Testing By and Of AI
SFWR ENG 3S03: Software Testing

Alicia Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material adapted from Dr.I.David

AI and Machine Learning (ML)

- ML: the process of inferring statistical patterns
  - These patterns are very good at mimicking human reasoning
- AI: any sort of in-silico 'intelligence'
  - Old days: typically, some sort of logic-based framework (e.g., symbolic AI, inductive reasoning: models defined by the human)
  - Nowadays: typically, the artifact of machine learning (e.g., neural AI, deductive reasoning: from data to models)
  - Why not both? (e.g., neuro-symbolic AI: systems that learn, reason, and make decisions)

Why AI/ML in Testing?

- AI/ML is useful for automating and improving the testing process
- However, testing AI/ML systems presents unique challenges that require specialized strategies

Testing by AI vs. of AI

- Testing by AI: Using AI/ML tools to automate traditional software testing tasks
  - [DDB+19]
- Testing of AI: Ensuring AI systems perform correctly and ethically
  - [BK20]
  - [MFBF+22]

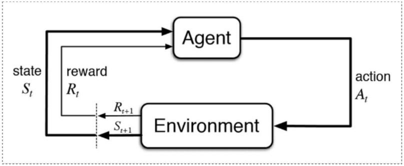
Traditional ML Techniques

- Supervised Learning: uses labeled data for training
    - Goal: predict outcome on unseen new data
- Unsupervised Learning: learns patterns in unlabeled data
    - Goal: clustering similar data points
- Reinforcement Learning: learns by interacting with an environment and receiving rewards or penalties
    - Goal: make decisions

Other ML Techniques

- Deep Learning (DL): uses neural networks with many layers, typically applied in supervised or unsupervised contexts
  - Used for: handling large amounts of data and complex tasks
- Specialized Models in ML
  - PINNs (Physics-Informed Neural Networks): used to solve PDEs by incorporating physical laws as part of the learning process
  - CNNs (Convolutional Neural Networks): specific DL models, used in image and video processing tasks
  - LLMs (Large Language Models): designed to understand and generate human language

Reinforcement Learning Process

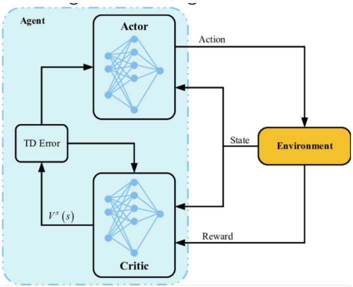


Source: [WS20]

- Agent: Learns and makes decisions
- Environment: System agent interacts with
- State S_t: Current state of the environment
- Action A_t: Move the agent makes
- Reward R_t: Feedback given for an action
- Policy π: Strategy that determines actions
- Value function: Measures long-term reward for a state

- Exploration: trying new actions to discover their effects
- Exploitation: using known actions that yield high rewards
- Trade-off
  - Too much exploration can be inefficient
  - Too much exploitation can miss better strategies
- Balance, e.g., ϵ-greedy
  - Exploration: the agent chooses a random action with probability ϵ
  - Exploitation: the agent chooses the action with the highest estimated reward with probability $1 - \epsilon$
  - Decay: over time, ϵ is reduced to shift focus from exploration to exploitation

- Policy-based: learn a policy $\pi(a|s)$, maps states to actions
- Value-based: learn the value of actions (e.g., Q-Learning, DQN)
- Actor-Critic Methods
  - Actor: learns the policy; short-term goals
  - Critic: evaluates the policy; guides Actor; long-term goals



Testing By RL

- Test case selection and prioritization
- Automated Test Generation
- Fuzzy Testing
- Exploratory Testing in Model-Based Testing
- Verification

Test case selection and prioritization

- Goal: Run the most valuable tests first, e.g., the ones that are most likely to fail
- Process
    - Agent learns from historical test results and code changes
    - Uses feedback (e.g., fault detection and coverage) as rewards
    - Continuously adapts as code evolves in the project
- Benefits: Reduces regression testing time while maximizing bug discovery
- Common algorithms: Q-learning or Deep Q-Networks (DQN)

Automated Test Generation

- Goal: generate effective test cases, especially for GUI or API testing
- Process
  - Agent explores the software interface
  - Observes GUI states and actions (e.g., clicks, swipes, inputs)
  - Learns which action sequences trigger new states or crashes
  - Reward: based on code coverage or fault discovery
- Benefits: Reduces manual effort in test case writing
- Common algorithms: Deep RL

Fuzzy Testing

- Goal: Focuses fuzzing effort on promising input areas
- Process
  - Learns which input patterns increase code coverage or crashes
  - State = current input/coverage
  - Action = mutate or try new input
  - Reward = new path explored or vulnerability found
- Benefits
  - Binary testing or embedded systems, e.g., IoT
  - Enhances fuzzers by guiding input mutation strategies

Exploratory Testing in Model-Based Testing

- Goal: guide exploration strategies to efficiently discover critical paths
- Process
  - Agent navigates state-transition models (e.g., UML)
  - Learns which paths are likely to reveal faults
  - Replaces random or exhaustive exploration with smart exploration
  - Reward: coverage, reaching rare states, fault detection
- Benefits
  - Safety-critical or embedded system testing
  - Reduces test suite size & maintains test effectiveness

Verification: calibration of Static Analysis (SA) rules

- State: current configuration of SA rules; context about the project (codebase size, language, history); recent feedback (ignored or fixed warnings)
- Action: enable/disable rules; adjust thresholds/severity levels; tune analysis depth or sensitivity
- Reward: fewer false positives (e.g., ignored warnings); higher developer engagement (e.g., fixed warnings); shorter analysis time without reducing useful results
- Benefits: find a balance between usefulness and overhead; continuously adapt to project evolution or team preferences

Challenges
Ipause

- Exploration vs. Exploitation balance
- Non-Deterministic Behavior: RL models can have inherent variability in their outputs
- Reward Design: Properly designing the reward function to align with desired system behavior is crucial for effective testing
- Scalability: Training RL models to handle complex, large-scale systems can require significant resources and time

Physics-Informed Neural Networks

- Solve partial differential equations (PDEs) by incorporating physical laws
- Work well with sparse or noisy data, improving robustness
- Generalize across different boundary conditions and parameter settings

PINNS Applications

- Fluid dynamics: solving Navier-Stokes equations for aerodynamics and turbulence modeling (e.g., drag forces)
- Heat transfer: modeling thermal diffusion and heat conduction problems
- Structural mechanics: simulating stress-strain behavior in materials
- Medical imaging: predicting biological processes based on partial observations

Challenges

- PINN methods are still in the early stage of maturity
- Training instability: require careful hyperparameter tuning; may struggle with complex PDEs
- Scalability: computational cost increases for high-dimensional problems
- Accuracy: hybrid approaches exist to combining PINNs with numerical solvers

Model-based Testing

- Testing for Cyber-Physical Systems
  - Model the expected physical behavior of the system
  - Test whether the software controller leads to physically plausible behavior
- Test oracles for physics-based systems where exact outputs aren't known (common in embedded systems or IoT)
  - PINN approximates what "should" happen based on physics
- Simulate the physical counterpart in a digital twin
  - Comparing simulated (PINN) behavior to the actual software-controlled device in real-time

Large Language Models (LLMs)

- DL models trained on large datasets to understand and generate human language
- Language modeling: given the previous context, predict the next word/token in a sequence
  - LLMs learn to model token sequence directly, using NNs
- Naturalness hypothesis
  - Source code is repetitive and predictable, much like natural language
  - Justifies applying language modeling techniques to code

Validation by LLMs

- Generate test cases from code or specifications
  - LLMs read source code, comments, and requirements and generate tests
- Natural Language to Test Automation Scripts
  - From plain English (e.g., "Test login fails when the password is wrong"), LLMs generate corresponding scripts
  - Bridge the gap between non-technical QA engineers and automation frameworks
- Static Analysis and Code Review Assistance
  - LLMs can spot anti-patterns or poor test coverage, missing assertions, overly generic tests, unused mocks, hardcoded values, flaky test smells

Validation by LLMs (cont'd)

- Generate Test Data
  - Realistic sample data (e.g., names, addresses, configs)
  - Edge cases or malformed inputs (e.g., fuzzy-style tests)
  - Synthetic datasets for ML testing or simulations
- Improve or Refactor Existing Tests
  - Enhance test readability or modularity
  - Add meaningful assertion messages
  - Refactor repetitive test into fixtures/test builders
- Analyze Test Results and Logs to
  - Summarize root causes
  - Suggest likely fixes
  - Correlate with recent code changes

Challenges

- LLMs can hallucinate incorrect tests if not carefully reviewed
- They may not understand complex domain-specific logic
- They work best when paired with a developer/tester in the loop

Of course, you know this very well at this point, you worked with Gen AI in this course!

- Testing Of AI: Requires special considerations
- AI: a very special type of software
  - Large
  - Non-deterministic
  - Non-transparent

Challenges in Testing AI/ML

- Lack of clear oracle
  - In many ML tasks (e.g., image recognition, NLP), there's no obvious correct output
  - Outputs are often probabilistic/involve uncertainty: hard to say definitively whether a result is right
  - e.g., is a model 80% confidence in "dog" good enough? What if the image is ambiguous?
- Non-deterministic behavior
  - ML models (especially DL) can produce different outputs depending on
    - Initialization
    - Stochastic training steps (e.g., dropout, data shuffling)
  - This makes regression testing and reproducibility harder

Challenges in Testing AI/ML (cont'd)

- Testing for generalization, not just functionality
  - ML systems are evaluated on how well they generalize to new data
  - One cannot just test for "does it return expected output"
  - Test on diverse inputs to catch issues like overfitting or data bias
- Bias, fairness, and explainability: test for ethical and societal impacts
  - Bias against certain groups
  - Unintended consequences
  - Require new types of metrics and tests (e.g., fairness metrics, SHAP/LIME explanations)

Challenges in Testing AI/ML (cont'd)

- Testing the Training Pipeline
  - ML systems are pipelines: data → preprocessing → model → postprocessing
  - Defects can occur anywhere in this chain (e.g., a defective data transformation could ruin accuracy)
  - Testing the entire pipeline (not just the final model) is crucial
- Lack of Mature Testing Tools
  - Traditional testing tools (unit tests, code coverage) are not designed for ML behavior
  - ML testing is still evolving, and best practices/tools vary widely by domain

- AI/ML are transforming software testing
  - There is no modern software engineering (including testing) without AI
  - How confident are you in your AI/ML skills?
- We scratched the surface here
  - There is much more to learn and understand
  - You should understand the basics, classics, and the new opportunities
  - The onus is on you to pick up all this knowledge and use it like a responsible and creative engineer should

You are expected to lead in an industry that is currently transforming: you, too, will become transformers of our industry soon

- Don't be a stranger: send me an email every once in a while, and educate me!

References I

Houssem Ben Braiek and Foutse Khomh.
On testing machine learning programs.
Journal of Systems and Software, 164:110542, 2020.

Vinicius HS Durelli, Rafael S Durelli, Simone S Borges, Andre T Endo, Marcelo M Eler, Diego RC Dias, and Marcelo P Guimarães.
Machine learning applied to software testing: A systematic mapping study.
IEEE Transactions on Reliability, 68(3):1189–1212, 2019.

References II

Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner.
Software engineering for ai-based systems: a survey.
ACM Transactions on Software Engineering and Methodology (TOSEM), 31(2):1–59, 2022.

Dasong Wang and Roland Snooks.
Artificial intuitions of generative design: an approach based on reinforcement learning.
In *The International Conference on Computational Design and Robotic Fabrication*, pages 189–198. Springer, 2020.