Appendix

Algorithm 1: Path Inference Algorithm

Input: CWE Tree (T), root node (s), beam size (β), target depth (td)

Output: Path with the highest merged probabilities

$path.nodes \leftarrow [s], path.logProb \leftarrow 0$;
$priorityQueue \leftarrow path$
while *priorityQueue is not empty* **do**
  $p \leftarrow priorityQueue.pop()$
  $d \leftarrow len(p.nodes)$
  if $d == td$ **then**
    return p `// return the best path`
  end
  $pred \leftarrow head_d(c, p.nodes[-1])$ `// predict at depth` d
  $pred.sort()$ `// sort predicts based on log probability from large to small`
  `// only choose the children of parent node`
  for *n in pred* **do**
    if *n not in children(p.nodes[-1], T)* **then**
      $pred.remove(n)$
    end
  end
  `// choose` β `nodes and update the PriorityQueue`
  for *n in pred[:β]* **do**
    $pp \leftarrow p.deepcopy()$ `// avoid changing` p
    $pp.nodes.add(n)$
    $pp.logProb \leftarrow (pp.logProb + n.logProb)$
    $PriorityQueue.add(pp)$
  end
end

The pseudo-code of our designed tree-structure-aware and beam-search-based path inference algorithm is shown in Algorithm 1. In each decoding step, we first pop the current optimal candidate (i.e., a partial path with the highest merged probability) from the `PriorityQueue`. Then, we try to grow it by adding one node from the next depth: ❶ use the depth-specific prediction header $head_d()$ to make predicts based on the information of the commit c (depth-specific commit embedding) and information of the current partial path $p.nodes[-1]$ (label embedding of the last CWE node of the

current partial path); ❷ select the top β (i.e., beam size) CWE nodes with the highest predicted probabilities from the children of the last CWE node of the current partial path. Each grown path is pushed into the `PriorityQueue`. Once a path with length of target depth is retrieved, all other candidate paths in the `PriorityQueue` are pruned. The retrieved path is returned as the one with the highest merged probabilities from the beam search.