# Hot Takes on Machine Learning for Program Analysis (Calgary)

Patrick Lam
University of Waterloo

14 May 2025


# My student said:

Deep Learning is one of the ("best") ways to do program analysis.

# What’s ML good at? Bad at?

We’ll talk about some ML techniques:

- Classification
- Inference
- Generative AI
- Analysis / Deduction

# Classification

Weka about to steal my lunch

# Classification

FSE 08: Bodden, Lam & Hendren. “Finding Programming Errors Earlier by Evaluating Runtime Monitors Ahead-of-Time”.

ML task:

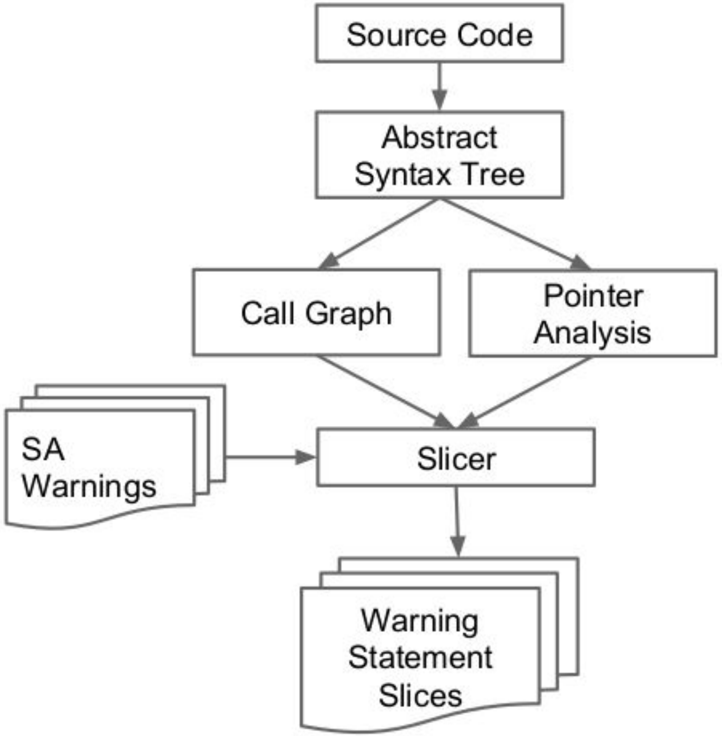
filter out likely false-positive Potential Points of Failure.

```
CALL = 0
|   ABORTED = 0
|   |   DELEGATE = 0
|   |   |   NO_CONTEXT = 0: TRUE_POSITIVE (11.0/1.0)
|   |   |   NO_CONTEXT = 1: FALSE_POSITIVE (4.0/1.0)
|   |   DELEGATE = 1: FALSE_POSITIVE (10.0)
|   ABORTED = 1: FALSE_POSITIVE (30.0)
CALL = 1: FALSE_POSITIVE (406.0/1.0)
```

# More Classification

MSR 14: Hanam, Tan, Holmes, and Lam.
“Finding Patterns in Static Analysis Alerts”.

Idea: rank importance of FindBugs alerts by extracting a feature vector & using ML.



# Classification via Deep Learning

ICSE 23: Steenhoek, Rahman, Jiles, and Le. “An Empirical Study of Deep Learning Models for Vulnerability Detection.”

## Classification question:

## does this code have a vulnerability or not?

This work studies the behaviour of 9 deep learning models on 2 datasets.

Their conclusion: DL outperforms static analysis.

# Regression?

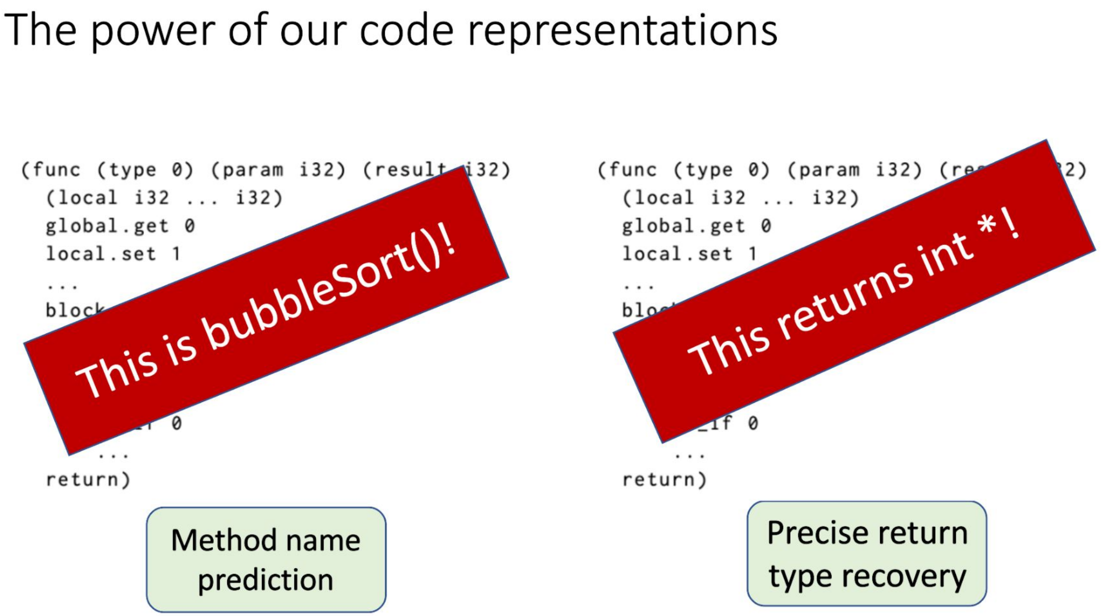
Instead of labels (classification), output numerical values within a range.

Not as amenable to PL/SE applications?

# Inference

# Code Representations for Improved Program Analysis

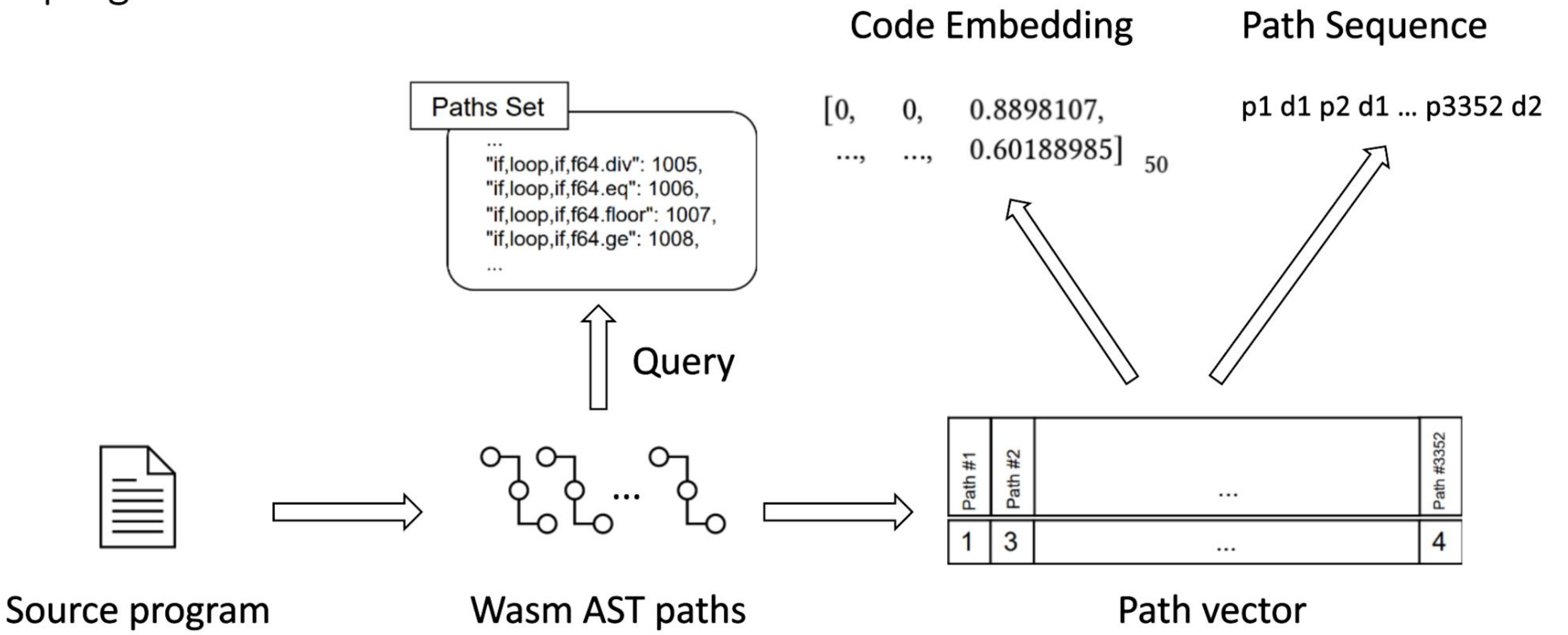
uploaded to arXiv; by Shirzad and Lam:



# How We Compute Our Code Representations

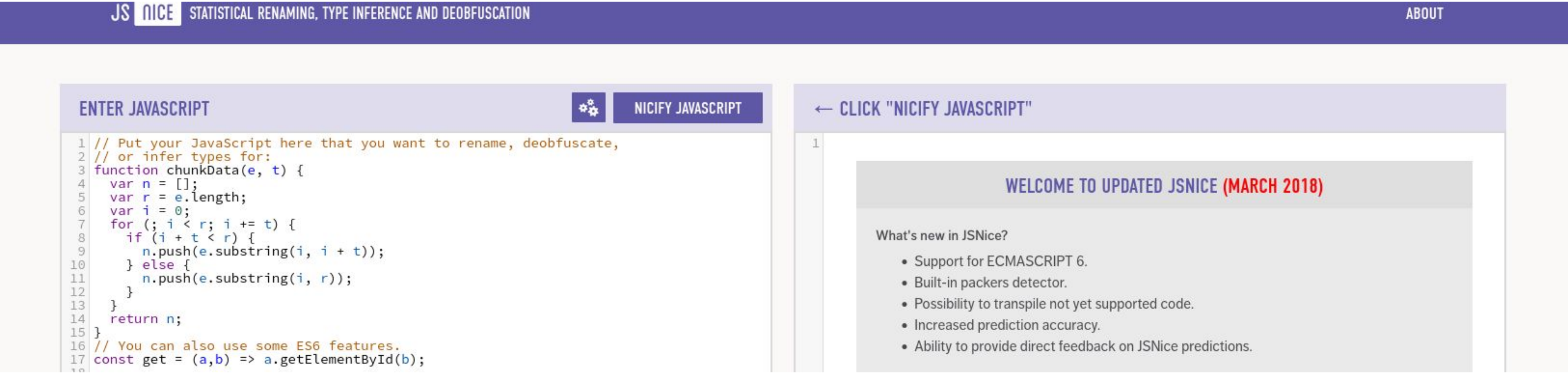
C program ---> Path vector

method name prediction,
precise return type recovery



# Another Application: JSNice–deobfuscating JavaScript

POPL 15. Raychev, Vechev & Krause. “Predicting Program Properties from “Big Code”.”




Infers likely identifier names and types via machine learning.

# Other applications of (broadly) inference

- Test generation
- Program repair
- Program synthesis

(Typically not machine learning techniques).

# Test Generation

Dan, Lam, Hoefler, and Vechev. OOPSLA 16. Modeling and Analysis of Remote Memory Access Programming.

```
X = 1                Y = 0
P0:                  P1:
a = X                put(X⁰, Y)
                     Y = get(X⁰)
                     flush(0)
                     b = Y
```

Expected outputs: $\langle a, b\rangle \in \{\langle 0, 0\rangle, \langle 1, 2\rangle\}$.

# Program Repair & Synthesis

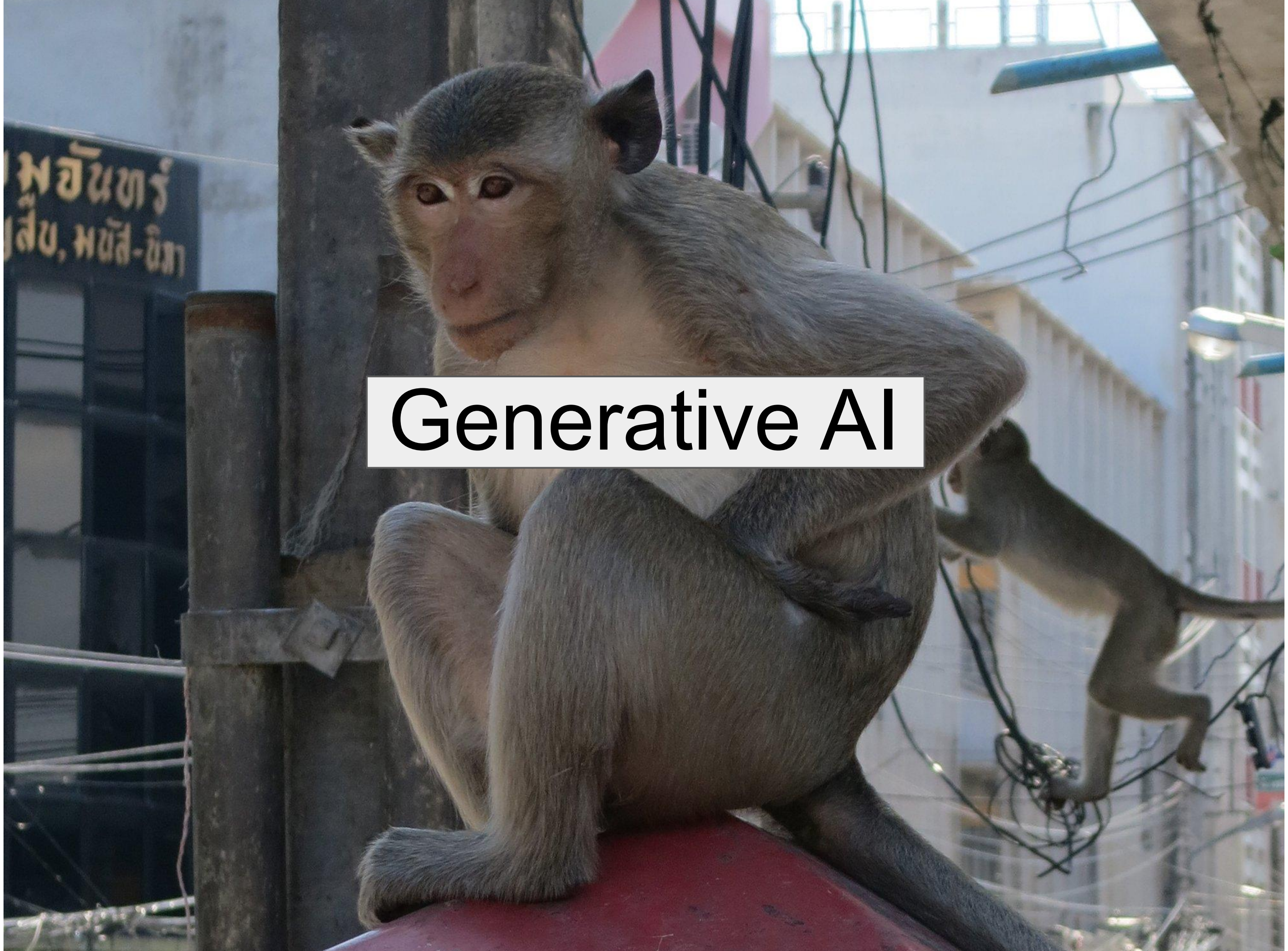
Per Armando Solar-Lezema:

- Program Synthesis corresponds to a class of techniques that are able to generate a program from a collection of artifacts that establish semantic and syntactic requirements for the generated code.

Usually, search for a suitable program.

```
in: [1,2,3,4,5,6,7,8]
out: [8,7,6,5,4,3,2,1]
```

[https://people.csail.mit.edu/asolar/SynthesisCourse/Lecture1.htm]


Generative AI

# Inference ++?

GitHub Copilot

ChatGPT

# Is generated code any good?

## #1 Let's do a user study!

Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. "Do Users Write More Insecure Code with AI Assistants?" CCS 23

Method: Recruited 47 participants, gave them Python/JS/C coding tasks
33/47 had Codex-based tool, 14/47 did not.

Answer: Codex-assisted code is more insecure,
but coders are more confident in the code!

# Is generated code any good?

## #2: Let’s verify using tests!

MSR 22. Nguyen & Nadi. “An empirical evaluation of GitHub copilot's code suggestions.”

> Asked Copilot to generate code for LeetCode problems,
> checked it with LeetCode test cases.
> Java 57% pass, JavaScript 27% pass.

# Is generated code any good?

## #3: Let’s use formal verification!

HATRA 22. Wong, Kothig, and Lam. “Exploring the Verifiability of Code Generated by GitHub Copilot.”

Attempt to formally verify Copilot generated code; 4/6 success.

But, when we don’t succeed, who’s the problem?

# Is generated code any good?

## #4: Let’s generate code interactively!

Kani Rust Verifier Blog. “Writing Code with ChatGPT? Improve it with Kani.”

Use model checking plus iterative applications to ChatGPT until it gets it right.

# My take on LLM-generated code (and LLMs in general)

1. Perhaps: experienced people can use LLMs to generate code more quickly than without.

   They do not have to trust the output;
   they can use it as a starting point & interact.

2. Novices **absolutely should not** use LLMs to generate code, because they can't interrogate the result.

# Deductive Reasoning

Back in the day…

ACL2, anyone? SMT solvers?

Old-school AI was search, not statistics.

Statistical approaches and reasoning…

???

I can’t really imagine how to do e.g. pointer analysis with statistical approaches.

# Pointer analysis

```
int * p, * q;
```

Question: do `*p` and `*q` possibly alias? That is,

```
    *p = 5;
    *q = 2;
```

What is `*p` going to contain?

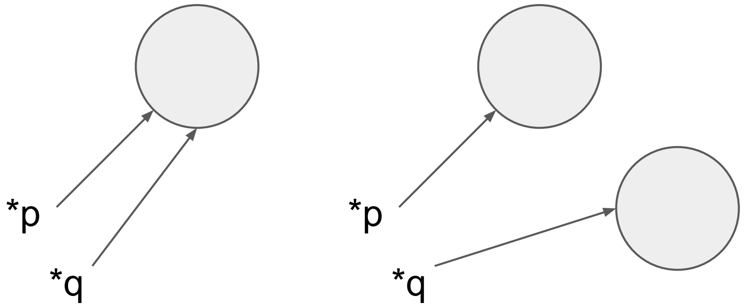
# Pointer analysis

```
int * p, * q;
```

Question: do `*p` and `*q` possibly alias? That is,

```
*p = 5;
*q = 2;
```

What is `*p` going to contain?



# CFL-Reachability

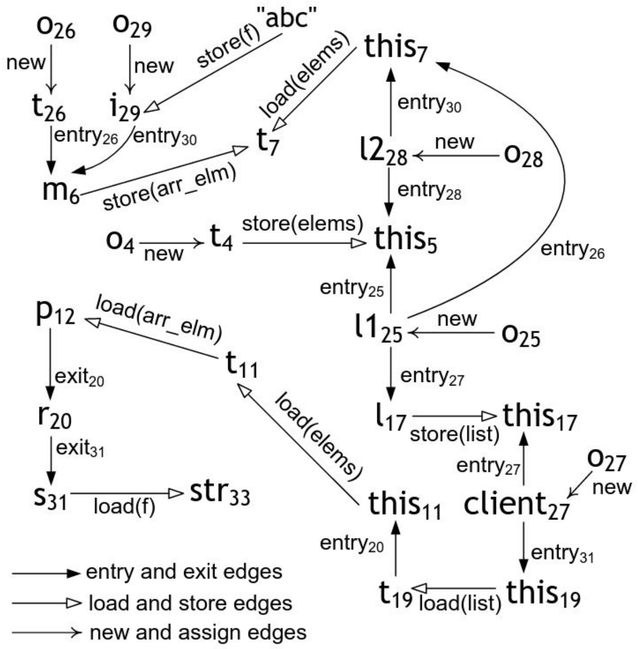



**Fig. 2.** Illustration of CFL-reachability

Xu, Rountev & Sridharan. ECOOP 09. Scaling CFL-Reachability-Based Points-To Analysis Using Context-Sensitive Must-Not-Alias Analysis.

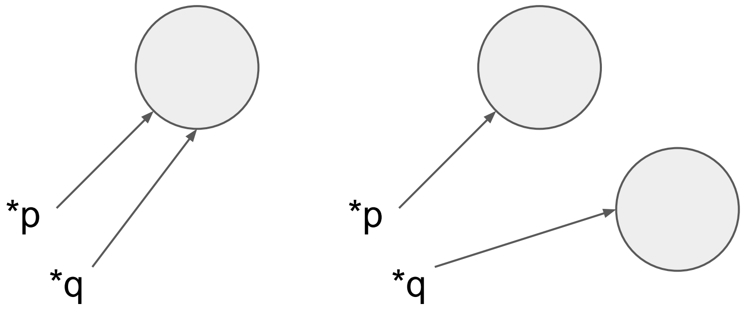
# Pointer analysis

```
int * p, * q;
```

Question: do `*p` and `*q` possibly alias? That is,

```
    *p = 5;
    *q = 2;
```

What is `*p` going to contain?




# Do you want reasoning, or do you want reckons?

# Counterpoint: Biology/Protein Structure Prediction

I am not an expert here!

I’ve read that large language models can help rule out protein structures that are unlikely to work, allowing scientists to focus on viable structures.

But again, it is possible to objectively check potential results;
perhaps this works because the space contains many winning results.

Is that insight transferable to program analysis? Synthesis?

# Hot Takes

# What does ChatGPT think? [part 1]

ChatGPT

`patricklam.ca/chatgpt-hot-takes`

[prompt engineering by Lucas Wojciechowski]

> Give me 10 hot takes on how machine learning will affect programming language / software engineering in the next 10 years

```
Developers will need to have a strong understanding of statistics
and data analysis in order to effectively incorporate machine
learning into their applications.

Natural language processing (NLP) and speech recognition will
become more prevalent, leading to the development of new
programming languages and tools that support these technologies.
```

# What does ChatGPT think? [part 2]

ChatGPT

`patricklam.ca/chatgpt-hot-takes`

[prompt engineering by Lucas Wojciechowski]

> Give me 10 hot takes on how machine learning will affect programming language / software engineering in the next 10 years

```
As machine learning becomes more widespread, developers will need
to prioritize the ethical implications of their work, such as data
privacy, bias, and accountability.

Developers will increasingly rely on machine learning to automate
software testing and debugging, reducing the time and effort
required to find and fix bugs.
```

# Hotter takes [part 1]

patricklam.ca/chatgpt-hot-takes

> Can you make the takes more controversial?

```
Machine learning will eventually make traditional programming
languages obsolete, as machines become better at creating software
than humans.

As machine learning becomes more advanced, developers will need to
focus on building systems that are transparent and explainable, in
order to ensure that they can be trusted by users.
```

# Hotter takes [part 2]

ChatGPT

patricklam.ca/chatgpt-hot-takes

> Can you make the takes more controversial?

```
Machine learning will eventually lead to the development of
conscious machines, creating a new era in which machines and
humans coexist as equals.

As machine learning becomes more prevalent, it will lead to the
creation of new types of software applications that are designed
to manipulate and exploit human emotions.
```

# Hottest takes [part 1]

ChatGPT

patricklam.ca/chatgpt-hot-takes

> Can you make the takes even more controversial?

The rise of machine learning will lead to the creation of new types of software applications that are designed to replace human workers, leading to widespread job displacement and social unrest.

The development of advanced machine learning algorithms will eventually lead to the creation of true artificial intelligence, capable of independent thought and decision making.

# Hottest takes [part 2]

ChatGPT

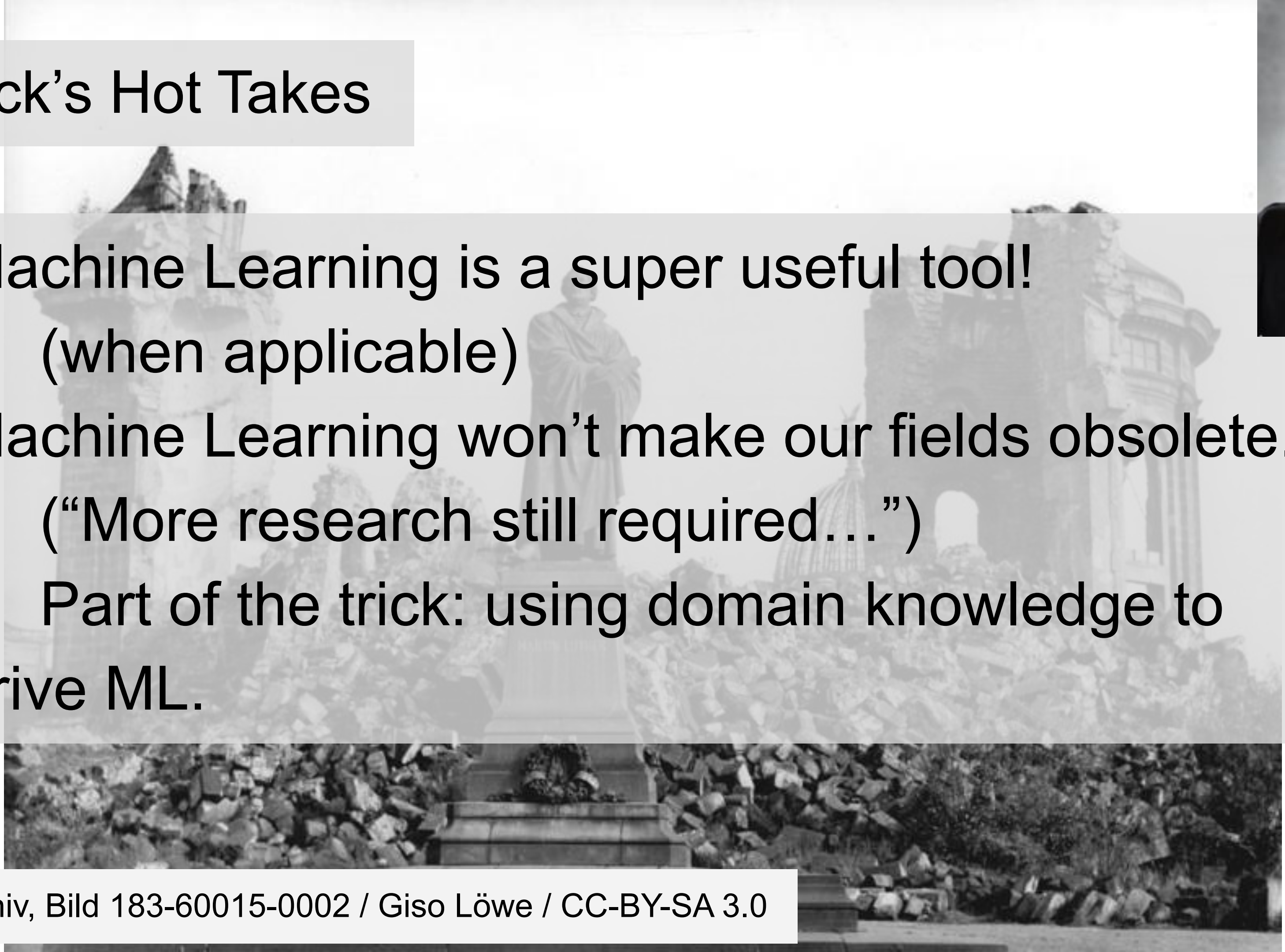
patricklam.ca/chatgpt-hot-takes

> Can you make the takes even more controversial?

The increasing use of machine learning in software development will lead to a new era of digital imperialism, with the most advanced machine learning algorithms being used to dominate and control global markets.

Eventually, the rise of machine learning will lead to a new era of technological singularity, in which machines become more intelligent than humans and take over the world.

# Patrick’s Hot Takes



- Machine Learning is a super useful tool!
  (when applicable)
- Machine Learning won’t make our fields obsolete.
  (“More research still required…”)
  Part of the trick: using domain knowledge to drive ML.

Why we’re not about to become obsolete:
a report from Computational Linguistics

e.g. Google Translate:
isn’t it all statistical?

# Computational Linguistics: not just statistics

Blackbox NLP workshop, EMNLP 2022. Muthupari, Halder, Sayeed, and Marton. “Where's the Learning in Representation Learning for Compositional Semantics and the Case of Thematic Fit”.

Question: understand why sometimes random word embeddings work as well as pretrained embeddings.

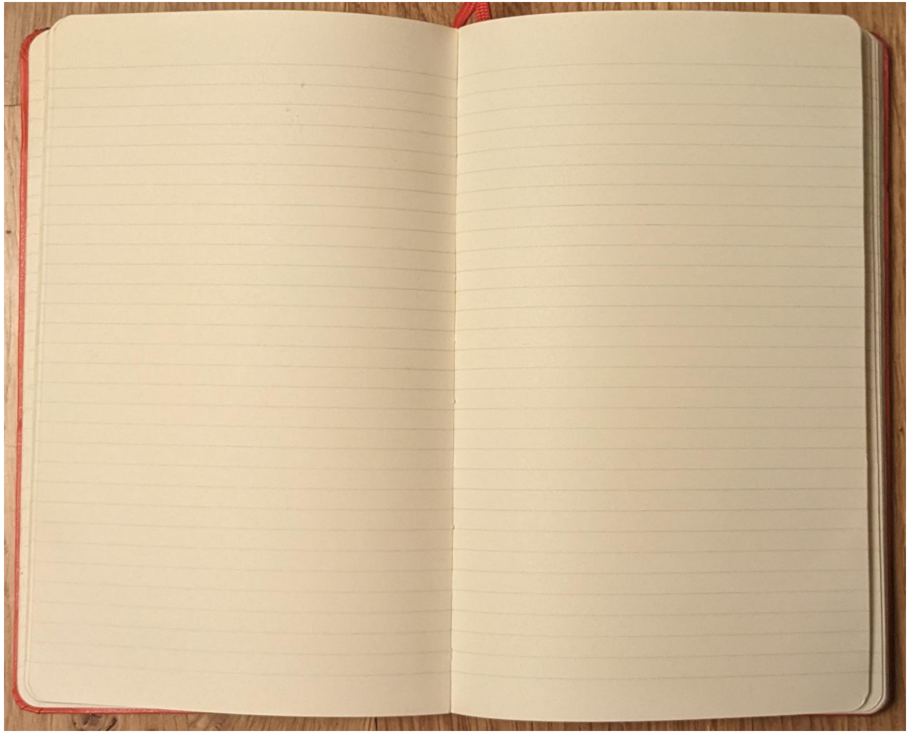
Tools: Need experimental efforts in linguistic representations; manipulate the model architecture.

# Patrick’s Hot Takes: Generative AI

> Give me 10 hot takes on how machine learning will affect programming language / software engineering in the next 10 years

- `“Generated code can be good actually;” but,`
- `“I wouldn’t trust generated code further than I can throw it.”`

# What Generative AI is Good At



# What Generative AI is Good At

# What Generative AI is bad at: novelty

# Other things generative AI is bad at

- Identifying the problem
- Hallucinations(*) / Objective truth
- Respecting IP
- Ensuring security

*: people say it’s gotten better?

# Identifying the problem

choosing a tool is the easy part…

# Objective Truth

What is the highest waterfall in New Zealand?

# Respecting IP

# Ensuring Security

# Food for thought

How does ChatGPT/Copilot code compare to hackathon code?

# Unrelated future work bonus content: Rust

I’m looking for thoughtful discussions on possible future research direction.

- Verifying the Rust standard library
  - also, client code needs to ensure safety properties; check them
- Unsafe code:
  - race detection on the unsafe code
  - unsafe code that calls foreign functions: verify the foreign code
- Static analysis & implications of ownership on pointer analysis

# Brief summary of Rust unsafe

- not a free-for-all
- enables 5 specific “superpowers” (eg dereference a raw pointer)

These superpowers enable the code to violate eg ownership constraints.

It becomes the programmer’s responsibility to ensure memory safety.

Expectation (linted): for each unsafe block, a safety comment says:

(1) why code is safe; or,

(2) required conditions for the code to be safe.

# Rust standard library verification

Amazon is verifying the Rust standard library & inviting participation.
Tools: Bounded model checking (kani), separation logic (VeriFast), etc.

“Verifying the standard library” means:

(1) writing the necessary contracts for stdlib functions
(2) assuming contracts, ensuring the absence of undefined behaviour

Additional challenge: contracts may need properties beyond those expressible in the type system; how can we establish & propagate these?

# Adventures with unsafe Rust

Race conditions:

- Safe Rust is free of race conditions:
    - must either have a lock or ownership to access memory.
- Unsafe Rust can have races (resulting in undefined behaviour)
  Can we port existing static race detection techniques to Unsafe Rust?

Foreign Function Interfaces (ffi):

- ffi calls must originate from Unsafe Rust;
- can we verify C code called from Rust and establish safety?
  (FFIChecker does this to some extent; we can do better).

# Static Analysis for Rust

The Rust type system imposes strong requirements on Rust code;
the Rust linter adds checks (eg safety properties have comments).

But, there isn't much static analysis for Rust out there. There is:

- pointer analysis for Rust (RUPTA, CC 24)
- MirChecker does some abstract interpretation for overflows & "lifetime corruption" (unsafe code leaves broken references around).

Compiler optimizations leveraging uniqueness?
Deeper correctness properties?

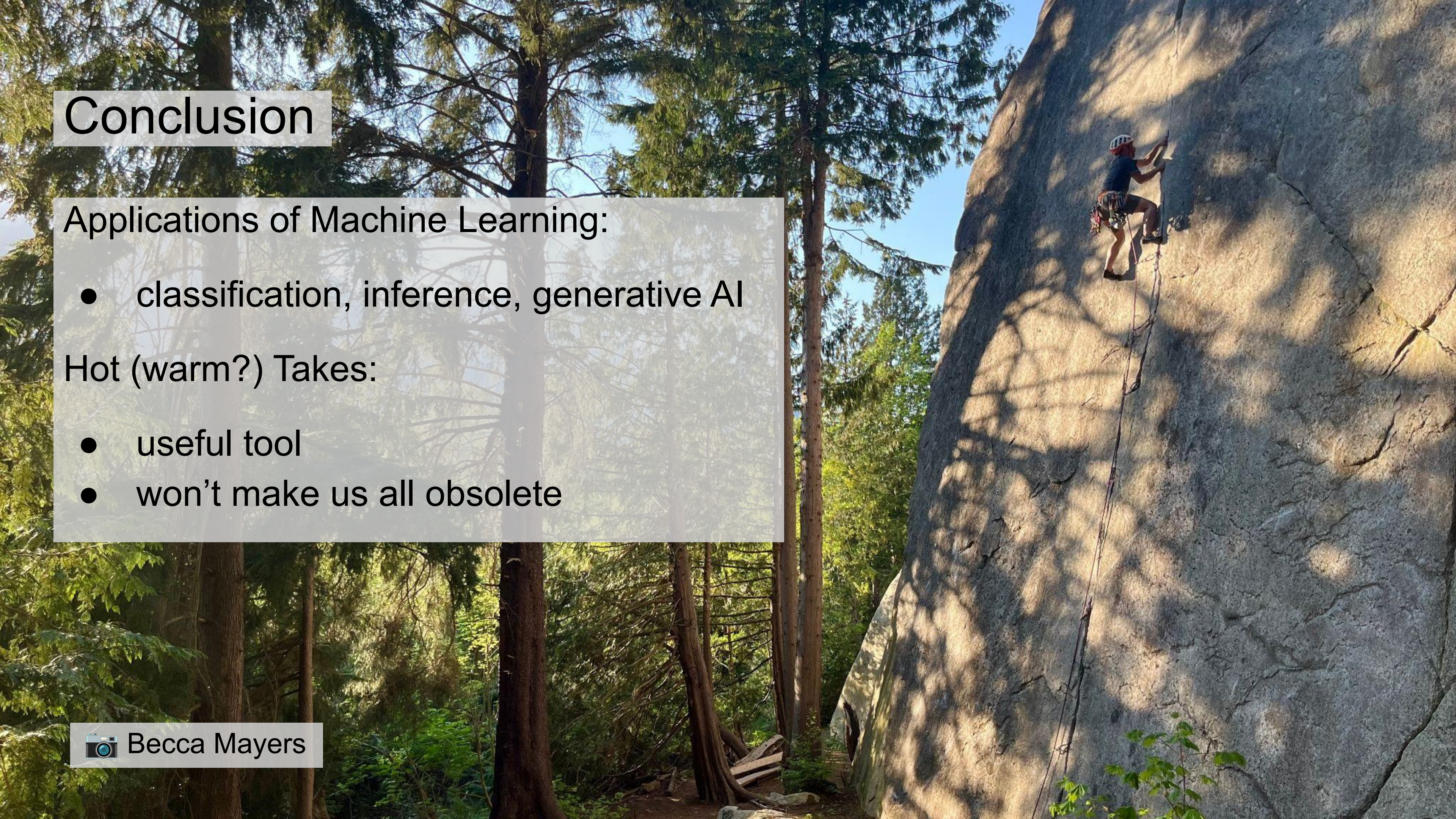
# Conclusion

Applications of Machine Learning:

- classification, inference, generative AI

Hot (warm?) Takes:

- useful tool
- won’t make us all obsolete



📷 Becca Mayers