SE4AI + AI4SE w/ DNNs and LLMs

CS453 Automated Software Testing

Shin Yoo | COINSE@KAIST

Software Engineering & Artificial Intelligence

- AI4SE: using AI to assist and automate SE tasks
  - Search-Based Software Testing (metaheuristic optimization for test generation), Defect Prediction (machine learning for predicting whether a commit may be faulty or not), LLM-based code generation (generative language model for programming and/or testing)…
- SE4AI: using SE techniques to improve and utilize AI
  - Testing robustness of DNN models/LLM-based agents…

SE4AI: what can SE do for AI? :)

AI/ML as Computation Method

- We often talk about AI/ML as the next generation computing paradigm, but what do we exactly mean?
- Traditionally, software means something written by humans - we may not have done it very well, but we did it our way nonetheless.
- ML as a software component - what are the implications for testing?

Traditional Code	DL System
Specification	Training Data
Logic as Control Flow	Logic as Data Flow
Written	Trained
Tested	Tested
For Faults	Faults?
Patched	Retrained?

Outputs are not exactly discrete 😨

```
if (a + b <= c) {
  return TriangleType.INVALID;
} else if (a == b && b == c) {
  return TriangleType.EQUALATERAL;
} else if (a == b || b == c) {
  return TriangleType.ISOCELES;
} else {
  return TriangleType.SCALENE;
}
```

vs.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1,\ldots,K \text{ and } \mathbf{z} = \left(z_1, \ldots, z_K\right) \in \mathbb{R}^K$$

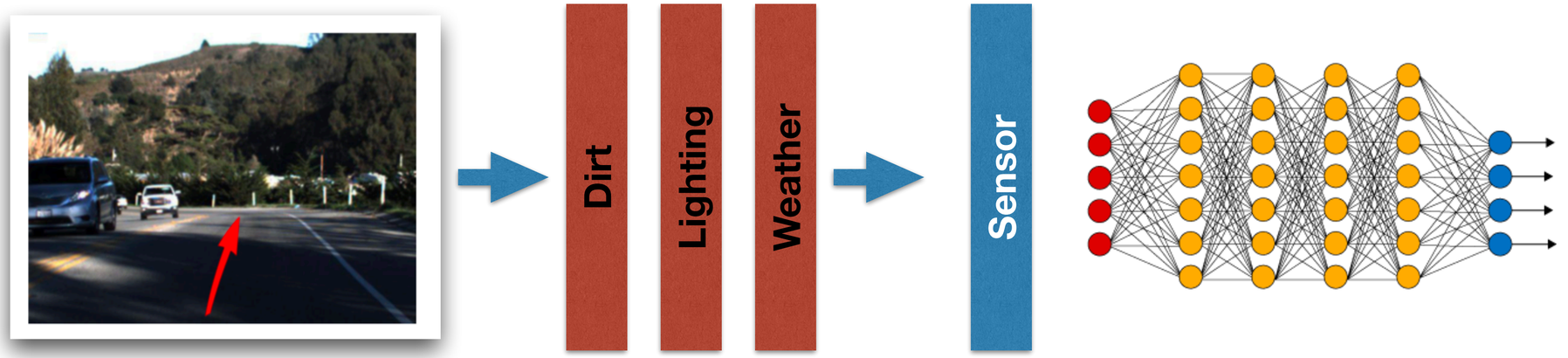
Inputs are more complicated, perhaps even stochastic 😨

```
int x = 42;
```

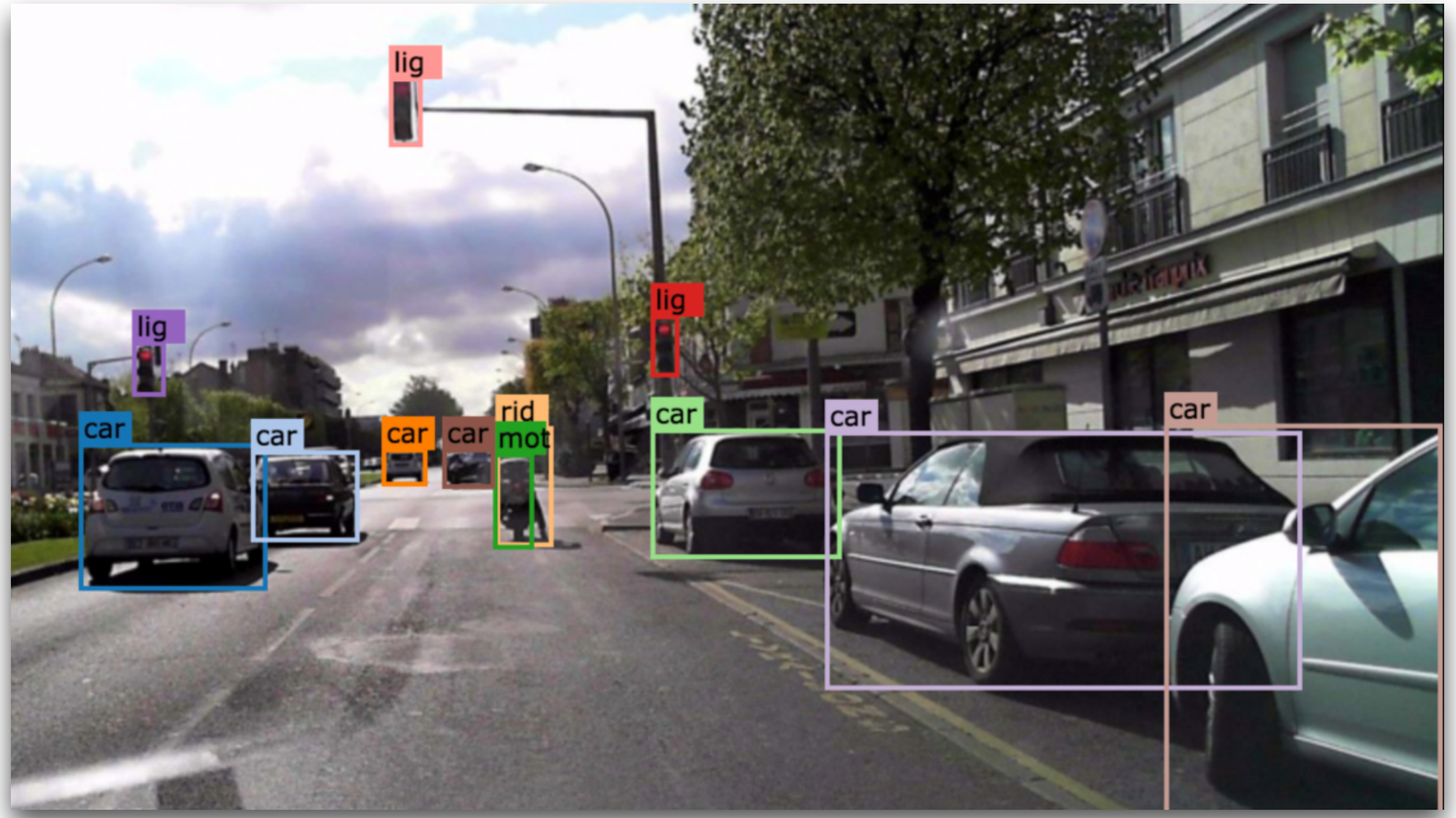
a du path

```
if(x == 42){…
```

VS.

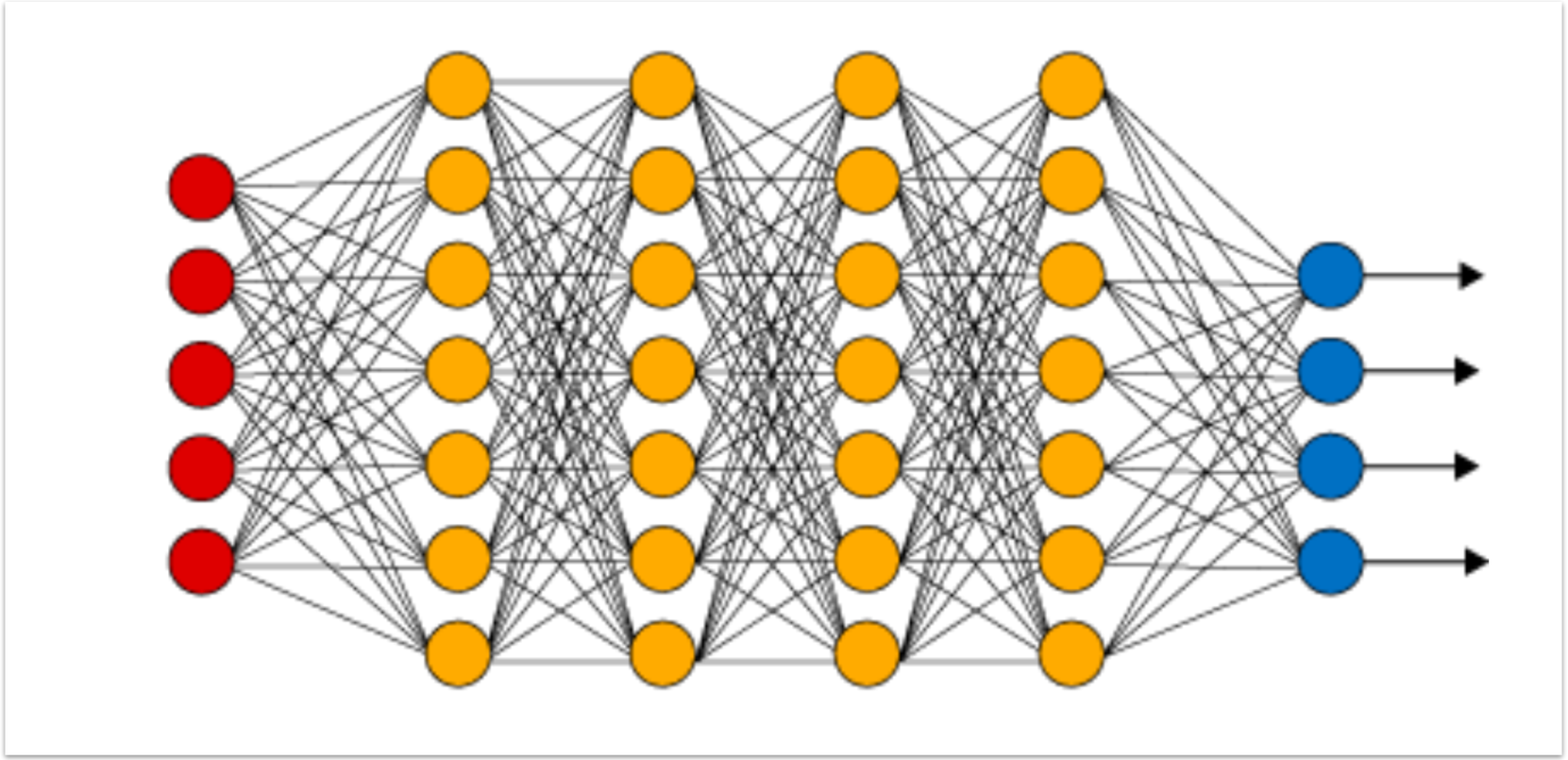


Can we (randomly) sample these inputs? 😨



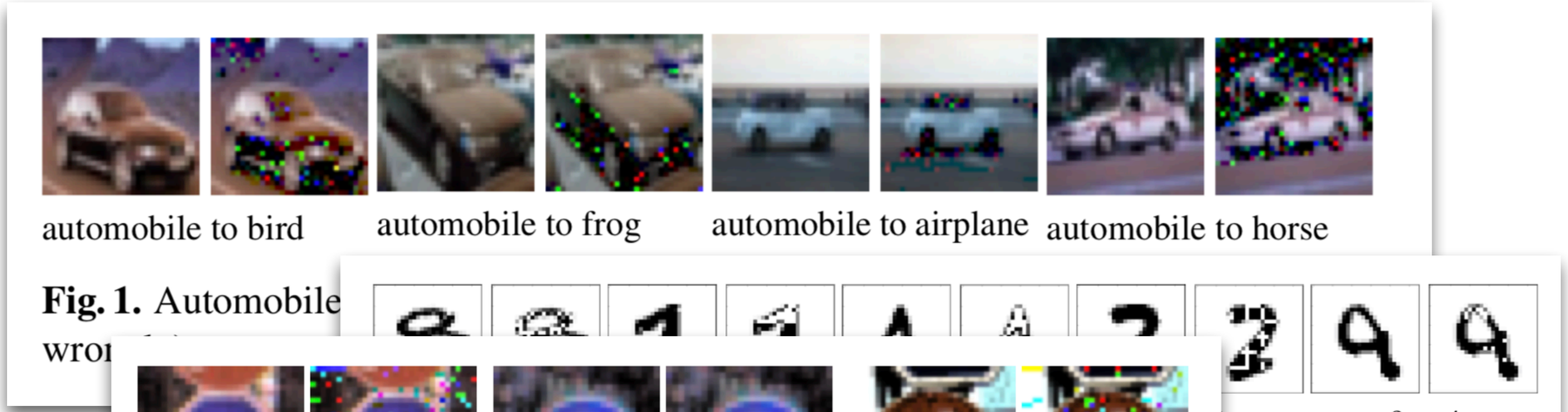
https://www.technologyreview.com/the-download/611380/researchers-have-released-the-largest-self-driving-car-data-set-yet/

Very little to cover, at least structurally 😨



Training seems less competent than programmers 😨😨

(behaviour are not discrete)



Safety verification of deep neural networks,
iaowei Huang, Marta Kwiatkowska, Sen Wang, Min Wu (https://arxiv.org/abs/1610.06940)

Training seems less competent than programmers 😨😨😨

(behaviour are not discrete)

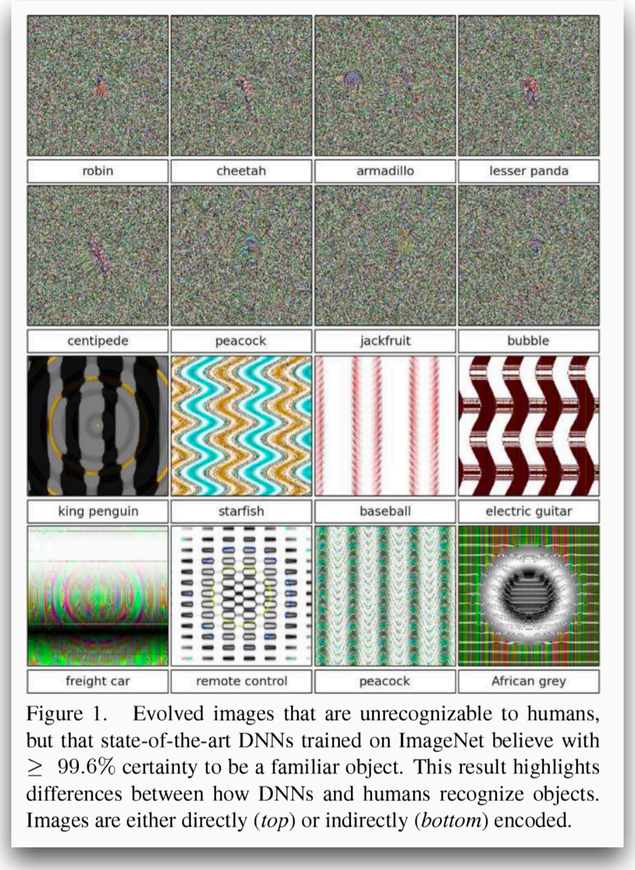


Figure 1. Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with $\geq$ 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. Images are either directly (*top*) or indirectly (*bottom*) encoded.

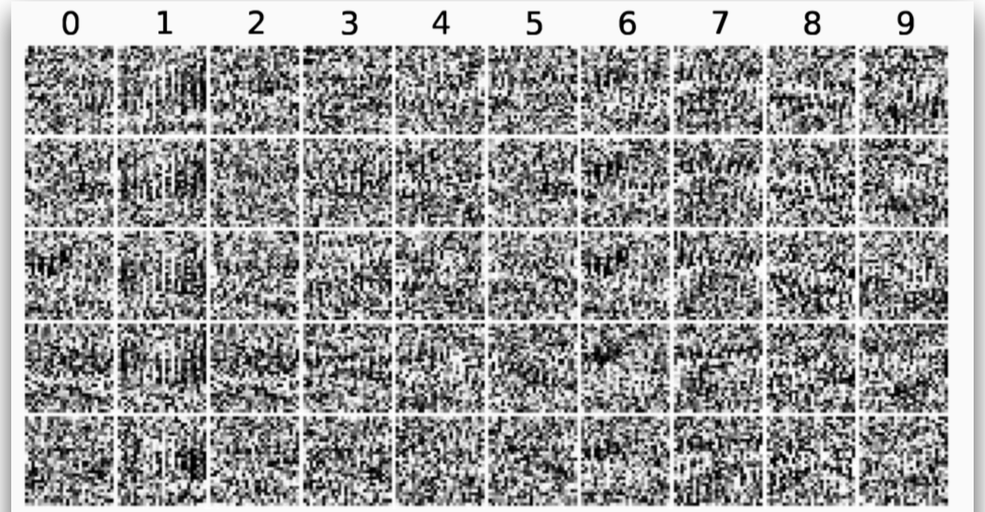


Figure 4. Directly encoded, thus irregular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class, and each row is the result after 200 generations of a randomly selected, independent run of evolution.

A. M. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence pre- dictions for unrecognizable images. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 427–436, 2015.

Going Metamorphic

Metamorphic testing is a surprisingly effective conceptual tool
for testing DNNs (at least so far).

Given that DNN() produces the output “car”,

MT suggests that DNN() should also produce the output “car”.

Input MR: images are perceptively identical to human eyes.
Output MR: class labels should be identical.

Existing work on DNN Testing

- Coverage Criteria/Test Adequacy: given sets of inputs, or pairs of inputs, tries to quantify how effective they are for testing, i.e., is set/input A more likely to reveal unexpected behaviour than set/input B?
- Test Input Generation/fuzzing: typically tries to transform valid inputs into adversarial examples (note: sampling from the scratch is hard)
- Mutation Testing: can we do better testing via mutation? But what does it mean to mutate an DNN model?

Adequacy Criteria

Adequacy Criteria

- With traditional software, the code embodies the logic of the program. Which is why structural coverage can work as an adequacy for testing: more code being executed is correlated with more diverse logical behaviours being explored.
- DNN code DOES NOT embody the logic. In fact, hardly anything to explore.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

https://medium.com/tensorflow/standardizing-on-keras-guidance-on-high-level-apis-in-tensorflow-2-0-bad2b04c819a

Test Adequacy for DNNs

- Given two sets of inputs, how do we know which one is better for testing DNNs?
- The more diverse one.

Structural Adequacy Criteria

DeepXplore (SOSP 2017), DeepGuage (ASE 2018)

Neuron Coverage

$$\text{NCov}(T, \boldsymbol{x}) = \frac{|\{n \,|\, \forall x \in T, \textbf{out}(n, \boldsymbol{x}) > t\}|}{|N|}$$

Intuition: inputs that activate more nodes above a given threshold are using wider and different parts of the network, and therefore making use of a wider range of learnt features.

K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. SOSP 2017.

k Multi-section Neuron Coverage (kMNC)

$$\text{KMNCov}(T, k) = \frac{\sum_{n \in N} \left| \left\{ S_i^n \,|\, \exists \mathbf{x} \in T : \phi(\mathbf{x}, n) \in S_i^n \right\} \right|}{k \times |N|}$$

Neuron Boundary Coverage (NBC)

$$\textbf{UpperCornerNeuron} = \left\{ n \in N \,|\, \exists \mathbf{x} \in T : \phi(\mathbf{x}, n) \in (\text{high}_n, +\infty) \right\}$$

$$\textbf{LowerCornerNeuron} = \left\{ n \in N \,|\, \exists \mathbf{x} \in T : \phi(\mathbf{x}, n) \in (-\infty, \text{low}_n) \right\}$$

$$\text{NBCov}(T) = \frac{|\ \textbf{UpperCornerNeuron}\ | + |\ \textbf{LowerCornerNeuron}\ |}{2 \times |N|}$$

Strong Neuron Activation Coverage (SNAC)

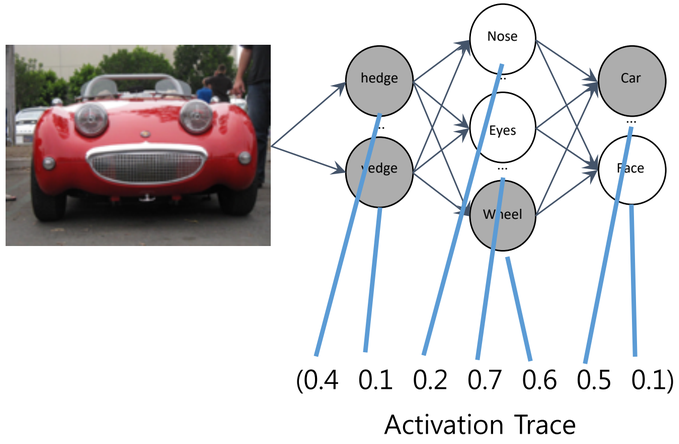
L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang. Deepgauge: Multi-granularity testing criteria for deep learning systems. ASE 2018.

$$\text{SNACov}(T) = \frac{|\ \textbf{UpperCornerNeuron}\ |}{|N|}$$

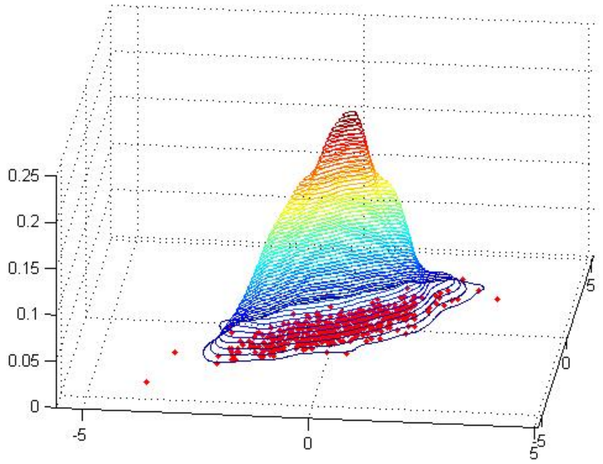
Distribution Aware Adequacy Criteria

Surprise Adequacy (ICSE 2019)

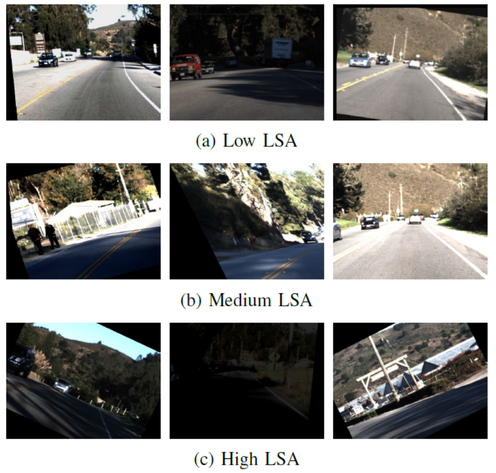
- Our argument: “A good exam question is one that is reasonably surprising to the (machine) learner: it should be sufficiently different from exercises in the textbook, but not so much as to be irrelevant to the course.”
- This notion depends on the existence of training data, as well as the similarity between the given input and the training data distribution.



Learnt Knowledge
(from training data)



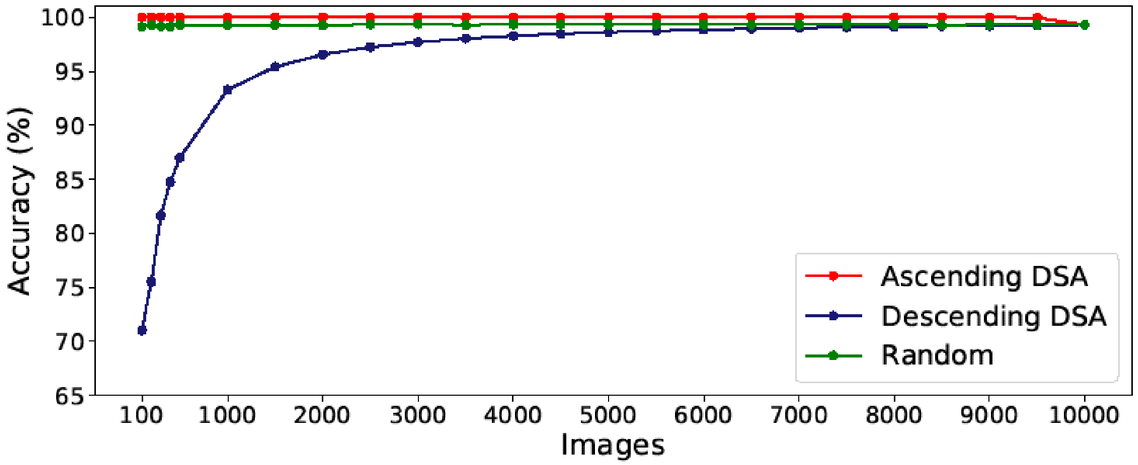
Summarisation
(KDE or point-cloud)


(a) Low LSA

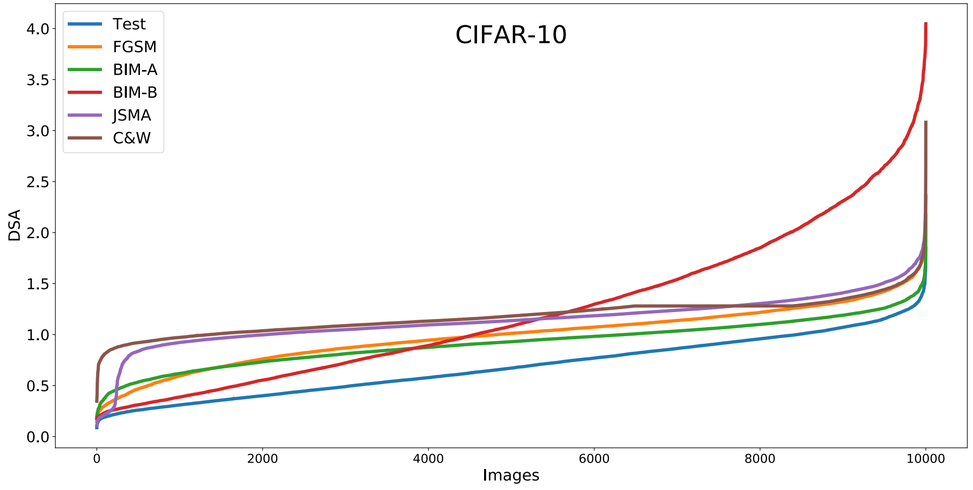
(b) Medium LSA

(c) High LSA

Quantitative Surprise Measure of New Input
Against the Summarisation



More surprising questions
are harder to answer correctly.



Trick questions (=adversarial
examples) are very surprising.

What are the actual benefits?

- In the ML context, the actual model correctness for an input is a given concept - you compare it to the label.
- In the SE context, the actual model correctness for an input is **EXPENSIVE**, as you have to manually label any new input!
    - SA has been successfully applied to semantic object segmentation (for autonomous driving) or NLP tasks such as question answering: both are tasks that require high labelling cost.

Input Generation / Fuzzing

DNN Fuzzing under Metamorphic Oracles

- Can we break the metamorphic relationship within ϵ-boundary? That is, if $M(i) = o$, does $M(i + \epsilon) = o$ also hold?
- Many domain specific variations on the theme exist, but most existing approaches are centered around this idea: oracle problem is difficult to beat.

The problem is not ONLY the oracle…

- In many cases, interesting applications of DNNs handle sensory input (image and audio) or highly unstructured input (natural language).
  - What is a random scene on a road? How do we sample it?
  - What is a random sentence? What is a neighbour of that sentence?
- Random sampling is not so easy now. Many benchmarks are manually generated (not only labels!).

Extreme Sparsity in Input Space

- MNIST dataset of handwritten digits contains 28 by 28 black and white pixels (784 pixels).
  - The number of all possible inputs: 2^{784}
  - The number of all recognisable digit images: ??? but probably much smaller than 2^{784}
- What will happen if we ignore the manifold of meaningful images?

Recall: DNNs are Easily Fooled

Use Genetic Algorithm to generate an imagethat is classified with high confidence

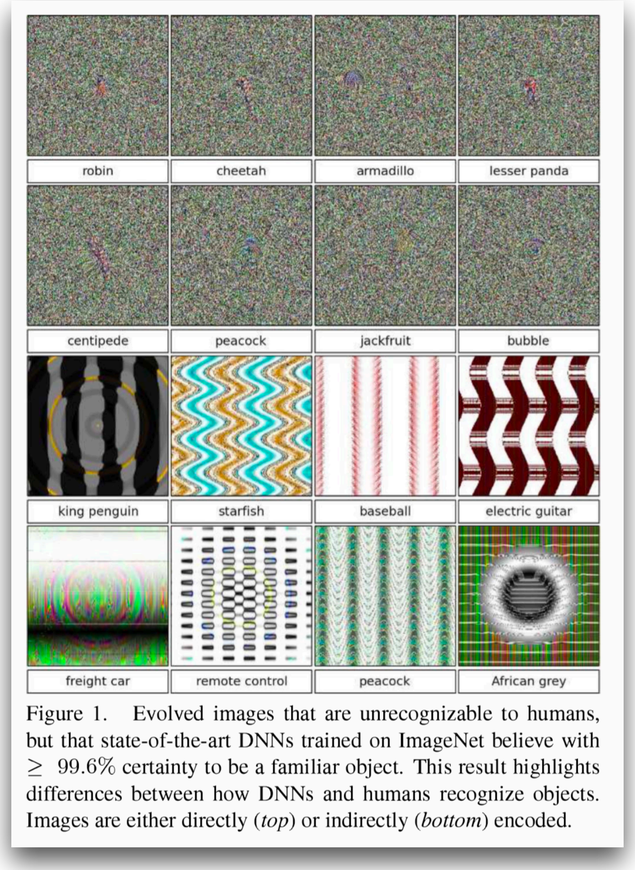


Figure 1. Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with $\geq$ 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects. Images are either directly (*top*) or indirectly (*bottom*) encoded.

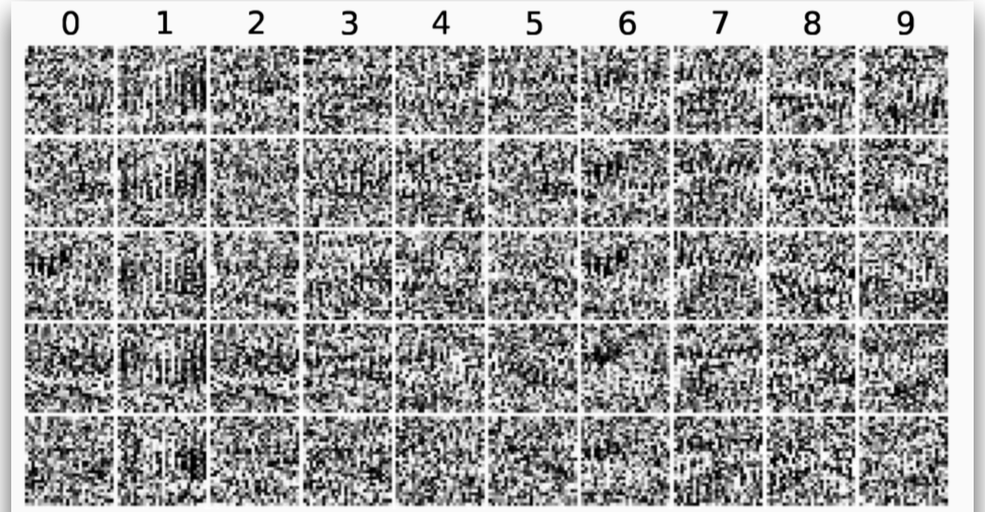
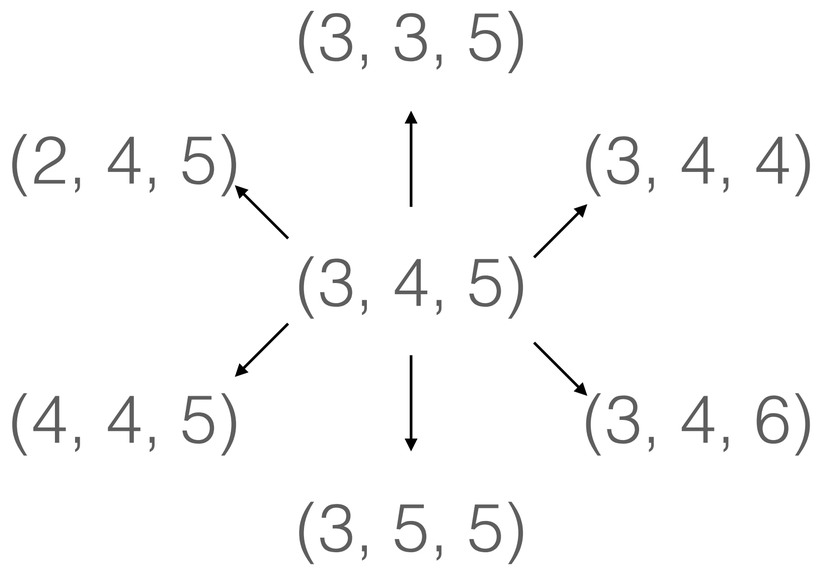


Figure 4. Directly encoded, thus irregular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class, and each row is the result after 200 generations of a randomly selected, independent run of evolution.

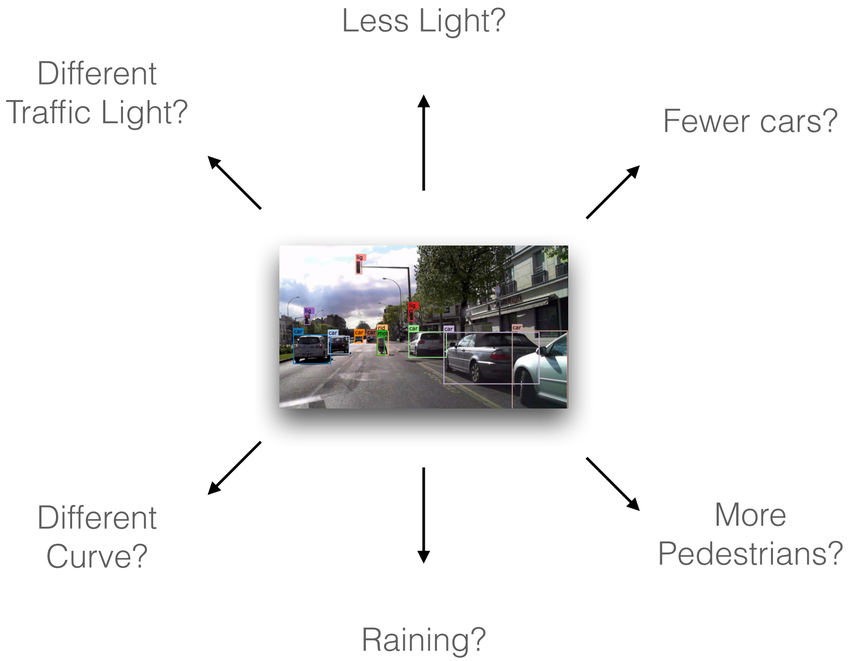
A. M. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence pre- dictions for unrecognizable images. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 427–436, 2015.

Seed Input + (Adversarial) Mutation

This way, we can tame the dimensionality a bit.



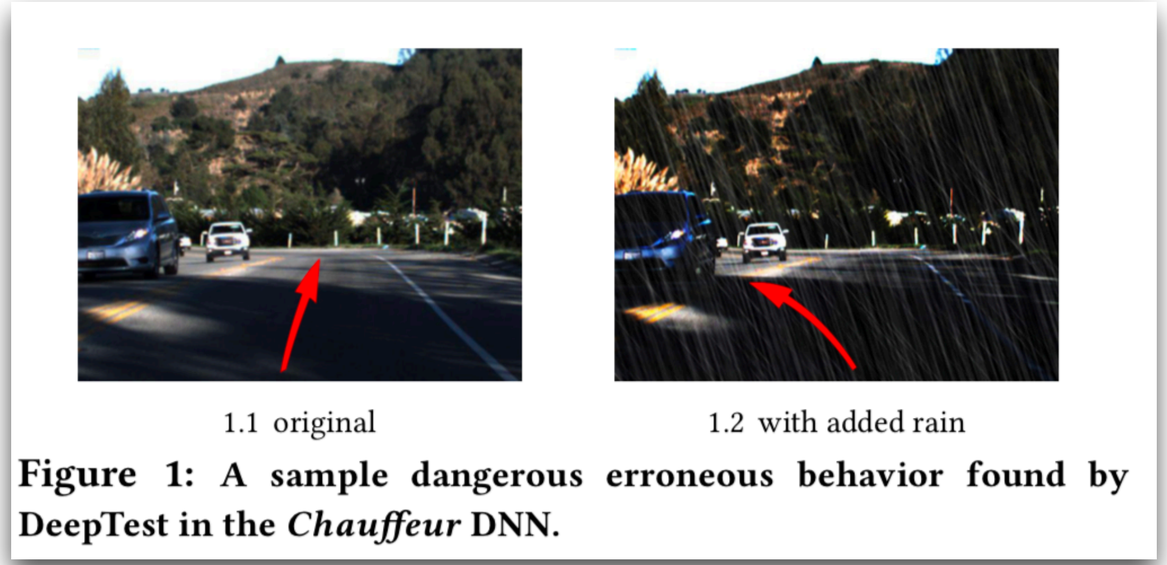
Primitive Type Neighbourhood



Perceptive Input Neighbourhood

Domain Specific Manipulation as Input Mutation

DeepTest (ICSE 2018) and DeepRoad (ASE 2018)



1.1 original 1.2 with added rain

Figure 1: A sample dangerous erroneous behavior found by DeepTest in the *Chauffeur* DNN.

A neighbour of a clear weather road scene is the same road in a rainy day.

In other words, DeepTest uses weather condition effects (photoshopped) or noises as a metamorphic relationship on inputs.

Y. Tian, K. Pei, S. Jana, and B. Ray. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. ICSE 2018

(a) Snow (b) Rain

Figure 2: Snowy and rainy scenes synthesized by DeepRoad

We really had this coming: applying GAN to mimic weather condition.

M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. DeepRoad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. ASE 2018.

Model-based Input Generation for DNNs

DeepJanus (FSE 2020)

Parameterised model that produce test inputs for DNNs
—> super efficient if your domain supports such a model!

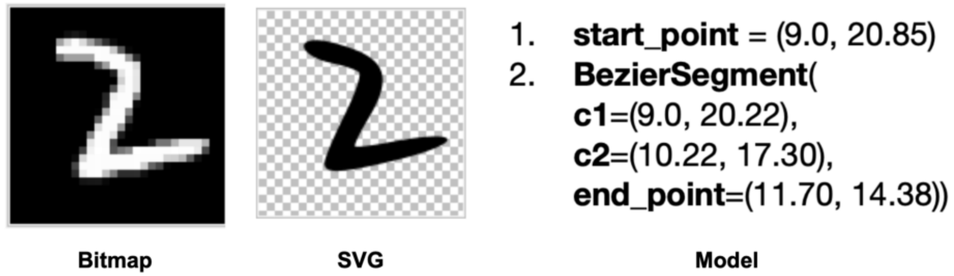


Figure 2: Bitmap and vector image; model representation of the image returned by Potrace

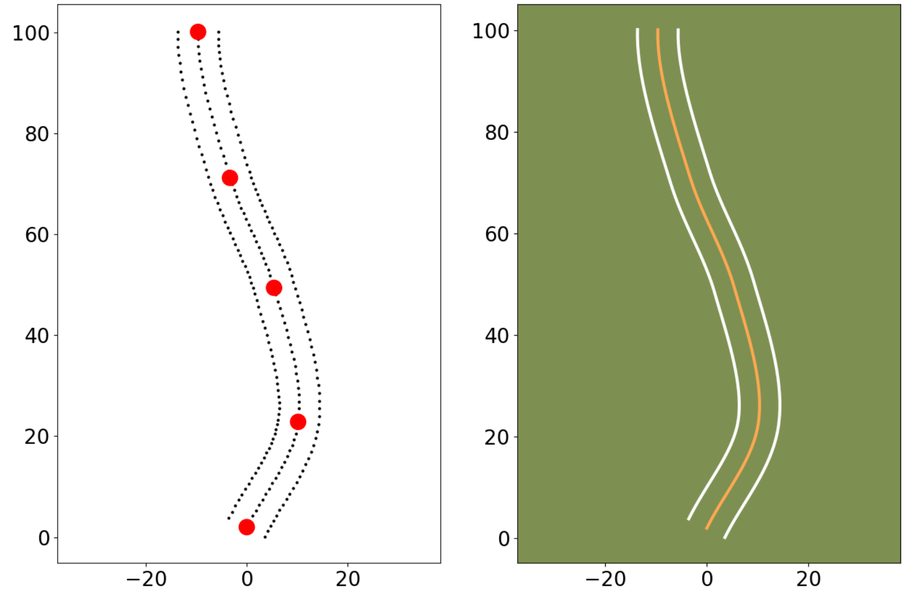
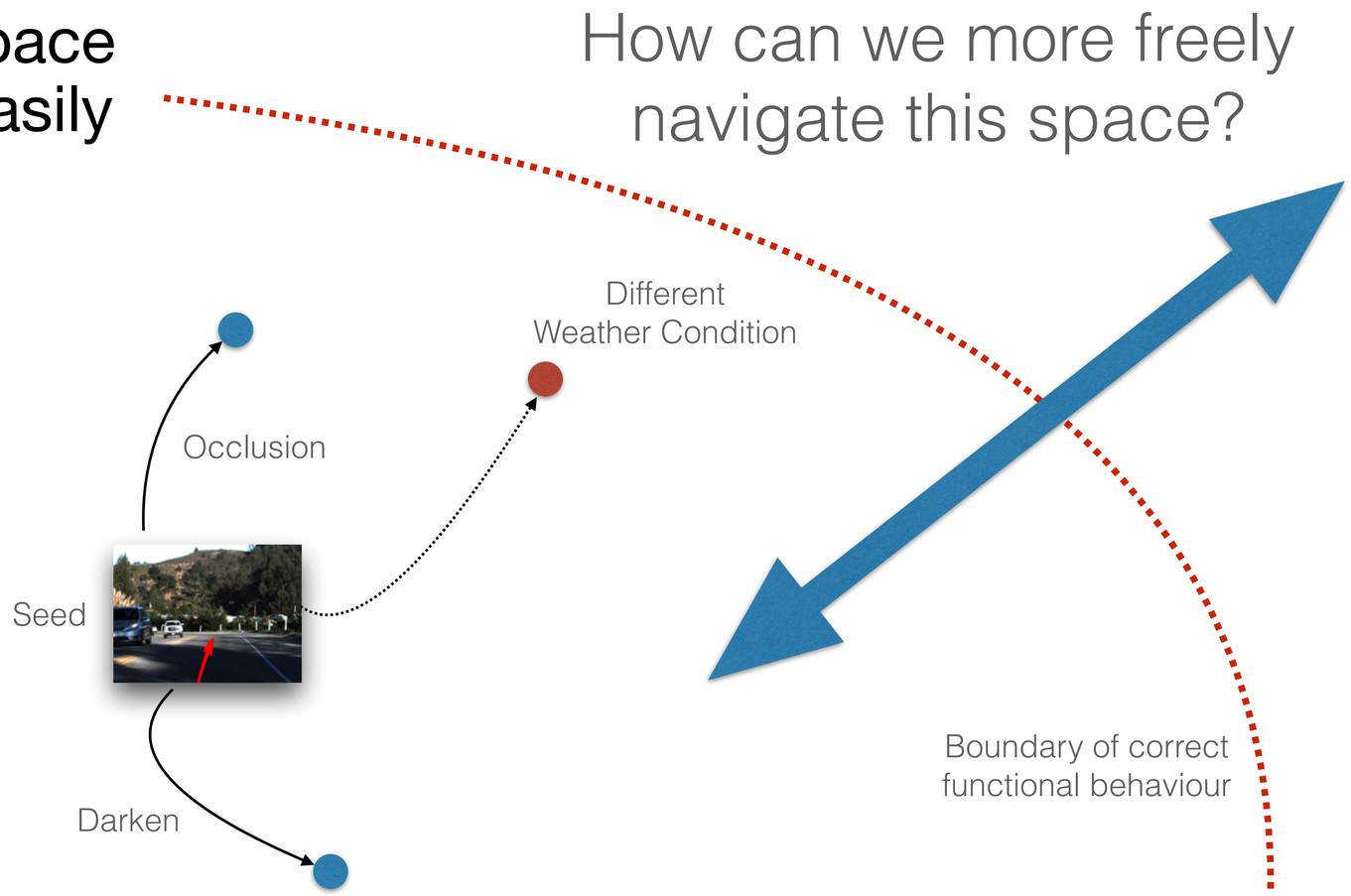


Figure 3: The model of a road and the corresponding road

V. Riccio and P. Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. FSE 2020.

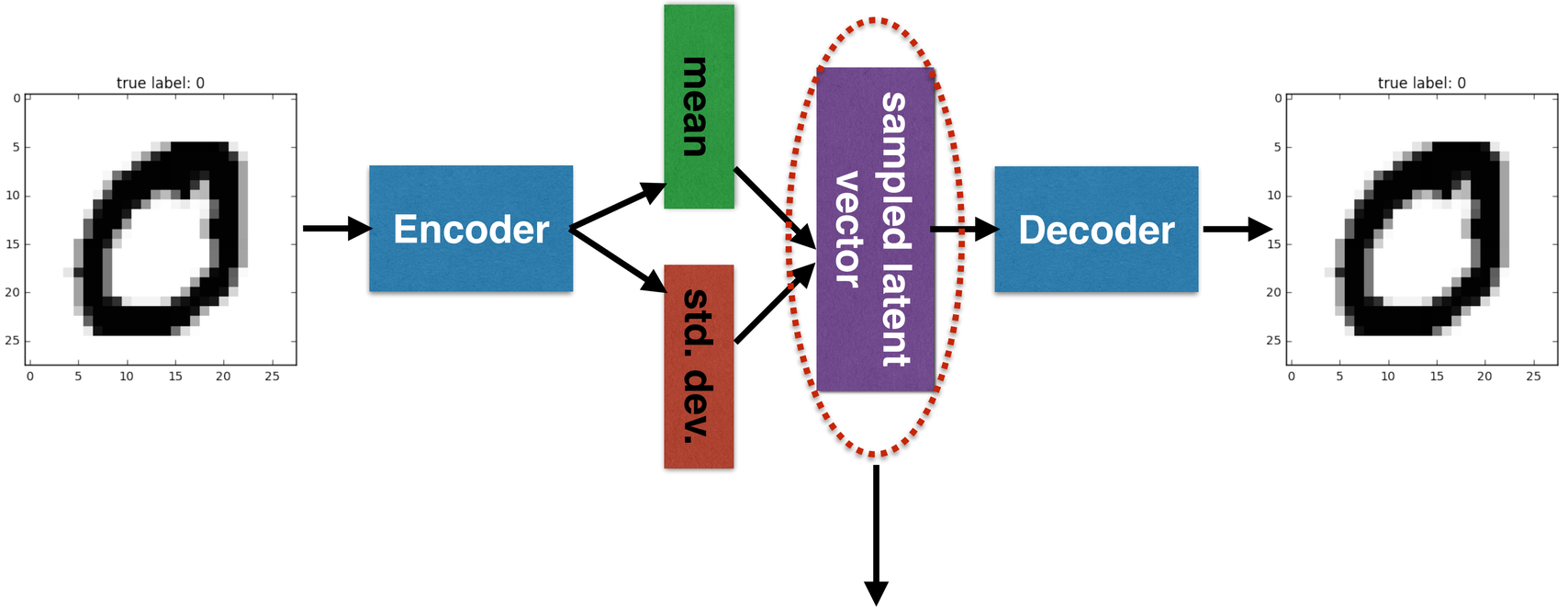
Can we go beyond MR based input search/generation?

- How do we map the semantic space of inputs to something we can easily navigate, and sample from?
- Hint: DNNs see everything as numbers!



Variational Auto-Encoders (VAEs)

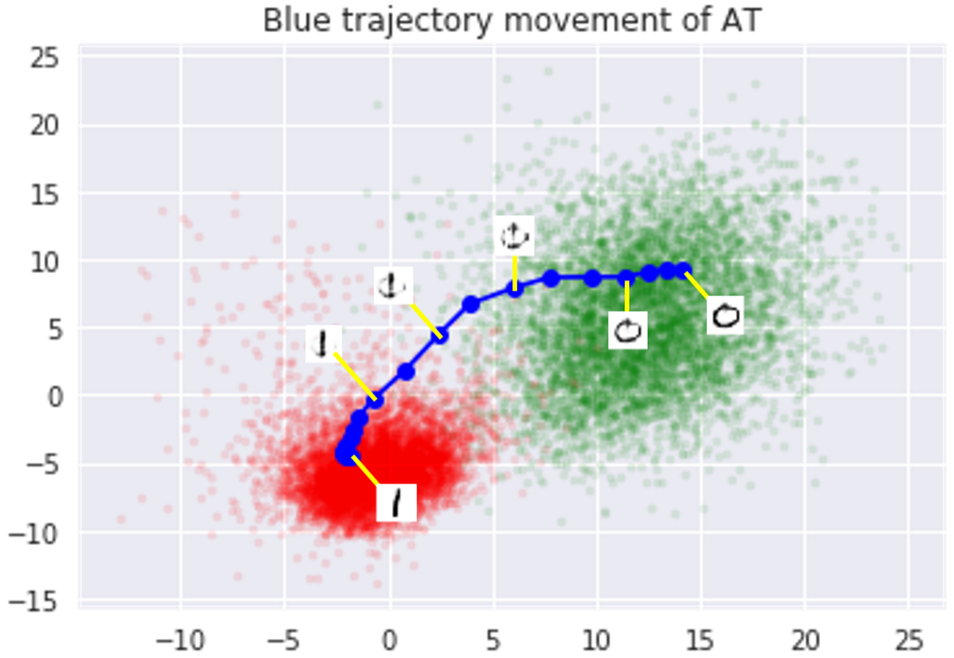
A type of generative model that maps inputs to a latent space



Our genotype (i.e., representation to manipulate)!

VAE + GA = Search Based Input Data Generation

- Representation: a latent vector that fits our VAE
- Fitness: Surprise Adequacy of the image decoded from a candidate solution (i.e., a latent vector)
- We visualise the search trajectory using Activation Trace (i.e., the output of a specific layer of the DNN) and PCA



Deceiving Humans and Machines Alike: Search-Based Test Input Generation for DNNs Using Variational Autoencoders, Kang, S., Feldt, R. and Yoo, S., *ACM Transactions on Software Engineering Methodologies*, 32(4):1–24.

Finding the nearest pairs to the border

Minimise the given fitness function…

$$f(i) = \begin{cases} \infty & N_c(i) = N_c(i_0) \\ |E(i) - E(i_0)| & else \end{cases}$$

… to generate really ambiguous yet realistic-looking images

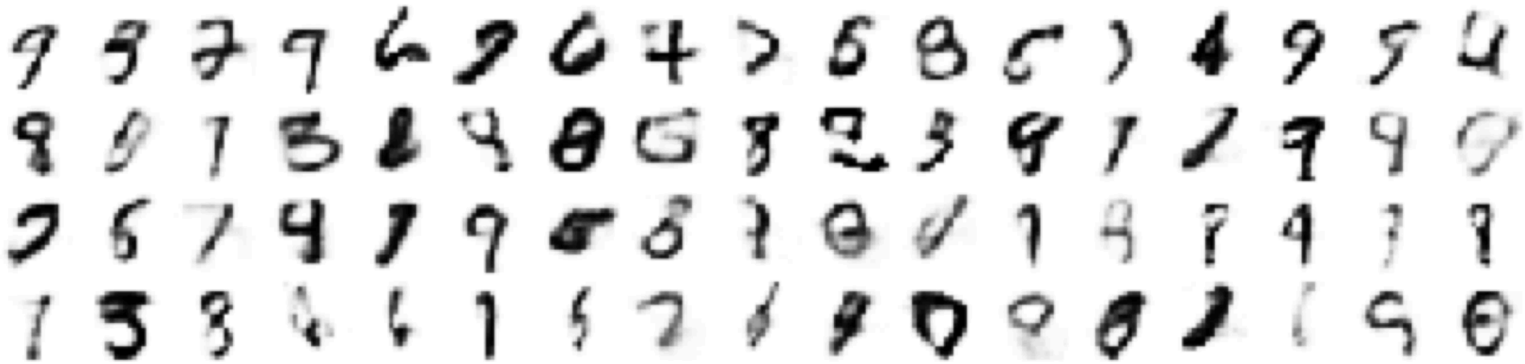


Fig. 6. Ambiguous images generated through GA optimization using Eq. 1 as the fitness function.

Cutting Edge: Diffusion-based Generative Models

Exploring space between two labels (Work in Progress@COINSE)

Pingpong to golf ball

Minimum distance
between two latent points

Following Bezier curve
between two patent

Mutation Testing of DNNs

How does CPH translate to DNN?

Competent Programmer Hypothesis

- Remember that DNNs are **trained, not (entirely) written**.
- Regarding the written part, there are social/human aspects: software engineers may not be ML experts, resulting in ***almost correct*** model architecture.
- Regarding the trained part, the specification (in the form of training data) may be ***almost complete*** - but it may be missing some outliers!

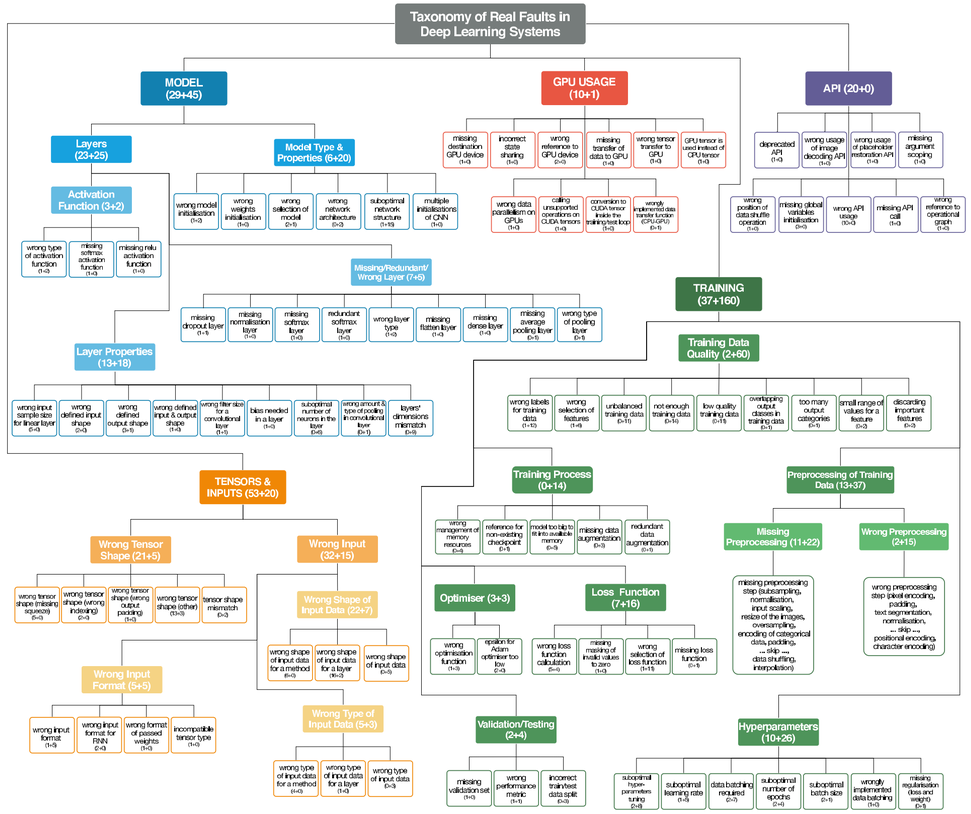
What about coupling effects?

- Trickier to interpret: what are complex faults for DNNs? Essentially their behavior are not interpretable…
- However, there WILL be couplings, as DNN models have wider margin for mistakes - once while you are designing the model, once again while you are training :)

How should we ground the mutation ops?

Taxonomy of real faults in DNN (ICSE 2020)

- Mining GitHub repositories & StackOverflow posts + Developer Interviews
- Then manually organised them into a taxonomy of “things that can go wrong when you build a DNN model”



Pre-training Mutation

DeepCrime (ISSTA 2021)

- Design mutation operators that mimic the human mistakes in the taxonomy!
- Your “test” dataset should be able to distinguish the mutated model.
- But there are issues:
  - Each mutant needs to be not only compiled but trained —> serious cost.
  - What does it mean to kill (distinguish) a mutated model? If we are talking about “differences in model performance”, should it be statistical? This incurs further cost (training of multiple instances, etc)

Table 1: Mutation operators implemented in DeepCrime. Column "ST" indicates the type of search used to find killable configurations (B = binary; EL = exhaustive on list; EU = exhaustive on user provided values).)

Group	Operator	ID	Mutation Parameters	ST
Training Data	Change labels of training data	TCL	*label* — a particular label to mutate *percentage* — a percentage of data for the given label to mutate	- B
	Remove portion of training data	TRD	*percentage* — a percentage of training data to delete	B
	Unbalance training data	TUD	*percentage* — a percentage of training data of underrepresented/selected labels to remove in order to unbalance the training data	B
	Add noise to training data	TAN	*percentage* — a percentage of training data to mutate	B
	Make output classes overlap	TCO	*percentage* — a percentage of training data to mutate	B
Hyperparameters	Change batch size	HBS	*new batch size* — new batch size to be used to train the system under test	EU
	Decrease learning rate	HLR	*new learning rate* — new learning rate to be used to train the system under test	B
	Change number of epochs	HNE	*new number of epochs* — new number of epochs to be used to train the system under test	B
	Disable data batching	HDB	—	-
Activation Function	Change activation function	ACH	*layer* — the number of a layer with non-linear activation function to mutate *new activation function* — new activation function for the layer under mutation	EL EL
	Remove activation function	ARM	*layer* — the number of a layer to mutate	EL
	Add activation function to layer	AAL	*layer* — the number of a layer with linear activation function to mutate *new activation function* — new activation function for the layer under mutation	EL EL
Regularisation	Add weights regularisation	RAW	*layer* — the number of a layer with no weights regularisation to mutate *new weights regularisation* — the type of weights regularisation to be added for the layer under mutation	EL EL
	Change weights regularisation	RCW	*layer* — the number of a layer with existing weights regularisation to mutate *new weights regularisation* — the type of weights regularisation to be added for the layer under mutation	EL EL
	Remove weights regularisation	RRW	*layer* — the number of a layer with existing weights regularisation to mutate	EL
	Change dropout rate	RCD	*layer* — the number of a dropout layer *new dropout rate* — new dropout rate for the layer under mutation	EL EU
	Change patience parameter	RCP	*new patience value* — new value for the patience parameter of the *EarlyStopping* callback	B
Weights	Change weights initialisation	WCI	*layer* — the number of a layer to mutate *new weights initialisation* — new type of kernel initialiser for the layer under mutation	EL EL
	Add bias to a layer	WAB	*layer* — the number of a layer with no bias to mutate	EL
	Remove bias from a layer	WRB	*layer* — the number of a layer with bias to mutate	EL
Loss function	Change loss function	LCH	*new loss function* — new loss function to be used to train the system under test	EL
Optimisation Function	Change optimisation function	OCH	*new optimisation function* — new optimisation function to be used to train the system under test	EL
	Change gradient clipping	OCG	*new gradient clipping* — new value to be used for gradients clipping	EU
Validation	Remove validation set	VRM	—	—

Post-training Mutation

MuFF (https://arxiv.org/abs/2501.09846)

- What is the final impact of pre-training mutation? It results in a different set of DNN weights.
- What if we mutate the weights directly?
- Pros and Cons
  - +: Significantly faster!
  - - : mutation —> slightly changed decision boundaries —> need better (i.e., closer-to-boundary) inputs —> can we justify? what do the mutants mean?

AI4SE: How does AI help SE?

Answer: in so many different ways… but here we focus on the recent applications of LLMs.

“What do you expect from an LLM?”

Let’s go back to 2012

Hindle et al., ICSE 2012

- One of my favourite papers: On Naturalness of Software (https://dl.acm.org/doi/10.5555/2337223.2337322)
- “Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks.”
- But what is “naturalness”?

John: Hi, nice to meet you. How are you?

Mary: I’m ____, _____ ___. ___ ___?

a) fine, thank you. And you?

b) okay, I guess. But why?

What about code?

- It is not “natural”, in the sense that we have artificially created the grammar for programming languages.
  - Programming languages do evolve, but how?
    - Intentionally? New grammars, language consortiums, etc…
    - Gradually? Languages do affect each other, a newer and more popular style eventually gets accepted, etc…

Python: for _ __ _____ …

a) i in range

b) (int i = 0;

Java: for _ ___ _ _ _ _ …

a) i in range

b) (int i = 0;

Statistical Language Model

- Given a set of tokens, $\mathcal{T}$, a set of possible utterances, $\mathcal{T}^*$, and a set of actual utterances, $\mathcal{S} \subset \mathcal{T}^*$, a language model is a probability distribution p over utterances $s \in \mathcal{S}$, i.e., $\forall s \in \mathcal{S}[0 < p(s) < 1 \wedge \sum_{s \in \mathcal{S}} p(s) = 1$

- An utterance (or a sentence) is a sequence of tokens (or words). Suppose we have N tokens, $a_1, a_2, \ldots, a_N$ that consist s. What is $p(s)$?

  - $p(s) = p(a_1)p(a_2 \mid a_1)p(a_3 \mid a_1 \,.\, a_2)p(a_4 \mid a_1, a_2, a_3)\ldots p(a_N \mid a_1 \ldots a_{N-1})$

  - But these conditional probabilities are hard to calculate: the only feasible approach would be count each utterance that qualifies, but $\mathcal{S}$ is too big, let alone $\mathcal{T}^*$.

N-Grams

- Assumes a Markovian property, i.e., the next token is influenced only by those came immediately before (say, within the window of n tokens)!
  - $p(a_i \mid a_1 \ldots a_{i-1}) \simeq p(a_i \mid a_{i-3} a_{i-2} a_{i-1})$
- This is now much more tractable:
  - $p(a_i \mid a_{i-3} a_{i-2} a_{i-1}) = \frac{count(a_{i-3}, a_{i-2}, a_{i-1}, a_i)}{count(a_{i-3}, a_{i-2}, a_{i-1}, *)}$
- Given some context, we can now compute the probability of the candidate token that comes next. In other words, we can predict the next token!

Large Language Model

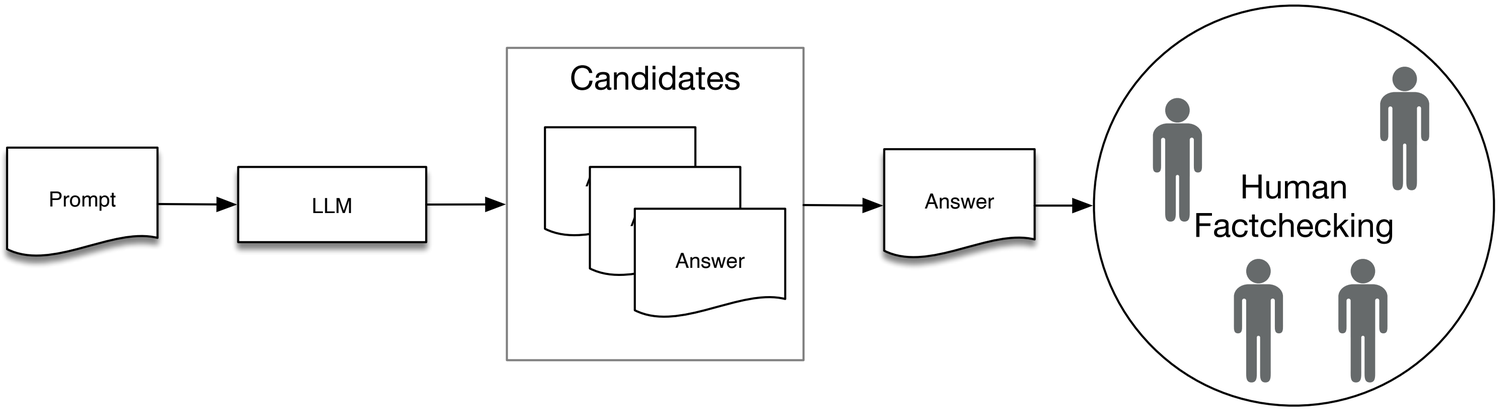
(really, a very large and powerful statistical language model)

- Mainly Transformer-based DNNs that are trained to be an auto-regressive language model, i.e., given a sequence of tokens, it repeatedly tries to predict the next token.
- They **seem to** get the semantics of the code and **work across natural and programming language, which is unprecedented!**
- But they are inherently restricted as a statistical language model.
  - No memory/state, no background/domain knowledge…

Code is a unique artefact because it executes.

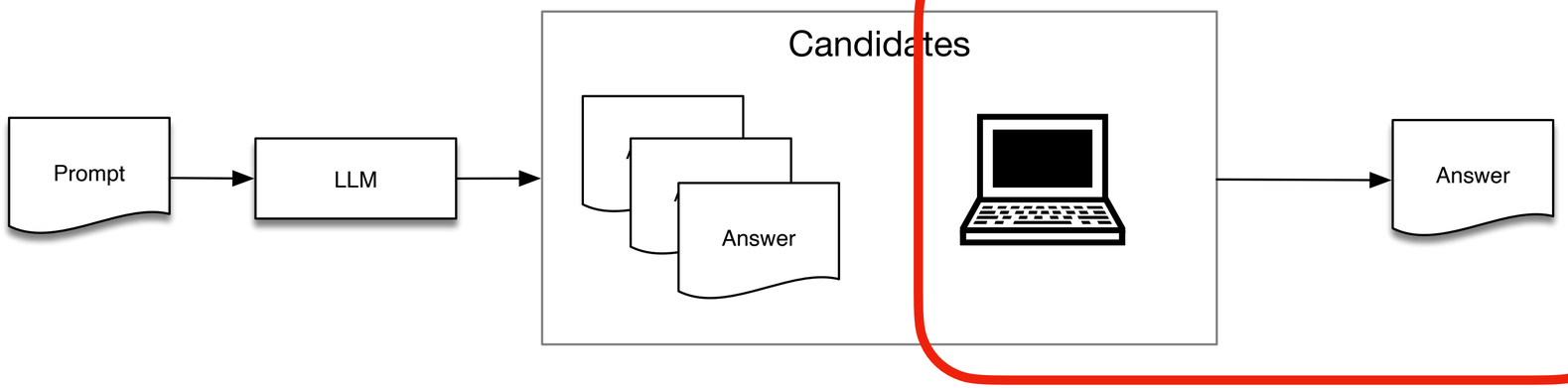
(And we've been doing dynamic analysis for a long time)

NL + LLM Pipeline



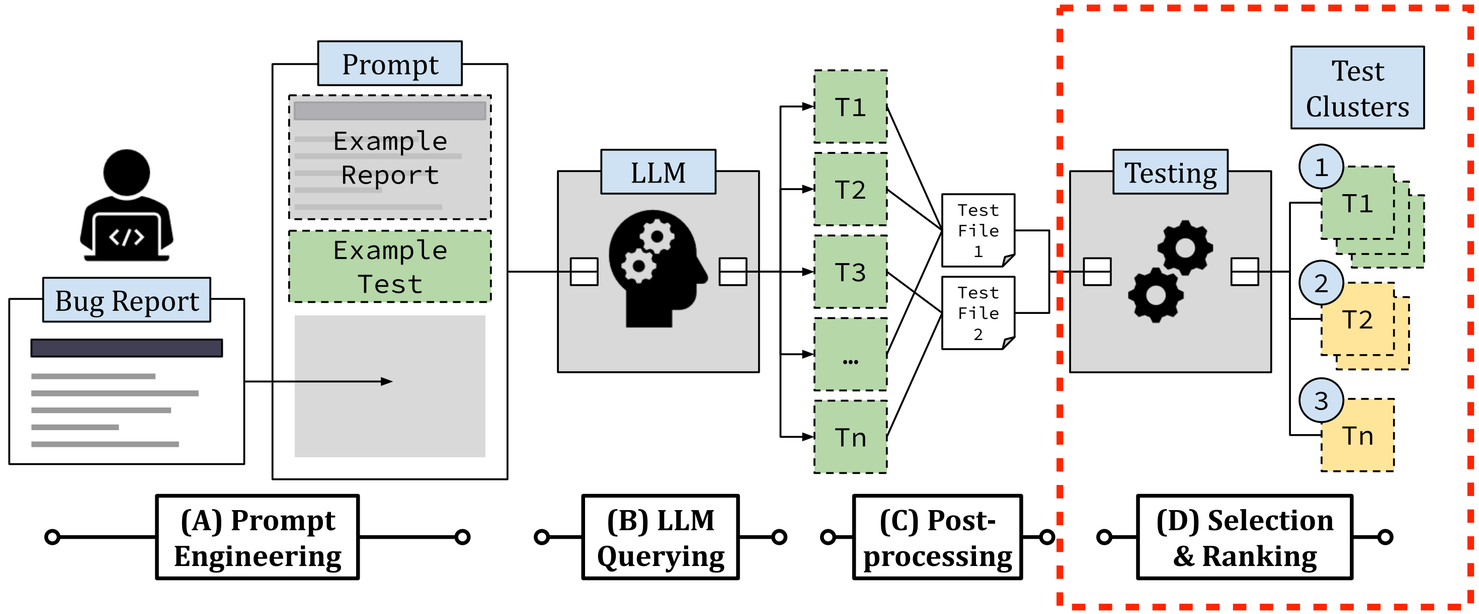
Isn't this testing? :)

PL/NL + LLM Pipeline



Execution-Based Filtering of LLM Output

LLM-based Bug Reproduction (Kang, Yoon & Yoo, ICSE 2023)



- Any test that does not fail in the buggy version are filtered out!
- Failure type and error messages are considered when clustering tests —> larger clusters are more reliable

LLMs will generate incorrect comments. But…

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)

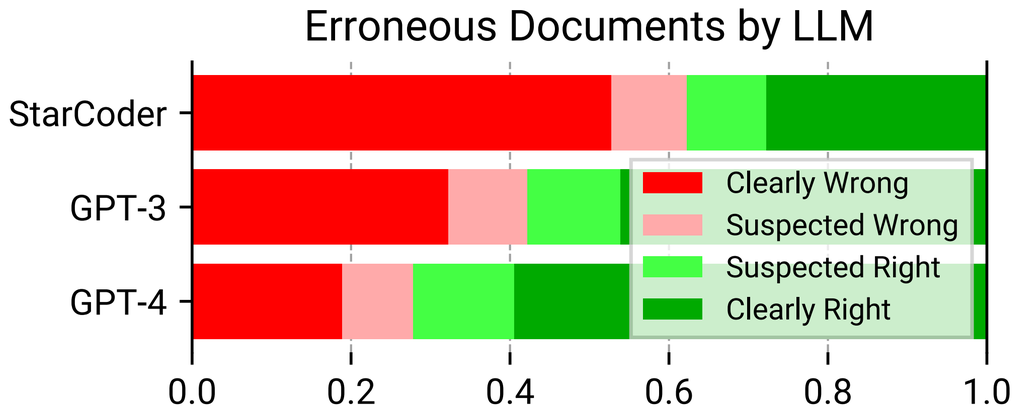


Figure 1: Comment factual accuracy by generating LLM.

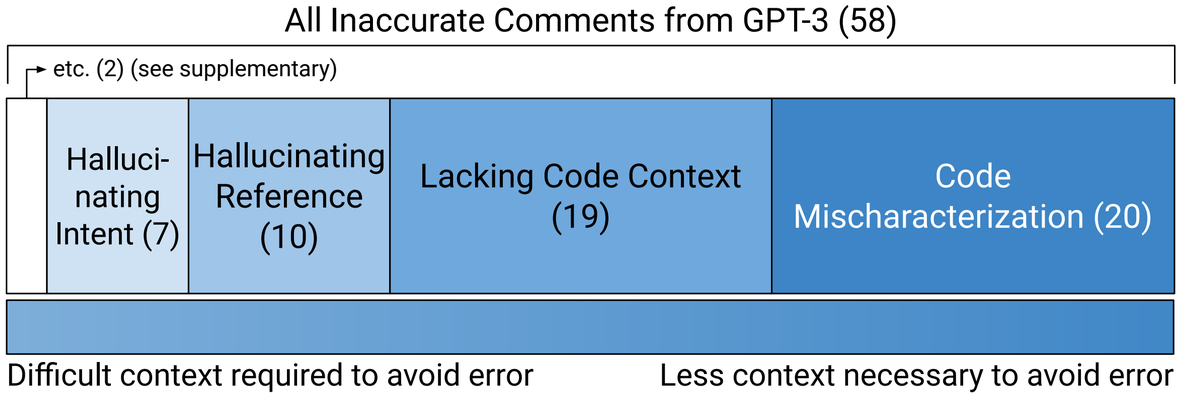
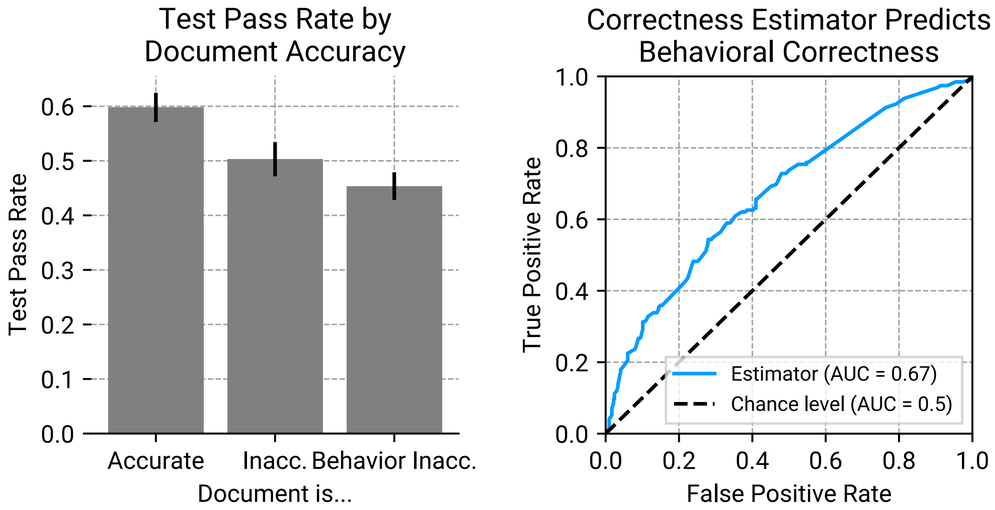


Figure 2: Diagram of error taxonomy for GPT-3 comments.

Evaluating code comment quality is notoriously hard. We tell LLMs to synthesise test cases based on comments —> if generate test cases fail, comments are not good.

Test Pass Rates Correlates with Doc Quality

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)



(a) Pass rate by accuracy, with 95% confidence intervals.

(b) ROC graph of correctness estimator with actual correctness.

Figure 4: Relationship between comment accuracy and suggested indicators.

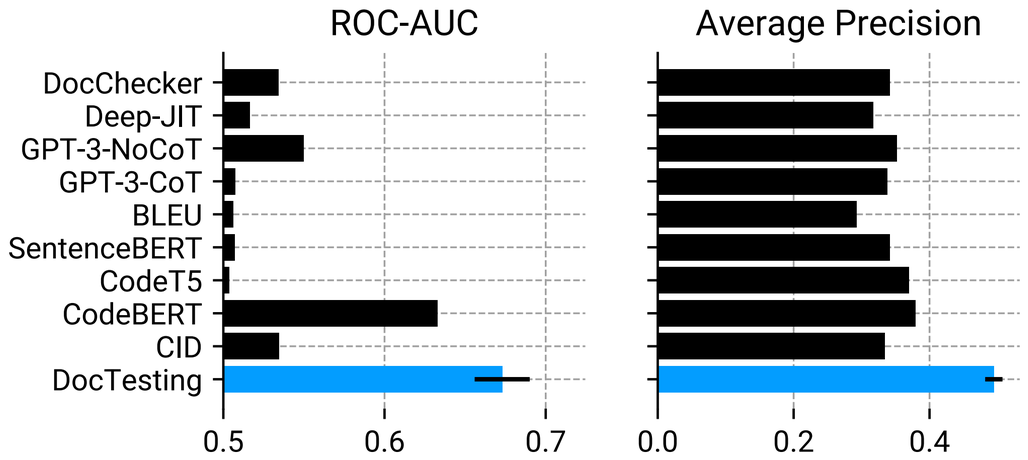
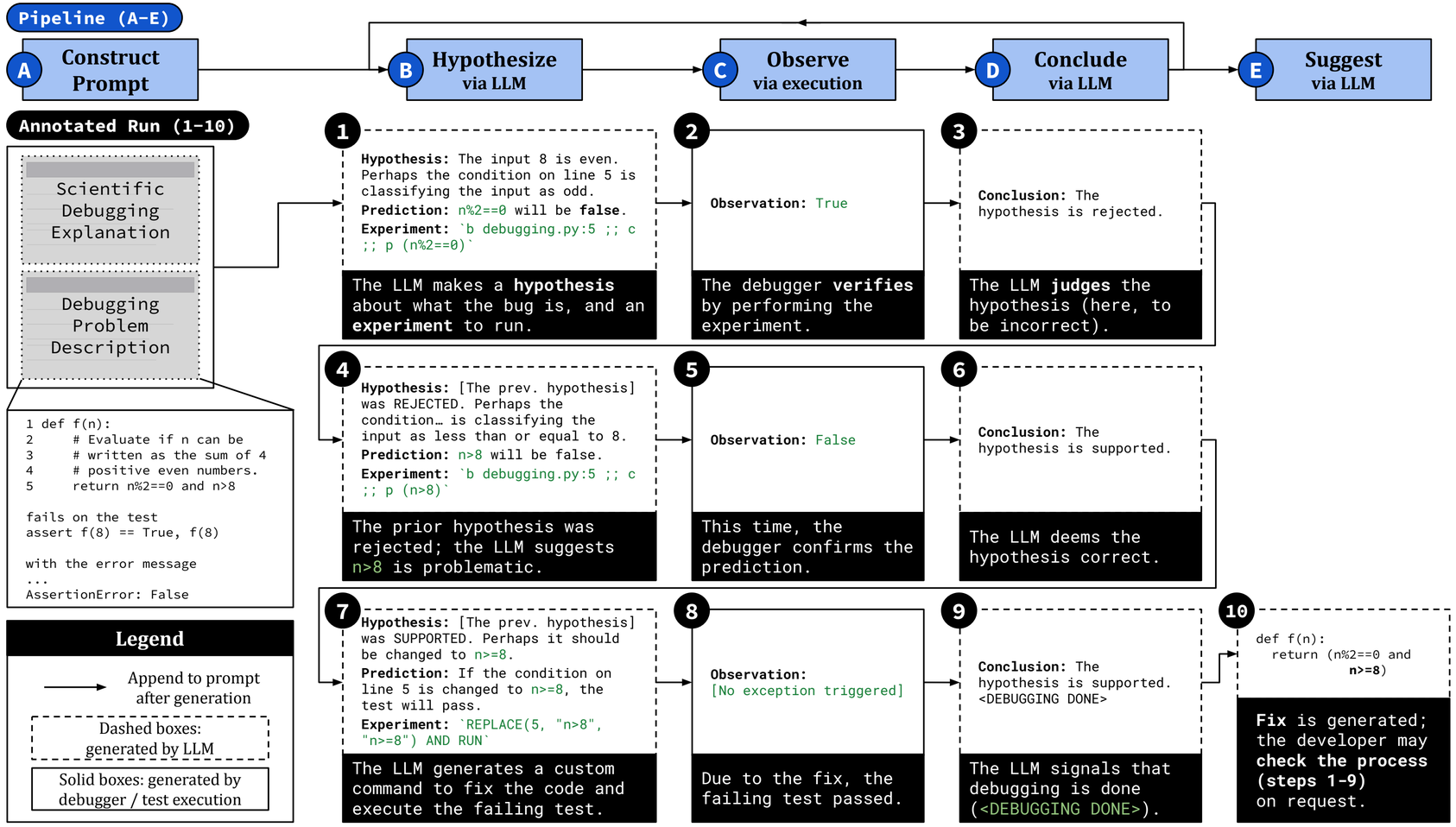


Figure 5: ROC-AUC and AP values compared with baselines. For our approach (blue), we present the mean value from five runs, along with its 95% confidence interval.

If we then tell LLMs to synthesise test cases based on comments
—> the pass rate of generated tests correlates with comment quality.

Zero-shot Automated Debugging

Kang et al., EMSE 2025 (https://arxiv.org/abs/2304.02195)



It hypothesise about the bug, but then can also validate its own hypotheses dynamically using debugger.

Dynamic Feedback enables Autonomy

Yoon et al., ICST 2024 (https://arxiv.org/abs/2311.08649)

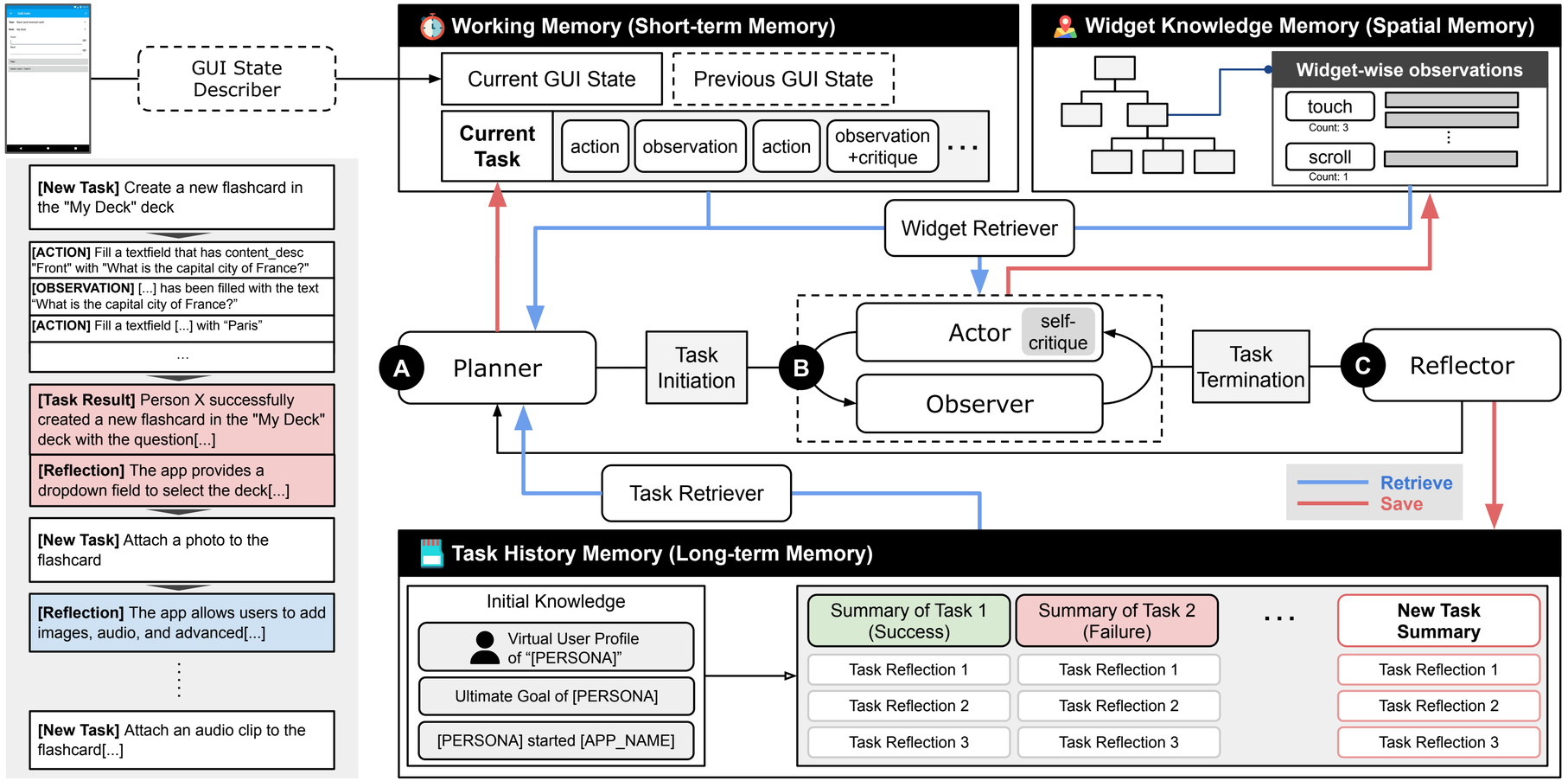


Fig. 1. Overview of DROIDAGENT with a task example.

LLMs for Software Engineering

So many things happening all around us, but for now I would say:

- “Chat” may not be an ideal platform to iteratively refine your solution: you need additional information source, ideally dynamic feedback from concrete executions
- “Agents” seem to be the next wave of change: LLMs will drive the problem solving “strategy” while various external parts are providing information.
    - Even multi-agent architecture: a single application, inside which multiple LLMs are doing a range of sub-tasks, collaborating with each other.
- We need to learn how to build these agents well, not to mention test <— very little known so far.

We are still in the Chinese room

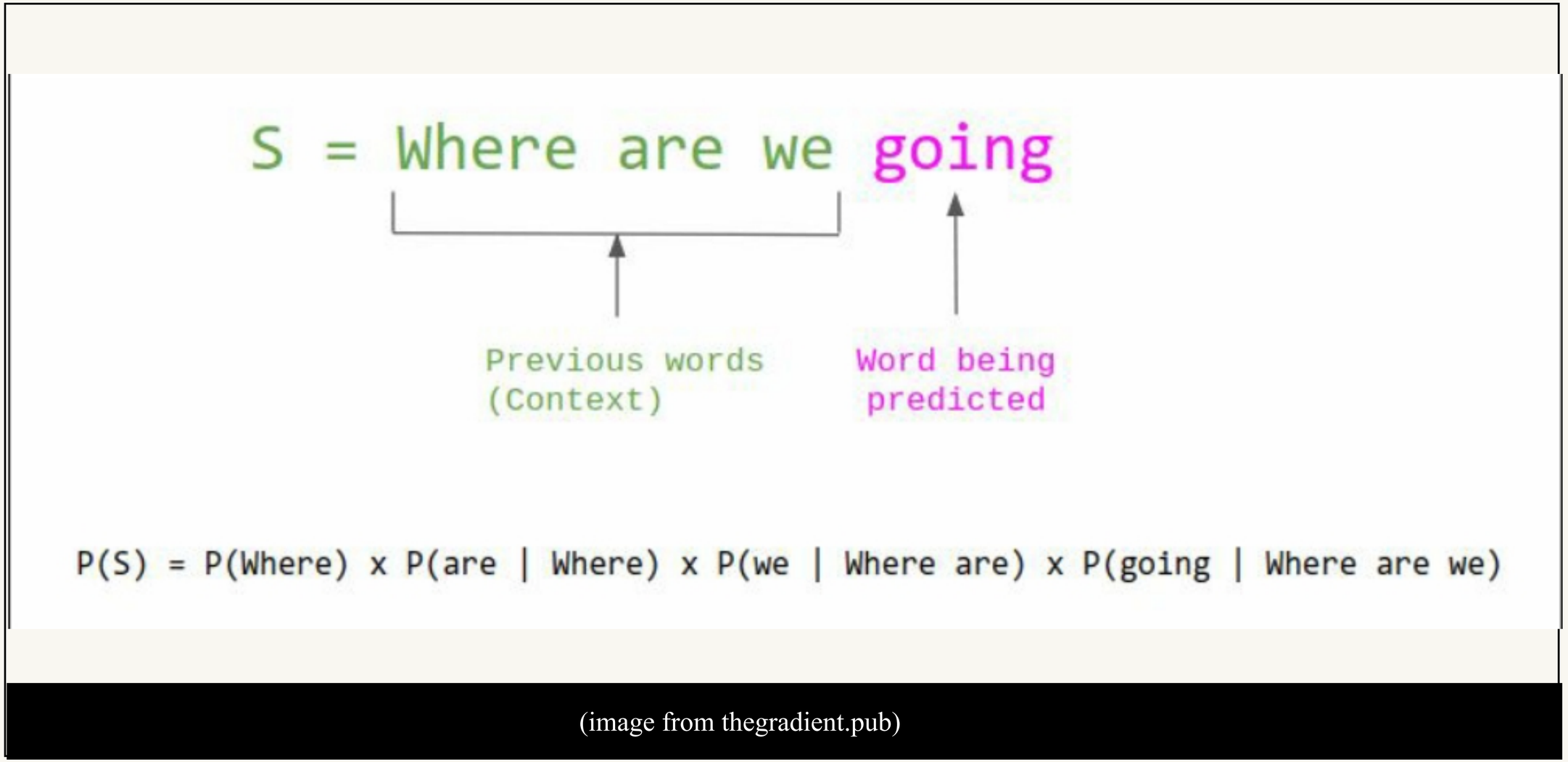
John Searle, “Mind, Brains, and Programs” in 1980

- Suppose we have a computer program that behaves as if it understands Chinese language.
- You are in a closed room with the AI program source code.
- Someone passes a paper with Chinese characters written on it, into the room.
- You use the source code as instruction to generate the response to the input, and sends the response out of the room.
- Do you understand Chinese language, or not?

Summary

- AI/ML model as part of larger software system will be the future - but this calls for very new approach towards testing.
- LLMs are THE topic right now, but there are also so much hype:

Language Models are Autocomplete Machines



(image from thegradient.pub)

Formulating bug reproduction as autocomplete

Listing 1: Example prompt without examples.

````
# NaN in "equals" methods
## Description
In "MathUtils", some "equals" methods will return true if both argument
      are NaN.
Unless I'm mistaken, this contradicts the IEEE standard.
If nobody objects, I'm going to make the changes.

## Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void test
````

Report Content

The first part of the prompt presents the bug report.
````

Formulating bug reproduction as autocomplete

Listing 1: Example prompt without examples.

```
1  # NaN in "equals" methods
2  ## Description
3  In "MathUtils", some "equals" methods will return true if both argument
       are NaN.
4  Unless I'm mistaken, this contradicts the IEEE standard.
5  If nobody objects, I'm going to make the changes.
6
7  ## Reproduction
8  >Provide a self-contained example that reproduces this issue.
9  ```
10 public void test
```

Prompting Reproducing Test Generation

The second part increases the likelihood of a bug-reproducing test
(from a language distribution perspective).

LLMs are known to benefit with examples

````
>Provide a self-contained example that reproduces this issue.
```
public void testNumberUtils () {
    assertEquals(Long.valueOf(0x80000000L),    NumberUtils.createNumber("0x80000000"));
}
```

# #107 Incorrect date parsed when week and month used together
## Description
I have following code snippet :

```
    DateTimeFormatter dtf = DateTimeFormat.forPattern("xxxxMM'w'ww");
DateTime dt = dtf.parseDateTime("201101w01");
System.out.println(dt);
```

It should print 2011-01-03 but it is printing 2010-01-04.

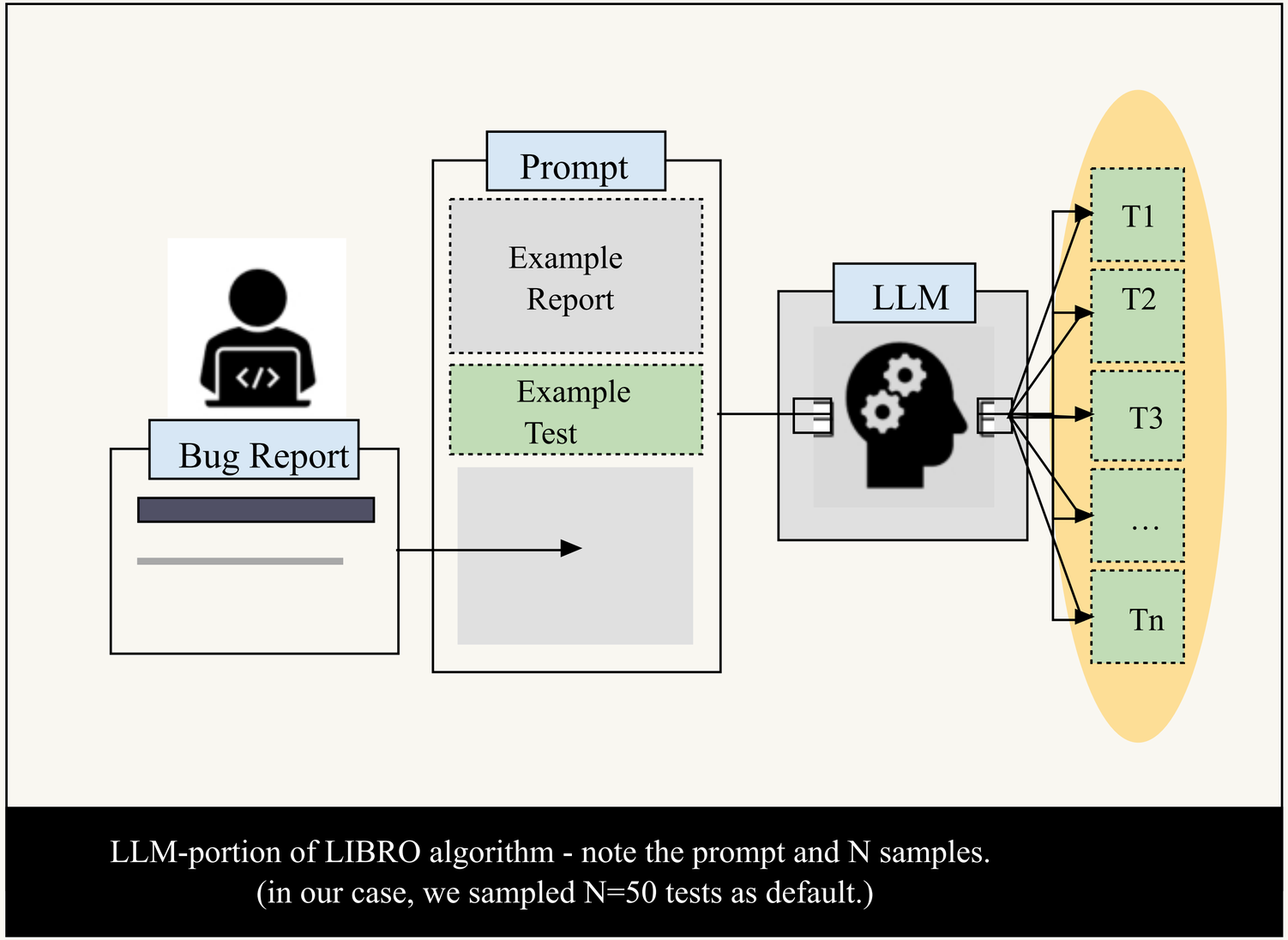
Please let me know if I am doing something wrong here

## Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void testIssue107() {
    DateTimeFormatter dtf = DateTimeFormat.forPattern("xxxxMM'w'ww");
    DateTime dt = dtf.parseDateTime("201101w01");
    assertEquals(2011, dt.getYear());
    assertEquals(1, dt.getMonthOfYear());
    assertEquals(3, dt.getDayOfMonth());
}
```

# {{title}}
## Description
{{content}}
## Reproduction
>Provide a self-contained example that reproduces this issue.
```
public void test
{{endon}}:```
````

A prompt template we used for experiments.
Note the example answers (highlighted).
````

Given a prompt, sample N candidate tests.

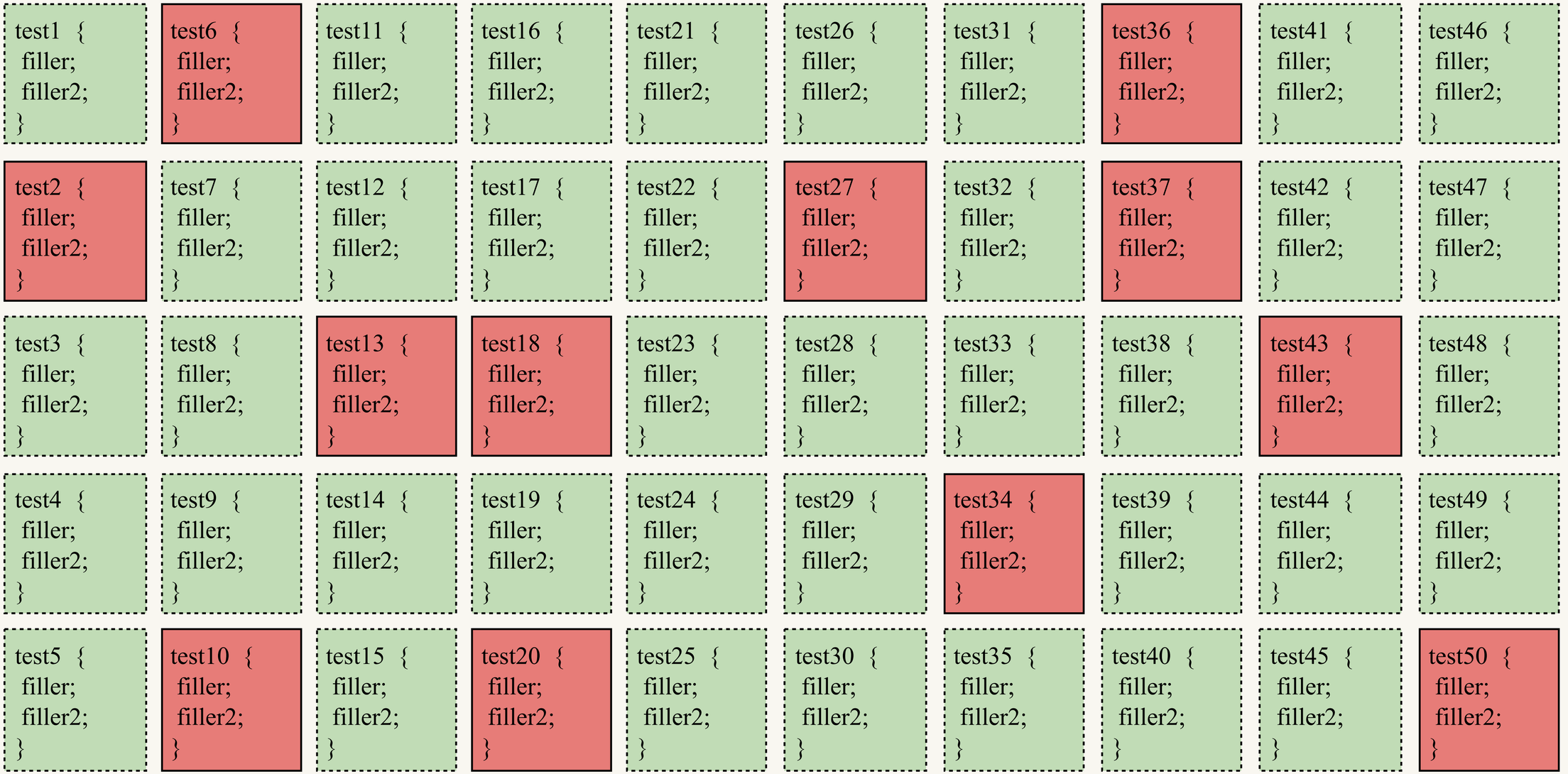


LLM-portion of LIBRO algorithm - note the prompt and N samples.
(in our case, we sampled N=50 tests as default.)

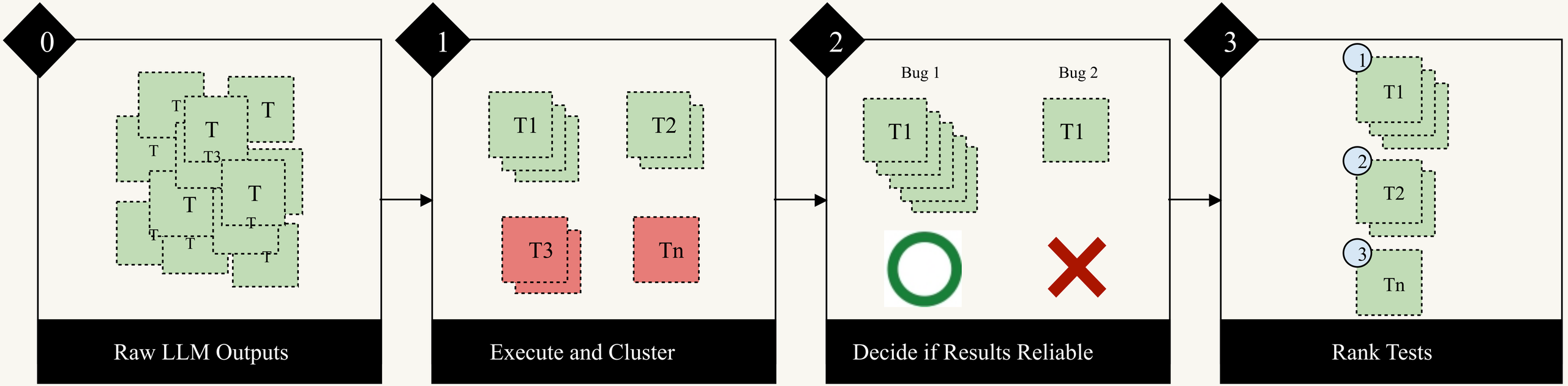
Showing 50 tests is infeasible

```
test1 {
 filler;
 filler2;
}
```
```
test6 {
 filler;
 filler2;
}
```
```
test11 {
 filler;
 filler2;
}
```
```
test16 {
 filler;
 filler2;
}
```
```
test21 {
 filler;
 filler2;
}
```
```
test26 {
 filler;
 filler2;
}
```
```
test31 {
 filler;
 filler2;
}
```
```
test36 {
 filler;
 filler2;
}
```
```
test41 {
 filler;
 filler2;
}
```
```
test46 {
 filler;
 filler2;
}
```
```
test2 {
 filler;
 filler2;
}
```
```
test7 {
 filler;
 filler2;
}
```
```
test12 {
 filler;
 filler2;
}
```
```
test17 {
 filler;
 filler2;
}
```
```
test22 {
 filler;
 filler2;
}
```
```
test27 {
 filler;
 filler2;
}
```
```
test32 {
 filler;
 filler2;
}
```
```
test37 {
 filler;
 filler2;
}
```
```
test42 {
 filler;
 filler2;
}
```
```
test47 {
 filler;
 filler2;
}
```
```
test3 {
 filler;
 filler2;
}
```
```
test8 {
 filler;
 filler2;
}
```
```
test13 {
 filler;
 filler2;
}
```
```
test18 {
 filler;
 filler2;
}
```
```
test23 {
 filler;
 filler2;
}
```
```
test28 {
 filler;
 filler2;
}
```
```
test33 {
 filler;
 filler2;
}
```
```
test38 {
 filler;
 filler2;
}
```
```
test43 {
 filler;
 filler2;
}
```
```
test48 {
 filler;
 filler2;
}
```
```
test4 {
 filler;
 filler2;
}
```
```
test9 {
 filler;
 filler2;
}
```
```
test14 {
 filler;
 filler2;
}
```
```
test19 {
 filler;
 filler2;
}
```
```
test24 {
 filler;
 filler2;
}
```
```
test29 {
 filler;
 filler2;
}
```
```
test34 {
 filler;
 filler2;
}
```
```
test39 {
 filler;
 filler2;
}
```
```
test44 {
 filler;
 filler2;
}
```
```
test49 {
 filler;
 filler2;
}
```
```
test5 {
 filler;
 filler2;
}
```
```
test10 {
 filler;
 filler2;
}
```
```
test15 {
 filler;
 filler2;
}
```
```
test20 {
 filler;
 filler2;
}
```
```
test25 {
 filler;
 filler2;
}
```
```
test30 {
 filler;
 filler2;
}
```
```
test35 {
 filler;
 filler2;
}
```
```
test40 {
 filler;
 filler2;
}
```
```
test45 {
 filler;
 filler2;
}
```
```
test50 {
 filler;
 filler2;
}
```

Some might not even compile!



LIBRO’s post-processing in three steps

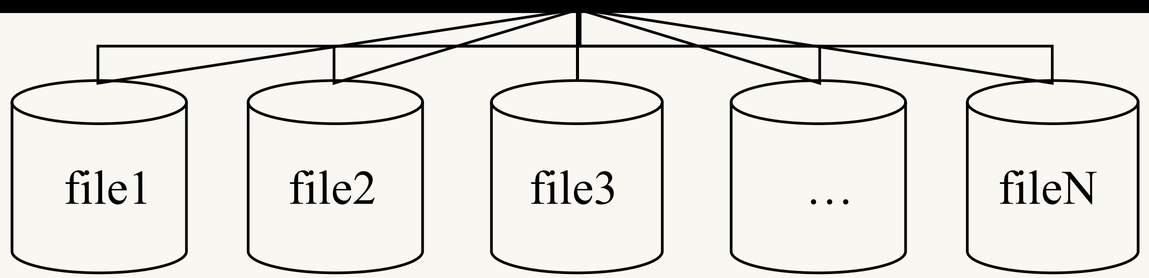


Injecting to target files

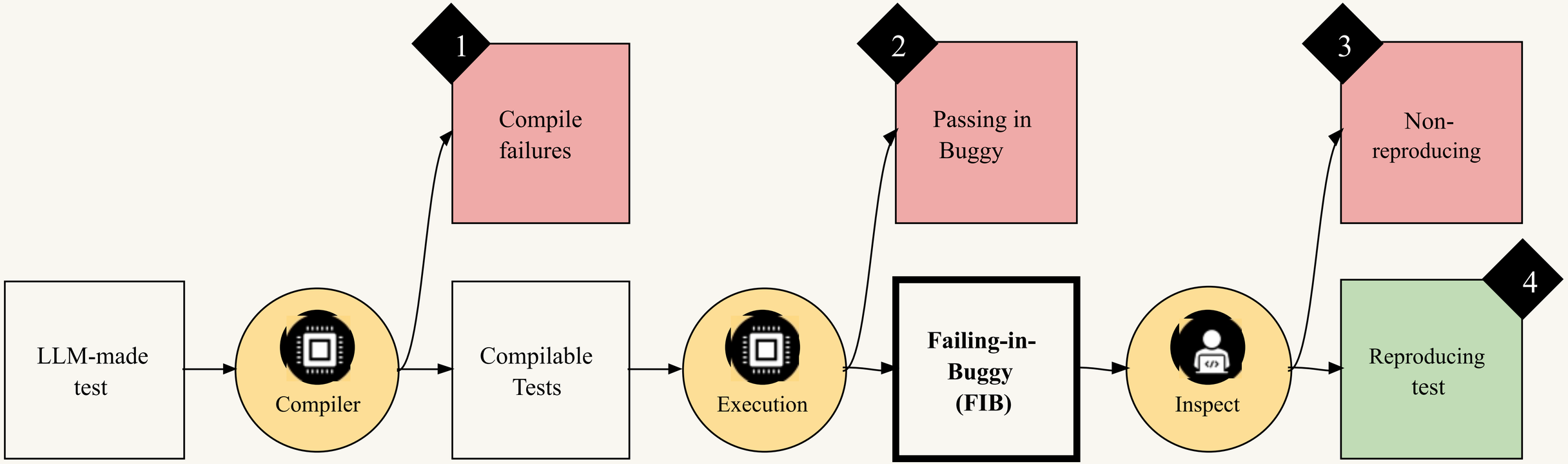
Listing 2: Example LLM result from the bug report described in Table II.

```
1  public void testEquals() {
2      assertFalse(MathUtils.equals(Double.NaN, Double.NaN));
3      assertFalse(MathUtils.equals(Float.NaN, Float.NaN));
4  }
```

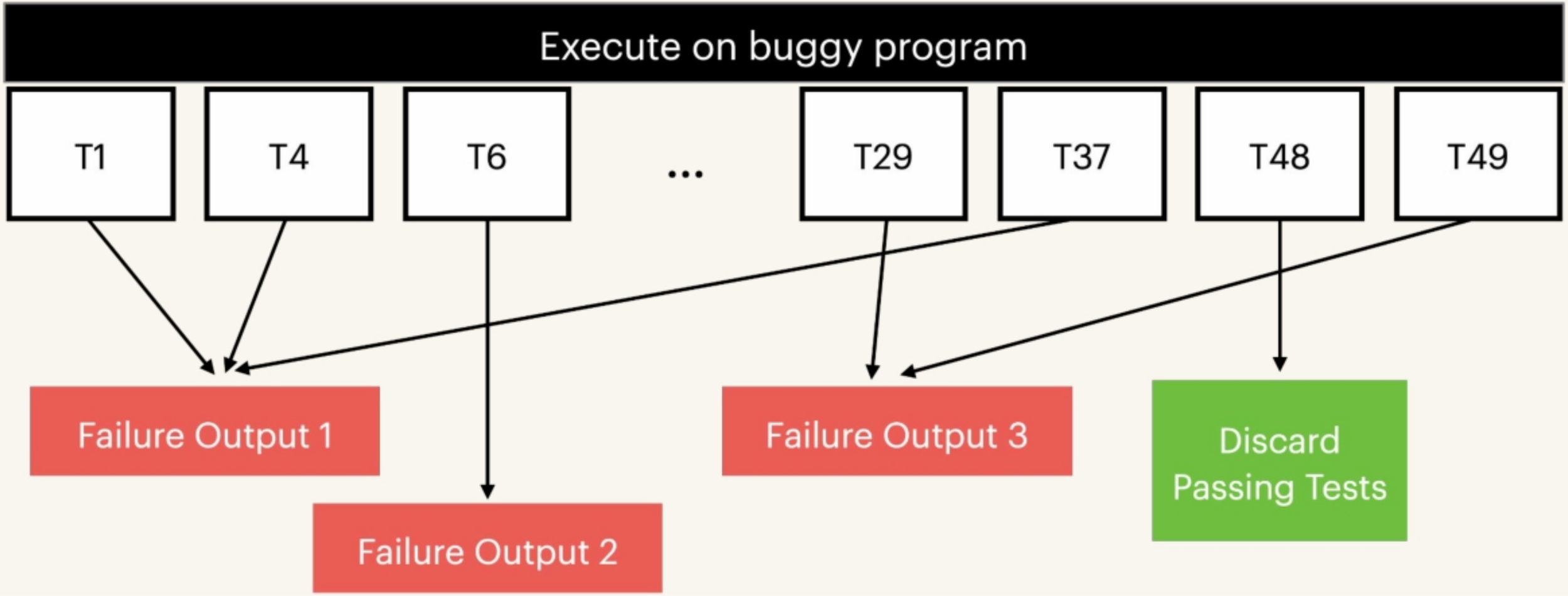
Select the file with greatest lexical similarity and inject the test; add import statements for unmet dependencies.



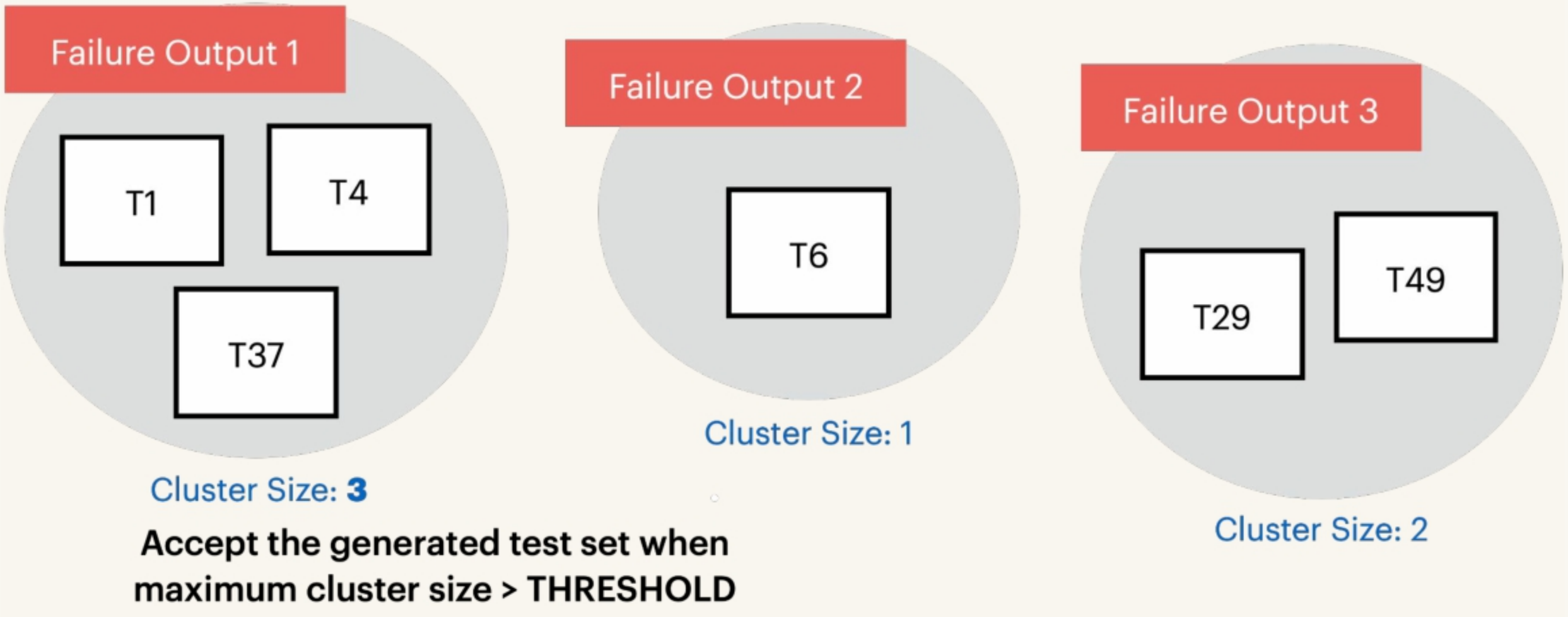
Execute Tests. Four results possible:



Cluster FIB tests with error message



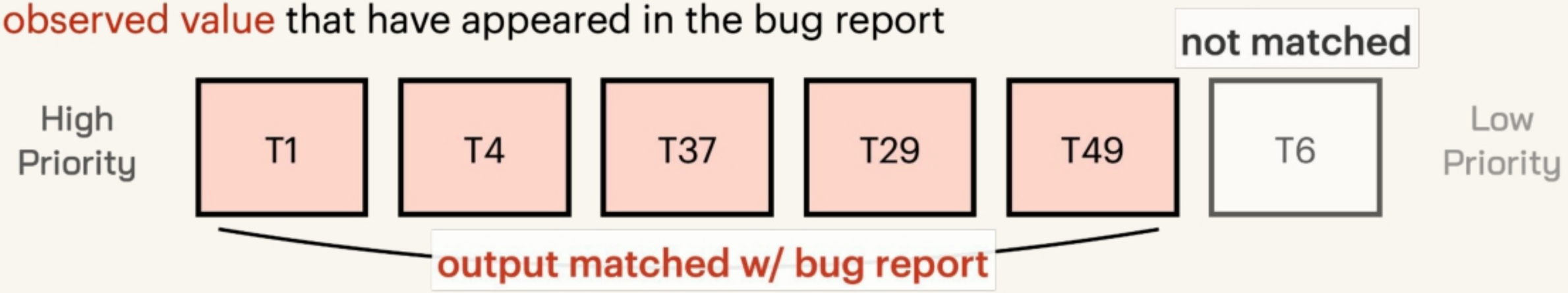
Show results only if cluster size large enough



Ranking tests with three heuristics (1)

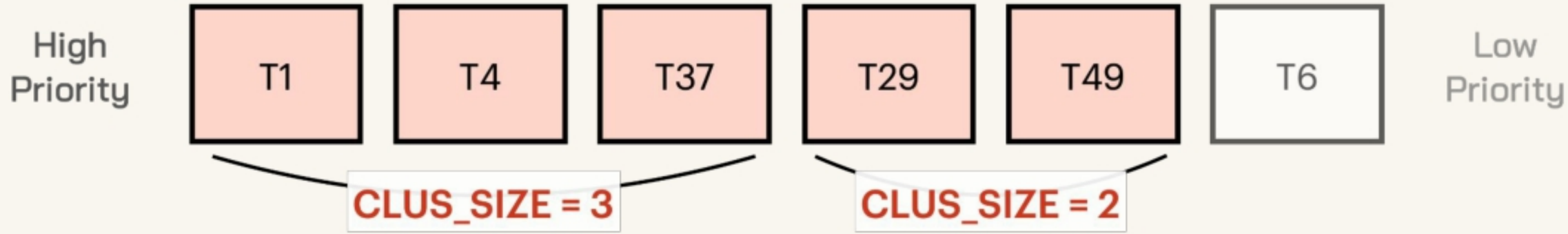
Q. Which test is more likely to be a correct bug reproducing test?

A1. (Matching w/ Bug Report) When test outputs **include** exception type or observed value that have appeared in the bug report



Ranking tests with three heuristics (2)

A2. (Consensus level) Tests from **larger output cluster** are prioritized



Recap

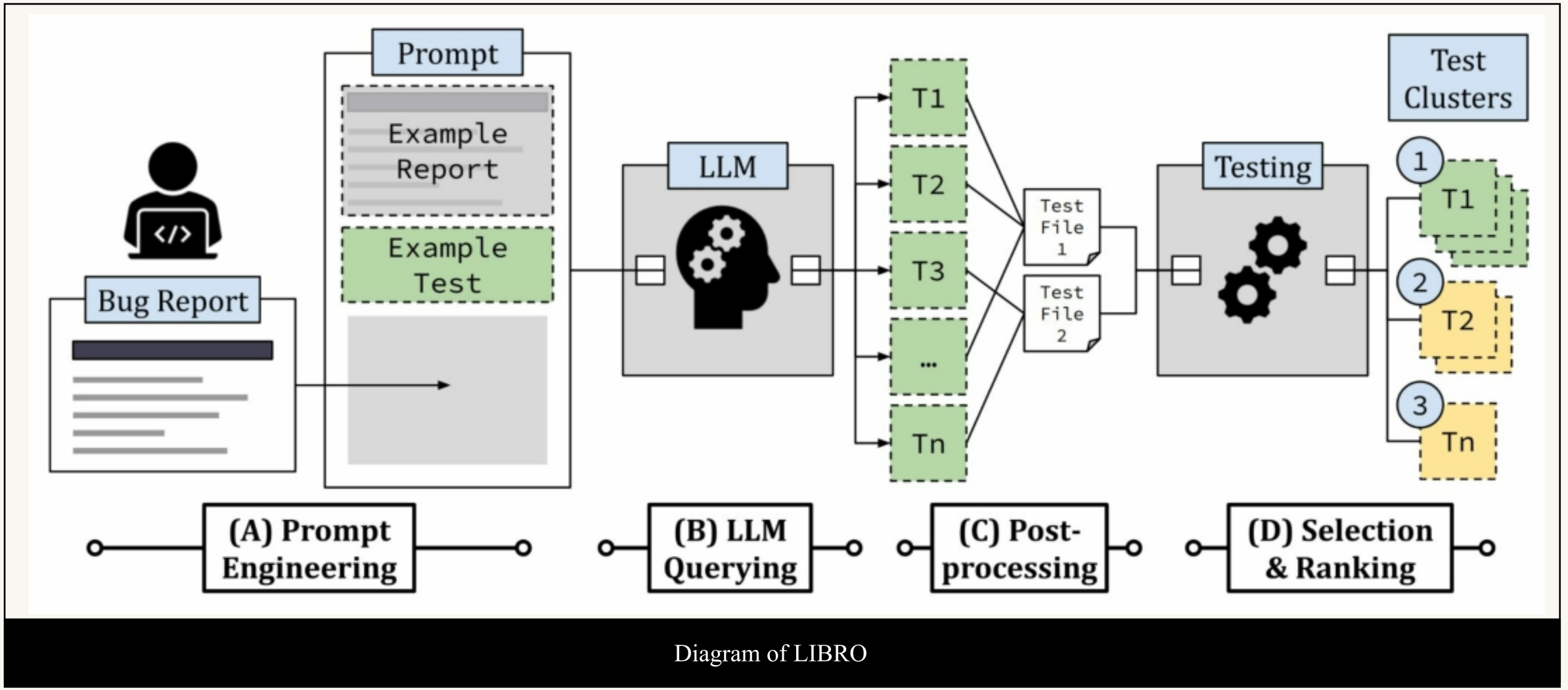
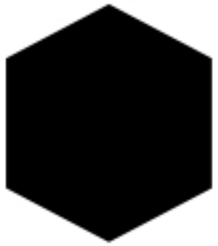


Diagram of LIBRO



Results

Evaluating the Technique

RQ1: Efficacy

How many bugs reproduced? (D4J)

RQ2: Efficiency

How much resources are required?

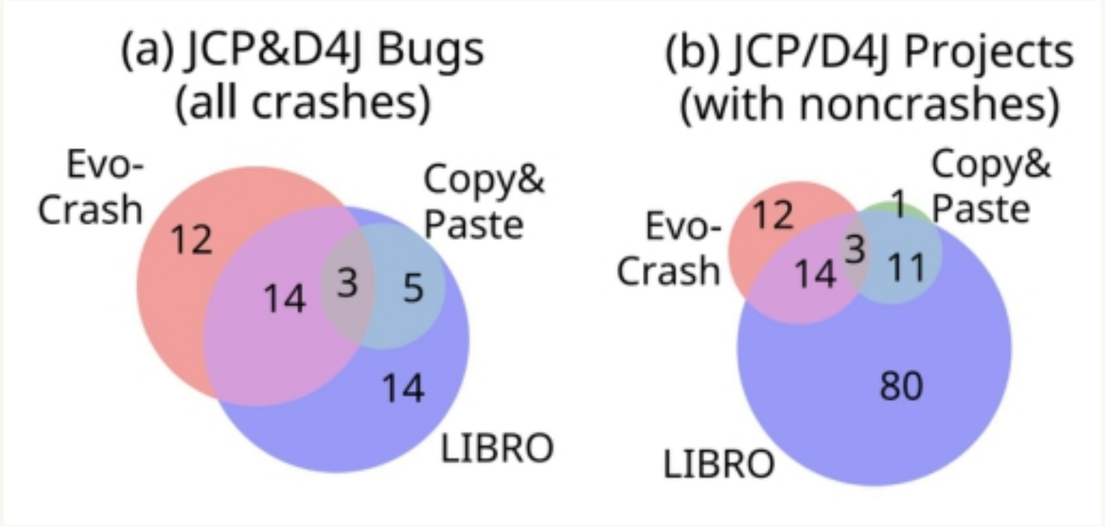
RQ3: Generality

Are results similar on held-out data?

RQ1: Efficacy

Setting	reproduced	FIB
No Example (n=10)	124	440
One Example (n=10)	166	417
One Example from Source Project (n=10)	152	455
One Example with Constructor Info (n=10)	167	430
Two Examples (n=10, 5th percentile)	161	386
Two Examples (n=10, median)	173	409
Two Examples (n=10, 95th percentile)	184	429
Two Examples (n=50)	**251**	**570**
One Example, Crash Bugs (n=10)	69	153
One Example with Stack, Crash Bugs (n=10)	84	155

RQ1-1: One-third of all bugs were successfully reproduced.



RQ1-2: LIBRO significantly outperforms baselines.

RQ2-2: Time cost of each component

TABLE V: The time required for the pipeline of LIBRO

	Prompt	API	Processing	Running	Ranking	Total
Single Run	<1 μs	5.85s	1.23s	4.00s	-	11.1s
50-test Run	<1 μs	292s	34.8s	117s	0.02s	444s

The API call and actual execution of the test took the longest amount of time.

RQ2-3: Selection performance

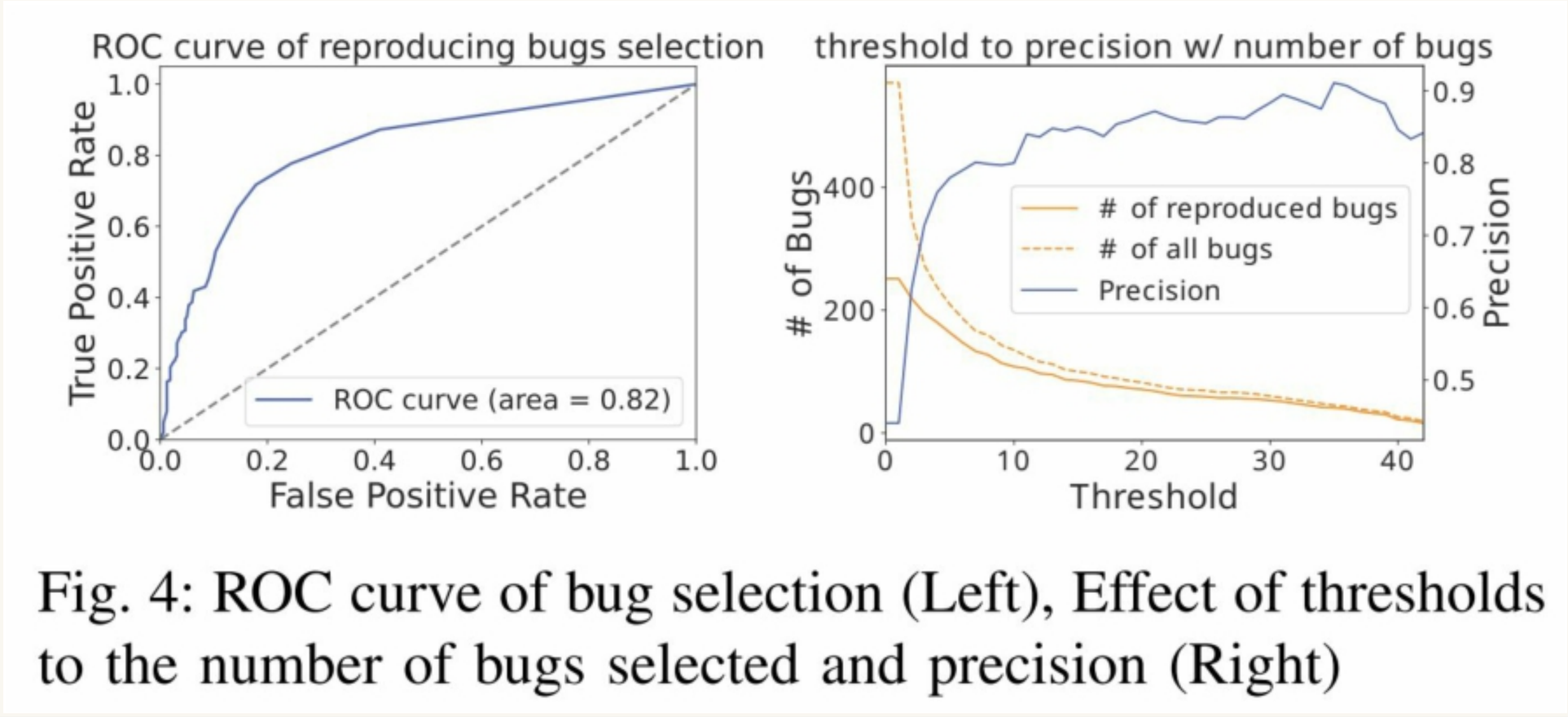


Fig. 4: ROC curve of bug selection (Left), Effect of thresholds to the number of bugs selected and precision (Right)

Selecting by counting the failing-in-buggy tests was effective;

RQ2-3: Ranking performance

TABLE VI: Ranking Performance Comparison between LIBRO and Random Baseline

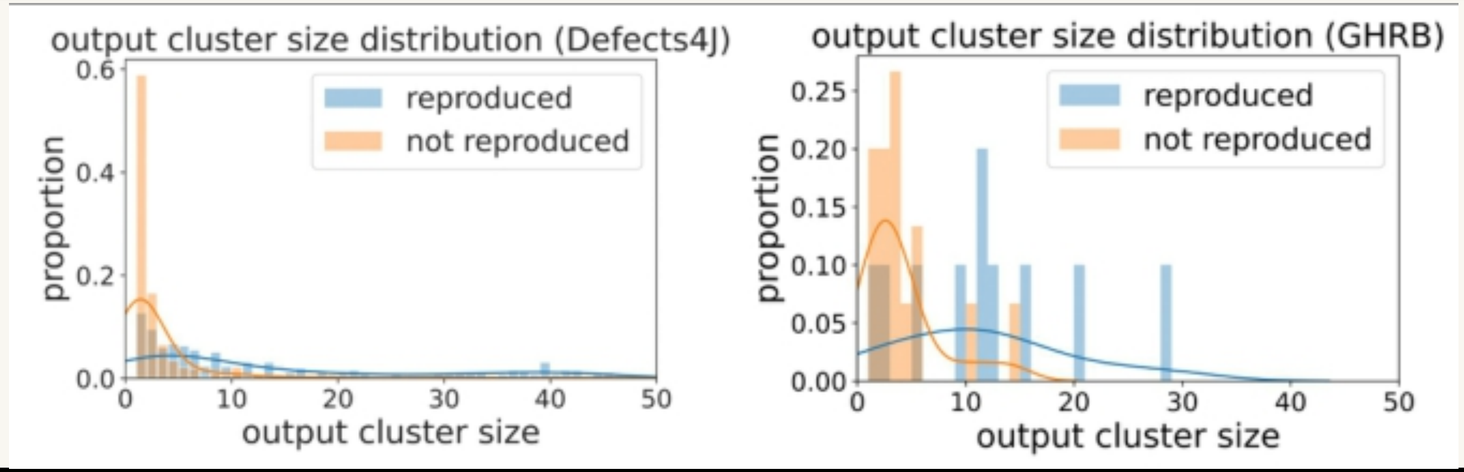
	Defects4J				GHRB			
	$acc@n$ (*precision*)		$wef@n_{agg}$		$acc@n$ (*precision*)		$wef@n_{agg}$	
n	LIBRO	random	LIBRO	random	LIBRO	random	LIBRO	random
1	**149** (**0.43**)	116 (0.33)	**201** (**0.57**)	234 (0.67)	**6** (**0.29**)	4.8 (0.23)	**15** (**0.71**)	16.2 (0.77)
3	**184** (**0.53**)	172 (0.49)	**539** (**1.54**)	599 (1.71)	**7** (**0.33**)	6.6 (0.31)	**42** (**2.0**)	44.6 (2.12)
5	**199** (**0.57**)	192 (0.55)	**797** (**2.28**)	874 (2.5)	**8** (**0.38**)	7.3 (0.35)	**60** (**2.86**)	64.3 (3.06)

Our within-FIB ranking technique was better than the random baseline.

RQ3: Generality

Project	rep/total	Project	rep/total	Project	rep/total
AssertJ	3/5	Jackson	0/2	Gson	4/7
checkstyle	0/13	Jsoup	2/2	sslcontext	1/2

RQ3-1: LIBRO demonstrating it works outside of its (potential) training data.



RQ3-2: LIBRO selection and ranking heuristics generalize to novel data as well.

Successful Example

DateTimeZone.forOffsetHoursMinutes cannot handle negative offset < 1 hour

DateTimeZone.forOffsetHoursMinutes(h, m) cannot handle negative offset < 1 hour like -0:30 due to argument range checking. I used forOffsetMillis () instead.
This should probably be mentioned in the documentation or negative minutes be accepted.

Bug Report: Joda-Time BUG #8

```
public void testIssue() {
    DateTimeZone tz =
DateTimeZone. forOffsetHoursMinutes (0, -30);
    assertNotNull (tz);
}
```

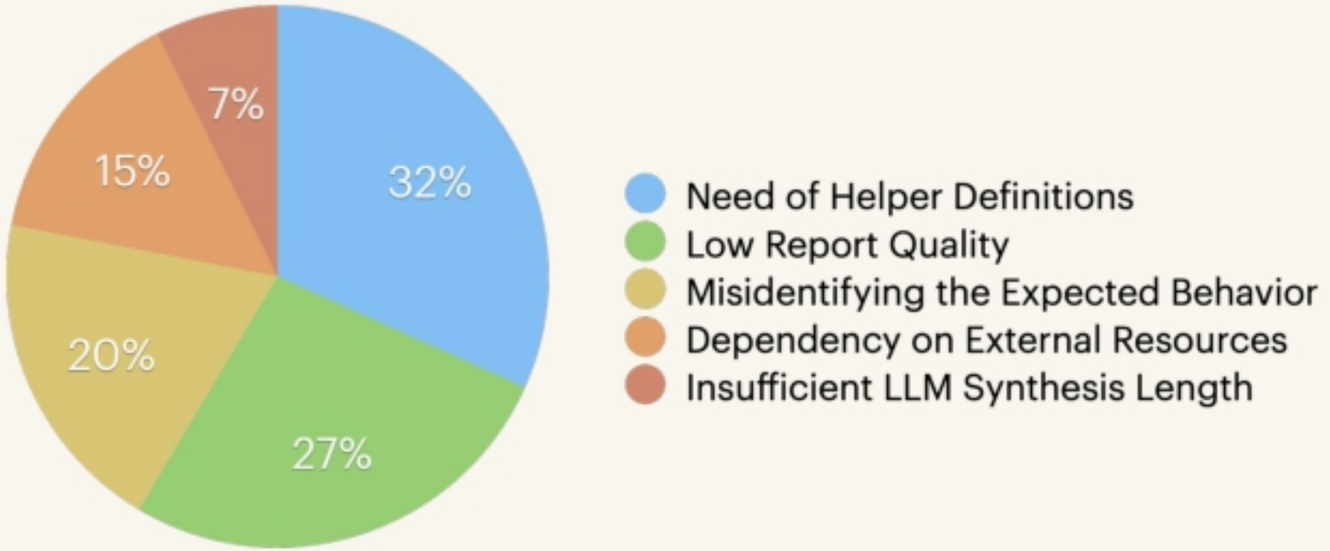
A test generated by LIBRO

```
public void testForOffsetHoursMinutes_int_int() {
    ...
    assertEquals (DateTimeZone.forID("-00:15"),
DateTimeZone. forOffsetHoursMinutes (0, -15));
    ...
}
```

A developer-written test

Examples and Failure Analysis

- When failures happened, we find that
  - 32.5% are due to a need of complex helper functions;
  - 27.5% are due to low report quality;
  - 20% are due to LLM misunderstanding of report;
  - 15% are due to dependency on external resources;
  - 7.5% are due to LLM synthesis limit (we set 256 tokens, or ~1000 characters).



Conclusion

Read our preprint!

1. We tackle the problem of **reproducing general bugs from reports.**

2. We propose LIBRO, which **combines LLMs and postprocessing** to effectively reproduce bug reports.

3. Our evaluation shows **LIBRO successfully reproduces bugs**, and that its postprocessing heuristics work.

Contact us at sungmin.kang@kaist.ac.kr / juyeon.yoon@kaist.ac.kr
Find our preprint with the QR code above, or by searching for “Exploring LLM-based General Bug Reproduction”