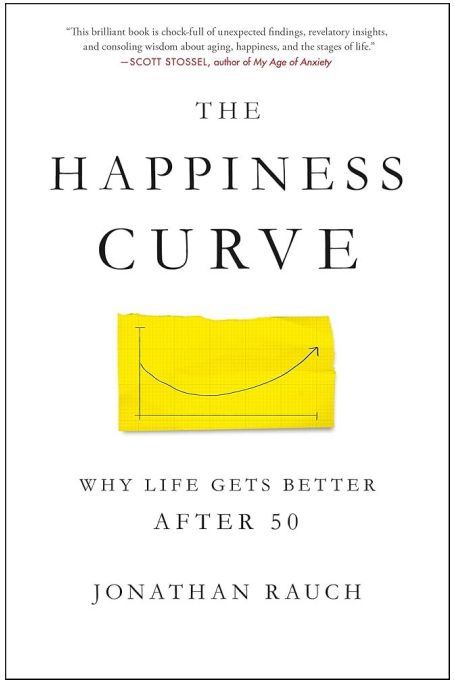
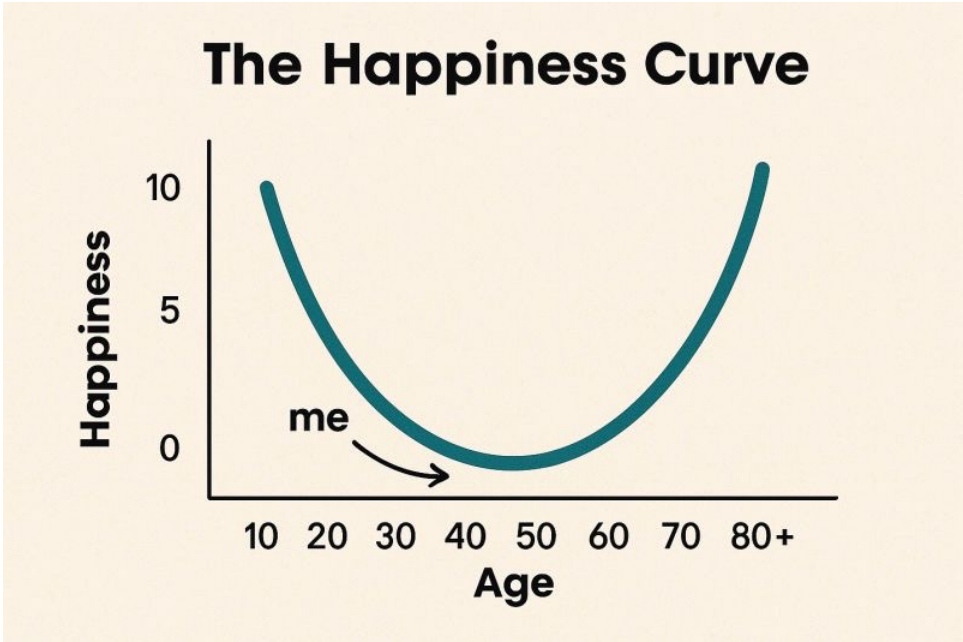
UCLA Samueli Computer Science

Happiness U-Curve: Navigating the AI Validation Bottleneck with Conformance Testing and Proof-Engineering

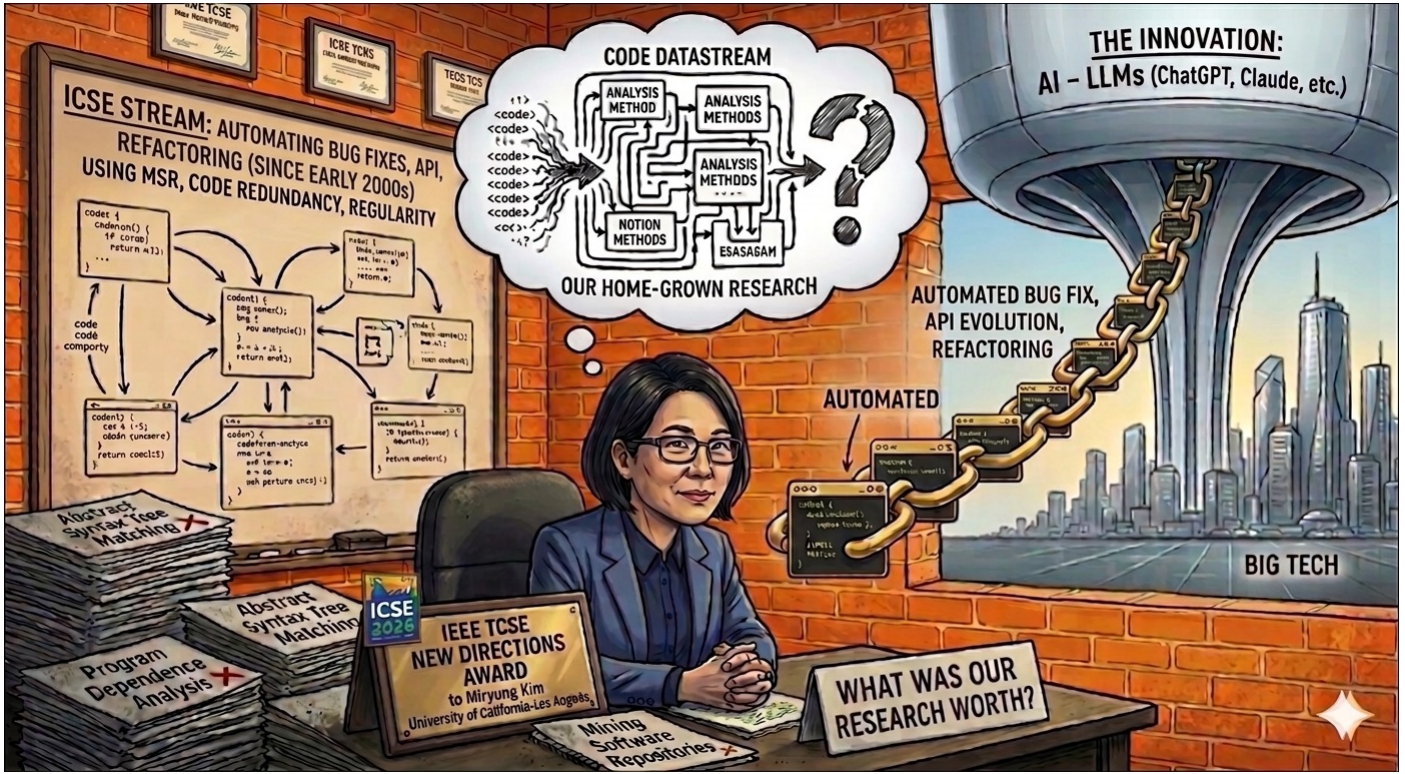
Miryung Kim
Professor and Vice Chair of Graduate Studies, UCLA Computer Science
Amazon Scholar at AWS
PLDI 2026 Keynote

Happiness U-Curve

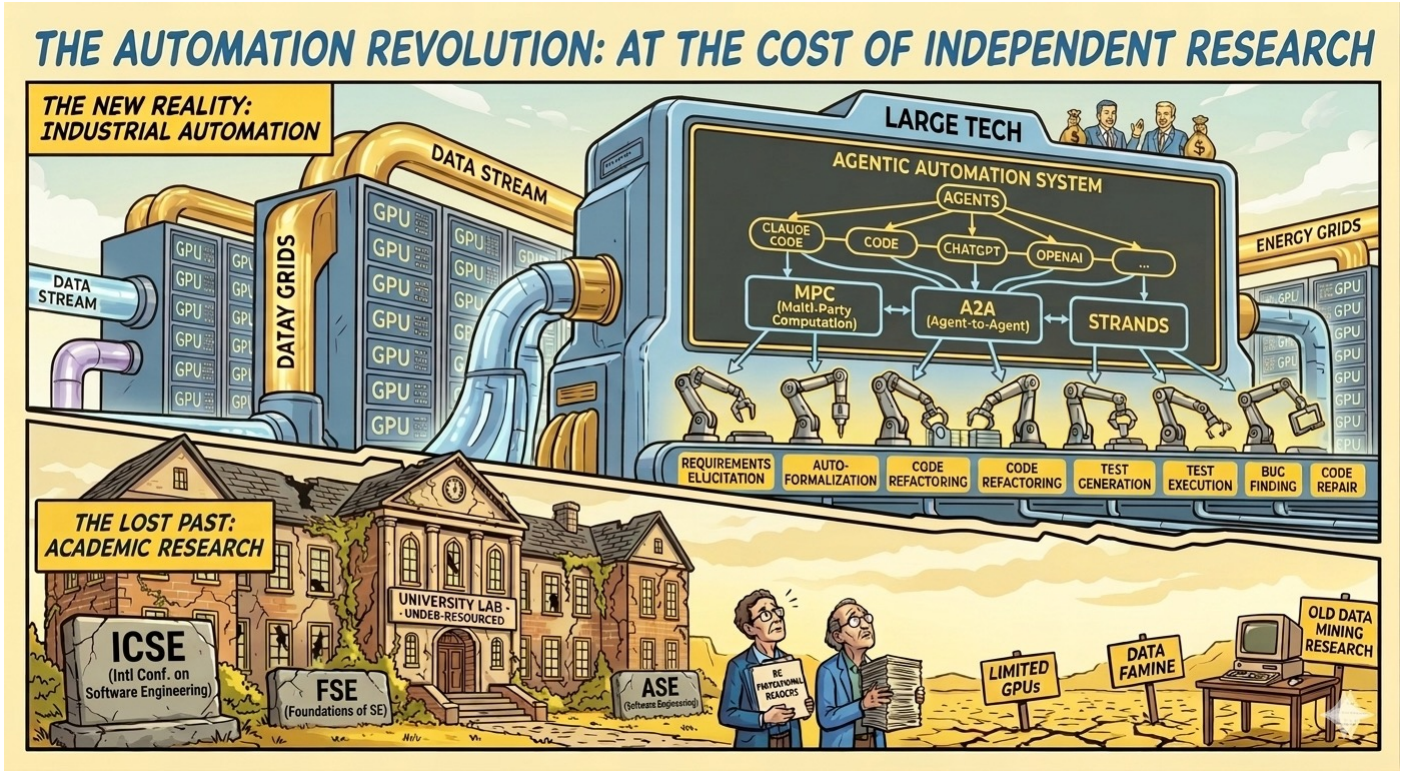




Existential Crisis of Data-Driven Software Analytics



Automation Revolution



Velocity Anecdotes

- **Professors**
  - Building world-class model counting solvers.
- **Founders**
  - Vibe-coding on a 3-hour drive from LA to San Diego.
- **Scientists**
  - Prototyping symbolic execution engines in 3-week sprints.

Velocity Anecdotes

- **Professors**
  - Building world-class model counting solvers.
- **Founders**
  - Vibe-coding on a 3-hour drive from LA to San Diego.
- **Scientists**
  - Prototyping symbolic execution engines in 3-week sprints.

- **Monthly Output**
  - 30 commits / month (mostly LEAN code)
- **Commit Density**
  - 91 files & 10.4K lines of code (LOC) per commit
- **Project Scale**
  - 367 files | 22K LOC produced in just 2 months

Vibe Coding: **The Toll on Quality**

How AI assistance impacts the formation of coding skills

Anthropic, Jan 2026

-17% quiz scores

Bad Vibes: AI-Generated Code is Vulnerable, Researchers Warn

Georgia Tech, Apr 2026

Increase in attack surfaces

Speed at the Cost of Quality

CMU, Jan, 2026

3-5X speed in first 2 months
+30% in warnings

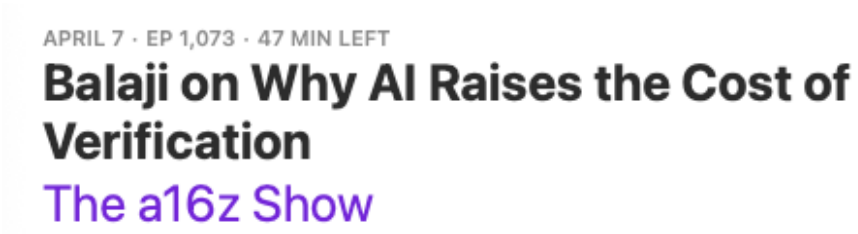
How Generative and Agentic AI Shift Concern from Technical Debt to Cognitive Debt

Margaret Storey, Feb 2026

Erosion in team understanding

Validation is the **new bottleneck in AI**

“Every tool that makes creation cheaper makes validation more expensive”

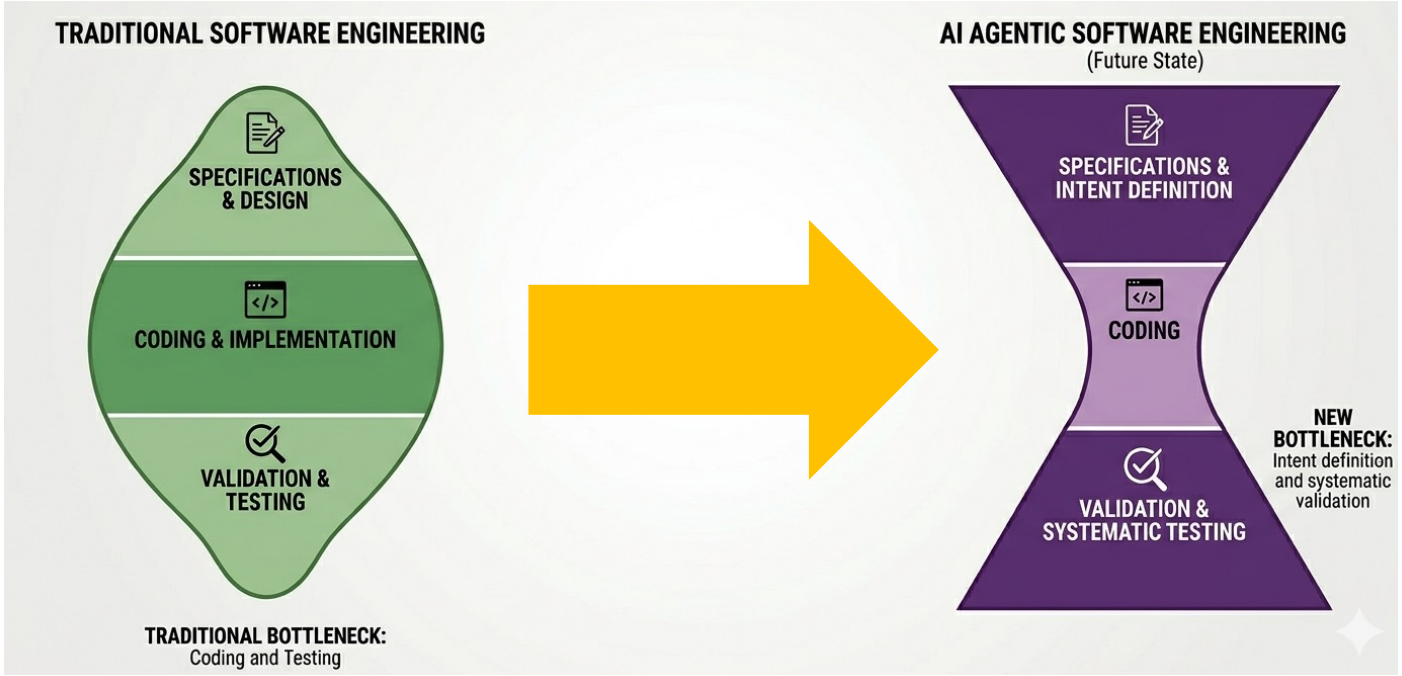


The bottleneck in AI engineering is no longer producing behaviour, but verifying it.

CSIRO

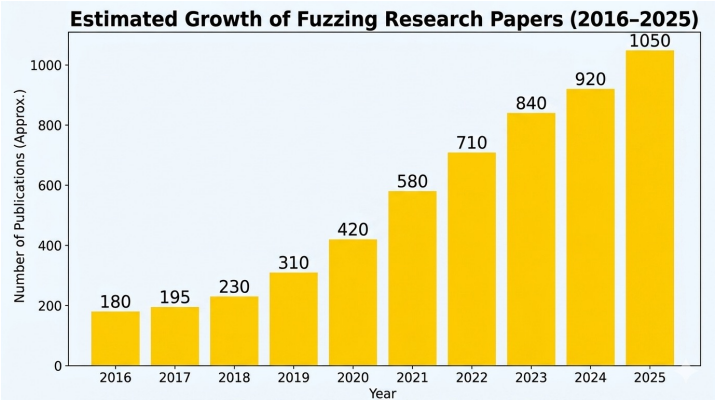
ICSE 2026 Keynote, “AI Engineering, from Software-Centric Systems to AIware-Centric Systems” Qinghua Lu

When the cost of creation becomes cheaper with AI, the **new bottleneck** is **validation.**



Talk Outline

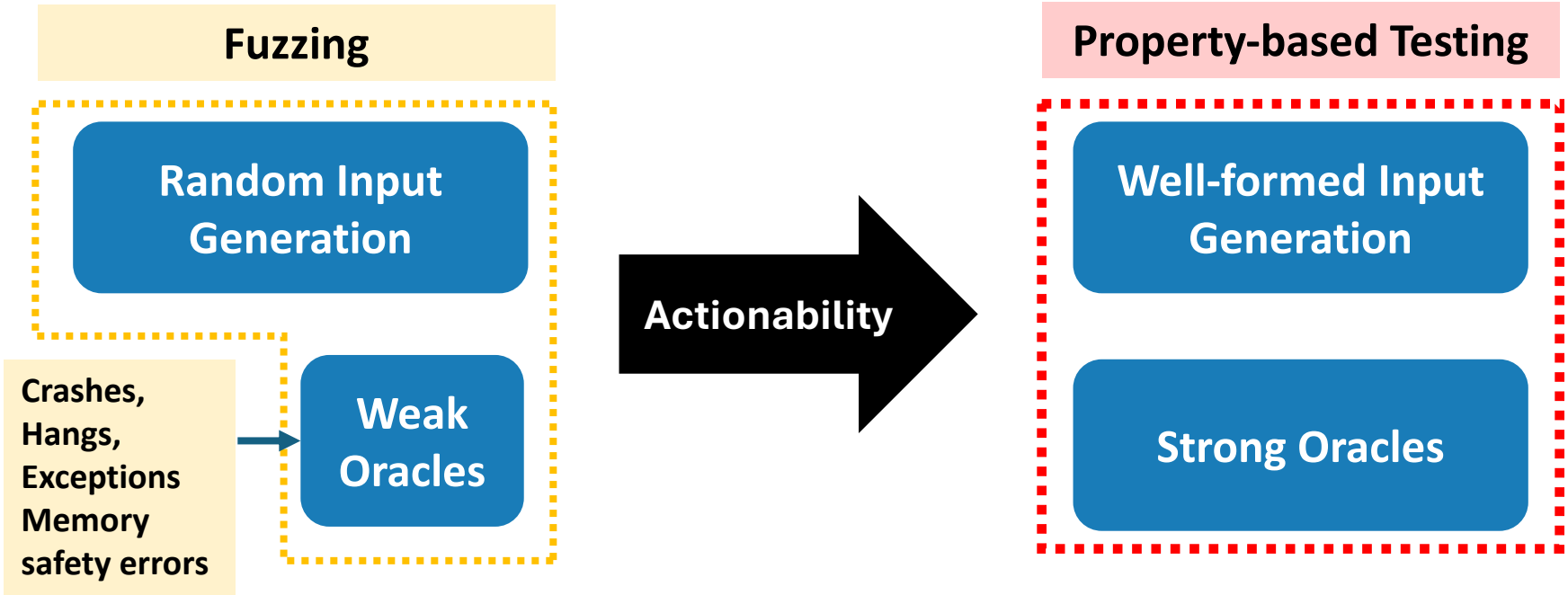
1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem
4. Harvest **recurring property patterns** and codify them as skeletons.



Shift from **Fuzzing** to **Property-based Testing**

"Fuzzing with well-formed input generation is necessary, but it does not solve the oracle problem, lacking actionability."

Terminology: Fuzzing & Property-based Testing



SE for Data Intensive Computing

Debugging

- Interactive Replay Debugger — BigDebug [ICSE 2016]
- Data Provenance — Titian [VLDB 2016]
- Incremental Optimization — Vega [SOCC 2016]
- Delta Debugging — BigSift [SOCC 2017]
- Performance Debugging — PerfDebug [SOCC 2019]
- Taint Analysis with Influence — FlowDebug [SOCC 2020]
- Spectra-based Operation Debug — OptDebug [SOCC 2021]

Testing

- Symbolic Execution w. Framework Abstraction — BigTest [FSE 2019]
- **Fuzz Testing** w. Framework Abstraction — BigFuzz [ASE 2020]
- Co-Dependence Aware **Fuzzing** — DepFuzz [FSE 2023]
- Natural **Fuzzing** — NaturalFuzz [ASE 2023]
- Natural Symbolic Execution — NaturalSym [FSE 2024]

SE for Heterogeneous Hardware

Transformation

- Interactive Refactoring for FPGA HLS
  HeteroRefactor [ICSE 2016]
- Automated Repair for FPGA HLS
  HeteroGen [ASPLOS 2022]

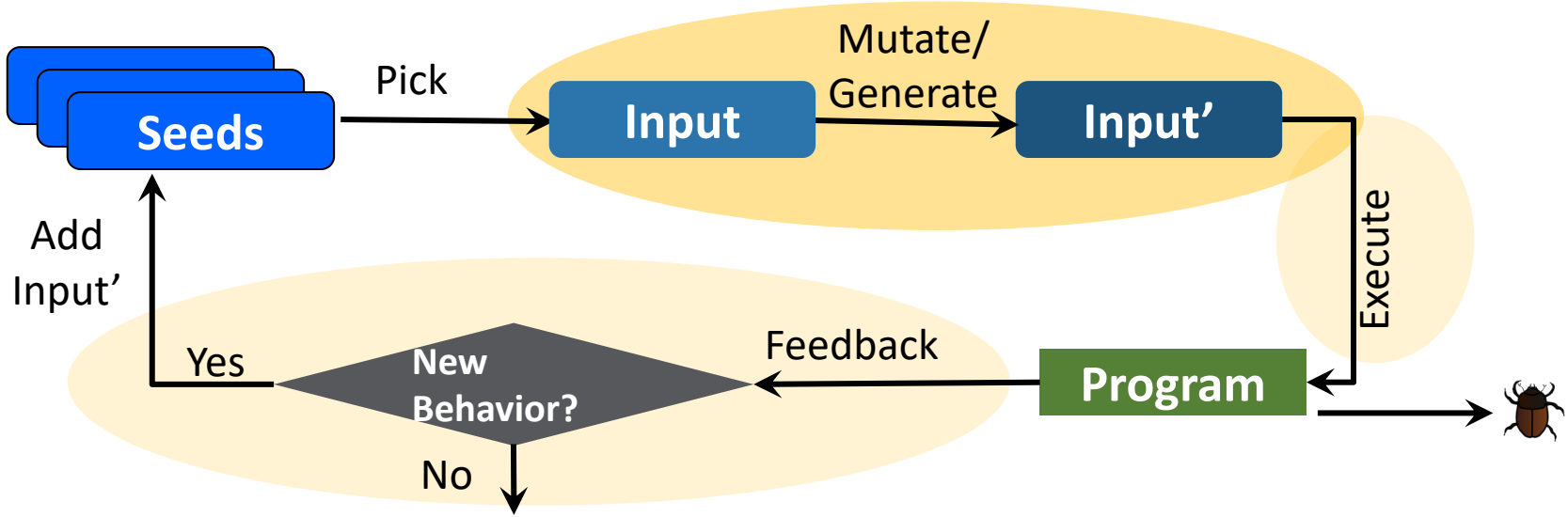
Debug

- Debugging MLIR Extensible Compilers
  DuoReduce [FSE 2025]

Testing

- Differential Testing for Quantum Frameworks
  QDiff [ASE 2021]
- **Fuzz Testing** w. HW Accelerator Spectra
  HeteroFuzz [FSE 2021]
- **Fuzzing** w. HW Probes & HW Acceleration
  HFuzz [FSE 2023]
- **Fuzzing** MLIR Compilers
  SynthFuzz [ICSE 2025]

Space of Fuzzer Specializations



Why Domain-Specialized Fuzzers? Fuzzers **cannot distinguish** ill-formed inputs from **incorrect outputs.**

Most errors are **input validation errors.**

With MLIR's Base Grammar:

```
"xklj.ZkdD"(%ixm) () -> (abdklx)
"o.xm"(%xlksdo) () -> ()
```

With MLIR HW Dialect Semantics:

```
"hw.module"() ({
  ^b(%a: i2):
    "hw.output"(%a) -> i2
})
```

Weak oracles such as

- Crashes
- Hangs
- Runtime exceptions
- Memory safety errors from sanitizers

Semantic input generators are necessary, but they **do not solve** the **oracle** problem

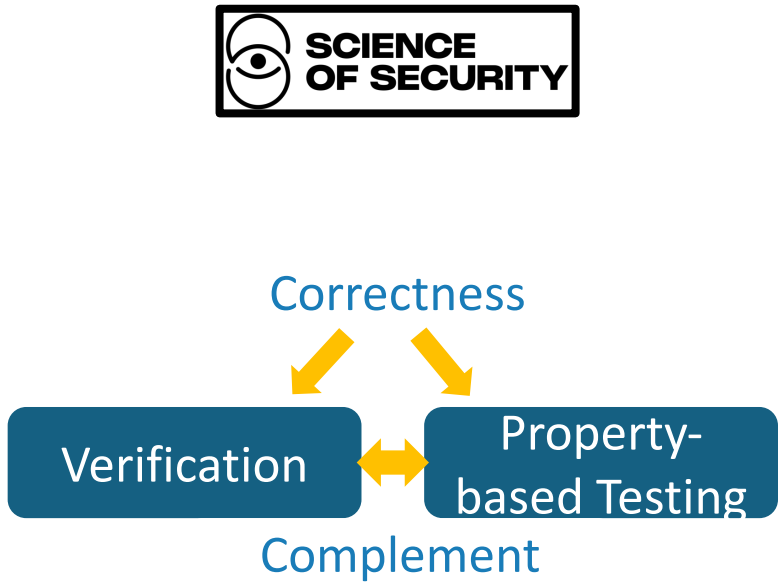
- Semantic generators produce **well-formed,** constraint-satisfying **inputs**
  - Fandango, ISLA
  - ProGRMR, SynthFuzz
  - QuickChick, Specimen

```
#Grammar
<start>::=<block> |
        "if "<paren_expr><stmt> |
        "int "<id>["="<expr>]";" |
        <id>"="<expr> | ...
# Def-use constraints
where forall <use_id> in
        <statement>..<expr>..<id>:
        exists <dec> in <declaration>:
                str(<dec>.<id>) == str(<id>) and
                is_before(<start>, <dec>, <use_id>)
```


José Antonio Zamudio Amaya, Marius Smytzek, and Andreas Zeller. 2025. FANDANGO: Evolving Language-Based Testing. Proc. ACM Softw. Eng. 2, ISSTA, Article ISSTA040 (July 2025), 23 pages. https://doi.org/10.1145/3728915

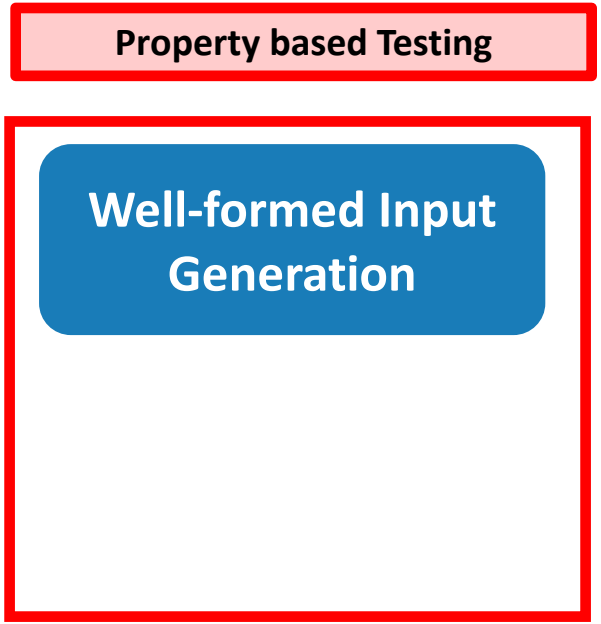
Change of Heart: Actionability from End-to-End Property-based Testing (1)

- Our team worked on a well-formed input generator.
- **Use Case Context:** The new authorization system's core logic was previously verified using formal methods and also need to be tested to ensure that the assumptions made at integration boundaries are correct.



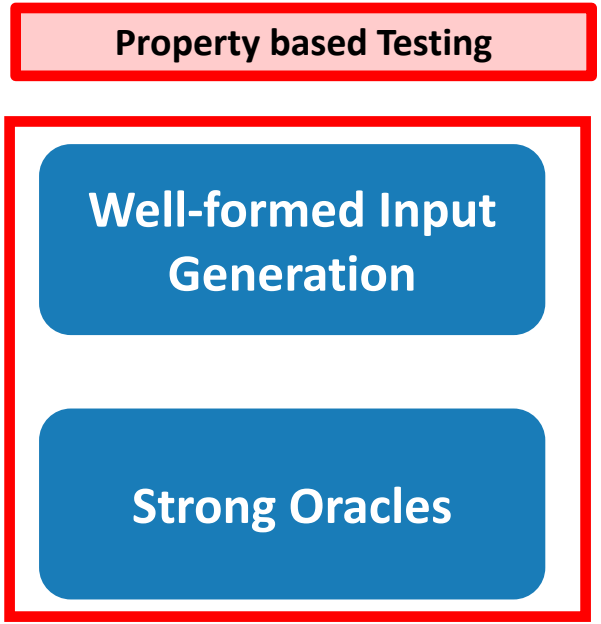
Change of Heart: Actionability from End-to-End Property-based Testing (2)

- **Test Input Generation:** A well-formed input generator
  - reduced test engine development effort by 75%,
  - improved coverage by 4%, and
  - detected 28% more pre-production issues,
  - all compared to a hand-coded generator.

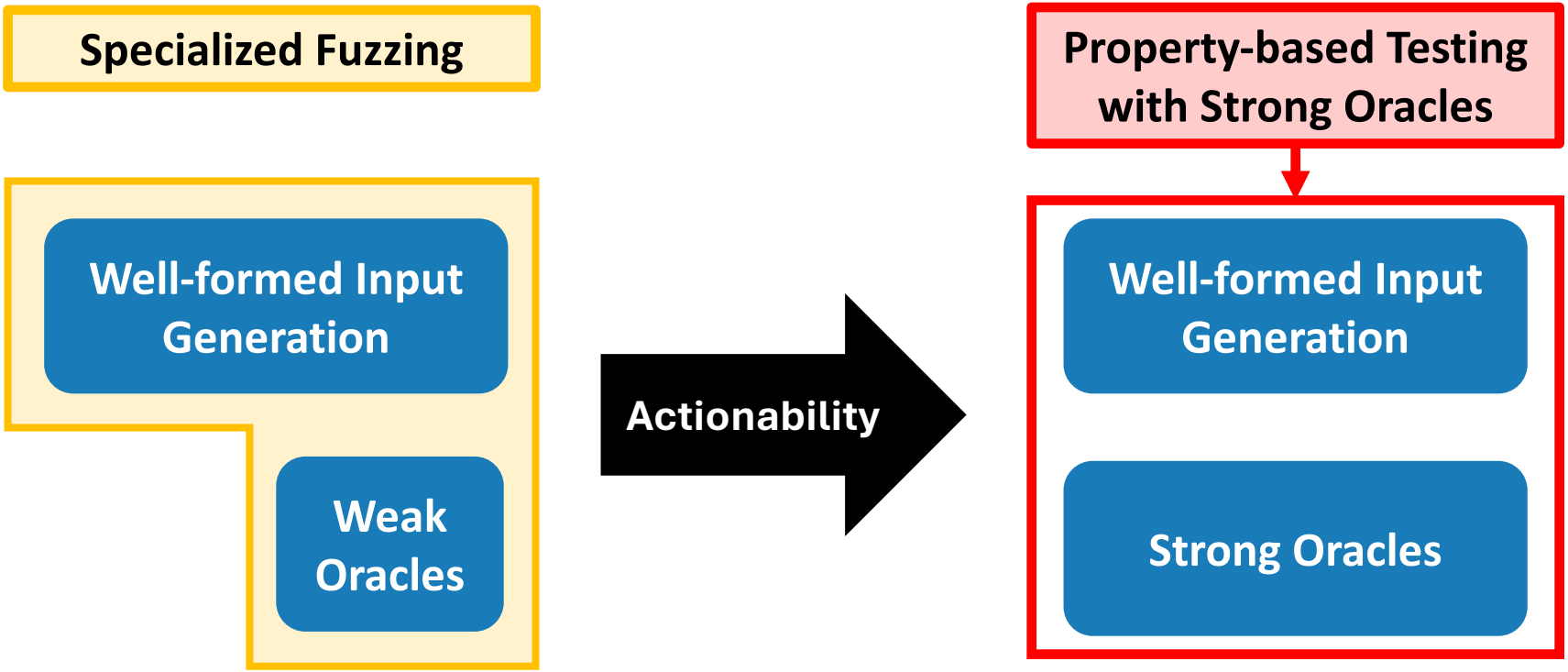


Change of Heart: Actionability from End-to-End Property-based Testing (3)

- **Test Oracle:** Approximately 80 property-based strong oracle checks were hand-coded.
- **Actionability:** **"All identified issues were remediated prior to production deployment."**



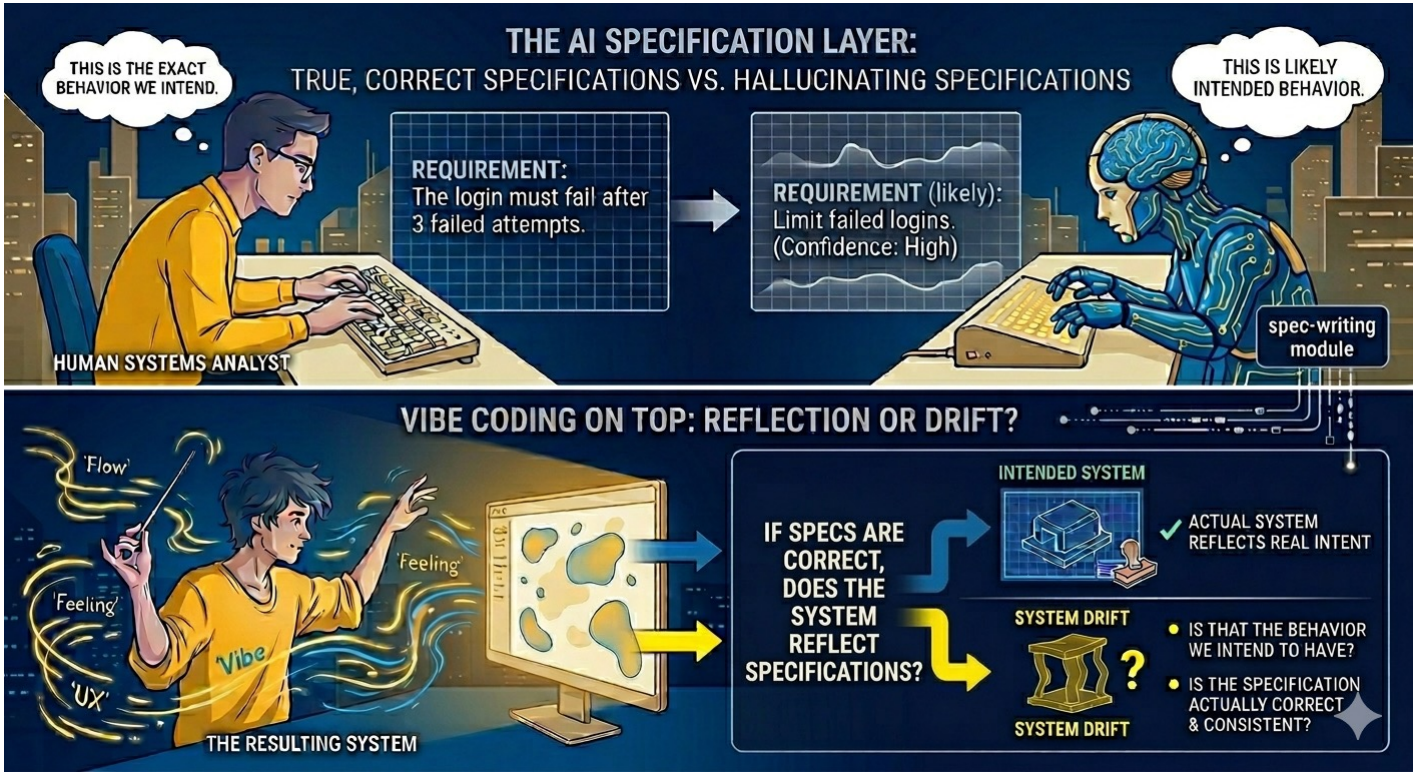
For actionability, we must solve **the spec problem**.



Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem
4. Harvest **recurring property patterns** and codify them as skeletons.
   - H1: Proof-skeletons ease proof-engineering
   - H2: PBT-skeletons amplify PBT effectively

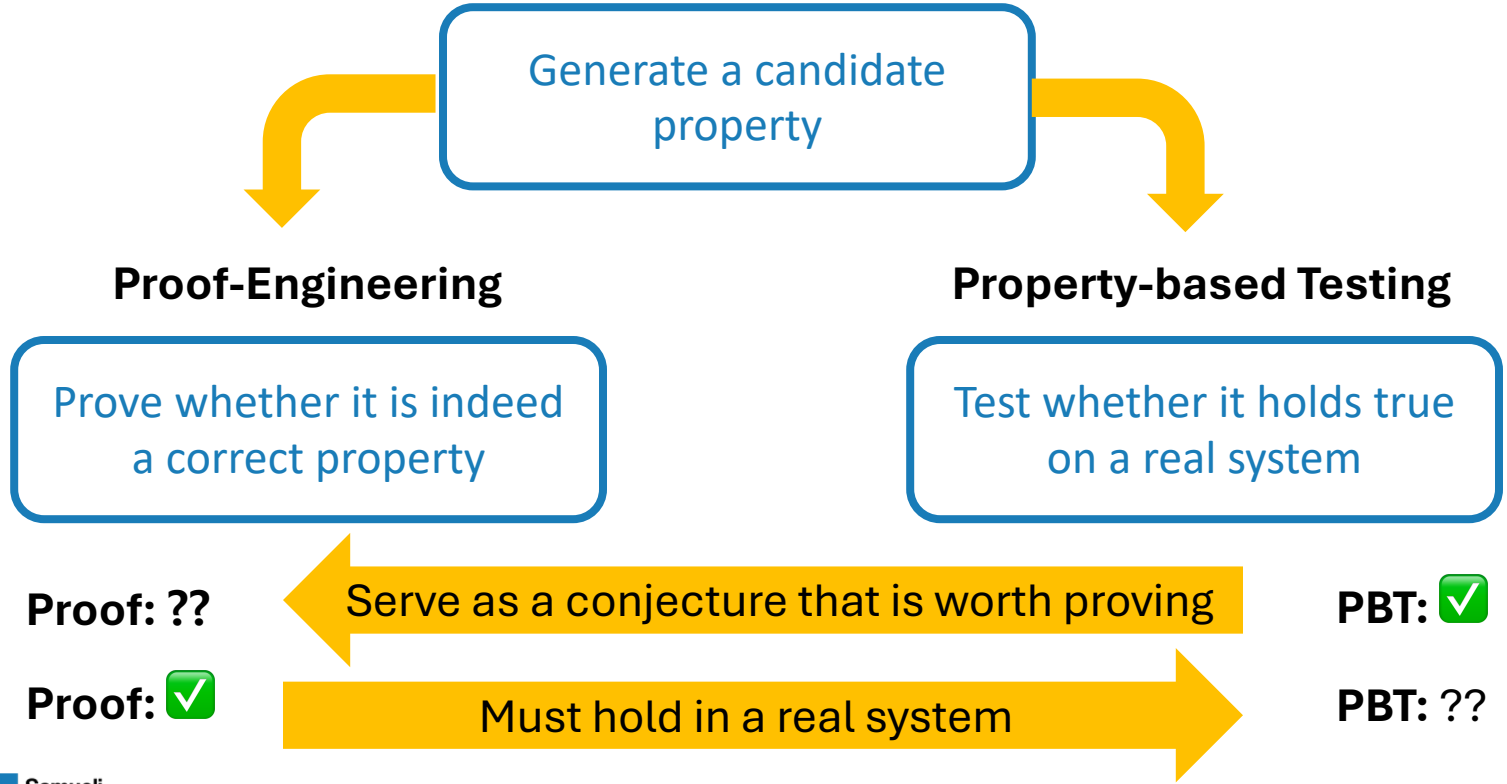
Specifications are also being generated by AI.



UCLA Samueli Computer Science

Intent Formalization: A Grand Challenge for Reliable Coding in the Age of AI Agents, Lahiri 2026

Dual Track Validation to Solve "Spec" Problem



UCLA Samueli Computer Science

On the Unreasonable Effectiveness of Property-Based Testing for Validating Formal Specifications, Feng and Sergey, May 2026

Key Insight: Recurring patterns in specifications

Patterns in Property Specifications for Finite-State Verification*

Matthew B. Dwyer
Kansas State University
Department of Computing and Information Sciences
Manhattan, KS 66506-2302
+1 785 532 6350
dwyer@cis.ksu.edu

George S. Avrunin
University of Massachusetts
Department of Mathematics and Statistics
Amherst, MA 01003-4515
+1 413 545 4251
avrunin@math.umass.edu

James C. Corbett
University of Hawai'i
Department of Information and Computer Science
Honolulu, HI 96822
+1 808 956 6107
corbett@hawaii.edu

ABSTRACT

Model checkers and other finite-state verification tools allow developers to detect certain kinds of errors automatically. Nevertheless, the transition of this technology from research to practice has been slow. While

cess support for formal methods.

We believe that the recent availability of tool support for finite-state verification provides an opportunity to overcome some of these barriers. Finite-state verification refers to a set of techniques for proving properties

2021 ACM SIGSOFT IMPACT PAPER AWARD (ICSE 1999, Los Angeles, California, May 1999).
"For enabling widespread use of temporal logic for program verification by raising the level of abstraction to common patterns."

Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem—how could we generate properties to use for PBT?
4. Harvest **recurring property patterns** and codify them as skeletons.

- H1: Can proof-skeletons ease proof-engineering?
- H2: Can PBT-skeletons amplify PBT effectively?

UCLA Samueli Computer Science

https://github.com/UCLA-SEAL/RealProof

Agentic Proof and Property-Based Testing via **Property-Skeletons in Data-Intensive Computing**

Seongmin Lee*

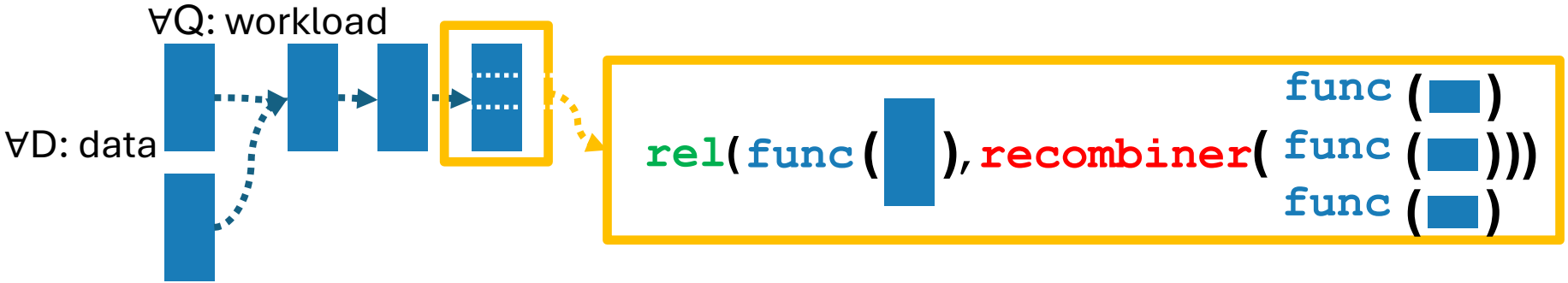
Thaddy Yaoxuan Wu*

Miryung Kim

UCLA Samueli Computer Science

Example Recurring Properties in Data Intensive Systems: Aggregation Decomposition

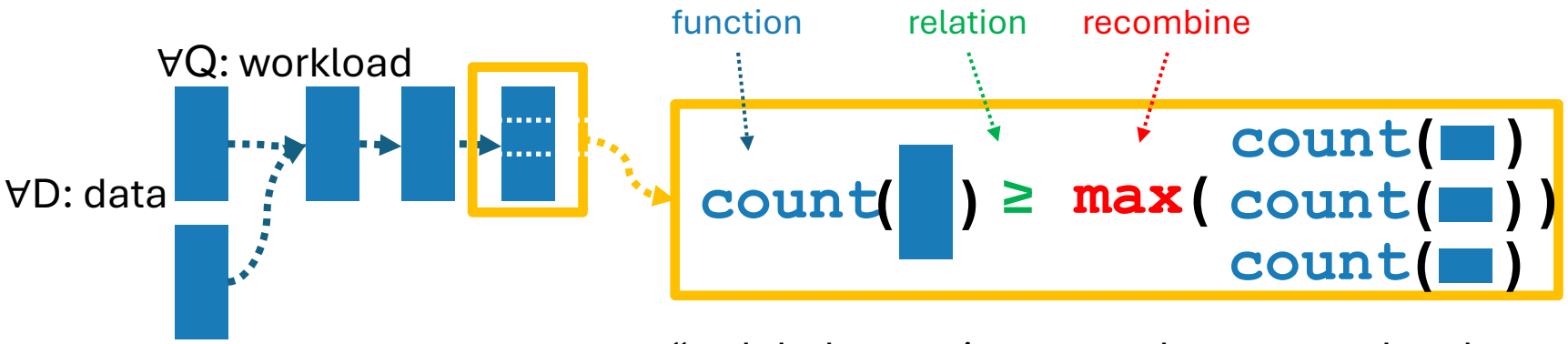
For all data D and workload Q, global `func` on the entire dataset has `relation` to local `func` followed by `recombiner`



"Operationalizing Property-Based Testing for Data-Intensive Scalable Computing Systems", Yaoxuan Wu et al. 2026 https://arxiv.org/abs/2606.11132

UCLA Samueli Computer Science

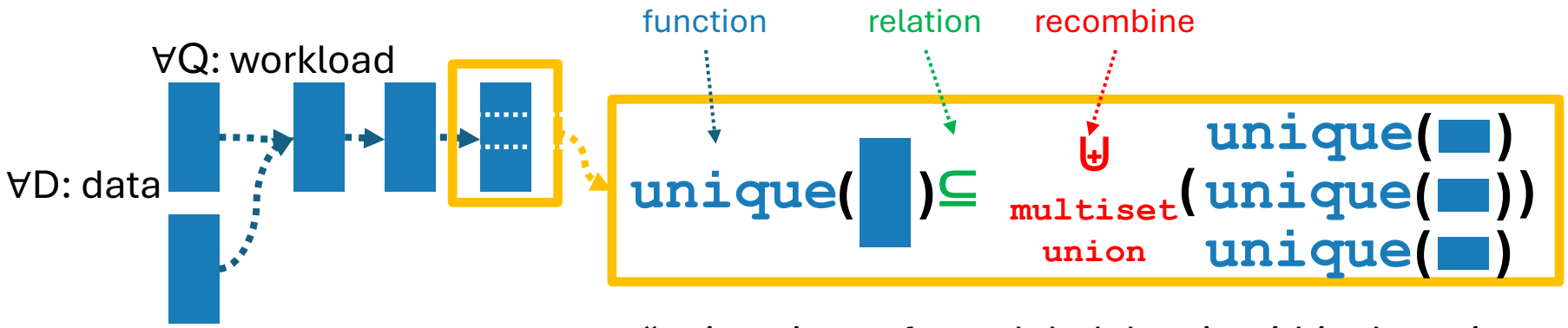
Concrete Instantiation 1: Aggregation Decomposition "a global count is greater or equal to max of sub counts"



"a global count is greater than or equal to the maximum of local counts"

"Operationalizing Property-Based Testing for Data-Intensive Scalable Computing Systems", Yaoxuan Wu et al. 2026 https://arxiv.org/abs/2606.11132

Concrete Instantiation 2: Aggregation Decomposition "unique items are contained in union of unique items"

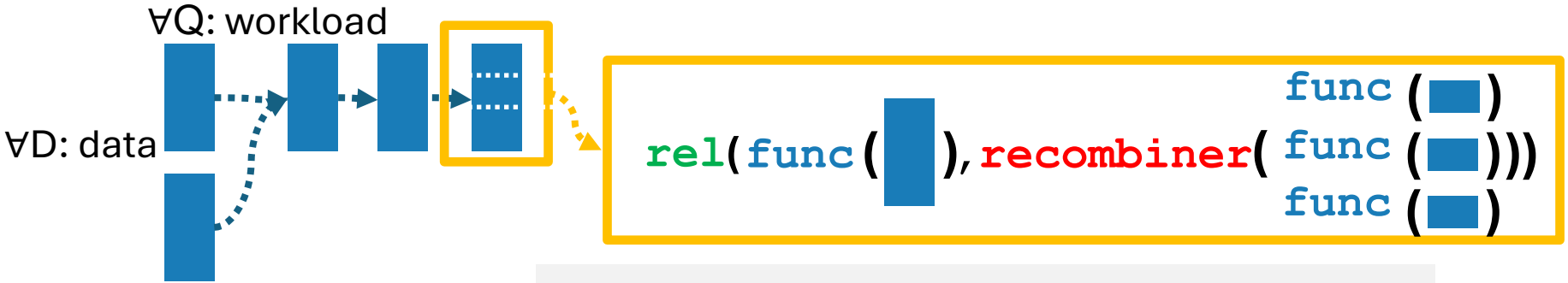


"unique items from global data is within the union of unique items on local partitions"

"Operationalizing Property-Based Testing for Data-Intensive Scalable Computing Systems", Yaoxuan Wu et al. 2026 https://arxiv.org/abs/2606.11132

Aggregation Decomposition Property Skeleton
in Data Intensive Systems

For all data D and workload Q, global `func` on the entire dataset has `relation` to local `func` followed by `recombiner`



Aggregation decomposition

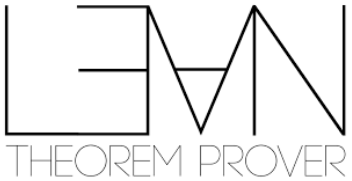
∀D,∀Q,∀key,∀func,∀recombiner,∀relation:
(func,recombiner,relation)∈Fagg⟹
relation(execute(D,Q.agg(func)),
execute(D,recombiner(Q.groupBy(key).agg(func))))

Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem—how could we generate properties to use for PBT?
4. Harvest **recurring property patterns** and codify them as skeletons.
   - H1: Can proof-skeletons ease proof-engineering?
   - H2: Can PBT-skeletons amplify PBT effectively?

The Proof World:
Diversify Properties and Ease Proof-Engineering

Proof: Model World



LOC	**2030**
Data Types	**16 (`Row`, `Database`, `Expr`, `DFOp`, etc.)**
Functions & Relations	**28 (`Row.append`, `runExpr`, etc.)**
Theorems	**71**

Proof Skeletons

① Hypothesize Candidate Lemmas

Agent

② Complete Proofs with Lemmas

Agent

Example Proof Skeleton in LEAN

Proof Agent

THEOREM PROVER

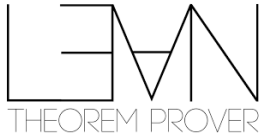
The Model World

Aggregation decomposition:
"For all data and computation pipelines, applying function globally has a relation to applying function and recombiner."

```
theorem skeleton_aggregation_decomposition
  (function : DB → α) (recombiner : List α → α) (relation : α →
α → Prop)
  (decompose_law : ∀ db : DB,
   relation (function db) (recombiner ((groupBy key
db).map function)))
  (pre : Workload) (init_db : DB) :
  relation
   (function (pre init_db))
   (recombiner ((groupBy key (pre init_db)).map function)) :=
by ...
```

Example Proof Skeleton in LEAN

Proof Agent



The Model World

Aggregation decomposition: For all data and computation pipelines, applying **function** globally has a **relation** to applying **function** and **recombiner**.

```
theorem skeleton_aggregation_decomposition
  (function : DB → α) (recombiner : List α → α) (relation : α →
α → Prop)
  (decompose_law : ∀ db : DB,
    relation (function db) (recombiner ((groupBy key
db).map function)))
  (pre : Workload) (init_db : DB) :
  relation
    (function (pre init_db))
    (recombiner ((groupBy key (pre init_db)).map function)) :=
by ...
```

Subgoal Lemma

Agentic Concrete Proof Synthesis

①**Hypothesize** a concrete lemma

```
(decompose_law : ∀ db : DB,
    relation (function db) (recombiner
((groupBy key db).map function)))
```

function → `count`
relation → ≥
recombiner → `max`

Proof Agent

THEOREM PROVER

Agentic Concrete Proof Synthesis

① **Hypothesize** a concrete lemma

```
(decompose_law : ∀ db : DB,
    relation (function db) (recombiner
((groupBy key db).map function)))
```

function → count
relation → ≥
recombiner → max

② **Prove** a specific lemma

```
theorem count_ge_max_decompose_law (db : DB) :
  count db ≥ max ((groupBy key db).map count) := by
  ...
```

Agentic Concrete Proof Synthesis

① **Hypothesize** a concrete lemma

```
(decompose_law : ∀ db : DB,
    relation (function db) (recombiner
((groupBy key db).map function)))
```

function → count
relation → ≥
recombiner → max

② **Prove** a specific lemma

```
theorem count_ge_max_decompose_law (db : DB) :
  count db ≥ max ((groupBy key db).map count) := by
  ...
```

③ **Derive** specialized end-to-end property

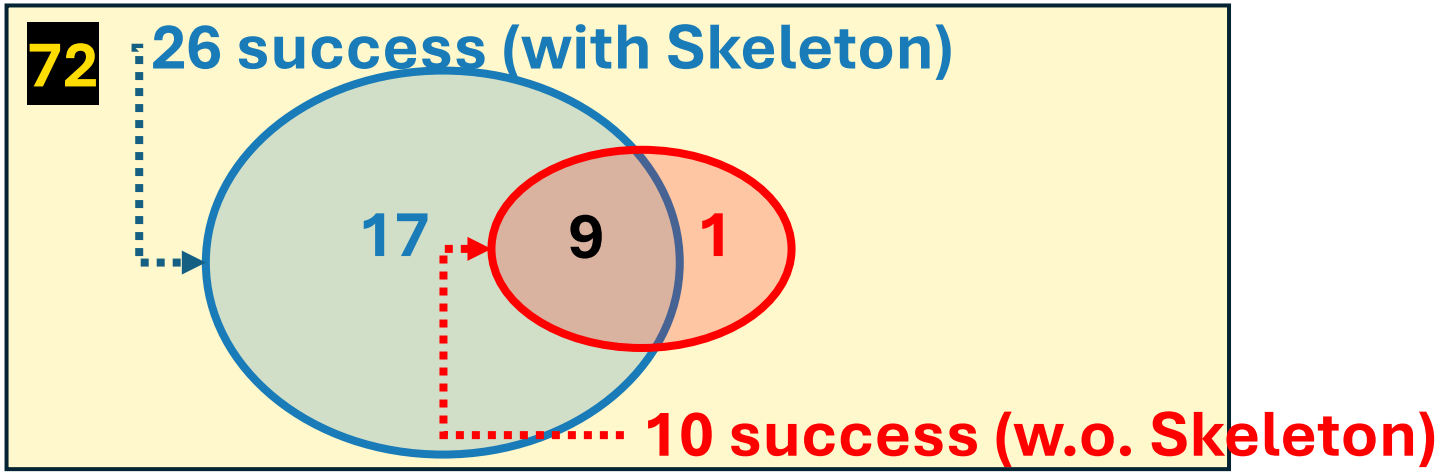
Use this lemma

```
theorem decomposition_count_ge_max
  (pre : Workload) (init_db : DB) :
  count (pre init_db) ≥ max ((groupBy key (pre
init_db)).map count) :=
  skeleton_aggregation_decomposition count max (· ≥ ·)
count_ge_max_decompose_law pre init_db
```

RQ1: Proof Skeletons Increase Success Rates

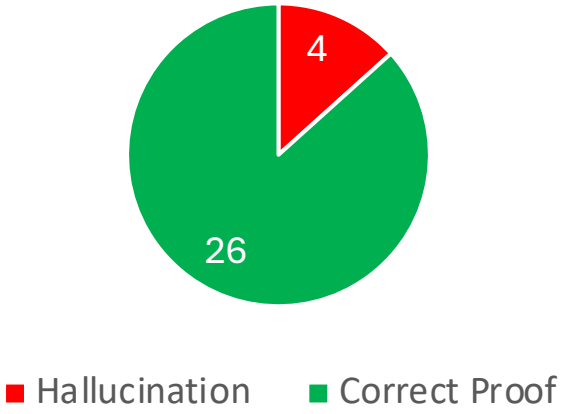
72 (provable by human)

=100 (hypothesized by LLM + enumerated based on pySpark APIs) – 28 (infeasible)

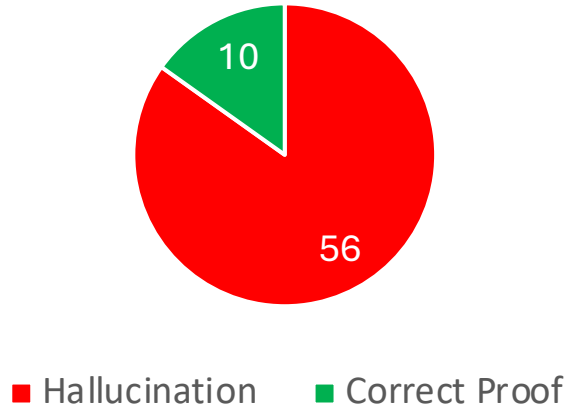


RQ2: Proof Skeletons Reduce Proof Hallucinations

Proof with Skeletons



Proof without Skeletons



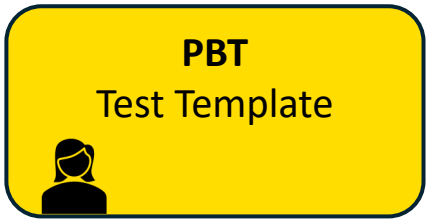
Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem—how could we generate properties to use for PBT?
4. Harvest **recurring property patterns** and codify them as skeletons.
   - H1: Can proof-skeletons ease proof-engineering?
   - H2: Can PBT-skeletons amplify PBT effectively?

The Testing World:
Bootstrap and Amplify Property-based Tests

PBT: Testing World

LOC	**13347**
Test Skeleton Classes	**5**
Abstract Test Methods	**32**
Well-formed Generator Categories	**7 (`Schema`, `Data`, `DAG`, `Op`, `Expr`, etc.)**



Agentic Test Plan Synthesis

Properties: Model World

"for all data and compute workloads, a global count is the sum of local counts"

PBT Agent

Select PySpark APIs needed for **function**, **recombine,** and **rel**

PBT: Testing World

```
{
"family": "aggregation_decomposition",
"description": "COUNT can be decomposed into local counts per group, then summed globally to exactly recover the global count of non-null values.",
 ...
"agg_function": "count",
"recombine_function": "sum",
"expected_relation": "equal",
 ...,
}
```

PBT Agent

Agentic PBT Skeleton Concretization

Abstract PBT Skeleton for aggregation_decomposition

```
class AggProperty(ABC):
  key_types = ['int', 'bool', 'string'] # SLOT 1
  val_types:   List[str] = ['int', 'float', 'double']  # SLOT 2
...
  @abstractmethod
  def emit_global(...) -> str:
  """Single expression: global aggregate over the whole table.
  def emit_local_recombine(...) -> str:
  """MUST return a SINGLE Python expression (no newlines, no statements) """
    def emit_check(...) -> List[str]:
      """OPTIONAL OVERRIDE""
```

Test Plan in JSON

PBT Agent

Override **abstract methods** and fill in **slots**

Concrete Test Subclass for "replace_count_with_groupby_count"

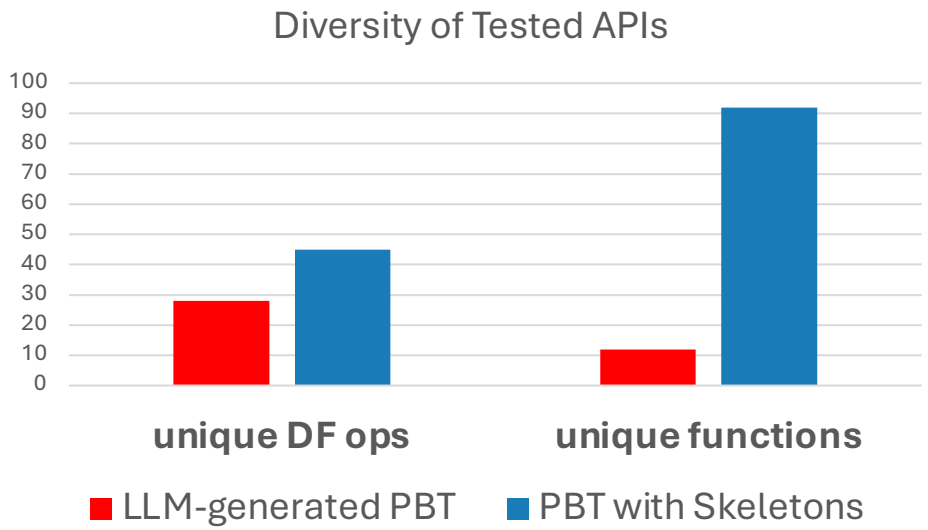
```
class CountProperty(AggProperty):
  key_types = ['int', 'bool', 'string'] # FILL IN SLOT 1
  val_types = ['int', 'float', 'double'] # FILL IN SLOT 2

  def emit_global(...):
    return f".agg(F.count())).collect()._r"

  def emit_local_recombine(...l):
    return (f".groupBy([])"
        f".agg(F.count().alias('_m'))"...

  def emit_check(...):
    return _check_approx_equal(q_global, q_local)
```

RQ1: PBT Skeletons Increase Test Diversity

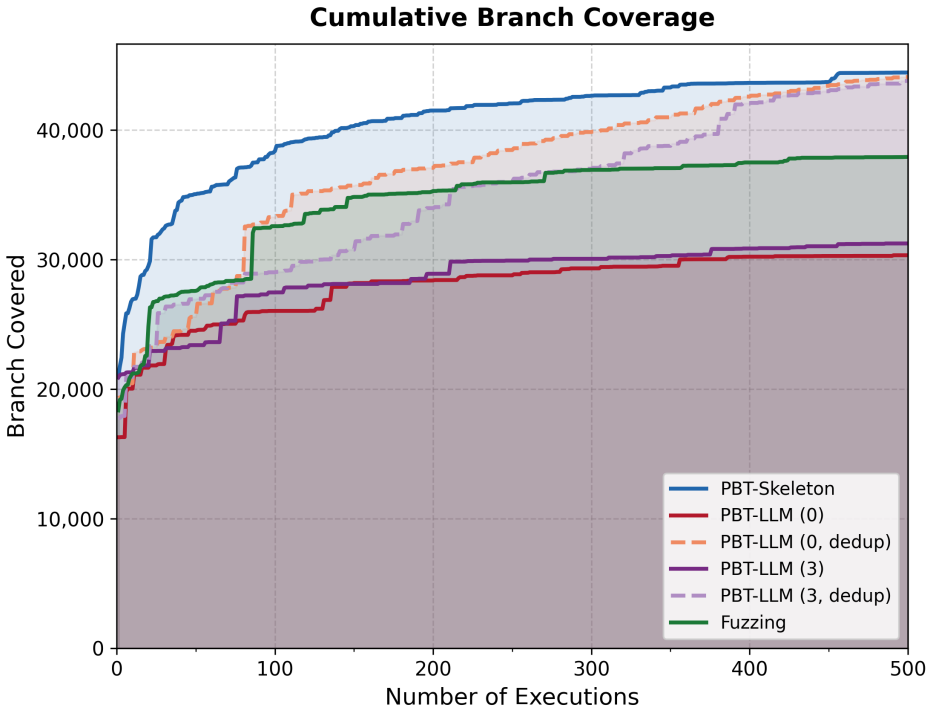


UCLA Samueli Computer Science

LLM-generated PBT is an approach similar to Kiro PBT https://kiro.dev/docs/specs/correctness/

RQ2: PBT Skeletons Match Fuzzing's Coverage

Branch coverage

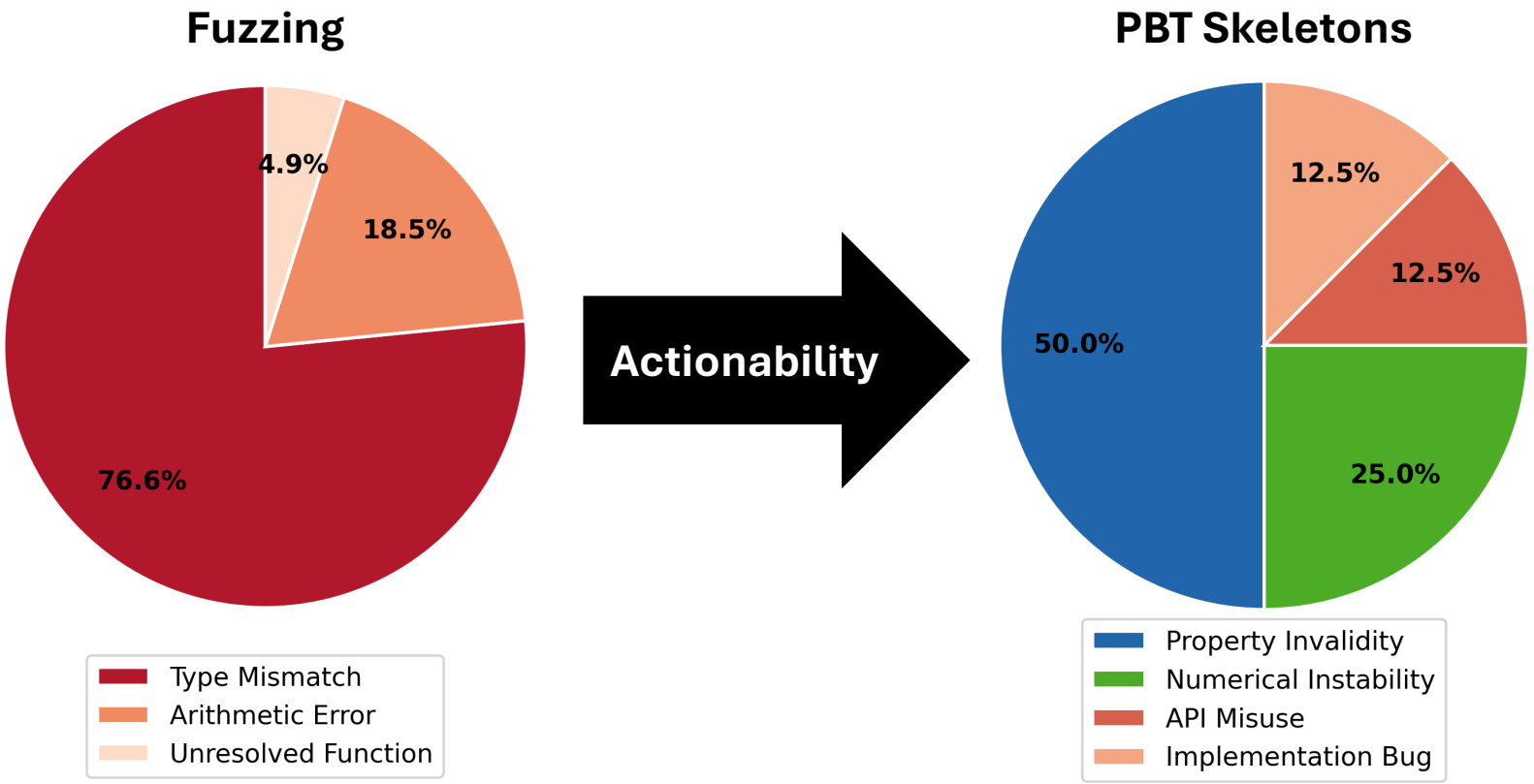


PBT-Skeleton
Fuzzing
PBT-LLM (0)
PBT-LLM (3)

Coverage: PBT-Skeleton > Fuzzing

UCLA Samueli Computer Science

RQ3: PBT Skeletons Are Effective in Detecting Semantic Drifts



Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem—how could we generate properties to use for PBT?
4. Harvest **recurring property patterns** and codify them as skeletons.
   - H1: Proof-skeletons ease proof-engineering
   - H2: PBT-skeletons amplify PBT effectively

UCLA Samueli Computer Science

Tensor Algebraic Property-Skeletons: Amplifying Property-Based Testing for AI Compiler tvm


Yuxin Qiu


Ben Limpanukorn


Seongmin Lee


Jiyuan Wang

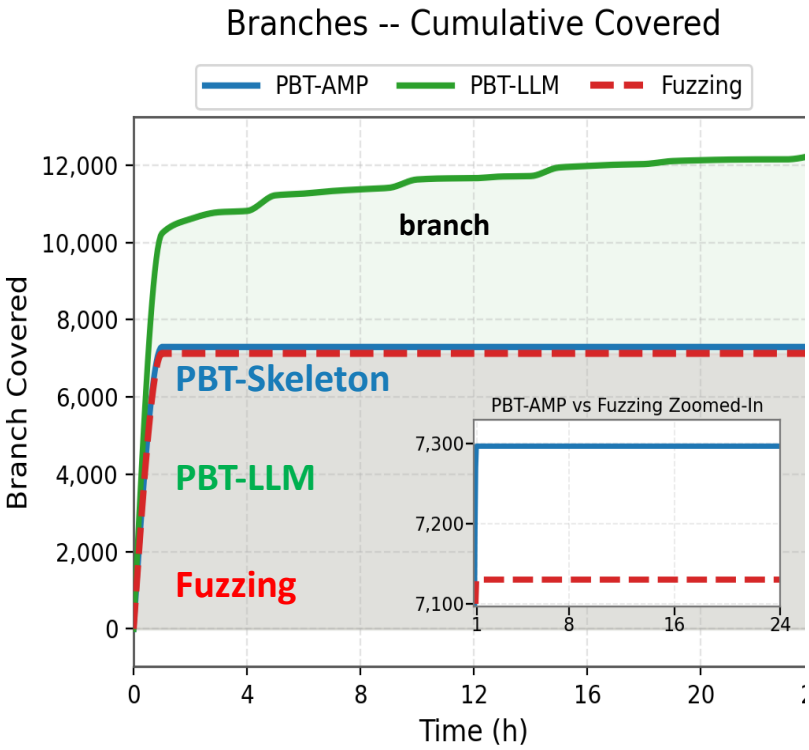

Miryung Kim


Qian Zhang

UCLA Samueli Computer Science

Yuxin Qiu et al. https://arxiv.org/abs/2606.06747 June 2026

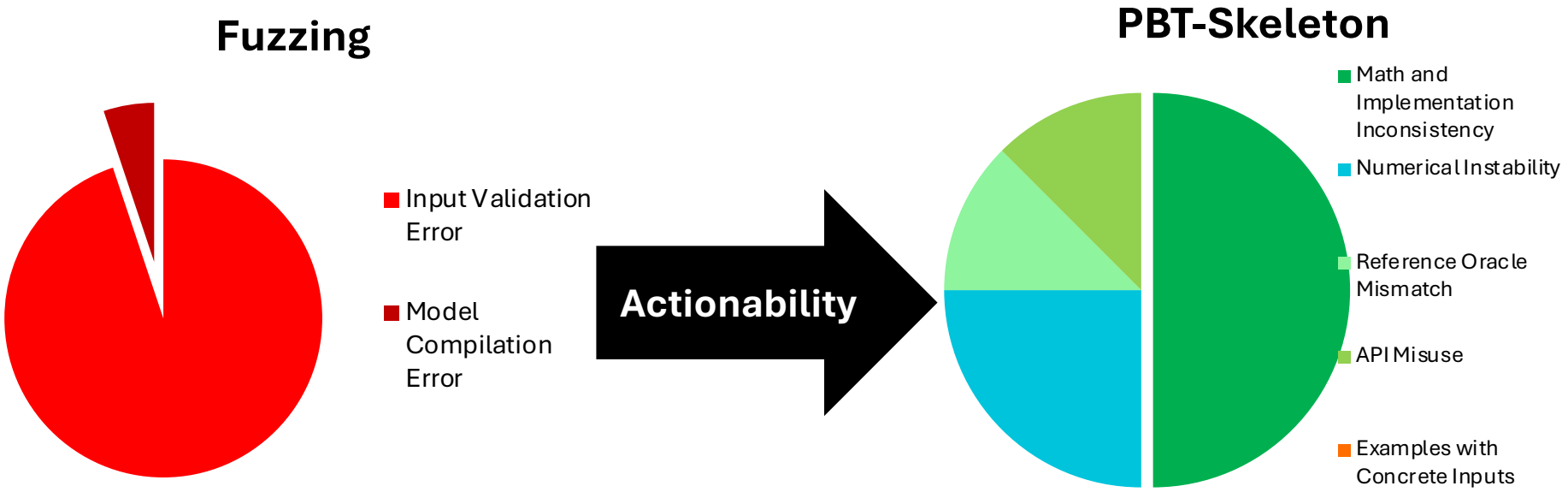
RQ1: PBT Skeletons Match Fuzzing's Coverage



Coverage: PBT-Skeleton ≃ Fuzzing

Coverage of PBT-LLM is misleading (mostly from error handling)

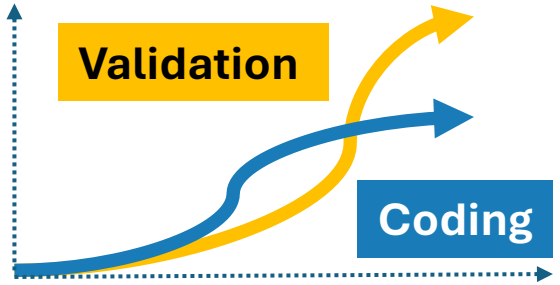
RQ2: PBT Skeletons Are Effective in Detecting Semantic Drifts



Input Validation Errors: PBT-Skeleton ≪ Fuzzing

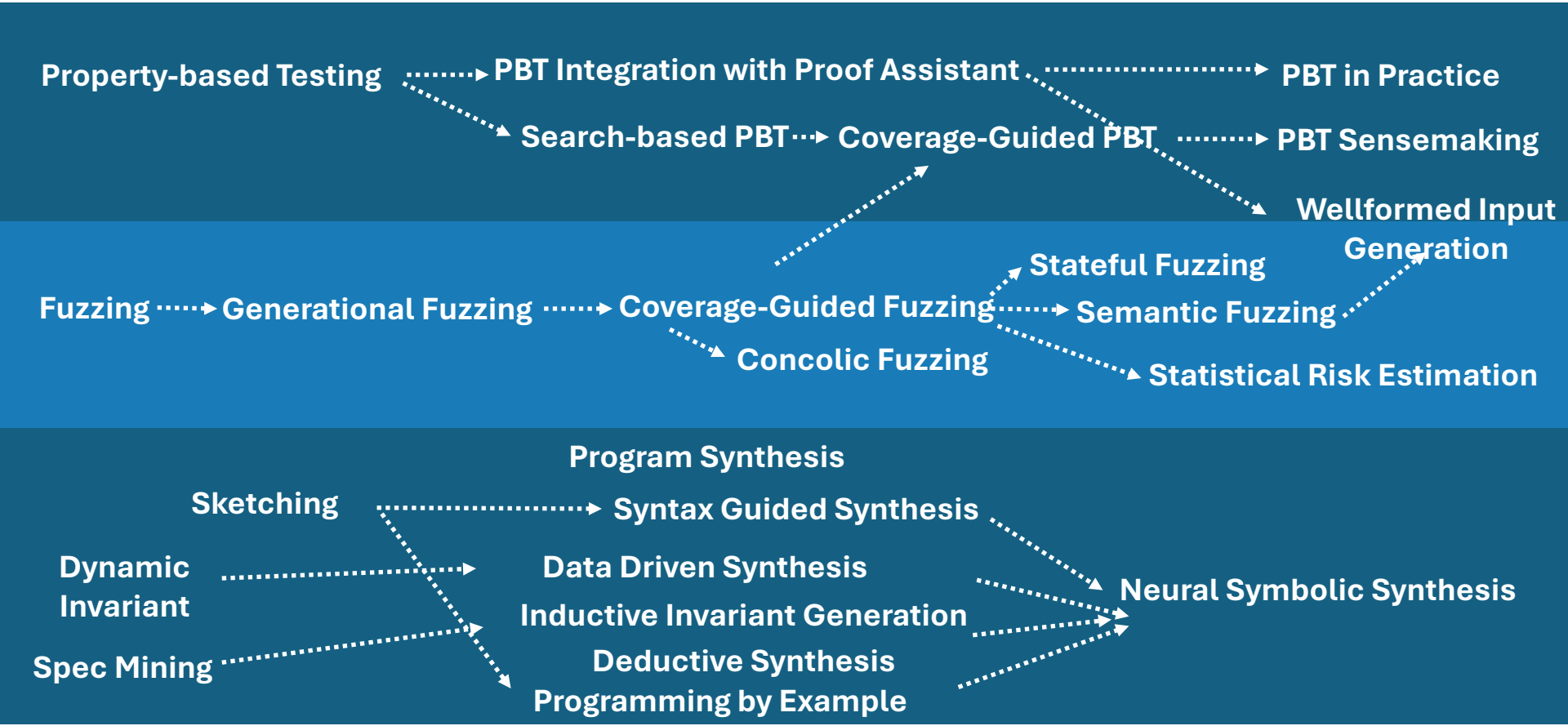
Outline

1. Validation is the **new bottleneck in AI**
2. Shift from **fuzzing** to **property-based testing**
3. Solve the **specification** problem—how could we generate properties to use for PBT?
4. Harvest **recurring property patterns** and codify them as skeletons.
5. What Next?



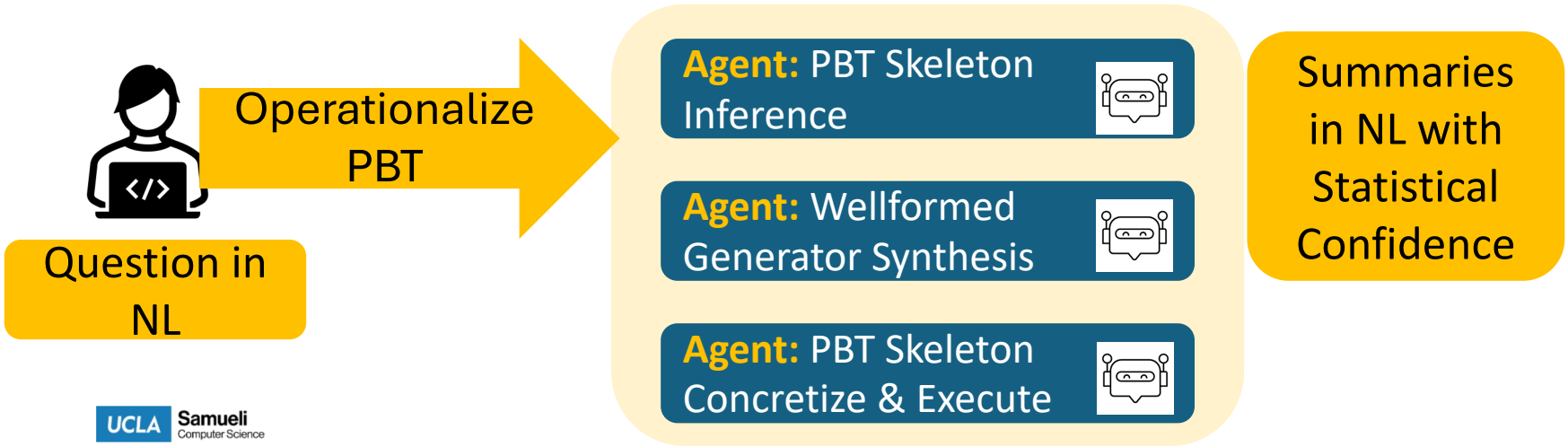
What Next?

Validation Must Outpace Creation

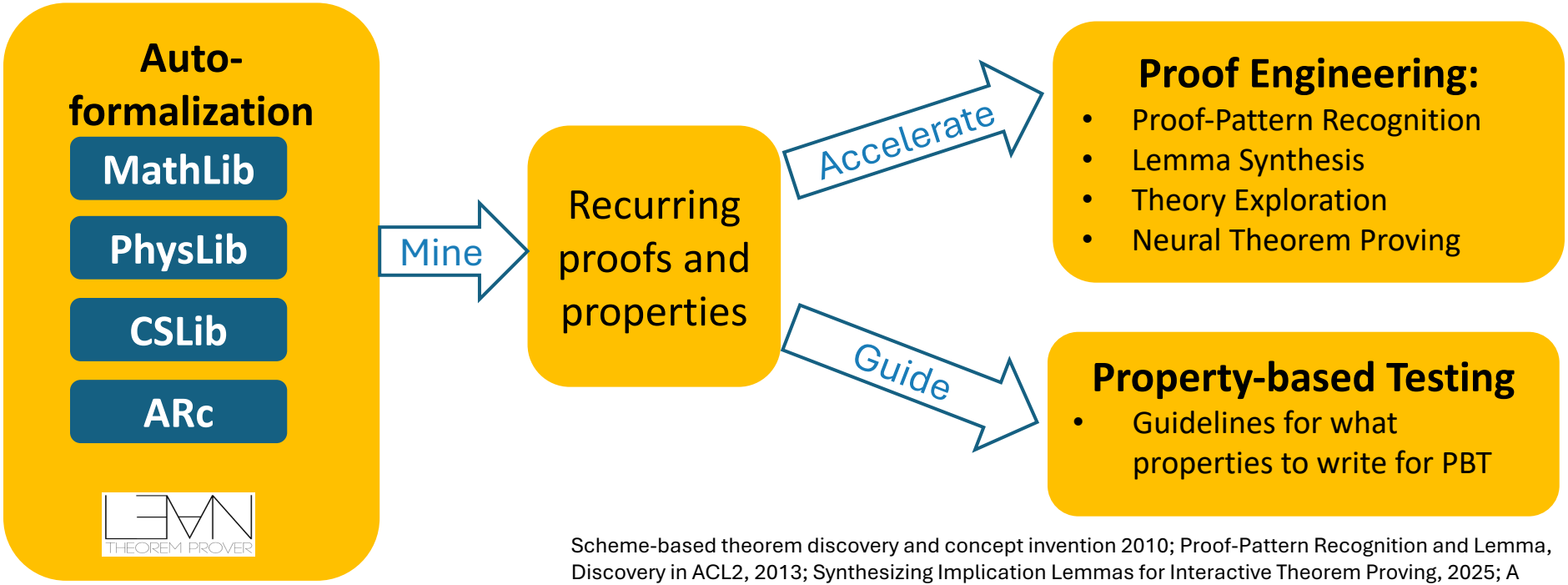

Property-based Testing
PBT Integration with Proof Assistant
PBT in Practice
Search-based PBT
Coverage-Guided PBT
PBT Sensemaking
Wellformed Input Generation
Fuzzing
Generational Fuzzing
Coverage-Guided Fuzzing
Stateful Fuzzing
Semantic Fuzzing
Concolic Fuzzing
Statistical Risk Estimation
Program Synthesis
Sketching
Syntax Guided Synthesis
Dynamic Invariant
Data Driven Synthesis
Neural Symbolic Synthesis
Inductive Invariant Generation
Spec Mining
Deductive Synthesis
Programming by Example

Vision 1: Socratic End-to-End Validation

- Why review 10 KLOC of AI-generated unit tests, if you can ask questions in NL and audit concise PBT summaries with statistical confidence?



Vision 2: Harvest and Codify Recurring Property Patterns



Scheme-based theorem discovery and concept invention 2010; Proof-Pattern Recognition and Lemma, Discovery in ACL2, 2013; Synthesizing Implication Lemmas for Interactive Theorem Proving, 2025; A Neurosymbolic Approach to Natural Language Formalization and Verification, 2025; Lemmanaid: Neuro-Symbolic Lemma Conjecturing, 2026; SITA: A Framework for Structure-to-Instance Theorem Autoformalization, 2026;
How to Specify It!: A Guide to Writing Properties of Pure Functions, 2020

Vision 3: Living Specs, Benchmarks for Migration

- **Prediction:** More Migration, Rewriting, Innovating!

Legacy Code

Language Migration

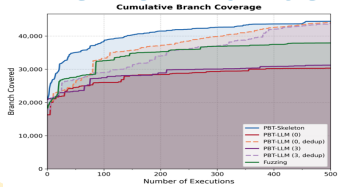
Platform Migration

Must Satisfy **Correctness** and **Performance**

Controlled Property-based Testing

Well-formed Input Generators with Distribution, Size, Complexity Controls

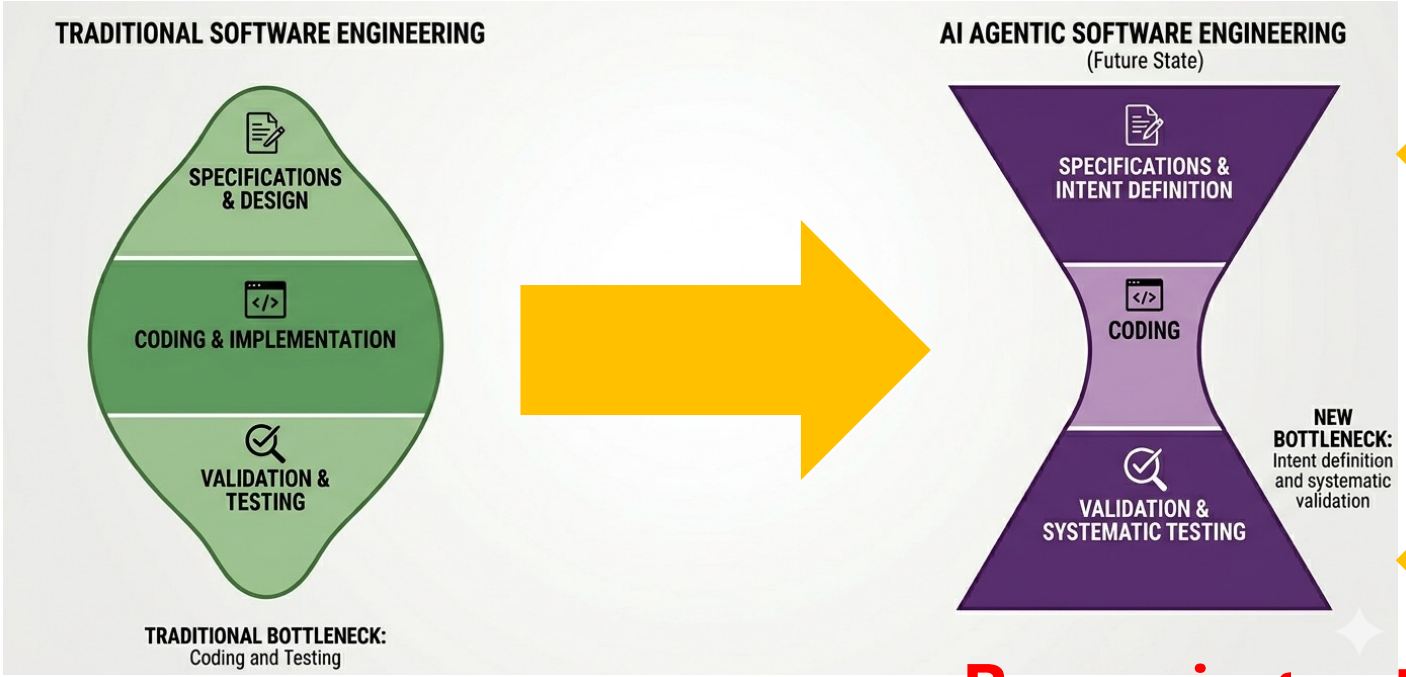
Performance Measurements



Latency

Throughput

Conclusion: Happiness U-Curve: Dual Track Validation with Recurring Validation Patterns



Recurring patterns in code

Recurring patterns in validation

UCLA Samueli Computer Science

Thank you!

AR @ AWS is hiring: ar-science@amazon.com