RTL Regression Test Selection using Machine Learning

ASP-DAC 2022

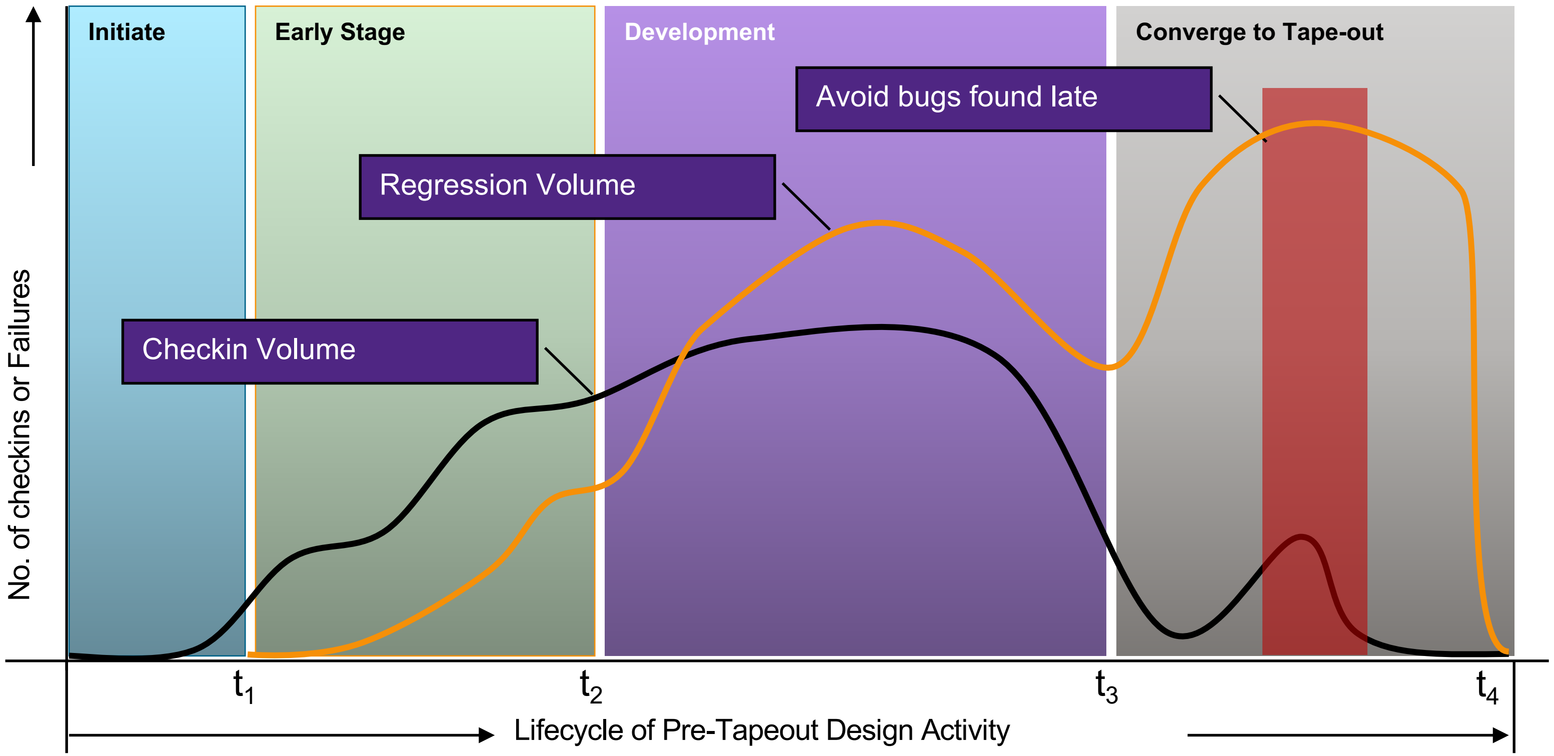
Ganapathy Parthasarathy*, Aabid Rushdi, Parivesh Choudhary, Saurav Nanda
Malan Evans, Hansika Gunasekara, and Sridhar Rajakumar

Agenda

- Problem Statement
- Background Work
- Overview
- Architecture
  - Data Model
  - ML Model Architecture
  - Why Ensemble?
- Experimental Results
- Conclusion

The RTL regression problem

Complexity of regressions in design life-cycle



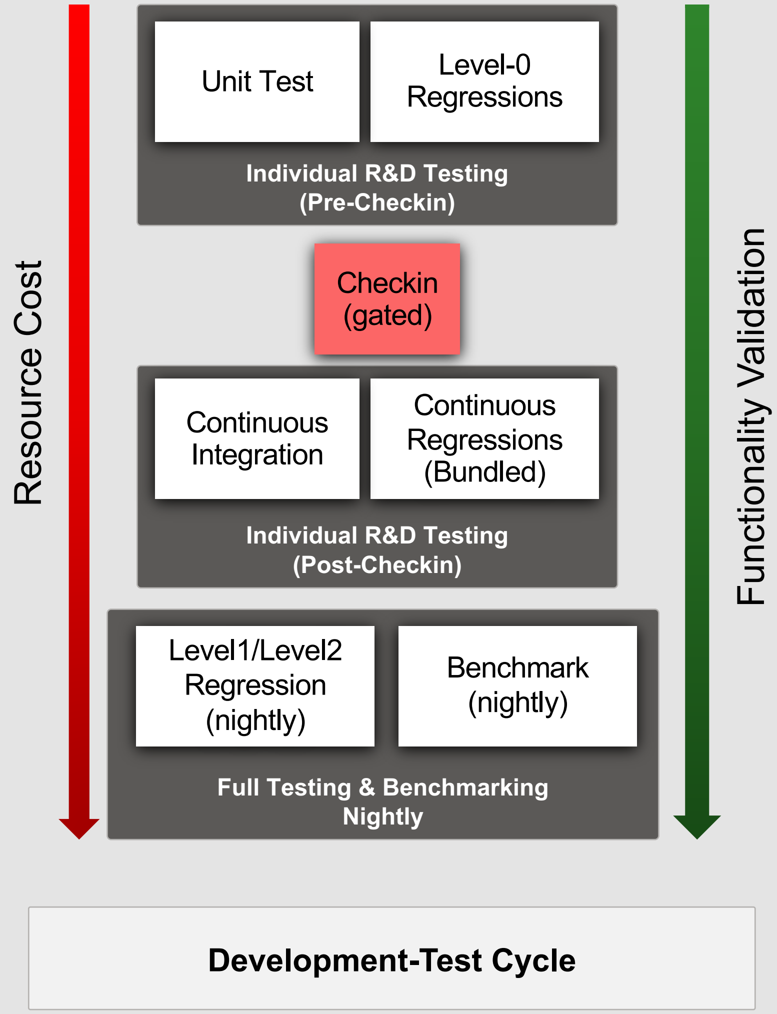
- **Assume**:
  - Regressions run from t1 to t4
  - # failures proportional to #check-ins
- **Early: Setup Design**
  - Test development
  - Low volume of regressions
- **Development: Develop Functionality**
  - Bulk of checkins are for functional changes
  - Relatively small number of test changes
  - Focus on fast/clean check-ins
  - Trade-off Bug-count v/s dev. speed
- **Converge: Functionally correct RTL**
  - Randomized regressions to find bugs
    - Explore design space maximally
    - Bulk of machine/license costs
  - Bulk of check-ins are for bug-fixes
  - Cost of late bug-fixes are very high

RTL Regression Test Selection Problem

Optimize RTL regressions across corners

- **Our approach: Model it as a probabilistic problem**
  - **Probabilistic TSP is formally defined as:**
  - **Given**
    - DUT_n (the n^{th} modified version of the DUT),
    - a test suite **T**, and a set of test requirements **R**
  - **Find**
    - a subset of tests, $\mathbf{T'} \in \mathbf{T}$ to test DUT_n such that **T′** achieves **R** with probability **P**.
    - Ideally, **T′** should contain all the test-cases in **T** that reveal faults in DUT_n.

- **Value**
  - **R&D: Higher Quality Checkins, Faster Development**
    - Gen. smaller high-quality set of tests for pre-checkin testing & validation
  - **DV/R&D: Catch Regression Failures early**
    - Front-load failures during periodic regression
  - **Infrastructure: Optimize on Resources and Quality**
    - Achieve same or higher quality targets with less (machines, licenses)



Prior Art

RTL Regression Test Selection Problem

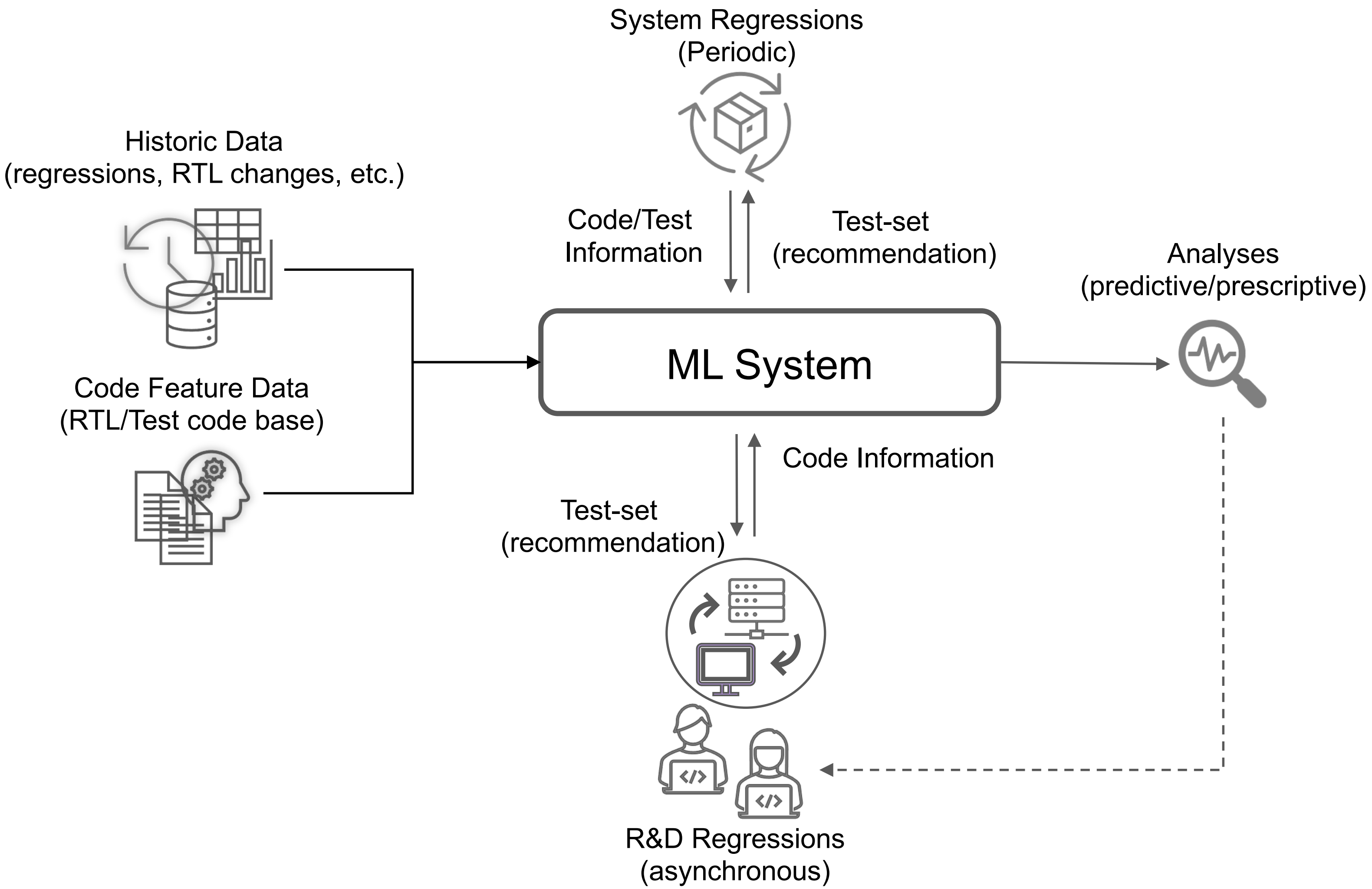
- Benjamin et al. and Guzey et al. focus on functional coverage targets.
- Gal et al. focused on bug detection.
- Farkash et al. uses a probabilistic metric based on coverage
  - Updated over time to rank-order files that are failure-prone.
  - File-based metrics used to optimize resources versus quality.
- Ioannides et al. surveys coverage-directed test-selection using ML
- Guzey et al. derives reduced test-set for functional coverage
  - Uses support vector machines
  - Learns an estimated mapping of the input sub-space of a given set of tests to a subset of tests.
  - represent equivalent functional coverage.
- **Significant prior art in Software (**Rothermel et. al and others**)**

- Our work and coverage-directed test selection techniques are orthogonal approaches.

- Our approach focuses on the probability of a bug being found independent of coverage targets.

RTL Test Selection using ML

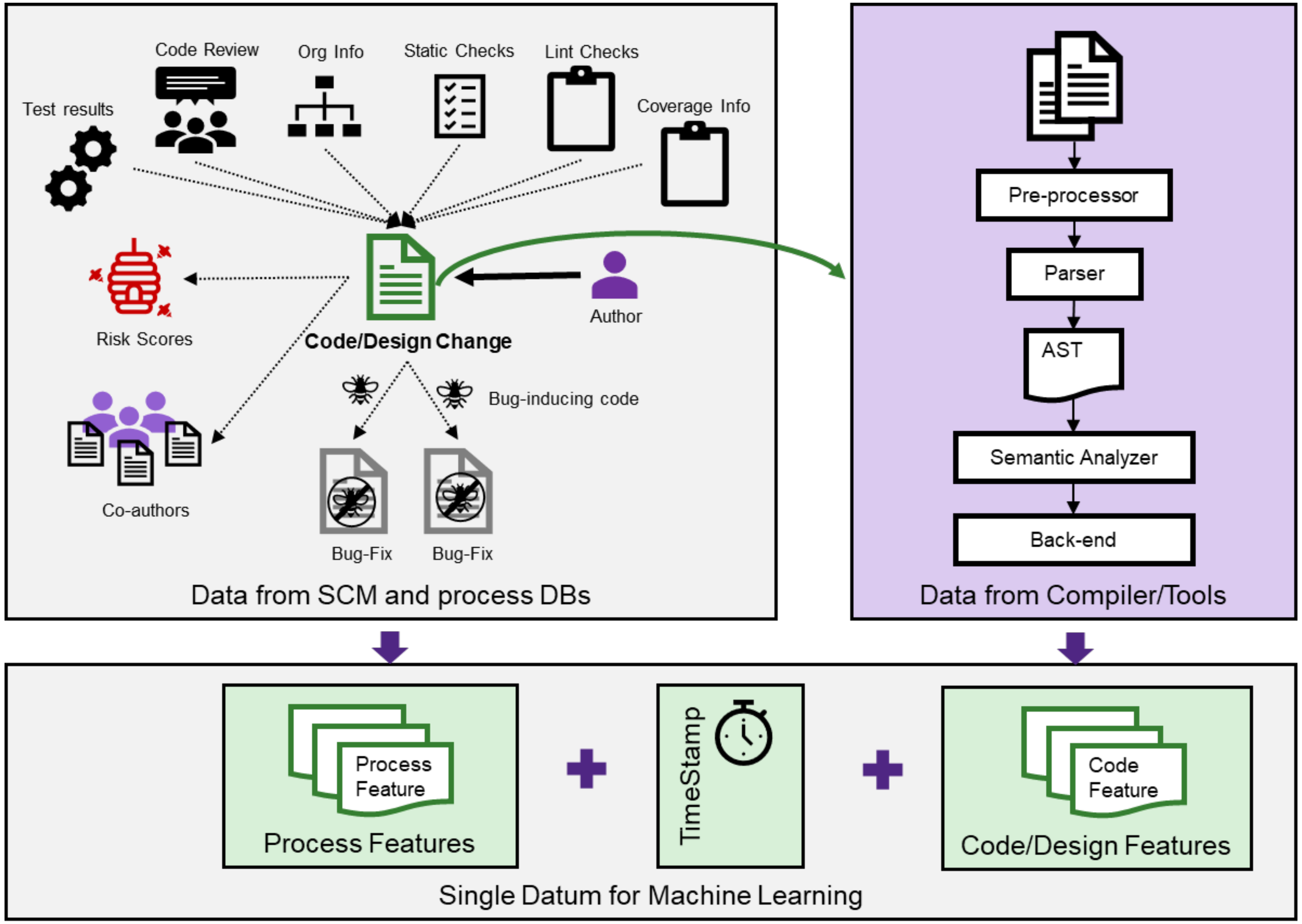
Overview



- **Optimize RTL regressions**
  - Cost of infrastructure, licenses
  - Time to completion
  - Quality of results
- **Adapt to design life-cycle**
  - Full ML pipeline in dynamic env.
  - Online ML model training
- **Minimal Total Cost of Ownership**
  - Lightweight Analysis
  - Minimal compute/disk requirements
  - Low latency
- **High reliability and availability**

Machine Learning Data Model

Core Datum – Code/Design/Test change and run data

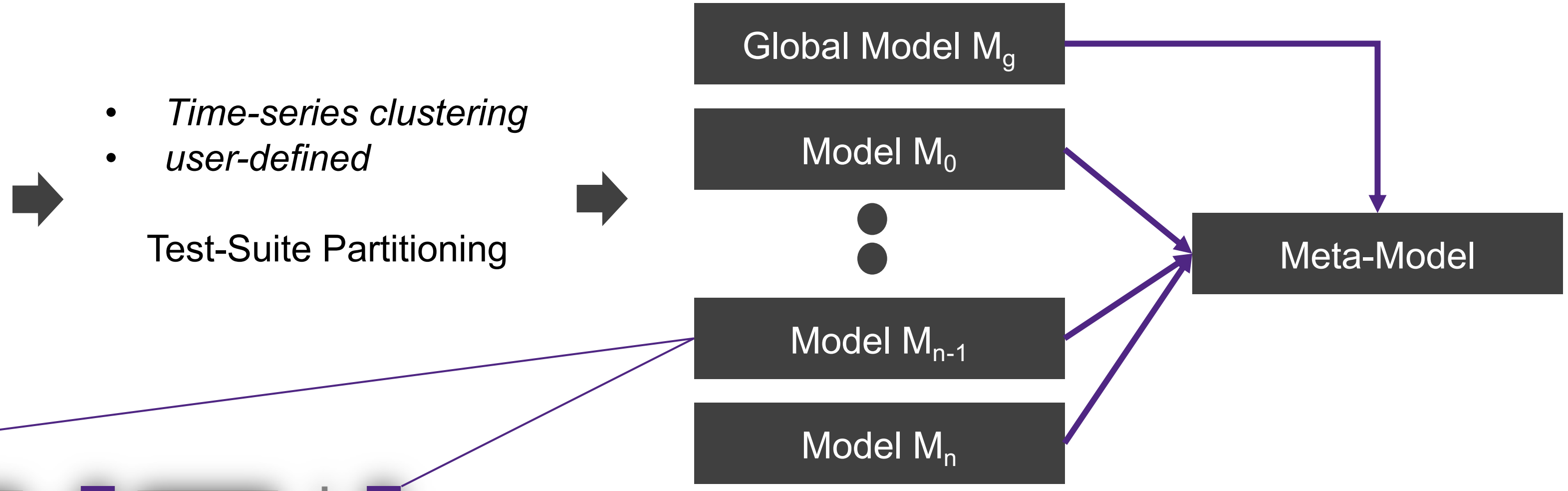


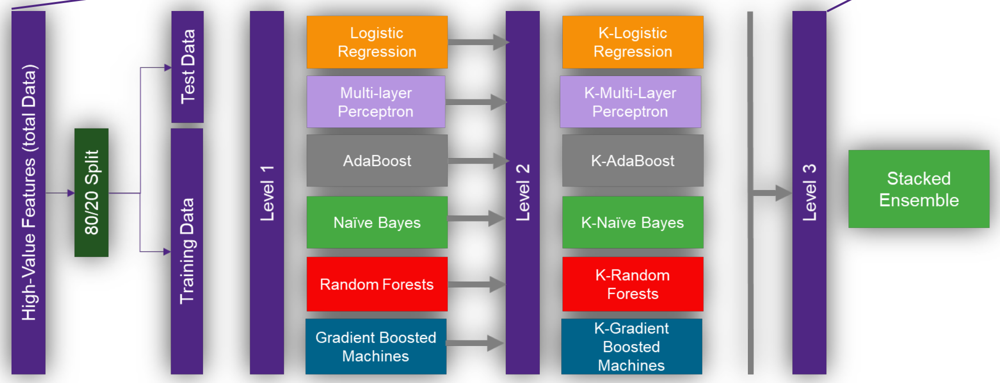
- **Information across time**
  - Timestamped Datums
- **Static Data per changelist**
  - Author, Files changed, Time-stamp
- **Linked Data from Ancillary DBs**
  - Bug-fixing changes
  - Regression/QOR/Coverage Results
  - Comments/defects from reviews
- **Derived Data**
  - Bug-inducing author, Other Authors
- **Compiler Data**
  - Hierarchical scope
  - complexity of change
  - Connectivity of change features

ML System Model Architecture

Ensemble model using individual predictors

Test-Suite Historical features



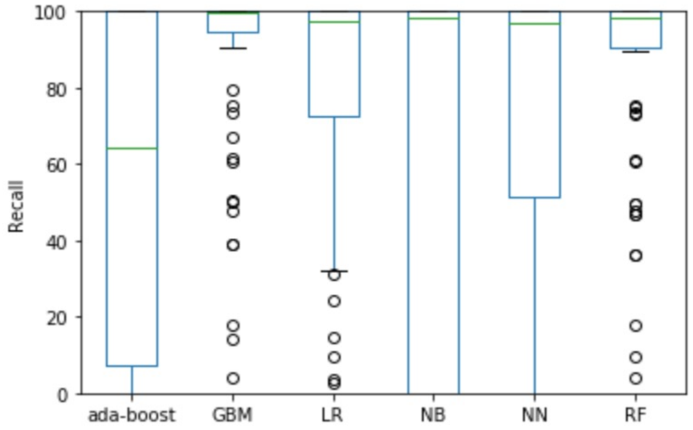


Core ML model pipeline

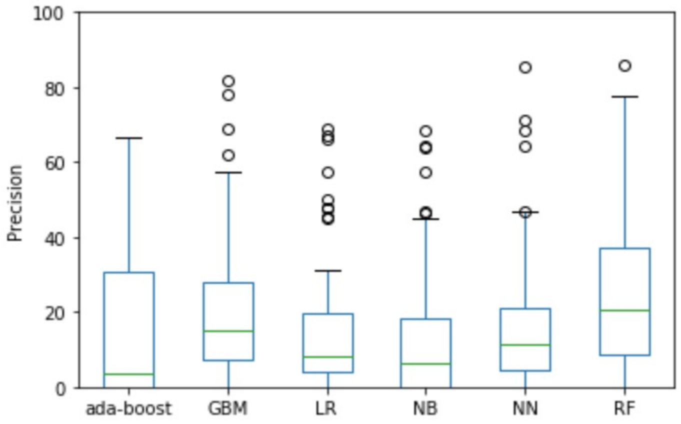
- One global model across all data
- Partition-level Model
    - Stacked ensemble per partition
    - Meta-learner to decide which model to use on the fly

Individual ML algorithms are not sufficient

Example on a real-world design



Test Efficiency (recall) Comparison



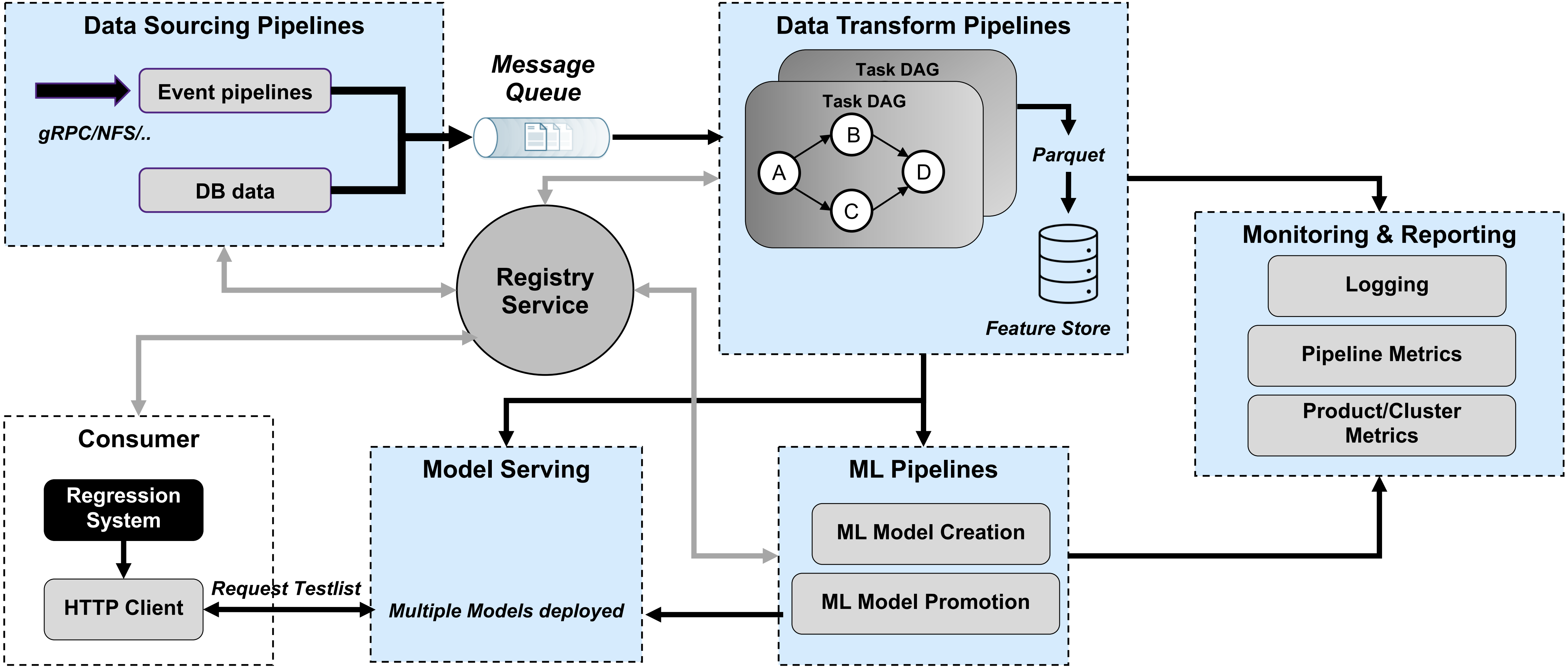
Precision Comparison

	GBM	LR	NB	NN	RF
Ada-boost	0.15	0.11	0.19	0.09	0.2
GBM		0.62	0.52	0.55	0.76
LR			0.6	0.72	0.5
NB				0.46	0.45
NN					0.44

- **Recall/Precision Comparison**
  - No single winner
- **Model performance Comparison**
  - Compare model estimated failure prob.
  - Order on estimated prob. of failure
  - Tversky index
    - Measures set similarity
    - Use on Ordered sets from models
    - See very different estimates

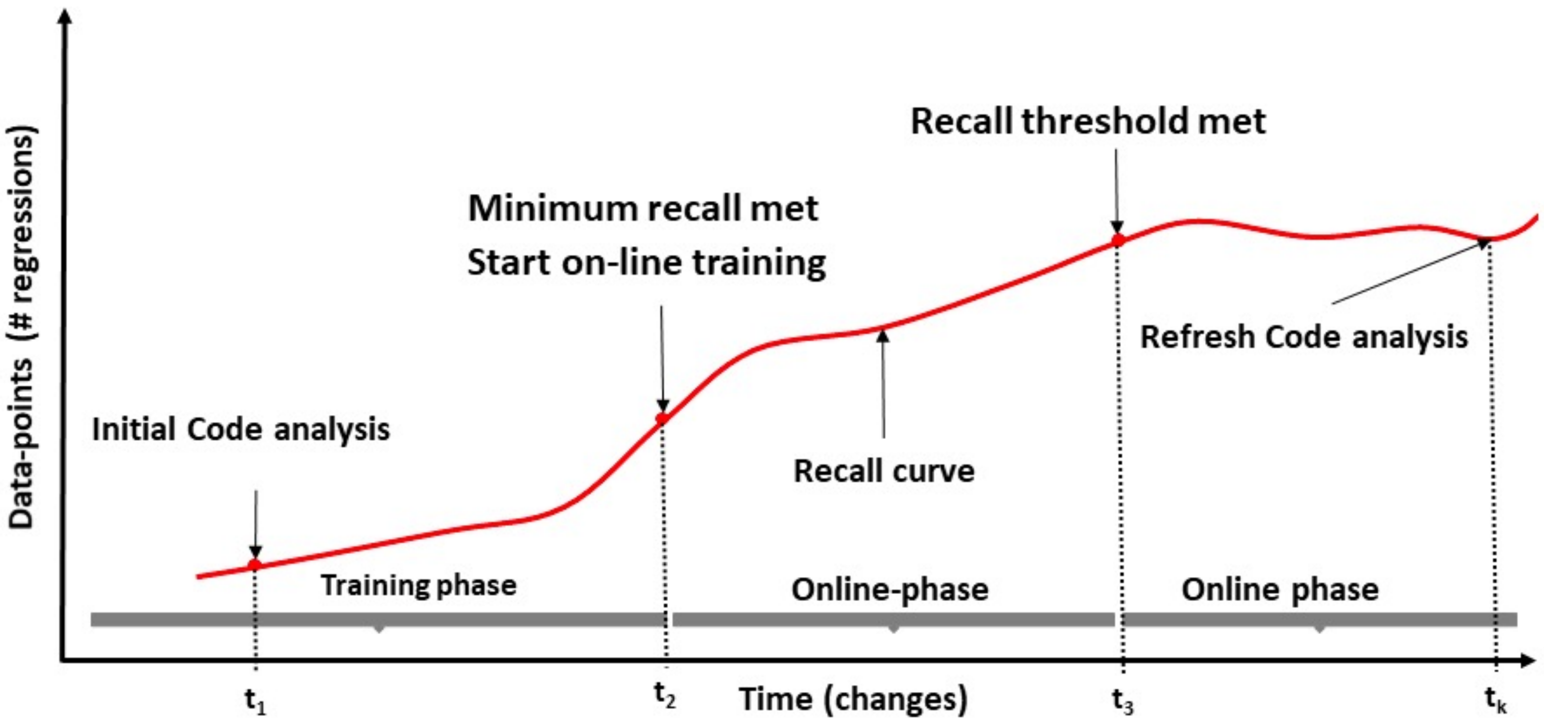
ML System Data Flow: Service Oriented Architecture

Distributed test/design data collection, transformation, and feature generation



Model Training Lifecycle

Online training and prediction



- **Train Model**
  - On n days of data (t_2)
- **Foreach changelist after training**
  - Tests actually run for current CL
  - Predict tests to be run for current CL
  - Compare predicted v/s actual
- **Incremental training after nth day**
  - **After** predictions for n+1th day
  - Combine changes in day
  - Use as n+1th training data-point
  - Serves as model for (n+2)th day
- **User defined minimum QOR**
  - User uses ML recommended tests
- **Periodic Data Refresh (t_k)**
  - Based on model metrics

Results

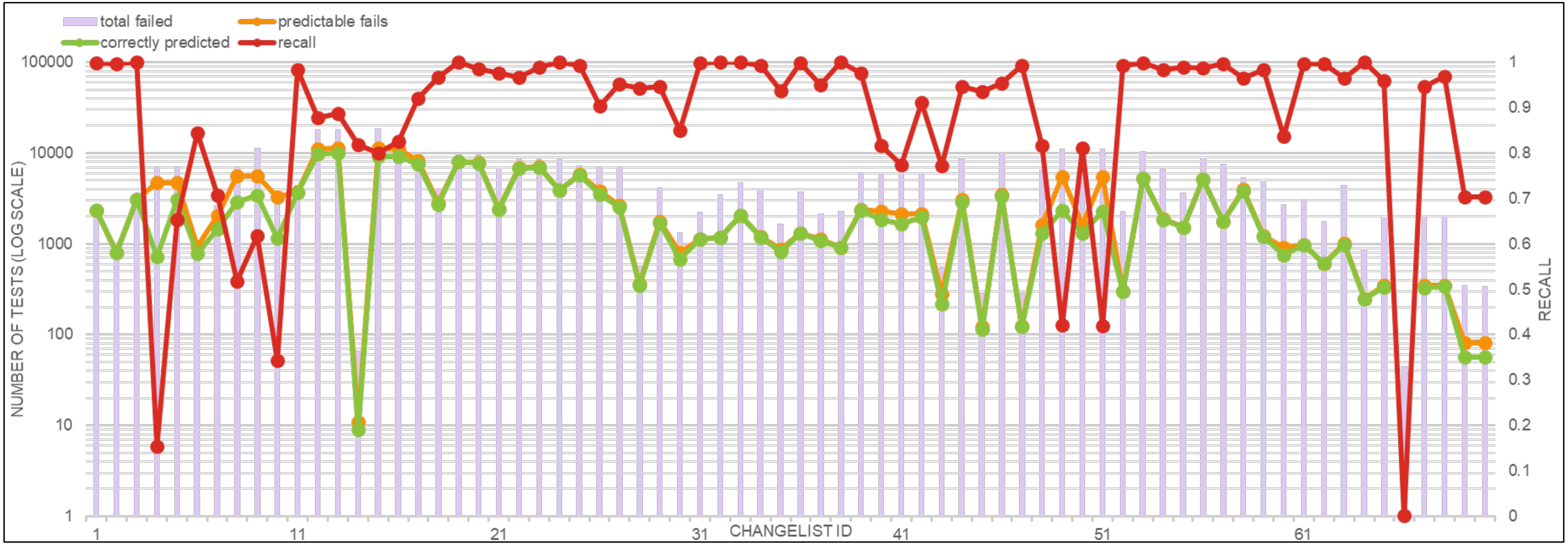
Prediction efficiency and test reduction results

Design	Configs	Training	# Regressions	Mean Recall	Median Recall	# Tests	Mean Predicted Tests	APFD	Reduction
D1	1	52	40	93.82%	98.01%	1311	843	70.93	35.69%
D2	13	36	62	91.88%	90.95%	1070	420	86.73	60.73%
D3	1	50	7	98.02%	99.56%	121207	9195	96.09	92.41%
D4	18	115	33	81.01%	100.00%	92	36	50.04	18.89%
D5	1	58	14	97.44%	100.00%	37	19	76.79	48.76%

- Regressions for all designs have a mix of randomized and directed tests
- Achieved > 90% median recall
- Reduced the avg no. of tests to be run by 10x to 20%
- Mean APFD scores shown demonstrates that ITS generates effective relative failure probabilities.
  - *Assume TF_i is first test case in ordered T, revealing fault i, then* ***AFPD*** $= 1 - \frac{TF_1 + TF_2 + \ldots + TFm}{nm} + \frac{1}{2n}$

Experimental Results (sample data on D3)

Model trained with 21 days of data. Predict on remaining 72 changelists over ~35 days



- Data for 158 changelists spread across 46 days, Total number of unique tests = 12107
- Recall: Mean = 88.14 %, Median = 95.36 %
- Median Size of predicted test-list = 9749. Reduction of ~10x.
- >99% of failing changelists captured with at least one true prediction

Conclusion

Discussion and future work

- Machine Learning test-selection in RTL functional verification flows is a promising approach.
- Experimental results support the utility of the method
  - Across distinctive design styles and test methodologies
  - High accuracy of detecting change-based failures
  - 35% to 10x reduction in regression size
- Limitations and Future work:
  - Variance in quality of results (QOR):
    - Dependent on the quality of data.
    - Can lead to unpredictable variations in the QOR that’s hard to resolve automatically.
  - Reduced QOR for constrained random tests

Thank You