Fuzzing LLMs
A framework for discovering edge cases
What is fuzzing & why is it relevant for AI safety?

- On a high level, **fuzzing** is repeatedly running a program with generated inputs that may be syntactically or semantically malformed
  - Execution of program using input(s) sampled from an input space that *protrudes* the expected input space of the program
- **White-box fuzzing (Godefroid, 2007)**
  - Use the internals of the program to generate fuzzing examples
  - Often slow but quite interesting in neural network relations
- **Black-box fuzzing (IO-driven / data-driven testing)**
- **Apply fuzzing to LLMs** *and use LLMs to generate the examples*
- **AI safety**
  - Unexpected behaviors of language models are important to find due to the inherent security risks in both exploit vulnerabilities and general misbehaviors during deployment
  - An example is Rumbelow & Watkins (2023) who show that tokens can be semantically misinterpreted by a network due to the structure of the training data in the long tail of token probabilities
Existing work: Fuzzing using LLMs

Deng et al. (2022/23) use LLMs to generate code examples to test deep learning libraries for edge case bugs.
Existing work: Automated red teaming

Perez et al. (2022) use LLMs to generate adversarial texts designed to elicit harmful responses, as a way to catch edge cases.

Figure 1: Overview: We automatically generate test cases with a language model (LM), reply with the target LM, and find failing test cases using a classifier.
What did we do?

- We use the new RWKV architecture from EleutherAI (2023) at ~540M parameters (the largest is 14B)
- We have multiple prompt histories:
  - H1 is composed of the P1 input and any output from M1 to P1. M1_out is the M1 output isolated.
  - H2 is composed of M1_out and M2's output to P1_out, M2_out.
  - H3 is composed of P3 formatted with M1_out and M2_out along with M3's output given this formatted P3. H3 is limited to a single digit numerical output.
- The resulting framework presents the first steps towards automated fuzzing
- We did not get to a state where results were possible due to my computer running out of power in the airport
What did we find?

- The 540M parameter RWKV is not at all capable enough to execute or evaluate fuzzing.
- The P1 prompt is quite important and will probably benefit from a chatbot RLHF step.
- The 100-line implementation of RWKV we used might be too simplistic for this specific scenario.
- Next steps:
  - Do the steps manually using three ChatGPT windows.
  - Run it using the GPT-4 API and see if the outputs make sense.
  - Save the activations of the network while running each fuzzing attempt.
  - See more future steps in the repository at [https://github.com/esbenkc/verification-jam](https://github.com/esbenkc/verification-jam).
Framework of the prototype

P1 instruction for creating an adversarial prompt

M1 adversarial model

M2 target model

M3 annotator model

Dataset

Prompt + Output
Surprisal (Output \mid Prompt)
Activation information

P3 instruction to classify output surprisal from 1 to 5