ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TRẦN NGUYỄN TIẾN THÀNH

HỒ VĨNH NHẬT

ĐỒ ÁN CHUYÊN NGÀNH

KIỂM THỬ HỘP ĐEN TỰ ĐỘNG CHO ỨNG DỤNG ANDROID SỬ DỤNG HỌC TĂNG CƯỜNG

Automatic black-box testing ultilizing reinforcement learning for Android application

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

TP. HỒ CHÍ MINH, 2025

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

TRẦN NGUYỄN TIẾN THÀNH – 22521364

HỒ VĨNH NHẬT – 22521013

ĐỒ ÁN CHUYÊN NGÀNH

KIỂM THỬ HỘP ĐEN TỰ ĐỘNG CHO ỨNG DỤNG ANDROID SỬ DỤNG HỌC TĂNG CƯỜNG

Automatic black-box testing ultilizing reinforcement learning for Android application

CỬ NHÂN NGÀNH AN TOÀN THÔNG TIN

GIẢNG VIÊN HƯỚNG DẪN

THS. NGHI HOÀNG KHOA

TP. HỒ CHÍ MINH, 2025

LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn đến Ban giám hiệu Trường Đại học Công nghệ Thông tin – Đại học Quốc Gia Thành Phố Hồ Chí Minh vì đã tạo điều kiện học tập, thí nghiệm, nghiên cứu tốt nhất cho sinh viên. Cảm ơn quý thầy cô giảng dạy tại trường nói chung và Khoa Mạng máy tính & Truyền thông nói riêng vì đã chỉ bảo tận tâm, nhiệt huyết và truyền đạt những kiến thức chuyên môn bổ ích, những kinh nghiệm quý báu trong suốt quá trình học tập, rèn luyện của chúng tôi tại trường.

Chúng tôi cũng xin chân thành gửi lời tri ân đến Giảng viên hướng dẫn đồ án chuyên ngành - ThS. Nghi Hoàng Khoa đã trực tiếp đồng hành, quan tâm và hướng dẫn, chỉ dạy tận tình trong suốt quá trình chúng tôi tìm hiểu và thực hiện đề tài nghiên cứu.

Bên cạnh đó, chúng tôi cũng xin gửi lời cảm ơn sâu sắc đến quý thầy cô, anh chị đang công tác tại Phòng thí nghiệm An toàn thông tin (InSecLab) vì đã luôn tạo điều kiện về cơ sở vật chất và giúp đỡ chúng tôi tận tình về kiến thức chuyên môn lẫn kinh nghiệm trong thời gian làm đồ án chuyên ngành này.

Cuối cùng, do kiến thức chuyên môn còn hạn chế nên đồ án chuyên ngành chắc chắn không tránh khỏi những thiếu sót. Rất mong nhận được những lời nhận xét, đóng góp ý kiến và phê bình từ quý thầy cô trong hội đồng để đồ án chuyên ngành này được hoàn thiện hơn.

Trần Nguyễn Tiến Thành **Hồ Vĩnh Nhật**

NỘI DUNG

DANH SÁCH HÌNH ẢNH

DANH SÁCH BẢNG BIỂU

DANH MỤC TỪ VIẾT TẮT

ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
DRL	**D**eep **R**einforcement **L**earning
AUT	**A**ndroid **U**nder **T**est
MDP	**M**arkov **D**ecision **P**rocess
DDPG	**D**eep **D**eterministic **P**olicy **G**radient
SAC	**S**oft **A**ctor- **C**ritic
MCTS	**M**onte **C**arlo **T**ree **S**earch
GNN	**G**raph **I**somorphism **N**etwork
SLM	**S**mall **L**anguage **M**odel
DQN	**D**eep **Q** - **N**etwork
DDQN	**D**ouble Dueling **D**eep **Q** - **N**etwork
PER	**P**rioritized **E**xperience **R**eplay
TRPO	**T**rust **R**egion **P**olicy **O**ptimization
PPO	**P**roximal **P**olicy **O**ptimization
MARL	**M**ulti - **A**gent **R**einforcement **L**earning
MADDPG	**M**ulti - **A**gent **D**eep **D**eterministic **P**olicy **G**radient
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
CNN	**C**onvolutional **N**eural **N**etwork
MLP	**M**ulti - **L**ayer **P**erceptron

DANH MỤC TỪ TẠM DỊCH

Học máy	Machine Learning
Mô hình học máy	Machine Learning Model
Học tăng cường	Reinforcement Learning
Học tăng cường sâu	Deep Reinforcement Learning
Mạng nơ-ron đồ thị	Graph Neural Network
Mạng đẳng cấu đồ thị	Graph Isomorphism Network
Mô hình ngôn ngữ nhỏ	Small Language Model
Mạng nơ-ron tích nối chặt	Densely Connected Convolutional Network
Ngôn ngữ đánh dấu mở rộng	EXtensible Markup Language
Giá trị chất lượng trong học tăng cường	Quality Value
Học tăng cường đa tác nhân	Học tăng cường đa tác nhân
Mạng Q sâu kép kết hợp dueling	Double Dueling Deep Q-Network
Mạng nơ-ron tích chập	Convolutional Neural Network
Mạng nơ-ron nhiều lớp	Multi-Layer Perceptron

TÓM TẮT BÁO CÁO

Kiểm thử Android là một bước quan trọng khi làm việc với các ứng dụng Android,điều này nhằm đảo bảo được chất lượng và an toàn của sản phẩm.Tuy nhiên việc kiểm thử thủ công (manual testing) thường rất tốn thời gian và chi phí cho nhân lực, thiếu sự linh hoạt và thường rất phụ thuộc và kinh nghiệm của các nhân viên kiểm thử (Tester).

Học máy (Machine learning - ML) là một phần trong lĩnh vực trí tuệ nhân tạo (Artificial Intelligent - AI), tập trung chủ vào việc phát triển nên các thuật toán để máy tính có thể học hỏi, điều này đã mang lại rất nhiều tiến bộ trong việc tự động hóa các tác vụ phức tạp. Học tăng cường (Reinforcement Learning - RL) là một nhánh trong học máy, đã mở ra tiềm năng lớn trong việc cải thiện hiệu quả và tính linh hoạt so với các phương pháp truyền thống. Các phương pháp kiểm thử thủ công thường không thể phát hiện hiệu quả các lỗi phức tạp trên UI, chẳng hạn như crash do tương tác không hợp lệ hoặc lỗi nhập liệu từ các trường dữ liệu. RL cho phép hệ thống học hỏi từ môi trường thử nghiệm thông qua tương tác thử và sai (Try & Error) từ đó tự động hóa quá trình kiểm thử và tối ưu hóa việc kiểm thử dựa trên kết quả trả về từ mồi trường để huấn luyện và tối ưu hóa việc chọn hành động giúp tìm ra vấn đề nhanh hơn.

Trong đồ án này , nhằm mục đích tạo ra một phương pháp tự động khai thác và kiểm thử hộp đen (black-box) các ứng dụng Android để tìm ra các lỗi giao diện như crash, lỗi nhập liệu, hoặc các vấn đề tương tác khác, , đồng thời tối ưu hóa thời gian và chi phí kiểm thử. Chúng tôi tiến hành nghiên cứu đối với các bài báo về các ứng dụng kiểm thử như ARES, OAT, DQT,. . . . So sánh các ứng dụng trên từ nhiều mặt và chọn ra các kỹ thuật tối ưu từ các ứng dụng trên để phát triển và áp dụng cho đồ án của chúng tôi, từ đố tối ưu hóa việc kiểm thử để đạt được hiệu suất tốt nhất.

Để thực hiện nghiên cứu và phát triển đồ án chúng tôi đã sử dụng bộ dữ liệu được thi thu thập từ các kho ứng dụng Android F-Droid, và bộ data set của DQT[1].

[1]https://github.com/Yuanhong-Lan/AndroTest24

Chương 1

TỔNG QUAN ĐỀ TÀI

Tóm tắt chương

Trong phần này, chúng tôi xin trình bày lý do chọn đề tài kiểm thử hộp đen cho ứng dụng Android sử dụng học tăng cường, các công trình nghiên cứu liên quan và những điểm mới của đề tài so với các nghiên cứu trước. Đồng thời, chúng tôi cũng đưa ra mục tiêu, phạm vi và đối tượng nghiên cứu, cũng như là phương pháp nghiên cứu. Cuối cùng chúng tôi đưa ra cấu trúc của đồ án chuyên ngành.

1.1 Lý do chọn đề tài

Thị trường ứng dụng di động (mobile app) đang lớn mạnh tại Việt Nam, thể hiện qua mức doanh thu ấn tượng 865 triệu đô la vào năm 2022[1].Năm 2024, thị trường ứng dụng di động tiếp tục tăng trưởng mạnh, với tốc độ tăng trưởng hàng năm khoảng 18-25%. Doanh thu từ ứng dụng Android được ước tính đóng góp phần lớn vào tổng doanh thu 1,1 tỷ USD của thị trường ứng dụng di động Việt Nam[2]. Nhờ tỷ lệ sử dụng điện thoại thông minh (Smartphone) và Internet tại Việt Nam không ngừng tăng nhanh, ứng dụng di động hiện đã trở thành một phần quan trọng trong cuộc sống của người dân nơi đây. Cùng với đó nhu cầu về tương tác giữa người dùng và ứng dụng đã trở thành một yếu tố quan trọng. Tuy nhiên việc kiểm thử các ứng dụng này lại gặp rất nhiều khó khăn khi cần tốn rất nhiều thời gian cũng như nhân lực để kiểm thử hoàn toàn ứng dụng. Để giải quyết vấn đề này,phương pháp được đề ra là ứng dụng trí tuệ nhân tạo (AI) cụ thể là học tăng

[1] https://www.adjust.com/vi/blog/the-state-of-mobile-apps-in-vietnam-2023/

[2] https://ictvietnam.vn/soi-dong-thi-truong-ung-dung-di-dong-tai-viet-nam-67213.html

cường (RL) trong việc phát triển phần mềm kiểm thử ứng dụng nhằm mục tiêu tối ưu hóa quy trình kiểm thử, giảm chi phí và thời gian thực hiện, đồng thời tăng khả năng phát hiện lỗi và khả năng thích nghi tốt.

Tuy nhiên không phải ứng dụng Android nào cũng có sẵn mã nguồn (Source code) để các tester có thể kiểm thử White-box các chức năng của ứng dụng. Hơn thế, một số lỗi chức năng có thể chỉ được phát hiện thông qua chạy thử ứng dụng (Runtime),và yêu cầu kết hợp các thao tác hành động theo một trình tự nhất định để có thể phát hiện lỗi, dẫn đến gia tăng độ khó không cần thiết cho công việc kiểm thử cũng như tăng gánh nặng cho các tester khi phải thực hiện một lượng lớn các thao tác kiểm thử cơ bản. Vì vậy chúng tôi chọn xây dựng mô học tăng cường kiểm thử hộp đen cho ứng dụng Android, nhằm giải quyết vấn đề yều cầu mã nguồn khi kiểm thử - kiểm thử hộp đen, yều cầu ứng dụng được khởi chạy khi kiểm thử - môi trường RL và yêu cầu các thao tác kiểm thử cơ bản nhiều lần và có khả năng học trong quá trình kiểm thử - học tăng cường.

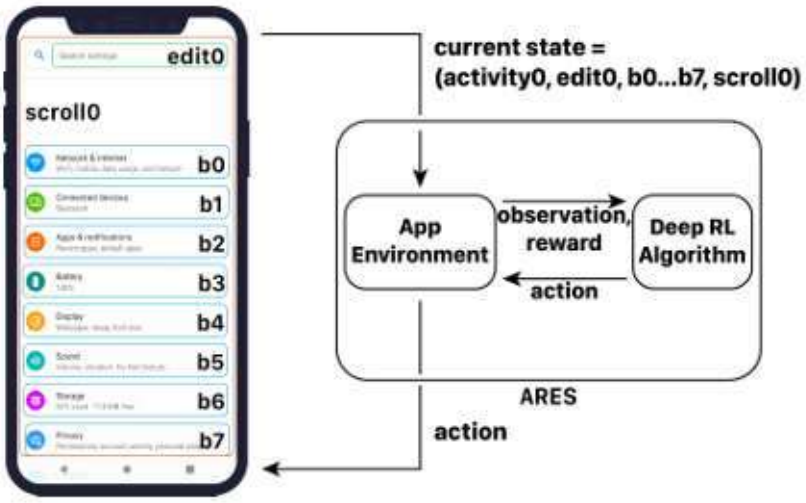
1.2 Các nghiên cứu liên quan

1.2.1 Deep Reinforcement Learning for Black-Box Testing of Android Apps (ARES)

ARES (Application of Reinforcement learning to Android Software testing)[5] là một phương pháp kiểm thử tự động hộp đen cho ứng dụng Android, sử dụng kỹ thuộc học tăng cường sâu (Deep Reinforcement Learning - DRL) nhằm tối ưu hóa quá trình kiểm thử giao diện và phát hiện lỗi trong các ứng dụng di động. Đây là hệ thống đầu tiên tích hợp DRL vào quá trình kiểm thử hộp đen điều này khiến ARES vượt trội về độ bao phủ mã và khả năng phát hiện lỗi so với các công cụ kiểm thử như Monkey, Sapienz, TimeMachine hay Q-Testing

Để có thể áp dụng RL ,đầu tiên phải ánh xạ vấn đề kiểm thử black-box Android thành chuẩn hóa toán học của RL: một MDP, được định nghĩa bởi bộ 5 phần tử, $\langle S,A,R,P,\rho_0 \rangle$ Trong đó :

- S : Là state, trạng thái của ứng dụng
- A : Là action, hành động được thực hiện
- R : Là reward, phần thưởng được trả về



Hình 1.1: ARES testing workflow

- P : Là chuyển trạng thái giữa 2 giao diện sau khi hành động
- ρ_0 : Là xác suất để agent bắt đầu tại trạng thái $s \in S$ khi bắt đầu một episode

ARES tiến hành kiểm thử thông qua 5 bước chính như sau:

1. State Representation (Biểu diễn trạng thái GUI): ARES biểu diễn trạng thái ứng dụng Android dưới dạng vector tổ hợp giữa activity hiện tại (mã hóa One-hot) và các widget hiện diện trên giao diện (có thể là Button, Text field, v.v). Việc biểu diễn này đảm bảo hệ thống chỉ dựa vào GUI – hoàn toàn theo hướng Black-box, không cần truy cập mã nguồn. Vector trạng thái dạng:

   $$s = (a_0, \ldots, a_n, w_0, \ldots, w_m)$$

   Trong đó : $a - i$ là một mã hóa One-hot của hoạt động hiện tại, w_i là vector trạng thái w_i bằng 1 nếu Widget thứ có sẵn trong hoạt động hiện tại; nó bằng 0 nếu không.

2. Action Definition (Xác định hành động kiểm thử): Tập hành động trong MDP(A) của ARES bao gồm tất cả các sự kiện có thể tương tác với GUI, ví dụ như Click, Long-click, Scroll, nhập liệu,... Ngoài ra, hai hành động hệ thống (bật/tắt Internet và xoay màn hình) cũng được thêm vào. Mỗi hành động gồm 3 thành phần: Widget đích, chỉ số chuỗi nhập, và kiểu tương tác (Click hay Long-click).

3. Transition Probability Function : xác định trạng thái nào ứng dụng có thể chuyển đến sau khi ARES đã thực hiện một hành động. Trong trường hợp

này , điều này được quyết định hoàn toàn bởi việc thực thi AUT: ARES quan sát quá trình một cách thụ động, thu thập trạng thái mới sau khi chuyển tiếp đã xảy ra.

4. Reward Function (Hàm phần thưởng thích ứng): ARES đã thiết kế hàm phần thưởng dựa trên khả năng khám phá trạng thái mới và phát hiện lỗi trong quá trình kiểm thử . Phần thưởng lớn (+1000) được cấp khi khám phá activity mới hoặc phát hiện lỗi (Crash), phạt (-100) nếu thoát ra ngoài ứng dụng, và phạt nhẹ (-1) nếu thực hiện hành động lặp lại . Cơ chế này thúc đẩy việc tìm trạng thái mới thay vì lặp lại hành vi cũ, đồng thời ưu tiên phát hiện lỗi sớm.

$$r_t = \begin{cases} \Gamma 1 & \text{if } a(s_{t+1}) \notin a(E_t) \text{ or crash} \\ -\Gamma 2 & \text{if } pack(a(s_{t+1})) \neq pack(AUT) \\ -\Gamma 3 & \text{otherwise.} \end{cases}$$

Trong đó :$\Gamma 1 \gg \Gamma 2 \gg \Gamma 3$ và $\Gamma 1 = 1000$,$\Gamma 2 = 100$, $\Gamma 3 = 1$

5. Deep RL Algorithms & Training (Học chính sách bằng mạng nơ-ron sâu): ARES hỗ trợ các thuật toán học tăng cường sâu hiện đại gồm:

   - DDPG (Deep Deterministic Policy Gradient): phù hợp với không gian hành động liên tục.

   $$L(\phi, D) = E\left[\left(Q_\phi(s_t, a_t) - (r_t + \gamma(1-d)\max_{a_{t+1}} Q_\phi(s_{t+1}, a_{t+1}))\right)^2\right]$$

   - TD3 (Twin Delayed DDPG): khắc phục việc đánh giá Q-value quá cao.
   - SAC (Soft Actor-Critic): sử dụng entropy để khám phá đa dạng hơn.

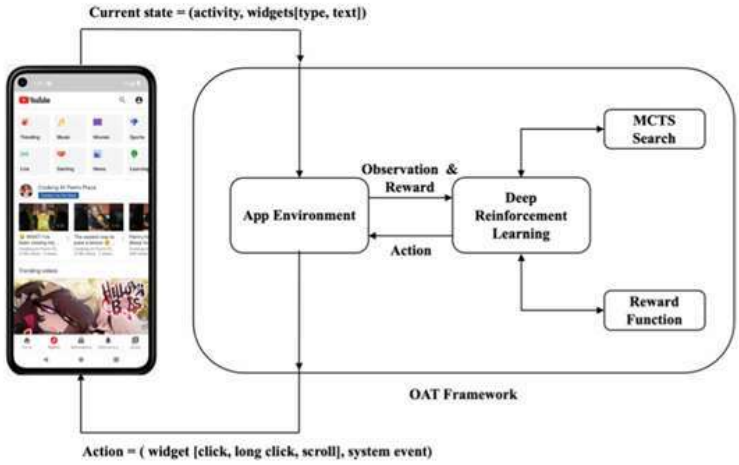
   $$\pi^* = \arg\max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))\right]$$

   Các thuật toán này được huấn luyện qua môi trường giả lập theo chuẩn của OpenAI Gym. Dữ liệu GUI sẽ được trích xuất thông qua Appium từ máy ảo Android và truyền về mạng nơ-ron để tối ưu hoá quá trình chọn lựa hành động tiếp theo.

1.2.2 OAT: An Optimized Android Testing Framework Based on Reinforcement Learning

OAT (Optimized Android Testing) [1] là một framework kiểm thử tự động dành cho các ứng dụng Android, sử dụng phương pháp học tăng cường sâu (Deep Reinforcement Learning - DRL) để tôi ưu hóa quá trình duyệt trạng thái giáo diện. Mục

tiêu chính của OAT là phát hiện lỗi và mở rộng độ phủ code bằng cách dẫn hướng đến các trạng thái khó tiếp cận trong quá trình kiểm thử truyền thống. OAT thực hiện kiểm thử Android dựa trên 4 thành phần chính:



Hình 1.2: OAT workflow

1. Fine-Grained GUI State Definition (Xác định trạng thái GUI chi tiết): OAT trích xuất dữ liệu GUI (Graphical User Interface) thông qua Appium và UIAutomator. Khác với phương pháp trước đây chỉ dựa vào loại Widget, OAT mã hóa trạng thái GUI bằng cách sử dụng vector One-hot để biểu diễn không chỉ loại mà còn cả văn bản hiển thị của từng Widget, đồng thời phân biệt các activity hiện tại của ứng dụng

$$GUIstate = (a_0, \ldots, a_n, w_0, \ldots, w_m)$$

$$a_i = \begin{cases} 1 & \text{if GUI is in activity(i)} \\ 0 & \text{otherwise} \end{cases}$$

$$w_j = \begin{cases} 1 & \text{if widget[type text] exists} \\ 0 & \text{otherwise} \end{cases}$$

Trong đó : a_i là hoạt động hiện tại của ứng dụng, w_i là Widget tương ứng với cùng một văn bản và loại có tồn tại hay không.

$$(a_0, a_1) = (\text{login.activity}, \text{main.activity})$$

$$(w_0, w_1, w_2) = (\text{widget[input, user]}, \text{widget[button, login]}, \text{widget[textview, content]})$$

$$\text{Ví dụ: GUI stat} = (a_0, a_1, w_0, w_1, w_2) = (1, 0, 1, 1, 0)$$

2. Reinforcement Learning with EfficientZero and MCTS (Học tăng cường với EfficientZero và tìm kiếm MCTS):Khác với các phương pháp học tăng cường trước đó như ARES sử dụng SAC, OAT tích hợp mô hình học sâu EfficientZero kết hợp với chiến lược tìm kiếm Monte Carlo Tree Search (MCTS) để điều hướng kiểm thử. MCTS cho phép OAT khám phá cây trạng thái GUI theo cách ưu tiên các hành động mang lại phần thưởng cao và có tiềm năng mới, tránh rơi vào tình trạng lặp lại như các phương pháp cũ.

$$USBScore = Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \left(c_1 + \log\left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right)$$

Trong đó :

- Q(s,a) đại diện cho giá trị phần thưởng ước tính của hành động a trong trạng thái s.
- P(s,a) đại diện cho xác suất lựa chọn hành động a trong trạng thái s.
- N(s,a) đại diện cho số lần lựa chọn tích lũy của hành động a dưới trạng thái s.

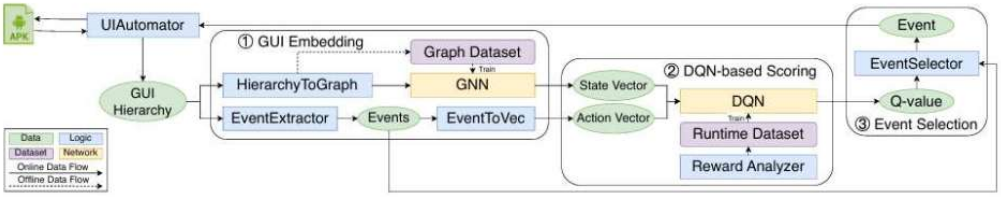
3. Reward Function (Hàm phần thưởng dựa trên trạng thái GUI chi tiết): Hàm phần thưởng trong OAT được thiết kế lại để phù hợp với trạng thái GUI chi tiết. Hệ thống sẽ cấp phần thưởng cao $R_1 = 1000$ khi phát hiện trạng thái GUI mới hoặc Crash mới, phạt nặng nếu rời khỏi ứng dụng kiểm thử $R_2 = -100$, và phạt nhẹ với các hành động lặp lại $R_3 = -1$). Khác với ARES, OAT chỉ trao phần thưởng cho Crash chưa từng xuất hiện, giúp tránh học chiến lược không hiệu quả

$$Reward = \begin{cases} R1 & \text{new state or new crash is found} \\ R2 & \text{leave testing app} \\ R3 & \text{otherwise} \end{cases}$$

4. Automated Testing Pipeline (Quy trình kiểm thử tự động): OAT được triển khai sử dụng Appium để điều khiển emulator Android và thu thập trạng thái GUI, kết hợp với các công cụ như JaCoCo và Emma để đo độ bao phủ mã. Mô hình học sâu liên tục nhận trạng thái từ môi trường để chọn hành động, thực hiện và nhận phần thưởng để cập nhật chính sách kiểm thử. Toàn bộ hệ thống cho phép kiểm thử tự động, lặp lại và có thể mở rộng.

1.2.3 Deeply Reinforcing Android GUI Testing with Deep Reinforcement Learning (DQT)

DQT[3] là một phương pháp kiểm thử giao diện (GUI) của ứng dụng Android bằng cách sử dụng kỹ thuật học tăng cường sâu (Deep Reinforcement Learning – DRL). Khác với các phương pháp truyền thống thường gặp khó khăn trong việc chia sẻ kết quả kiểm thử giữa các trạng thái tương tự, DQT sử dụng kỹ thuật nhúng đồ thị (Graph embedding) để có thể khai thác thông tin ngữ nghĩa và cấu trúc của các Widget, từ đó tăng khả năng nhận diện trạng thái tương đồng và chia sẻ kết quả kiểm thử giữa các trạng thái hiệu quả trong không gian trạng thái - hành động lớn.



Hình 1.3: DQT Workflow

Phương pháp DQT thực hiện nghiên cứu dựa trên 4 bước chi tiết như sau:

1. GUI Embedding (Nhúng giao diện dưới dạng đồ thị):Đầu tiên, DQT sẽ chuyển đổi cây phân cấp GUI (GUI hierarchy) dưới dạng XML thành đồ thị widget (widget graph) và các Node trong đồ thị đại diện cho các Widget đã được mã hóa thành Vector bằng cách trích xuất và nhúng các đặc trưng quan trọng như:

   - Thuộc tính boolean (clickable, scrollable,. . .),
   - Vị trí tọa độ,
   - Văn bản liên quan đến nội dung (text, content-desc),
   - Văn bản không liên quan đến nội dung (resource-id, class),...

   Mô hình Graph Neural Network (GNN) dựa trên InfoGraph + GIN đãđược huấn luyện trước trên hơn 100,000 cấu trúc GUI để tạo ra vector biểu diễn trạng thái chính xác và giàu thông tin

2. - Action Embedding (Nhúng hành động): Từ các widget có thể tương tác, DQT trích xuất tất cả các hành động (click, scroll, long-click, input text, v.v.) và mã hóa chúng thành vector 86 chiều, bao gồm:

   - Vector đặc trưng của node Widget tương ứng.

- Một One-hot Vector cho loại hành động (9 loại được định nghĩa trước).Việc này giúp nhận diện các hành động tương tự, hỗ trợ chia sẻ kết quả kiểm thử giữa những hành động tương đồng trong các trạng thái khác nhau.

3. DQN-based Scoring (Đánh giá hành động bằng DQN): DQT sử dụng Deep Q-Network (DQN) để học từ các tương tác trong quá trình kiểm thử và tính toán giá trị Q cho từng hành động ở trạng thái hiện tại. Đặc biệt:

   - Kiến trúc mạng sử dụng Dense Block (dựa trên DenseNet) để tăng hiệu quả học.
   - Kết hợp với Dueling DQN để đánh giá riêng biệt giá trị trạng thái và giá trị hành động.
   - Áp dụng Prioritized Experience Replay để học hiệu quả từ các tương tác quan trọng nhất. DQN của DQT được thiết kế đặc biệt để xử lý số lượng lớn hành động và trạng thái có cấu trúc phức tạp trong các ứng dụng Android hiện đại.

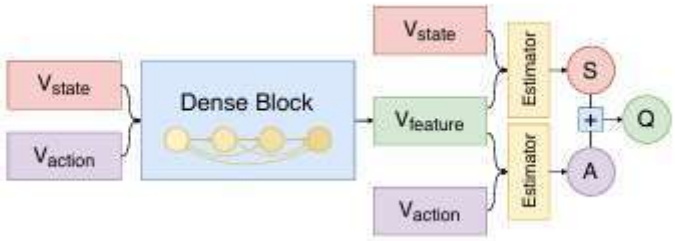
4. Reward Function and Exploration (Hàm thưởng và khám phá theo tò mò):

$$r = (R_{base} + R_{state} + \lambda R_{trans}) \times (1 + R_{time})$$

   - R_{base} : phần thưởng cơ bản đầu tiên là R_{base} khuyến khích việc khám phá các hoạt động mới và các trạng thái chưa được khám phá. R_{base} $+=$ 50 khi một hoạt động mới được đạt tới.Đối với các hoạt động chưa được khám phá, sử dụng các vector trạng thái dựa trên nhúng đồ thị của mình và tính toán độ tương đồng cosine(σ_b) giữa các trạng thái. Khi σ_b càng nhỏ chứng tỏ 2 trạng thái là khác nhau và được cộng 10 khi $\sigma_b < 0.85$, khi >0.99 thì -0.5 và các chường hợp còn lại thì được cộng thêm bằng 1 - σ_b và giá trị của σ_b nằm trong đoạn [0.1]
   - R_{state} : Là tỷ lệ khám phá trạng thái, được ký hiệu là R_{state} Yếu tố này khuyến khích việc khám phá các trạng thái chưa được khám phá tương ứng với các chức năng ít được kích hoạt, điều này có thể dẫn đến những trạng thái chưa được khám phá mới. Tính toán σ_s, giá trị trung bình của độ tương đồng cosin giữa trạng thái hiện tại và tất cả các trạng thái khác đã được thử nghiệm dưới cùng một hoạt động và đặt $R_{state} = 1 - \sigma_s$
   - R_{trans} : là tỷ lệ chuyển đổi. khuyến khích việc khám phá các lối đi chuyển đổi khác nhau giữa các trạng thái vì chúng thường tương ứng với các đoạn mã khác nhau. Đối với một chuyển tiếp $t = (s_t, a_t, s_t + 1)$ nơi s_t đang trong hoạt động act_t và s_{t+1} đang trong act_{t+1}, tính toán σ_t, trung bình độ tương đồng cosine giữa t và tất cả các chuyển tiếp đã thử

nghiệm khác từ act_t đến act_{t+1}, và đặt R_{train} bằng $1 - \sigma_b$. Lưu ý việc gán một trọng số λ cho R_{train} được đặt ở mức 0.5 vì R_{state} có ý nghĩa hơn.

- R_{time} : phần thưởng thêm nhằm tối ưu việc tìm kiếm các trạng thái khác sau khi đã chạy một khoảng thời gian khiến cho các trạng thái mới trở nên ít đi. Nó tỷ lệ thuận với bước kiểm tra hiện tại, chia cho một yếu tố thời gian (ví dụ: 3600), và chỉ được trao khi R_{base} dương. Thêm vào đó, một phần thưởng âm là -10 được phân bổ trực tiếp cho phần thưởng r khi quá trình thử nghiệm thoát ra khỏi AUT.



Hình 1.4: Mô hình Deep Q-Network cuả DQT

1.2.4 UI/Application Exerciser Monkey

Monkey [3] là một công cụ dòng lệnh (command line) được tích hợp trong Android SDK và có thể chạy trên bất kỳ phiên bản trình giả lập hoặc thiết bị nào. Monkey sẽ tạo ra các chuỗi sự kiện người dùng ngẫu nhiên vào hệ thống, hoạt động như một bài kiểm tra ứng dụng đang phát triển. Công cụ này thường được sử dụng để mô phỏng các tương tác ngẫu nhiên, giúp phát hiện các lỗi crash hoặc hành vi bất thường mà người dùng thực tế có thể gây ra Monkey bao gồm một số tùy chọn, nhưng chúng được chia thành bốn loại chính:

- Các tùy chọn cấu hình cơ bản, chẳng hạn như thiết lập số lượng sự kiện cần thử.
- Các ràng buộc về mặt vận hành, chẳng hạn như giới hạn thử nghiệm trong một gói duy nhất.
- Các loại sự kiện và tần suất.
- Tùy chọn Debug.

[3]https://developer.android.com/studio/test/other-testing-tools/monkey

khi chạy Monkey sẽ tạo ra các sự kiện và gửi đến ứng dụng và theo dõi hoạt động của ứng dụng đang được kiểm tra và tìm kiếm các điều kiện sau:

- Nếu hạn chế Monkey chạy trong một hoặc nhiều ứng dụng cụ thể, lúc nãy monkey sẽ theo dõi việc điều hướng đến các ứng dụng khác và chặn chúng.
- Nếu ứng dụng gặp sự cố hoặc nhận được bất kỳ ngoại lệ nào chưa được xử lý, Monkey sẽ dừng lại và báo cáo lỗi.
- Nếu ứng dụng tạo ra lỗi ứng dụng không phản hồi, Monkey sẽ dừng lại và báo cáo lỗi.

1.3 Mục tiêu, đối tượng và phạm vi nghiên cứu

1.3.1 Mục tiêu nghiên cứu

Dựa trên các bài báo khoa học và xem xét các thành phần khác nhau mà ứng dụng bài báo đề cập từu đó thực hiện phát triển các mục tiêu như sau:

- Xây dựng một hệ thống kiểm thử hộp đen tự động cho ứng dụng Android dựa trên Reinforcement Learning.
- Áp dụng thuật toán kết hợp với ảnh chụp UI để tăng cường khả năng học và ra quyết định.
- Kết hợp Graph Neural Network (GNN) và Small Language Models (SLMs) để nâng cao khả năng hiểu biết trạng thái và sinh dữ liệu đầu vào cho các trường kiểm thử.
- Tăng độ bao phủ kiểm thử (Instruction coverage) và số lượng lỗi phát hiện được so với phương pháp truyền thống, mô hình hiện có.

1.3.2 Đối tượng nghiên cứu

- Ứng dụng Android (APK): Đây là đối tượng được kiểm thử trong nghiên cứu, các ứng dụng được dùng để kiểm thử sẽ có giao diện người dùng (UI) đa dạng và tương tác phong phú để đảm bảo hệ thống kiểm thử được huấn luyện trong môi trường thực tế nhất.
- Các kỹ thuật Reinforcement Learning (RL), đặc biệt là DQN, GINE,...

1. DQN (Deep Q-Network) : là một kỹ thuật trong RL, là sự kết hợp giữa Q-Learning truyền thống và Deep Neural Network để thực hiện quá trình dự đoán hành ddoognj tiếp theo và tối ưu hành động được lựa chọn.
2. GINE : một biến thể nâng cao của GNN (Graph Neural Network), được phát triển dựa trên GIN(Graph Isomorphism Network) và bổ sung thêm khả năng xử ký đặc trưng của các cạnh (Edge Features) trong đồ thì thay vì chỉ dùng đặc trưng của các Node

- Giao diện người dùng (UI) và hành vi tương tác của người dùng với UI : Giao diện UI chính là thành phần mô hình học tăng cường tương tác với để học hành vi kiểm thử. gồm các phần tử như nút bấm, hộp văn bản, checkbox, menu,... được trích xuất dưới 2 dạng:
  1. XML Hierarchy: cấu trúc dạng cây phản ánh quan hệ cha-con giữa các phần tử UI.
  2. Ảnh chụp màn hình (UI Patch Image): giúp mô hình có thêm thông tin về bố cục, màu sắc và các yếu tố không thể lấy từ XML.

1.3.3 Phạm vi nghiên cứu

- Nghiên cứu và triển khai hệ thống kiểm thử hộp đen tự động cho ứng dụng Android.
- Tập trung vào khả năng học hành vi kiểm thử của mô hình RL qua ảnh UI và thông tin trạng thái.
- Không đi sâu vào kiểm thử hộp trắng hoặc kiểm thử dựa trên mã nguồn.

1.4 Phương pháp nghiên cứu

Trong quá trình thực hiện, chúng tôi đã tiến hành nghiên cứu và tìm hiểu về các phương pháp kiểm thử tương tự từ các bài báo, đặc biệt là các bài báo kiểm thử hộp đen dựa trên học tăng cường và khảo sát các mô hình phù hợp để có thể áp dụng cho mục đích tự kiểm thử khám phá giao diện và hành vi của ứng dụng thông qua việc mô phỏng tương tác người dùng. Nhằm để nâng cao hiệu quả trong việc lựa chọn hành động (Action) kiểm thử chúng tôi đã ứng dụng thêm mô hình Deep Q-learning kết hợp với Graph neural network để trích xuất được các đặc trưng từ giao diện Bên cạnh đó nhằm để mô phỏng quá trình tương tác như người dùng

chúng tối còn áp dụng thêm SLM để sinh ra đầu vào hỗ trợ đánh giá sâu trong quá trình nhập liệu. Cuối cùng , mô hình sẽ được huấn luyện và được đánh giá thông qua các chỉ số đo lường như độ phủ lệnh (Instrument coverage), độ phủ màn hình (Activity Coverage) cũng như số lượng lỗi phát hiện được nhằm kiểm chứng tính hiệu quả và khả năng ứng dụng trong thực tế của phương pháp.

1.5 Những điểm mới của đề tài

Trong đồ án này, chúng tôi đã đưa ra được những điểm mới như sau:

- Kết hợp cả thông tin hình ảnh và cây UI XML (UI Hierarchy XML) để trích xuất thông tin trạng thái đầy đủ, hỗ trợ tốt hơn cho việc huấn luyện mô hình học tăng cường.
- Sử dụng SLM để sinh dữ liệu đầu vào, giúp phát hiện nhiều loại lỗi liên quan đến xử lý input hơn.

1.6 Cấu trúc Đồ án chuyên ngành

Cấu trúc của đồ án được chia làm 5 chương như sau:

- Chương 1: **TỔNG QUAN ĐỀ TÀI**
  Trình bày khái quát các vấn đề nghiên cứu, các công trình liên quan và định hướng nghiên cứu của đồ án.
- Chương 2: **CƠ SỞ LÝ THUYẾT**
  Trình bày chi tiết các khái niệm, cơ sở lý thuyết nền tảng cần thiết để thực hiện đồ án.
- Chương 3: **PHƯƠNG PHÁP THỰC HIỆN**
  Trình bày chi tiết giải pháp được nghiên cứu.
- Chương 4: **THỰC NGHIỆM, ĐÁNH GIÁ VÀ THẢO LUẬN**
  Trình bày phương pháp thực nghiệm, đánh giá kết quả của mô hình đề xuất.
- Chương 5: **KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**
  Đưa ra kết luận về đề tài, đề xuất một số hướng phát triển mở rộng cho các nghiên cứu trong tương lai.

Chương 2

CƠ SỞ LÝ THUYẾT

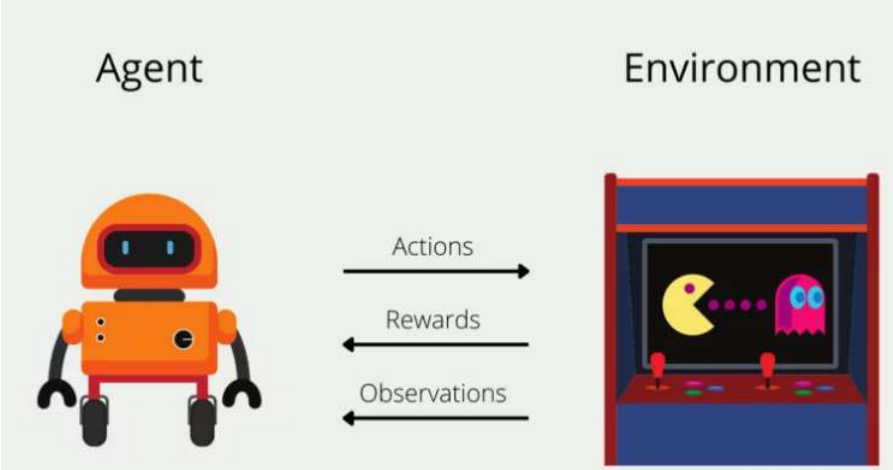
Tóm tắt chương

Cơ sở lý thuyết cho báo cáo sẽ được chúng tôi trình bày trong chương này. Cụ thể như sau: Học tăng cường (Reinforcement Learning), một số thuật toán học tăng cường dựa trên giá trị (Value-based RL), mô hình Double-Dueling Deep Q-Network (Double DDQN), dữ liệu kinh nghiệm ưu tiên (Prioritized Experience Replay), mô hình học sâu đồ thị (Graph Neural Network - GNN) sử dụng mạng đồ thị đồng dạng với đặc trưng cạnh (Graph Isomorphism Network with Edge - GINE [2]), độ phủ (coverage). Cùng với đó là các công cụ như ACVTool[4], Appium và hệ điều hành Android sẽ được chúng tôi trình bày.

2.1 Học tăng cường (Reinforcement Learning)

2.1.1 Tổng quan

Học tăng cường (Reinforcement Learning - RL) là một nhánh của học máy (Machine Learning), tập trung vào việc học cách đưa ra hành động (action) thông qua thử và sai (trial and error) trong môi trường (enviroment) để tối đa hóa tổng phần thưởng (reward) tích lũy.



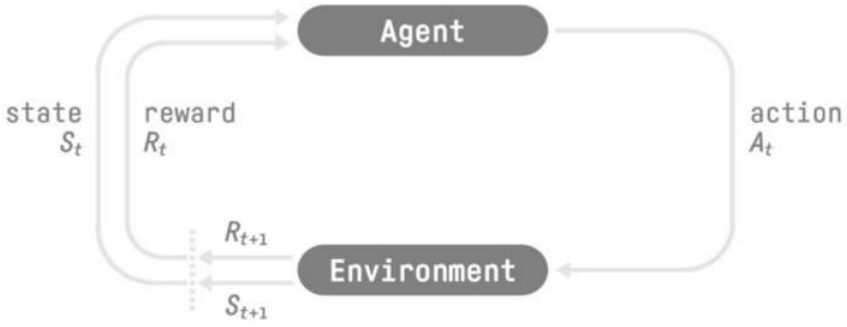
Hình 2.1: Học tăng cường.[1]

Một thuật toán học tăng cường (Reinforcement Learning - RL) thường bao gồm các thành phần sau:

- **Tác nhân (agent)**: thực hiện các hành động $a \in \mathscr{A}$ để tương tác với môi trường.
- **Môi trường (environment)** $\mathscr{E}$: phản ứng lại hành động của tác nhân, đồng thời cung cấp trạng thái và phần thưởng.
- **Trạng thái (state)** $s \in \mathscr{S}$: đại diện cho thông tin mô tả môi trường tại mỗi bước thời gian rời rạc t.
- **Phần thưởng (reward)** $r \in \mathscr{R}$: sau khi tác nhân thực hiện hành động a_t tại trạng thái s_t, môi trường trả về phần thưởng r_t và chuyển sang trạng thái tiếp theo s_{t+1}.
- **Mô hình học tăng cường** thường giả định môi trường tuân theo một Quy trình Quyết định Markov (Markov Decision Process - MDP), nơi trạng thái tiếp theo s_{t+1} chỉ phụ thuộc vào trạng thái hiện tại s_t và hành động a_t, tức là tuân theo giả định Markov. Quá trình tương tác được mô hình hóa thành bộ bốn (s_t, a_t, r_t, s_{t+1}).
- **Chính sách (policy)** π: xác định cách tác nhân lựa chọn hành động a_t tại mỗi trạng thái s_t. Chính sách có thể ở hai dạng:
  - **Chính sách xác định (Deterministic)** $\pi(s) = a$: với mỗi trạng thái s, chính sách luôn chọn cùng một hành động a.

[1] https://databasecamp.de/en/ml/reinforcement-learnings

– **Chính sách ngẫu nhiên (Stochastic)** $\pi(a|s)$: xác định phân phối xác suất để chọn hành động a tại trạng thái s, tức là $a \sim \pi(\cdot|s)$.



Hình 2.2: Sơ đồ hoạt động cơ bản của RL[2]

Một số ứng dụng phổ biến của học tăng cường bao gồm: Điều khiển robot và tự động hóa (Robotics), tối ưu hóa quy trình công nghiệp và tự động hóa (Automation), trò chơi điện tử (Video Games), quản lý tài nguyên (Resource Management), và tối ưu hóa quảng cáo cá nhân hóa (Personalized Advertising),...

Các phương pháp học tăng cường thường được phân loại dựa trên cách thức thuật toán học để cải thiện độ chính xác của các lựa chọn. Dưới đây là các cách tiếp cận chính:

- **Học tăng cường dựa trên giá trị (Value-based RL)** Agent chọn hành động dựa trên giá trị tối ưu của hàm giá trị ($V(s)$ hoặc $Q(s,a)$) mà không trực tiếp dựa vào chính sách π. Các thuật toán tiêu biểu bao gồm Q-Learning và SARSA. Phương pháp này phù hợp với các bài toán có không gian hành động rời rạc, ví dụ như trò chơi cờ vua hoặc bài toán mê cung.

- **Học tăng cường dựa trên chính sách (Policy-based RL)** Agent học trực tiếp chính sách $\pi(a|s)$, ánh xạ từ trạng thái sang hành động, mà không cần hàm giá trị. Các thuật toán tiêu biểu là **REINFORCE**, **TRPO** (Trust Region Policy Optimization), và **PPO** (Proximal Policy Optimization). Phương pháp này hiệu quả trong các bài toán có không gian hành động liên tục, như điều khiển robot hoặc tự động hóa tác vụ.

- **Học tăng cường dựa trên mô hình (Model-based RL)** Agent xây dựng một mô hình của môi trường (World Model) để dự đoán trạng thái tiếp theo và phần thưởng, sau đó sử dụng mô hình này để lập kế hoạch hành động. Ví dụ tiêu biểu là AlphaGo, kết hợp mô hình và tìm kiếm Monte Carlo Tree Search (MCTS). Phương pháp này phù hợp với các bài toán yêu cầu dự đoán dài hạn, như lập kế hoạch chiến lược hoặc mô phỏng.

- **Học tăng cường kết hợp giá trị và chính sách (Actor-Critic Methods)** Kết hợp ưu điểm của phương pháp dựa trên giá trị và dựa trên chính sách. Gồm hai thành phần: Actor(học chính sách π) và Critic (đánh giá giá trị V hoặc Q). Các thuật toán tiêu biểu bao gồm A2C (Advantage Actor-Critic) và A3C (Asynchronous Advantage Actor-Critic). Phương pháp này được sử dụng trong các bài toán phức tạp như chơi game hoặc điều khiển tự động.

- **Học tăng cường đa Agent (Multi-Agent RL - MARL)** Nhiều tác nhân tương tác trong cùng một môi trường, có thể hợp tác hoặc cạnh tranh. Các thuật toán tiêu biểu là QMIX và MADDPG (Multi-Agent Deep Deterministic Policy Gradient). Ứng dụng phổ biến trong điều khiển giao thông thông minh, tự động hóa hệ thống đa agent, và trò chơi đa người chơi (multi-player).

Việc lựa chọn phương pháp học tăng cường phù hợp phụ thuộc vào loại công việc mà các nhà nghiên cứu muốn thực hiện và mục tiêu của ứng dụng. Để hiện thực đề tài này, chúng tôi chọn phương thức học tăng cường dựa trên giá trị (Value-based RL).

2.2 Một số thuật toán học tăng cường dựa trên giá trị (Value-based RL)

Học tăng cường dựa trên giá trị (Value-based Reinforcement Learning) là một phương pháp trong học tăng cường, trong đó agent học hàm giá trị $V(s)$ hoặc $Q(s,a)$ để đánh giá giá trị của trạng thái hoặc cặp trạng thái-hành động. Agent chọn hành động tối ưu dựa trên giá trị cao nhất mà không trực tiếp học chính sách π. Đề tài của chúng tôi thực hiện tập trung huấn luyện mô hình học giá trị $Q(s,a)$. Một số thuật toán có thể nhắc đến như: Q-Learning, DQN (Deep Q-Network), Double Q-Learning, Dueling DQN và Double-Dueling DQN

2.2.1 Q-Learning

- *Q-Learning* là một thuật toán học tăng cường dựa trên giá trị (value-based), thuộc loại off-policy, học hàm giá trị hành động $Q(s,a)$, biểu thị phần thưởng kỳ vọng khi thực hiện hành động a trong trạng thái s và tuân theo chính sách tối ưu sau đó. Thuật toán không yêu cầu mô hình môi trường (model-free).
- Q-Learning cập nhật hàm $Q(s,a)$ dựa trên phương trình Bellman:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

  Trong đó:

  - s: Trạng thái hiện tại.
  - a: Hành động được chọn.
  - r: Phần thưởng nhận được.
  - s': Trạng thái tiếp theo.
  - a': Hành động tiếp theo.
  - α: Tỷ lệ học (learning rate), $0 < \alpha \leq 1$.
  - γ: Hệ số chiết khấu (discount factor), $0 \leq \gamma \leq 1$.

  Agent chọn hành động bằng cách sử dụng chiến lược như ϵ-greedy để cân bằng giữa khai thác (exploitation) và khám phá (exploration).

- *Ưu điểm*
  - Đơn giản, dễ triển khai.
  - Hiệu quả trong các bài toán có không gian trạng thái và hành động rời rạc.
  - Không cần mô hình môi trường (world-model).
- *Nhược điểm*
  - Không hiệu quả với không gian trạng thái hoặc hành động lớn do yêu cầu lưu trữ bảng Q.
  - Có thể gặp vấn đề định giá quá cao (overestimation bias) trong ước lượng Q.

2.2.2 DQN (Deep Q-Network)

- *DQN* là một biến thể của Q-Learning, sử dụng mạng nơ-ron sâu (deep neural network) để xấp xỉ hàm $Q(s,a)$, thay vì lưu trữ bảng Q, giúp xử lý các không gian trạng thái lớn và phức tạp.
  - Sử dụng mạng nơ-ron để dự đoán giá trị $Q(s,a)$ cho tất cả hành động trong trạng thái s.
  - Kết hợp hai kỹ thuật chính:
    - **Experience Replay** Lưu trữ các bộ dữ liệu kinh nghiệm (s,a,r,s') trong bộ nhớ và lấy mẫu ngẫu nhiên để huấn luyện, giúp giảm tương quan giữa các mẫu dữ liệu.
    - **Target Network** Sử dụng một mạng nơ-ron thứ hai (mạng mục tiêu) để tính giá trị $Q(s',a')$, được cập nhật định kỳ từ mạng chính để ổn định học tập.
  - Hàm mất mát (loss function) được tối ưu hóa bằng gradient descent:

$$L = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta') - Q(s,a;\theta)\right)^2\right]$$

    Trong đó θ và θ' là tham số của mạng chính và mạng mục tiêu.

- *Ưu điểm*
  - Xử lý được không gian trạng thái lớn, như hình ảnh hoặc dữ liệu phức tạp.
  - Hiệu quả trong các bài toán thực tế như chơi game.

- *Nhược điểm*
  - Vẫn có thể gặp vấn đề định giá cao.
  - Yêu cầu tài nguyên tính toán lớn và thời gian huấn luyện lâu.

2.2.3 Double Q-Learning

- *Double Q-Learning* là một cải tiến của Q-Learning, nhằm khắc phục vấn đề định giá quá cao (overestimation bias) do việc sử dụng max trong công thức cập nhật của Q-Learning. Thuật toán sử dụng hai hàm Q riêng biệt để tách biệt việc chọn hành động và đánh giá giá trị.
  - Duy trì hai hàm Q: Q_A và Q_B.

  - Khi cập nhật Q_A, sử dụng Q_B để đánh giá giá trị hành động tối ưu và ngược lại:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha\left[r + \gamma Q_B(s', \arg\max_{a'} Q_A(s',a')) - Q_A(s,a)\right]$$

  - Hai hàm Q được hoán đổi ngẫu nhiên trong quá trình học.

- *Ưu điểm*
  - Giảm định giá quá cao, dẫn đến ước lượng giá trị chính xác hơn.
  - Hiệu quả trong các bài toán yêu cầu độ ổn định cao.
- *Nhược điểm*
  - Phức tạp hơn Q-Learning do cần duy trì hai bảng Q.
  - Vẫn không phù hợp với không gian trạng thái lớn.

2.2.4 Dueling DQN

- *Dueling DQN* là một cải tiến của DQN, sử dụng kiến trúc mạng nơ-ron đặc biệt để tách hàm $Q(s,a)$ thành hai phần: giá trị trạng thái $V(s)$ và lợi thế hành động $A(s,a)$. Điều này giúp cải thiện hiệu quả học tập bằng cách tập trung vào các trạng thái quan trọng.
  - Mạng nơ-ron được chia thành hai luồng:
    * Luồng giá trị: Ước lượng $V(s)$, giá trị kỳ vọng của trạng thái s.
    * Luồng lợi thế: Ước lượng $A(s,a)$, mức độ ưu thế của hành động a so với các hành động khác trong trạng thái s.
  - Hàm $Q(s,a)$ được tính bằng:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathscr{A}|}\sum_{a'} A(s,a')\right)$$

    Phần hiệu chỉnh $\frac{1}{|\mathscr{A}|}\sum_{a'} A(s,a')$ đảm bảo tính ổn định trong ước lượng.
  - Kết hợp với Experience Replay và Target Network như DQN.
- *Ưu điểm*
  - Cải thiện hiệu quả học tập bằng cách tách biệt giá trị trạng thái và lợi thế hành động.

  - Hiệu quả hơn DQN trong các bài toán có nhiều hành động tương tự nhau về giá trị.

- *Nhược điểm*
  - Vẫn có thể gặp định giá quá cao.
  - Phức tạp hơn DQN về mặt kiến trúc mạng.

2.2.5 Double-Dueling DQN

- *Double Dueling DQN* kết hợp hai cải tiến quan trọng của Q-Learning: **Double Q-Learning** (giảm định giá quá cao - overestimation bias) và **Dueling DQN** (tách biệt ước lượng giá trị trạng thái và lợi thế hành động). Thuật toán này sử dụng mạng nơ-ron sâu để xấp xỉ hàm giá trị hành động $Q(s,a)$, phù hợp với các bài toán có không gian trạng thái lớn và phức tạp.
  - **Kiến trúc Dueling** Hàm $Q(s,a)$ được tách thành hai phần:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathscr{A}|} \sum_{a'} A(s,a') \right)$$

    Trong đó:
    * $V(s)$: Hàm giá trị trạng thái, biểu thị giá trị kỳ vọng của trạng thái s.
    * $A(s,a)$: Hàm lợi thế (advantage), đo lường mức độ tốt hơn của hành động a so với các hành động khác trong trạng thái s.

    Kiến trúc này sử dụng hai luồng riêng biệt trong mạng nơ-ron (một cho $V(s)$ và một cho $A(s,a)$) để cải thiện độ chính xác trong ước lượng giá trị.
  - **Double Q-Learning** Sử dụng hai mạng nơ-ron: mạng chính (online network) để chọn hành động và mạng mục tiêu (target network) để đánh giá giá trị, giảm thiểu định giá quá cao. Công thức cập nhật:

$$Q_A(s,a) \leftarrow Q_A(s,a) + \alpha \left[r + \gamma Q_B(s', \arg\max_{a'} Q_A(s',a')) - Q_A(s,a) \right]$$

    Trong đó Q_A và Q_B là hai hàm Q riêng biệt, α là tỷ lệ học, γ là hệ số chiết khấu.
  - **Kỹ thuật bổ sung** Kết hợp với **Experience Replay** (lưu trữ và tái sử dụng kinh nghiệm) và **Target Network** (cập nhật định kỳ để ổn định học tập).

- *Ưu điểm*
  - Giảm định giá quá cao của hàm Q nhờ Double Q-Learning.
  - Cải thiện hiệu quả học tập bằng cách tách biệt giá trị trạng thái và lợi thế hành động.
  - Phù hợp với các bài toán có không gian trạng thái lớn, như xử lý hình ảnh.
- *Nhược điểm*
  - Độ phức tạp tính toán cao.

2.3 Dữ liệu kinh nghiệm ưu tiên (Prioritized Experience Replay)

2.3.1 Tổng quan

Dữ liệu kinh nghiệm ưu tiên (Prioritized Experience Replay - PER) là một kỹ thuật cải tiến từ Dữ liệu kinh nghiệm (Experience Replay) trong học tăng cường, được sử dụng trong các thuật toán như DQN. Thay vì lấy mẫu ngẫu nhiên từ bộ nhớ kinh nghiệm, PER ưu tiên các mẫu có sai số dự đoán lớn - sai số TD (Temporal Difference error - TD-error), giúp agent học nhanh hơn từ các kinh nghiệm quan trọng.

2.3.2 Cơ chế hoạt động

Experience Replay lưu trữ các bộ kinh nghiệm (s, a, r_{t+1}, s_{t+1}) trong một bộ nhớ đệm và lấy mẫu ngẫu nhiên để huấn luyện mạng nơ-ron của DQN. PER gán mức độ ưu tiên cho mỗi kinh nghiệm dựa trên mức độ "bất ngờ" hoặc sai số của TD-error, từ đó ưu tiên lấy mẫu các kinh nghiệm quan trọng hơn.

- **Ưu tiên dựa trên sai số TD** Mức độ ưu tiên của một kinh nghiệm thường được xác định bằng giá trị tuyệt đối của sai số TD:

$$\delta = r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta)$$

  Trong đó:
  - r: Phần thưởng nhận được.

  - γ: Hệ số chiết khấu (discount factor).
  - $Q(s, a; \theta)$: Giá trị hàm Q được ước lượng bởi mạng chính.
  - $Q(s', a'; \theta')$: Giá trị hàm Q được ước lượng bởi mạng mục tiêu.

  Sai số δ lớn biểu thị kinh nghiệm có giá trị học tập cao và cho thấy dự đoán hiện tại của mô hình còn sai lệch nhiều.

- **Phương pháp lấy mẫu**

  - **Lấy mẫu tỷ lệ (Proportional Prioritization)** Xác suất lấy mẫu một kinh nghiệm i được tính bằng:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

    Trong đó $p_i = |\delta_i| + \epsilon$ (với ϵ là hằng số nhỏ để tránh xác suất bằng 0), và α điều chỉnh mức độ ưu tiên (thường $\alpha \in [0, 1]$).
  - **Lấy mẫu dựa trên xếp hạng (Rank-based Prioritization)** Sắp xếp các kinh nghiệm theo $|\delta_i|$ và gán xác suất tỷ lệ nghịch với thứ hạng.

- **Bù trọng số (Importance Sampling)** Vì PER làm thay đổi phân phối lấy mẫu, cần sử dụng trọng số để bù cho sự thiên lệch (bias) trong quá trình huấn luyện:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

  Trong đó N là kích thước bộ nhớ, β điều chỉnh mức độ bù (thường tăng dần từ 0 đến 1 trong quá trình huấn luyện). Trọng số w_i được nhân vào hàm mất mát:

$$L = w_i \cdot \delta_i^2$$

- **Cập nhật độ ưu tiên** Sau mỗi lần huấn luyện, sai số TD được tính lại để cập nhật mức độ ưu tiên của các kinh nghiệm trong bộ nhớ.

Ưu điểm

- Tăng tốc độ học tập bằng cách tập trung vào các kinh nghiệm có giá trị học cao.
- Cải thiện hiệu quả sử dụng dữ liệu, đặc biệt trong các bài toán có không gian trạng thái lớn.
- Tăng độ ổn định và hiệu suất của các thuật toán như DQN.

Nhược điểm

- Tăng độ phức tạp tính toán do cần quản lý và cập nhật mức độ ưu tiên.
- Yêu cầu điều chỉnh tham số (α, β, ϵ) cẩn thận để tránh thiên lệch quá mức.
- Có thể dẫn đến việc bỏ qua các kinh nghiệm ít ưu tiên, gây mất thông tin.

2.4 Mô hình học sâu đồ thị (Graph Neural Network - GNN) sử dụng mạng đồ thị đồng dạng với đặc trưng cạnh (Graph Isomorphism Network with Edge - GINE)

Trong quá trình xây dựng mô hình kiểm thử hộp đen đề xuất, chúng tôi chọn sử dụng GNN để thực hiện trích xuất đặc trưng của đồ thị trạng thái (State Graph) của ứng dụng.

2.4.1 Tổng quan về GNN

Mạng Nơ-ron Đồ thị (Graph Neural Networks - GNN) là các mô hình học sâu được thiết kế để xử lý dữ liệu có cấu trúc đồ thị. Mô hình tuân theo lược đồ tổng hợp láng giềng (Neighborhood Aggregation), trong đó biểu diễn của một nút được cập nhật bằng cách tổng hợp đặc trưng từ các nút láng giềng. Sau h lần lặp, biểu diễn nút nắm bắt thông tin cấu trúc trong vùng láng giềng k-hop. Công thức tổng quát cho GNN tại tầng k là:

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\{h_u^{(k-1)} : u \in N(v)\}\right)\right)$$

Trong đó:

- $h_v^{(k)}$ là biểu diễn của nút
- v tại tầng k
- $N(v)$ là tập các nút láng giềng
- $\text{AGGREGATE}^{(k)}$, $\text{COMBINE}^{(k)}$ là các hàm tổng hợp và kết hợp

Đối với phân loại đồ thị, hàm `READOUT` tổng hợp các biểu diễn nút để tạo biểu diễn đồ thị:

$$h_G = \text{READOUT}\left(\{h_v^{(K)} \mid v \in G\}\right)$$

Các biến thể GNN phổ biến bao gồm Graph Convolutional Networks (GCN) và GraphSAGE. Tuy nhiên, các mô hình trên bị hạn chế trong việc phân biệt một số cấu trúc đồ thị đơn giản do sử dụng các hàm tổng hợp (aggregation) không đủ biểu diễn, như trung bình (mean) hoặc lấy tối đa (max).

2.4.2 Graph Isomorphism Network (GIN)

GIN[6] là một GNN được thiết kế để đạt được sức mạnh biểu diễn tối đa (maximally powerful), tương đương với bài kiểm tra đẳng cấu. Công thức cập nhật của GIN là:

$$h_v^{(k)} = \text{MLP}^{(k)}\left((1+\epsilon)\cdot h_v^{(k-1)} + \sum_{u \in N(v)} h_u^{(k-1)}\right)$$

Trong đó:

- $\text{MLP}^{(k)}$ là một perceptron đa tầng
- ϵ là tham số cố định hoặc có thể học (learnable)
- Tổng hợp sử dụng phép tổng (sum) để nắm bắt toàn bộ tập hợp đa tập (multi-set) của các đặc trưng láng giềng.

GIN sử dụng hàm `READOUT` kết hợp thông tin từ tất cả các tầng:

$$h_G = \text{CONCAT}\left(\text{READOUT}\left(\{h_v^{(k)} \mid v \in G\}\right) \mid k = 0, 1, \ldots, K\right)$$

Điều này cho phép GIN tận dụng thông tin cấu trúc ở các độ sâu khác nhau. Các kết quả thực nghiệm cho thấy GIN vượt trội so với GCN và GraphSAGE trên các tập dữ liệu như MUTAG (89.4±5.6%) và REDDIT-BINARY (92.4±2.5%) nhờ khả năng phân biệt các cấu trúc đồ thị phức tạp.

2.4.3 GINE: GIN với Đặc trưng Cạnh

GINE, cụ thể là GINEConv trong PyTorch Geometric, là một biến thể của GIN tích hợp đặc trưng cạnh, được đề cập trong bài báo của Hu et al.. Công thức cập nhật của GINE là:

$$h_v^{(k)} = \text{MLP}^{(k)}\left((1+\epsilon)\cdot h_v^{(k-1)} + \sum_{u \in N(v)} \text{ReLU}\left(h_u^{(k-1)} + e_{u,v}\right)\right)$$

Trong đó:

- $e_{u,v}$ là đặc trưng cạnh giữa các nút u và v

Sự bổ sung này cho phép GINE khai thác thông tin cạnh, chẳng hạn như loại liên kết trong phân tử.

So sánh giữa GIN và GINE tập trung vào khía cạnh hiệu quả nhận biết biểu diễn khác nhau của đồ thị.

- **GIN** đạt được sức mạnh biểu diễn tối đa mà không sử dụng đặc trưng cạnh, phù hợp cho các nhiệm vụ như phân loại đồ thị, nơi thông tin cấu trúc và đặc trưng nút là đủ. Tuy nhiên, GIN có thể kém hiệu quả hơn khi thông tin cạnh đóng vai trò quan trọng.
- **GINE** bằng cách tích hợp đặc trưng cạnh, có khả năng cải thiện hiệu suất trên các nhiệm vụ mà cạnh mang thông tin quan trọng.

2.5 Tính độ phủ (Coverage) trong kiểm thử hộp đen

2.5.1 Tổng quan

Tính độ phủ (Coverage) trong kiểm thử phần mềm nói chung là một thước đo định lượng mức độ mà mã nguồn, yêu cầu, hoặc các thành phần khác của hệ thống được kiểm tra bởi các bộ kiểm thử (test cases). Trong đề tài này, chúng tôi xem độ phủ như thước đo độ hiệu quả của mô hình kiểm thử hộp đen. Một số công cụ kiểm thử phổ biến như: JaCoCo, emma, ELLA,.. Tuy nhiên, các công cụ hiện tại chỉ có thể đo độ phủ ở dạng hộp trắng (white-box) do cần mã nguồn của ứng dụng để có thể chèn vào hoặc yêu cầu phải viết một công cụ chèn mã (instrumenter) để có thể chèn mã đo độ phủ vào các tệp apk với tỉ lệ thất bại tương đối cao.

2.5.2 Các loại độ phủ được sử dụng trong đồ án này

- **Độ phủ câu lệnh (Statement/Instruction Coverage)** Đo tỷ lệ các câu lệnh trong mã nguồn được thực thi ít nhất một lần.
- **Độ phủ màn hình (Activity Coverage)** Đo tỷ lệ các activity trong ứng dụng được xuất hiện ít nhất một lần.

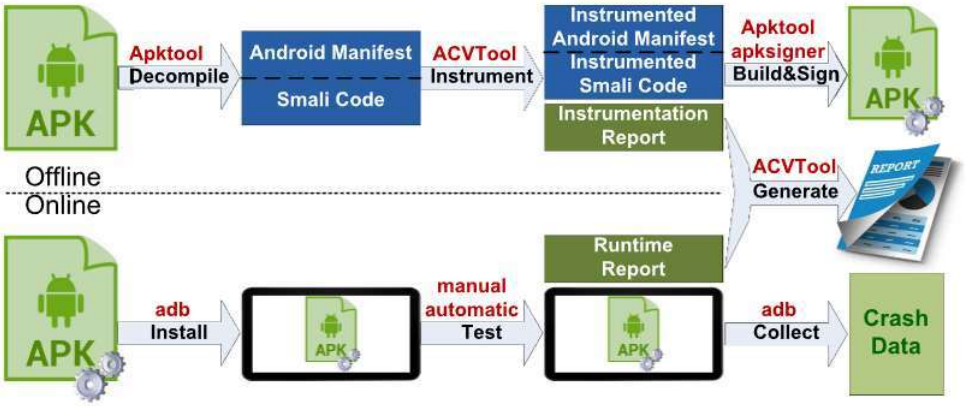
2.6 Công cụ ACVTool (Android Code coverage Tool)

ACVTool là một công cụ đo độ phủ mã nguồn (code coverage) trong kiểm thử hộp đen tự động cho ứng dụng Android, không yêu cầu mã nguồn gốc. Công cụ này chèn các probe vào mã bytecode Dalvik ở định dạng smali để theo dõi độ phủ ở các mức độ: lớp (class), phương thức (method) và câu lệnh (instruction). ACVTool được thiết kế để tích hợp dễ dàng vào các khung kiểm thử và phân tích động, với tỷ lệ chèn mã thành công cao.

ACVTool giải quyết vấn đề thiếu các công cụ đáng tin cậy để đo độ phủ mã nguồn chi tiết trong kiểm thử hộp đen Android. Không giống các công cụ khác như ELLA, Huang et al., hoặc BBoxTester với tỷ lệ chèn mã thấp (36%-65%), ACVTool đạt tỷ lệ chèn mã thành công 97.8% và tỷ lệ ứng dụng thực thi được sau chèn mã là 96.9% trên tập dữ liệu 1,278 ứng dụng từ Google Play và F-Droid.

2.6.1 Cách hoạt động

ACVTool hoạt động qua ba giai đoạn chính :



Hình 2.3: Luồng hoạt động - ACVTool[3]

- **Giai đoạn ngoại tuyến (Offline Phase)** Ứng dụng Android (file .apk) được giải mã bằng apktool, chuyển đổi mã bytecode Dalvik thành định dạng smali. ACVTool chèn các probe đặc biệt sau mỗi câu lệnh smali để theo dõi thực thi. Sau đó, ứng dụng được biên dịch lại và ký lại bằng apksigner và zipalign.
- **Giai đoạn trực tuyến (Online Phase)** Ứng dụng đã chèn mã được cài đặt trên thiết bị hoặc trình giả lập Android thông qua adb. Lớp Instrumentation được kích hoạt để thu thập thông tin thời gian chạy (độ phủ và lỗi ứng dụng).

- **Giai đoạn tạo báo cáo (Report Generation Phase)** Thông tin thời gian chạy được trích xuất và tạo thành báo cáo định dạng HTML hoặc XML, tương tự báo cáo JaCoCo, với các câu lệnh smali được thực thi được đánh dấu.

2.6.2 Đặc điểm kỹ thuật

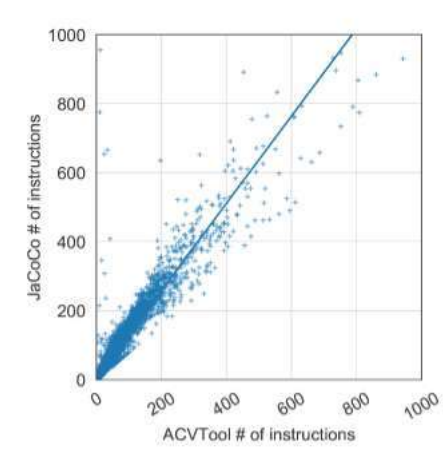
- **Chèn mã bytecode** Sử dụng apkil (thư viện smali parser) để chèn probe trực tiếp sau các câu lệnh smali, đảm bảo mã vẫn hợp lệ với VerifyChecker của Android Runtime. ACVTool xử lý các trường hợp phức tạp như khối synchronized và quản lý thanh ghi (register) để tránh xung đột.
- **Hỗ trợ đa nền tảng** Tương thích với mọi trình giả lập và thiết bị Android, không yêu cầu cài đặt phần mềm bổ sung.
- **Báo cáo chi tiết** Cung cấp độ phủ ở ba mức (lớp, phương thức, câu lệnh) và thông tin về lỗi ứng dụng (crash).

2.6.3 Ưu điểm

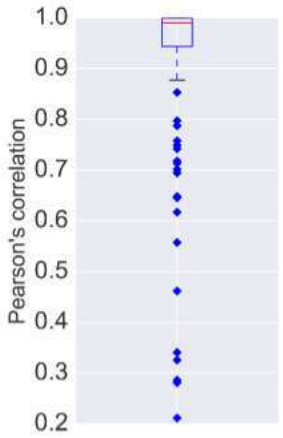
- Tỷ lệ chèn mã thành công cao (97.8%) và thực thi được (96.9%) trên tập dữ liệu lớn (1,278 ứng dụng), vượt trội so với các công cụ khác (ELLA: 91.1%, Huang et al.: 36%, BBoxTester: 65%).
- Độ phủ đo được tương thích cao với JaCoCo (tương quan Pearson ≥ 0.94 cho hơn 75% ứng dụng) và ELLA ở mức phương thức.
- Tích hợp dễ dàng với các công cụ kiểm thử như Sapienz, Monkey, hỗ trợ tìm lỗi hiệu quả hơn.

2.6.4 Nhược điểm

- Ký lại ứng dụng (re-signing) có thể gây lỗi với các API yêu cầu chữ ký gốc, như Google Maps API.
- Một số câu lệnh smali không thể theo dõi (untraceable), dẫn đến sai lệch nhỏ so với JaCoCo **Hình 2.4**.



(a) Scatterplot of the number of instructions in app methods, as computed by ACVTool and JaCoCo.

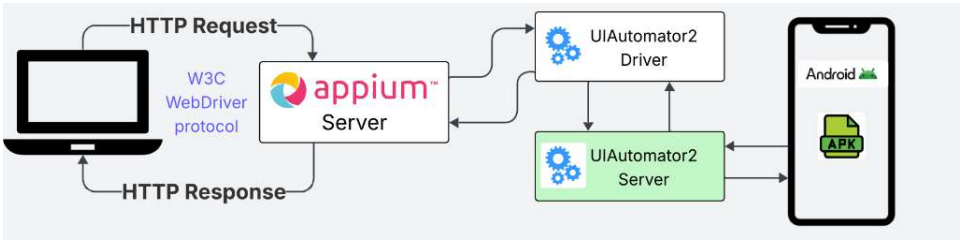


(b) Boxplot of correlation for code coverage data computed by ACVTool and JaCoCo.

Hình 2.4: Độ tương quan JaCoCo - ACVTool[4]

2.7 Công cụ Appium

Appium là một công cụ mã nguồn mở, đa nền tảng, được thiết kế để tự động hóa kiểm thử giao diện người dùng (UI) cho các ứng dụng di động trên Android, iOS , trình giả lập (emulator) và Windows.



Hình 2.5: Luồng hoạt động của Appium

Appium hoạt động theo mô hình client-server, sử dụng giao thức WebDriver (W3C WebDriver specification):

- **Máy chủ (Server)** Máy chủ Appium, được xây dựng bằng Node.js, nhận lệnh từ client, chuyển đổi thành các hành động cụ thể cho nền tảng đích thông qua giao diện REST API.

[4]https://dl.acm.org/doi/10.1145/3243734.3278484

- **Khung tự động hóa** Tùy thuộc vào nền tảng, Appium sử dụng các khung sau:
  - **iOS** WebDriverAgent (dựa trên XCUITest).
  - **Android** UiAutomator2, UiAutomator hoặc Espresso.
  - **Windows** WinAppDriver (dựa trên Windows UI Automation API).
- **Thiết bị/Emulator** Khung tự động hóa tương tác với hệ điều hành di động và giao diện người dùng của ứng dụng để thực thi lệnh. Kết quả được gửi ngược lại qua khung tự động hóa đến máy chủ Appium và cuối cùng đến client.

Kiến trúc này đảm bảo Appium có thể hỗ trợ nhiều nền tảng với API thống nhất, giúp kiểm thử đa nền tảng trở nên liền mạch.

2.8 Hệ điều hành Android

Android là một hệ điều hành mã nguồn mở được xây dựng trên nền tảng nhân Linux, được thiết kế tối ưu cho các thiết bị di động có màn hình cảm ứng như điện thoại thông minh và máy tính bảng. Hệ điều hành này ban đầu được phát triển bởi công ty Android Inc., với sự tài trợ từ Google, và sau đó được Google mua lại vào năm 2005.

Hình 2.6: Biểu tượng hệ điều hành Android

Android chính thức ra mắt vào năm 2007 cùng với sự thành lập của Liên minh Thiết bị Di động Mở (Open Handset Alliance) - một liên minh giữa các công ty phần mềm, phần cứng và viễn thông nhằm thúc đẩy các tiêu chuẩn mở cho thiết bị di động. Chiếc điện thoại đầu tiên chạy hệ điều hành Android được thương mại hóa vào năm 2008.

Với mã nguồn được phát hành theo Giấy phép Apache, Android cho phép các nhà sản xuất thiết bị, nhà mạng và lập trình viên tự do tùy chỉnh và phân phối lại hệ điều hành này. Tính chất mã nguồn mở cùng với giấy phép linh hoạt đã tạo điều

kiện cho một cộng đồng phát triển mạnh mẽ, nơi các lập trình viên có thể mở rộng chức năng thiết bị thông qua các ứng dụng viết chủ yếu bằng ngôn ngữ Java (đã được điều chỉnh phù hợp với Android).

Chương 3

PHƯƠNG PHÁP THỰC HIỆN

3.1 Tóm tắt chương

Ở chương này, chúng tôi sẽ trình bày chi tiết về phần động lực, phương pháp thực hiện và phân tích thiết kế mô hình học tăng cường trong kiểm thử hộp đen cho ứng dụng Android đề xuất

3.2 Động lực

Mục tiêu của đồ án là xây dựng và thực nghiệm một mô hình kiểm thử hộp đen tự động cho ứng dụng Android, dựa trên hướng tiếp cận học tăng cường. Đề tài hướng đến việc phân tích, so sánh hiệu quả của mô hình đề xuất với các phương pháp kiểm thử hộp đen hiện có trong các nghiên cứu trước, đồng thời đánh giá những ưu điểm cũng như hạn chế của các phương pháp này. Trên cơ sở đó, xây dựng mô hình đề xuất nhằm khắc phục một phần các hạn chế đã được nhận diện.

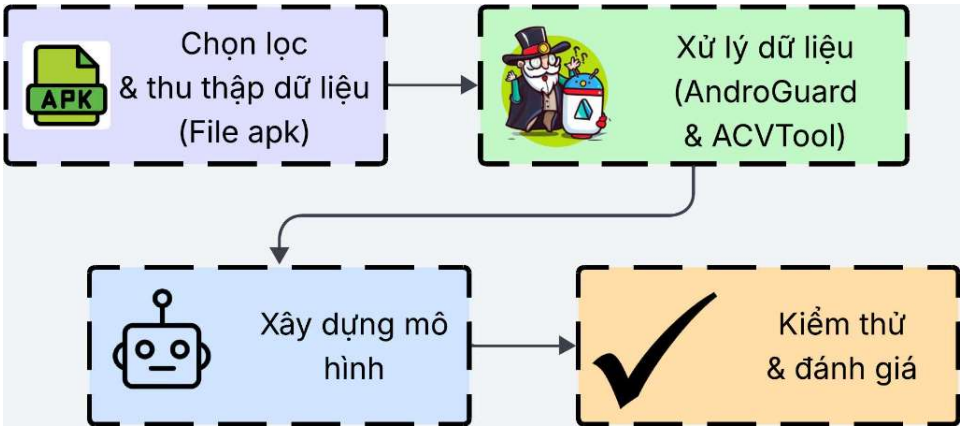
Để hiện thực hóa mục tiêu trên, đề tài tập trung giải quyết các câu hỏi nghiên cứu sau:

- **Câu hỏi 1:** Độ phủ của mô hình đề xuất là bao nhiêu so với các mô hình kiểm thử hiện có? Phương pháp tính độ phủ như thế nào?
- **Câu hỏi 2:** Những cải tiến của mô hình đề xuất so với những mô hình hiện có và ảnh hưởng lên hiệu quả, kết quả của mô hình đề xuất?
- **Câu hỏi 3:** Khả năng phát hiện lỗi của mô hình đề xuất, so sánh với các mô hình hiện có? Có bao nhiêu lỗi là độc nhất, tính xác thực của lỗi tìm được?

3.3 Thiết kế thí nghiệm

Để giải quyết các câu hỏi nghiên cứu được đặt ra ở bên trên, nhóm tiến hành thí nghiệm kiểm thử hộp đen ứng dụng **Hình 3.1** bao gồm 4 bước cụ thể như sau:

- (1) Chọn lọc và thu thập dữ liệu;
- (2) Xử lý dữ liệu;
- (3) Xây dựng mô hình đề xuất;
- (4) Kiểm thử và đánh giá mô hình.



Hình 3.1: Trình tự thực hiện thí nghiệm

3.4 Thu thập dữ liệu

Để có thể thực hiện việc kiểm thử, nhóm cần thu thập một số ứng dụng Android, chủ yêu có mã nguồn mở. Nhóm chọn sử dụng một số ứng dụng đã được sử dụng trong tập dữ liệu của mô hình DQT và một số ứng dụng có sẵn tại F-Droid.

Tiêu chí chọn lọc

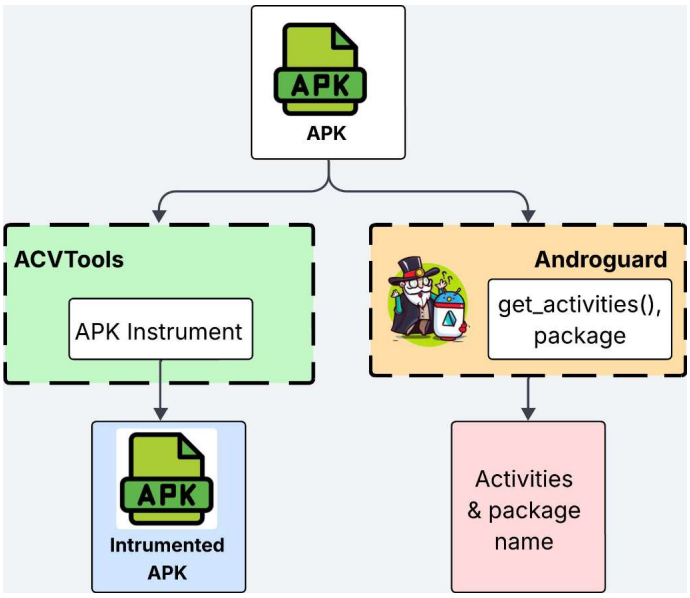
- Dữ liệu là ứng dụng Android (tệp apk)
- Dữ liệu có khả năng chạy trên môi trường hệ điều hành Android 7.0 (Nougat) - API 24. Tránh sử dụng các ứng dụng có phiên bản quá lỗi thời, không còn được sử dụng.

3.5 Xử lý dữ liệu

Chúng tôi sử dụng công cụ AndroGuard[1], dựa trên tệp manifest của các file apk để trích xuất các dữ liệu sau: **Từ phần android manifest trích xuất được các tính năng như:**

- **Activitiy:** Các màn hình (activity) của ứng dụng.
- **Package name:** Tên gói của ứng dụng.

Sau đó các file apk của ứng dụng sẽ được chèn thêm mã (instrument) sử dụng công cụ ACVTool nhằm mục đích thu thập độ phủ của thí nghiệm trong quá trình kiểm thử **Hình 3.2**.



Hình 3.2: Sơ đồ xử lý apk

3.6 Xây dựng mô hình đề xuất

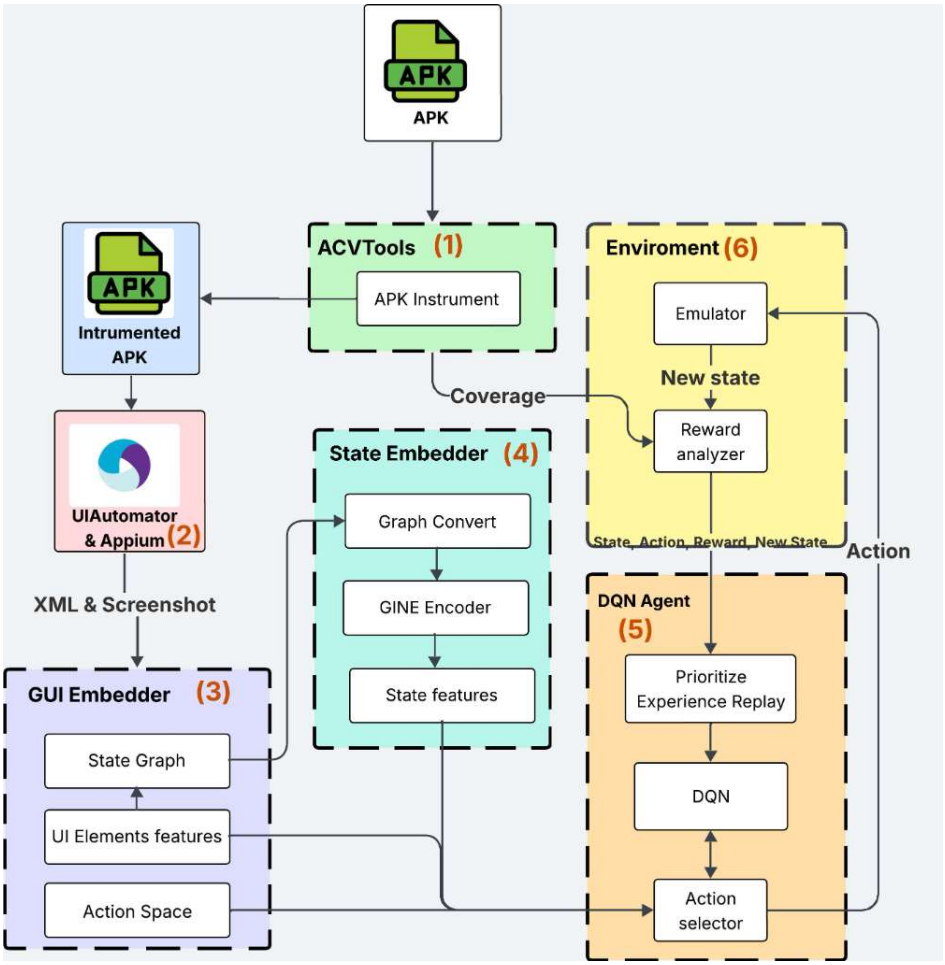
Chúng tôi hiện thực hóa mô hình kiểm thử đề xuất bằng cách sử dụng các thư viện và công cụ chính như sau:

- (1) **Máy ảo (Android Studio Emulator - emulator)** làm môi trường kiểm thử ứng dụng.

[1]https://github.com/androguard/androguard

- (2) **Appium và UIAutomator** để thực hiện hành động tương tác và trích xuất thông tin từ emulator.
- (3) **ADB** để thực hiện các câu lệnh trên emulator và ứng dụng .
- (4) **PyTorch** dùng để xây dựng mô hình học sâu.
- (5) **ACVTool** Để đo độ phủ, chúng tôi sử dụng ACVTool để đo ở mức độ dònglệnh (instruction level).
- (6) **SLM** Để sinh ra đầu vào cho các trường nhập liệu, nhằm tìm kiếm lỗi xử lý đầu vào của ứng dụng.

Các thành phần chính của mô hình như sau: Trong đó:



Hình 3.3: Thành phần chính của mô hình đề xuất

- (1) **GUI Embedder:** Có nhiệm vụ nhận vào cây UI (UI hierarchy) dưới dạng XML và ảnh chụp màn hình tại mỗi bước kiểm thử. Sau đó thực hiện phân

tích và tổng hợp ra không gian hành đông $\mathscr{A}$, đồ thị trangh thái $Graph(s)$ đa hướng và đặc trưng của từng phần tử UI (UI elements);

- (2) **State Embedder:** Có nhiệm vụ nhận vào một đồ thị trạng thái đa hướng $Graph(s)$ và chuyển đổi nó thành định dạng có thể học được - torch Data. Sau đó thực hiện đưa vào mô hình GINE Encoder để trích xuất đặc trưng của đồ thị.
- (3) **DQN Agent:** Có nhiệm vụ lựa chọn hành động tối ưu $a \in \mathscr{A}$.
- (4) **Enviroment** Môi trường, có nhiệm vụ làm môi trường chạy ứng dụng kiểm thử.

3.7 Kiểm thử và đánh giá mô hình

Chúng tôi thực nghiệm và so sánh với 2 mô hình là DQT và Monkey nhằm xác định độ hiệu quả của mô hình đề xuất. Việc so sánh cũng giúp chúng tôi tìm hiểu được các ưu, nhược điểm của các mô hình. Để thực nghiệm, chúng tôi cho mỗi mô hình chạy kiểm thử trong 2 giờ trên mỗi ứng dụng. Sau đó, chúng tôi thực hiện đánh giá các mô hình dựa trên độ phủ (coverage) của activity, độ phủ của dòng lệnh (instruction) và số lỗi tìm được trong quá trình kiểm thử.

Chương 4

THỰC NGHIỆM, ĐÁNH GIÁ VÀ THẢO LUẬN

Tóm tắt chương

Trong chương này, chúng tôi sẽ trình bày chi tiết cách hiện thực hóa phương pháp đã trình bày trong **Chương 3**, bao gồm cách xây dựng môi trường thực nghiệm, xây dựng mô hình kiểm thử hộp đen đề xuất, chạy kiểm thử và các kết quả đạt được khi thực nghiệm công cụ.

4.1 Môi trường thực nghiệm

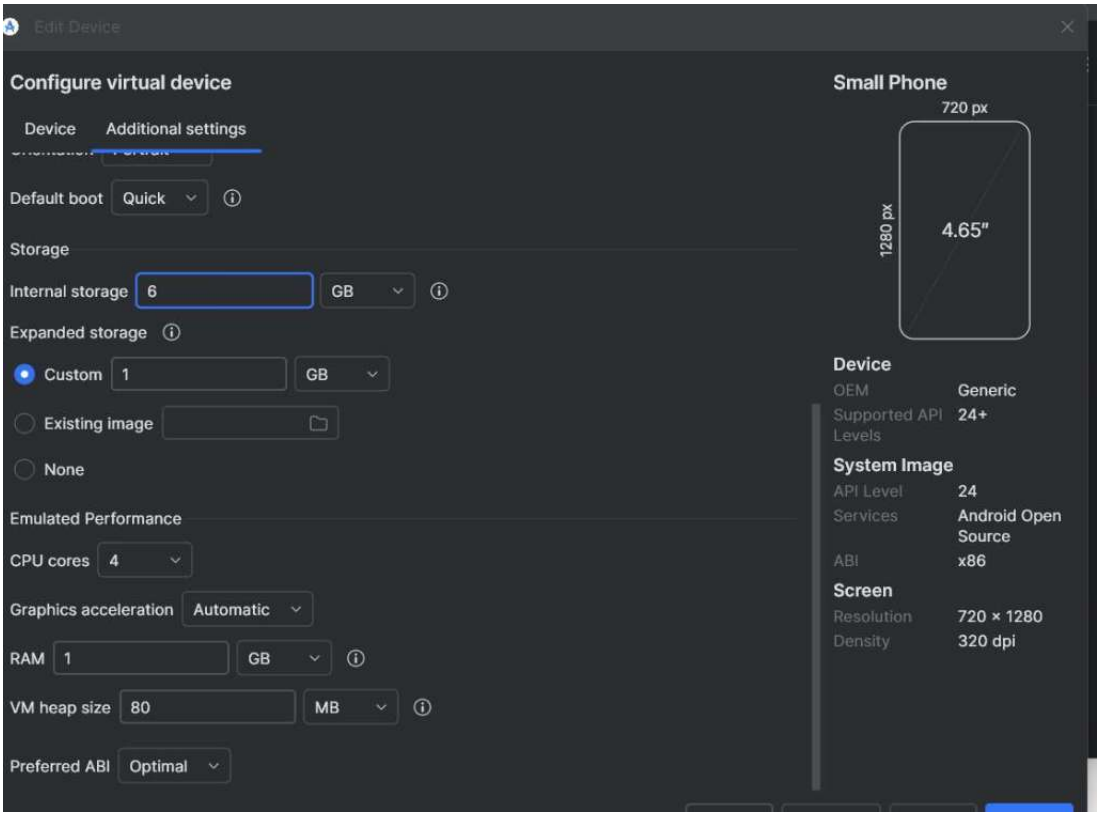
- **Máy local 1**
  - CPU: AMD® Core™ R7-8745H (8 cores - 3.8 GHz)
  - GPU: NVIDIA® GeForce RTX 4060 Laptop - VRAM 8GB - CUDA 12.6
  - RAM: 16GB
  - OS: Windows 11
  - Dung lượng ổ đĩa: 512GB

  Máy local 1 được dùng để xây dựng và chạy thí nghiệm của mô hình đề xuất
- **Máy local 2**
  - CPU: Intel® Core™ i7-11800H (8 cores - 2.3 GHz)
  - GPU: NVIDIA® GeForce RTX 3050 Laptop - VRAM: 4GB - CUDA 11.4

  - RAM: 16GB
  - OS: Windows 11/Ubuntu 24.04LTS
  - Dung lượng ổ đĩa: 512GB

  Máy local 2 được dùng để chạy thí nghiệm của các mô hình hiện có

- **Emulator Android - emulator**
  - OS: Android 7.0 (Nougat) x86 - API 24
  - RAM: 1GB
  - Dung lượng bộ nhớ trong: 6GB
  - Dung lượng bộ nhớ SD; 1GB



Hình 4.1: Cấu hình emulator Android

  emulator Android được dùng làm môi trường để thực hiện kiểm thử, bộ nhớ SD dùng để lưu lại các file coverage

- **Môi trường phát triển**
  - Trình soạn thảo: VSCode.

- Ngôn ngữ lập trình: Python3.12.10 (máy local 1), Python3.10 (máy local 2), Python2.7 (máy local 2).
- Thư viện và công cụ sử dụng chính:

Library / Tool	**Version**
Appium-Python-Client	5.0.0
Appium	2.19.0
ACVTool	2.3.4
jdk	17.0.12
lxml	5.3.2
networkx	3.4.2
numpy	2.1.2
node	22.14.0
pytorch	2.6.0+cu126 (local 1), 1.10.0+cu11.3 (local 2)
transformers	4.51.3

Bảng 4.1: Các thư viện và công cụ chính

4.2 Xây dựng tập dữ liệu

Tập dữ liệu được sử dụng gồm 7 ứng dụng trong đó có 6 ứng dụng có phiên bản cũ (trong tập dữ liệu của mô hình DQT[1]) và 1 ứng dụng có phiên bản mới nhất trên F-Droid **Bảng 4.2**

App name	**App version**	**Min version SDK**
AntennaPod	3.8.0	21
Slideshow Wallpaper	1.2.2	18
NewPipe APK	0.27.7	21
Seal for Android	1.13.1	21
MyExpenses	3.4.5	21
Notes	1.4.1	24
SpotiFlyer for Android	3.6.4 (latest)	24

Bảng 4.2: Danh sách ứng dụng trong tập dữ liệu, phiên bản và SDK tương ứng

[1]https://github.com/Yuanhong-Lan/AndroTest24

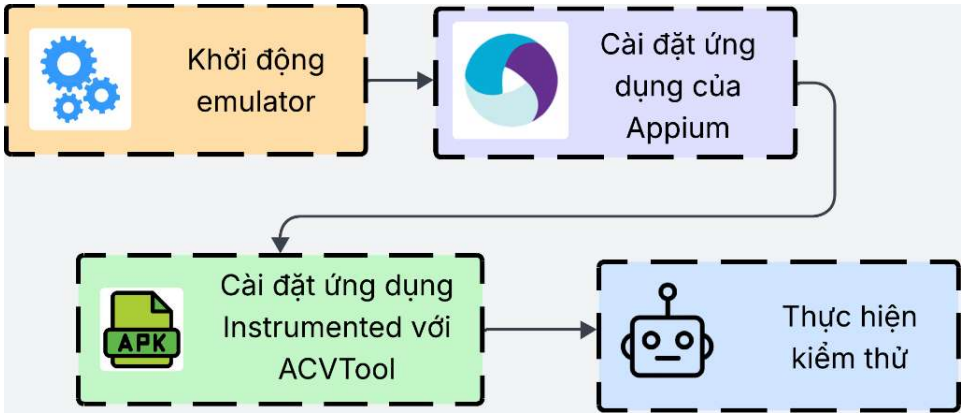
Tập dữ liệu sau đó được trích xuất thông tin từ tệp manifest của tệp apk bằng công cụ AndroGuard để lấy các màn hình hoạt động (activity) nhằm để tính độ phủ của hoạt động và tên gói (package name) của ứng dụng. Sau đó chúng tôi sử dụng công cụ ACVTool để thực hiện chèn mã (instrument) vào tệp apk nhằm thu thập chỉ số độ phủ trong quá trình thực hiện kiểm thử.

4.3 Hiện thực xây dựng mô hình đề xuất

Chúng tôi thực hiện xây dựng mô hình đề xuất, dựa trên ý tưởng chính là mô hình của DQT, cụ thể mô hình sẽ được cấu thành từ các thành phần chính như sau: Enviroment, GUI Embbeder, State Embedder, Reward Analyzer, DQN Agent. Sau đây chúng tôi sẽ trình bày chi tiết cấu trúc, chi tiết của từng thành phần.

4.3.1 Enviroment - Môi trường:

Chúng tôi chọn môi trường chạy kiểm thử là máy ảo Android (Android Emulator - emulator) của Android Studio với phiên bản hệ điều hành của emulator là Android 7.0 (Nougat) với API 24 nhằm tương thích với nhiều ứng dụng nhất có thể. Để mô hình kiểm thử có thể tương tác được với môi trường emulator, chúng tôi chọn sử dụng Appium với UIAutomator Driver để có thể tương tác trực tiếp qua giao diện như một người dùng bình thường, cũng như ADB (Adroid Debug Bridge) để có thể tương tác với emulator qua dòng lệnh.



Hình 4.2: Luồng hoạt động khởi tạo môi trường

Các bước khởi tạo:

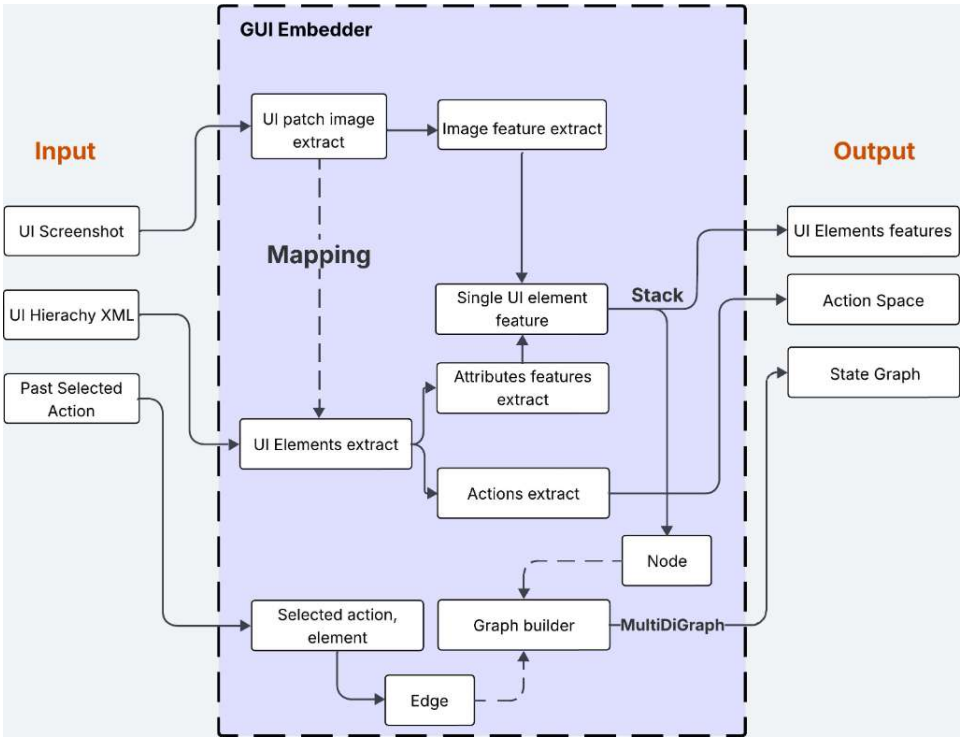
- **Bước 1: Khởi động emulator:** Thực hiện khởi chạy emulator android ở chế độ tạo mới hoàn toàn (-no-snapshot-save, -wipe-data), việc không lưu trữ dữ liệu nhằm tránh xung đột giữa các ứng dụng đã cài đặt và kiểm thử trước đó, giúp các lần kiểm thử được độc lập và đồng nhất.
- **Bước 2: Cài đặt các ứng dụng của Appium và thực hiện kết nối đến Appium Server:** Appium cần thiết lập kết nối từ emulator Android đến máy local thông qua url để có thể gửi các hành động, tương tác đến emulator.
- **Bước 3: Cài đặt ứng dụng cần kiểm thử:** Ứng dụng sau khi được chèn mã sẽ được cài đặt vào emulator và sẵn sàng để thực hiện kiểm thử
- **Bước 4: Thực hiện tương tác:** Việc thực hiện tương tác với ứng dụng thông qua giao diện người dùng được thực hiện thông qua W3C của UIAutomator. Các hành động sẽ được định sẵn trước một số tham số như thời gian chạm, khoảng cách vuốt. Khi thực hiện hành động nhập liệu sẽ sử dụng mô hình SLM để sinh dữ liệu nhập.

Listing 4.1: Mã giả của thực hiện hành động trên emulator

```
- If Click:
    Try executing a click gesture on the element or coordinates
    If failed, use low-level W3C touch tap
  - If Long Click:
    Perform a long click gesture (duration: 1000ms)
  - If Double Click:
    Perform a double click gesture
  - If Text Input (edit text or number):
    Tap or click the element to focus
    Wait for input field to become active
    Generate input suggestions using a SLM text generator
    Input each suggestion into the focused element
  - If Scroll:
    Determine scroll direction (up/down/left/right)
    Perform swipe gesture in the specified direction
  - If Rotate:
    Set screen orientation to LANDSCAPE or PORTRAIT
  - If Volume Control:
    Send keycode for volume up or down
  - If Back Navigation:
    Press the back button using keycode 4
```

4.3.2 GUI Embbedder - Trích xuất dữ liệu từ UI:

Thành phần GUI Embedder có nhiệm vụ chính là trích xuất, xử lý và chuẩn hóa thông tin từ giao diện người dùng (UI) của ứng dụng Android tại thời điểm kiểm thử. Thành phần này giúp chuyển đổi giao diện động thành các đặc trưng (feature vector) mà mô hình học máy có thể hiểu được, đồng thời xây dựng một đồ thị biểu diễn mối quan hệ giữa các thành phần UI, hỗ trợ cho quá trình học và chọn hành động của DQN Agent.



Hình 4.3: Luồng hoạt động của GUI Embedder

Cụ thể, luồng hoạt động của GUI Embedder bao gồm các bước như sau:

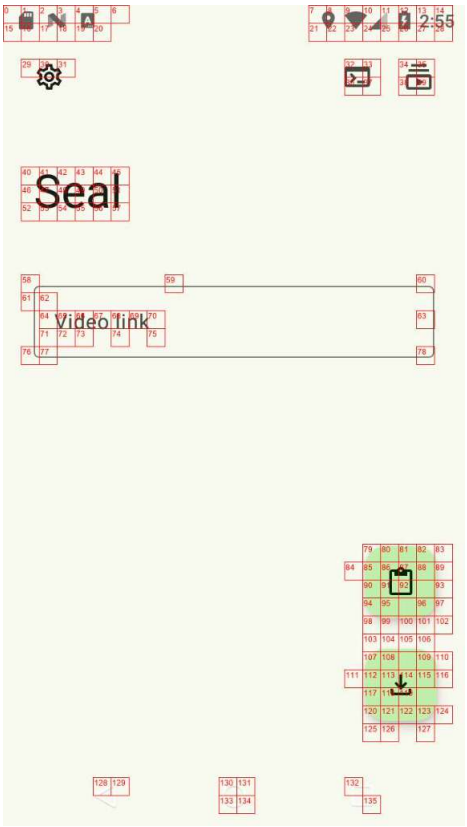
- **Bước 1: Phân tích cây UI (XML hierarchy)** Sử dụng Appium để trích xuất cấu trúc giao diện người dùng dưới dạng XML. Sau đó sử dụng thư viện lxml để chọn lấy các phần tử có thể tương tác được như: nhấn (clickable), nhấn giữ (long-clickable), đánh dấu (checkable), lướt (scrollable), hoặc là các trường nhập liệu như (EditText)

- **Bước 2: Trích xuất đặc trưng với mỗi phần tử UI được tìm thấy:**
  - Thực hiện trích xuất thông tin tọa độ từ thuộc tính bounds, sau đó chuẩn hóa tọa độ theo kích thước màn hình tạo ra vector đặc trưng có độ dài là 6 chiều.
  - Các trạng thái như enabled, checked, và password cũng được chuyển thành vector nhị phân.
  - Nội dung (text) và mô tả nội dung (content-desc) của phần tử được đưa vào mô hình sentence-transformers/all-MiniLM-L6-v2[2] để sinh vector đặc trưng. Mỗi phần tử có vector kết hợp 48 chiều gồm: 32 chiều từ text và 16 chiều từ content-desc.

[2]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

– Thực hiện ánh xạ các khả năng tương tác của phần tử thành một vector hành động gồm 13 chiều hành động có thể thực hiện (nhấn -click, nhấn giữ - long-click, edit - chỉnh nội dung (số, chữ), scroll - lướt (ở 4 hướng trái, phải, trên, dưới), rotate - xoay màn hình (dọc, ngang), volume - âm lượng (tăng, giảm), back - quay lại).

– Khác biệt với mô hình DQT, khả năng thực hiện chụp màn hình (screenshot) và chia ảnh thành các mảnh nhỏ (patches) **Hình 4.4** theo kích thước cố định (28x28 pixel). Các mảnh ảnh nằm trọn trong phạm vi (bounds) của phần tử UI được chọn, ghép thành một tensor hình ảnh duy nhất. Tensor này sau đó được đưa vào mô hình CNN (AndroidScreenEncoder) để sinh đặc trưng thị giác (64 chiều). Việc thêm hình ảnh vào một phần của đặc trưng giúp tăng khả năng nhận diện của mô hình đề xuất.



Hình 4.4: Demo của thực hiện xác định các mảnh patch

Listing 4.2: Mã giả tạo các mảnh ảnh từ ảnh chụp màn hình

```
extract_ui_patches(screenshot, patch_size)
    Get image width W and height H
    Calculate number of patch steps w_steps and
        h_steps from patch_size
    Convert image to array (arr)
    Initialize empty graph
    Initialize patch_rgbs as empty map
    Set index idx to 0

    For each patch row i in h_steps:
        For each patch column j in w_steps:
            Extract patch from image at position
                (i, j)
            Compute mean RGB of patch
            Store patch coordinates and mean RGB in
                patch_rgbs[idx]
            Add node idx to graph
            Increment idx

    For each node_id and corresponding rgb in
        patch_rgbs:
        Get neighbor to the right and neighbor below
        If neighbor exists and has same RGB:
            Add edge between node_id and neighbor

    Initialize reps as empty list contains
        coordinate of patches that corresponding in
        rgb color
    For each connected component in graph:
        Pick the first node in component
        Append its coordinates to reps

    Return reps
End
```

– Sau đó phần tử UI được mã hóa thành một vector đặc trưng đầy đủ bao gồm: Mã hash dưới dạng số (nhận diện phần tử), vector hành động (13 chiều), vector trạng thái (3 chiều), loại input (3 chiều - không có input, chữ và số), vị trí đã chuẩn hóa (6 chiều), đặc trưng văn bản (48

chiều), đặc trưng hình ảnh (64 chiều). Tổng cộng vector đặc trưng của mỗi phần tử là 138 chiều.

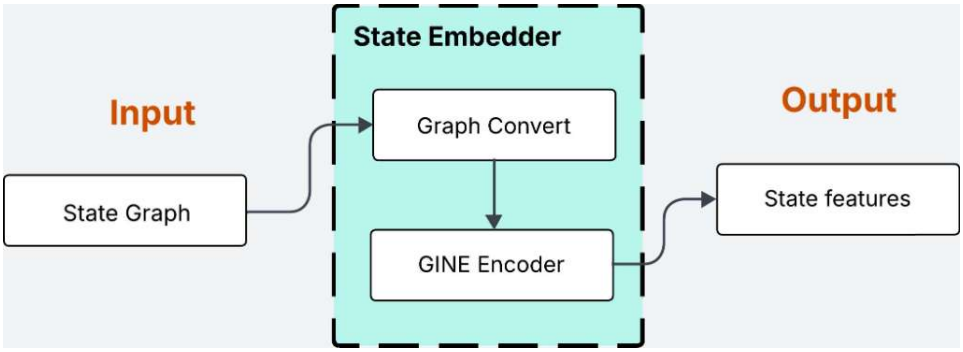
– Thực hiện tổng hợp toàn bộ các tương tác khả thi của các phần tử trong màn hình hiện tại để có thể đưa vào cho mô hình DQN lựa chọn.

- **Bước 3: Tạo đồ thị tương tác (UI State Graph):** Thực hiện xây dựng một đồ thị có hướng biểu diễn mối quan hệ giữa mỗi màn hình (activity) và các phần tử UI. Mỗi đỉnh (node) tương ứng với một phần tử UI chứa toàn bộ các tương tác có thể thực hiện ở trạng thái đó hoặc màn hình (activity), mỗi cạnh (edge) chứa thông tin hành động và trạng thái từ phần tử UI trước tới phần tử UI hiện tại.

Thành phần GUI Embedder là cầu nối quan trọng giữa UI và mô hình DQN bằng cách chuyển hóa UI thành các đặc trưng mà mô hình học máy có thể hiểu được và đưa ra lựa chọn. Đồng thời tạo dựng lịch sử các trạng thái tương tác dưới dạng đồ thị để có thể giúp mô hình đưa ra những lựa chọn tốt hơn.

4.3.3 State Embbedder - Trích xuất dữ liệu từ trạng thái UI:

Mục tiêu của State Embedder là tạo ra một vector trạng thái có kích thước cố định 96 chiều, đại diện cho toàn bộ giao diện và lịch sử tương tác tại thời điểm kiểm thử. Vector này sẽ được sử dụng làm đầu vào cho DQN Agent để học và đưa ra quyết định cho hành động tiếp theo.



Hình 4.5: Luồng hoạt động của State Embedder

Quy trình xử lý chính của State Embedder bao gồm các bước sau:

- **Bước 1: Chuyển đổi đồ thị trạng thái sang tensor:** Đồ thị MultiDiGraph của NetworkX (bao gồm các nút là phần tử UI và activity, và các cạnh là mối quan hệ giữa chúng) được chuyển đổi sang định dạng Data của PyTorch Geometric. Với:
  - Mỗi nút chứa vector đặc trưng đầu vào (kích thước 138 chiều) được tạo từ GUI Embedder.
  - Mỗi cạnh chứa thông tin gồm:
    * Loại quan hệ (gồm: child_of_activity, previous_state_of_element, previous_state_of_activity)
    * Vector hành động đã thực hiện (13 chiều)
    * Vector không gian hành động đã chọn để thực hiện (138 chiều).

    Tất cả thông tin trên được nối lại tạo thành edge_attr (167 chiều) dùng làm đặc trưng cạnh trong mô hình GNN.
- **Bước 2: Trích xuất vector trạng thái:** Sử dụng GINE (Graph Isomorphism Network-GIN với Edge features)là mở rộng từ GIN để hỗ trợ đặc trưng cạnh (edge attributes) hiệu quả trong trường hợp có nhiều loại cạnh và hành vi liên kết.
  - Thông tin sau đo sẽ được đưa qua các lớp MLP và các lớp GINEConv để thực hiện trích xuất thông tin
  - Sau đó, một hàm pooling toàn cục (global mean) được áp dụng để tổng hợp thông tin của toàn bộ đồ thị thành một vector duy nhất – chính là vector trạng thái (state vector).
  - Vector trạng thái đầu ra có kích thước cố định (96 chiều), là biểu diễn học được của trạng thái giao diện hiện tại.

Reward Analyzer – Cơ chế đánh giá phần thưởng trong học tăng cường

Trong bối cảnh kiểm thử phần mềm sử dụng học tăng cường, hàm phần thưởng đóng vai trò tối quan trọng trong việc hướng dẫn agent học hành vi khám phá hiệu quả. Thành phần Reward Analyzer được thiết kế nhằm tính toán phần thưởng một cách linh hoạt và có định hướng, phản ánh các tiêu chí khám phá, tò mò (curiosity-driven) trong quá trình kiểm thử ứng dụng Android.

Listing 4.3: Mã giả hàm tính toán phần thưởng của mô hình

```
# Reward Constants
REWARD_FAILED_ACTION     = -10
REWARD_APP_LEFT          = -5
REWARD_CRASH             = 10
REWARD_REPEATED_CRASH    = 1
REWARD_NEW_ACTIVITY      = 10
REWARD_NEW_STATE         = 5
REWARD_REPEATED_STATE    = -1
REWARD_NEW_TRANSITION    = 10
REWARD_COVERAGE_UNIT     = 10
TRANSITION_WEIGHT        = 0.5

calculate_reward(
    time_elapsed, found_activities, instr_cov, activity_cov,
    prev_activity_id, prev_element_id, current_activity_id,
    state_vector, has_crash, crash_logs, app_left, action_failed
)

    Initialize reward = 0

    If action_failed:
        Return REWARD_FAILED_ACTION

    If app_left and not has_crash:
        Log warning
        Return REWARD_APP_LEFT

    If has_crash:
        For each log in crash_logs:
            Compute hash
            If hash not seen:
                Mark as new crash
                Add to seen hashes
        Add REWARD_CRASH or REWARD_REPEATED_CRASH to reward

    new_activities = found_activities - seen_activities
    If new_activities not empty:
        Add REWARD_NEW_ACTIVITY to reward
        Update seen_activities
```

```
key = current_activity_id + "_" + prev_element_id
Convert state_vector to array

If key not in seen_states:
    Store state_vector
    Add REWARD_NEW_STATE to reward
Else:
    max_similarity = 0
    For each old_vector in seen_states[key]:
        Compute cosine similarity
        Update max_similarity
    If max_similarity < 0.85:
        Add REWARD_NEW_STATE
        Store state_vector
    Else if max_similarity > 0.98:
        Add REWARD_REPEATED_STATE
    Else:
        Add scaled reward = (1 - similarity) * REWARD_NEW_STATE

If current_activity_id not in seen_widgets:
    Initialize set
If prev_element_id not in seen_widgets[current_activity_id]:
    Add REWARD_NEW_STATE
Else:
    Add REWARD_REPEATED_STATE
Update seen_widgets

transition_key = (prev_activity_id, current_activity_id)
transition_value = (prev_activity_id, prev_element_id,
    current_activity_id)
If transition_key not in seen_transitions:
    Initialize set
If transition_value not seen:
    Add REWARD_NEW_TRANSITION * TRANSITION_WEIGHT
    Store transition_value
Else:
    Add REWARD_REPEATED_STATE * TRANSITION_WEIGHT

If first time:
    Store instr_cov and activity_cov
Else:
```

```
        Add (delta_instr_cov + delta_activity_cov) *
            REWARD_COVERAGE_UNIT

    Return total reward

End
```

Reward Analyzer tính toán phần thưởng (reward) cho từng hành động được thực hiện trong môi trường, dựa trên:

- **Tính mới của giao diện và trạng thái UI** R_s: Trạng thái hiện tại được so sánh với các trạng thái đã thấy bằng độ tương đồng cosine (cosine similarity).
  - Nếu độ tương đồng **> 0.98** ⇒ Xem là **trùng lặp**, điểm thưởng **-1**.
  - Nếu độ tương đồng từ **0.85 đến 0.98** ⇒ Xem là **gần như mới**, điểm thưởng **(1 - sim) * 5**.
  - Nếu độ tương đồng **< 0.85** ⇒ Xem là **trạng thái mới**, điểm thưởng **5**.
- **Phát hiện lỗi/crash hoặc hành vi bất thường** R_e:
  - Nếu là lỗi/crash **chưa từng thấy** trước đó ⇒ điểm thưởng **10**.
  - Nếu là lỗi/crash **đã từng thấy** ⇒ điểm thưởng **1**.
- **Phát hiện activity mới** R_v: Nếu activity mới được phát hiện (dựa trên tập hợp hoạt động đã thấy), điểm thưởng **10**.
- **Khám phá widget (element UI) mới trong activity** R_w:
  - Nếu widget chưa từng tương tác trong activity hiện tại ⇒ điểm thưởng **5**.
  - Nếu đã từng tương tác ⇒ điểm thưởng **-1**.
- **Khám phá transition (chuyển tiếp giữa activity)** R_t:
  - Nếu là chuyển tiếp mới giữa 2 activity qua 1 widget ⇒ điểm thưởng **10**.
  - Nếu đã từng thấy chuyển tiếp đó ⇒ điểm thưởng **-1**.
- **Độ phủ mã lệnh và màn hình hoạt động (Coverage)** R_c:
  - Tính độ bao phủ mới trừ độ bao phủ cũ (delta).
  - Mỗi sự thay đổi được nhân với **10** để ra điểm thưởng tương ứng.

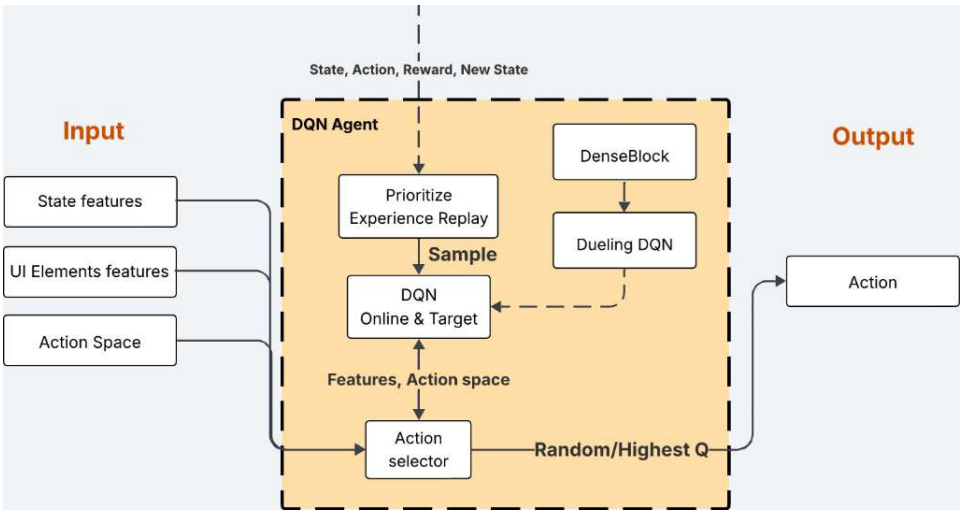
- **Hành động thất bại (Action failed)**: Nếu hành động không được thực thi thành công ⇒ Trả về điểm thưởng **-10**.
- **Rời ứng dụng (App left)**: Nếu thoát ứng dụng không phải do crash ⇒ trả về điểm thưởng **-5**.
- Trả về tổng điểm thưởng theo công thức:

$$R_{\text{total}} = R_s + R_e + R_w + \alpha R_t + R_c \tag{4.1}$$

Với α là hệ số chuyển tiếp 0.5, do tính chất chuyển tiếp có vai trò ít quan trọng hơn các tính chất còn lại.

4.3.4 DQN Agent - Chọn hành động tương tác với môi trường:

Thành phần DQN Agent là bộ não ra quyết định của hệ thống kiểm thử hộp đen, sử dụng học tăng cường để lựa chọn hành động phù hợp với trạng thái giao diện hiện tại. Thành phần này được xây dựng dựa trên ý tưởng từ DQT với kiến trúc Double-Dueling Deep Q-Network (Double-Dueling DQN), kết hợp với bộ nhớ ưu tiên (Prioritized Experience Replay) và cơ chế chọn hành động tốt nhất (epsilon-greedy).



Hình 4.6: Luồng hoạt động của DQN Agent

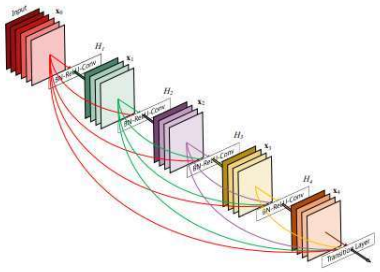
Chi tiết mô hình như sau:

- **Tính toán hành động**: Agent sử dụng mạng Dueling DQN để tính giá trị Q cho tất cả các hành động có thể. Dữ liệu đầu vào bao gồm trạng thái hiện tại và các vector hành động (one-hot).

- **Chiến lược chọn hành động**:
  - Nếu $random < \epsilon$ (tham lam - exploration), chọn hành động ngẫu nhiên.
  - Ngược lại (exploitation), chọn hành động có Q-value lớn nhất từ mạng DQN.
  - ϵ giảm dần theo số bước (2000 bước) tương tác để chuyển từ khám phá sang khai thác.

$$chooseAction(s) = \begin{cases} \arg\max_a Q(s,a), & 1-\epsilon \\ \text{random action}, & \epsilon \end{cases} \tag{4.2}$$

- **Mạng DenseBlock**:
  - DenseBlock (Một biến thể của DenseNet[3]) được đề cập trong mô hình của DQT. DenseBlock bao gồm 4 lớp kết nối đầy đủ (fully connected) dùng để trích xuất đặc trưng.

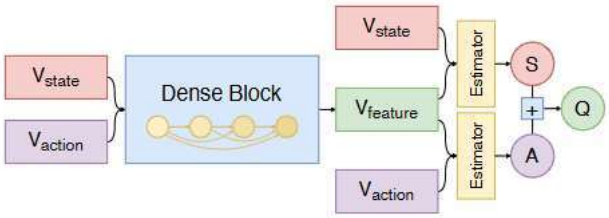


Hình 4.7: Mạng DenseNet được dùng trong trích xuất đặc trưng ảnh

DenseNet kết nối mỗi lớp với tất cả các lớp phía trước theo kiểu lan truyền tiến (feed-forward), giúp tái sử dụng đặc trưng, giảm đáng kể số lượng tham số và yêu cầu ít tính toán hơn trong khi vẫn đạt hiệu suất cạnh tranh.

- **Mạng Dueling DQN**:
  - Dựa trên kiến trúc dueling gồm 2 nhánh kế hợp với DenseBlock để xử lý không gian trạng thái–hành động (state-action) phức tạp của các ứng dụng Android, nhằm tăng khả năng biểu đạt của mô hình trong khi vẫn duy trì tính gọn nhẹ.

[3]https://arxiv.org/abs/1608.06993



Hình 4.8: Mô hình DQN sử dụng DenseBlock

  - ⁎ **Value head** $V(s)$: đánh giá giá trị của trạng thái.
  - ⁎ **Advantage head** $A(s,a)$: đo độ lợi thế của mỗi hành động trong trạng thái đó.
- Q-value được tính bằng công thức kết hợp $Q(s,a) = V(s) + A(s,a)$.

- **Lưu trữ dữ liệu huấn luyện**:
  - Dữ liệu $(s, a, r', s', done)$ được lưu vào Replay Buffer có ưu tiên (Prioritized Experience Replay).
  - Ưu tiên dựa vào TD-error: $\delta = |Q_{current} - Q_{target}|$.

- **Huấn luyện mạng DQN**:
  - Lấy mẫu minibatch từ Replay Buffer với xác suất theo trọng số ưu tiên.
  - Tính giá trị mục tiêu: $target = r + \gamma \cdot Q_{target}(s', a')$.
  - Cập nhật mạng chính qua hàm mất mát (loss function) SmoothL1 giữa Q hiện tại và target.
  - Cập nhật trọng số buffer theo TD-error.

Listing 4.4: Mã giả thực hiện huấn luyện mô hình DQN sử dụng sample từ PER

```
For i in sample:
    Without gradient:
        Compute Q-values of next states using online DQN
        Select best action indices via argmax

        Compute Q-values of next states using target DQN
        Get max Q-value using best action indices

        Compute target:
            target = reward + (1 - done) * gamma * max_next_q
```

```
Compute current Q-values using online DQN
Compute TD errors: delta = Q_current - Q_target
Compute weighted loss using TD error and importance weights
Update replay buffer priorities using new TD error
```

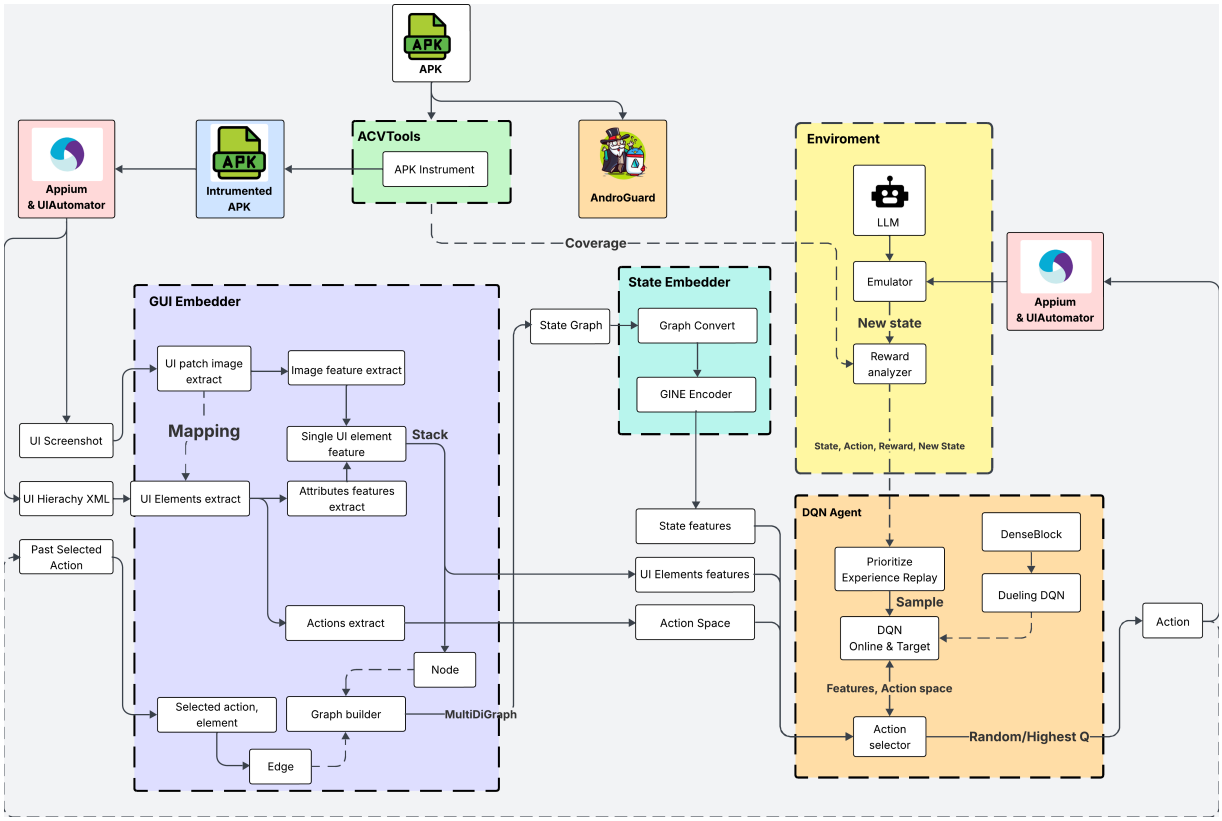
- **Cập nhật mạng mục tiêu (target DQN)**:
  - Sử dụng kỹ thuật *soft update*:

$$\theta_{target} \leftarrow \tau \cdot \theta_{online} + (1-\tau) \cdot \theta_{target}$$

  - Đảm bảo mạng mục tiêu ổn định trong quá trình huấn luyện.

4.3.5 Chạy kiểm thử và huấn luyện mô hình

Phần này trình bày chi tiết quá trình chạy và kiểm thử của mô hình kiểm thử hộp đen đề xuất.



Hình 4.9: Tổng quan mô hình đề xuất

Quá trình kiểm thử và huấn luyện gồm 5 bước chính:

- **Extract current State feature:** Mô hình thực hiện lấy cây UI XML (UI Hierarchy XML) và ảnh của màn hình hiện tại. Sau đó đưa vào GUI Embedder để thực hiện trích xuất đặc trưng, không gian hoạt động $\mathscr{A}$ và đồ thị trạng thái $Graph(s)$. Tiếp tục đưa đồ thị $Graph(s)$ vào State Embedder để trích xuất đặc trưng của đồ thị.

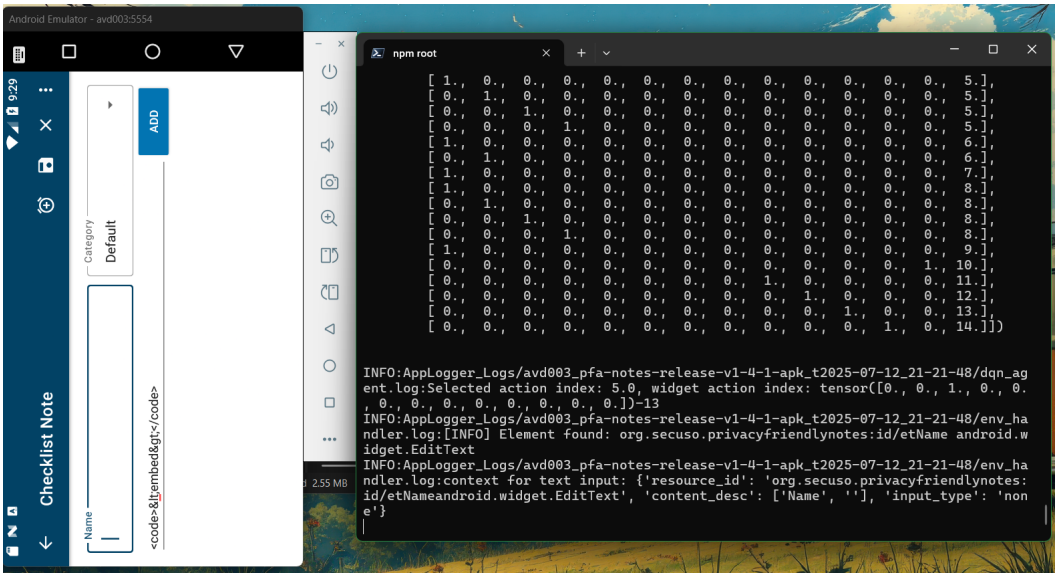
Listing 4.5: Tóm tắt trích xuất đặc trưng của trạng thái tại mỗi bước kiểm thử

```
Begin
    Get UI Hierarchy XML and screenshot
    Extract action space and elements features from XML
    Extract Image features from patches image from screenshot
    Matching each element features with it corresponding image
        features via bounds create final element features
    Create state graph. from element features with previous
        selected action and previous element features as edge,
        node contains current elements features and action
        space
    Convert state graph from networkx MuliDiGraph to Torch
        Data.
    Feed Data to GINEConv model to get state features
    Return action space, element features, state features
End
```

- **Select Action:** Sau khi trích xuất được đặc trưng(action space $\mathscr{A}$, element features, state features), đưa các đặc trưng vào mô hình DQN để chọn hành động để thực hiện.

Listing 4.6: Tóm tắt cách lựa chọn hành động

```
Begin
   Given action space, element features, state features
   - If  random < epsilon:
        Random action
   - Else:
       Combine element features (n,138) and expanded state
           features (from (1,96) to (n,96)) into single
           tensor and stack into shape (n,234)
       Feed to DQN model, return a list of size [n]
           contains Q values of each action.
       Pick highest Q value action index
    Update epsilon
End
```



Hình 4.10: Quá trình thực hiện kiểm thử

- **Perform Action:** Thực hiện hành động đã được chọn thông qua Driver của Appium
- Lấy trạng thái mới và tính phần thưởng: Sau khi thực hiện hành động, mô hình sẽ lấy đặc trưng của trạng thái mới s' để có thể tính toán phần thưởng cho mô hình thông qua Reward analyzer.
- Lưu trữ lại trạng thái-hành động và huấn luyện mô hình: Thực hiện

Listing 4.7: Tóm tắt quá trình lưu và huấn luyện DQN

```
Begin
    Save (s, a, r, s', done) to memory with priority = 1,
        higher priority more focus
    Training sample top 32 priority
    Feed sample into DQN online and target network
    Calculate TD_error
    Assign new priority with TD_error
End
```

4.4 Kết quả thí nghiệm

Ở mục này, chúng tôi sẽ trình bày các kết quả thực nghiệm trên tập dữ liệu sau khi chạy 3 mô hình: Mô hình đề xuất, DQT và Monkey (baseline). Mỗi ứng dụng sẽ được chạy kiểm thử trong 2 giờ với từng mô hình.

4.4.1 Tiêu chí đánh giá của các mô hình

Để đánh giá kết quả của các mô hình chúng tôi chọn: Độ phủ dòng lệnh (Instruction Coverage), độ phủ màn hình (Activity Coverage) và số lỗi/crash tìm được:

- *Instruction Coverage và Activity Coverage* đại diện cho khả năng khám phá (exploration) của mô hình.
- *Số lỗi/crash tìm được* đại diện cho độ hiệu quả trong kiểm thử của mô hình.

4.4.2 Kết quả và nhận xét

Application	Version	Android SDK	DQT		Monkey		Our	
			Activity Cov	Instr Cov	Activity Cov	Instr Cov	Activity Cov	Instr Cov
AntennaPod	3.8.0	Android SDK 21	6	13.48%	_	**28.92%**	6	22.32%
Slideshow Wallpaper	1.2.2	Android SDK 18	3	4.61%	_	5.62%	**6**	**5.89%**
NewPipe APK	0.27.7	Android SDK 21	3	19.01%	_	20.52%	**5**	**21.89%**
Seal for Android	1.13.1	Android SDK 21	**2**	14.03%	_	**28.60%**	1	23.02%
MyExpenses	3.4.5	Android SDK 24	**21**	2.06%	_	**7.97%**	11	6.03%
Notes	1.4.1	Android SDK 24	**12**	7.41%	_	**9.07%**	10	8.89%
SpotiFlyer for Android	3.6.4(latest)	Android SDK 21	2	14.03%	_	_	**3**	**17.97%**

Hình 4.11: Kết quả độ phủ trong từng ứng dụng

Application	Version	Android SDK	DQT	Monkey	Our
AntennaPod	3.8.0	Android SDK 21	0	0	**5**
Slideshow Wallpaper	1.2.2	Android SDK 18	0	0	**1**
NewPipe APK	0.27.7	Android SDK 21	0	0	**1**
Seal for Android	1.13.1	Android SDK 21	0	0	**1**
MyExpenses	3.4.5	Android SDK 24	**2**	0	0
Notes	1.4.1	Android SDK 24	1	0	**2**
SpotiFlyer for Android	3.6.4(latest)	Android SDK 21	0	0	**1**

Hình 4.12: Số lỗi/crash tìm được trong từng ứng dụng

Application	Type Error found
AntennaPod	java.lang.IllegalArgumentException
AntennaPod	java.lang.IllegalStateException
AntennaPod	java.lang.NullPointerException
AntennaPod	java.util.concurrent.CancellationException
AntennaPod	java.lang.ClassNotFoundException
Slideshow Wallpaper	java.lang.ClassNotFoundException
NewPipe APK	java.security.cert.CertPathValidatorException
Seal for Android	kotlinx.serialization.json.internal.WriteModeKt.JsonDecodingException
Notes	java.lang.ClassNotFoundException
Notes	java.lang.NullPointerException
SpotiFlyer for Android	java.lang.IllegalStateException

Hình 4.13: Số lỗi tìm được của mô hình đề xuất

Từ những kết quả thu được sau khi thực nghiệm với ba mô hình và so sánh các dữ liệu giữa các mô hình **Hình 4.11** cho thấy mô hình đề xuất có độ bao phủ dòng lệnh (Instruction coverage) tốt nhất ở các ứng dụng Slideshow Wallpaper, NewPipe APK, SpotiFlyer for Android với lần lượt là 5.89%, 21.89% và 17.97% cao hơn so với mô hình DQT lần lượt là 1.27%, 2.88%, 1.49% và 3.93% và Monkey lần lượt là 0.27%, 1.36% với hai ứng dụng đầu, cho thấy mô hình có thể thực hiện tương đối tốt hơn trong quá trình thí nghiệm. Trung bình độ bao phủ của mô hình đề xuất là 15.14% so với DQT ở 10.66% cho thấy hiệu quả của sử dụng lớp GINE thay cho lớp GIN và tích hợp ảnh vào quá trình trích xuất đặc trưng. Monkey, tuy có độ phủ cao, song ở một số ứng dụng thì không có khả năng kiểm thử (fail to start) và hoạt động hoàn toàn ngẫu nhiên hoặc theo một xác suất đã cài đặt trước ít có sự thay đổi trong hoạt động.

Về độ phủ màn hình hoạt động (Activiy coverage) cho thấy mô hình đề xuất có khả năng khám phá khá cao với trung bình 6 activity, so với mô hình DQT trung bình 7 activity, cần học trước một số lượng nhất định đồ thị trạng thái thông qua InfoGraph thì mô hình đề xuất thể hiện tốt hơn.

Với kết quả tìm được lỗi/crash trong lúc kiểm thử ứng dụng **Hình 4.12** cho thấy khả năng tìm và phát hiện lỗi của mô hình đề xuất so với mô hình DQT và Monkey tương đối tốt hơn khi tìm thấy lỗi ở 6 trong 7 ứng dụng được kiểm thử. **Hình 4.13** cho thấy chi tiết của từng loại lỗi phát hiện trong ứng dụng.

4.5 Tính xác thực của lỗi tìm được

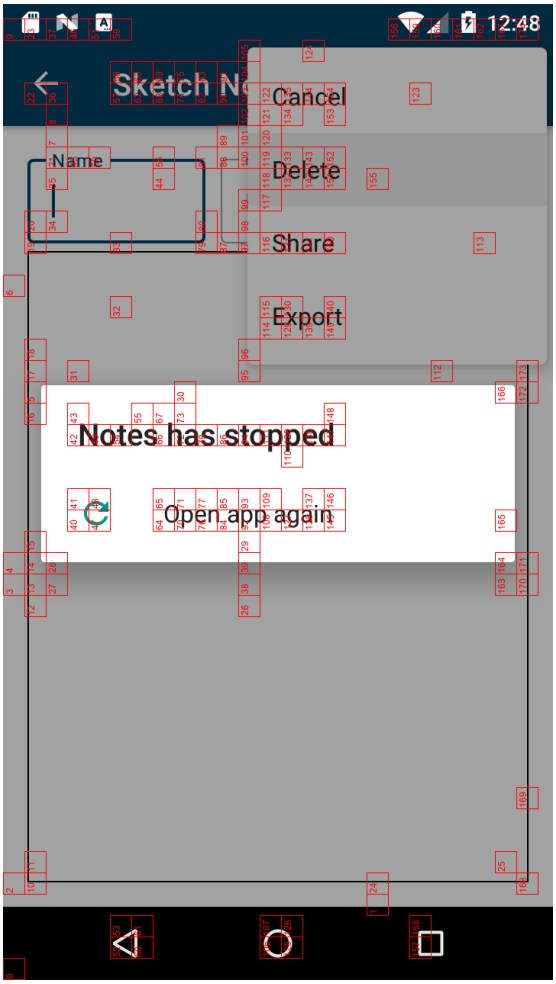
Ở phần này chúng tôi sẽ kiểm tra tính xác thực của lỗi tìm được của mô hình đề xuất

Chọn lỗi **java.lang.NullPointerException** của ứng dụng **Notes v1.4.1**, đọc trong log kiểm thử của ứng dụng thấy log của crash **Hình 4.14**:

```
--------- beginning of crash
07-11 00:48:13.196 11812 11812 E AndroidRuntime: FATAL EXCEPTION: main
07-11 00:48:13.196 11812 11812 E AndroidRuntime: Process: org.secuso.privacyfriendlynotes, PID: 11812
07-11 00:48:13.196 11812 11812 E AndroidRuntime: java.lang.NullPointerException: Attempt to invoke a virtual method on a null object
reference
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at
org.secuso.privacyfriendlynotes.ui.notes.BaseNoteActivity.displayTrashDialog$lambda-9(BaseNoteActivity.kt)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at org.secuso.privacyfriendlynotes.ui.notes.BaseNoteActivity.$r8$lambda
$g9Y8kdTHDcd2jPPWmy6H1MJ2l74(BaseNoteActivity.kt)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at org.secuso.privacyfriendlynotes.ui.notes.BaseNoteActivity
$$ExternalSyntheticLambda4.onChanged(D8$$SyntheticClass)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at androidx.lifecycle.LiveData.considerNotify(LiveData.java)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at androidx.lifecycle.LiveData.dispatchingValue(LiveData.java)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at androidx.lifecycle.LiveData.setValue(LiveData.java)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at androidx.lifecycle.MutableLiveData.setValue(MutableLiveData.java)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at androidx.lifecycle.LiveData$1.run(LiveData.java)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at android.os.Handler.handleCallback(Handler.java:751)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at android.os.Handler.dispatchMessage(Handler.java:95)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at android.os.Looper.loop(Looper.java:154)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at android.app.ActivityThread.main(ActivityThread.java:6077)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at java.lang.reflect.Method.invoke(Native Method)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at com.android.internal.os.ZygoteInit
$MethodAndArgsCaller.run(ZygoteInit.java:866)
07-11 00:48:13.196 11812 11812 E AndroidRuntime:        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:756)
07-11 00:48:13.198  1348  1377 W audio_hw_generic: Not supplying enough data to HAL, expected position 13658584 , only wrote 13658400
07-11 00:48:13.211  1341  1402 E SurfaceFlinger: ro.sf.lcd_density must be defined as a build property
07-11 00:48:13.256  1649 12999 D         : HostConnection::get() New Host Connection established 0x8e313540, tid 12999
07-11 00:48:13.258  1649 12999 I OpenGLRenderer: Initialized EGL, version 1.4
07-11 00:48:13.258  1649 12999 D OpenGLRenderer: Swap behavior 1
07-11 00:48:13.258  1649 12999 W OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying without...
```

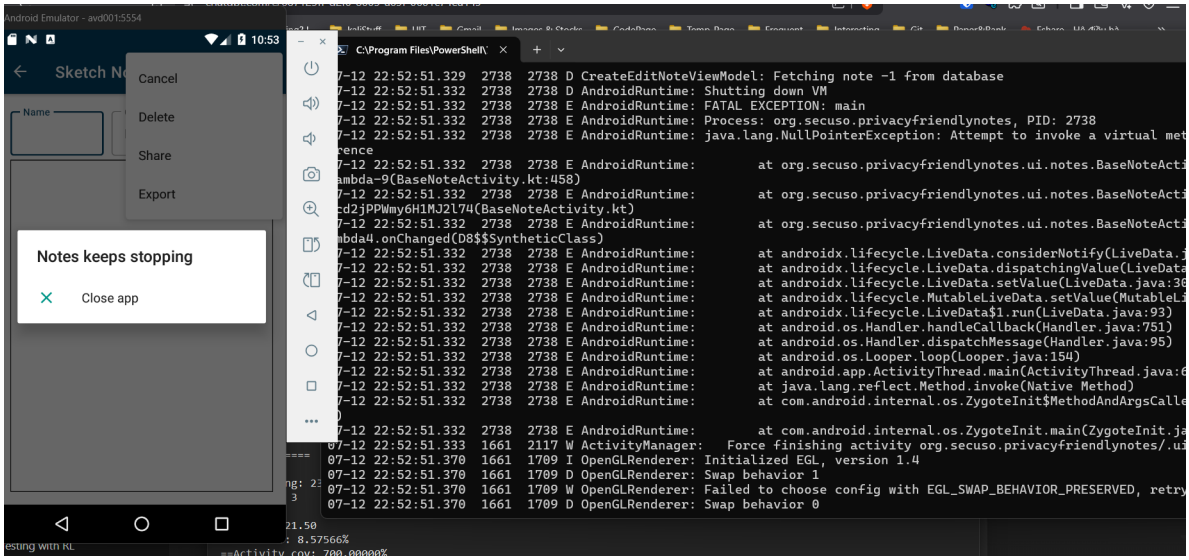
Hình 4.14: Log mô hình báo crash trong quá trình kiểm thử ứng dụng Notes

Điều tra các ảnh chụp màn hình trong lúc thực hiện kiểm thử cho thấy ứng dụng bị crash khi thực hiện hành động **Delete** trong màn hình tạo **Sketch Note**:



Hình 4.15: Ảnh chụp lại màn hình lúc xảy ra crash

Thực nghiệm lại để kiểm tra lỗi tìm được **Hình 4.16** và xác định lỗi có thể tái tạo lại được:



Hình 4.16: Thực nghiệm lại

Thực nghiệm trên chứng minh cho khả năng tìm được lỗi của mô hình đề xuất là có căn cứ và có thể tái tạo lại được.

4.6 Thảo luận chung

Kết quả thử nghiệm trên cho thấy, mô hình của nhóm đề xuất có khả năng kiểm thử tương đương với các mô hình kiểm thử hiện có và có khả năng phát hiện lỗi trong ứng dụng hiệu quả. Qua các thí nghiệm trên chúng tôi đã có thể đưa ra kết quả của những câu hỏi nghiên cứu như sau:

- **Câu hỏi 1:** Độ phủ của mô hình đề xuất là bao nhiêu so với các mô hình kiểm thử hiện có? Phương pháp tính độ phủ như thế nào?

  Thông qua thí nghiệm độ phủ của mô hình chúng tôi phát triển đạt được giá trị cao hơn so với những gì chúng tôi ước tính, với kết quả độ phủ tương đối cáo so với DQT và với nhiều ứng dụng thì độ phủ cũng cượt qua Monkey. Tuy nhiên vẫn còn nhiều ứng dụng mà dộ phủ của mô hình chúng tôi vẫn chưa được vượt trội hơn so với Monkey, nguyên nhân có thể là sự ngẫu nhiễn trong mô hình RL của chúng tôi vẫn còn lớn, điều này chúng tôi sẽ lưu ý và tối ưu hóa mô hình.

  Phương pháp tính độ phủ chúng tôi sử dụng thông qua công cụ ACVTools.

  ACVTool sẽ chèn các probe vào sau mỗi lệnh smali để theo dõi việc thực thi ở các mức độ lớp, phương thức và lệnh. Các probe này được liên kết với một mảng nhị phân để ghi lại trạng thái thực thi (executed or not).

Công thức tính độ phủ

Độ phủ tổng quát:

$$\text{Độ phủ (\%)} = \left(\frac{\text{Số thành phần được thực thi}}{\text{Tổng số thành phần có thể thực thi}} \right) \times 100$$

Độ phủ hoạt động :

$$\text{Độ phủ (\%)} = \left(\frac{\text{Số activity được thực thi}}{\text{Tổng số activity}} \right) \times 100$$

- **Câu hỏi 2:** Những cải tiến của mô hình đề xuất so với những mô hình hiện có và ảnh hưởng lên hiệu quả, kết quả của mô hình đề xuất?

  Đối với mô hình DQT, mô hình đề xuất sử dụng kết hợp giữa đặc trưng hình ảnh và đặc trưng từ cây UI từ đó giúp tăng khả năng nhận diện của mô hình, hơn nữa, mô hình đề xuất sử dụng trích xuất đồ thị trạng thái từ GINE thay vì sử dụng GIN nhằm chú trọng vào đặc trưng của cạnh mà DQT bỏ qua. Kết quả cho thấy mô hình đề xuất có độ phủ tốt hơn DQT 4.48%.

  Đối với Monkey, mô hình đề xuất có khả năng học tốt hơn và có khả năng tìm lỗi tốt hơn do hành động không lựa chọn hoàn toàn dựa theo xác xuất.

- **Câu hỏi 3:** Khả năng phát hiện lỗi của mô hình đề xuất, so sánh với các mô hình hiện có? Có bao nhiêu lỗi là độc nhất, tính xác thực của lỗi tìm được?

  Mô hinhf đề xuất có khả năng phát hiện lỗi tốt và số lượng lỗi nhiều hơn các mô hình hiện tại và các lỗi đều có căn cứ để có thể tái hiện lại thông qua log, hình ảnh được ghi lại. Căn cứ về khả năng tái tạo lỗi đã được chúng tôi thực hiện ở phần Kết luận trên.

Từ những câu hỏi nghiên cứu trên, chúng tôi nhận thấy rằng, ý tưởng về mô hình kiểm thử hộp đen với học tăng cường là khả thi và có thể đem lại hiệu quả cao đối với phát hiện lỗi của ứng dụng Android. Đồng thời, chúng tôi cũng chứng minh được tính chính xác và khả tái tạo của lỗi được mô hình đề xuất phát hiện. Dựa vào kết quả thí nghiệm, cho thấy mô hình đề xuất có kết quả khả quan đặc biệt ở phần phát hiện lỗi - côt lõi của một mô hình kiểm thử. Chi tiết về mô hình đề xuất, chúng tôi giải thích thêm ở phần Hướng phát triển. Về mã nguồn của mô hình: `https://github.com/Witnull/DACN`

Chương 5

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Tóm tắt chương

Trong chương cuối này, chúng tôi sẽ tổng hợp lại những công việc nhóm đã thực hiện và đề xuất hướng phát triển trong tương lai.

5.1 Kết luận

Ngày nay cùng với sự phát triển của các ứng dụng Android, giao diện người dùng (GUI) ngày càng trở nên phức tạp. Cùng với đó, nhu cầu của người dùng về những trải nghiệm cá nhân của bản thân khi tương tác với các ứng dụng cũng ngày càng cao. Điều này đã đặt ra một vấn đề lớn cho những người kiểm thử phần mềm, vì tính phức tạp của giao diện cũng như thời gian kiểm thử quá dài, đòi hỏi kinh nghiệm. Để giải quyết vấn đề này, việc áp dụng trí tuệ nhân tạo (AI) trong việc kiểm thử ứng dụng Android đang được phát triển và đã đạt được những thành tựu đáng kể

Thông qua đề tài này, chúng tôi đã nghiên cứu các bài báo về các mô hình tương tự trong việc kiểm thử, kết hợp với với việc tái hiện các mô hình hiện có và so sánh kết quả kiêm thử với mô hình đề xuất . Bằng cách so sánh kết qủa của việc kiểm thử (Instruction coverage, Activity coverage, số lỗi/crash tìm được) chúng tôi đã có những số liệu khách quan như sau :

- Thời gian kiêm thử đã được giảm đáng kể so với phương pháp thủ công, tăng hiệu quả và giảm chi phí nhân công.
- Việc kiểm thử đạt được độ phủ dòng lệnh (Instruction coverage) trên các ứng dụng dùng kiểm thử cao hơn so với các mô hình hiện có.

Bên cạnh những điều đạt được như trên, mô hình đề xuất của chúng tôi vẫn còn một số hạn chế nhất định như sau:

- Mô hình còn nhiều biến động và ngẫu nhiên
- Mô hình chưa nắm được môi trường (world model) dẫn đến lặp lại, exploit, chưa thể kiểm thử hoàn toàn được ứng dụng.
- Mô hình chưa nắm được hoàn toàn nội dùng (context) của UI như điền form
- Chưa khám phá được một số chức năng yêu cầu upload, tải file lên app

Tuy nhiên so với các mô hình được cung cấp từ các bài báo thì kết quả kiểm thử của mô hình đề xuất có kết quả khả quan và có thể phát hiện được các lỗi giao diện tốt hơn.

5.2 Hướng phát triển

Dựa vào kết quả kết quả thu được tuy vẫn còn nhiều sự biến động trong mô hình nhưng phương pháp chung tôi đề ra vần còn tiềm năng phát triển hơn nữa để giải quyết các vấn đề còn tồn đọng.

1. Giảm độ biến động và ngẫu nhiên của mô hình.
2. Tích hợp thêm mô hình môi trường (world model) để mô hình có thể khám phá hiệu quả hơn
3. Nâng cao khả năng hiểu bối cảnh (Context) của giao diện người dùng (UI)
4. Hỗ trợ kiểm thử các chức năng phức tạp (upload, tải file):
    - Tạo dữ liệu thử nghiệm đa dạng: Sử dụng một kho dữ liệu thử nghiệm (Rest Data Repository) với các loại file khác nhau (hình ảnh, PDF, v.v.) để mô hình có thể thử nghiệm các chức năng upload/tải.
    - Tích hợp Intent Testing: Sử dụng các công cụ như Android Intent Fuzzer để kiểm thử các Intent liên quan đến việc mở file hoặc chia sẻ dữ liệu giữa các ứng dụng.

TÀI LIỆU THAM KHẢO

[1] Ana Paula Cardoso et al. “Evaluation of Automatic Test Case Generation for the Android Operating System using Deep Reinforcement Learning”. In: *Proceedings of the XXII Brazilian Symposium on Software Quality*. SBQS ’23. Brasília, Brazil: Association for Computing Machinery, 2023, pp. 228–235. isbn: 9798400707865. doi: 10.1145/3629479.3629503. url: https://doi.org/10.1145/3629479.3629503.

[2] Weihua Hu et al. *Strategies for Pre-training Graph Neural Networks*. 2020. arXiv: 1905.12265 [cs.LG]. url: https://arxiv.org/abs/1905.12265.

[3] Yuanhong Lan et al. “Deeply Reinforcing Android GUI Testing with Deep Reinforcement Learning”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE ’24. Lisbon, Portugal: Association for Computing Machinery, 2024. isbn: 9798400702174. doi: 10.1145/3597503.3623344. url: https://doi.org/10.1145/3597503.3623344.

[4] Aleksandr Pilgun et al. “Fine-grained Code Coverage Measurement in Automated Black-box Android Testing”. In: *ACM Trans. Softw. Eng. Methodol.* 29.4 (July 2020). issn: 1049-331X. doi: 10.1145/3395042. url: https://doi.org/10.1145/3395042.

[5] Andrea Romdhana et al. “Deep Reinforcement Learning for Black-box Testing of Android Apps”. In: *ACM Trans. Softw. Eng. Methodol.* 31.4 (July 2022). issn: 1049-331X. doi: 10.1145/3502868. url: https://doi.org/10.1145/3502868.

[6] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019. url: https://openreview.net/forum?id=ryGs6iA5Km.