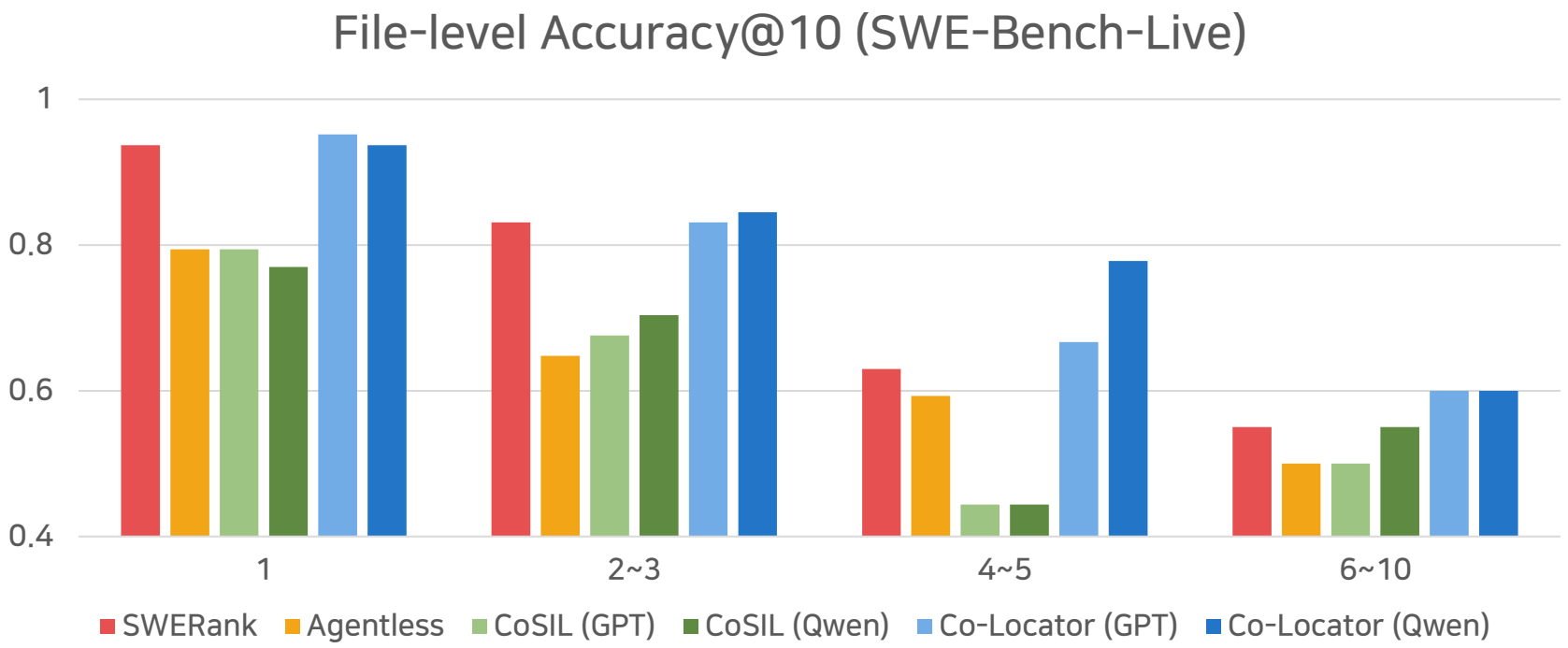
03 Per-Category Analysis

- Compare Co-Locator with the best-performing baseline from each category
  - **SWERank** from traditional retrieval-based approaches
  - **Agentless** from procedural approaches
  - **CoSIL** from agent-based approaches
- Co-Locator **consistently outperforms all baselines across all issue categories**



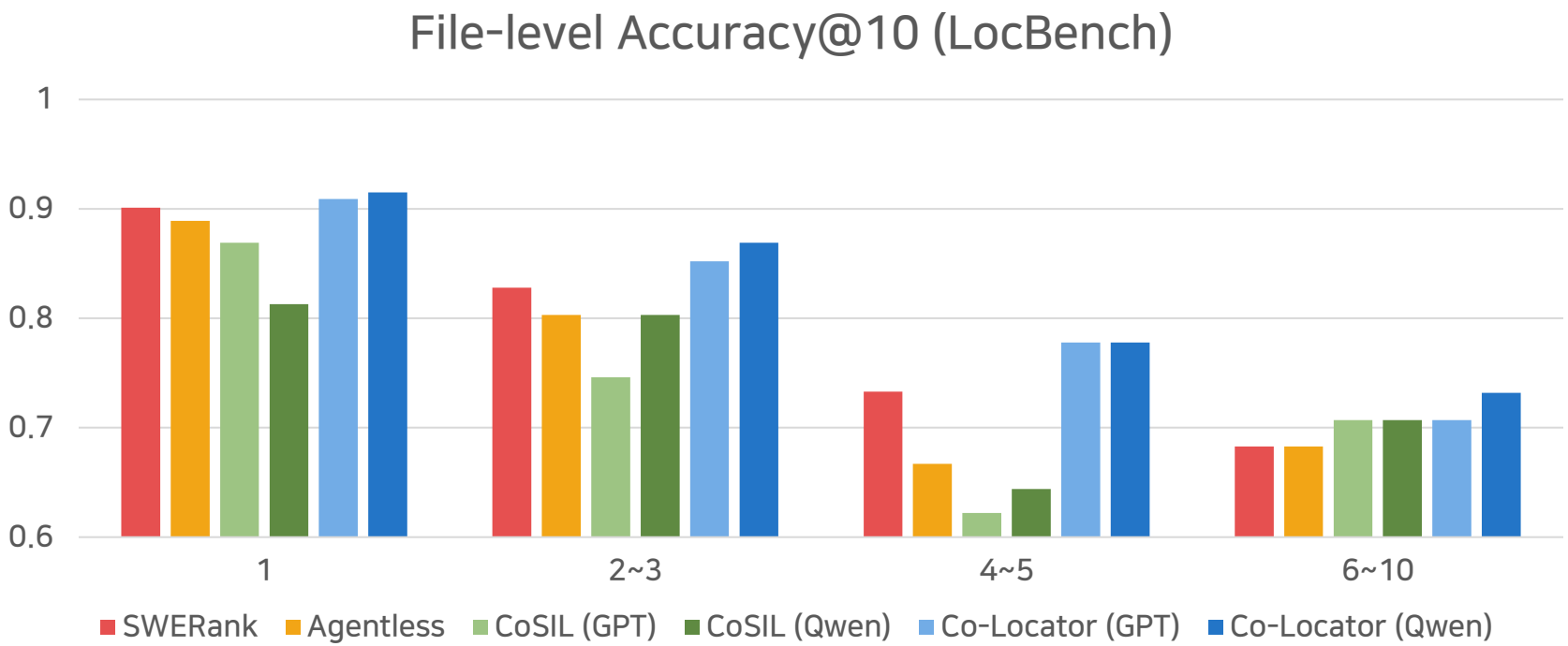
03 Performance by Issue Complexity

- Evaluate performance across issue complexity levels based on the number of ground-truth functions
  - 1 function / 2-3 functions / 4-5 functions / 6-10 functions
- Co-Locator **consistently outperforms all baselines across all complexity levels**



03 Performance by Issue Complexity

- Evaluate performance across issue complexity levels based on the number of ground-truth functions
  - 1 function / 2-3 functions / 4-5 functions / 6-10 functions
- Co-Locator **consistently outperforms all baselines across all complexity levels**



03 Impacts on Code Generation

- Analyze the impact of code localization accuracy on **code generation performance** using Agentless
  - Generate patches with Agentless **using the code locations predicted by each code localization method**
  - Include an **Oracle setting** that provides ground-truth locations to estimate the upper bound of performance
- Co-Locator achieves the **highest code generation performance** among the compared methods
  - Locations identified by Co-Locator⁺ are **nearly as effective as using ground-truth locations**

	Resolve Rate (%)
SWERank⁺	11.1
Agentless	10.6
CoSIL	9.4
Co-Locator⁺	13.1
Oracle	14.8

03 Cost Analysis of Agent-based Approaches

- Conduct a cost analysis comparing Co-Locator with agent-based approaches
  - **Cost ($)**: Average cost per instance incurred during the agentic reasoning process in United States dollar
  - **Runtime (minute)**: The average runtime per instance
  - File-level Acc@10, Module-level Acc@15, and Function-level Acc@15 are also included
- Co-Locator achieves **high cost-effectiveness among agent-based approaches**
  - Co-Locator⁻ achieves the **lowest cost and highest accuracy among agent-based methods without reranking**

	SWE-Bench-Live				LocBench				Runtime (minute)
	Cost ($)	File	Module	Function	Cost ($)	File	Module	Function	
OrcaLoca	0.216	0.332	0.254	0.201	0.246	0.521	0.364	0.280	2.665
LocAgent	0.224	0.557	0.381	0.258	0.263	0.707	0.518	0.405	0.172
CoSIL	0.217	0.697	0.590	0.434	0.264	0.811	0.666	0.550	0.559
Co-Locator⁻	**0.213**	0.832	0.787	0.607	**0.170**	0.848	0.746	0.632	**0.055**
Co-Locator⁺	0.374	**0.857**	**0.811**	**0.635**	0.402	**0.871**	**0.786**	**0.686**	0.106

04 Conclusion

- Construct a **codebase knowledge graph** that captures **design rationales and developer intents under development traces**

- Propose **Co-Locator**, a collaborative multi-agent framework for code localization using **code and knowledge graphs**
  - Consists of three specialized agents: **CG agent**, **KG agent**, and **Retriever agent**
  - Enables **collaborative reasoning over functional dependencies and historical context** to identify locations to fix

- Experimental results demonstrate the effectiveness of Co-Locator in code localization
  - Shows the **benefits of leveraging knowledge graphs** for code localization
  - Consistently achieves **superior performance across all issue categories and complexity levels**
  - **Improves downstream code generation performance** while achieving **high cost-effectiveness**

Thank You!

Our datasets and codes are available at:
https://github.com/bdi-lab/Co-Locator

You can find us at:
{chanyoung.chung, kjm10122, jjwhang}@kaist.ac.kr
https://bdi-lab.kaist.ac.kr

KAIST BDILab
BIG DATA INTELLIGENCE