Oral Exam Notes: Learning-Based Static Program Slicing

Group 6: NS-Slicer

Time Limit: 5 Minutes

1. The Problem (0:00 - 1:00)

- **Context:** Developers constantly copy code from StackOverflow. A study found 99 vulnerable snippets propagated to 2,589 GitHub repositories.
- **The Challenge:** We need to analyze these snippets for vulnerabilities.
- **Failure of Traditional Tools:** Traditional Static Analysis (like *JavaSlicer* or *Joern*) requires **complete, compilable code** to build a System Dependence Graph (SDG).
- **Gap:** StackOverflow snippets are incomplete (missing variable declarations, missing imports). Traditional tools crash or produce empty results on them.

2. The Solution: NS-Slicer (1:00 - 2:00)

- **Proposal:** A Deep Learning approach called **NS-Slicer**.
- **Core Idea:** Instead of logically building a dependency graph, treat Program Slicing as a **classification task**.
- **Hypothesis:** Pre-Trained Language Models (PLMs) like CodeBERT have implicitly learned "Variable-Statement Dependencies" during their pre-training.
- **Goal:** Predict backward and forward slices for *incomplete* code without needing compilation.

3. Methodology (2:00 - 3:00)

- **Model Backbone:** They utilize **GraphCodeBERT** because it understands data flow better than standard BERT.
- **Input:** The source code (tokens) + a Slicing Criterion (a specific variable at a specific line).
- **Architecture:**
  1. **Encoders:** Convert code tokens into embeddings.
  2. **Pooling:** Aggregates token embeddings into "Variable Embeddings" and "Statement Embeddings".
  3. **Decoders:** Two separate Multi-Layer Perceptrons (MLP). One predicts the **Backward Slice** (what affects this variable?), one predicts the **Forward Slice** (what does this variable affect?).

4. Evaluation (3:00 - 4:00)

- **Accuracy on Complete Code:** Achieved an F1-score of **96.77%**, matching ground-truth tools almost perfectly.
- **Accuracy on Partial Code:** They simulated partial code by deleting 5-15% of lines. The model maintained high accuracy (F1 **94.66% - 96.62%**), proving it is robust to missing context.
- **Downstream Application (Vulnerability Detection):**
  - They plugged NS-Slicer into a vulnerability detector called *VulDeePecker*.
  - Because NS-Slicer could handle the partial code, it improved Vulnerability Detection F1-scores by **15.1%** (up to 73.38%).

5. Critique & Conclusion (4:00 - 5:00)

- **Strengths:**
  - **Robustness:** It works where traditional analysis fails (broken/partial code).
  - **Dual Capability:** Handles both forward and backward slicing effectively.
- **Weaknesses (Crucial for exam):**
  - **Input Limit:** Restricted by RoBERTa's **512-token limit**. It cannot slice large files, only snippets.
  - **Black Box:** Unlike traditional slicing, there is no mathematical guarantee of correctness. It is a probabilistic prediction.
  - **Aliasing:** It struggles slightly with complex variable aliasing (two variables pointing to the same memory), seeing a 12% performance drop.