A Interface Visualization

Figure 1 illustrates the client-side integration of BackportCheck within the Gerrit environment.To demonstrate the tool's utility, we describe the complete decision-making workflow: The interaction is initiated when the Release Manager clicks the **"Analyze Eligibility"** button injected into the review interface. The system then displays the decision overlay, immediately presenting the **Decision Banner (A)**. This provides immediate feedback via a color-coded probabilistic verdict, allowing the maintainer to instantly triage the change. To understand the rationale, the user reviews the **AI Explanation (B)**, where the Large Language Model synthesizes complex risk factors into a coherent natural language justification.

To validate this assessment, **the Risk Dashboard (C)** displays the key metrics, such as *Author Reliability* and *Code Spread*, allowing maintainers to verify that the AI's reasoning is grounded in hard data rather than opaque logic. Furthermore, the interface provides **Threshold Controls (D)** to adjust the decision boundary in real-time according to the user's context. Lastly, the **Metric Definitions (E)** component provides on-demand explanations for each indicator, ensuring that the calculation logic remains accessible and transparent.

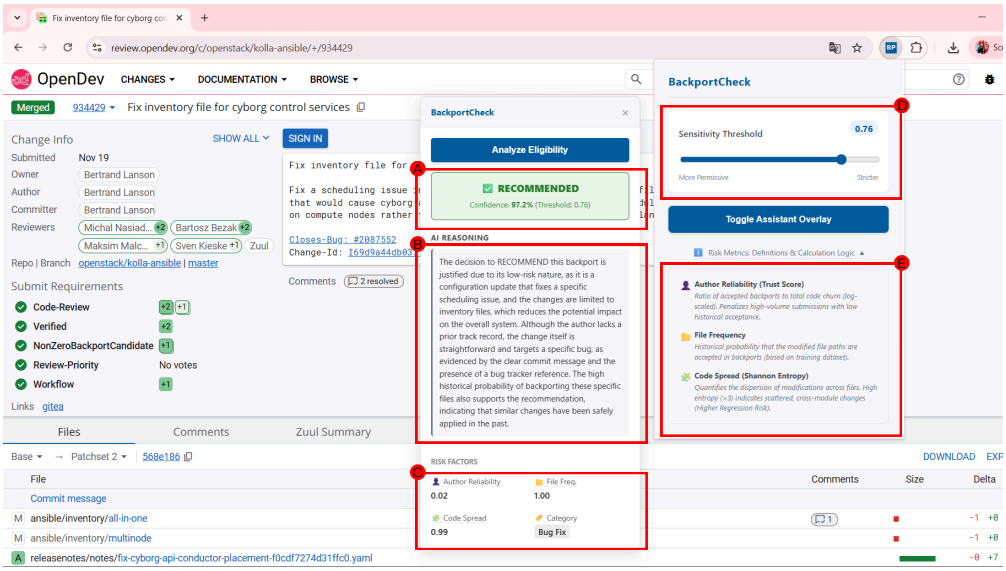

Fig. 1. **Annotated BackportCheck Interface.** The overlay augments the Gerrit code review screen with five decision-support layers: (A) The probabilistic verdict banner, (B) Natural language AI reasoning, (C) The quantitative risk dashboard, (D) Sensitivity threshold controls, and (E) Metric definitions for transparency.

B BackportCheck features

Table 1 details the comprehensive list of 37 features extracted by the inference engine, categorized by the dimensions used in the risk reporting schema.

C Explainability Prompt Structure

To ensure consistent explanations, we inject the XGBoost predictions into the structured template shown in Listing 2.

Table 1. **Comprehensive Taxonomy of the 37 State-Aware Features.** The feature vector is constructed to capture both the technical complexity of the code and the socio-technical context of the submission.

Dimension	Feature Variable	Definition & Rationale
1. Historical Context (Socio-Technical State)		
Author Reputation	`author_trust_score`	Calculated as $T_a = \frac{N_{accepted}}{\log(1+\sum C_{churn})}$. Balances acceptance count against churn volume to proxy author reliability and consistency.
	`author_success_rate`	Ratio of the author's previously accepted backports to total submissions. Indicates historical alignment with project policies.
	`author_submission_count`	Total number of past changes submitted. Serves as a quantitative proxy for author experience.
Environmental Bias	`project_acceptance_rate`	Baseline acceptance rate of the specific sub-project, used to normalize author performance against project strictness.
	`historical_file_prob`	Probability estimate based on historical data indicating how frequently changes to this specific file path are backported.
2. Complexity & Volume (Entropy)		
Scatter Metrics	`change_entropy`	Shannon Entropy ($H(X)$) of change dispersion. High values (> 5) indicate logic scattered across multiple modules, increasing verification difficulty.
	`safe_entropy_interaction`	Interaction term correlating low entropy with low-risk file types (e.g., documentation or configuration).
Cognitive Load	`churn_density`	Average lines changed per file. High density suggests complex logic concentrated within specific modules.
	`churn_log_size`	Logarithmic scale of total lines changed, normalizing the impact of large refactorings or automated updates.
	`file_count`	Total number of modified files. Represents the scope or "blast radius" of the modification.
	`deletion_ratio`	Ratio of deleted lines to total churn. High values typically indicate code cleanup or removal of unused logic.
3. Semantic Intent (NLP & Regex)		
Classification	`is_fix`	Boolean: Commit message references a bug fix pattern. Patches addressing crashes are primary candidates for uplift.
	`is_feature`	Boolean: Commit introduces new functionality. Feature additions are typically lower priority for stable backports.
	`is_refactor`	Boolean: Commit indicates restructuring without functional changes.
	`is_revert`	Boolean: Commit reverts a previous change. Reverts often address immediate regressions and are prioritized.
	`is_maintenance`	Boolean: Routine tasks such as version increments or dependency updates.
	`is_deployment`	Boolean: Changes specific to deployment scripts or playbooks.
Clarity	`msg_readability_ease`	Flesch Reading Ease score. Higher scores correlate with clearer justifications, facilitating review.
	`msg_gunning_fog`	Gunning Fog index estimating linguistic complexity of the commit message.
	`references_bug_tracker`	Boolean: Message links to an external issue tracker. Indicates compliance with process guidelines.
	`has_subject_tag`	Boolean: Adherence to project-specific subject tagging conventions.
4. Domain Risk & Safety (Heuristics)		
High Risk	`modifies_db_migration`	Boolean: Modifications to database schema migration files. Indicates potential data compatibility impact.
	`modifies_public_api`	Boolean: Modifications to public API definitions. High risk of breaking backward compatibility.
	`modifies_dependencies`	Boolean: Changes to dependency specifications. Risk of introducing downstream compatibility issues.
Value Driver	`has_security_impact`	Boolean: Presence of security-related keywords (e.g., CVE). Security fixes are high-priority candidates.
Safety	`is_pure_config`	Boolean: Change is confined to configuration files (.conf, .ini).
	`is_doc_only`	Boolean: Change modifies only documentation files (.rst, .md).
	`modifies_config`	Boolean: Configuration files are involved in the change set.
5. Structural Topology & Meta Context		
Architecture	`directory_depth`	Maximum directory depth of modified files. Deep nesting often corresponds to core logic modifications.
	`file_extension_entropy`	Heterogeneity of file types (e.g., mixing SQL, Python, and Bash). Indicates broad cross-domain impact.
Composition	`config_line_ratio`	Proportion of lines changed that belong to configuration files.
	`code_line_ratio`	Proportion of lines changed that belong to source code files.
	`is_test_change`	Boolean: Change is confined to test suites, posing minimal regression risk to production code.
	`is_ci_change`	Boolean: Changes to Continuous Integration pipelines.
Metadata	`is_bot`	Boolean: Commit generated by automated automation tools.
	`has_gerrit_topic`	Boolean: Change is associated with a specific topic series in the review system.
	`is_deployment_project`	Boolean: Repository belongs to a deployment tool context.

```
SYSTEM: Act as a Senior OpenStack Release Manager. Justify the decision to {VERDICT} this backport.
=== DECISION ===
VERDICT: {VERDICT} (Confidence: {PROB}%, Threshold: {THRESHOLD})
CATEGORY: {UI_DISPLAY_TYPE}
=== CONTEXT: HOW TO INTERPRET THE DATA ===
- Author Trust Score: Ratio of accepted backports. 0.0=New, >0.5=Trusted.
- Historical File Prob: Probability these specific files are usually backported.
- Change Entropy: Code complexity (0=Simple, >4=Complex/Scattered).
- Churn Density: Lines changed per file (High = Dense/Risky).
- Modifies DB/API: Critical risk factors.
=== FULL FEATURE VECTOR (Internal Data) ===
{FEATURE_VECTOR_JSON}
=== CHANGE ARTIFACTS ===
Commit Message: "{COMMIT_MESSAGE}"
Files Modified: {FILE_LIST_STRING}
=== INSTRUCTIONS ===
Write a professional, 2-3 sentence justification.

    (1) Analyze the Vector: Look at the "Full Feature Vector" above. Find the anomalies or strong signals.
    (2) Synthesize: Do not list the numbers. Explain their meaning.
          • Instead of "Is Pure Config is Yes", say "The change is a low-risk configuration update."
          • Instead of "Trust is 0.0", say "The author lacks a prior track record."
    (3) Explain the Verdict:
          • If REJECTED: Is it the Author? The Complexity? The specific Files?
          • If ACCEPTED: Is it the Safety (Config/Doc)? The High Trust?

RESPONSE:
```

Fig. 2. **The Prediction-Aligned Prompt Template.** The system dynamically populates the placeholders (in brackets) with the inference results and the raw feature vector before sending the request to the LLM.

D Validation via Formal Concept Analysis (FCA)

To empirically ground the feature taxonomy and justify the dashboard layout, we employed a data-driven approach inspired by **Formal Concept Analysis (FCA)** [3]. We followed a three-step methodology to extract stable decision patterns from the historical dataset of 3,422 changes:

(1) **Data Discretization (Context Creation):** Since association rule mining necessitates categorical data, continuous feature values were transformed into boolean attributes using statistical quantiles. Values $\leq Q_1$ were mapped to *Low*, values between Q_1 and Q_3 to *Medium*, and values $> Q_3$ to *High*. This transformation preserves the relative distribution of the data while enabling categorical analysis.

(2) **Rule Extraction:** We utilized the **Apriori algorithm** [1] to identify frequent itemsets and association rules. To ensure the extracted rules were statistically significant, we enforced the following constraints:
   - **Minimum Support = 0.02:** Focusing on patterns appearing in at least 2% of the dataset.
   - **Minimum Confidence = 0.65:** Ensuring a high probability of the rule's validity.
   - **Minimum Lift $\geq$ 1.0:** We utilized the Lift metric [2] to filter out coincidental correlations, retaining only positive dependencies.

(3) **Semantic Grouping:** The resulting dominant rules (Confidence $\approx$ 1.0, Lift $>$ 1) were clustered into semantic families. As shown in Table **??**, these families correspond to the dashboard components: rules predicting rejection formed the *Risk Patterns* panel, while rules predicting acceptance formed the *Intrinsic Safety* and *Logic Safety* panels.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, Santiago, Chile, 487–499.

Table 2. Key Association Rules Derived from Historical Data. These stable patterns justify the grouping of features into the "Risk" and "Safety" dashboard panels.

Antecedents (Contextual Signals)	Outcome	Conf.	Lift
Risk Patterns (Rejection Signals)			
`Low Success Rate` + `Low Readability` + `Is BugFix` + `High File Count`	**Rejected**	1.00	31.4
`Low Code Ratio` + `Med Trust` + `High File Count` + `Is BugFix`	**Rejected**	1.00	30.8
`High Entropy` + `Deep Directory` + `New Author`	**Rejected**	1.00	28.5
`CI Change` + `Low Readability` + `Modifies DB`	**Rejected**	1.00	25.1
Intrinsic Safety (Configuration & Documentation)			
`Config Only` + `Low Entropy` + `High Trust` + `Med Readability`	**Accepted**	1.00	5.9
`Low File Count` + `Config Only` + `High Success Rate`	**Accepted**	1.00	5.9
`Low Code Ratio` + `Config Only` + `High Trust`	**Accepted**	1.00	5.9
Logic Safety (Trusted Code Patterns)			
`High Trust` + `Med Readability` + `Low Entropy` + `Low File Count`	**Accepted**	1.00	5.9
`High Success Rate` + `Low Directory Depth` + `Med Readability`	**Accepted**	1.00	5.9
`High Trust` + `Med Config Ratio` + `Low Entropy`	**Accepted**	1.00	5.9

[2] Sergey Brin, Rajeev Motwani, and Craig Silverstein. 1997. Beyond market baskets: Generalizing association rules to correlations. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 265–276.

[3] Bernhard Ganter and Rudolf Wille. 1999. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, Berlin, Heidelberg.