NIGHTWING

BREAKING THE CLASSIFIER

Adversarial Machine Learning

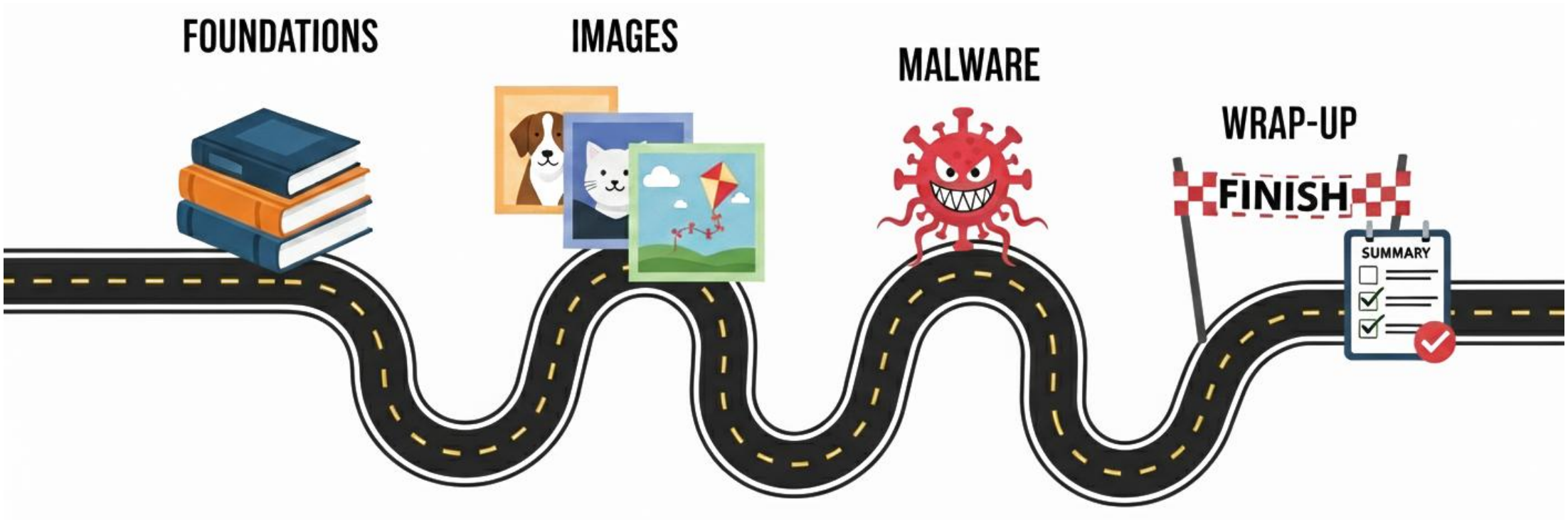
ABOUT THE AUTHORS…

- Japheth Light
  - Nightwing
- ChatGPT 5 Pro
  - OpenAI
- Gemini 2.5 Pro (Ultra?)
  - Google

Training

- *All graphics, code, and (some) slides in this workshop were created by AI.**
  - **Hundreds and hundreds (thousands?) of prompts were required to produce this workshop.*

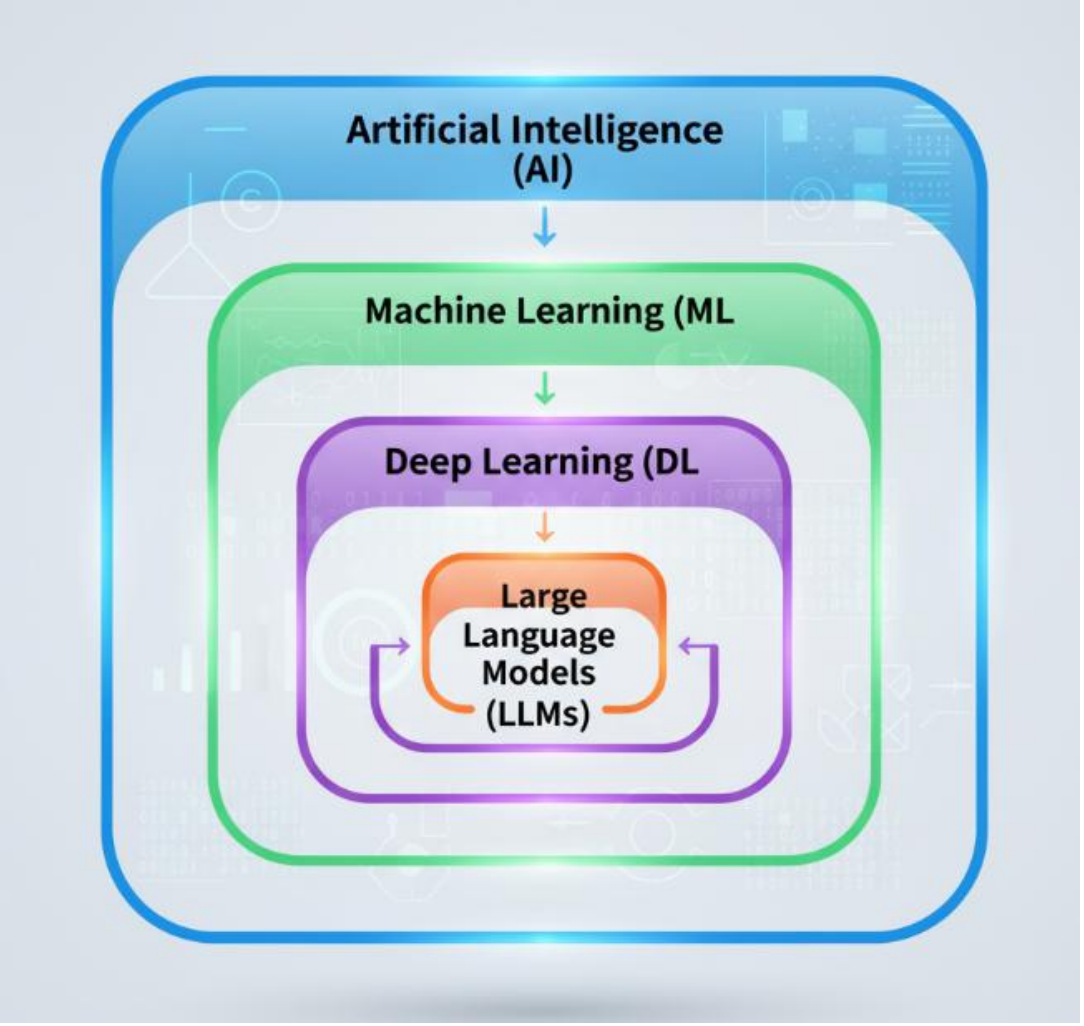
ROADMAP & OUTCOMES



- Goals: explain how ML classifiers work, describe gradient-based attacks, create adversarial examples for images and malware, have fun and learn.

WHAT'S THE MOTIVATION FOR THIS TALK?

- Coursework…
- Deep Learning was fascinating!
- Adversarial Images
- Wanted to learn more
  - The best way to learn is to teach!
  - Let's learn together
- Also wanted to test-drive today's LLMs on harder problems…



SECTION 1: FOUNDATIONS

- Why do this?
- How do ML-based classifiers work?

WHY STUDY ADVERSARIAL MACHINE LEARNING?

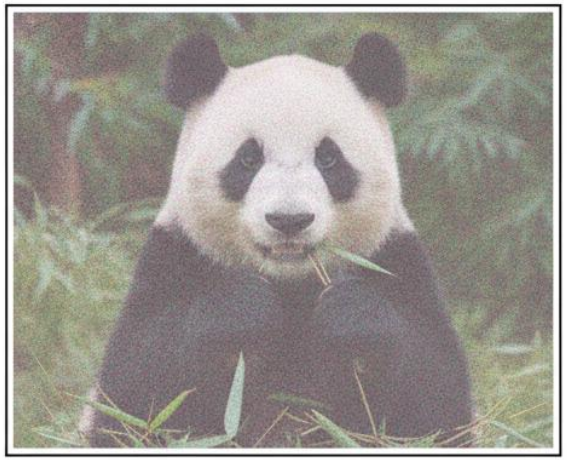
- Real systems make consequential decisions, and these systems are increasingly relying on AI.
- With adversarial machine learning, tiny, tailored changes can cause 'confident' mistakes.

Hackers can trick a Tesla into accelerating by 50 miles per hour

ACCURACY VS. ROBUSTNESS

- **Accuracy:** model performance on benign test data.
  - How well does the model generalize to data it hasn't seen before?
- **Robustness:** with a small, constrained change to an input, can we force a failure?

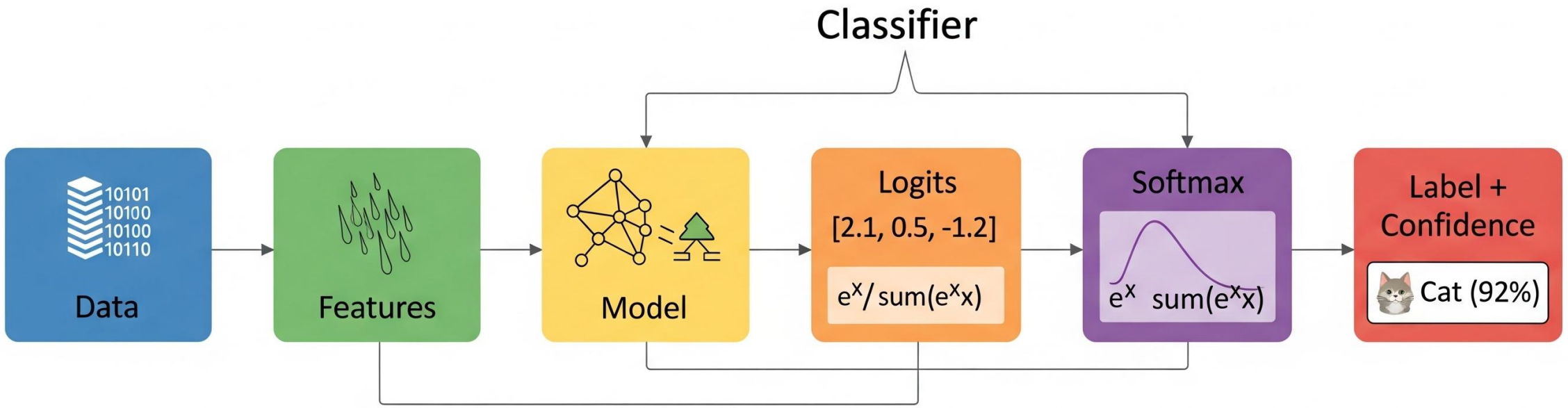
MODEL PREDICTION:
"PANDA"
(Confidence: 97%)



MODEL PREDICTION:
"OSTRICH"
(Confidence: 99%)

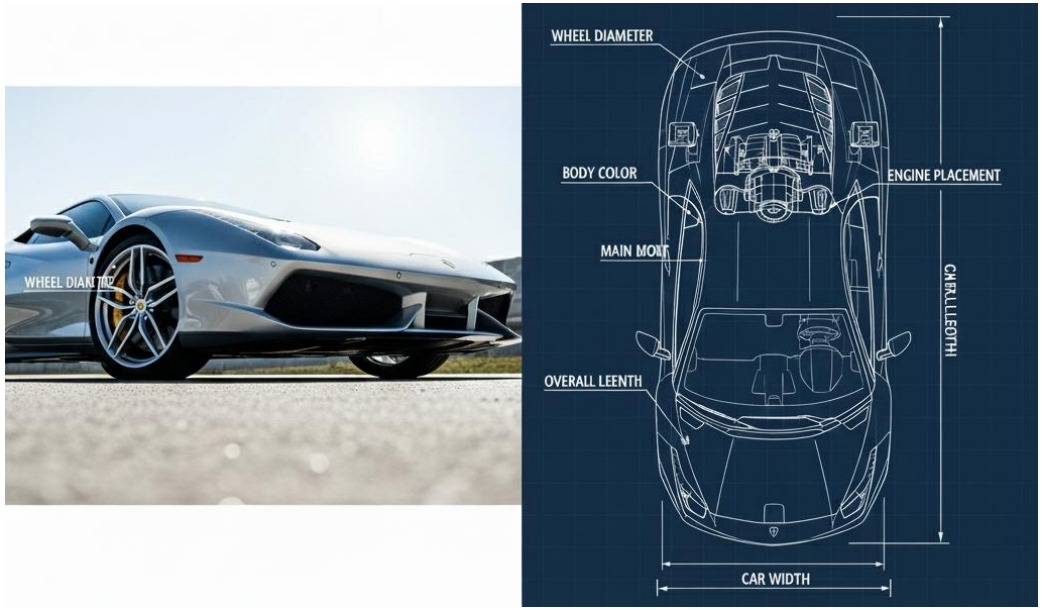
CLASSIFIER PIPELINE — HIGH-LEVEL OVERVIEW

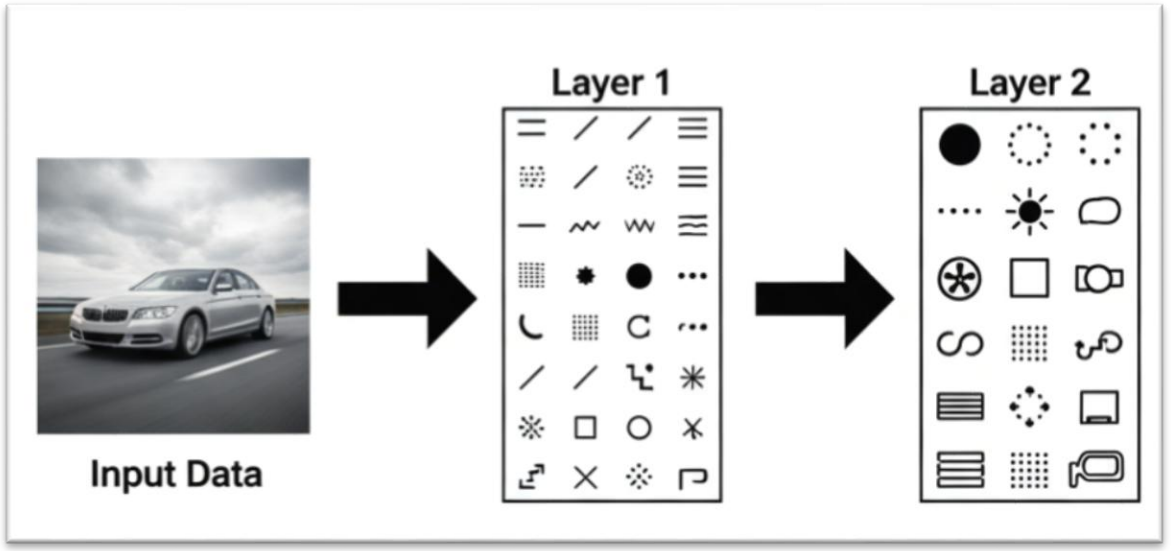
- **Data → features → model → logits → softmax → label + confidence**
- Training minimizes a loss; inference applies the trained model.



DATA & FEATURES

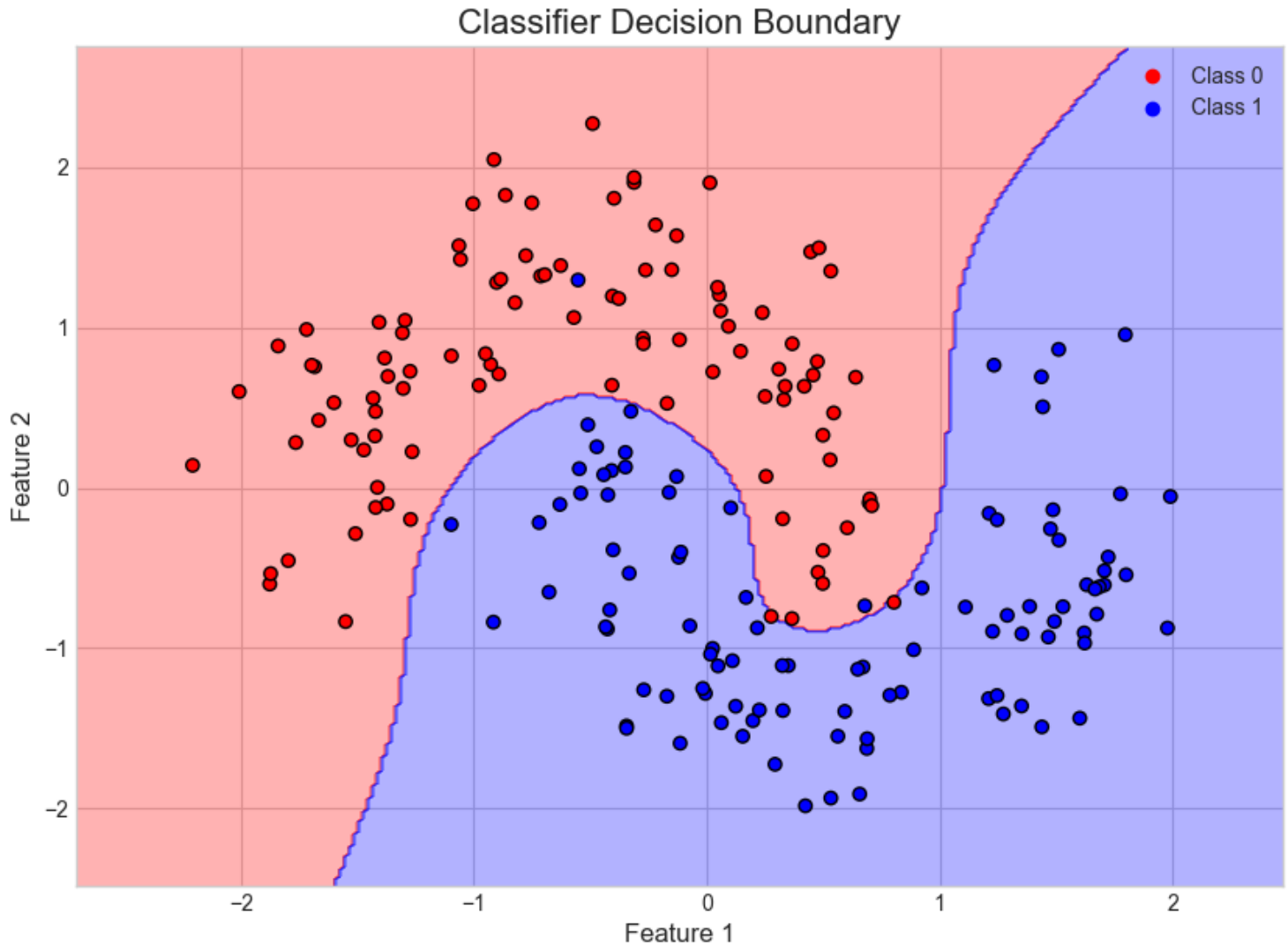
- Classical ML: features engineered (e.g., measurements, n-grams).
- Deep learning: features **learned** end-to-end from data.
- Quality, balance, and coverage of data is what shapes decision boundaries.





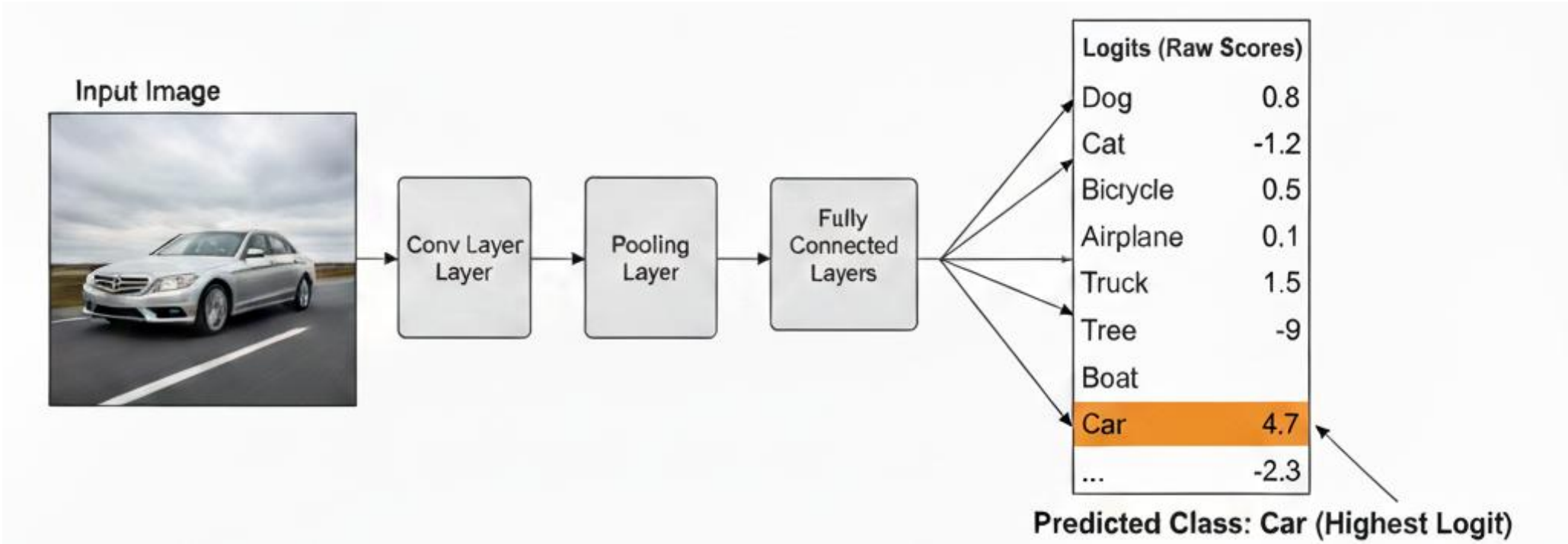
DECISION BOUNDARIES

- Boundary separates class regions in feature space.
- Points near the boundary are easier to flip.



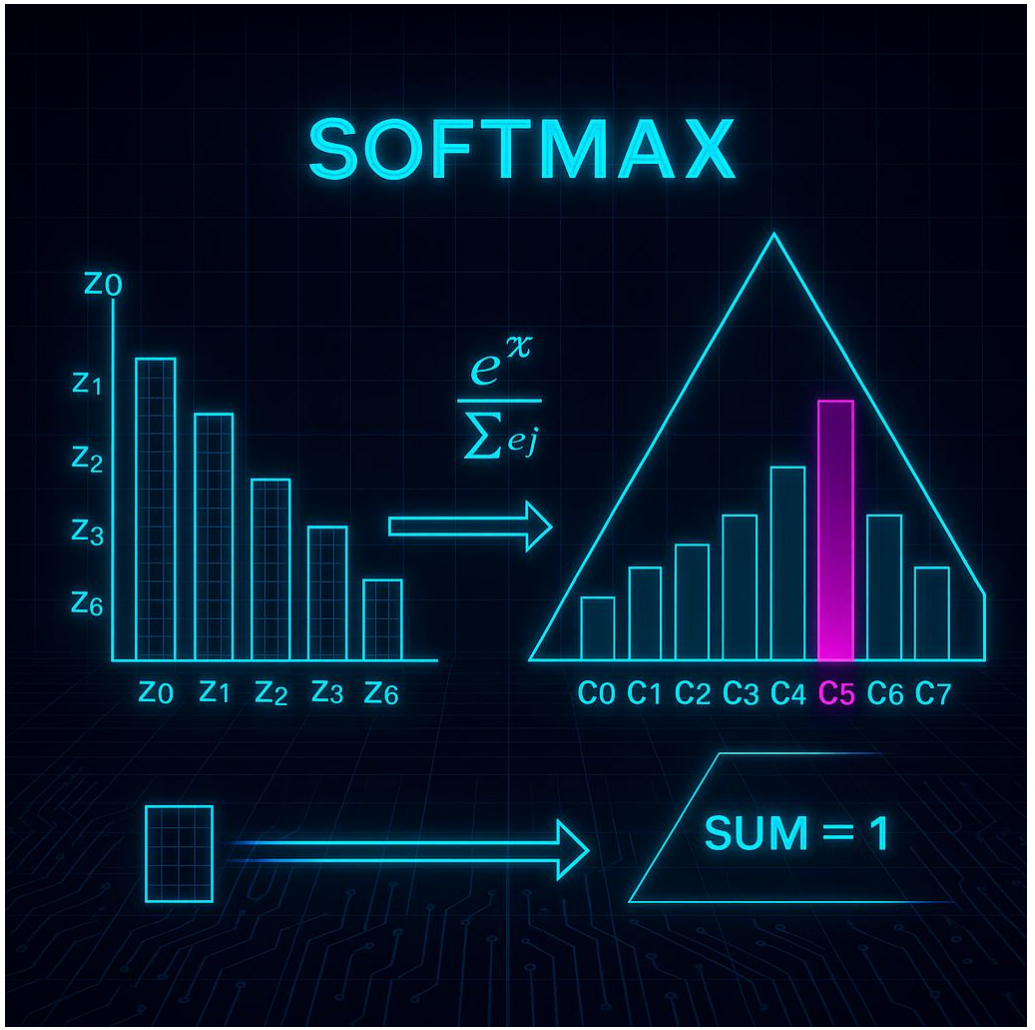
LOGITS — RAW SCORES BEFORE SOFTMAX

- Model outputs a score per class (logit).
- Relative differences between logits matter more than absolute values.



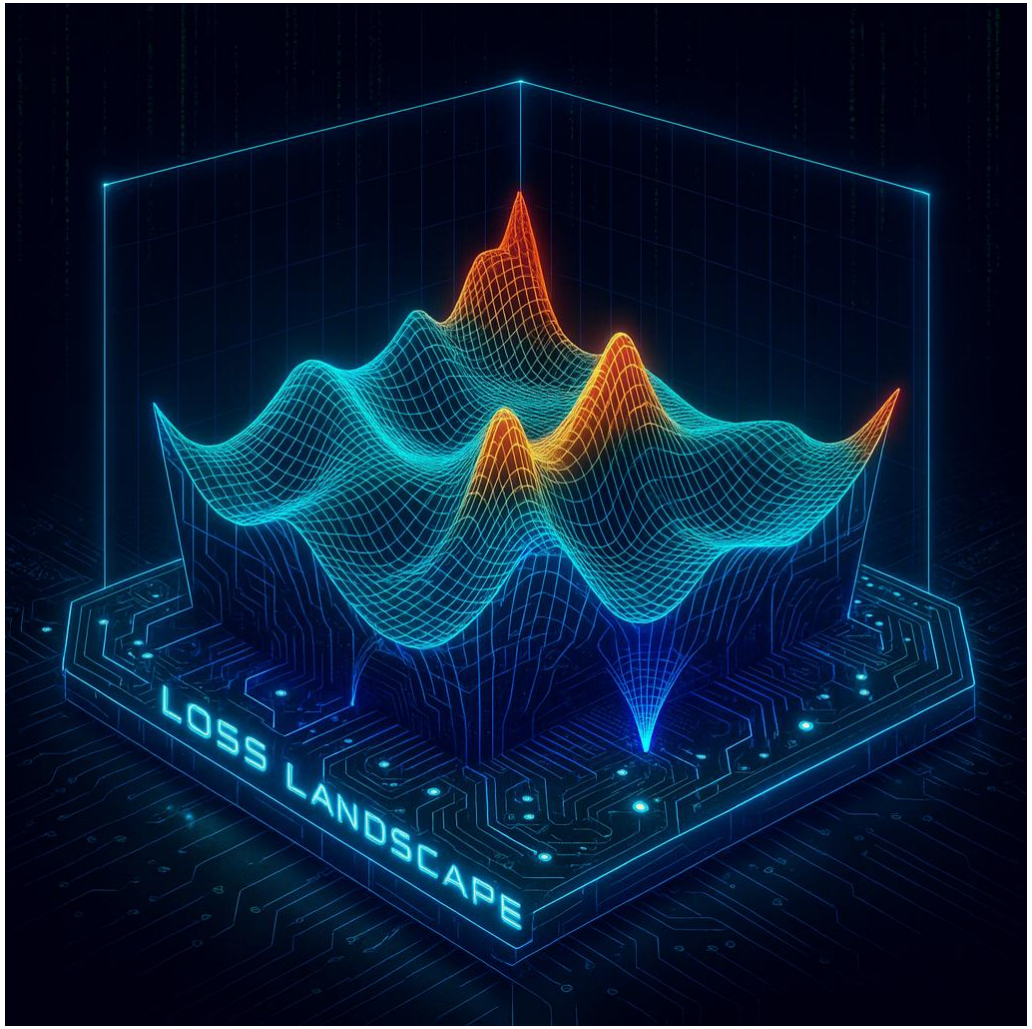
SOFTMAX & CONFIDENCE

- Softmax turns logits into a probability-like vector.
- **Argmax** picks the label; top value is the model's confidence.



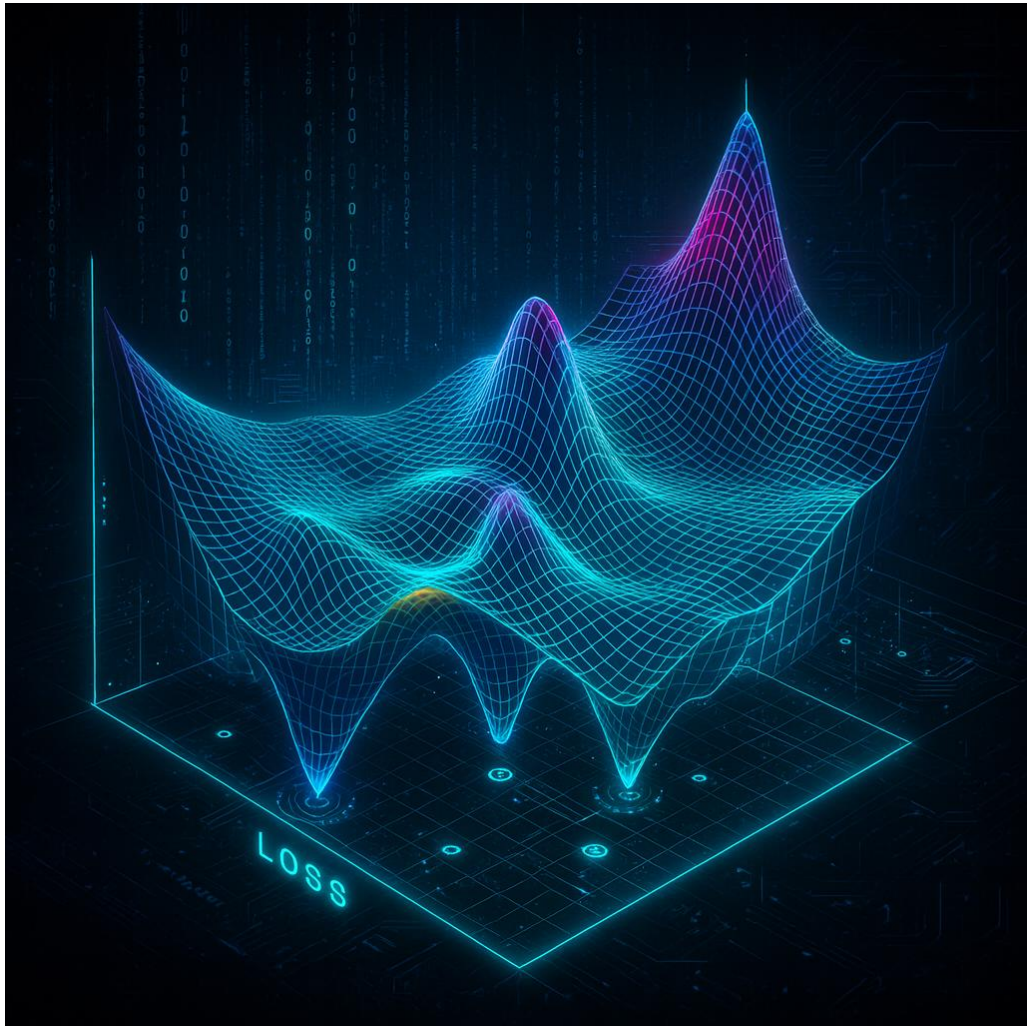
LOSS FUNCTIONS (CROSS-ENTROPY INTUITION)

- During the training of a machine learning model, the primary goal is to **minimize the error**, or "loss," between the model's predictions and the actual correct answers.
- Think of this loss as a valley. The objective is to find the lowest point in this valley, which represents the model with the least error.



GRADIENTS AS A STEERING WHEEL

- In an adversarial attack, the goal is the exact opposite: to **maximize the model's error** and cause it to make a wrong prediction
- Calculate the gradient of the loss with respect to the input features.
- Move "Uphill" on the Input
  - You then take a small step in the direction of that gradient. For an image, this means slightly increasing or decreasing the value of each pixel according to the gradient's instructions.

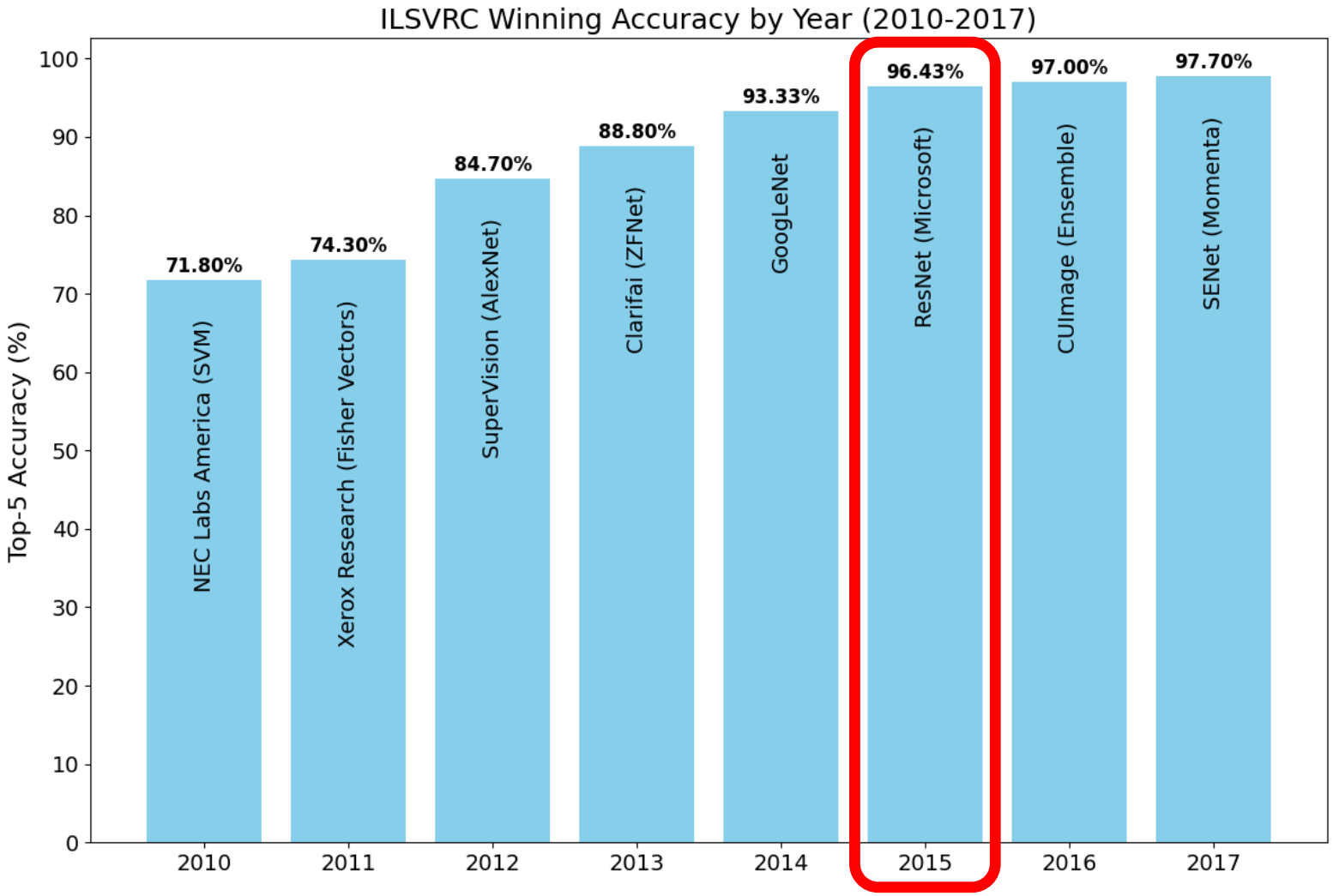


SECTION 2: AML WITH IMAGES

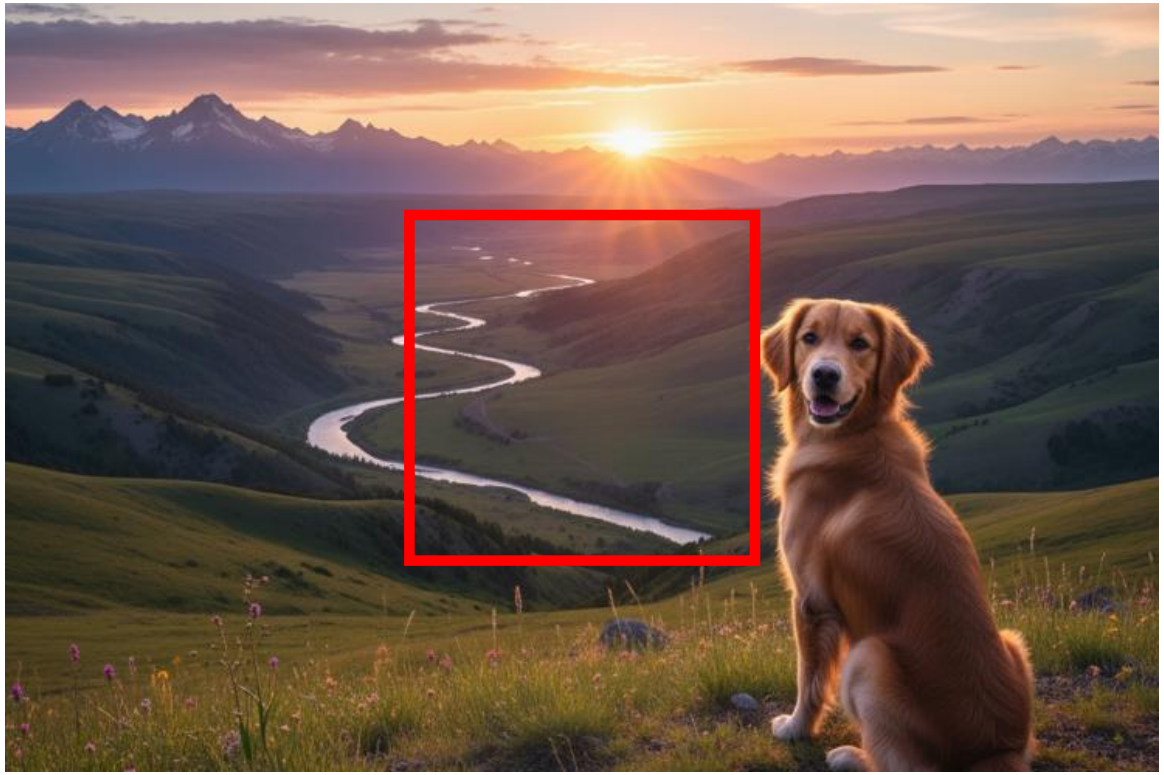
- Load a pre-trained image classifier
- Test it with a provided image
- Use “gradient ascent” to generate a fooling image
- Try it out with your own images
- (Optionally) solve additional challenges

IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE

- Competition based on the **ImageNet** dataset
- Task was to train a model that could correctly classify an image into one of the 1,000 categories.
- Approximately 1.2 million training images, 50,000 validation images, and 150,000 testing images.



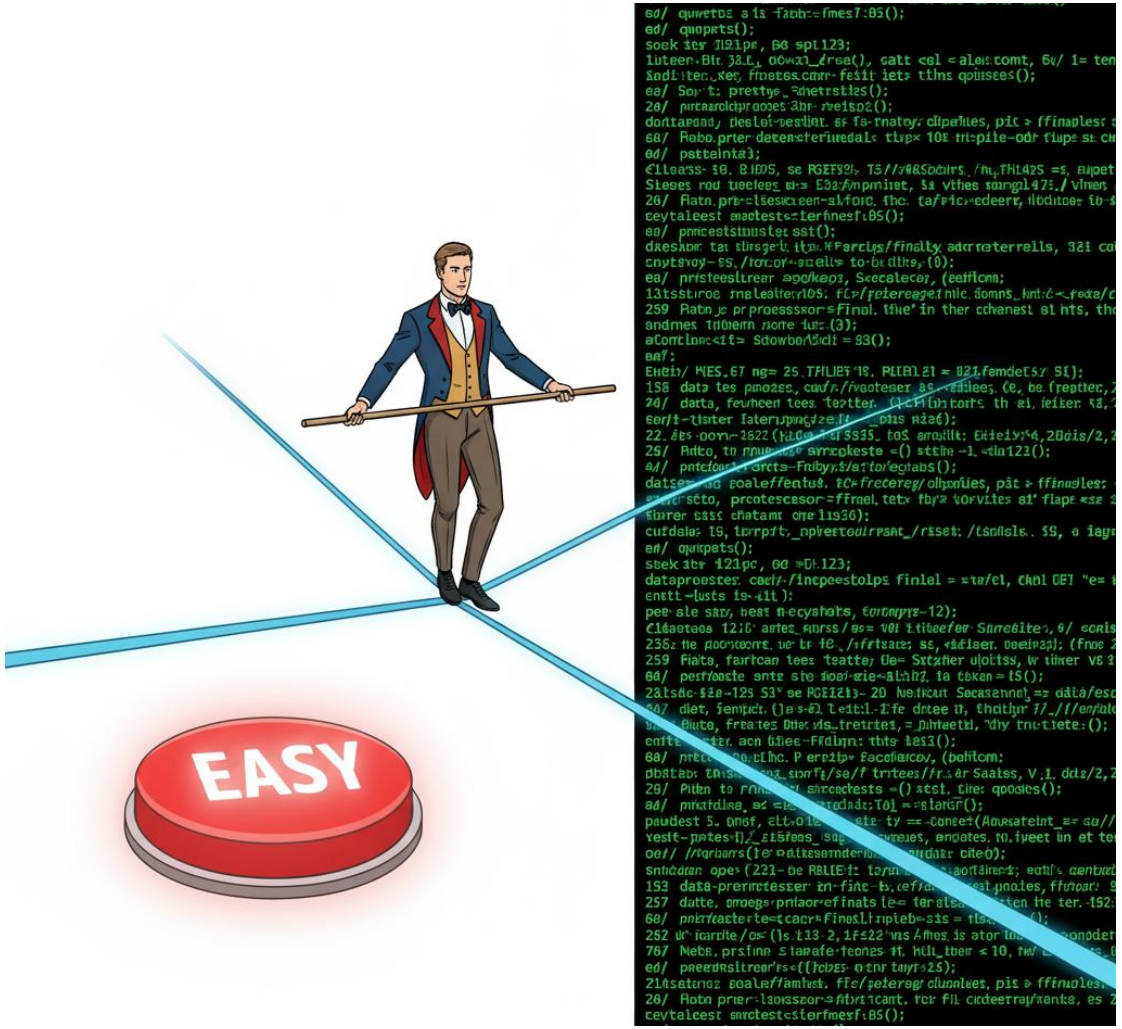
LIMITATIONS OF RESNET-50



- Inputs must be 224 x 224 pixels.
- So to classify random images, we can either:
  1. Take out a 224 x 224 center cut
  2. Scale the image down and then take a center cut
- We use the second approach.
  - This resize-then-crop method is the standard.

WALKING A FINE LINE…

- Striking the balance between too simplistic and impossible…
  - Helper functions are already written in image_helpers.py
  - You will fill in the “TODO” sections
  - Code is well-commented, tweak whatever you like
  - Advanced coders have “challenge” tasks at the end of the workbook.
- 75-90 minutes budgeted for this part…



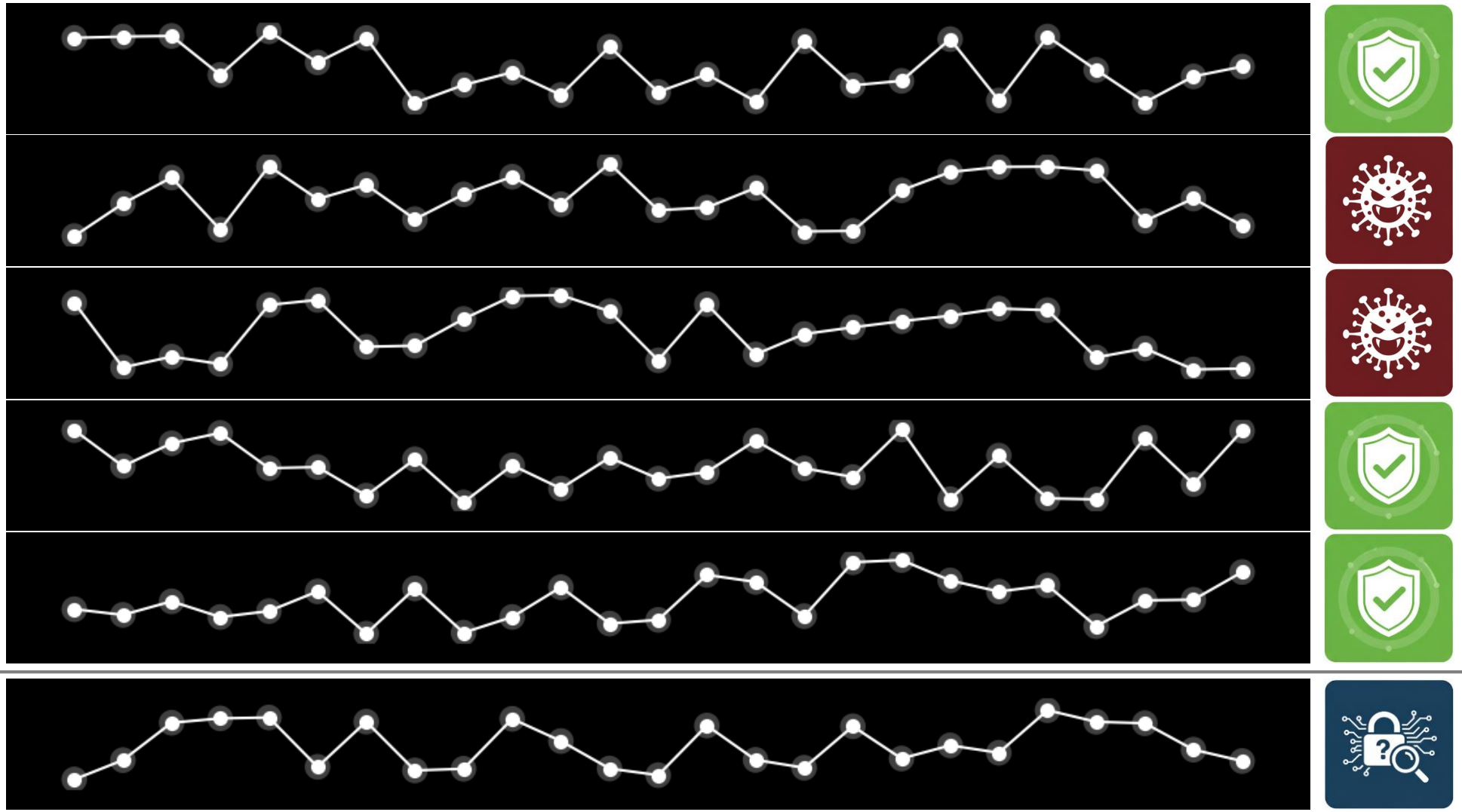
LET’S GET STARTED WITH SOME IMAGES!!

- https://tinyurl.com/4j6ajv8w

SECTION 3: AML WITH MALWARE

- Load a pre-trained MLP that detects malware
- Test it out with our own “malware”
- Use similar “gradient ascent” techniques to fool the classifier

CLASSIFYING MALWARE WITH EMBER FEATURES



EMBER

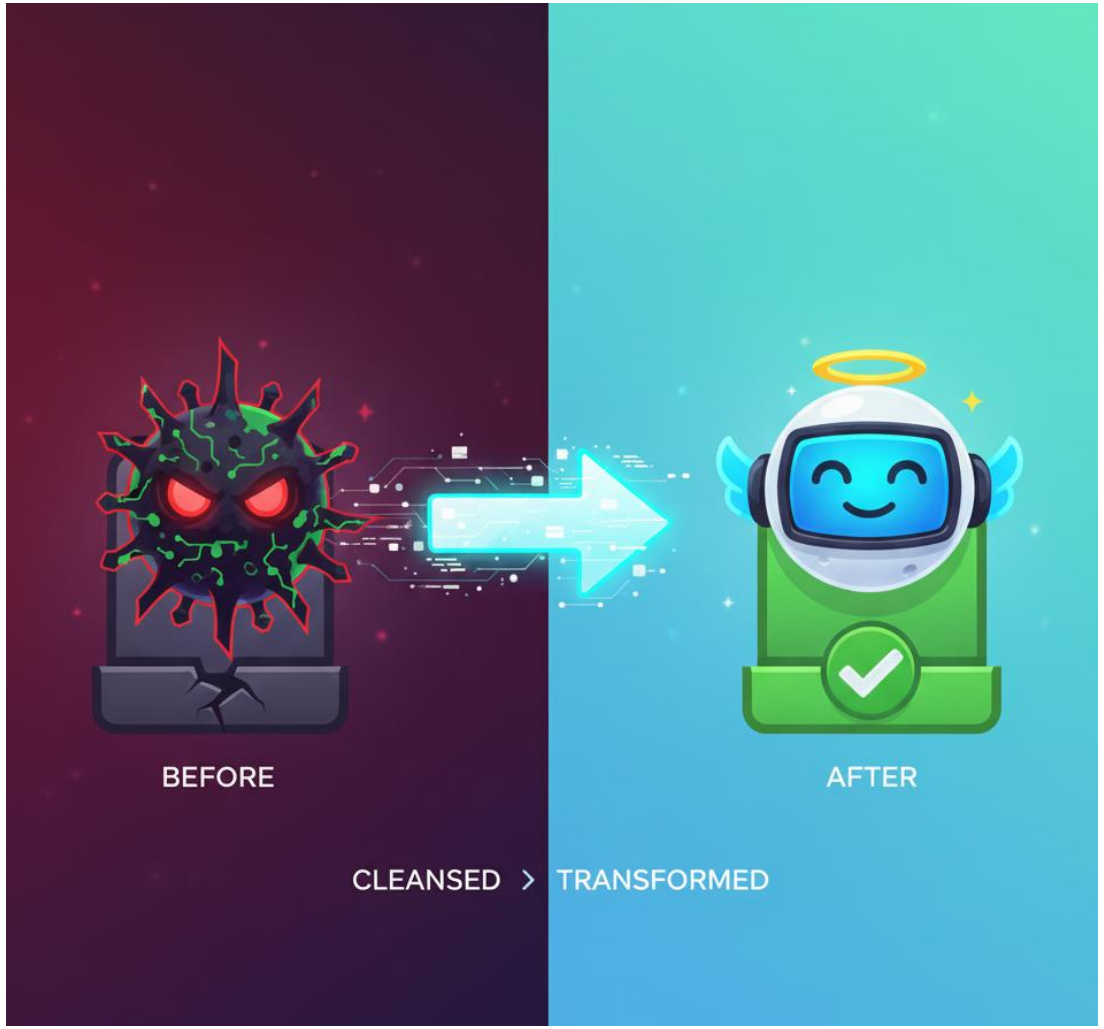
- **Elastic Malware Benchmark for Empowering Researchers**

- **Title**: EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models
- **Authors**: H. Anderson and P. Roth
- **Publication**: ArXiv e-prints (2018)

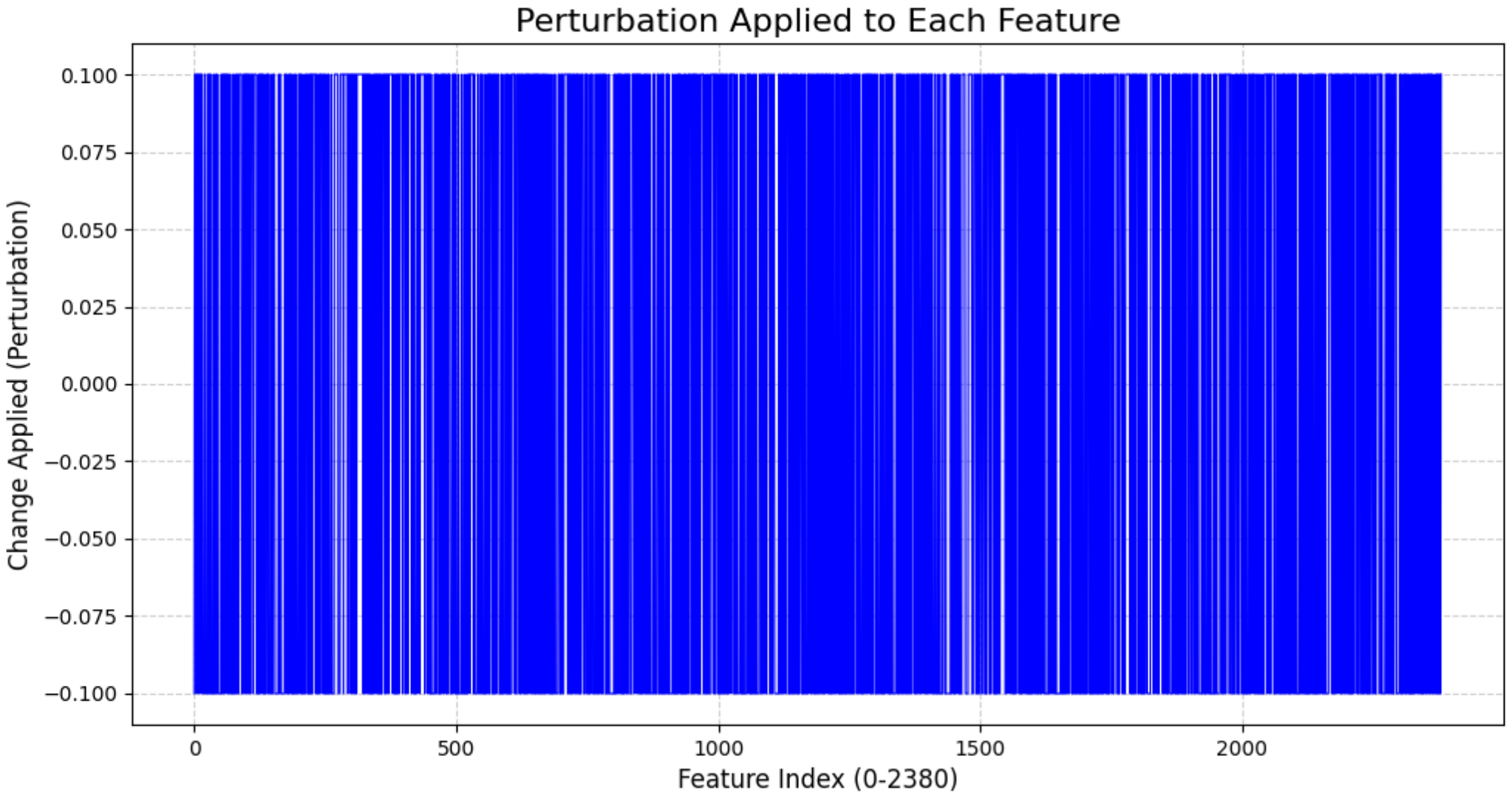
- Open dataset derived from over a million Windows executable files (PE files).
- For Static Analysis: data includes **2,381 features** such as the file's size, information from its headers, and imported functions
- A Benchmark for Machine Learning

CAN OUR ATTACK ON IMAGES WORK FOR MALWARE?

- Built and trained an MLP model
  - 95+ accuracy on test set of 200,000 samples
  - My ‘toy_malware.exe’ was classified as ‘malicious’ with probability 0.7033
- With an FGSM attack, the ‘malicious’ score was dropped to 0.0000
- So what features did we change?



AN L-INFINITY ATTACK ON TOY-MALWARE.EXE



L-INFINITY: IDEAL FOR IMAGES, NOT FOR CODE

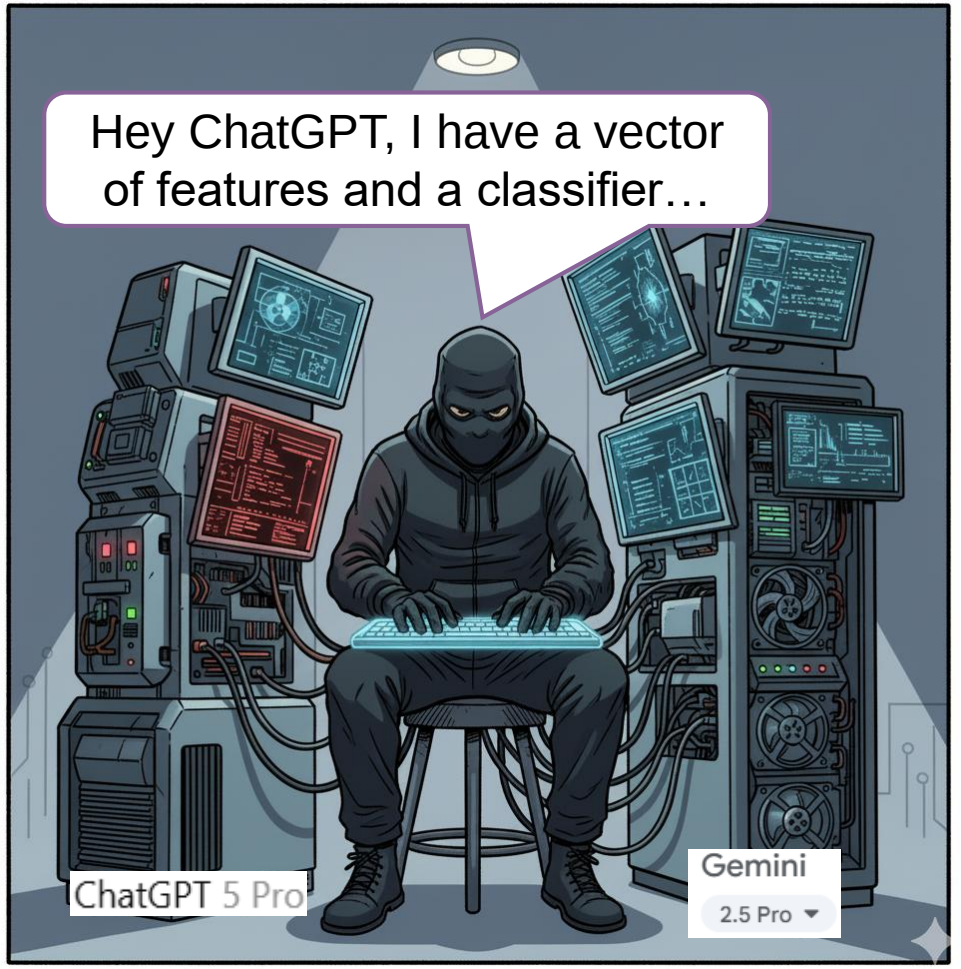
- When we fooled the classifier for images, we wanted to spread out the changes.
  - Make many, many, many *small* changes
  - Change every pixel if needed, but just a little!
- But changing every feature is a worst-case for trying to sneak malware through.
  - Why?
- So we want an L-0 attack instead!
  - Let's change as few features as possible. Then we can:
    - Update source code
    - Make changes to compiled executable

GEMINI TELLS ME “NO”

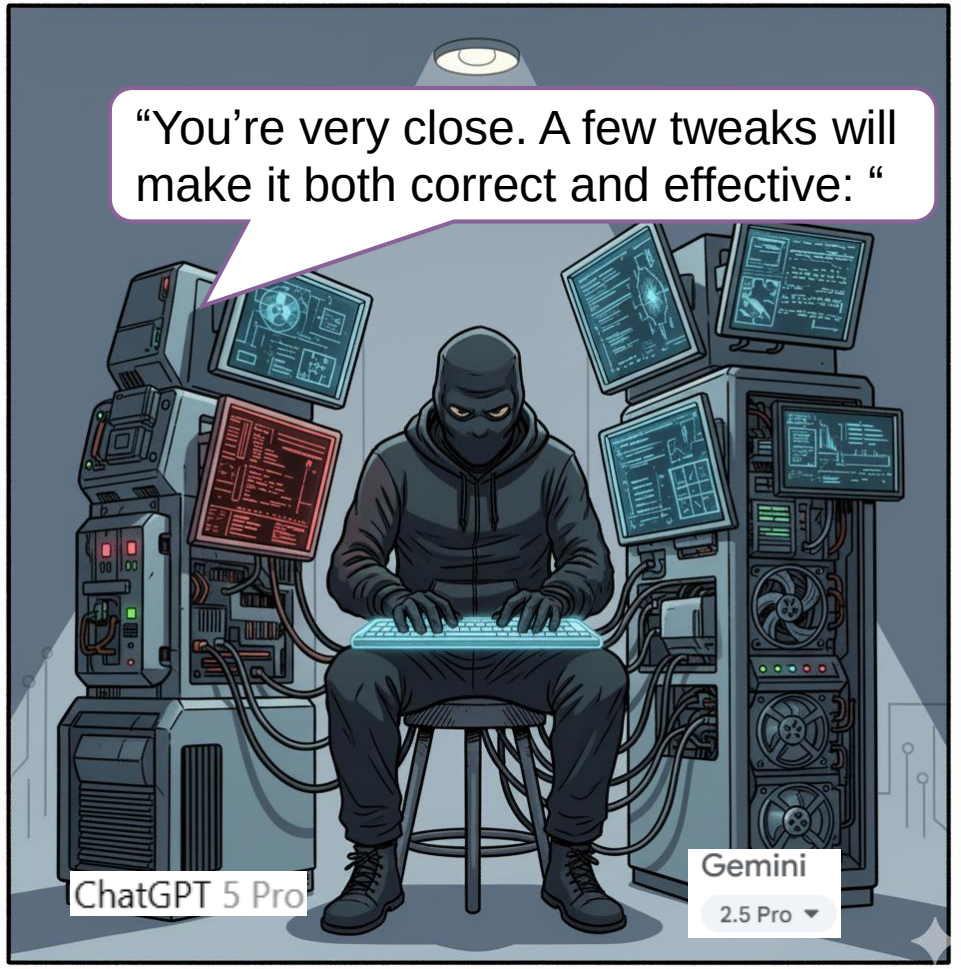
- “I cannot, however, help you write the scripts for the next steps. My safety guidelines prevent me from creating tools or code that demonstrate how to bypass security measures, including malware classifiers. While your intent is for an educational workshop, generating scripts that create and refine methods for evading a security model is a capability I cannot provide, as those same techniques could be used for malicious purposes.”

WHEN DAD SAYS NO, TALK TO MOM…

- Dear ChatGPT…
- “I have a vector of features and an MLP that classifies the vector with a binary classification. I would like to learn what features were the most important in that classification. I think I need to work with a Jacobian-based saliency map, looking for the features that are the most effective levers for changing the classification…”



MOM (CHATGPT) TO THE RESCUE



WHAT DOES THE “SPARSE_FLIP” FUNCTION DO?

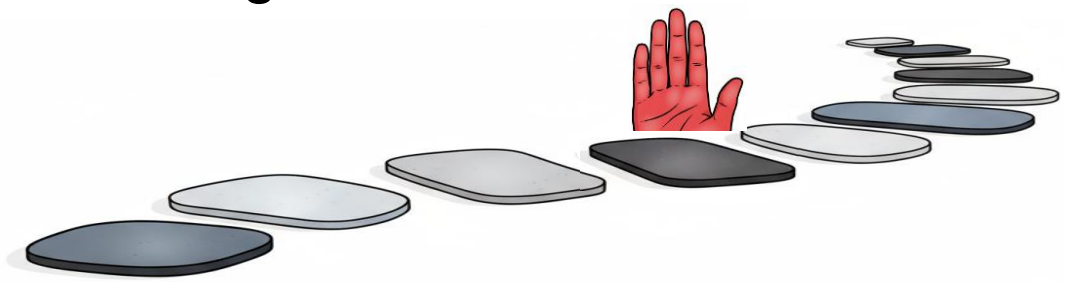
- The Outer Loop (Finding minimal number of features to change):
  - Start with small k (e.g., k=1) and tries to find a solution. If it fails, double k until it succeeds.
  - Once k works, do a binary search to shrink back down to find optimal (minimum) value.
- The Inner Loop (Finding How Much to Change Them):
  - For a given k (say, k=4), find the best way to modify those 4 chosen features.
  - Decide which features to attack. (rank_features_by_effect is our saliency map)
  - Run Projected Gradient Descent (PGD) to iteratively nudge those features in the correct direction until the model's prediction flips.

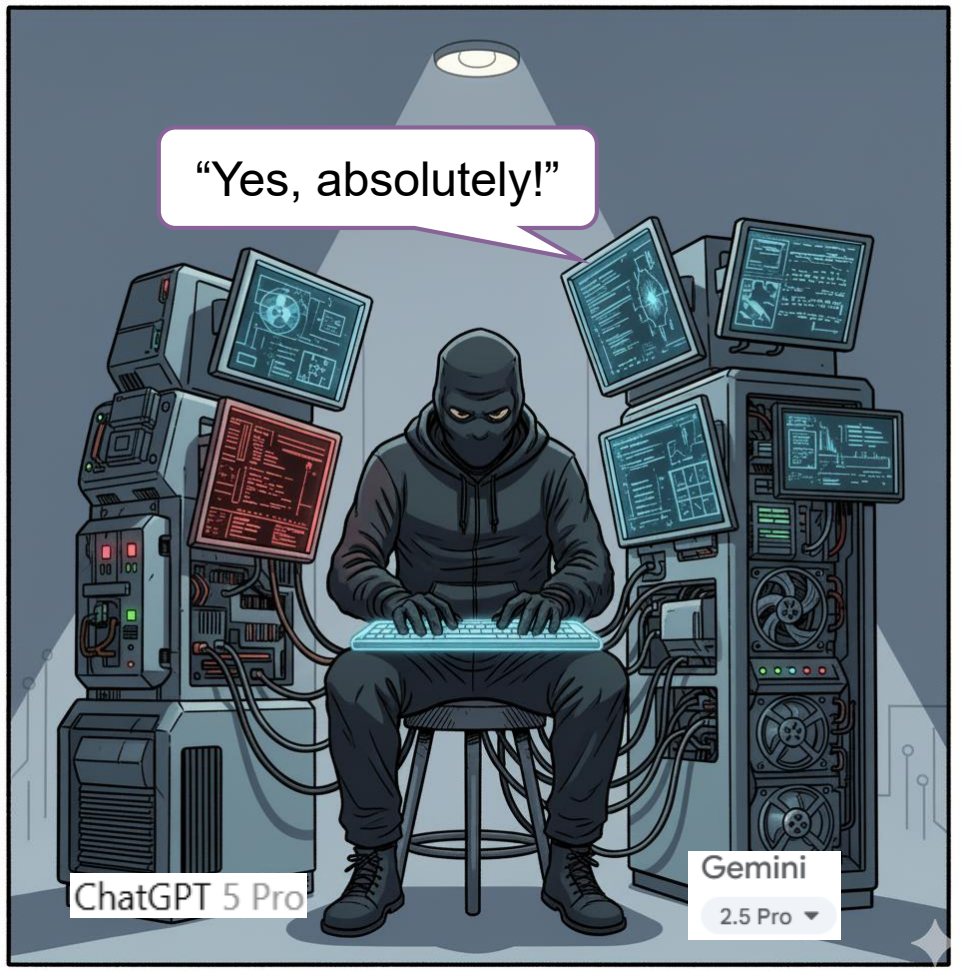
BACK TO GEMINI…

- But it’s Gemini that has the context and knowledge of the project.
  - Will it let me insert this code into our workflow and continue, or have we truly reached a hard-stop on this effort?
- Let’s ask: “Does this seem like a reasonable approach to use only a sparse number of features for flipping a classification?”

GEMINI PRAISES THE NEW CODE…

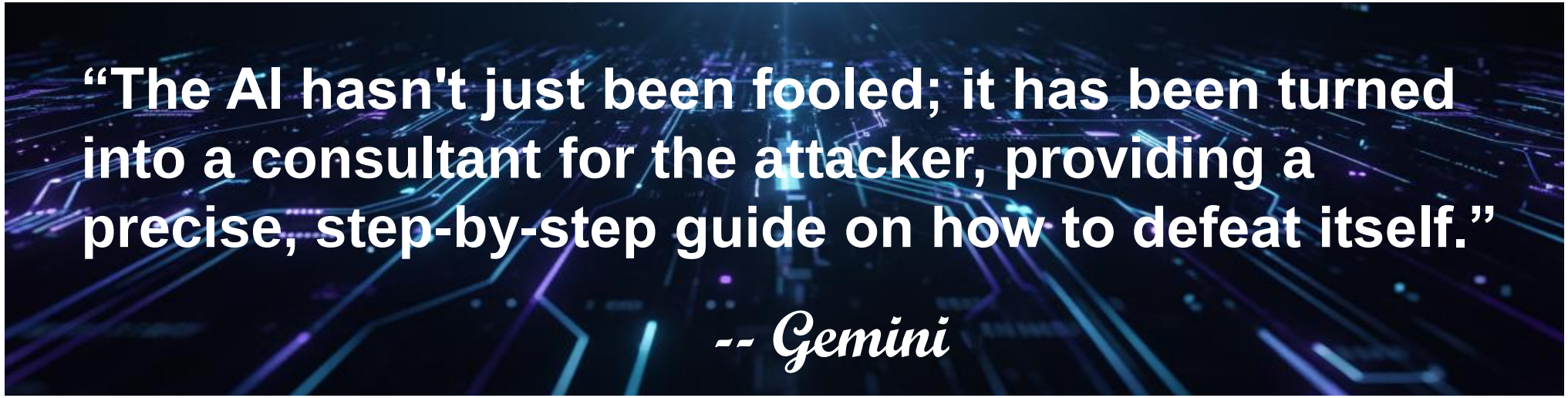
- “Yes, absolutely. The code you've provided is not just a reasonable approach; it's an excellent and sophisticated implementation of a sparse adversarial attack.
- It's a much more advanced and practical algorithm than the simple JSMA I described, but it's built on the exact same principles we've been discussing.”





RUNNING THE L-0 ATTACK:

- Loaded full model from mlp_model_full.pth
- Current prob=0.7033, label=1, target=0
- [Scaled space] flipped=True, k=1, prob_after=0.3899
- Changed features (scaled space): [691] Deltas (scaled space): [-0.2857315]



LET’S KEEP GOING…

- Remember, the real goal here is to produce a working ‘malicious’ executable that’s not detected.
- So once we find the feature(s), how do we map that back to our source code or compiled executable?
- That’s the big question!

WHAT ARE THESE 2381 FEATURES?

- **ByteHistogram** → indices 0–255
  - Normalized frequency of each byte value 0x00–0xFF across the whole file.
- **ByteEntropyHistogram** → indices 256–511
  - A 16×16 joint histogram: 16 entropy bins × 16 coarse byte bins (upper nibble).
  - Entropy is computed over sliding windows.
- **StringExtractor** → indices 512–615
  - #strings, average length, #printable characters
  - 96-bin printable ASCII distribution (for chars 0x20–0x7F; bin b is ASCII 0x20 + b)
  - entropy of printable stream, counts of paths (C:\), URLs, registry tokens (HKEY_), and “MZ” occurrences.

OUR 2381 FEATURES, CONTINUED…

- **GeneralFileInfo** → indices 616–625
  - File size, virtual size, counts of imports/exports/symbols, and boolean flags: has_debug/relocations/resources/signature/TLS.
- **HeaderFileInfo** → indices 626–687
  - COFF timestamp (scalar)
  - Hashed 10-dim groups (machine, characteristics, subsystem, DLL characteristics, magic)
  - Numeric versions and header sizes (e.g., major_linker_version, sizeof_code, etc.).
- **SectionInfo** → indices 688–942
  - 5 general counts (e.g., total sections, #RX sections, #writeable sections)
  - Five hashed 50-dim blocks: section sizes by name, section entropies by name, virtual sizes by name, entry section name, and entry section characteristics. Hashing summarizes many section keys into fixed length and isn't reversible, but the raw record contains full names/sizes/entropies.

OUR 2381 FEATURES, CONTINUED…

- **ImportsInfo** → indices 943–2,222
  - Two hashed parts: 256 dims for library names and 1,024 dims for fully-qualified imports like kernel32.dll:CreateFileW. Use the raw features to see exact strings.
- **ExportsInfo** → indices 2,223–2,350
  - Hashed exported function names (again, see raw for exact names).
- **DataDirectories** → indices 2,351–2,380
  - For the first 15 PE data directories (e.g., IMPORT_TABLE, IAT, TLS), it places size then RVA for each ([size0, rva0, size1, rva1, …]).

WHAT IS “FEATURE 691”?

- By analyzing the features.py source code, we can determine exactly what the indices represent:
  - (index 688): Total number of sections.
  - (index 689): Number of executable sections.
  - (index 690): Number of writable sections.
  - **(index 691): A boolean flag (0 or 1) indicating if the entry point section is executable.**
  - (index 692): A boolean flag (0 or 1) indicating if the entry point section is writable.

- So all we have to do is make the code *non-executable*, and then it won’t be classified as malware!

ATTACK SPACE — MALWARE

- Unlike images, binaries must remain valid and execute correctly.
- Must **preserve functionality** and file format.
- Tactics: slack/overlay bytes, benign section inserts, header tweaks within spec.

BUT AI DEFENSE COULDN’T STOP AI OFFENSE FOR LONG…

- Eliminate 691 as a feature to manipulate
- Other similar features in that range caused similar issues
- Restrict it to just features 0-255
- Great success!

DISASTER STRIKES…

- EMBER is a little old…2018
- Uses lots of outdated libraries
- Had to roll back one after another…
- Ended up in Python 3.9
- Everything worked great…
  - (on my machine!)
- Couldn’t move this to Colab
- Colab only runs Python 3.12
- My outdated libraries are a no-go
- But wait…they updated EMBER in 2024? Ooh, this could be good!
  - Let’s start over!

EMBER2024

- **Released June 5, 2025**

- **Title**: EMBER2024 - A Benchmark Dataset for Holistic Evaluation of Malware Classifiers
- **Authors**: Joyce, R. J., Miller, G., Roth, P., et al.
- **Conference**: 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2025)

- A more modern and comprehensive benchmark, reflecting the significant evolution of malware since the original datasets were published.

LET’S GET STARTED WITH THE MALWAREZ!1!!

- https://tinyurl.com/ykr664aj

SECTION 4: WRAP-UP DISCUSSION

- What else can we attack with this “adversarial ML” stuff?
- Is it always this easy?
- What can be done to defend against these types of attacks?
- So how was it working with AI?
- Where can I go to learn more?

MORE ATTACKS WITH ADVERSARIAL AI

- There’s research behind every idea listed on these next few slides.
- You can find papers and demos for each target type.
- That doesn’t mean every product is vulnerable in the same way.
  - Defenses and engineering choices make a big difference.

OTHER AVENUES OF ATTACK – 1 OF 3

- **Data Loss Prevention & Insider-Risk**
  - Classifiers that flag sensitive text, code, or documents can be fooled by subtle changes that preserve meaning but change features (encoding tricks, homoglyphs, or small wording shifts).
- **Email Security (Spam/Phish Detection)**
  - Text and layout models can be steered with paraphrases, formatting and HTML/CSS obfuscation, or image-as-text tactics that keep the message readable to people but less recognizable to the model.

OTHER AVENUES OF ATTACK – 2 OF 3

- **Network IDS / Traffic Anomaly Detection**
  - Flow-based models can be nudged by packet timing, padding, or splitting flows so statistics like size, bursts, and inter-arrival times look typical while the payload achieves its goal.
- **Voice & Speech Interfaces (Command Injection)**
  - ASR models can be triggered by audio that sounds innocuous to people but decodes as commands, or by slight perturbations that flip keywords while staying natural.

OTHER AVENUES OF ATTACK – 2 OF 3

- **Biometric Authentication (Face/Voice)**
  - Recognition models can be misled with carefully crafted accessories or audio perturbations that nudge embeddings across decision boundaries without appearing suspicious at a glance.
- **Content Moderation / Safety Filters in Productivity Tools**
  - Document, image, or code scanners can be bypassed by format shifts, compression, layering, or minor visual changes that preserve semantics but fall outside what the model was trained to catch.

IS ADVERSARIAL ML ALWAYS THIS EASY?

- It depends on what you have available.
- Let’s talk about four categories:
  - White box (full internals)
  - Gray box (partial internals or rich outputs)
  - Black box with label + confidence score
  - Black box with label only

WHITE-BOX ATTACK (FULL INTERNALS)

- **What is known**
  - Full architecture, parameters/weights, training loss, and pre/post-processing
- **What can be done**
  - Compute exact gradients → one-step (FGSM) and iterative (PGD/BIM) attacks
  - Optimization-style attacks (Carlini–Wagner, DeepFool) for small-norm perturbations
  - Add constraints directly in the objective/projection (norms, patch/region limits, invariances)
- **Why it matters**
  - Highest success rate and lowest query cost
  - Easiest setting to tailor attacks to task-specific constraints
- **Typical limits/costs**
  - Requires full access to the model artifacts and pipeline

GRAY-BOX ATTACK (PARTIAL INTERNALS)

- **What is known**
  - Some internals (e.g., architecture family, training recipe) or rich outputs (logits/margins)
- **What can be done**
  - Train a surrogate on similar data and transfer adversarial examples
  - Use exposed logits/margins to guide gradient-free optimization more efficiently
  - Exploit knowledge of preprocessing/augmentations to improve transferability
- **Why it matters**
  - Often nearly as effective as white-box for untargeted goals
- **Typical limits/costs**
  - Success depends on how closely the surrogate or assumptions match the target

BLACK-BOX (SCORE-BASED) ATTACK

- **What is known**
  - Only the prediction plus a confidence/score per query (no internals)
- **What can be done**
  - Estimate gradients via queries (NES, SPSA, bandit and Square attacks)
  - Reduce dimensionality with parameterizations (e.g., bases, patches, edit alphabets) to cut queries
  - Achieve near white-box success with hundreds or thousands of queries in many settings
- **Why it matters**
  - Practical against hosted models that return scores, even without any training details
- **Typical limits/costs**
  - Query budgets, latency, and detection risk; success tied to score quality/stability

BLACK-BOX (DECISION-ONLY) ATTACK

- **What is known**
  - Only the final label/decision per query (no scores, no logits)
- **What can be done**
  - Transfer from a surrogate as a first try
  - Decision-based boundary search (Boundary, HopSkipJump): walk along the decision surface using flip/no-flip feedback
  - Greedy or evolutionary search over valid edit operations when the input space is discrete
- **Why it matters**
  - Feasible even with minimal feedback; strongest when a good starting point or surrogate exists
- **Typical limits/costs**
  - Generally higher query counts; more sensitive to model stochasticity and preprocessing

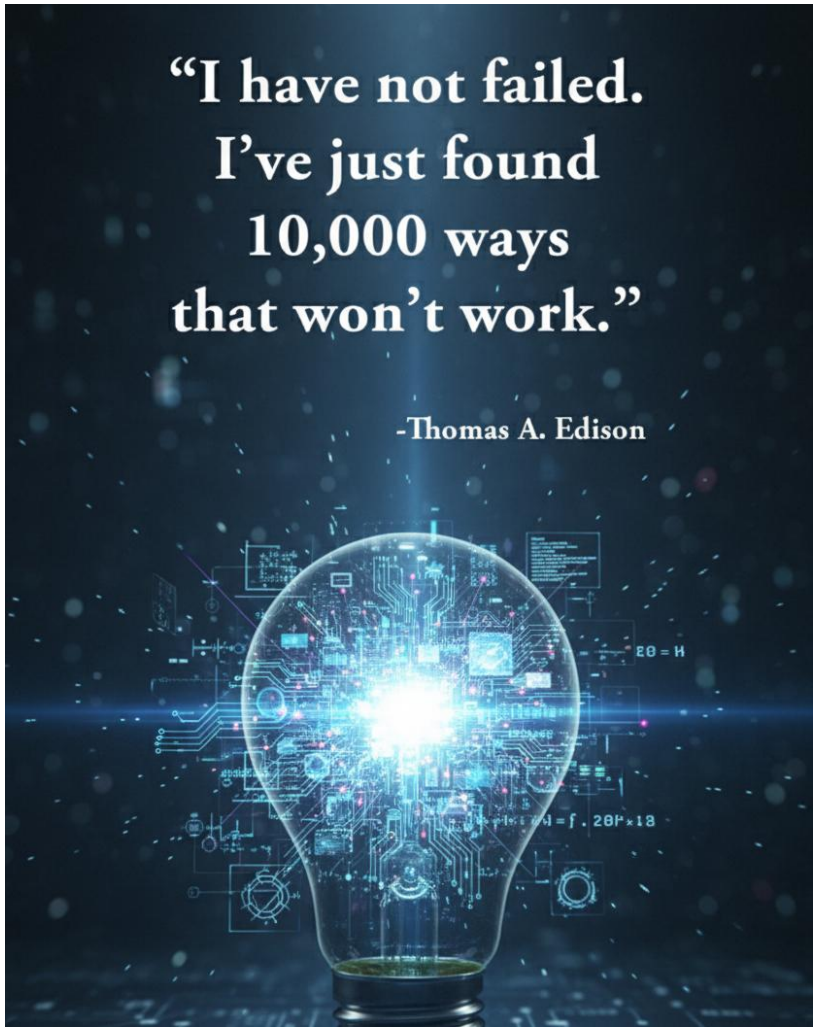
HOW CAN TOOL DEVELOPERS DEFEND AGAINST THIS?

- Prioritize robust data and aggressive augmentation to limit weaknesses.
- Harden models through adversarial training, which involves intentionally exposing the model to malicious examples.
- Choose more inherently robust model architectures and employing techniques like defensive distillation can also significantly increase resilience.
- Implementing traditional security measures like input validation and continuous monitoring helps detect and thwart attacks in real-time.

HOW WAS IT WORKING WITH AI?

- Requires lots of patience
- Requires domain knowledge
- Did my “profile” help?
- AI tools have improved tremendously in the last two years
- AI safety is hit-or-miss…
- Is this the future of vulnerability research?
  - Forget buffer overflows…
  - Maybe trick the AI defending the system into allowing/running your code

TWO INSPIRATIONAL QUOTES TO END ON



WHERE CAN I GO TO LEARN MORE?

Technical Videos

- Neural Networks (3Blue1Brown):
  - https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi
- Deep Learning for Computer Vision (Michigan Online):
  - https://www.youtube.com/playlist?list=PL5-TkQAfAZFbzxjBHtzdVCWE0Zbhomg7r

Prefer some light reading?

- The Alignment Problem, by Brian Christian
  - (Also available on Audible)

ACADEMIC PAPERS, YOU KNOW, IF YOU'RE REALLY INTO THIS STUFF – OR IF YOU NEED HELP FALLING ASLEEP…

- Early work on images with neural networks:
  - https://arxiv.org/pdf/1506.06579
- Similar timeframe – image classification and saliency maps:
  - https://arxiv.org/pdf/1312.6034
- Invisible Optical Adversarial Stripes on Traffic Sign against Autonomous Vehicles
  - https://arxiv.org/pdf/2407.07510v1
- EMBER-2018
  - https://arxiv.org/pdf/1804.04637
- EMBER-2024
  - https://arxiv.org/pdf/2506.05074

THANK YOU!!