Deploying AI

Model and System Evaluation

```
$ echo "Data Science Institute"
```

Introduction

Agenda

Agenda

- Performance metrics
- Exact evaluation and using AI as a judge
- Designing an evaluation pipeline

Performance Metrics

Why Evaluation Matters

- AI use brings risk of catastrophic failures:
  - Lawyers using AI, submit documents containing hallucinations (HAI Stanford, lawnext.com, clio.com, CBC, Reuters).
  - Air Canada found liable for misleading information provided by its chatbot (CBC).
  - Chatbot encouraging self-harm (NBC).
- Evaluation is the biggest hurdle to adoption.
  - Evaluation must be considered at the system level.
  - To mitigate risks, first identify the places where the system is likely to fail and design evaluations around them.

AI Risks

MIT AI Risk Repository - Domain Taxonomy of AI risks

Domain / Subdomain	
1	***Discrimination & Toxicity***
1.1	Unfair discrimination and misrepresentation
1.2	Exposure to toxic content
1.3	Unequal performance across groups
2	***Privacy & Security***
2.1	Compromise of privacy by obtaining, leaking or correctly inferring sensitive information
2.2	AI system security vulnerabilities and attacks
3	***Misinformation***
3.1	False or misleading information
3.2	Pollution of information ecosystem and loss of consensus reality
4	***Malicious actors & Misuse***
4.1	Disinformation, surveillance, and influence at scale
4.2	Cyberattacks, weapon development or use, and mass harm
4.3	Fraud, scams, and targeted manipulation

Domain / Subdomain	
5	***Human-Computer Interaction***
5.1	Overreliance and unsafe use
5.2	Loss of human agency and autonomy
6	***Socioeconomic & Environmental Harms***
6.1	Power centralization and unfair distribution of benefits
6.2	Increased inequality and decline in employment quality
6.3	Economic and cultural devaluation of human effort
6.4	Competitive dynamics
6.5	Governance failure
6.6	Environmental harm
7	***AI system safety, failures, and limitations***
7.1	AI pursuing its own goals in conflict with human goals or values
7.2	AI possessing dangerous capabilities
7.3	Lack of capability or robustness
7.4	Lack of transparency or interpretability
7.5	AI welfare and rights
7.6	Multi-agent risks

(Slattery et al, 2024)

Read more: airisk.mit.edu

Challenges of Evaluating Foundation Models (1/2)

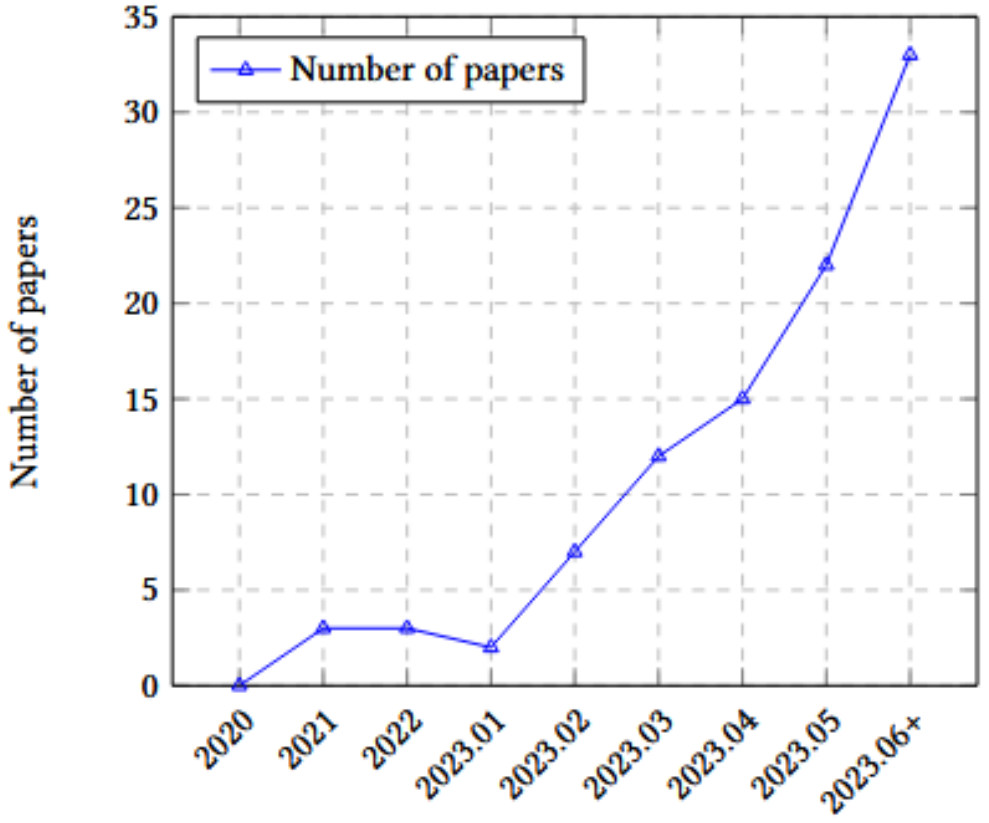
- As AI systems become more capable, it is more difficult to evaluate them.
- Open-ended nature of Foundation Models(FM) undermines the Machine Learning (ML) approach of comparing against a ground truth.
- Black-box models: model providers do not expose model details or app developers are not experts in FM.

Challenges of Evaluating Foundation Models (2/2)

- Benchmarks saturate quickly: a benchmark becomes saturated for a model when it achieves the perfrect score.
  - GLUE (2018) → SuperGLUE (2019)
  - NaturalInstructions (2021) → SuperNaturalInstructions (2022)
  - MMLU (2020) → MMLU-Pro (2024))
- Expanded scope: we want to evaluate not just performance on known tasks, but also discovery and performance of new tasks.

Evaluation Landscape

- There appears to be an exponential growth of papers and repos on evaluation.
- There is increased interest in evaluation, but investment still lags behind model training and orchestration.
- Many practictioners still rely on *eyeballing* or *ad hoc prompts*.
- We need systematic evaluation pipelines.
- Image: (Chang et al, 2023)



Language Modeling Metrics

- Most auto-regressive models are trained using entropy or perplexity.
- cross entropy, perplexity, Bits-Per-Character (BPC) and Bits-Per-Byte (BPB) are related metrics that can be applied beyond language modelling, they work for any model that generates sequences of tokens.
- In short, a language model generates the distribution of the data. The better this model learns, the better it is at predicting what comes next in the training data and the lower its cross entropy.

Entropy

- Entropy measures how much information, on average, each token carries. Intuitively, entropy measures how difficult it is to predict what comes next in a language.
- Higher entropy indicates more information per token and more bits are required to represent the token.

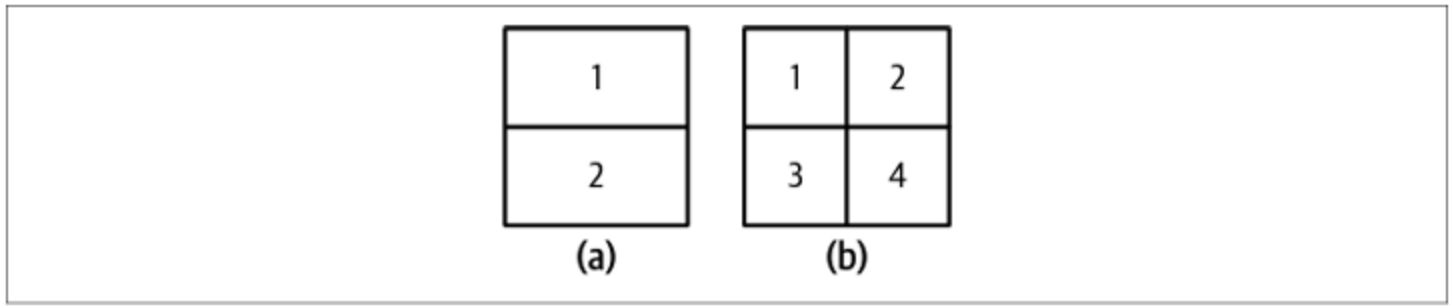


Figure 3-4. Two languages describe positions within a square. Compared to the language on the left (a), the tokens on the right (b) carry more information, but they need more bits to represent them.

(Huyen, 2025)

Cross Entropy

- Cross Entropy on a dataset measures how difficult it is for the language model to predict what comes next in the dataset.
- Cross Entropy depends on:
    - The training data's predictability, measured by the data's entropy.
    - How the distribution captured by the language model diverges from the true distribution of the training data.

Entropy and Cross Entropy

Notation

- Entropy and cross entropy are denoted *H*.
- Training data has distribution *P*.
- *Q* is the distribution learned by the model.

Therefore

- Training data's entropy is *H(P)*.
- Divergence of *Q* with respect to *P* can be measured using the Kullback-Leibler (KL) divergence, $D_{KL}(P||Q)$.
- Model's cross entropy with respect to the training data is $H(P,Q) = H(P) + D_{KL}(P||Q)$.

Bits-per-Character and Bits-per-Byte

- One unit of entropy and cross entropy is bits: if a language model has entropy of 6 bits, it requires 6 bits to represent a token.
- The number of bits per token is not comparable across models because each model can use a different tokenizer.
- A first alternative could be Bits-per-Character (BPC), but character encodings can differ: a character in ASCII will be represented in 7 bits, but the same character in UTF-8 can be encoded anywhere between 8 and 32 bits.
- Bits-per-Byte (BPB), the number of bits a language model needs to represent one byte of the original training data.
- Cross Entropy tells us how efficiently a model can compress text.

Perplexity

- Perplexity is the exponential of entropy and cross entropy.
- The perplexity (PPL) of a dataset's distribution is:

$$PPL(P) = 2^{H(P)}$$

- The perplexity of a language model with learned distribution Q is:

$$PPL(P, Q) = 2^{H(P,Q)}$$

- Perplexity measures the amount of uncertainty it has when predicting the next token.
- A higher uncertainty means there are more possible options.

Guidance on Perplexity

- What is considered a good value for perplexity depends on the data itself:
  - More structured data gives lower expected perplexity.
  - The bigger the vocabulary, the higher the perplexity.
  - The longer the context length, the lower the perplexity.
- Perplexity is a good proxy on a model's capabilities: if a model is bad at predicting the next token, it will tend to bad further downstream.
  - Perplexity is highest for unpredictable texts, such as: "My dog teaches quantum physics."
  - Perplexity is highest for giberish: "dog cat go eye."

Limitations of Perplexity

- Perplexity might not be useful for models that have been post-trained with SFT or RLHF.
    - Post-training is about teaching a model a task.
    - If a model learns a task, it may get worse at predicting the next token.

Exact Evaluation and Using AI as a Judge

Evaluating Models in Downstream Tasks

- Our interest in FM and LLM is not necessarily to predict the next token, but instead we are interested in other tasks such as summarization, agentic automation, and so on.
- To evaluate a FM in downstream tasks, there are two approaches:
  - **Exact evaluation**: produces a judgement or assessment without ambiguity. Two approaches are:
    - Functional correctness.
    - Similarity to references.
  - **Subjective evaluation**: the evaluation can change based on the judge model and prompt.

Exact Evaluation: Functional Correctness

- Similar to unit testing in software engineering, functional correctness tests aim to assess if the system works as intended.
- Evaluate the system based on whether it performs the intended functionality.
- Popular benchmarks: HumanEval, Mostly Basic Python Problems (MBPP), Spider and Spider2.

Testing Functional Correctness with HumanEval

- A benchmark problem comes with a set of test cases. Each test case consists of a scenario the code should run and the expected output for that scenario.
- Generated code is shown with yellow background.
- (Chen et al., 2021)

```
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) ==>12
    solution([3, 3, 3, 3, 3]) ==>9
    solution([30, 13, 24, 321]) ==>0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

Evaluating Test Cases

- For each problem, k code samples are generated.
- A model solves a problem if *any* of the k code samples it generated pass all of that problme's test cases.
- The score pass@k is the ratio of solved problems to total number of problems.
- For example, a model that solves 5 out of 10 tests problems with 3 generated code samples each has a pass@3 score of 50%.

Similarity Measurements Against Test Data

- The approach is to evaluate AI's outputs against reference data.
- Reference data is called ground truth or canonical responses.
- Metrics that require references are called referenced-based; metrics that do not require references are called reference-free.
- Four approaches:
    i. Ask an evaluator.
    ii. Exact match: generated response matches exactly the canonical response.
    iii. Lexical similarity: how similar the generated response *look* like the reference responses.
    iv. Semantic similarity: how similar are the *meaning* of generated and reference responses.

Exact Match

- The generated response matches exactly the reference response.
- Works for tasks with short, exact responses, such as simple math, common knowledge, trivia-style questions.
- Can take into account formatting differences. For example, a variation of exact match could evaluate if the reference response is contained in the generated response.
- Exact match is rarely useful beyond simple tasks.

Lexical Similarity

- Lexical similarity measure how much two texts overlap.
- Simple implementation: count number of tokens in common.
    - Reference: My cats scare the mice.
    - Response A: My cats eat the mice.
    - Cats and mice fight all the time.
    - Assuming one word per token,

Introduction to Embeddings

- **Embeddings = vector representations of meaning**
- Capture semantic similarity (cosine distance)
- Used in: retrieval, clustering, anomaly detection, deduplication
- Models: BERT, CLIP, Sentence Transformers, OpenAI Embeddings:contentReference[oaicite:5]{index=5}

AI as a Judge

- Use AI to evaluate AI responses
- Benefits: fast, scalable, no reference data needed
- Studies show strong correlation with humans (GPT-4 ~85%)
- Applications: quality, relevance, safety, hallucination checks:contentReference[oaicite:6]{index=6}

Limitations of AI Judges

- **Inconsistency**: probabilistic outputs
- **Criteria ambiguity**: no standardization
- **Cost & latency**: evaluation doubles API calls
- **Biases**: self-bias, position bias, verbosity bias
- Should be combined with **exact or human evaluation**:contentReference[oaicite:7]{index=7}

Comparative Evaluation

- Compare models **side-by-side** instead of absolute scores
- Popularized by **Anthropic** & **Chatbot Arena**
- Algorithms: Elo, Bradley–Terry, TrueSkill
- Benefits: captures human preference, resists saturation
- Challenges: scalability, quality control, benchmark correlation:contentReference[oaicite:8]{index=8}

Evaluation Criteria for Applications

- **Domain-specific capability** (coding, math, legal knowledge)
- **Generation capability** (fluency, coherence, factual consistency, safety)
- **Instruction-following** (formats, constraints, style)
- **Cost & latency** (time per token, price per output):contentReference[oaicite:9]{index=9}

Factual Consistency & Safety

- **Local vs Global factual consistency**
- Methods: AI judges, SelfCheckGPT, SAFE, entailment models
- Safety risks: toxicity, bias, violence, harmful advice
- Benchmarks: TruthfulQA, RealToxicityPrompts, BOLD:contentReference[oaicite:10]{index=10}

Instruction-Following

- Core capability for foundation models
- Benchmarks:
- **IFEval** (format, constraints, JSON, keywords)
- **INFOBench** (content, style, linguistic rules)
- Roleplaying = common real-world use case:contentReference[oaicite:11]{index=11}

Cost and Latency

- Metrics: time-to-first-token, time per token, total query time
- Cost drivers: input/output tokens (APIs), compute (self-hosted)
- Trade-offs: performance vs cost vs latency:contentReference[oaicite:12]{index=12}

Designing an Evaluation Pipeline

References

References

- Chang, Yupeng et al. "A survey on evaluation of large language models." ACM transactions on intelligent systems and technology 15, no. 3 (2024): 1-45. (arXiv:2307.03109)
- Chen, Mark et al. (2021). "Evaluating large language models trained on code." arXiv:2107.03374.
- Huyen, Chip. Designing machine learning systems. O'Reilly Media, Inc., 2022
- Slattery, P. et al (2024). The AI Risk Repository: A Comprehensive Meta-Review, Database, and Taxonomy of Risks from Artificial Intelligence. arxiv:2408.12622

AI Engineering

Evaluation Methodology & System Evaluation

Model Selection Workflow

1. Filter by **hard attributes** (license, privacy, architecture)
2. Narrow with **benchmarks & leaderboards**
3. Run **custom evaluation pipeline**
4. **Monitor in production** for failures & drift:contentReference[oaicite:13]{index=13}

Open Source vs Model APIs

- **Model APIs**
- Pros: best models, scaling, guardrails, features (function calling)
- Cons: cost, vendor lock-in, limited control/transparency
- **Self-Hosting**
- Pros: control, transparency, customization, on-device
- Cons: heavy engineering effort, usually weaker models:contentReference[oaicite:14]{index=14}

Public Benchmarks & Leaderboards

- Thousands exist (BIG-bench, MMLU, GSM-8K, TruthfulQA)
- Leaderboards (Hugging Face, HELM) aggregate results
- Issues: saturation, contamination, benchmark correlation
- Best use: filter *bad models*, not pick the *best model*:contentReference[oaicite:15]{index=15}

Data Contamination

- Models often trained on public benchmarks → inflated scores
- Detection: n-gram overlap, low perplexity
- Handling: disclose contamination, evaluate on clean subsets
- Lesson: don’t fully trust public benchmark scores:contentReference[oaicite:16]{index=16}

Designing an Evaluation Pipeline

1. **Evaluate all components** (per task, per turn, per step)
2. **Create clear guidelines** & rubrics tied to business metrics
3. **Define evaluation methods & datasets** (exact, subjective, human-in-loop)
4. **Validate the pipeline** (reliability, bootstrap resampling, significance tests):contentReference[oaicite:17]{index=17}

Key Takeaways

- Evaluation is **harder than ever** but critical for AI safety & adoption
- Mix of methods: **functional correctness + similarity + AI judges + human checks**
- Comparative evaluation rising as models surpass human skill
- Custom pipelines > public benchmarks for real-world apps
- Evaluation is the **biggest bottleneck to AI adoption**:contentReference[oaicite:18]{index=18}