**1. Detection Engine (Code Smell Search)**

🔴 Weaknesses:

* Rules are **static and hand-crafted**, not adaptive.
* High **false positives/false negatives** in smell detection.
* **Context-insensitive** (doesn’t learn project-specific coding styles).

✅ Feasible ML-driven focus:

* **Adaptive Smell Detection**: train a model to refine detection rules based on project/domain feedback.
* **Context-Aware Detection**: e.g., differentiate between “bad smell” vs “acceptable trade-off” in certain projects.

⚖️ Scope: Pick **1 weakness here** (likely *false positive reduction with ML*).

**2. Prioritization Logic**

🔴 Weaknesses:

* Severity levels (Blocker, Critical, etc.) are **static, rule-based**.
* No consideration of **historical defect density**, commit frequency, or developer feedback.
* Prioritization doesn’t change over time as projects evolve.

✅ Feasible ML-driven focus:

* **Feedback-driven ranking engine** (your novel approach!) → ML model re-prioritizes smells based on developer validation.
* **Historical-context prioritization**: use past bug history + change frequency to adjust severity.

⚖️ Scope: Pick **2 weaknesses here** (because this is your main research contribution).

**3. Code Health Metric**

🔴 Weaknesses:

* Simplistic aggregate score (just counts issues with weights).
* Doesn’t include runtime/behavioral data or evolving project context.
* May misrepresent actual maintainability risks.

✅ Feasible ML-driven focus:

* **Dynamic health metric**: re-weight issues based on project history + feedback.
* Could merge with prioritization work.

⚖️ Scope: **Optional** → Fold this into your prioritization work (don’t treat it as separate unless you have extra time).

**4. Defect Density Overlay**

🔴 Weaknesses:

* Based only on **static code density** (LOC vs. issues).
* Doesn’t correlate with actual bug fix history.
* Ignores team velocity, ownership, or defect reoccurrence patterns.

✅ Feasible ML-driven focus:

* **Bug-history aware defect density**: incorporate past Jira/GitHub issues into density calculation.
* Could tie into your automatic ticket creation.

⚖️ Scope: Pick **1 weakness here** (if you want a “bridge” between detection and ticket automation).

**5. Behavioral & ML Patterns**

🔴 Weaknesses:

* SonarQube doesn’t natively learn from developer behavior.
* No real ML loop (all rules are predefined).
* No personalization per team/project.

✅ Feasible ML-driven focus:

* **Feedback Loop ML System**: let developers mark false positives / re-rank issues → system retrains.
* **Team-customized prioritization**: per-project severity learning.

⚖️ Scope: This is **your main novelty** → definitely include **1 weakness here** (feedback loop).

**📌 Final Feasible Scope by Dec 2026**

* **Detection Engine** → 1 weakness (ML-based false positive reduction).
* **Prioritization Logic** → 2 weaknesses (feedback-driven ranking + historical context).
* **Defect Density Overlay** → 1 weakness (bug-history aware density).
* **Behavioral & ML Patterns** → 1 weakness (feedback loop integration).
* (**Code Health Metric**) → optional, merge into prioritization if time permits.

👉 That’s **~5 weaknesses total** (core + optional) — totally feasible in your timeframe while balancing classes. It gives you enough *breadth* (across domains) and *depth* (ML/automation focus) for a strong thesis.