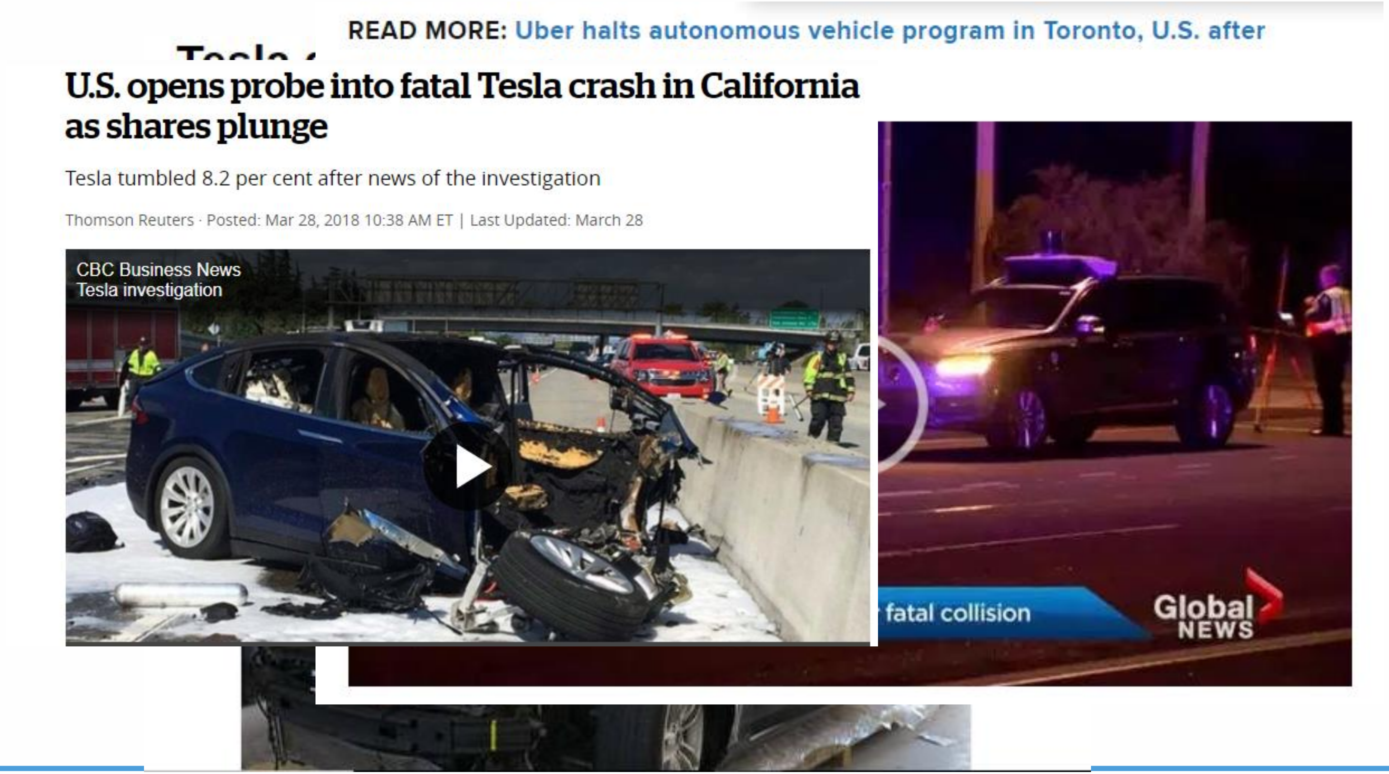
Machine Learning and Safety in Automotive Software

Rick Salay

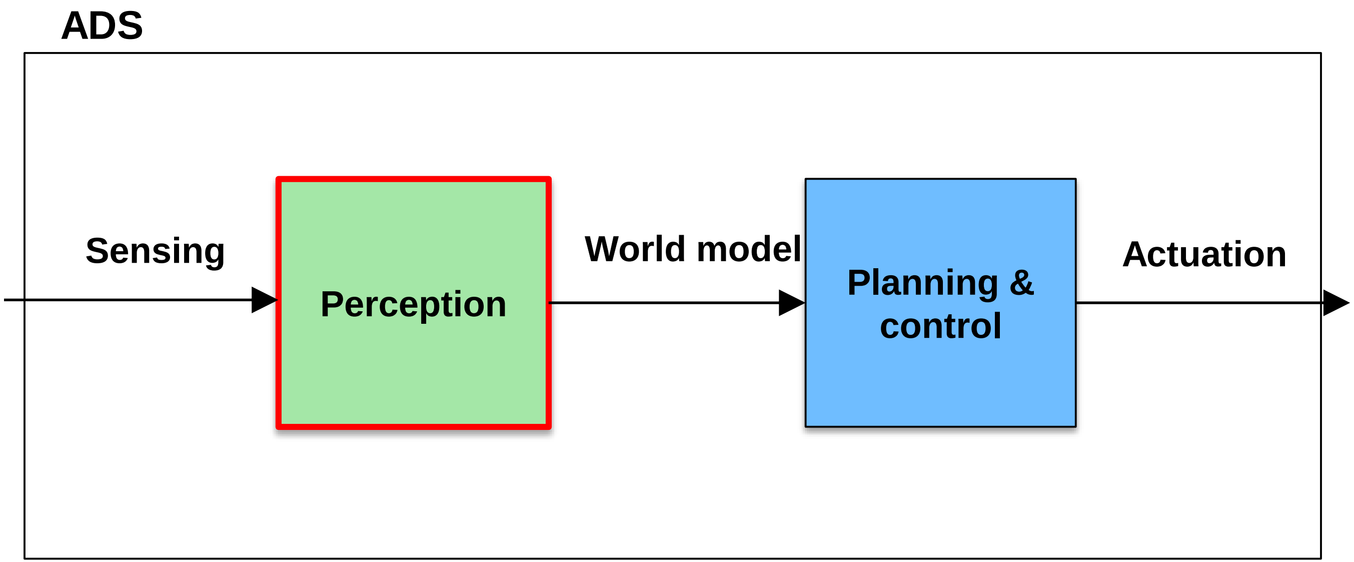
February 4, 2019


READ MORE: Uber halts autonomous vehicle program in Toronto, U.S. after
U.S. opens probe into fatal Tesla crash in California as shares plunge
Tesla tumbled 8.2 per cent after news of the investigation
Thomson Reuters · Posted: Mar 28, 2018 10:38 AM ET | Last Updated: March 28
CBC Business News
Tesla investigation
fatal collision
Global NEWS

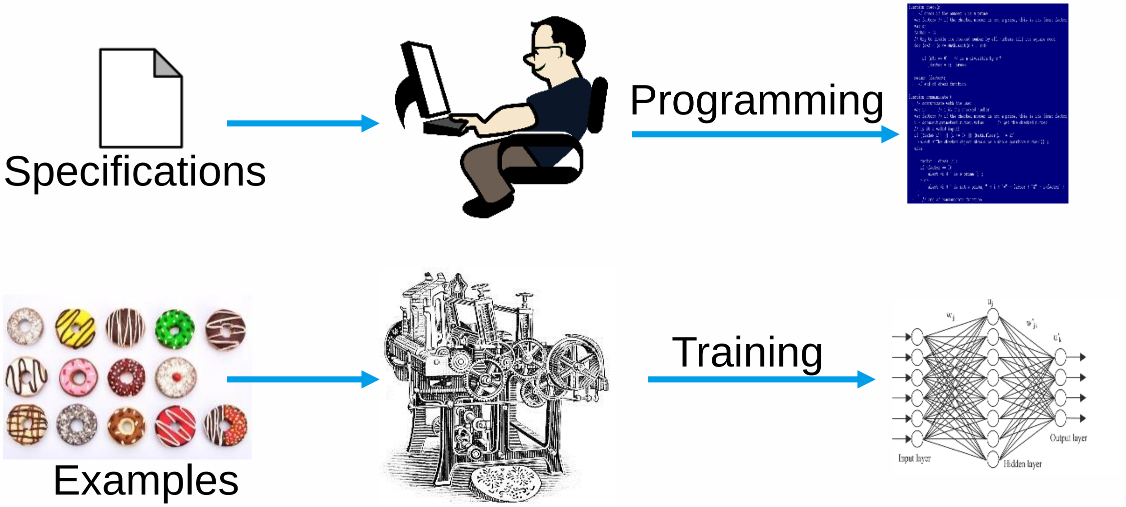
Agenda – two strategies for safety assurance of ADS and ML

1. **Hazard-based automotive safety standard (ISO 26262)**
   - Will focus on key ML obstacles to V&V
     - lack of specification
     - lack of interpretability
   - Will discuss research directions to address these
2. **Measurement uncertainty-reduction based (specifically for perception)**
   - Identifying factors contributing to uncertainty and methods to address them

Focus on Perception and Supervised Learning

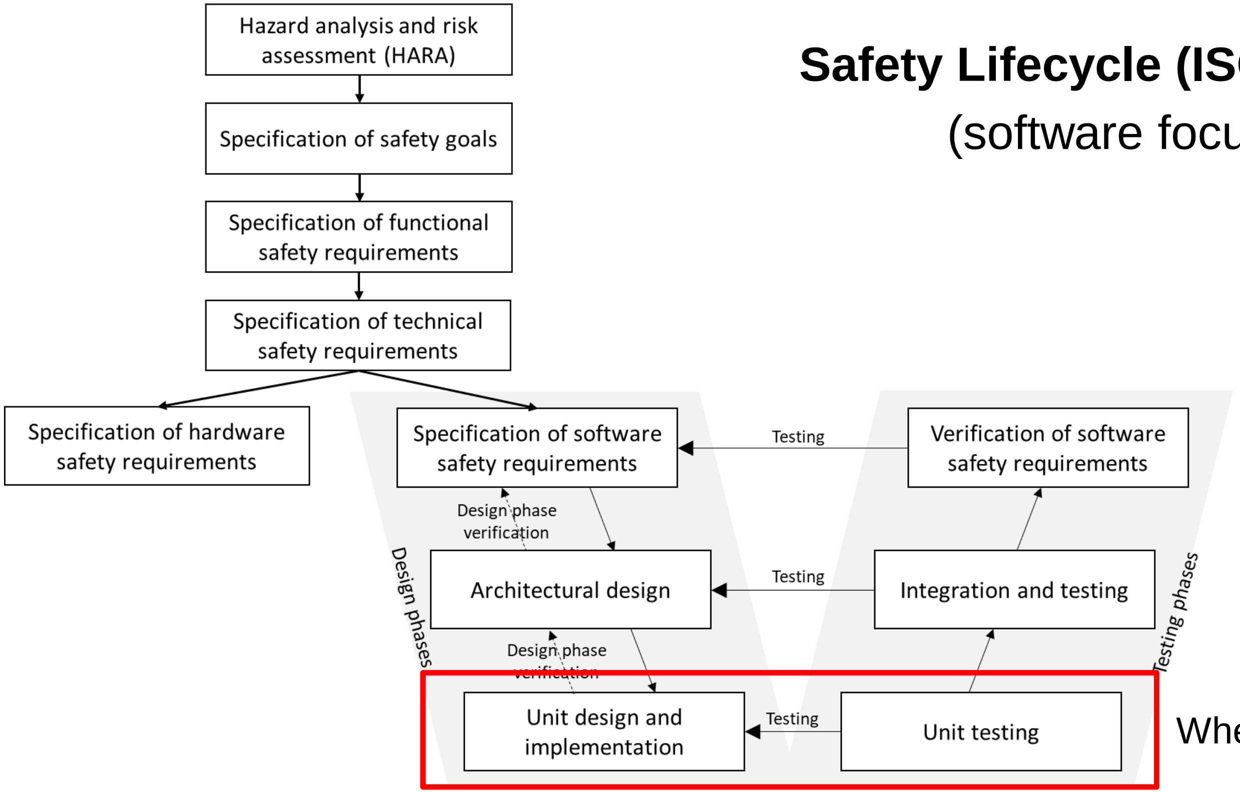


Two ways to implement software: Programming vs. Training (ML)

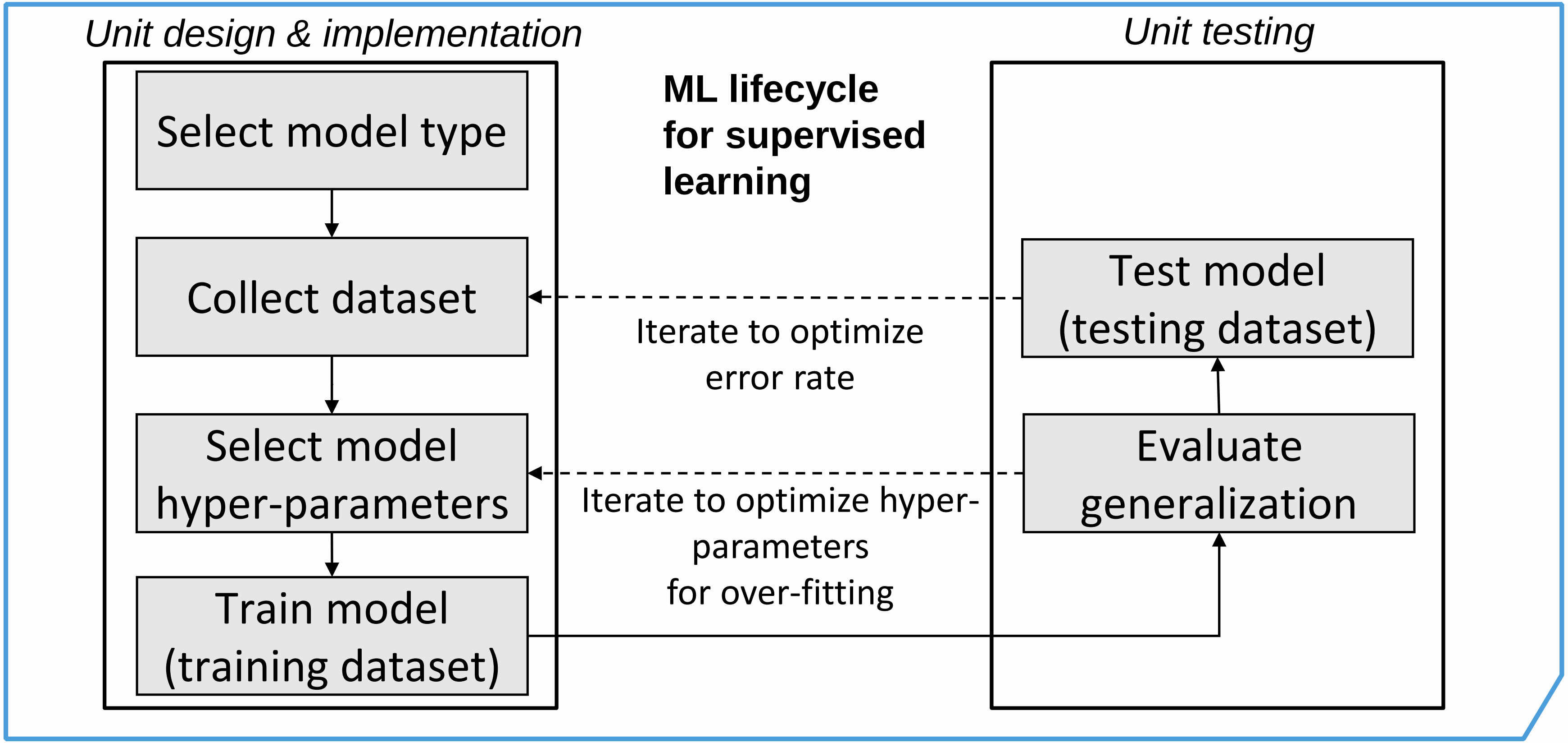


Safety through a hazard-based automotive safety standard (ISO 26262)

Safety Lifecycle (ISO 26262)
(software focus)



Where ML is used


Unit design & implementation
Unit testing
ML lifecycle for supervised learning
Select model type
Collect dataset
Select model hyper-parameters
Train model (training dataset)
Test model (testing dataset)
Evaluate generalization
Iterate to optimize error rate
Iterate to optimize hyper-parameters for over-fitting

ISO 26262 Approach to Software Safety

Recommends a *particular level of rigor* in developing safety critical software
- different levels for ASIL A-D
- consists of 83 software development techniques (34 at unit level)

Assumption: following the recommendations reduces residual risk of hazard due to SW failure to an acceptable level

Best Practices

Table 8 — Design principles for software unit design and implementation

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[ab]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[ab]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[abc]	++	++	++	++
1j	No recursions	+	+	++	++

[a] Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

[b] Methods 1g and 1i are not applicable in assembler programming.

[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

Verification

Table 9 — Methods for the verification of software unit design and implementation

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Testing

Table 11 — Methods for deriving test cases for software unit testing

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

[a] Equivalence classes can be identified based on the division of inputs and outputs, such that a representative test value can be selected for each class.

[b] This method applies to interfaces, values approaching and crossing the boundaries and out of range values.

[c] Error guessing tests can be based on data collected through a "lessons learned" process and expert judgment.

Fault Tolerance

Table 5 — Mechanisms for error handling at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Static recovery mechanism[a]	+	+	+	+
1b	Graceful degradation[b]	+	+	++	++
1c	Independent parallel redundancy[c]	o	o	+	++
1d	Correcting codes for data	+	+	+	+

a Static recovery mechanisms can include the use of recovery blocks, backward recovery, forward recovery and recovery through repetition.

b Graceful degradation at the software level refers to prioritizing functions to minimize the adverse effects of potential failures on functional safety.

c Independent parallel redundancy can be realized as dissimilar software in each parallel path.

Techniques

Unit Level

Best Practices	Prevent faults
Verification	Find and repair faults (and build confidence)
Testing	
Fault Tolerance	Live with faults

Assumes programmed software!

Techniques

Q: How well do ISO 26262 software recommendations apply to ML components?

Based on: Salay, Rick, and Krzysztof Czarnecki. "Using machine learning safely in automotive software: An assessment and adaption of software process requirements in iso 26262." *arXiv preprint arXiv:1808.01614* (2018).

Software technique classification

N/A – technique is not applicable to ML
Adapt – technique can be applied to ML with some adaptation
Use – technique can be used with ML as-is

Table 9 –

	Methods	ASIL A	ASIL B	ASIL C	ASIL D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Techniques

Best Practices	Prevent faults
Verification	Find and repair faults
Testing	
Fault Tolerance	Live with faults

Best Practices

Consist of coding guidelines, notation styles, principles

Table 8 — Design principles for software unit design and implementation

	Methods	ASI A	B
1a	One entry and one exit point in subprograms and functions[a]		
1b	No dynamic objects or variables, or else online test during their creation[a]		
1c	Initialization of variables		
1d	No multiple use of variable names[a]		
1e	Avoid global variables or else justify their usage[a]		
1f	Limited use of pointers[a]		
1g	No implicit type conversions[ab]		
1h	No hidden data flow or control flow[c]		
1i	No unconditional jumps[abc]		
1j	No recursions		+

[a] Methods 1a, 1b, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based (

[b] Methods 1g and 1i are not applicable in assembler programming.

[c] Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.

		D
1a	Natural languag	++
1b	Informal notatio	+
1c	Semi-formal not	++
1d	Formal notation	+

Mostly N/A

Table 1 — Topics to be covered by modelling and coding guidelines

	Topics	ASIL A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required.

[b] The objectives of method 1b are

— Exclusion of ambiguously defined language constructs which may be interpreted differently by different modellers, programmers, code generators or compilers.

— Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.

— Exclusion of language constructs which could result in unhandled run-time errors.

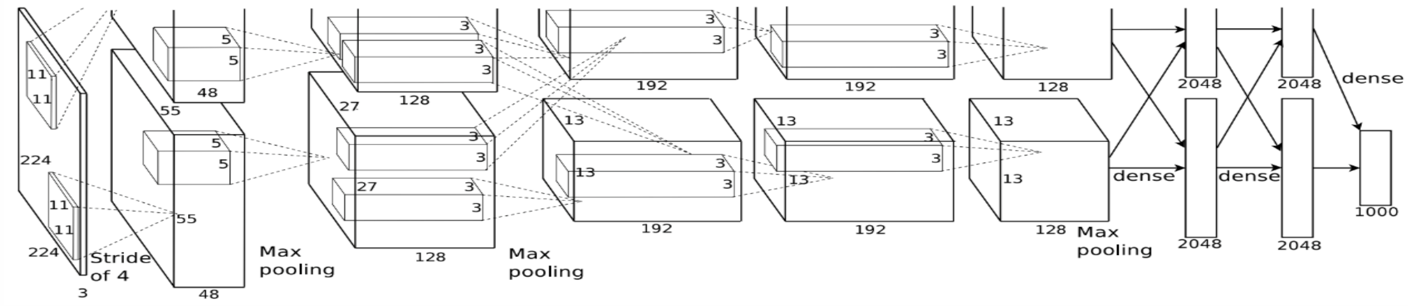
[c] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.

Strongly biased toward (imperative) programming languages!

What about ML-specific best practices?

ML has low maturity compared to traditional programming

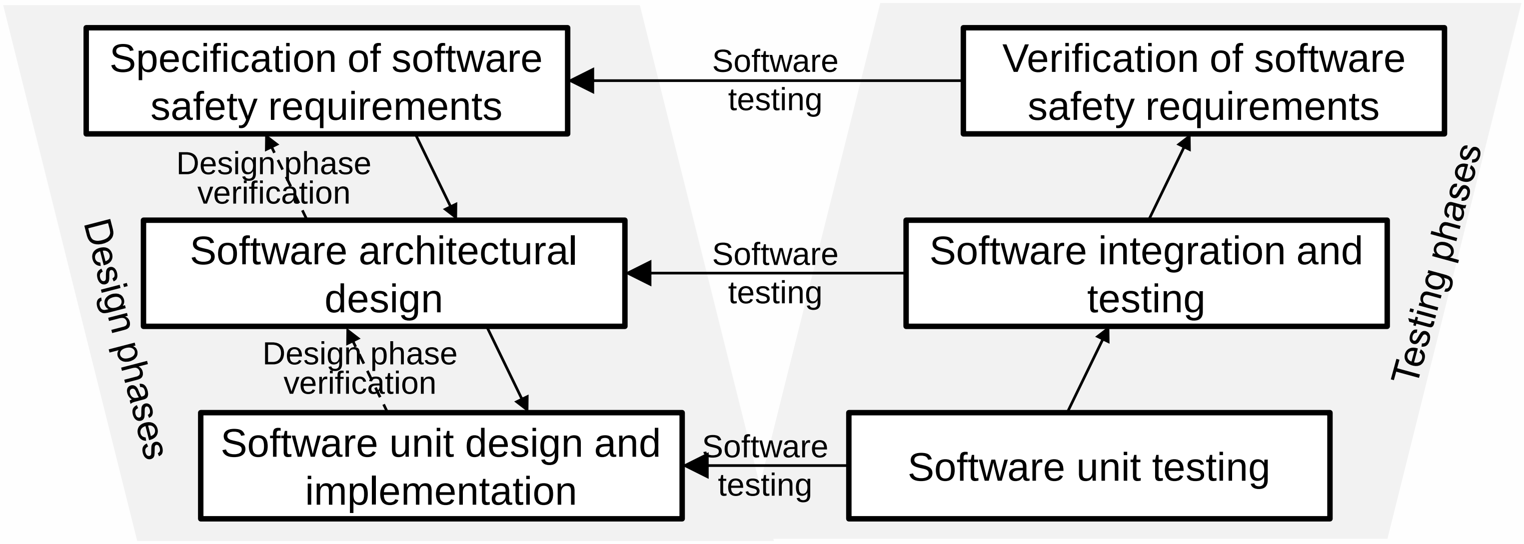
Best practices are emerging
E.g. standardized methods for deep neural networks



Techniques

Best Practices	Prevent faults
Verification	Find and repair faults
Testing	
Fault Tolerance	Live with faults

“V” Model of Software Development



Techniques that are adaptable to ML

Adapt: Static analysis of trained models is feasible
e.g., NN property checking via SMT

Table 9 — Method

1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Techniques directly applicable to ML

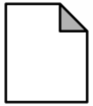
Use: Black box testing can be done on ML components

Table 10 — M

	Methods	ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

Complete Specification Assumption

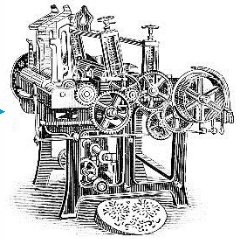
"V" model assumes that complete specifications exist and are sufficiently detailed



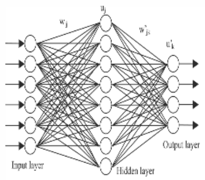
Specifications

Programming

Examples



Training

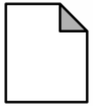


No specification, only training dataset

- Dataset always incomplete

Interpretability Assumption

Many verification and testing techniques require that the implementation be human understandable (interpretable)

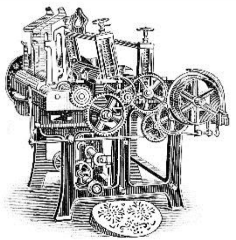


Specifications

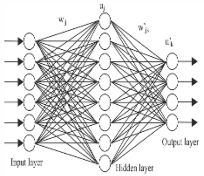
Programming

✓ Written by humans

Examples



Training



✗ Not interpretable

Impact of specification and interpretability on verification and testing techniques (unit level)

Mean (Std dev) across ASILs
Perfect score is 1.0

	Verification	Testing
Not interpretable	(0.01)	(0.01)

Summary

- Specification is important for verification and testing
- Interpretability is critical for verification

Complete Specification Assumption

Is the complete specification assumption reasonable?
Not for advanced functionality: ADAS, ADS

Hard to specify:
Perception tasks

e.g., What are complete necessary and sufficient conditions to identify a pedestrian?

Hard to specify:
Planning tasks in an open environment

Complete Specification Assumption

Is the complete specification assumption reasonable?
Not for advanced functionality: ADAS, ADS!

No specification => hard to direct a programmer

Conclusion: Machine Learning is preferred approach!

No specification => nothing to verify against!

Complete Specification Assumption : How to address?

Some specifications with ML components still possible: two kinds

Partial behavioural specifications (PBS)

Assumptions

e.g. illumination > 15000 lux

Necessary/Sufficient conditions

e.g., pedestrian < 9 feet tall

Invariants, equivariants

e.g., classification is invariant to rotation

Complete *data* specifications (DS)

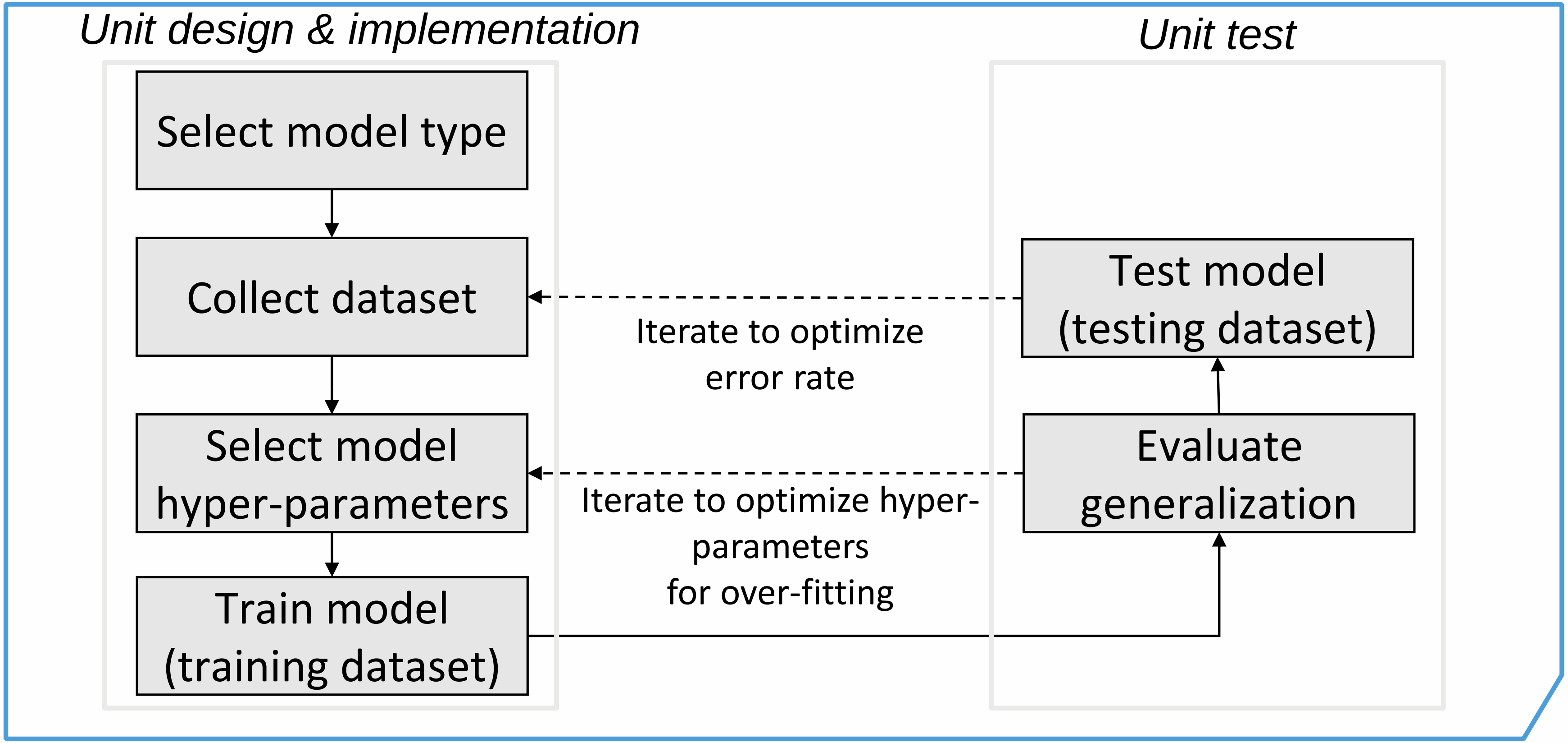
Domain coverage requirements

e.g., pedestrian equivalence classes

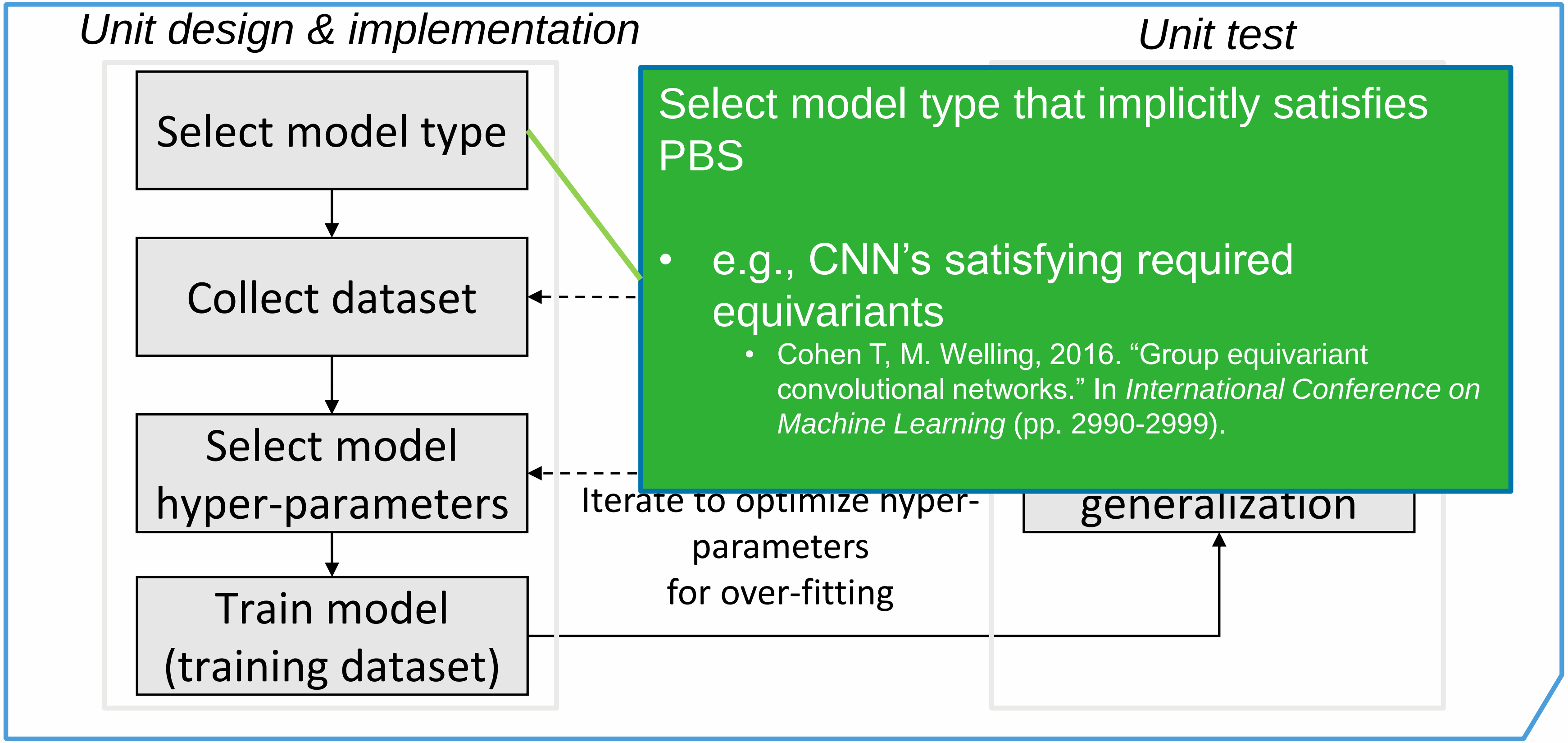
Risk profiling of inputs

e.g. severity of misclassifying different subclasses of objects

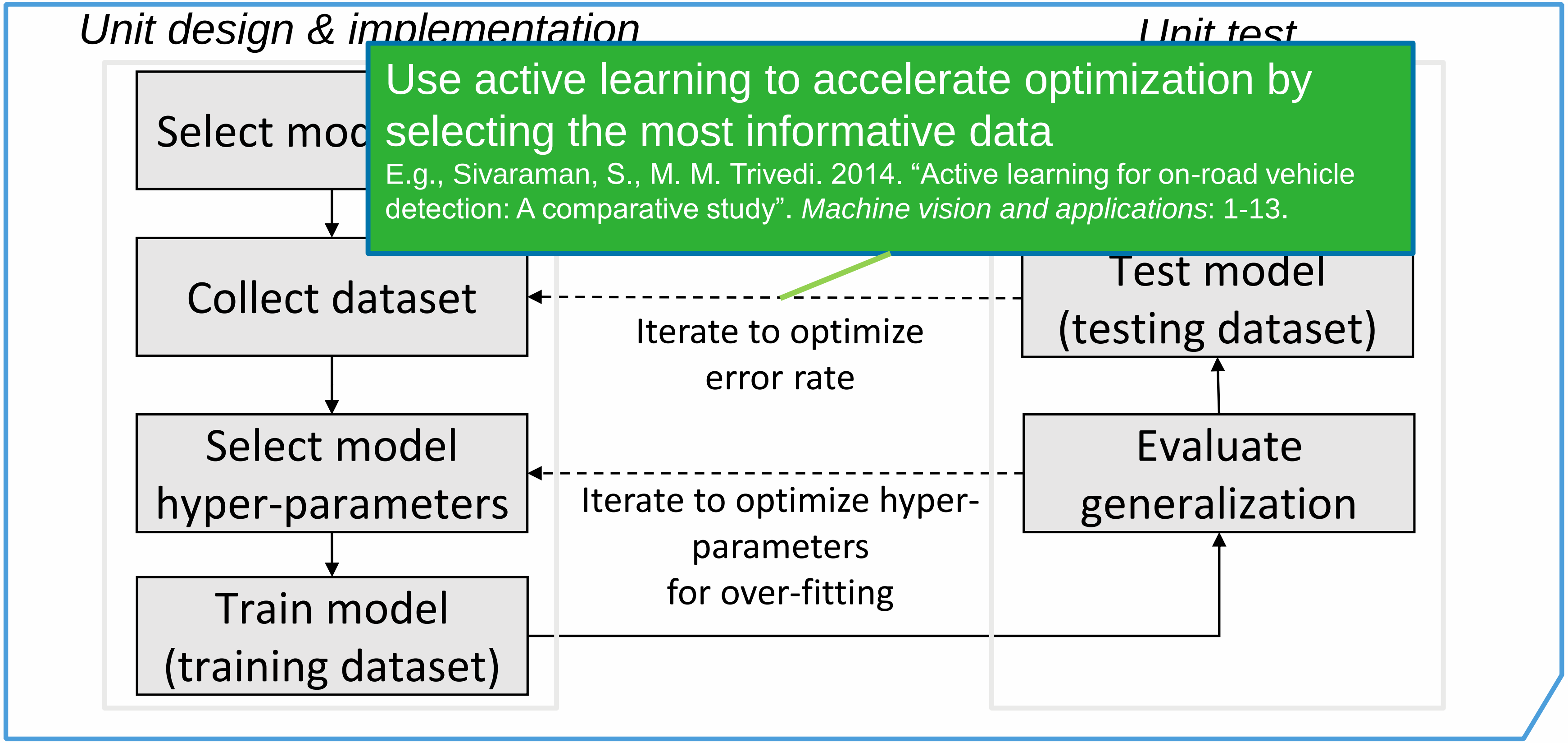
Where to Use Specifications



Where to Use Specifications



Where to Use Specifications



Where to Use Specifications

Unit design & implementation

Select model type

Collect dataset

Select model hyper-parameters

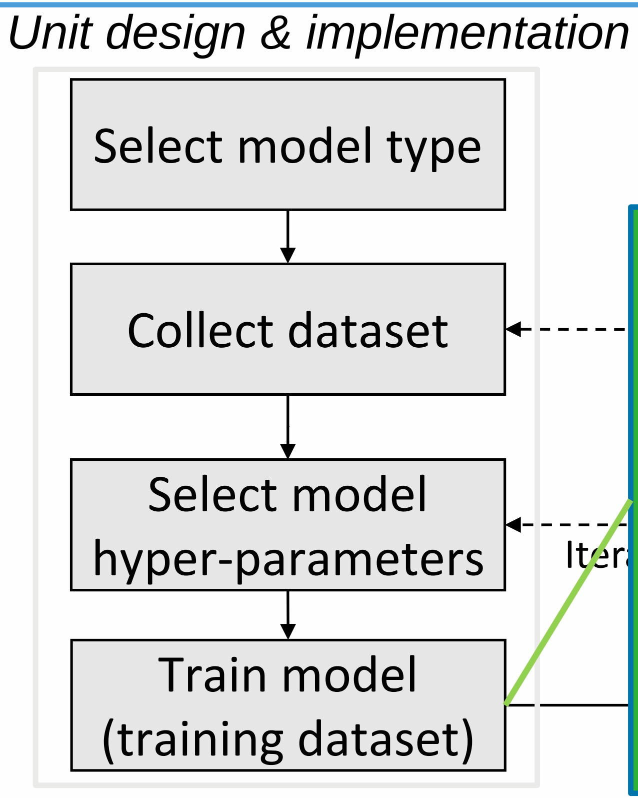
Train model (training dataset)

Iter

Unit test

- Ensure dataset satisfies specs
  - DS (e.g., all ped poses represented)
  - PBS (e.g., all ped <= 9ft)
- Use PBS to augment dataset so that specs are learned by model
  - e.g., generate non-ped > 9ft
  - e.g., use GANs to generate invariant examples
    - Liu M.Y., T. Breuel, J. Kautz, 2017. “Unsupervised Image-to-Image Translation Networks”. *arXiv preprint* arXiv:1703.00848

Where to Use Specifications



Unit test

Incorporate PBS into loss function to penalize models that violate them and guide learning.

e.g.,

Xu J., Z. Zhang, T. Friedman, Y. Liang, G.V. Broeck, 2017. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge". arXiv preprint arXiv:1711.11157.

Vedaldi A., M. Blaschko, A. Zisserman, 2011. "Learning equivariant structured output SVM regressors." In *Computer Vision (ICCV), IEEE International Conference on 2011* (pp. 959-966). IEEE.

Where to Use Specifications

Unit design & implementation

Select model type

- Use test coverage metrics designed for ML
  - E.g., Sun, Y., X. Huang, and D. Kroening. "Testing Deep Neural Networks." *arXiv preprint* arXiv:1803.04792 (2018).
- Use explanation techniques to diagnose why tests pass or fail
  - Koopman, P. and M. Wagner. "Toward a Framework for Highly Automated Vehicle Safety Validation," SAE World Congress, 2018. SAE-2018-01-1071.

mize

e hyper-

s

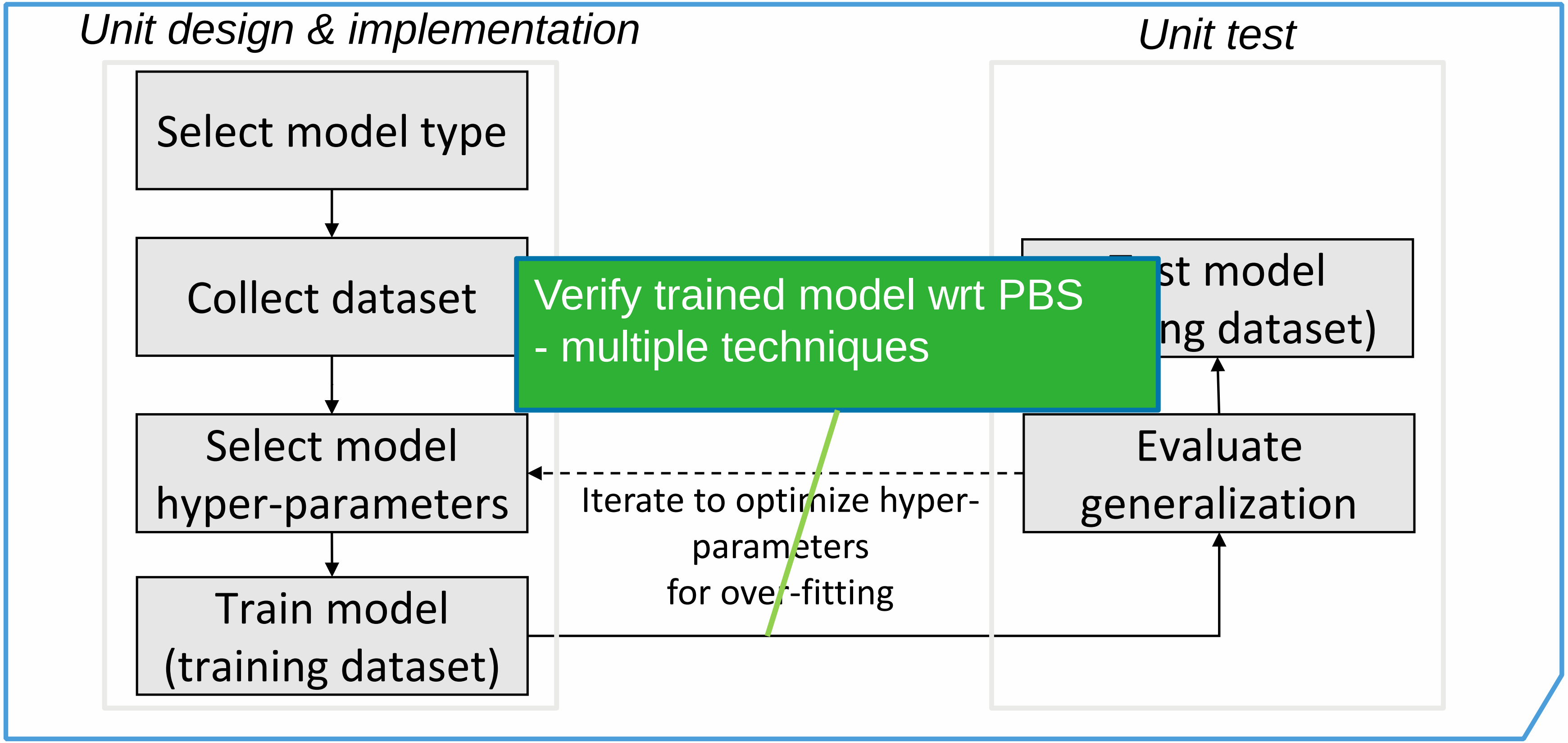
ng

Unit test

Test model (testing dataset)

Evaluate generalization

Where to Use Specifications



Where to Use Specifications

Verification Techniques for ML

Requires interpretability – use interpretability enhancing techniques discussed below

Methods		ASIL A	ASIL B	ASIL C	ASIL D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Where to Use Specifications

Verification Techniques for ML

Combine formal and non-formal techniques
e.g. falsification

- Dreossi, T., A. Donzé, and S.A. Seshia. "Compositional falsification of cyber-physical systems with machine learning components." In *NASA Formal Methods Symposium,* pp. 357-372. Springer, Cham, 2017.

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Where to Use Specifications

Verification Techniques for ML

Proof that model satisfies PBS

- Seshia, S.A., D. Sadigh, and S.S. Sastry. "Towards verified artificial intelligence." *arXiv preprint* arXiv:1606.08514 (2016).

Proof of minimum adversarial attack radius

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Where to Use Specifications

Verification Techniques for ML

These are code-specific techniques.

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Where to Use Specifications

Verification Techniques for ML

PBS property checking

- E.g., Katz, G., C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. 2017. “Re-luplex: An Efficient SMT Solver for Verifying Deep Neural Networks". *arXiv preprint arXiv*:1702.01135

Abstract Interpretation

- E.g., Gehr, T., M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. "AI^2: Safety and robustness certification of neural networks with abstract interpretation." In Security and Privacy (SP), 2018 IEEE Symposium on. 2018.

		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Where to Use Specifications

Verification Techniques for ML

Translate the model to another semantically equivalent representation for which analysis tools exist

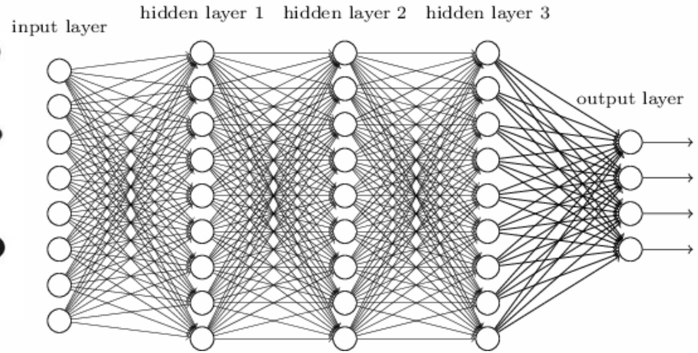
- E.g., Weiss, G., Y. Goldberg, and E. Yahav. "Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples." *arXiv preprint* arXiv:1711.09576 (2017).

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[bc]	+	+	++	++
1f	Data flow analysis[bc]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

Interpretability Assumption

More powerful ML model => Less interpretable

Naïve Bayes | Decision Tree | Bayesian Network | Random Decision Forest | Support Vector Machine | Deep Neural Network



Interpretability Assumption : How to address?

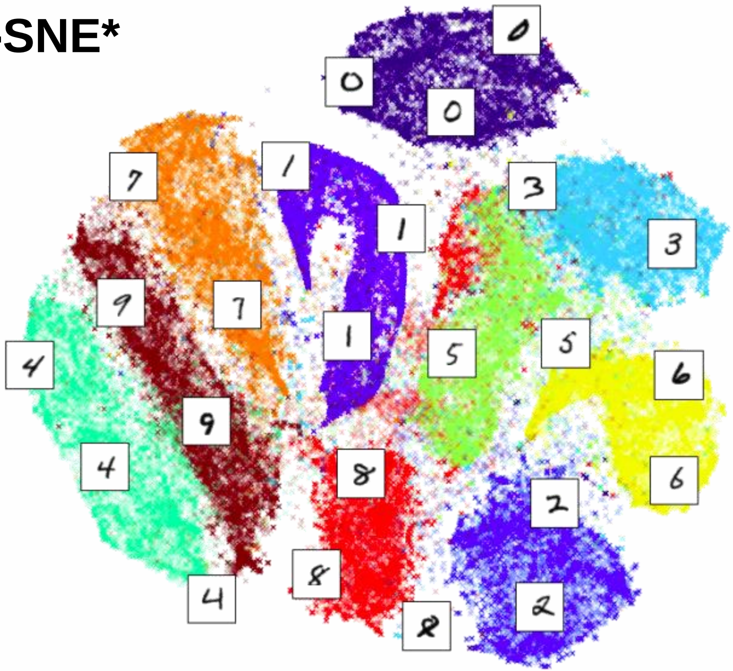
Require use of interpretable models
or, provide justification why not (safety case)
and use interpretability increasing techniques

Model Visualization	Dependency Analysis
Rule Extraction	Natural Language
Saliency Maps	DARPA XAI

Interpretability Increasing Techniques

Global visualization: t-SNE*
through dimensionality
reduction

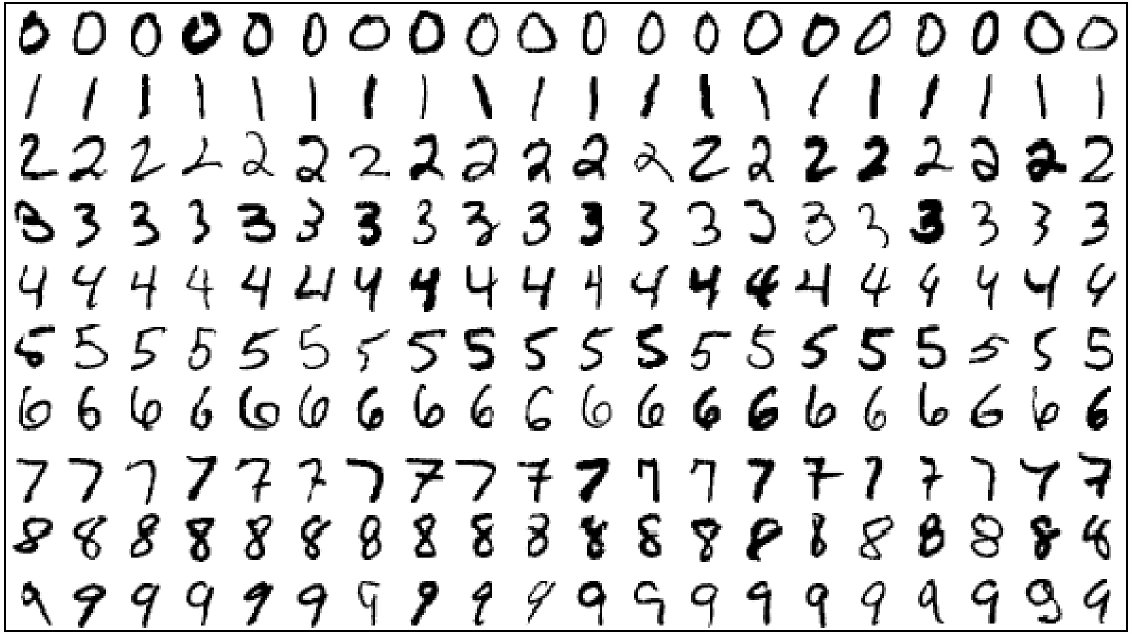
t-SNE
(t-distributed
Stochastic
Neighbor
Embedding)



MNIST in 2D

* Maaten, L. van der, and G. Hinton. "Visualizing data using t-SNE." *Journal of machine learning research* no. 9, Nov (2008): 2579-2605.

MNIST * data set



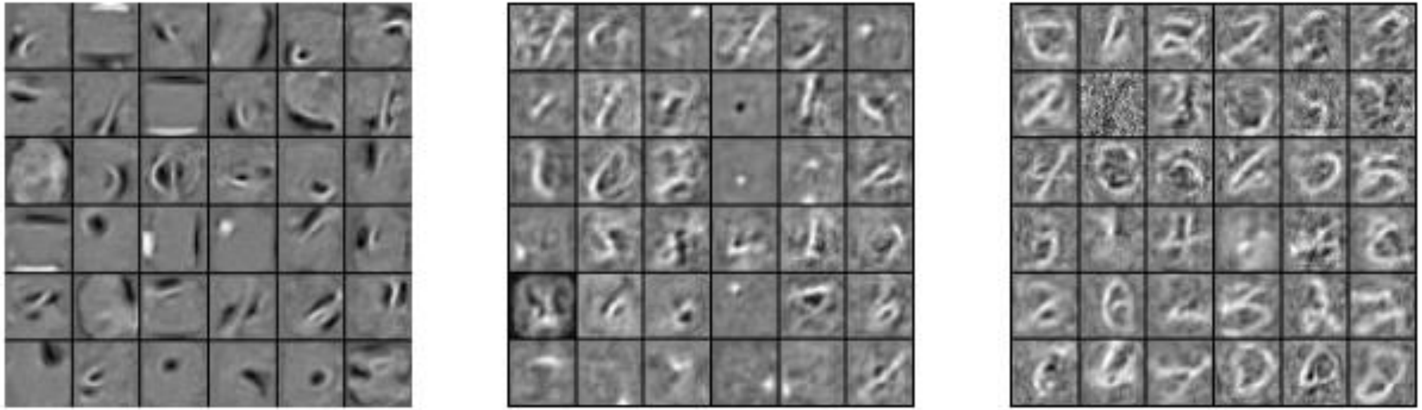
* http://yann.lecun.com/exdb/mnist/

Interpretability Increasing Techniques

Global visualization: Activation Maximization

data: MNIST

Layer 1 Layer 2 Layer 3

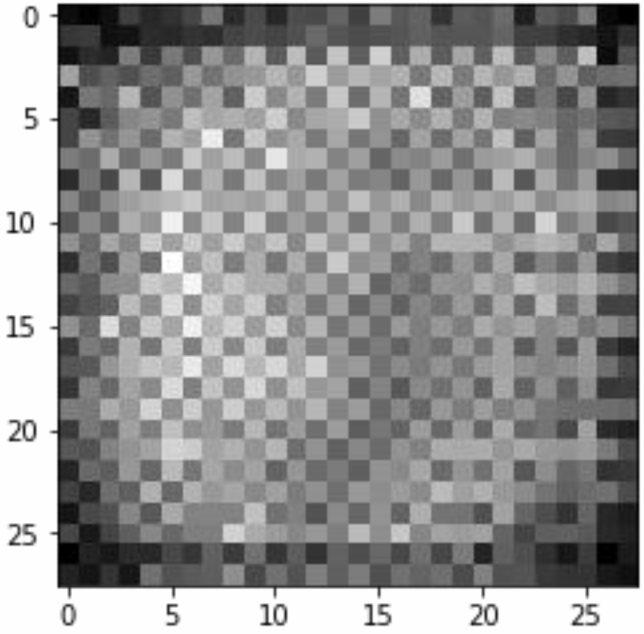


* Erhan, Dumitru, Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Visualizing higher-layer features of a deep network." University of Montreal 1341, no. 3 (2009): 1.

Interpretability Increasing Techniques

Global feature importance

data: MNIST

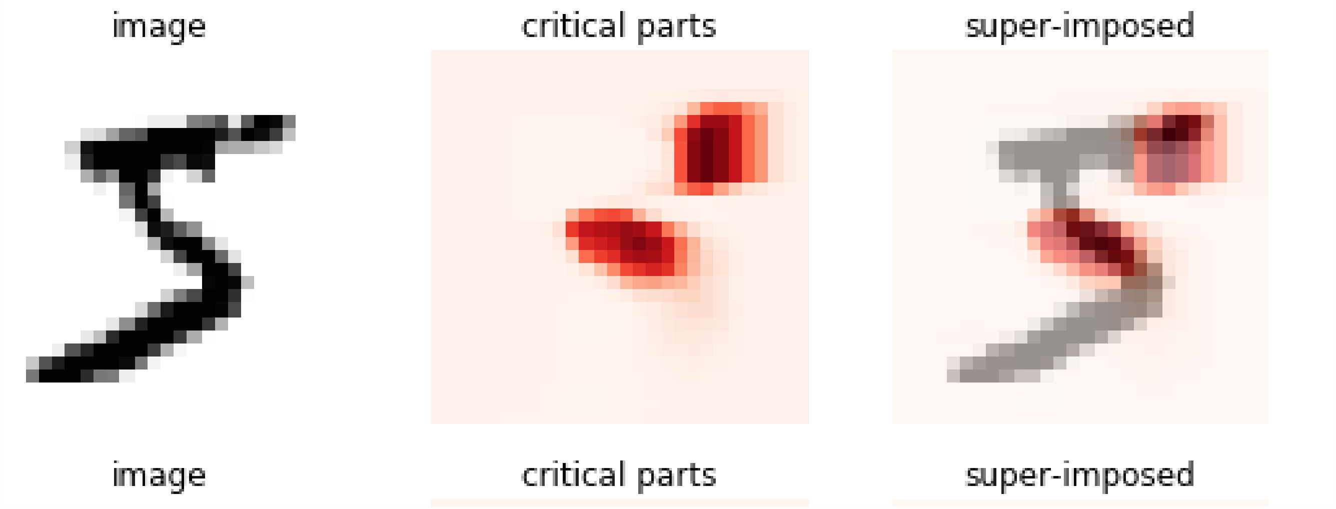


https://www.kaggle.com/c/digit-recognizer/discussion/70350

Interpretability Increasing Techniques

Local explanation: Occlusion map

data: MNIST



Techniques

Summary

- Key assumptions are not met by ML: complete specification and interpretability
- However, research is active in these areas to address the shortfall

Techniques

Best Practices	Prevent faults
Verification	Find and repair faults
Testing	
Fault Tolerance	Live with faults

Fault Tolerance

Can use as-is for ML

Fault tolerance strategies are architecture-level and can be component implementation agnostic

But error detection/handling should use programming!

Some ML-oriented Fault Tolerance Methods

Ensemble methods
- Use multiple classifiers and aggregate their results

Safety envelope
- Use ML components only within safe contexts – e.g. to choose among a set of safe actions

Simplex architecture
- Monitor when ML component is unreliable and switch to a reliable (but usually conservative) non-ML component – requires “uncertainty” check on ML component

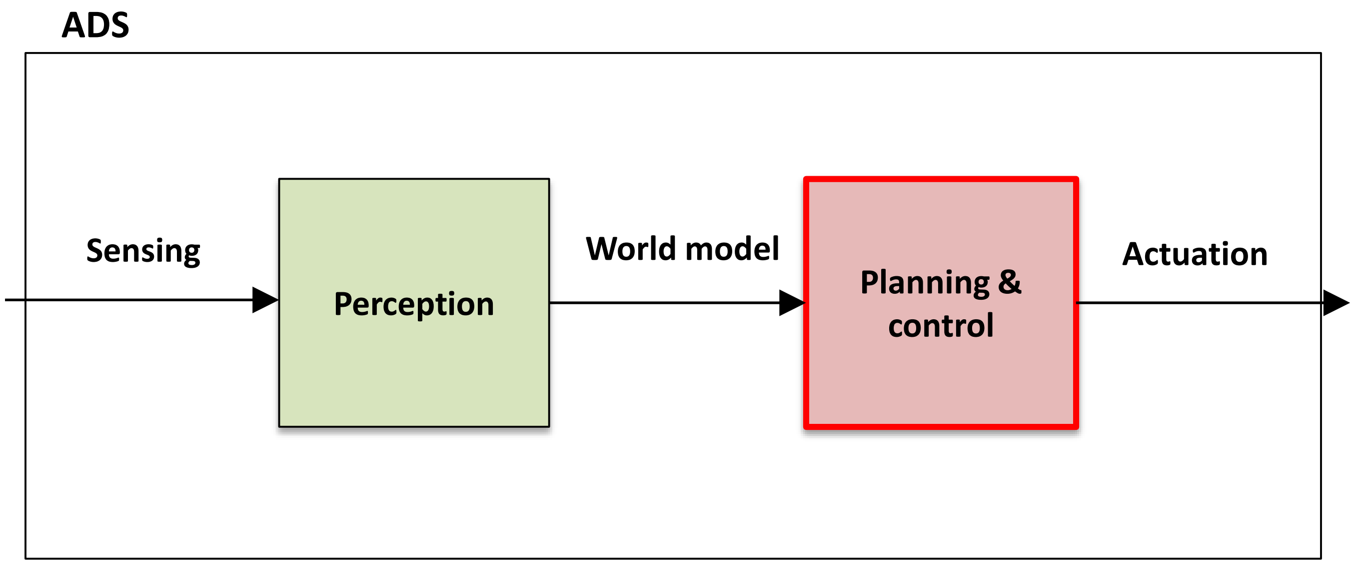
Runtime verification + Fail Safety
- Monitor PBS satisfaction and go to fail safe behaviour if PBS is violated at run-time

Summary

Q: How well do ISO 26262 SW recommendations fit ML?

Best Practices	Prevent faults	N/A – but ML best practices will emerge (unclear of impact)
Verification	Find and repair faults	Adapt/Use – if specification and interpretability problems are addressed (research is active)
Testing		
Fault Tolerance	Live with faults	Use – Fault tolerance techniques can be used directly

What about planning & control?



Main type of ML in actuation/control: Reinforcement Learning (RL)

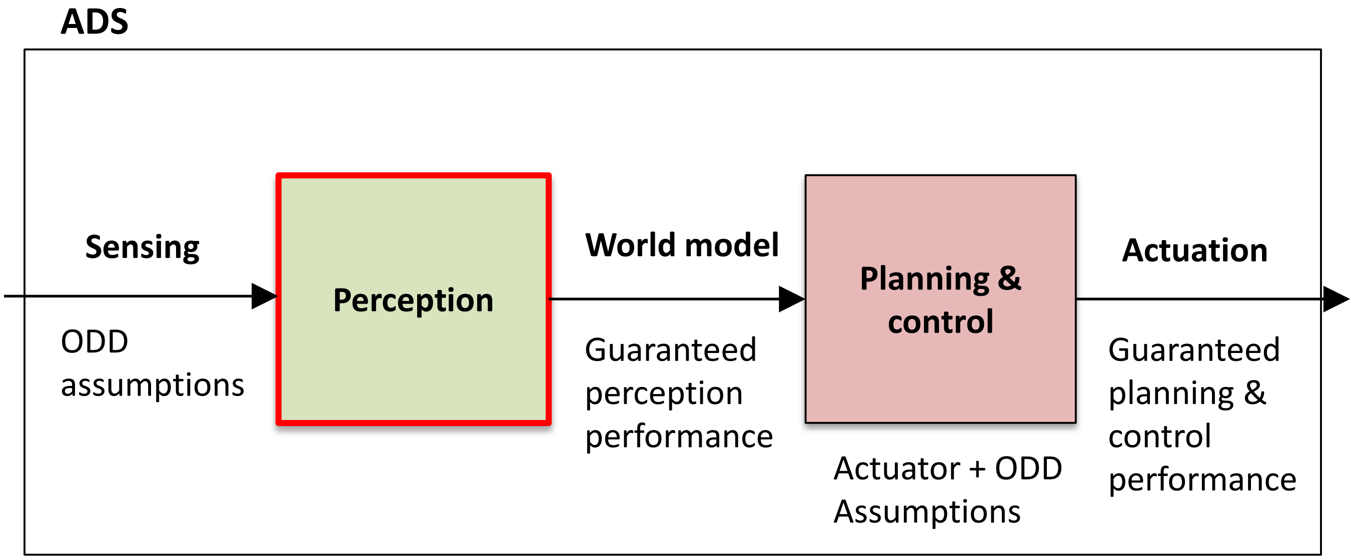
- learn an optimal control policy by training with simulation + reward function
- exploration/exploitation trade-off
- leaning could be on-line as well

Some unique safety issues:

- e.g., reward function does not incorporate (safety) risk
- e.g., model learns to “game” the reward function
- See: Amodei, D., C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mane. 2016. “Concrete problems in AI safety". *arXiv preprint arXiv*:1606.06565 .

Safety through (Measurement) Uncertainty-Reduction

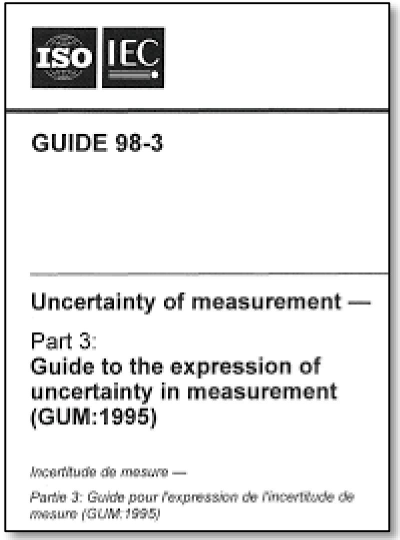
Managing Perceptual Uncertainty in ML



The following slides are based on Krzysztof Czarnecki and Rick Salay.
Towards a Framework to Manage Perceptual Uncertainty for Safe Automated Driving.
In WAISE, Västerås, Sweden, 2018
https://uwaterloo.ca/wise-lab/publications/towards-framework-manage-perceptual-uncertainty-safe

Guide to the Expression of Uncertainty in Measurement (GUM)

- True accuracy unknowable
  - Accuracy in ML wrt. test set only
- Must estimate uncertainty


ISO IEC

GUIDE 98-3

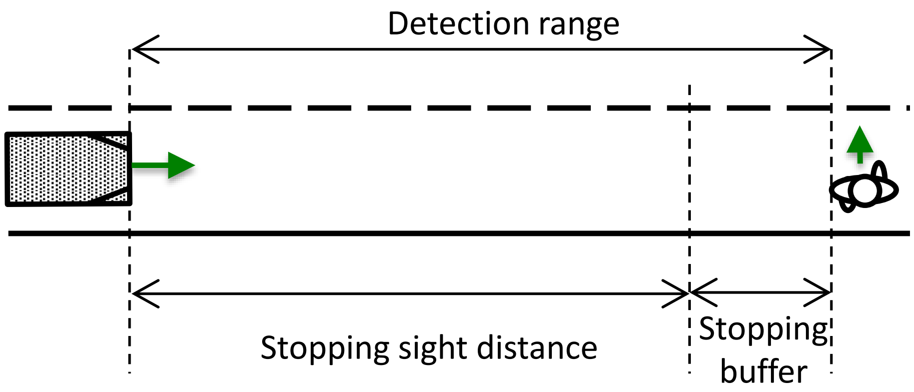
Uncertainty of measurement —

Part 3:
Guide to the expression of uncertainty in measurement (GUM:1995)

Incertitude de mesure —

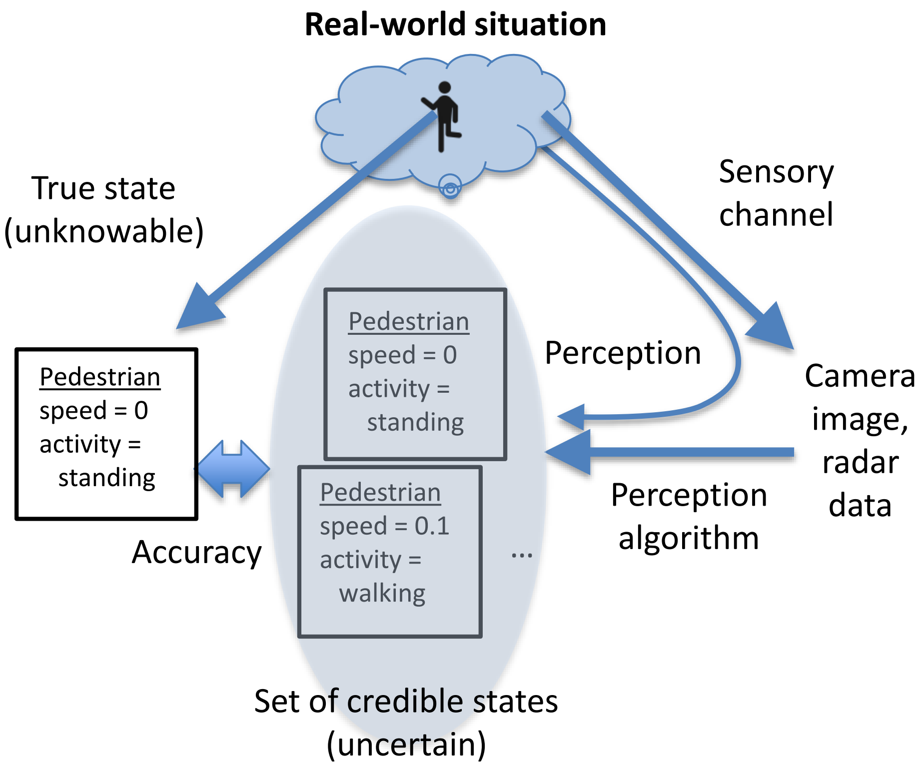
Partie 3: Guide pour l'expression de l'incertitude de mesure (GUM:1995)

Sample Scenario-Dependent Perception-Performance Safety-Requirement Spec

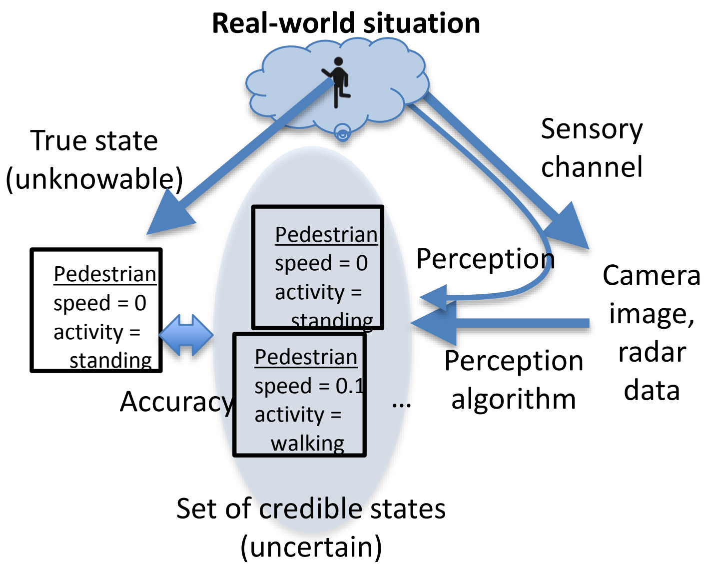


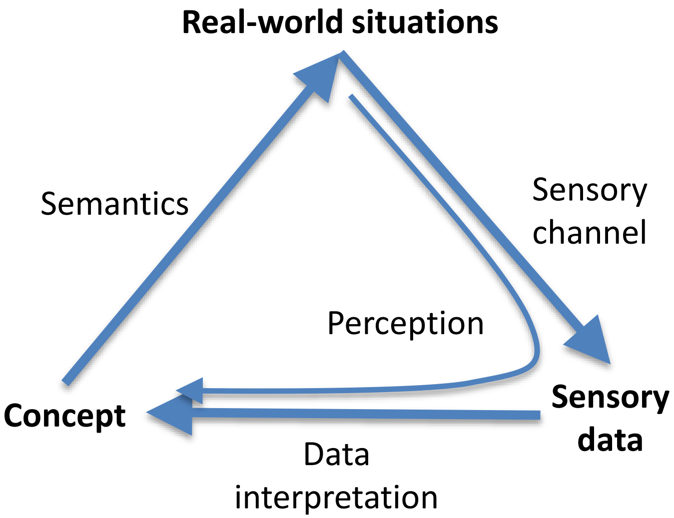
Detect pedestrians on the roadway
within range of 10 m and with maximum perception-reaction delay of 0.5 s
with missed detection **probability** of 10^{-9} or less
with localization **uncertainty** of ± 0.5 m or better
within ODD conditions

Perception Triangle (Instance-Level)



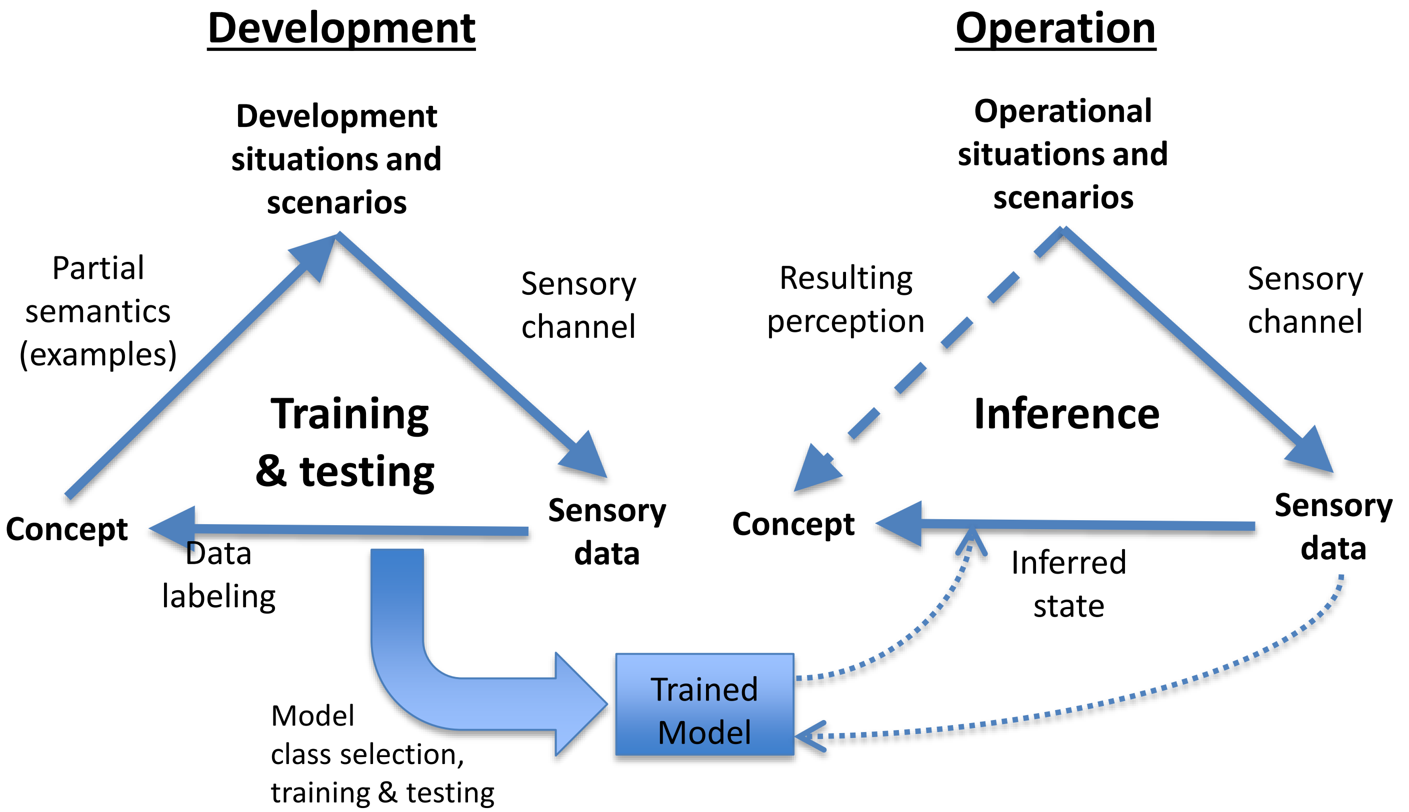
Perceptual Triangle



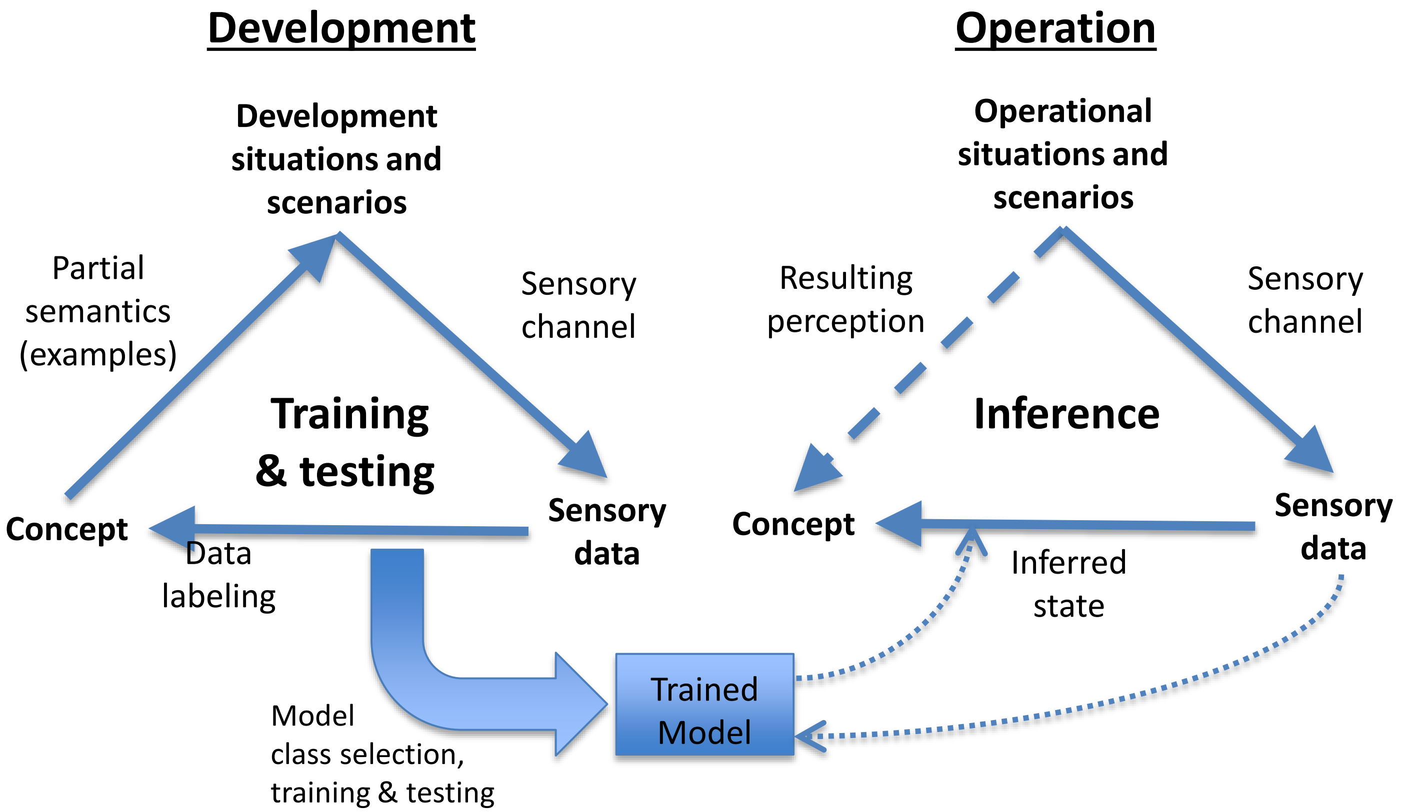


Instance-level | **Domain-level (generic)**

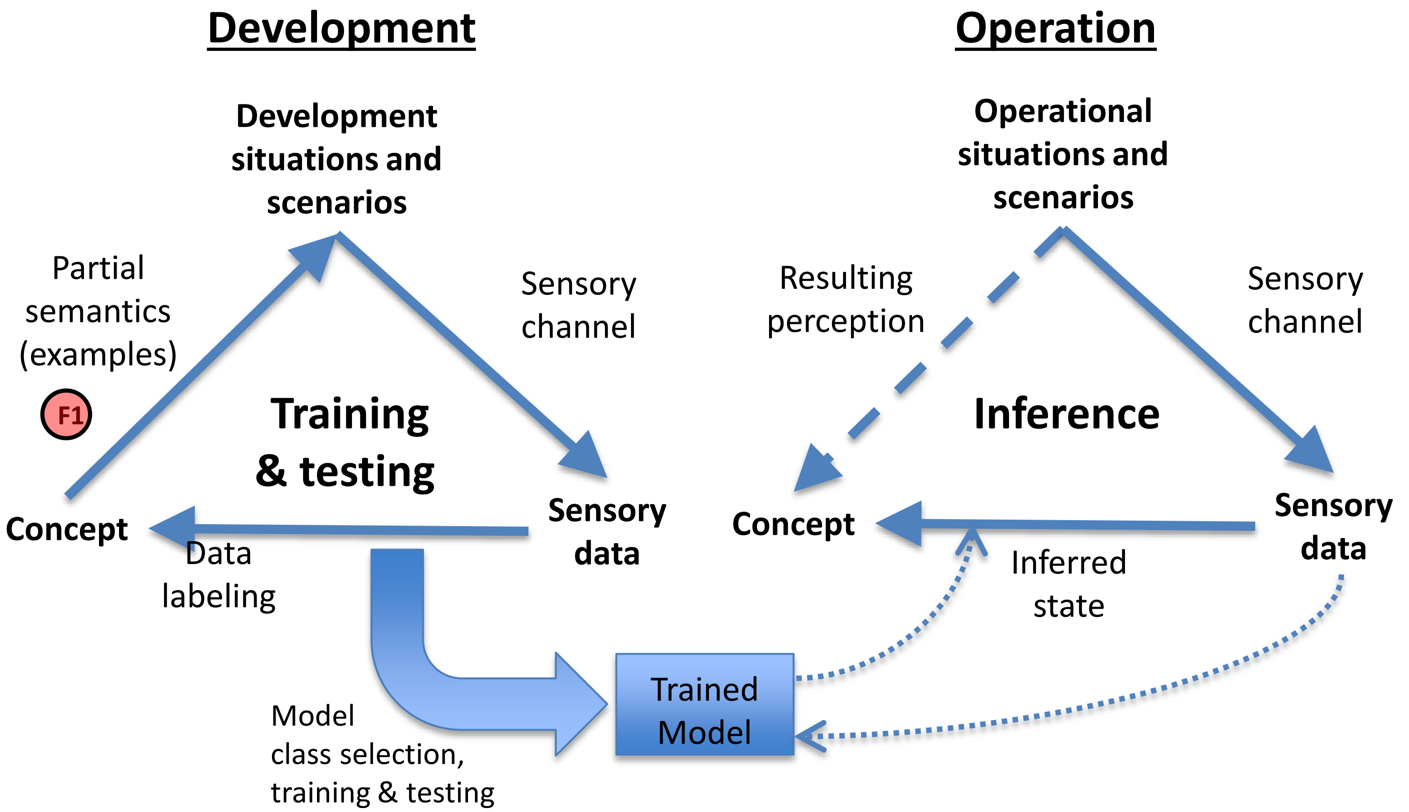
Perceptual Triangle When Using Supervised ML



Factors Influencing Uncertainty



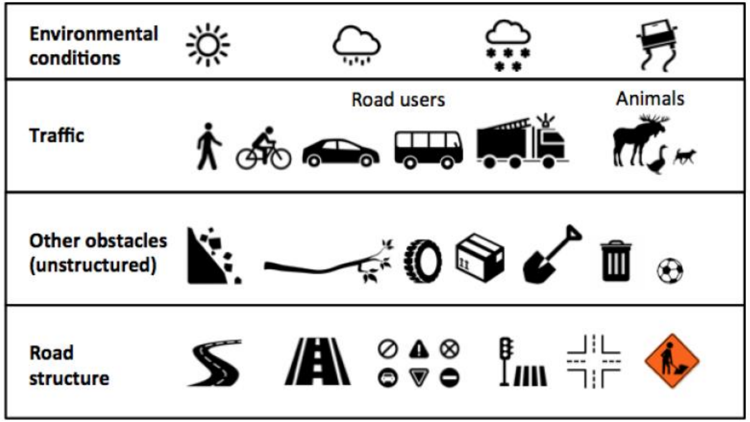
F1: Conceptual Uncertainty



F1: Conceptual Uncertainty
Pedestrian or Cyclist?

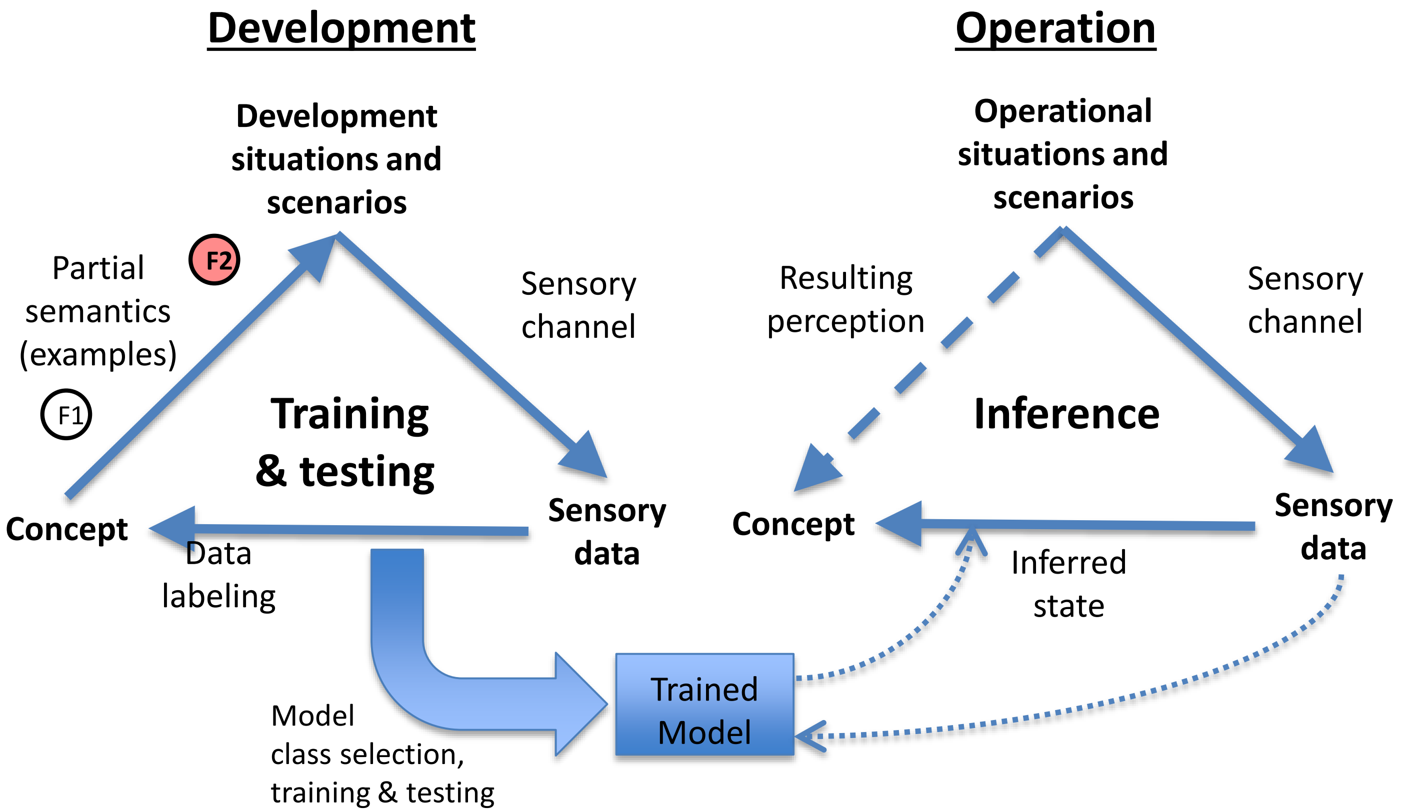
F1: Conceptual Uncertainty

- Assessed by expert review or labeling disagreement
- Reduced by developing standard ontologies
  - – E.g., WISE Drive Ontology

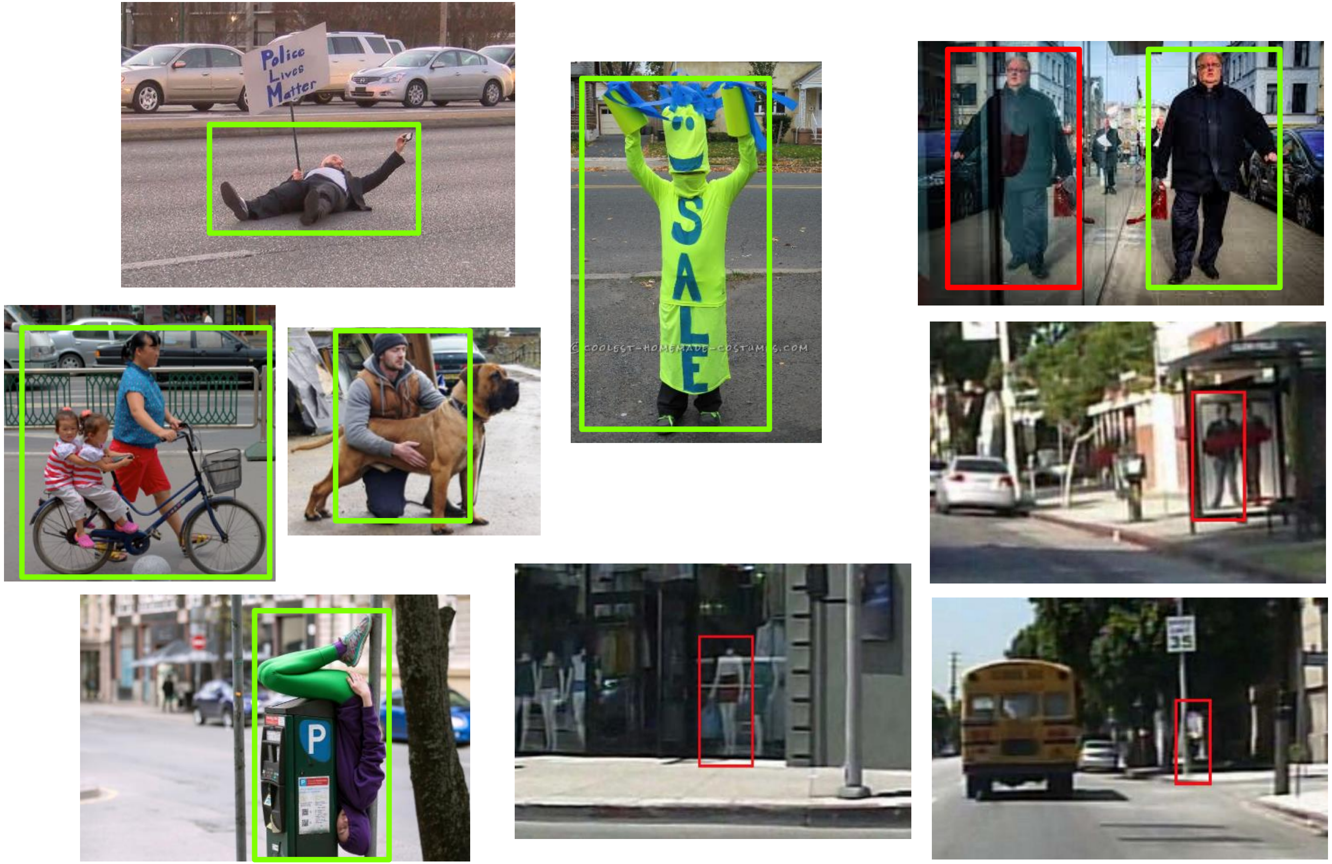


https://uwaterloo.ca/wise-lab/projects/wise-drive-requirements-analysis-framework-automated-driving

F2: Development Scenario Coverage

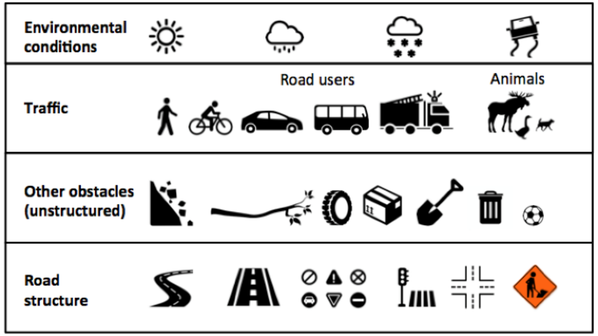


F2: Development Scenario Coverage



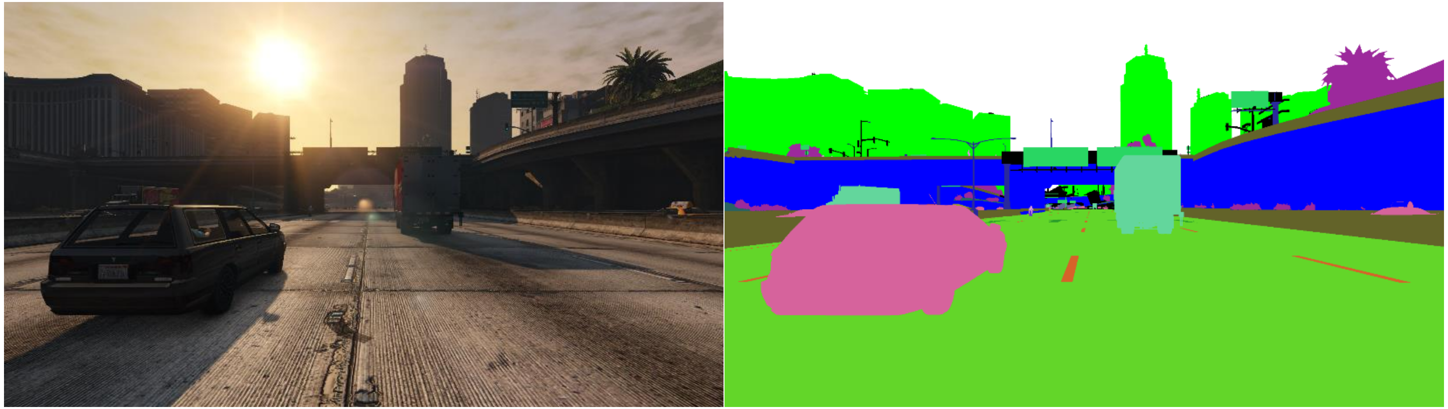
F2: Development Scenario Coverage

- Assessed with respect to ontologies and field validation targets
  - Must include positive/negative and near-hit/near-miss examples



- Challenge: how much data is enough?

Synthetic data sets



Angus et al. Unlimited Road-scene Synthetic Annotation (URSA) Dataset, ITCS'18

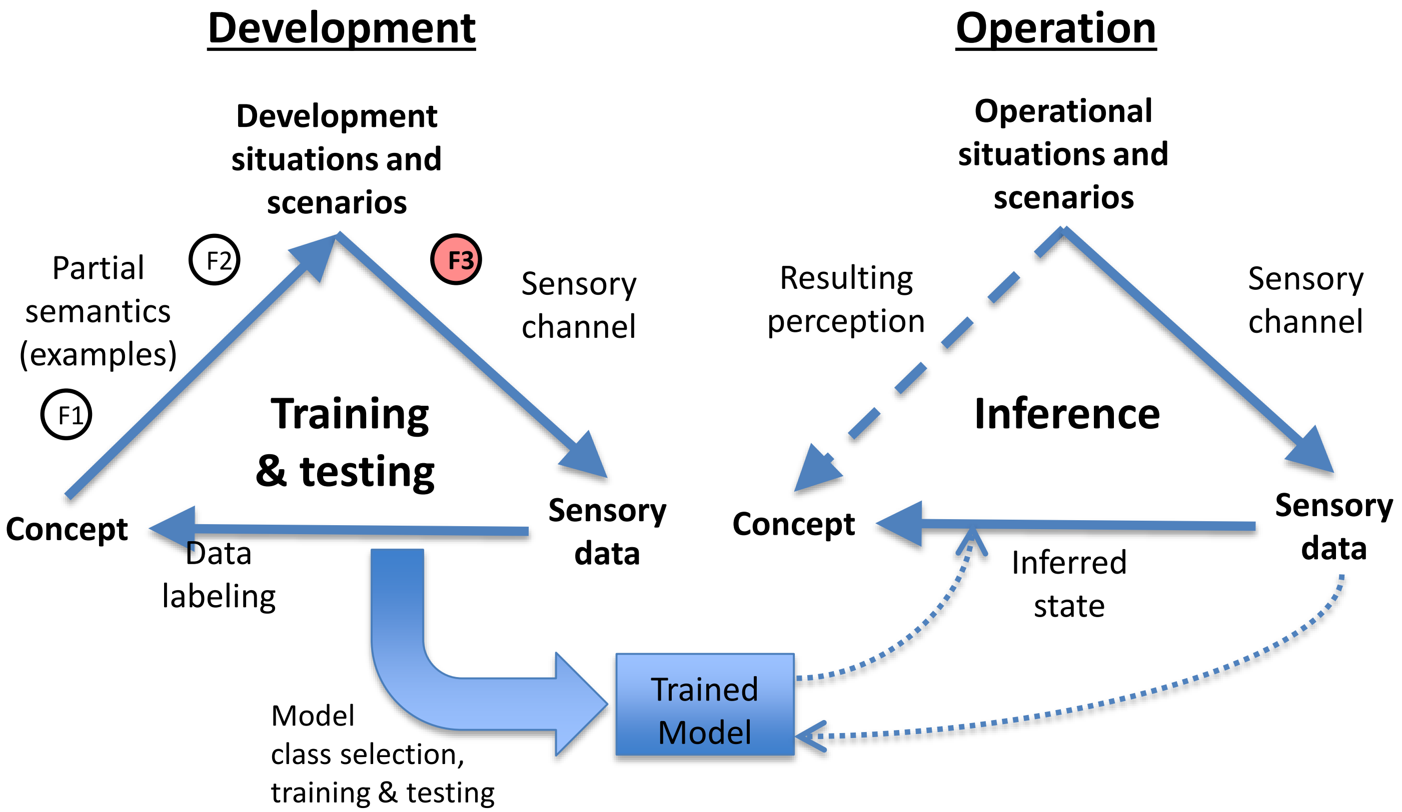
https://uwaterloo.ca/wise-lab/ursa

Active Learning

Data selection criteria

1. Uncertainty
2. Coverage & diversity
3. Collection & labeling cost
4. Risk profile

F3: Scene Uncertainty

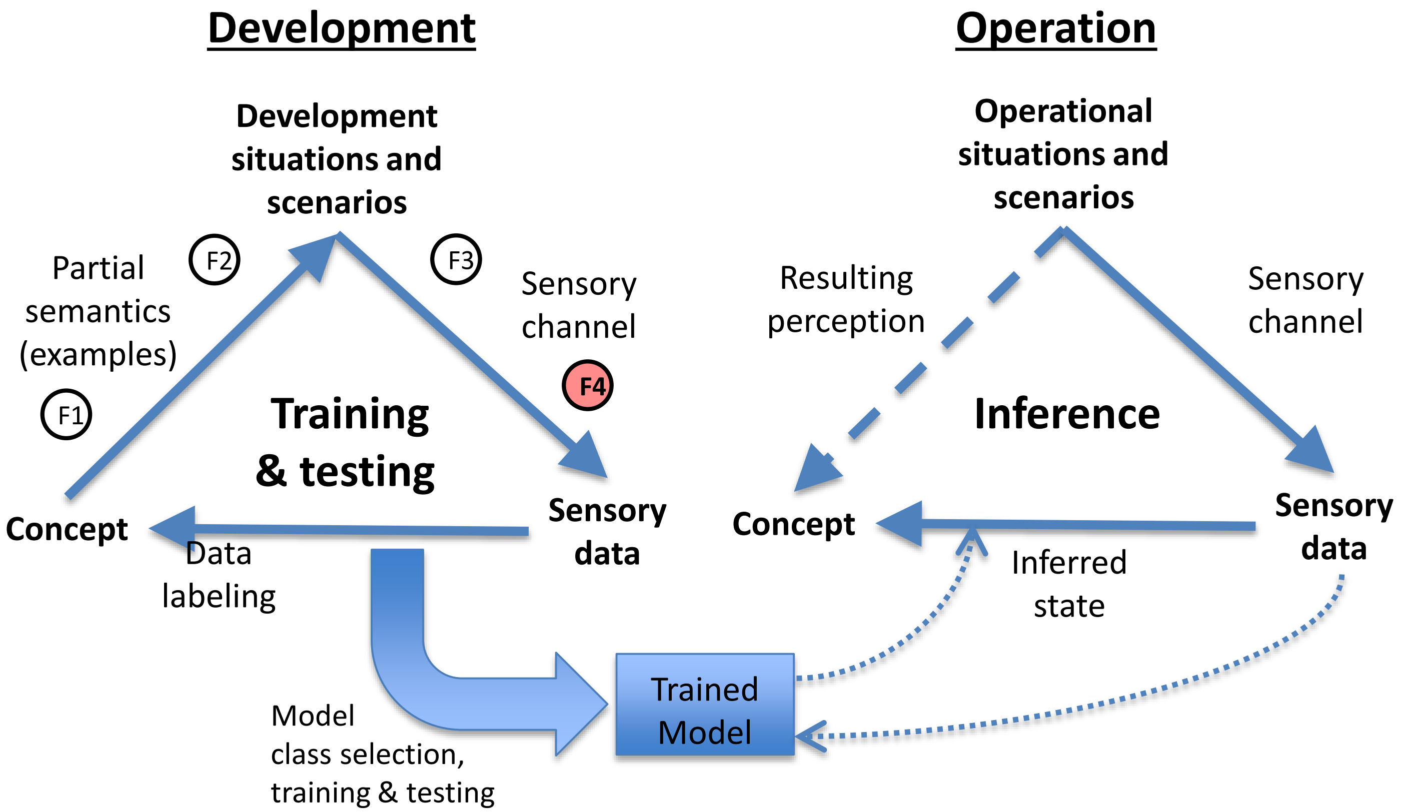


F3: Scene Uncertainty

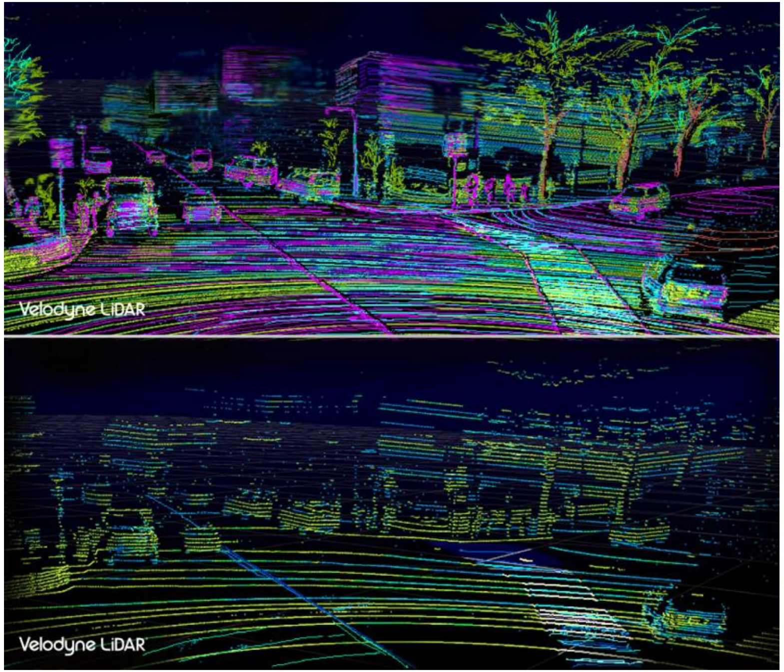
F3: Scene Uncertainty

- Surrogate measures
  - range, scale, occlusion level, atmospheric visibility, illumination, clutter and crowding level
- May compare test set accuracy and output confidence with these measures
- Also part of development data set coverage

F4: Sensor Properties



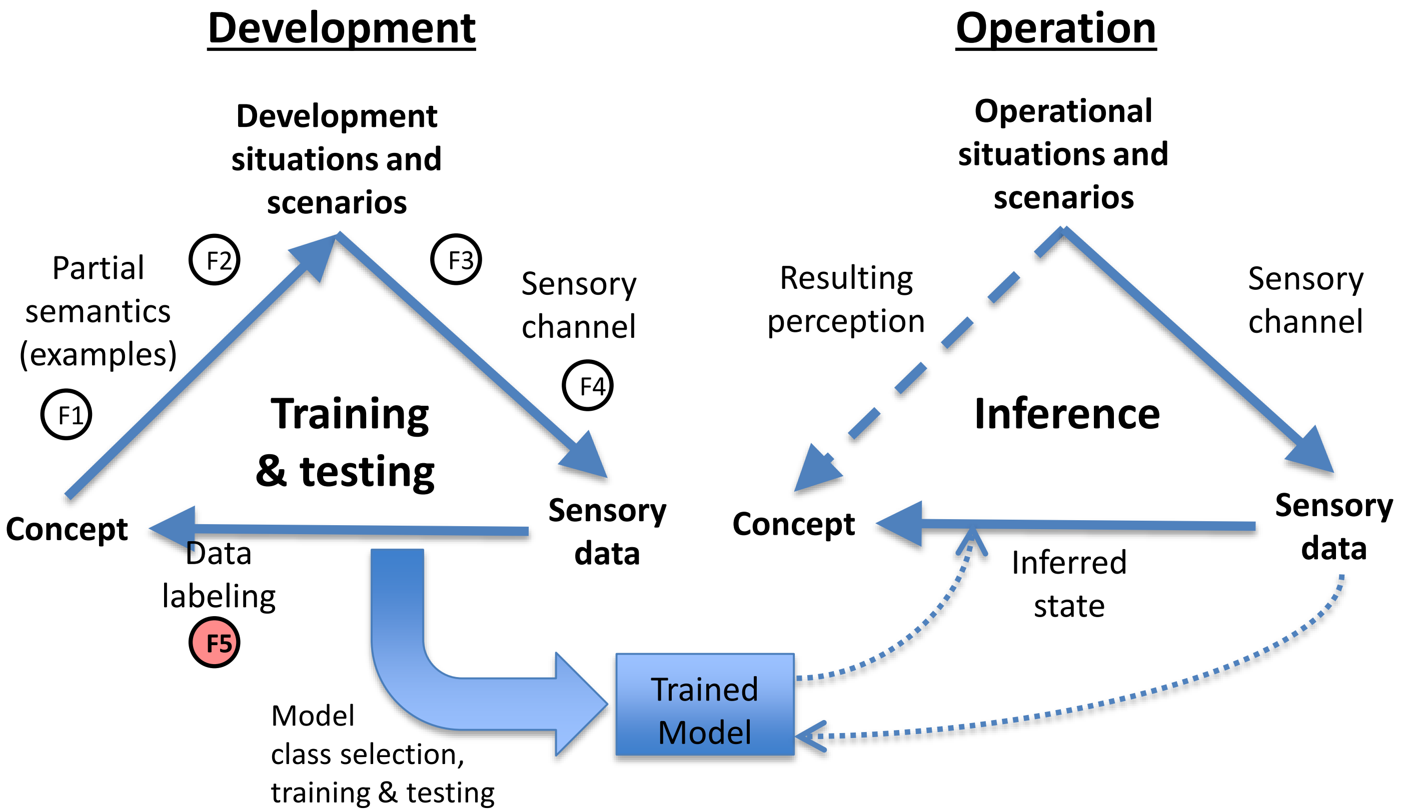
F4: Sensor Properties



F4: Sensor Properties

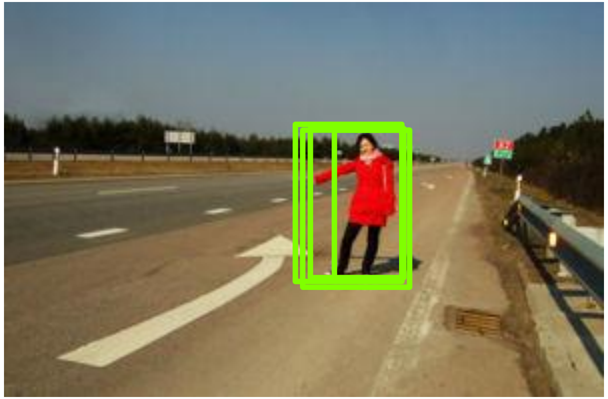
- Mature engineering discipline
  - Determining sensor properties to capture sufficient information
  - Mode, range, resolution, sensitivity, placement, etc.
- However, interaction between ML algorithms and sensor properties must be assessed
  - E.g., how effective is ML is ignoring sensor noise or artifacts?

F5: Label Uncertainty

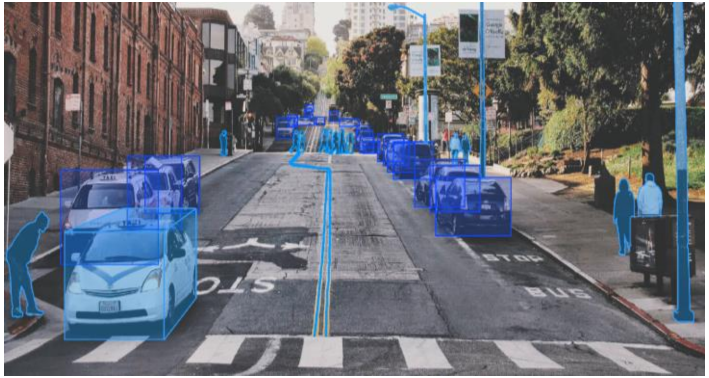


F5: Label Uncertainty

Class: cyclist vs. pedestrian



Bounding box placement uncertainty

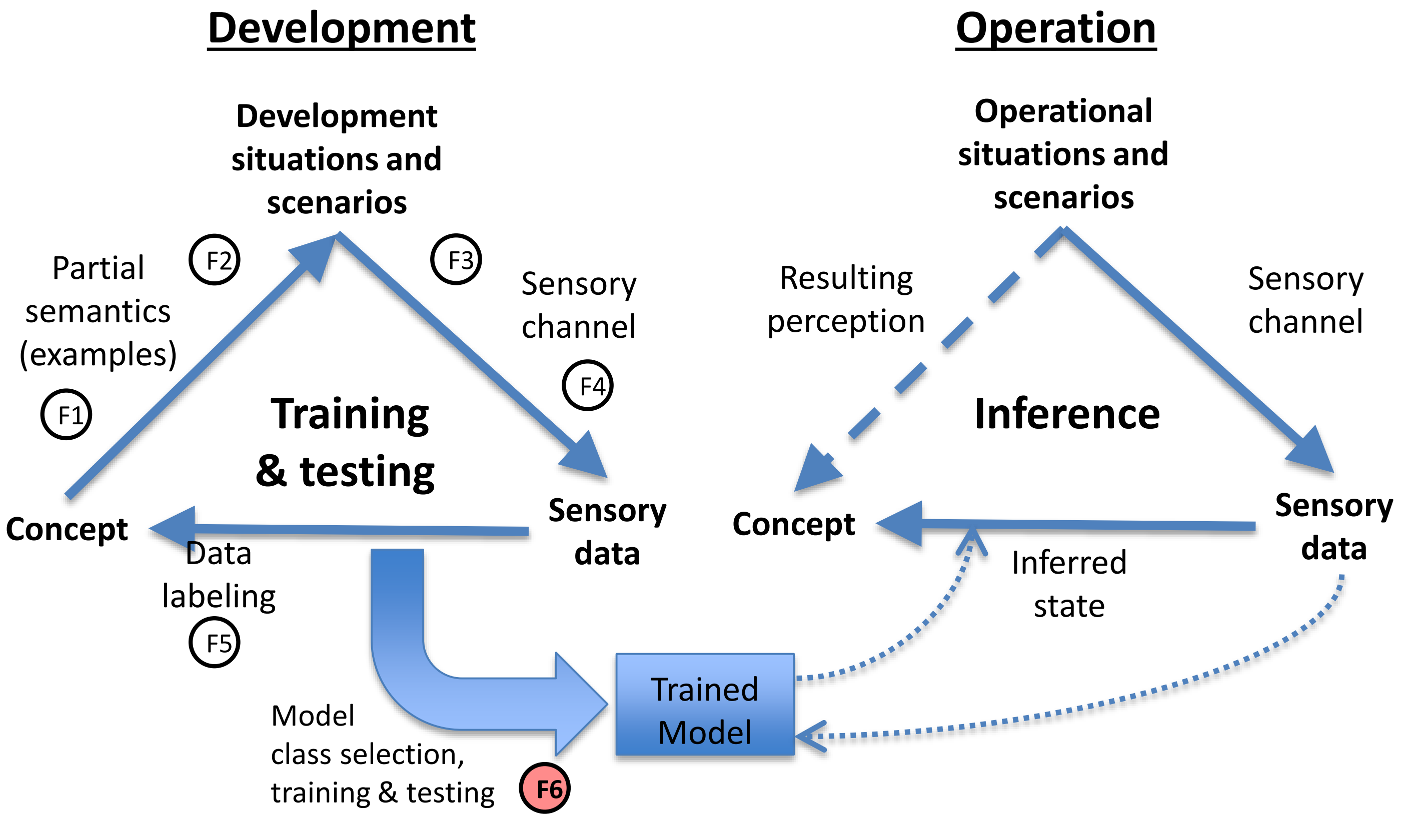


3D bounding box placement is challenging

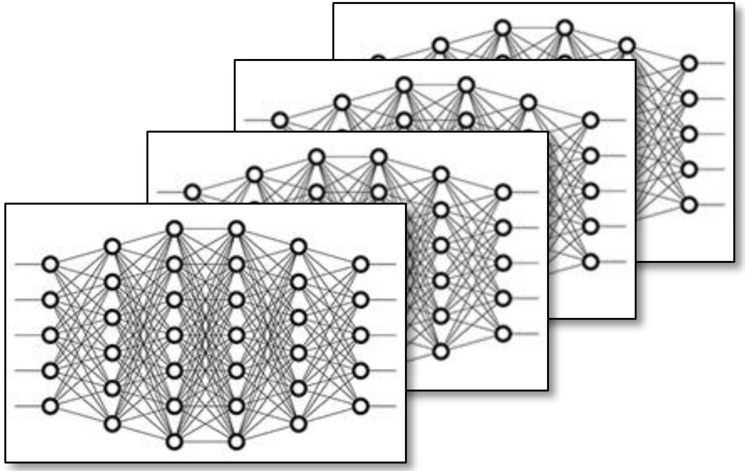
F5: Label Uncertainty

- Assessed by expert review and labeler disagreement
  - Existing research on determining number of labelers in crowd sourcing
  - E.g., may need as many as 6 independent votes
- Reduction measures
  - Conceptual clarity (F1)
  - Quality control
    - Clear instructions, training, verification, etc.
    - Bread and butter of labeling companies

F6: Model Uncertainty



F6: Model Uncertainty



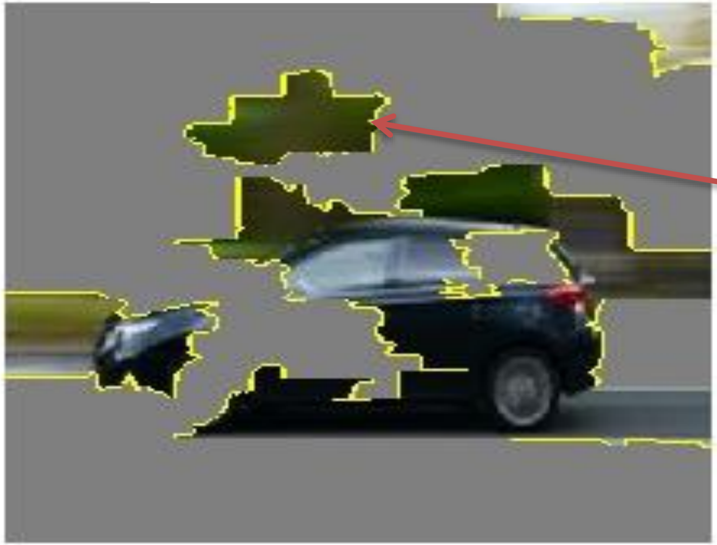
What model was learned in training?
What decisions will it make in operation?

F6: Model Uncertainty

1. Explanation methods help validate features
2. Robustness measures help assess risk of misclassification
3. Bayesian deep learning can help assess model uncertainty

Deep Learning and Explanations

Passenger car



The explanation shows
that a tree
contributed
to the classification
decision
(method: LIME)

The top 15 features (superpixels) used to classify corresponding input image as a car by an Inception network trained on ImageNet

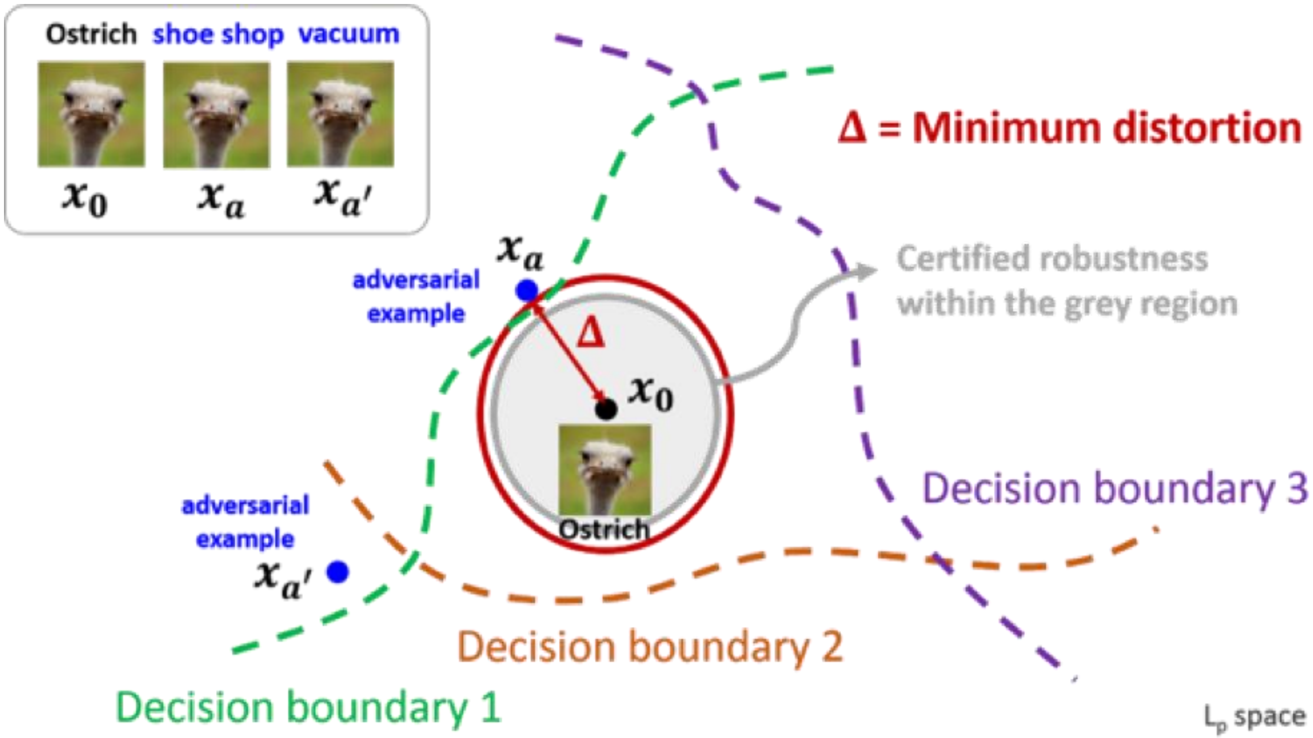
(see LIME at https://github.com/marcotcr/lime)

Adversarial Stickers

Misclassified as speed signs

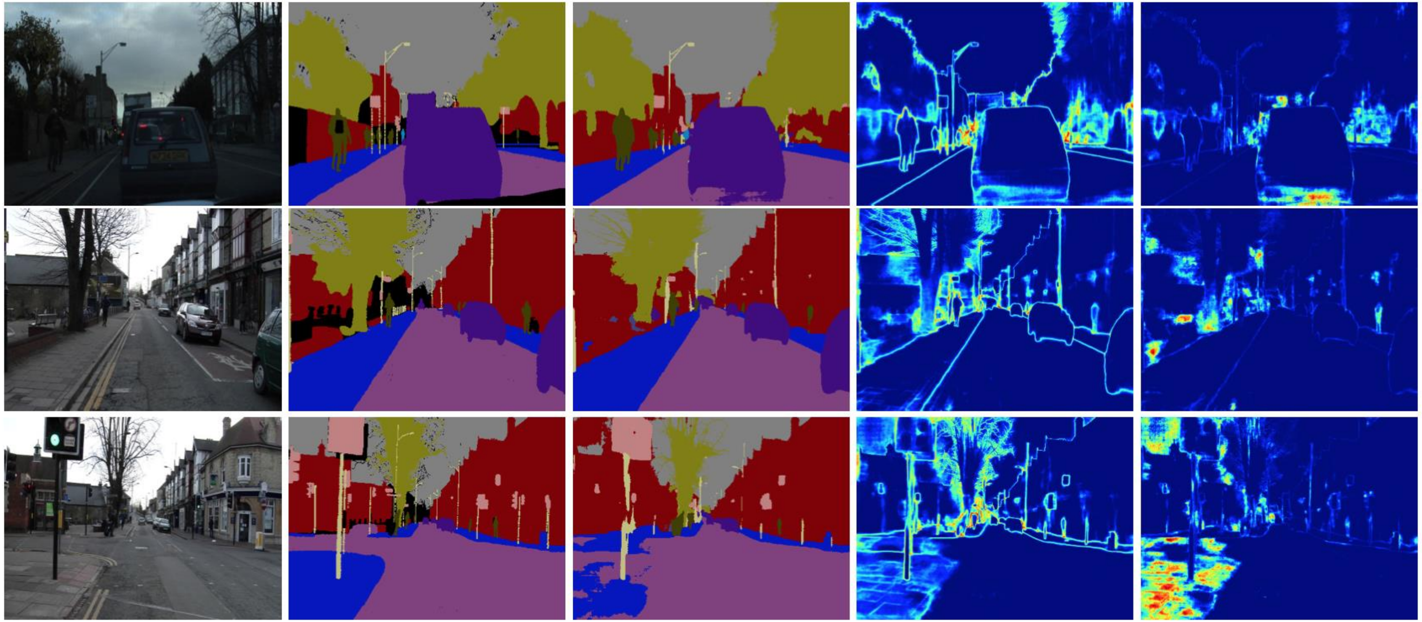
Evtimov et al.

Robustness Measures



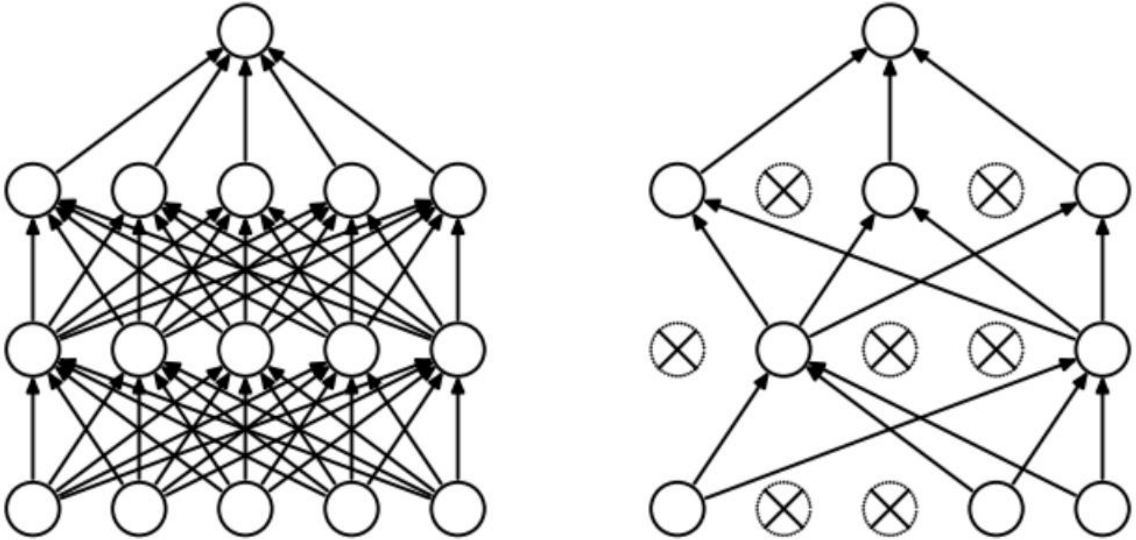
CLEVER approach by IBM

Aleatoric and Epistemic Uncertainty


(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

Yarin Gal, et al., https://arxiv.org/abs/1703.04977

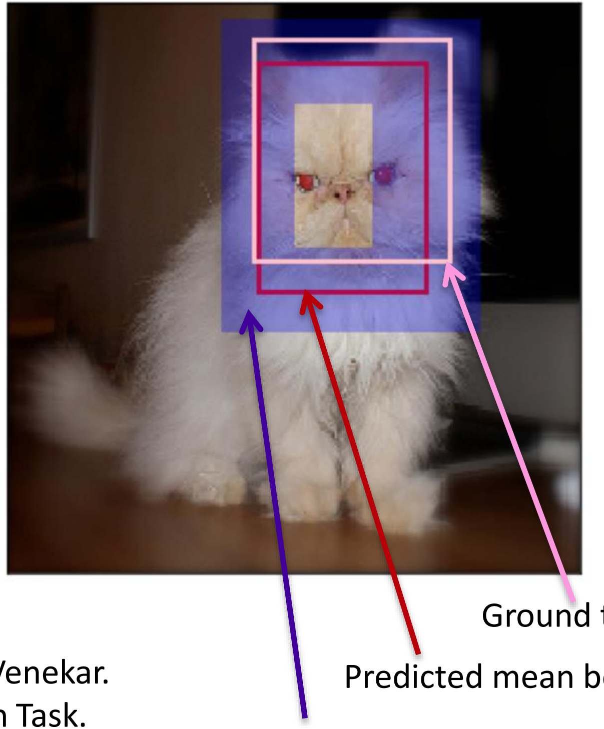
Dropout



(a) Standard Neural Net (b) After applying dropout.

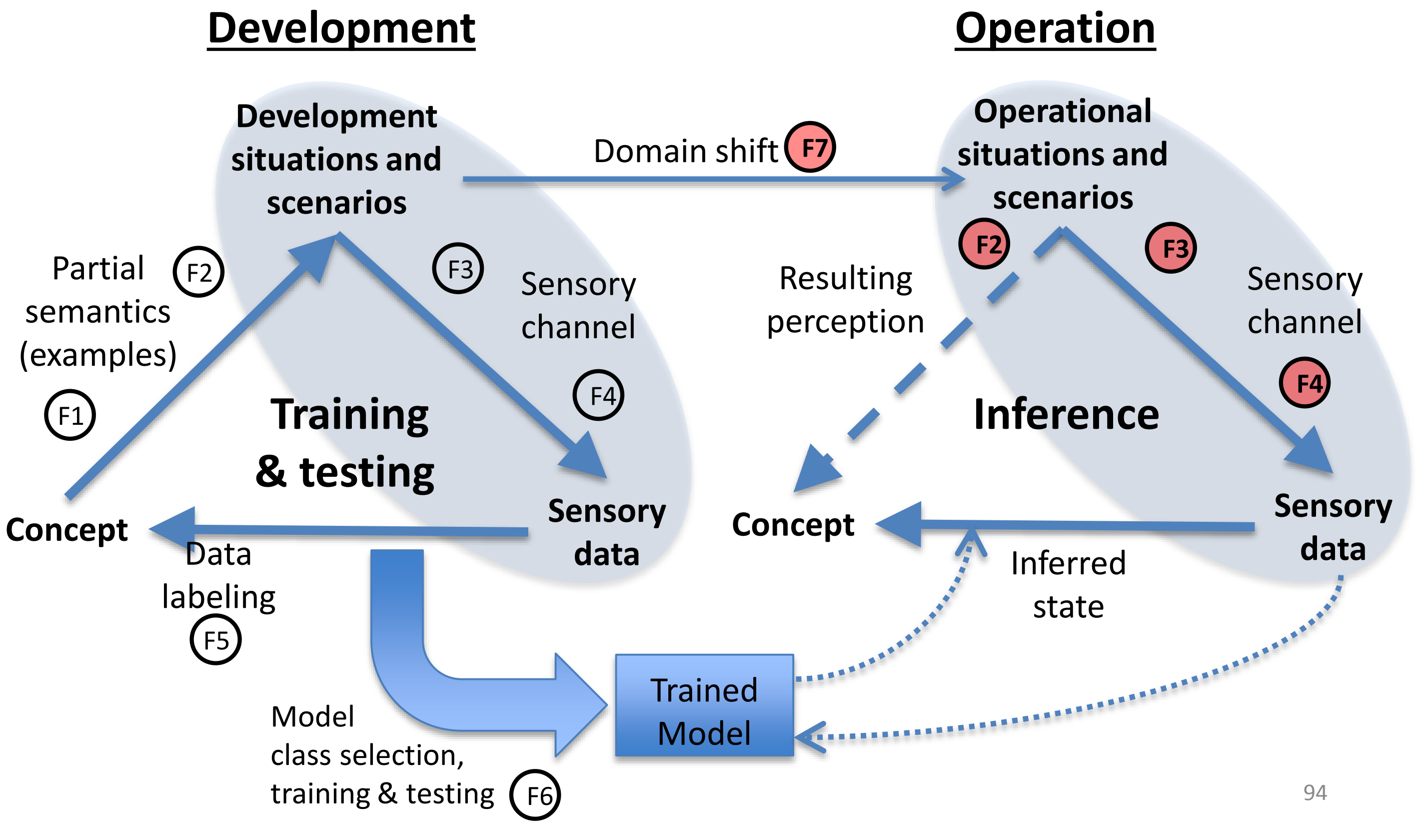
Methods for Confidence Estimation

1. Model uncertainty using MC Dropout
2. Data uncertainty using heteroschedastic regression
3. Confidence calibration



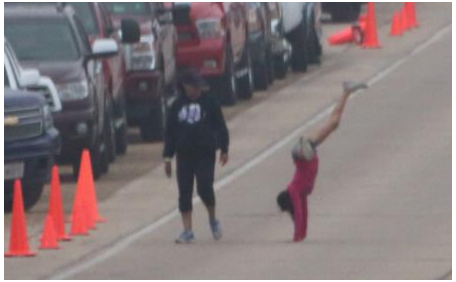
Phan, Salay, Czarnecki, Abdelzad, Denouden, Venekar. Calibrating Uncertainties in Object Localization Task. NIPS workshop. 2018, https://arxiv.org/abs/1811.11210

F7: Operational Domain Uncertainty



F7: Operational Domain Uncertainty

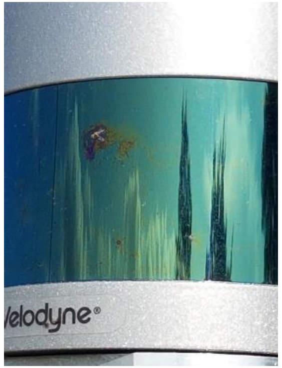
F2



New pedestrian pose

New type of car shape

F3 F4



Fly splatters on LIDAR

F4

Camera miscalibration

F7: Operational Domain Uncertainty

- Assess situation novelty at operation time
  - E.g., autoencoders, partial specs
- Assess impact of level of sensor miscalibration on perceptual uncertainty
- Monitor sensor parameters and ODD

Sample Incorrect Detections

Thank you

Questions?