An Evaluation of the State-of-the-Art Testing Algorithms for Android Mobile Applications

ARUL MATHUR

Henry Gunn High School

Motivation

Why does mobile testing matter?

Online experience is becoming **Mobile-First**

- Banking
- Entertainment
- Social apps
- Gaming
- VC
- Health
- Business
- Hobbies

… and more

Problem Statement

SECURITY

USER EXPERIENCE

PRODUCTIVITY

Research Methodology

IDENTIFY	Identify commonly used approaches in industry and research
UNDERSTAND	Literature review to deep dive into different testing methodologies
BENCHMARK	Establish standard benchmark
COMPARE	Compare **CODE COVERAGE** and **FAULT DETECTION** of different approaches against the benchmark.

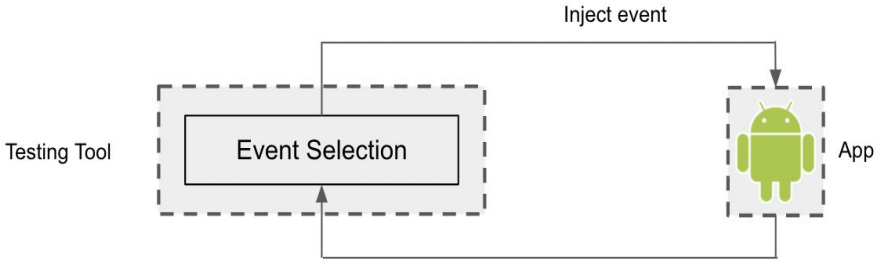
Android has 70+% of mobile OS market share. We use it to do the comparisons

Approaches to Testing

Random Testing

Send random, independent inputs to the app

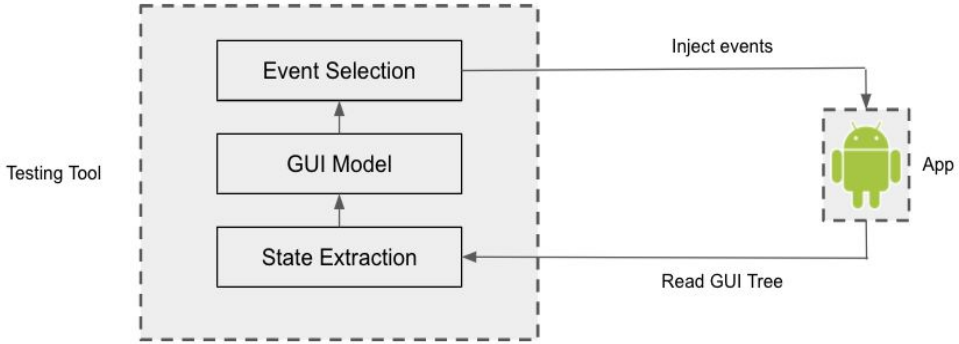
- ✓ Fully blackbox
- ✗ Difficult to reproduce crashes
- ✗ Prior info not leveraged



Model Based Testing

Infer app model and state from GUI, and exploit it

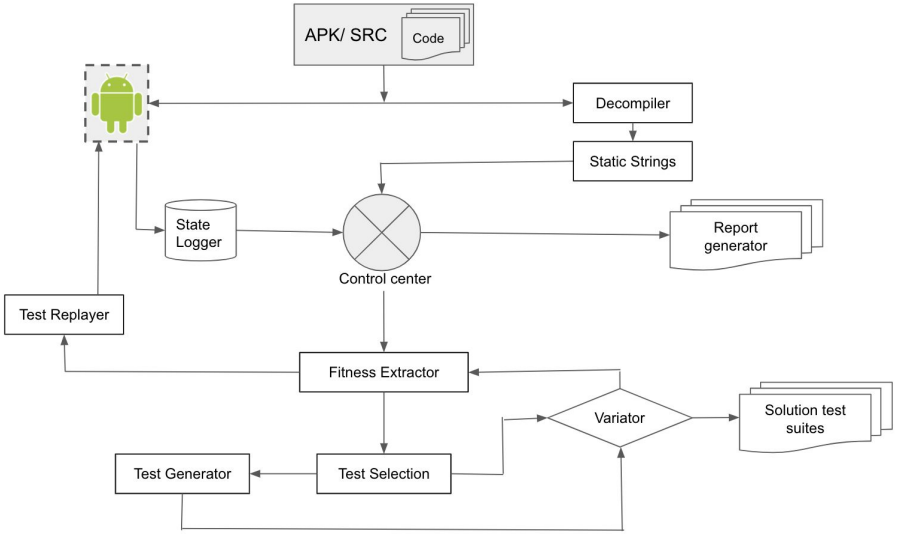
- ✓ Reuses learned information
- ✓ Easy to reproduce crashes
- ✗ Model cannot capture all information



Systematic Testing

Use symbolic execution to get high test coverage

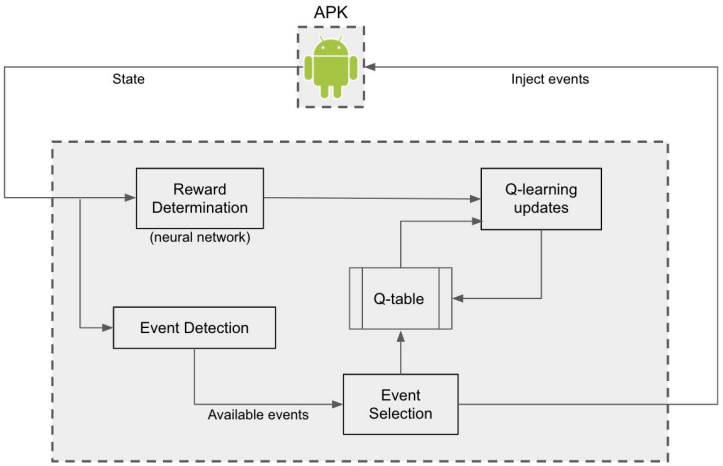
- ✓ Efficient use of internals
- ✗ Not blackbox
- ✗ Path explosion leads to complexity



Machine Learning Based Testing

Train model on many apps to learn app behaviors, and test using that knowledge

Trial/error nature lends to regressive learning algorithms



- ✓ Deep learning about apps
- ✓ Leverage info across apps
- ✗ Complex to build due to variety of events and widgets

Results and Findings

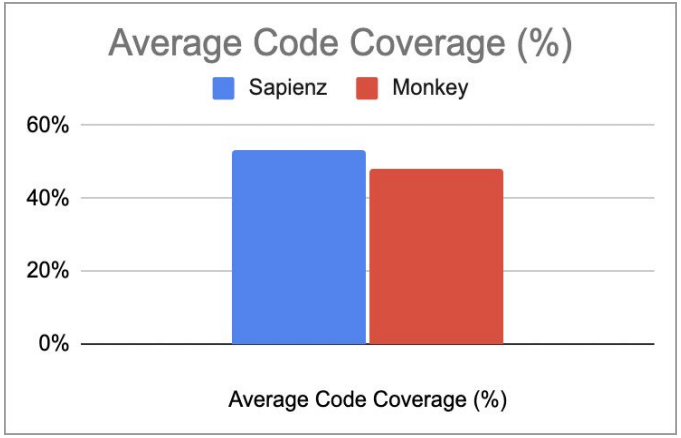
Monkey

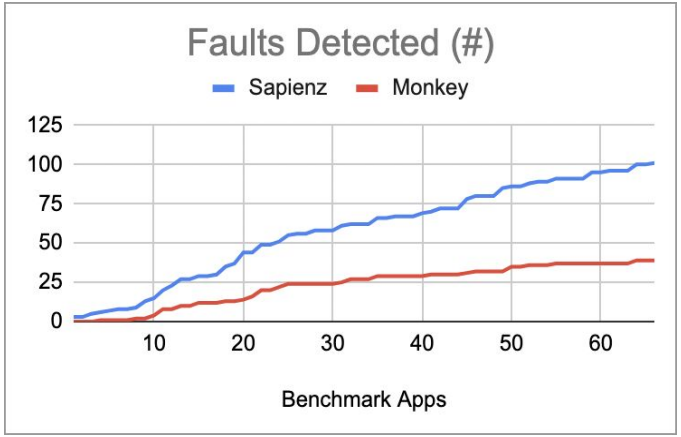
- Google's default Android testing tool
- Random testing, no learning
- Pros
  - Out-of-the-box, simple. Widely used.
- Cons
  - Bug recreation is hard
  - Long execution times
  - No ability of event selection

Selected as benchmark for wide adoption

Sapienz

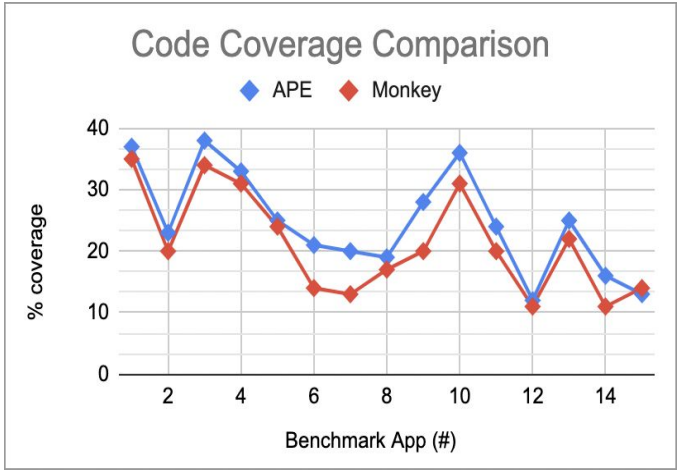
- White box testing by instrumenting the code
- Provides different capabilities depending on level of access
- White box testing is resource intensive and non-leverageable across apps

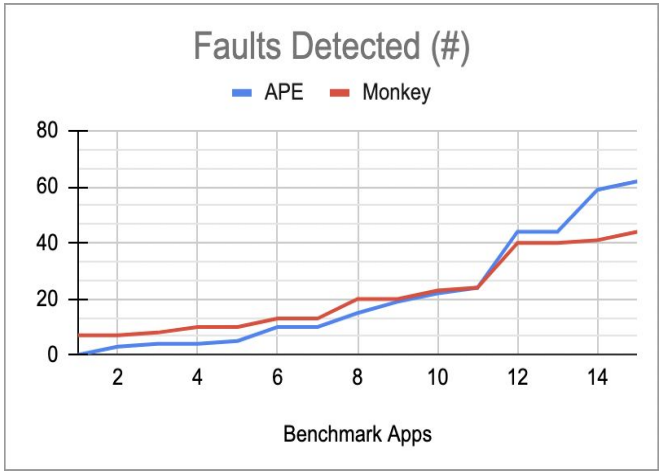




APE

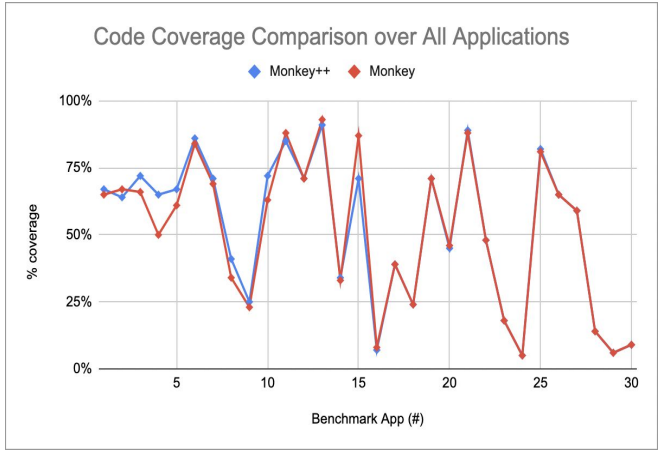
- Starts with an abstract model, and tunes it as it tests and learns.
- Dynamic tuning well suited for complex GUI trees
- Effective blackbox testing

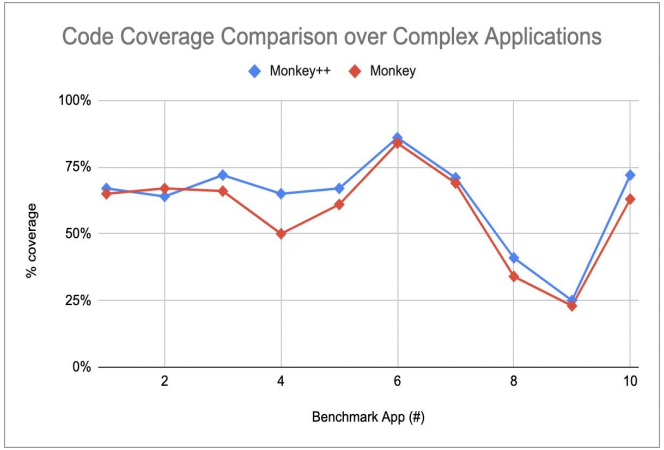




DeepGUI/Monkey++

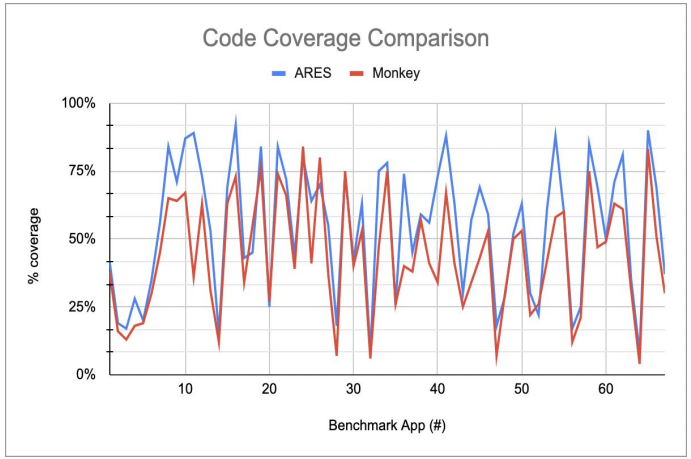
- Uses machine learning to create heat map of pixels likely to be widgets
- Sophisticated event selection process
- Implemented in Monkey, creating Monkey++

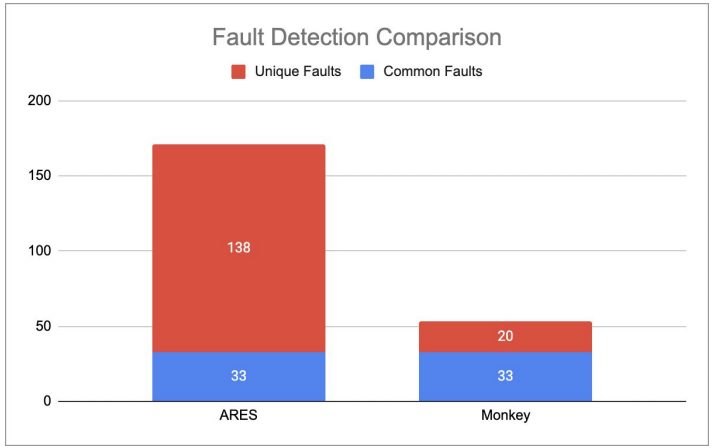




ARES

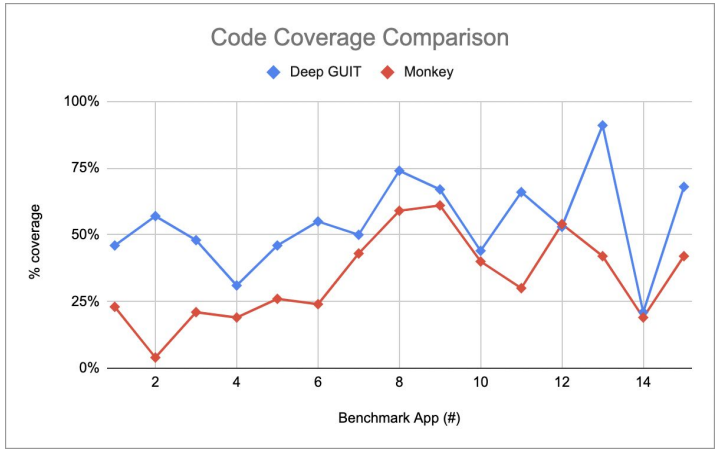
- Generic, black box testing tool that learns and evolves
- Neural network semantics - trial and error testing with reward system
- Users can plug in any of the neural learning algorithms

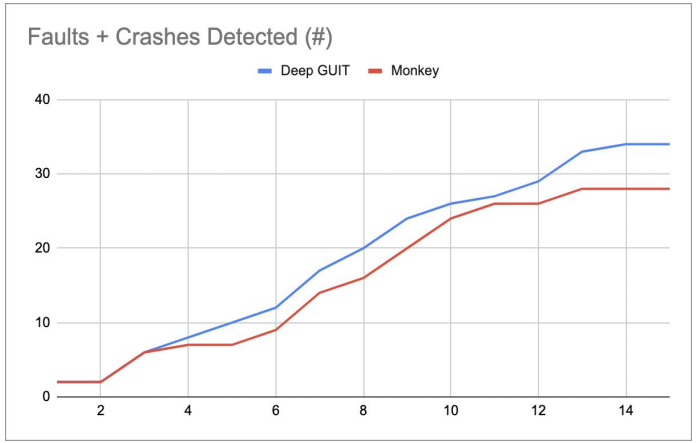




DeepGUIT

- Q-learning: Understand value of an action in a certain state, and try to get to a more "interesting" state from there
- Whitebox testing, needs code instrumentation to be effective
- Can lead to an explosion of # of state-action pairs in complex apps





Conclusion and Implications

Conclusions

- ML-based tools most effective
- Neural algorithms have potential to create intelligent automated tools
- Sapienz has very effective fault detection
- Monkey++ demonstrates significantly more efficient event detection

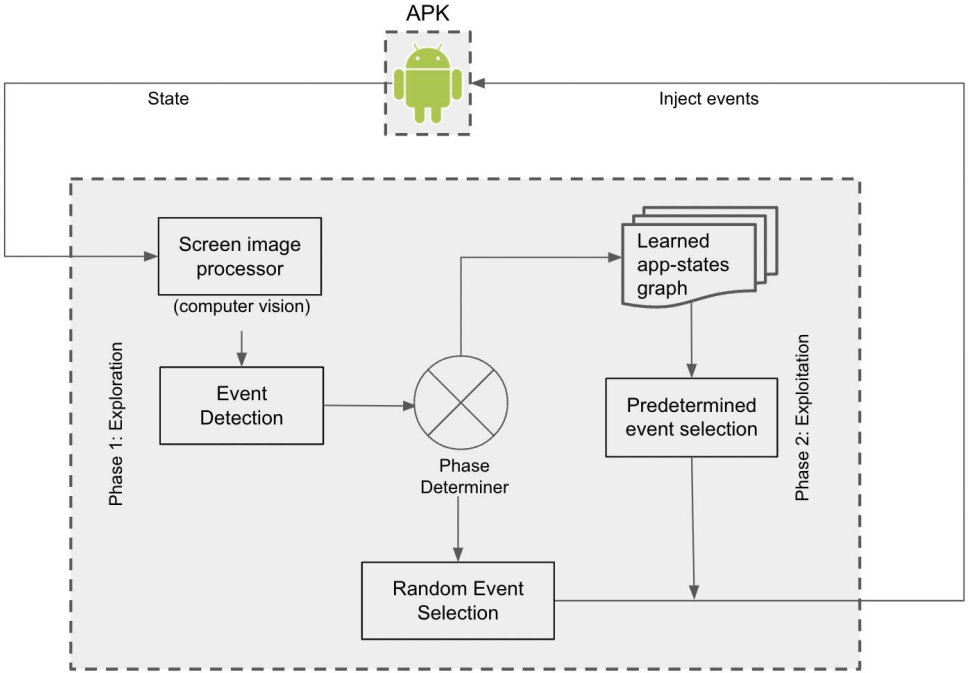
	Effective		Fastest	Best fault detection	Best code coverage
	Sapienz	**APE**	**Monkey++**	**ARES**	**DeepGUIT**
Code coverage improvement	10.4%	16.7%	8.3%	23.4%	**58.8%**
Fault detection improvement	**159.0%**	41.0%	-	**223.0%**	21.4%
Testing modes	Multi-mode	Black-box	Black-box	Black-box	Black-box
Testing Approach	Systematic	Model	Random	ML	ML

But, there are gaps

- Random or model based testing
  - Can't handle complex workflows and states
  - Faults difficult to reproduce in random testing
  - Either low code coverage or long testing times - both lead to poor fault detection
- Neural Self Learning methods
  - Need very a large set of training apps to get trained
  - Time intensive to build. Need specialized skills.
  - Android events and widgets are complex and varied and upsets self learning
  - Runs into memory pressure with complex workflows

My Algorithm

- Create app models for testing
- Use computer vision to understand GUI
- Discover states with many unexplored events to maximize code coverage
- Implements regressive learning to improve event selection
- Adapt reward mechanisms to optimize finding security vulnerabilities



References

[1] Laricchia, F. (2022, February 7). Mobile OS market share 2021. Statista. Retrieved April 11, 2022, from https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems- since-2009/

[2] Google. 2020. UI/Application Exerciser Monkey. Retrieved April 4, 2022, from https://developer.android.com/studio/test/monkey

[3] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. 2019. Practical GUI testing of Android applications via model abstraction and refinement. In Proceedings of the 41st International Conference on Software Engineering (ICSE).

[4] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, “Guided, Stochastic Model-Based GUI Testing of Android Apps,” in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017), 2017, pp. 245–256.

[5] Y.-M. Baek and D.-H. Bae, “Automated Model-based Android GUI Testing Using Multi-level GUI Comparison Criteria,” in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016), 2016, pp. 238–249.

[6] W. Yang, M. R. Prasad, and T. Xie, “A Grey-box Approach for Automated GUI-model Generation of Mobile Applications,” in International Conference on Fundamental Approaches to Software Engineering (FASE 2013). Springer, 2013, pp. 250–265

[7] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. 2019. Practical GUI testing of Android applications via model abstraction and refinement. In Proceedings of the 41st International Conference on Software Engineering (ICSE).

[8] W. Choi, G. Necula, and K. Sen, “Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning,” in Proceedings of the 2013 ACM SIGPLAN International Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA 2013), 2013, pp. 623–640.

[9] Eliane Collins, Auri M. R. Vincenzi, Arilo C. Dias-Neto, and José Carlos Maldonado. 2021. Deep Reinforcement Learning based Android Application GUI Testing. In Brazilian Symposium on Software Engineering (SBES ’21), September 27-October 1, 2021, Joinville, Brazil. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474624.3474634

[10] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. 2012. Automated concolic testing of smartphone apps. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, 1–11.

[11] F. Yazdani Banafshe Daragh and S. Malek, "Deep GUI: Black-box GUI Input Generation with Deep Learning," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021, pp. 905-916, https://doi.org/10.1109/ASE51524.2021.9678778.

[12] Duling Lai and Julia Rubin. 2019. Goal-driven exploration for Android applications. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 115–127.

References

[13] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek. 2014. Evodroid: Segmented evolutionary testing of android apps. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 599–609.
[14] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective automated testing for Android applications. In Proceedings of the 25th International Symposium on Software Testing and Analysis. ACM, 94–105.
[15] Zhen Dong, Marcel Bohme, Lucia Cojocaru, and Abhik Roychoudhury. 2020. Time-travel Testing of Android Apps. In Proceedings of the 42nd International Conference on Software Engineering (ICSE). 481–492.
[16] Yavuz Koroglu, Alper Sen, Ozlem Muslu, Yunus Mete, Ceyda Ulker, Tolga Tanriverdi, and Yunus Donmez. 2018. QBE: QLearning-based exploration of android applications. In 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). IEEE, 105–115.
[17] Tuyet Vuong and Shingo Takada. 2019. Semantic analysis for deep Q-network in android GUI testing. In 31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019. Knowledge Systems Institute Graduate School, 123–128.
[18] Leonardo Mariani, Mauro Pezze, Oliviero Riganelli, and Mauro Santoro. 2012. Autoblacktest: Automatic black-box testing of interactive applications. In 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. IEEE, 81–90.
[19] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement Learning Based Curiosity-Driven Testing of Android Applications. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (Virtual Event, USA) (ISSTA 2020). Association for Computing Machinery, New York, NY, USA, 153–164. https://doi.org/10.1145/3395363.3397354
[20] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. 2021. Deep Reinforcement Learning for Black-Box Testing of Android Apps. ACM Trans. Softw. Eng. Methodol. 01, 1, Article 000 (January 2021), 26 pages. https://doi.org/10.1145/1122445.1122456
[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015).
[22] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 (2018).
[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018).

Thank You

Any Questions?