Requirements Engineering Retrospectives on Machine ~~Learnin~~g Learners and Verification

Joseph Scott

August 1, 2023

Outline

Introductions

1. My name is Joseph Scott. I am a final year Ph.D. Student in AI
   1. Advisors: Vijay Ganesh, Derek Rayside
2. What is AI?
   1. AI/AGI means anything to many different people and has different definitions
   2. One thing that is invariant amongst scholars: AI can reason
   3. My opinion: A truly general AI system must be able to do all *orthogonal* forms of reasoning.
3. How do we reason? Some prominent ones
   1. Deductive Reasoning – conclusions are derived from a set of premises that are considered to be true.
   2. Inductive Reasoning – conclusions are drawn from specific observations or examples.
   3. Others (e.g., abductive, common sense, cultural and moral, analogical, etc.).
4. During my Ph.D., I have worked mostly in AI systems (not philosophy ☺). How do modern AI systems realize these blackbox forms of reasoning?

Modern Logic Solving – Deductive AI (1/3)

A logic solver is a tool that can determine whether or not a logical formula has a solution.

The input to a logic solver is a formula (i.e., written in propositional logic, first-order logic).

The output of a logic solver is a satisfying assignment to all of the inputs of a formula, if one exists, else a certificate of unsatisfiability (or unknown).

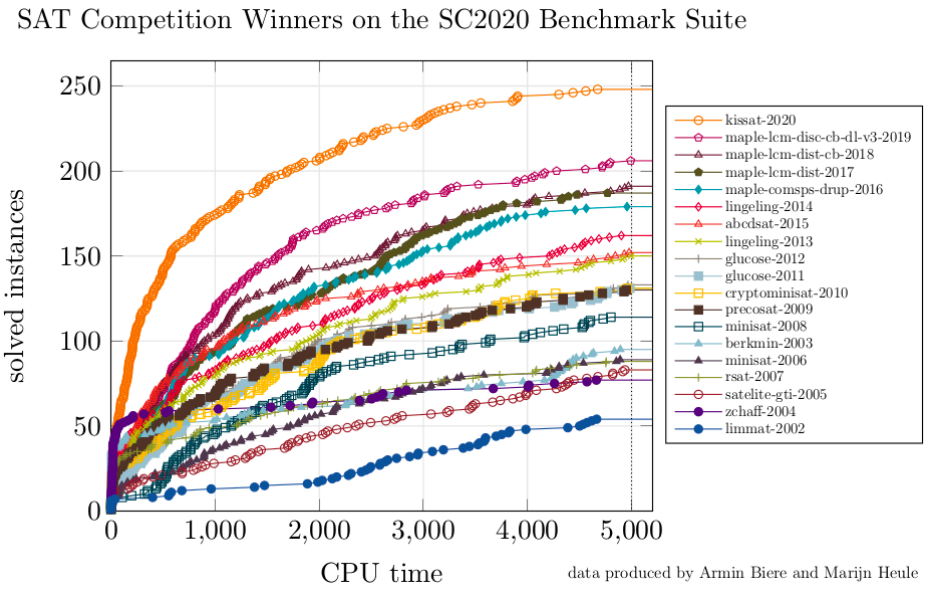
Determining the satisfiability of a formula lies at the heart of automated theorem proving, verification, and artificial intelligence.

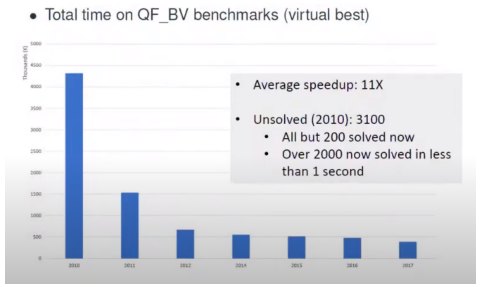
$$\Sigma \vDash \psi \iff (\bigwedge_{\phi \in \Sigma} \phi \wedge \neg\psi) \text{ is } \texttt{SAT}$$

However, solvers suffer from poor worst-case performance, often observing exponential runtimes (or worse) or even undecidable.

Modern Logic Solving – Deductive AI (2/3)

1. Logic solvers have observed a significant empirical improvement in recent years
2. This is despite exponential runtimes or even undecidable in the worst case
3. While some select hand-crafted formulas remain to be quite challenging to solve, modern solvers can scale to industrial-grade problems:
   1. Intel Chip Design
   2. AWS Zelkova





Modern Logic Solving – Deductive AI (3/3)

1. Logic solvers frequently overcome worst-case complexity in practice
2. This is in part due to the ability of a solver to exploit:
   1. Syntactic structure and patterns
   2. Heuristics based on inductive principles from performance histories
3. Recently, breakthroughs in machine learning and pattern recognition has had impacts in many fields
4. A major question of my PhD: Can we use machine learning to exploit these further?

Modern Machine Learning – Inductive AI

- **Inductive AI:** Instead of explicitly programming rules (deductive AI), modern machine learning algorithms learn patterns directly from data.
- **Software 2.0:** Machine learning represents a shift in programming paradigm, often termed as "Software 2.0". In this approach, systems are taught rather than explicitly programmed.
- **Key Components:**
  - *Data:* The "fuel" for inductive AI, used to learn patterns.
  - *Models:* The "engine" that learns from data, ranging from linear models to deep neural networks.
  - *Tasks:* The "direction" for learning, defined by a specific problem (e.g., classification, regression, reinforcement learning).
- **Advantages:**
  - *Flexibility:* Can adapt to new data, making them more resilient and dynamic than traditional software.
  - *Capability:* Can learn to perform complex tasks that are difficult to explicitly program.
- **Challenges:**
  - *Data Dependence:* The quality and quantity of available data can significantly impact performance.
  - *Interpretability:* Software 2.0 systems often operate as "black boxes", making understanding and debugging them difficult.

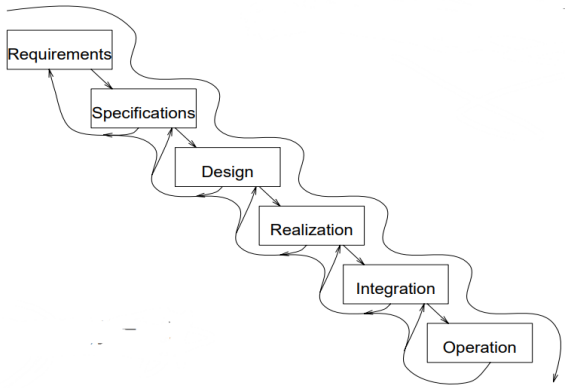
Motivation: Why a Retrospective?

- **Evolution:** As we transition from Software 1.0 to Software 2.0, understanding past practices gives context to our present challenges.
- **Learning from the Past:** The successes and failures of requirements engineering in Software 1.0 provide valuable lessons to inform the development of Software 2.0.
- **Preparation:** A thorough understanding of the state of requirements engineering in Software 1.0 helps us prepare for the complexities and new challenges posed by Software 2.0.
- **Improvement:** Identifying the gaps in our current practices can lead to the development of new methodologies better suited for Software 2.0's data-driven paradigm.

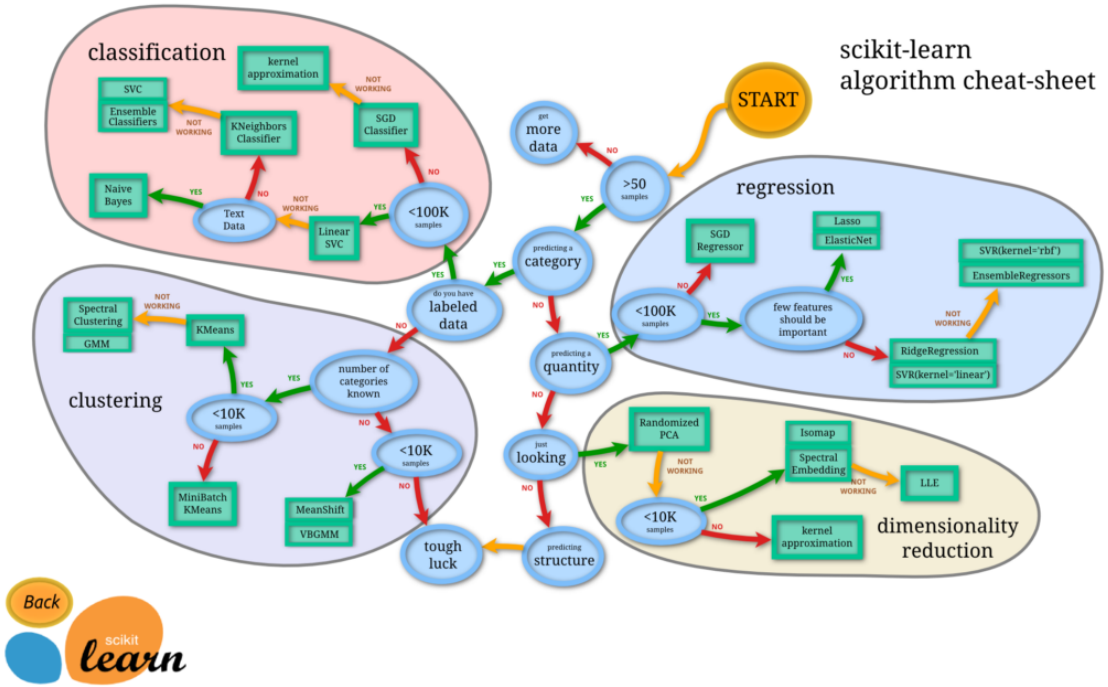
Outline

The Waterfall Model – Traditional Software(1/7)

1. **Requirements** – Define what the client needs in the software.
2. **Specification** – Translate requirements into technical terms for the development team.
3. **Design** – Architect the software, deciding how it will operate, its interfaces, and data structures.
4. **Realization** – Code the software, ensuring each component works as per design.
5. **Integration** – Combine the individually tested components into a single system, ensuring they function correctly together.
6. **Operation** – Work required post-deployment to keep the software running smoothly, including bug fixes, updates, and improvements.

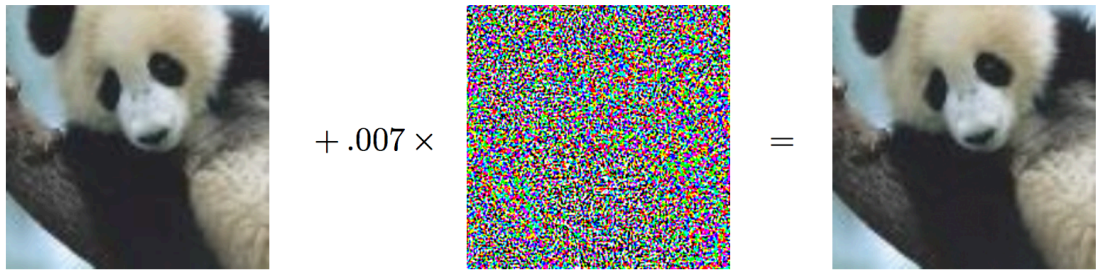


The Waterfall Model for ML – Requirements (2/7)



- What type of problem(s) are you trying to solve?
- Classification? Regression? Clustering?
- What data do you have? What data can you collect?

The Waterfall Model for ML – Specifications (3/7)

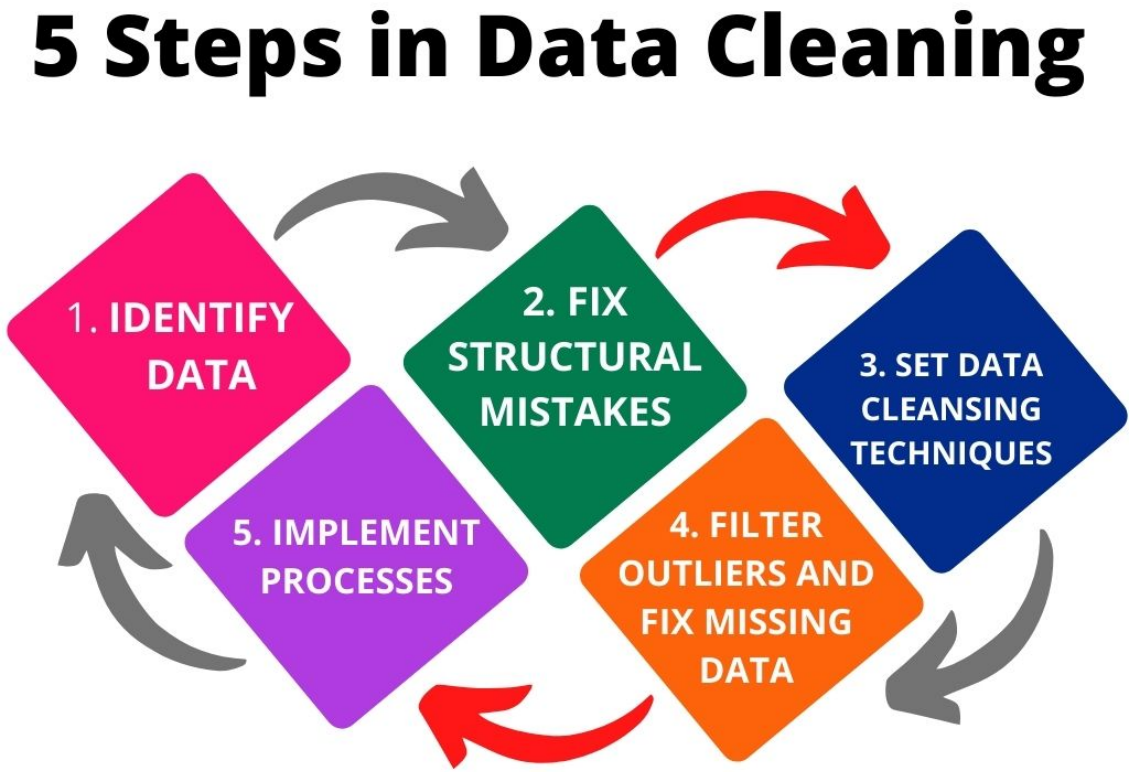


- Define success metrics and data needs.
- Consider model interpretability, robustness, scalability, and latency.
- Perform exploratory data analysis to guide subsequent steps.
- Formal Methods?

The Waterfall Model for ML – Design (4/7)

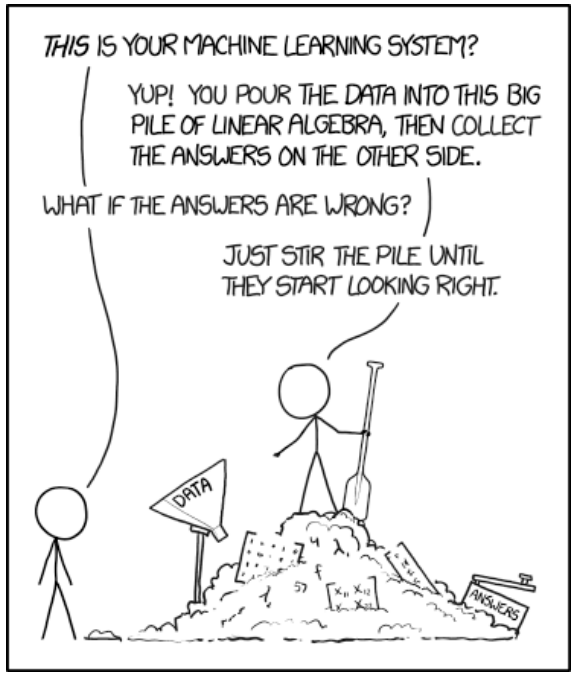
- Plan ML model architecture and data pipeline.
- Choose the type of ML model, cost function, and optimization method.
- Account for data privacy, model fairness, and explainability.

The Waterfall Model for ML – Realization (5/7)



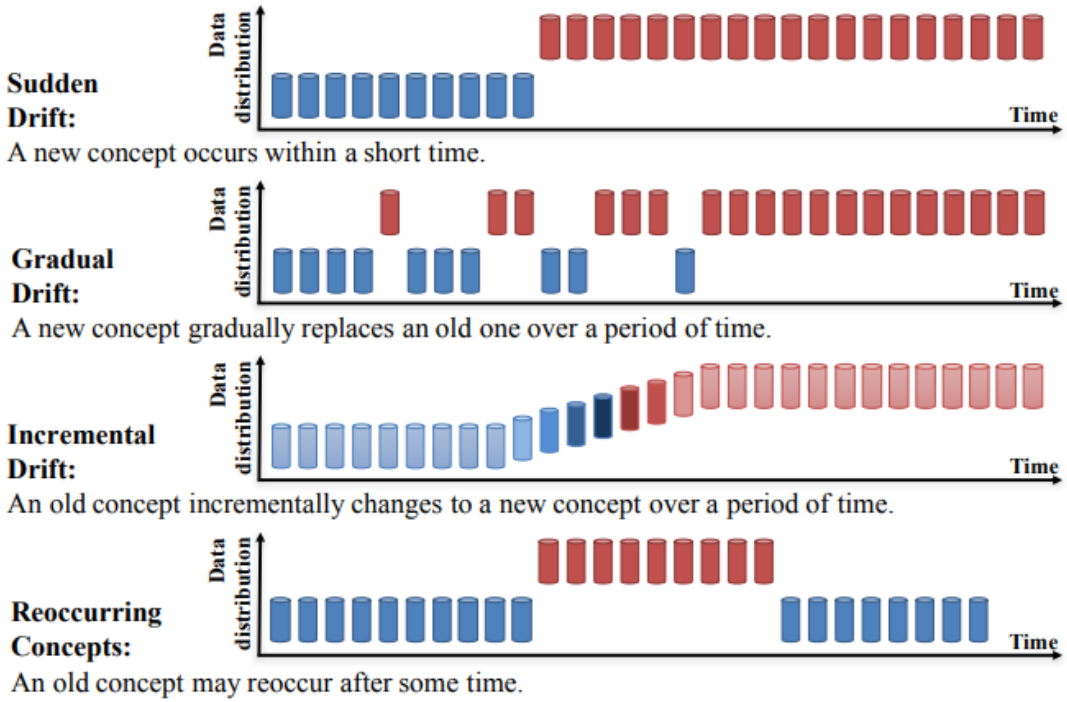
- Gather, clean, and preprocess data.
- Handle missing values, categorical variables, and create new features.
- Train the model using an optimization algorithm.

The Waterfall Model for ML – Integration (6/7)



- Deploy the ML model into the existing system.
- Validate the model with test data.
- Ensure ML system works well with other software components.

The Waterfall Model for ML – Operation (7/7)



- Monitor model performance and retrain as needed with new data.
- Handle model drift or changes in the data distribution.
- Maintain the ML system to ensure performance over time.

Other RE Models

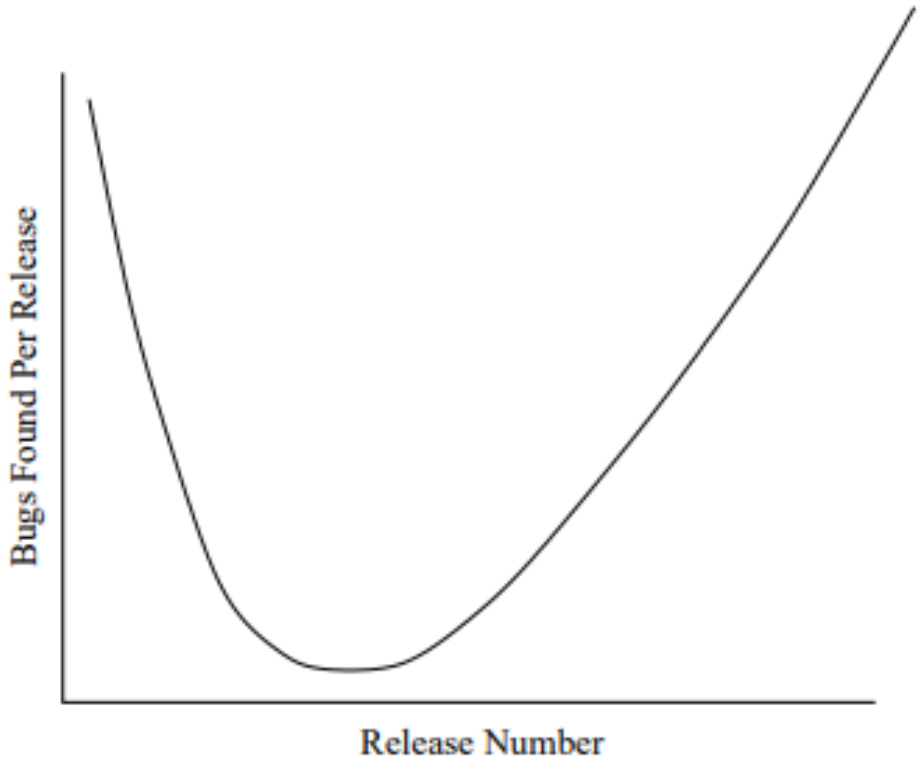
1. Spiral Model
    1. Similar to waterfall, but with more emphasis on a more systematic and iterative approach to risk management.
    2. In ML, each iteration allows for refinement of model, incorporating feedback, and reassessing risks, including evaluation of model performance, and potential biases.
2. V Model
    1. Requires a strong emphasis on requirement definition and corresponding validation procedures from the start, reducing the risk of error and costly fixes later on.
    2. In ML, careful upfront analysis is crucial to ensure the right data is collected and that the model's outputs will actually meet the project's needs. Likewise, validation procedures must be robust to handle the inherent uncertainty in ML outputs.
3. Agile
    1. Emphasizes iterative development with constant feedback, close collaboration, and adaptability over strict planning.
    2. In ML, Agile can allow for quick integration of new data, adjustments based on model performance, and adaptability to changing project needs or data environments.

Which RE model is best for ML?

1. All models have value, bringing structure (Waterfall), risk mitigation (Spiral), or flexibility (Agile) to ML projects.
2. Yet, there is no silver bullet. Every ML project is unique, requiring a tailored approach or even a hybrid of these models.
3. The 'no silver bullet' concept reminds us that success in ML requirements engineering comes from a thoughtful and adaptable approach, rather than rigid adherence to one single model.

Belady-Lehman Phenomenon in ML

- An observation of software becoming more complex and less reliable over time due
- Why it applies to ML:
  - Model's performance can degrade due to concept drift.
  - Users better understand and exploit what the model is optimizing for
- Why it might not apply to ML:
  - ML models can adapt to changes through learning.
  - Periodic retraining can mitigate aging effects.



Information Hiding in ML

- A principle that hides implementation details to allow for changes in abstraction without affecting its users.
- Why it applies to ML:
  - Protects sensitive data (e.g., weights, training and label data) from unnecessary exposure.
  - Enhances system safety and privacy by restricting access to critical parts.
  - Allows flexibility in modifying model internals without disrupting overall system.
- Why it might not apply to ML:
  - Complete hiding of weights might limit model interpretability and tuning.
  - In distributed learning environments, data privacy and hiding becomes more complex.

Emotional Requirements Engineering in ML

- **Definition:**
  - An approach to requirements engineering attentive to social, emotional, political, and cultural factors among users.
- **Why it applies to ML:**
  - Helps prevent biased model behavior (e.g., an image recognition system wrongly classifying individuals of certain races or genders due to biased training data).
  - Facilitates user acceptance and ethical use of ML systems (e.g., ensuring a language translation model doesn't favor any cultural or social group).
- **Challenges:**
  - Emotional and cultural factors can be hard to interpret and model; for example, a sentiment analysis model might misinterpret text from different cultural contexts.
  - Emotional RE requires deep understanding of user emotions and societal nuances, which may exceed the current capabilities of ML models.

Formal Methods in ML

- **Definition:**
  - FMs are mathematically rigorous techniques and tools for specification, design, and verification of software and hardware systems.
- **Why they apply to ML:**
  - Improves model reliability by ensuring mathematical correctness (e.g., formal verification of neural network properties).
  - Facilitates rigorous reasoning and proof of consistency, reducing bugs and errors early in model design.
- **Why they might not apply to ML:**
  - Many ML concepts (e.g., "model interpretability") are hard to formalize, as von Neumann and Morgenstern observed.
    1. "There's no point to using exact methods where there's no clarity in the concepts and issues to which they are to be applied."
  - FMs primarily ensure internal consistency, but ML also deals with external validity - the alignment of model behavior with real-world phenomena. Formal methods may not adequately address this aspect.

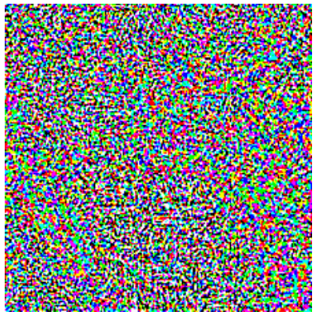
Outline

Motivation

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

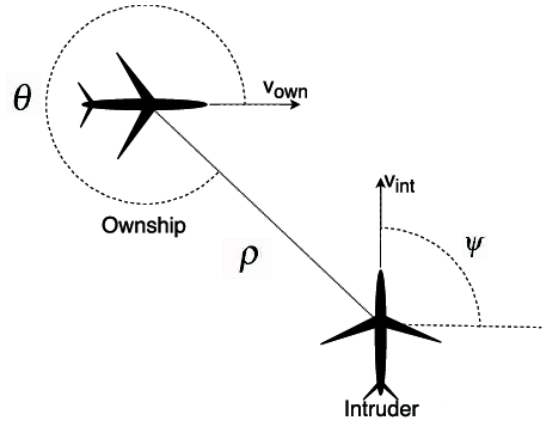
"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

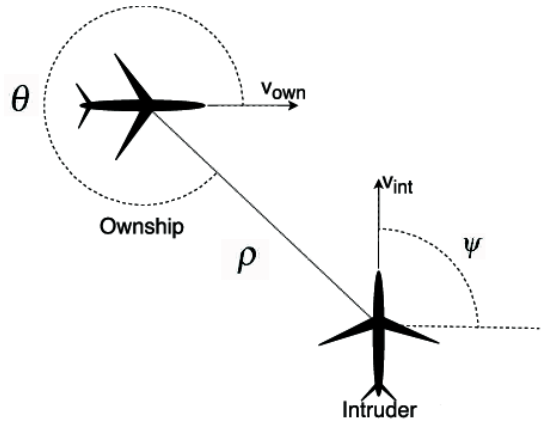
"gibbon"
99.3 % confidence

1. Advances in machine learning have lead to incredible new applications
2. However, they remain brittle, despite eager deployment into safety-critical-systems

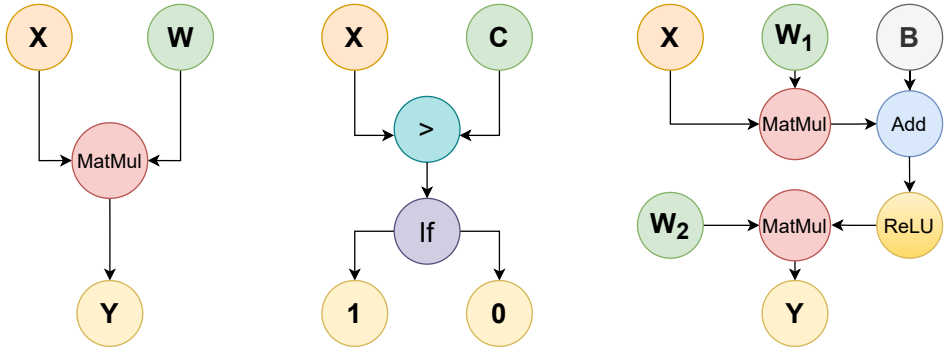


Motivation

1. Advances in machine learning have lead to incredible new applications
2. However, they remain brittle, despite eager deployment into safety-critical-systems



Verification of Neural Networks



1. Let C be a computation graph, representing a machine learning model
    1. Neural Networks
    2. Decision Trees
2. Let ψ be a linear specification ψ over the input/output behaviour of C.
3. The neural network verification question asks if ψ is valid over C ($C \vDash \psi$)

The Goose Tool

1. Goose is a solver for VNN-LIB benchmarks
2. Goose is a meta-solver – a solver designed to call subsolvers
3. Goose leverages three key synergetic techniques
   1. ML driven algorithm selection
   2. Probabilistic Satisfiability Inference
   3. Time Iterative Portfolio Deepening
4. We evaluate Goose in a VNN-COMP environment and see a 41.3% improvement over the 2021 competition winner.

Etymology: Named after the Canadian Goose, in honor of Canada

Goose Input/Output

1. **Input:**
   1. A computation graph (DNN)
      1. Open Neural Network Exchange (ONNX) format
   2. A specification
      1. Assertions on the DNN's input
      2. Assertions on the DNN's output
2. **Output:**
   1. A satisfying assignment such that the input and output constraints are satisfied
   2. A certificate of correctness/robustness

What are the requirements of ML?

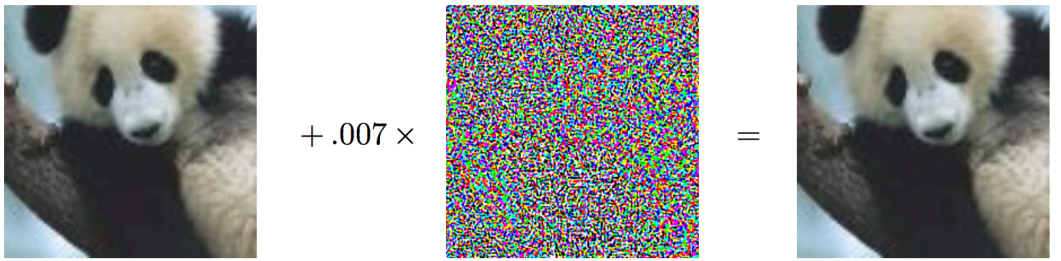
From my perspective, there are two sorts of specifications in this space.

1. General specifications
2. Instance Ad-Hoc specifications

Coming up with general specifications is hard. What are the requirements of ML from a training perspective?

1. Accuracy
2. Model Complexity (regularization)

Local Robustness



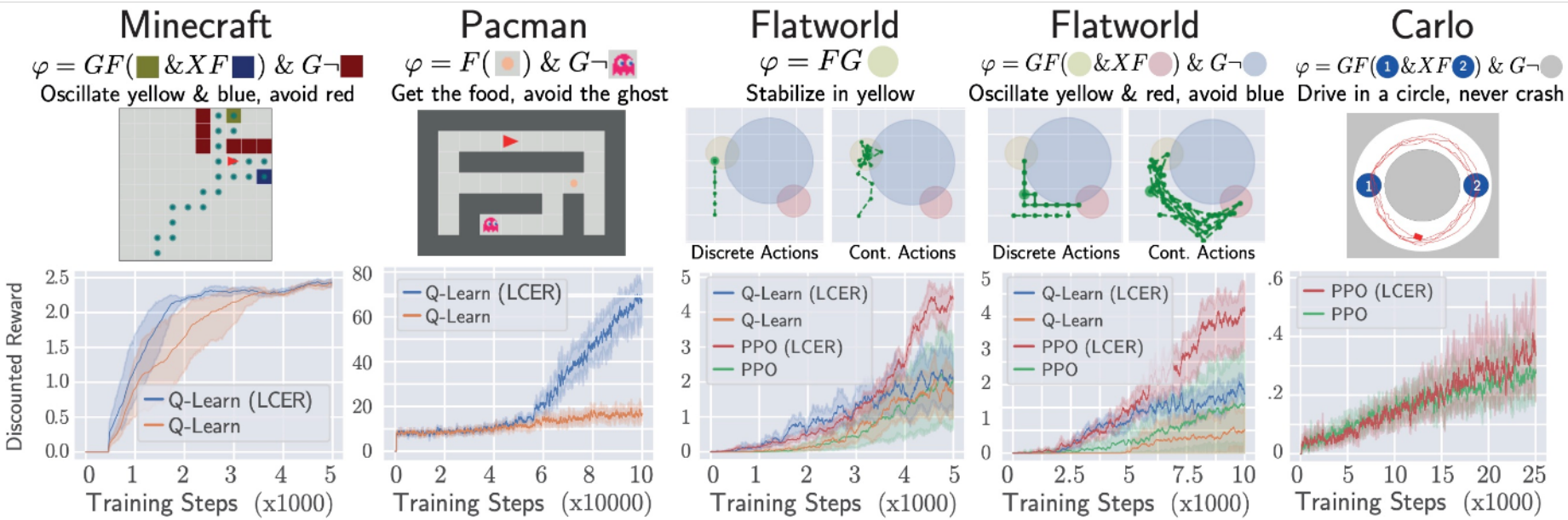
- **Definition:** Local robustness in machine learning refers to the invariance of a model's predictions in a small ϵ-ball around the input x. Mathematically, $\forall x' \in B(x, \epsilon)$, the model's prediction remains the same: $f(x) = f(x')$.
- **Significance:**
  - Ensures model stability: Minor perturbations should not drastically alter the model's predictions.
  - Critical for security: Helps protect against adversarial attacks.
- **Challenges:**
  - Trade-off with accuracy: Increased robustness can sometimes lower performance on clean data.
  - Complexity: Constructing robust models can be computationally expensive and require complex techniques.

Other General Requirements?

1. What are other general requirements?
2. This question seems very difficult.
3. Going back to Von Neuman
    1. "There's no point to using exact methods where there's no clarity in the concepts and issues to which they are to be applied."

Ad Hoc Examples

When your X,Y have fixed semantics (i.e., tabular data, not perception) its often possible to engineer specifications in an Ad-Hoc fashion



Outline

Conclusion

- We explored the retrospective of requirements engineering in the context of machine learning.
- We discussed different models and approaches, including the Waterfall Model, Spiral Model, V Model, and Agile, and their applicability to machine learning.
- We also highlighted specific challenges and considerations in ML requirements, such as local robustness, information hiding, emotional requirements engineering, and formal methods.
- While there is no one-size-fits-all solution, understanding these requirements and considering them in the development process can contribute to the success and reliability of machine learning systems.
- Moving forward, it is essential to continue exploring and refining requirements engineering practices to address the unique characteristics and challenges of machine learning.