Supplementary Material for BackportCheck: An Automated Risk Reporting Tool for Code Review

A Feature Engineering Taxonomy

Table 1 details the comprehensive list of 37 features extracted by the inference engine. To ensure clarity, each feature is defined individually with its specific rationale for risk assessment.

Author's Contact Information:

B Explainability Prompt Structure

To ensure consistent and trustworthy explanations, the raw feature vector and XGBoost predictions are injected into the structured template below (Figure 1) before being processed by the LLM.

```
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.
    (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. 1. **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.

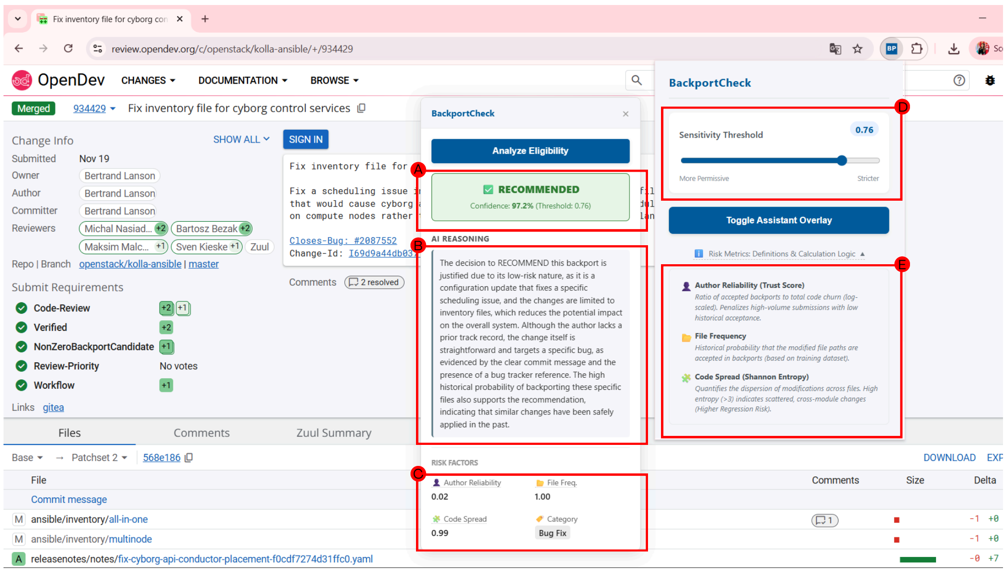
C Validation via Formal Concept Analysis (FCA)

To validate the utility of our features and justify the dashboard design, we applied Association Rule Mining (Apriori algorithm) on the historical dataset of 3,422 changes. Table 2 presents the statistically dominant decision rules (Confidence = 1.0, Lift > 1) that emerged from the data. These rules were grouped into "Concept Families" to structure the BackportCheck user interface.

D Interface Visualization

Figure 2 illustrates the client-side integration of BackportCheck within the Gerrit environment. The interface is structured into five functional zones to facilitate transparent decision-making:

The workflow begins with the **Decision Banner (A)**, which provides an immediate probabilistic verdict based on the model's score. This signal is contextualized by the **AI Explanation (B)**, where the language model synthesizes the risk factors into a natural language justification. For verification, the **Risk Dashboard (C)** exposes the underlying quantitative metrics, such as Author Reliability, File Frequency, and Code Spread. Additionally, the interface includes **Threshold Controls (D)**, allowing users to calibrate the model's strictness, and **Metric Definitions (E)** to ensure the calculation logic remains transparent to the maintainer.



Change Info
Submitted Nov 19
Owner Bertrand Lanson
Author Bertrand Lanson
Committer Bertrand Lanson
Reviewers Michal Nasiad... +2 Bartosz Bezak +2 Maksim Malc... +1 Sven Kieske +1 Zuul
Repo | Branch openstack/kolla-ansible | master

Submit Requirements
Code-Review +2 +1
Verified +2
NonZeroBackportCandidate +1
Review-Priority No votes
Workflow +1
Links gitea

```
Fix inventory file for

Fix a scheduling issue i
that would cause cyborg
on compute nodes rather

Closes-Bug: #2087552
Change-Id: I69d9a44db03
```

Comments 2 resolved

BackportCheck

RECOMMENDED
Confidence: **97.2%** (Threshold: 0.76)

AI REASONING

The decision to RECOMMEND this backport is justified due to its low-risk nature, as it is a configuration update that fixes a specific scheduling issue, and the changes are limited to inventory files, which reduces the potential impact on the overall system. Although the author lacks a prior track record, the change itself is straightforward and targets a specific bug, as evidenced by the clear commit message and the presence of a bug tracker reference. The high historical probability of backporting these specific files also supports the recommendation, indicating that similar changes have been safely applied in the past.

RISK FACTORS

Author Reliability	File Freq.
0.02	1.00
Code Spread	**Category**
0.99	Bug Fix

BackportCheck

Sensitivity Threshold 0.76
More Permissive — Stricter

Risk Metrics: Definitions & Calculation Logic ▲

Author Reliability (Trust Score)
Ratio of accepted backports to total code churn (log-scaled). Penalizes high-volume submissions with low historical acceptance.

File Frequency
Historical probability that the modified file paths are accepted in backports (based on training dataset).

Code Spread (Shannon Entropy)
Quantifies the dispersion of modifications across files. High entropy (>3) indicates scattered, cross-module changes (Higher Regression Risk).

File	Comments	Size	Delta
Commit message			
M ansible/inventory/all-in-one	1	🟥	-1 +0
M ansible/inventory/multinode		🟥	-1 +0
A releasenotes/notes/fix-cyborg-api-conductor-placement-f0cdf7274d31ffc0.yaml		🟩	-0 +7

Fig. 2. **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.

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 $T_a = \frac{N_{accepted}}{\log(1+\sum C_{churn})}$. Penalizes massive code dumps while rewarding consistent accepted patches.
	`author_success_rate`	The ratio of the author's previously accepted backports to total submissions. Indicates alignment with policy.
	`author_submission_count`	The total number of past changes submitted. Serves as a proxy for the author's experience level.
Environmental Bias	`project_acceptance_rate`	The baseline acceptance rate of the specific sub-project (e.g., Nova vs. Horizon), used to normalize performance.
	`historical_file_prob`	Bayesian probability estimate: "How often have changes to this specific file path been backported in the past?"
2. Complexity & Volume (Entropy)		
Scatter Metrics	`change_entropy`	Shannon Entropy ($H(X)$) of change dispersion. High entropy (> 5) implies "Spaghetti Code" (scattered logic).
	`safe_entropy_interaction`	Interaction term correlating low entropy with safe file types (e.g., config files).
Cognitive Load	`churn_density`	Average lines changed per file. High density implies complex logic concentrated in single modules.
	`churn_log_size`	Logarithmic scale of total lines changed, smoothing the impact of massive refactorings.
	`file_count`	Total number of modified files. A proxy for the "blast radius" of the change.
	`deletion_ratio`	Ratio of deleted lines to total churn. High values indicate cleanup/dead-code removal (lower risk).
3. Semantic Intent (NLP & Regex)		
Classification	`is_fix`	Boolean: Commit message explicitly references a bug fix pattern.
	`is_feature`	Boolean: Commit introduces a new feature (typically rejected in stable backports).
	`is_refactor`	Boolean: Commit indicates a refactoring operation without functional changes.
	`is_revert`	Boolean: Commit is a revert of a previous change (high priority, generally low risk).
	`is_maintenance`	Boolean: Routine maintenance tasks like version bumps or dependency upgrades.
	`is_deployment`	Boolean: Changes to deployment-specific scripts/playbooks.
Clarity	`msg_readability_ease`	Flesch Reading Ease score. Higher scores indicate clearer, simpler justifications.
	`msg_gunning_fog`	Gunning Fog index estimating years of education needed to understand the text.
	`references_bug_tracker`	Boolean: Message links to an external issue tracker (e.g., Launchpad/Jira).
	`has_subject_tag`	Boolean: Compliance with project subject-tagging guidelines (e.g., '[Nova]').
4. Domain Risk & Safety (Heuristics)		
High Risk	`modifies_db_migration`	Boolean: Changes to database schema migration files (Alembic). Critical regression risk.
	`modifies_public_api`	Boolean: Changes to public API definitions. High risk of breaking backward compatibility.
	`modifies_dependencies`	Boolean: Changes to dependency files ('requirements.txt'). Risk of downstream breakage.
Value Driver	`has_security_impact`	Boolean: Presence of security keywords (CVE, vulnerability). High value overrides risk.
Safety	`is_pure_config`	Boolean: Change is exclusively limited to configuration files ('.conf', '.ini').
	`is_doc_only`	Boolean: Change modifies only documentation files ('.rst', '.md').
	`modifies_config`	Boolean: Configuration files are involved (even if mixed with code).
5. Structural Topology & Meta Context		
Architecture	`directory_depth`	Maximum directory depth of modified files. Deep nesting often implies core logic.
	`file_extension_entropy`	Heterogeneity of file types (e.g., mixing SQL, Python, and Bash in one commit).
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 (Python/C++).
	`is_test_change`	Boolean: Change is confined to test suites.
	`is_ci_change`	Boolean: Changes to Continuous Integration pipelines (Zuul/Tox).
Metadata	`is_bot`	Boolean: Commit generated by automated bots (e.g., translation syncs).
	`has_gerrit_topic`	Boolean: Change is part of a named topic series in Gerrit.
	`is_deployment_project`	Boolean: Repository belongs to a deployment tool (e.g., Kolla, Ansible).

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