

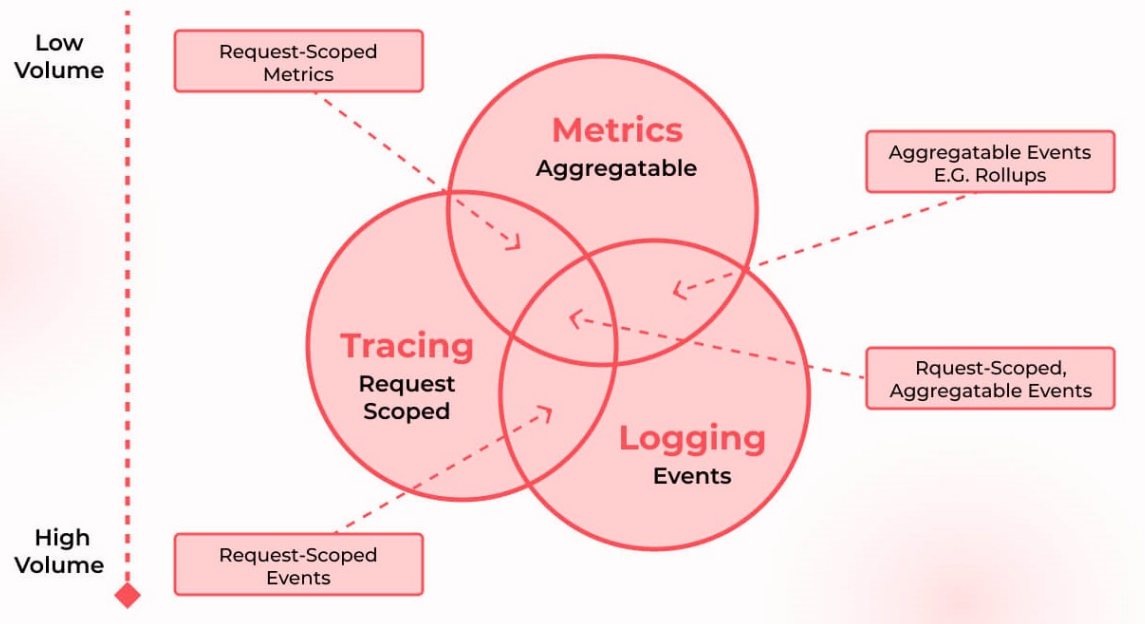
Automated Proactive Logging Quality Improvement for Large-Scale Codebases

Yichen Li, Jinyang Liu, Junsong Pu, Zhihan Jiang, Zhuangbin Chen, Xiao He, Tieying Zhang, Jianjun Chen, Yi Li, Rui Shi, Michael R. Lyu



The Cornerstone of System Reliability: Observability

- The Three Pillars of System Observability: Modern systems rely on Metrics, Traces, and Logs
 - Metrics tell you that a problem happened.
 - Traces tell you where it happened.
 - But Logs are often the only around truth to tell you why it happened.

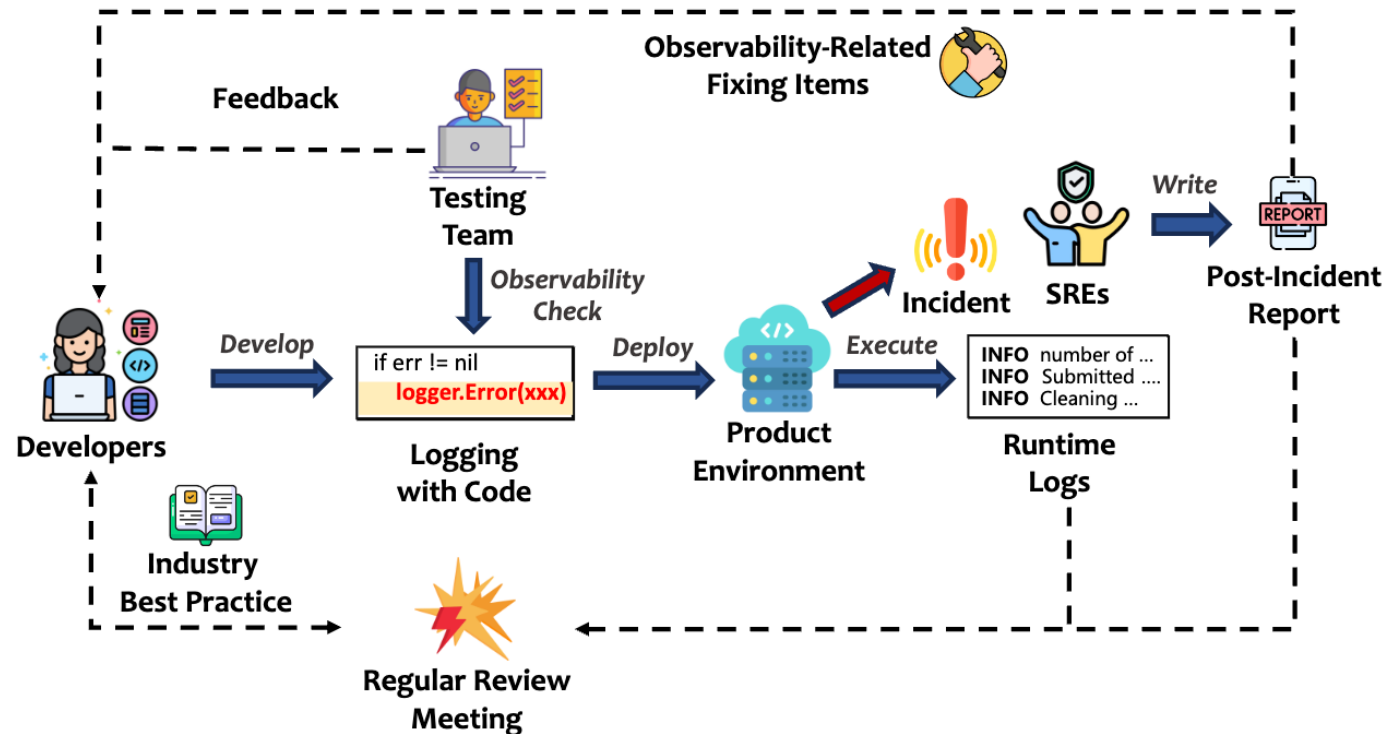


*“The most effective debugging tool is still careful thought, coupled with judiciously placed **print statements**.”*

-- Brian Kernighan

The Problem: Logging Debt

- However, the traditional process for improving it is broken:
 - ✓ **Manual & Meeting-Driven:** Relies on post-incident reviews and SRE meetings.
 - ✓ **Reactive:** Fixes problems after an incident occurs.
 - ✓ **Unscalable:** Cannot keep pace with the rapid evolution of large-scale codebases.



The Gap: Why Can't Existing Tools Help?

- **Claim:** Tools are Reactive.
- **Fact 1:** Existing tools require a developer to first select a method to improve.
 - They cannot answer the critical whether-to-log question at scale.
- **Fact 2:** Logging is SPARSE in real-world codebases.

↙
The hardest part

➤ **Conclusion:** A tool that just inserts logs everywhere is not practical.

Systems	LOC	# Logging Statements	# Methods	# Methods with Logging Statements	Logging Change Action Rate %		
					Insertion	Modification	Variable Enhancement
System-A	2.6M	6,163	31,486	3,180	41.5	6.9	51.6
System-B	43k	386	1,262	149	38.7	3.4	57.0
System-C	314k	830	2,984	387	33.0	3.5	63.5
System-D	845k	5,824	10,509	2,738	49.0	9.1	41.9
Total	3.8M	13203	46241	6454	44.1	7.5	48.4

The Gap: Why Can't Existing Tools Help?

- **Claim:** Existing tools are narrow.
- **Fact 1:** Existing tools focus only on insertion.
- **Fact 2:** Logging insertion is only 44.1% of real-world work.

➤ **Conclusion:** We need to cope with logging

We need a Proactive and Holistic logging tool.

```
...
+ logs.Logger.WithFields(logrus.Fields{"object_type": fmt.Sprintf("%T",
obj))).Errorf("failed to get object GVK: %v", err)
...
```

(a) Example of logging statement insertion.

```
if err != nil {
- logger.Errorf(err)
+ logs.Logger.WithFields(logrus.Fields{"resource": resource.Name,
+ "namespace": namespace(obj))).Errorf(err)...
```

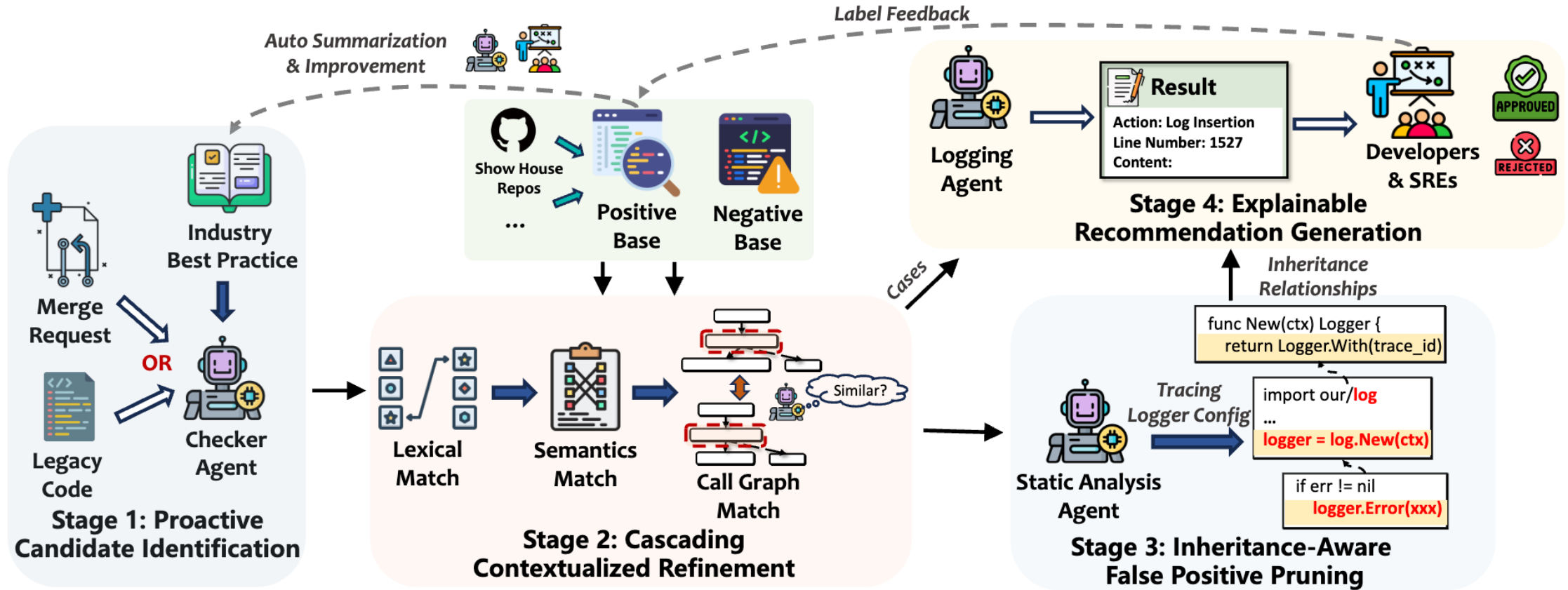
(b) Example of logging statement modification.

```
failed to get modified configuration for %s:
-- → Thrown for logging
```

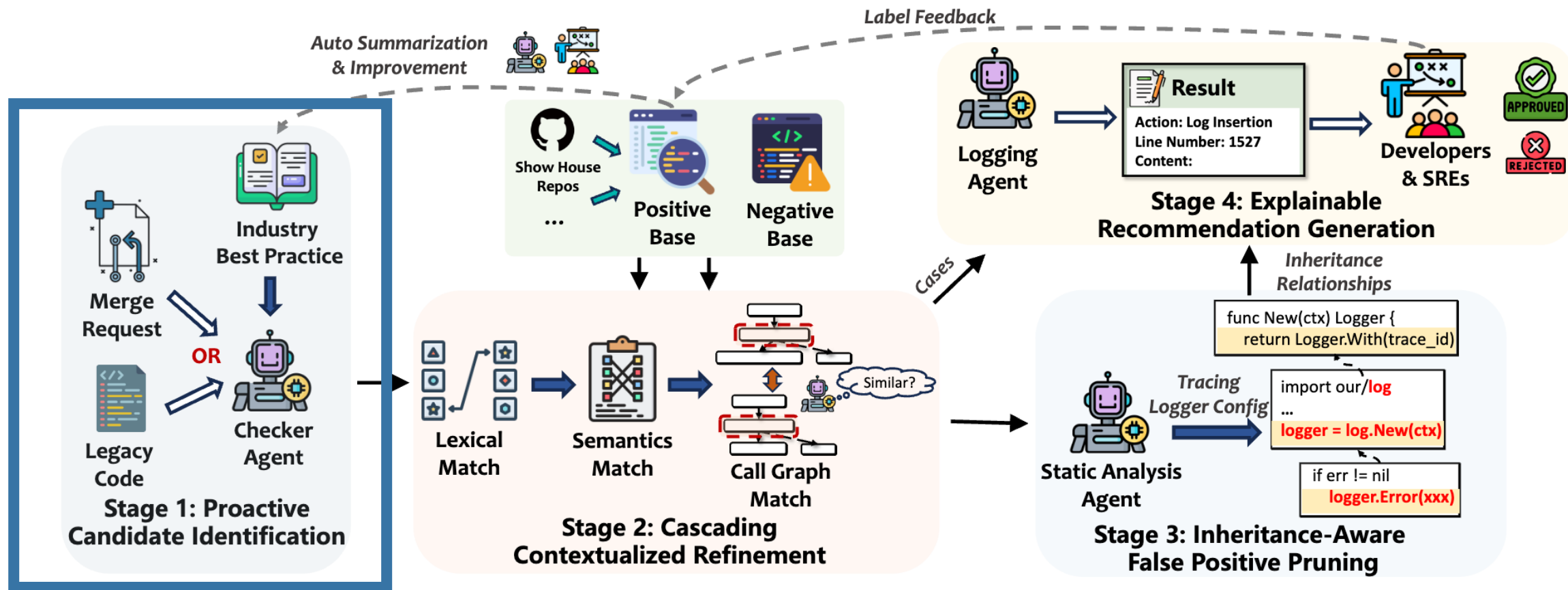
(c) Example of logging variable enhancement.

Systems	LOC	# Logging Statements	# Methods	# Methods with Logging Statements	Logging Change Action Rate %		
					Insertion	Modification	Variable Enhancement
System-A	2.6M	6,163	31,486	3,180	41.5	6.9	51.6
System-B	43k	386	1,262	149	38.7	3.4	57.0
System-C	314k	830	2,984	387	33.0	3.5	63.5
System-D	845k	5,824	10,509	2,738	49.0	9.1	41.9
Total	3.8M	13203	46241	6454	44.1	7.5	48.4

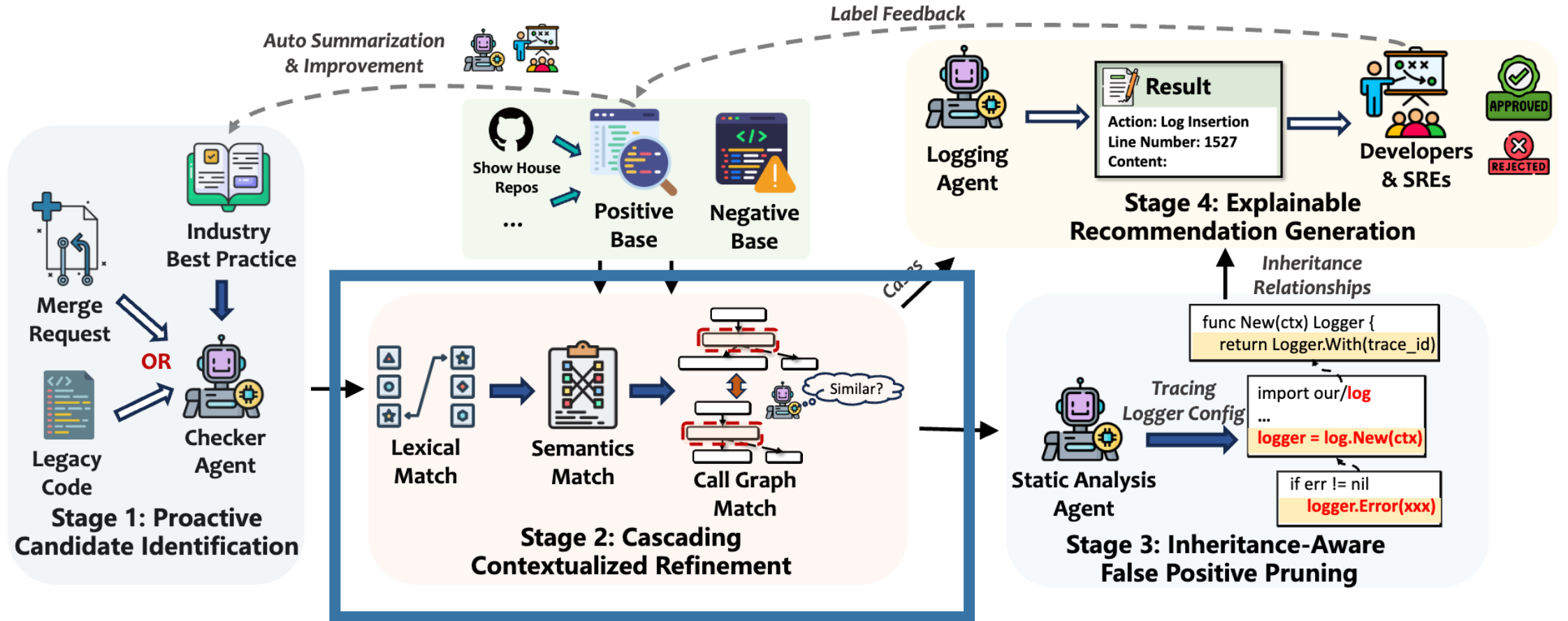
Our Proposal: LogImprover



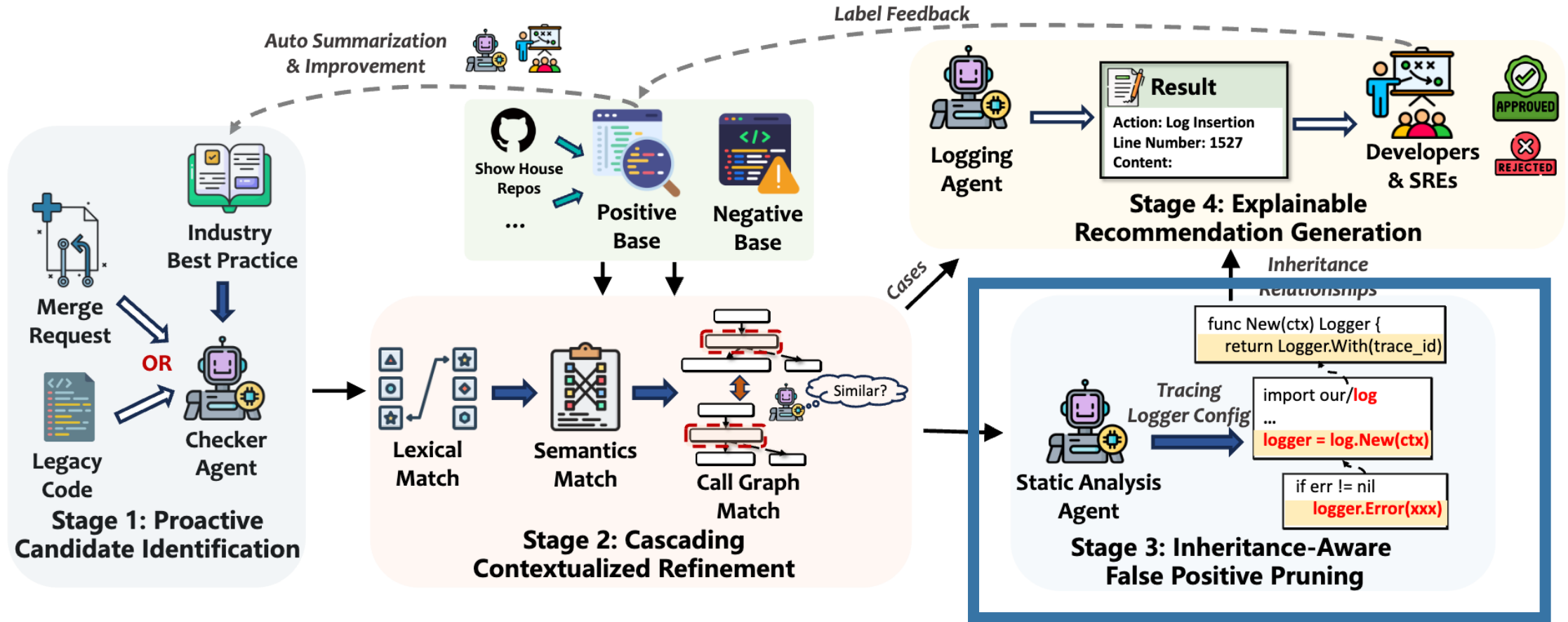
Our Proposal: LogImprover



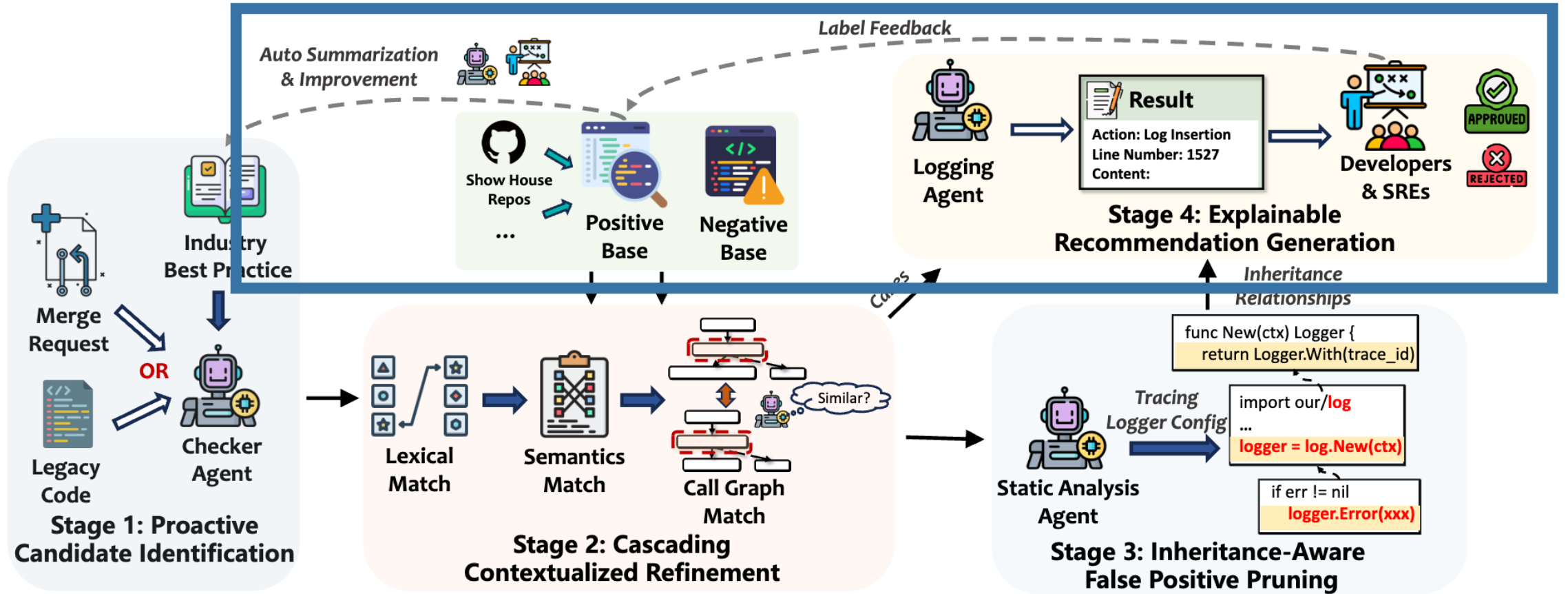
Our Proposal: LogImprover



Our Proposal: LogImprover



Our Proposal: LogImprover

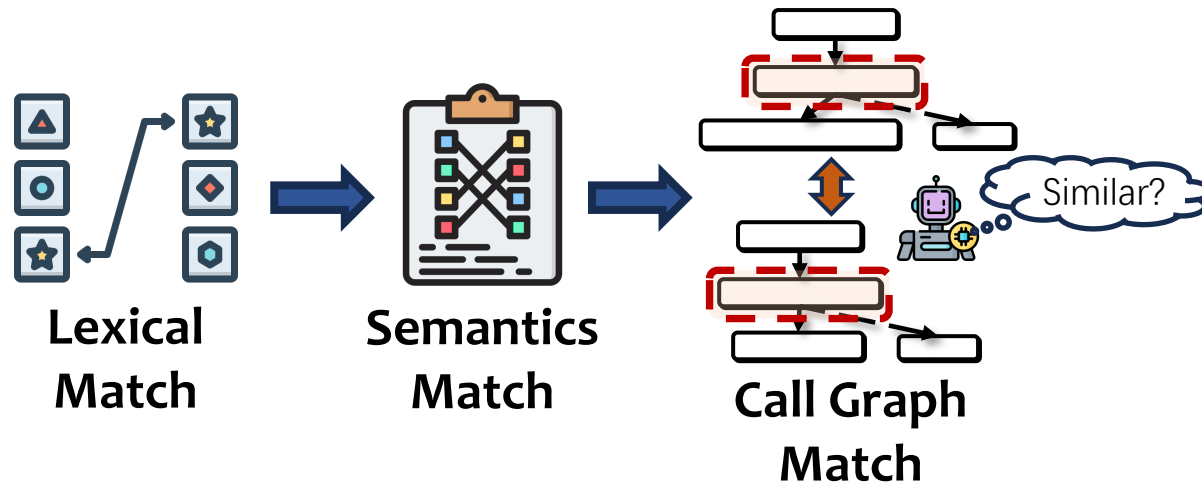


Main Contribution: Two Paradigm Shifts

- Shift 1: From Reactive Generation → Proactive Discovery
 - **Old Way:** A developer picks a method, and invoke the logging tool.
 - **Our Way:** The tool proactively scans the entire codebase to discover whether and where to improve.
- Shift 2: From Simple Insertion → Holistic Patch Generation.
 - **Old Way:** Tools focus only on inserting new logging statements.
 - **Our Way:** The tool generates holistic patches that mirror real developer work: Insertion, Modification, and Variable Enhancement.

Stage 2: Cascading Contextualized RAG

- **Problem:** Abstract rules are too general to follow.
- **Solution:** Ground suggestions in peer-validated evidence from post-incident fixes and show house repos.
- **Cascading RAG Pipeline:**
 1. Lexical Filtering (BM25): Fast, coarse-grained filter .
 2. Semantic Re-ranking (Dense): Matches functional *intent* .
 3. Dependency-Aware Re-ranking (LLM): Matches *structural* context (call graph isomorphism)



Stage 3: Inheritance-Aware False Positive Pruning

- **Logging Responsibility**

- A method may intentionally not log, instead enhancing an error (e.g., `fmt.Errorf`) for the caller to handle.
- We analyze the call stack to respect these boundaries.

- **Implicit Logger Inheritance**

- Modern frameworks automatically inject context (e.g., `trace_id`) via a base logger.
- Our Log Config Agent (Static Analysis + LLM) builds a profile of this implicit behavior to prevent redundant suggestions.

```
func databaseQuery() error {  
    dbErr := errors.New("record not found")  
    return fmt.Errorf("querying data%w", dbErr)  
}  
  
func processRequest() {  
    err := databaseQuery()  
    if err != nil {  
        logger.Error("request failed: %v", err)  
    }  
}
```

(a) An example of logging responsibility.

```
type BaseLogger struct {...}  
func (l *BaseLogger) Error(msg string, fields ...) {  
    fields["trace_id"] = getTraceIDFromCtx(ctx)  
}  
  
type ServiceLogger struct {  
    logging.BaseLogger  
    serviceName string  
}  
  
func handleRequest(ctx context.Context) {  
    logger := services.NewLogger("order")  
    if err := processOrder(); err != nil {  
        logger.Error(ctx, "xxx"...)  
    }  
}
```

(b) An example of logger inheritance.

Evaluation Setup: Close-World

- Dataset (with real-world logging patch as label)
 - ✓ 10 mature, large-scale cloud services at ByteDance (3.8M Go LoC)
- Baselines.
 - ✓ SCLogger , LANCE
 - ✓ Raw LLM, Raw LLM + RAG
- Metrics (Each stage is conditioned on the success of the previous one.)
 - ✓ Decision (D-F1): Is the whether-to-log decision correct?
 - ✓ Position (P-ACC): Is the location correct?
 - ✓ Content (L-ACC, V-F1, etc.): Is the log content (level, variables) correct?
 - ✓ Overall Quality (Judge Score): A holistic score for valid suggestions.

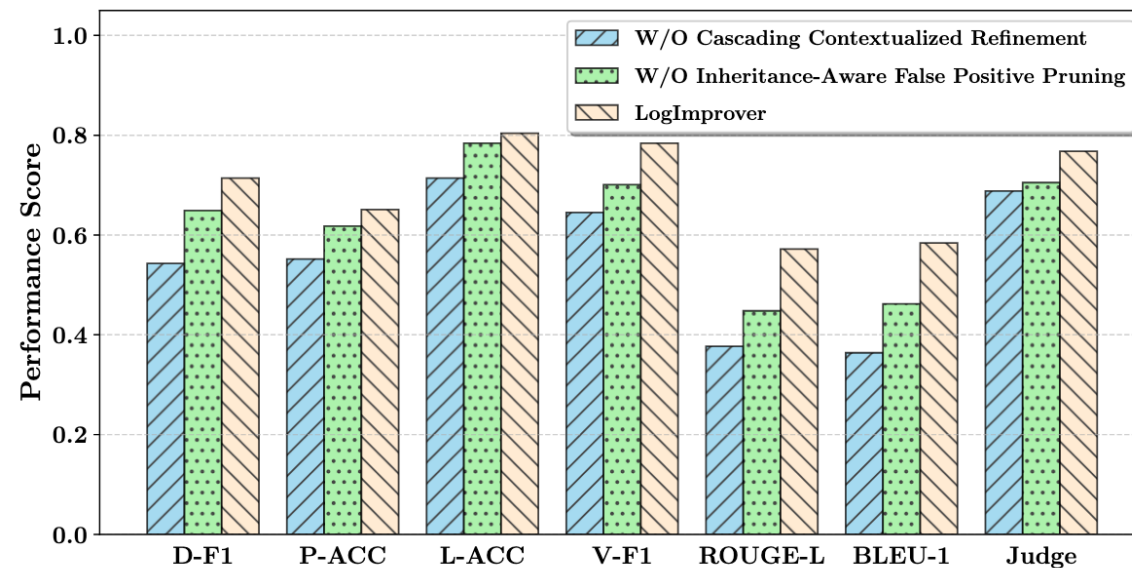
RQ1: How effective is LOGIMPROVER?

- Logging Statement Insertion
 - We are 28.2% better at the critical whether-to-log Decision (D-F1).
- Logging Statement Modification
 - LogImprover again leads in deciding which logs to modify and generating superior content.
- Logging Variable Enhancement
 - LogImprover outperform baselines in identifying where to enhance error variables and generate the most valuable enhancements

Task	Model	Decision (D-F1)	Position (P-ACC)	Levels (L-ACC)	Variables (V-F1)	Texts		Judge
						ROUGE-L	BLEU-1	
Logging Statement Insertion	LANCE	0.192	0.285	0.549	0.357	0.244	0.183	0.196
	Raw LLM	0.485	0.521	0.682	0.632	0.335	0.305	0.657
	Raw LLM + RAG	0.548	0.579	0.739	0.702	0.418	0.426	0.694
	SCLogger	0.557	0.623	0.795	0.725	0.585	0.553	0.725
	LOGIMPROVER	0.714	0.651	0.804	0.784	0.572	0.584	0.768
Logging Statement Modification	Raw LLM	0.594	—	0.732	0.682	0.684	0.671	0.708
	Raw LLM + RAG	0.658	—	0.810	0.723	0.706	0.675	0.735
	LOGIMPROVER	0.766	—	0.876	0.817	0.687	0.664	0.792
Logging Variable Enhancement	Raw LLM	0.564	—	—	—	0.515	0.481	0.657
	Raw LLM + RAG	0.591	—	—	—	0.547	0.498	0.691
	LOGIMPROVER	0.648	—	—	—	0.551	0.532	0.738

RQ2: How do the different stages contribute?

- W/O Stage 1 (Proactive Scan)
 - Token cost per method increased by 4x.
 - Role: Essential for efficiency and scalability.
- W/O Stage 2 (Contextualized Refinement)
 - Severe performance drop across all metrics.
 - Role: Grounding in concrete examples (RAG) is the core driver of quality.
- W/O Stage 3 (Inheritance-Aware Pruning)
 - Significant quality drop; introduces critical noise.
 - Role: Provides structural pruning and precision



Real-World Deployment & Case Study

- Deployed at ByteDance, scanning legacy code and new logging patches.
- Key Result.
 - ✓ 68.12% Overall Labeled Developer Acceptance Rate
- Validating the Holistic" Approach
 - ✓ 46.4% Variable Enhancement
 - ✓ 38.6% Insertion
 - ✓ 15.0% Modification

```
    }  
    return  
}  
  
func rollbackWorkload(c kubernetes.Interface, obj runtime.Object, revision string) error {  
    gvk := obj.GetObjectKind().GroupVersionKind()  
    r, err := resource.RollbackerFor(gvk.GroupKind(), c)   
    if err != nil {  
-       return err  
+       return fmt.Errorf("get rollbacker for %s: %w", gvk, err)  
    }  
    result, err := r.Rollback(obj, nil, revision, c)   
    if err != nil {  
+       logs.Logger.WithFields(logrus.Fields{"revision": revision}).Errorf("rollback failed: %v", err)  
    }  
    return err  
}
```

1. Structured Context: The Rollback is a pivotal state-changing operation. making its failure context critically important. This insertion moves beyond logging a simple error string by adding structured fields like revision.

2. Call Chain Robustness: Analysis revealed the risk of silent failures, as all upstream callers simply propagate the error without logging it. This insertion will make the failed rollback be recorded at its source, regardless of the caller's implementation.

3. Consistency with Similar Cases: This pattern is well-established within our codebase. For example, xxx.RollbackRelease also logs its own failures, confirming that rollback errors are considered high-priority events.

4. Best Practices: This change follows the design principle of responsibility boundaries. The rollbackWorkload function, which executes the side-effect, is the most logical place to log the outcome. It has the most context and responsibility. Delegating this to callers is fragile and shifts responsibility, which is not in line with our internal logging specifications.

Apply Reject View Reason

Takeaways

- Our empirical study revealed real-world logging improvement is **holistic**; the majority of developer effort is on modification and enhancement, not just insertion.
- Effective logging recommendations are derived from mining the **silent practices** of mature, peer-validated codebases, not from abstract principles.
- High precision requires pruning; analyzing **logging responsibility** and implicit **logger behavior** is as critical as generation itself.
- High developer acceptance is achieved not just by the patch, but by synthesizing all pipeline **evidence** into a convincing, multi-faceted argument that builds trust .

**Thanks and
we are hiring US
summer intern!**

