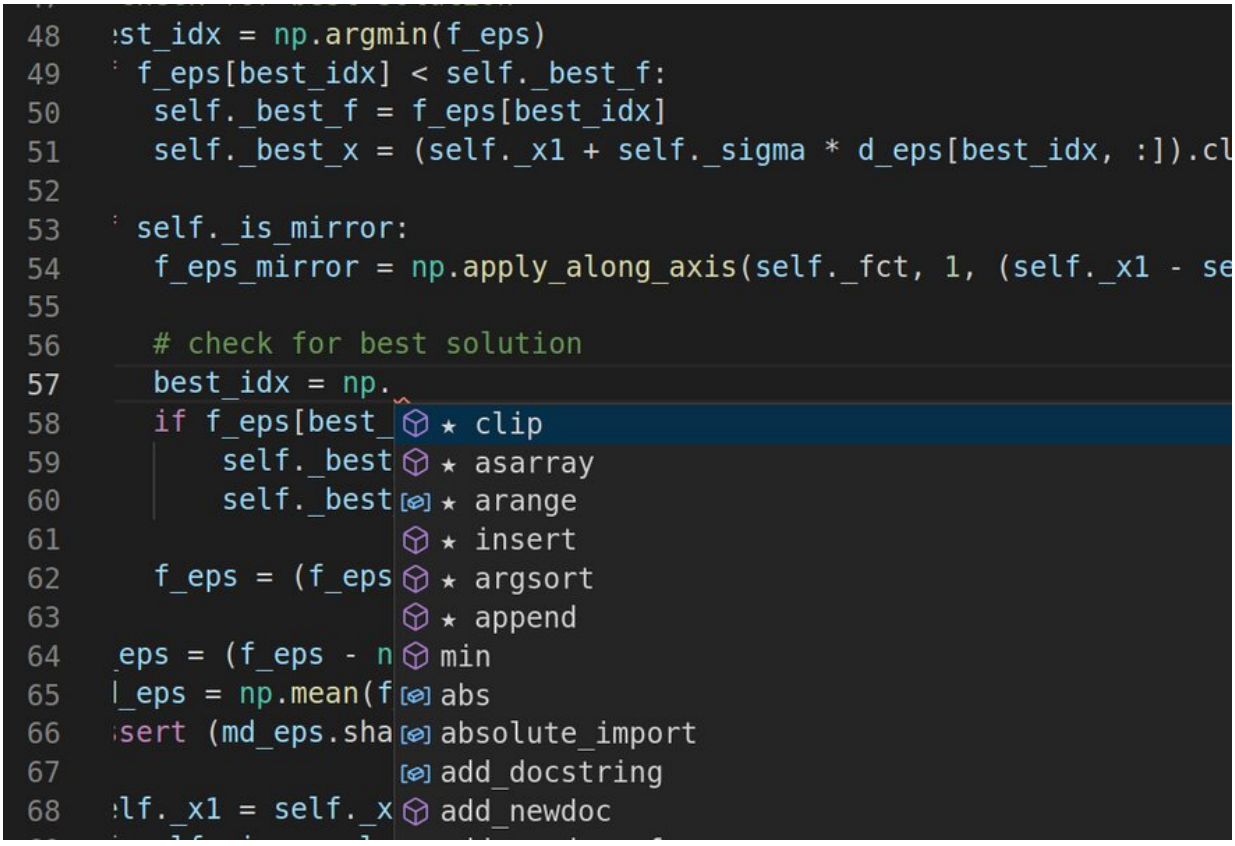
Toward deep-learning models that can reason about code more like humans

April 16 2021, by Kim Martineau



A framework built by MIT and IBM researchers finds and fixes weaknesses in automated programming tools that leave them open to attack. One tool (pictured) reads along as programmers write and suggests code. Here, it picks a function among thousands of options in Python's NumPy library that best suits the task at hand. Credit: Shashank Srikant

Whatever business a company may be in, software plays an increasingly vital role, from managing inventory to interfacing with customers. Software developers, as a result, are in greater demand than ever, and that's driving the push to automate some of the easier tasks that take up their time.

Productivity tools like Eclipse and Visual Studio suggest snippets of code that developers can easily drop into their work as they write. These automated features are powered by sophisticated language models that have learned to read and write computer code after absorbing thousands of examples. But like other deep learning models trained on big datasets without explicit instructions, language models designed for code-processing have baked-in vulnerabilities.

"Unless you're really careful, a hacker can subtly manipulate inputs to these models to make them predict anything," says Shashank Srikant, a graduate student in MIT's Department of Electrical Engineering and Computer Science. "We're trying to study and prevent that."

In a new paper, Srikant and the MIT-IBM Watson AI Lab unveil an automated method for finding weaknesses in code-processing models, and retraining them to be more resilient against attacks. It's part of a broader effort by MIT researcher Una-May O'Reilly and IBM-affiliated researcher Sijia Liu to harness AI to make automated programming tools smarter and more secure. The team will present its results next month at the International Conference on Learning Representations.

A machine capable of programming itself once seemed like science fiction. But an exponential rise in computing power, advances in natural language processing, and a glut of free code on the internet have made it possible to automate at least some aspects of software design.

Trained on GitHub and other program-sharing websites, code-processing

models learn to generate programs just as other language models learn to write news stories or poetry. This allows them to act as a smart assistant, predicting what software developers will do next, and offering an assist. They might suggest programs that fit the task at hand, or generate program summaries to document how the software works. Code-processing models can also be trained to find and fix bugs. But despite their potential to boost productivity and improve software quality, they pose security risks that researchers are just starting to uncover.

Srikant and his colleagues have found that code-processing models can be deceived simply by renaming a variable, inserting a bogus print statement, or introducing other cosmetic operations into programs the model tries to process. These subtly altered programs function normally, but dupe the model into processing them incorrectly, rendering the wrong decision.

The mistakes can have serious consequences for code-processing models of all types. A malware-detection model might be tricked into mistaking a malicious program for benign. A code-completion model might be duped into offering wrong or malicious suggestions. In both cases, viruses may sneak by the unsuspecting programmer. A similar problem plagues computer vision models: Edit a few key pixels in an input image and the model can confuse pigs for planes, and turtles for rifles, as other MIT research has shown.

Like the best language models, code-processing models have one crucial flaw: They're experts on the statistical relationships among words and phrases, but only vaguely grasp their true meaning. OpenAI's GPT-3 language model, for example, can write prose that veers from eloquent to nonsensical, but only a human reader can tell the difference.

Code-processing models are no different. "If they're really learning intrinsic properties of the program, then it should be hard to fool them,"

says Srikant. "But they're not. They're currently relatively easy to deceive."

In the paper, the researchers propose a framework for automatically altering programs to expose weak points in the models processing them. It solves a two-part optimization problem; an algorithm identifies sites in a program where adding or replacing text causes the model to make the biggest errors. It also identifies what kinds of edits pose the greatest threat.

What the framework reveals, the researchers say, is just how brittle some models are. Their text summarization model failed a third of the time when a single edit was made to a program; it failed more than half of the time when five edits were made, they report. On the flip side, they show that the model is able to learn from its mistakes, and in the process potentially gain a deeper understanding of programming.

"Our framework for attacking the model, and retraining it on those particular exploits, could potentially help code-processing models get a better grasp of the program's intent," says Liu, co-senior author of the study. "That's an exciting direction waiting to be explored."

In the background, a larger question remains: what exactly are these black-box deep-learning models learning? "Do they reason about code the way humans do, and if not, how can we make them?" says O'Reilly. "That's the grand challenge ahead for us."

More information: Generating Adversarial Computer Programs using Optimized Obfuscations. openreview.net/forum?id=PH5PH9ZO_4

research, innovation and teaching.

Provided by Massachusetts Institute of Technology