Measuring Rule-Following in Language Models

Benjamin Laufer [1] **Jon Kleinberg** [2]

Abstract

There are many instances in which we might want a language model to follow rules, including grammatical constraints and the avoidance of hateful or toxic statements. Following rules necessitates, first, keeping track of states where the rule applies, and second, following the rule when in these states. We introduce a formal framework for evaluating rule-following in language models, focusing on rules that can be expressed using deterministic finite automata (DFAs). We say a language model (ϵ, δ)-matches a DFA A if 1) it behaves similarly on all prefixes that lead to the same state in the DFA (up to divergence level ϵ) and 2) it distinguishes between prefixes that lead to different DFA states by exhibiting at least δ compliance over the set of suffixes that are treated differently by the automaton. To formalize this definition, we define probability distributions over the Myhill-Nerode Boundary (MNB) to measure a model's ability to distinguish between valid and invalid suffixes. Our approach defines rule compliance in probabilistic terms, offering a principled way to evaluate model behavior in light of specified language-generation constraints. We implement an experimental setup in which we train transformer models on synthetic languages and evaluate the compliance of their outputs. We discuss generalizations of our framework – including to non-regular languages – and the implications for evaluating and controlling language models.

1. Introduction

As generative AI gets adopted across sectors, there is interest in guaranteeing that these models comport with rules. At the most basic level, any language has grammatical rules, and designers want the output of a language model to follow them. Beyond grammar rules, a wide range of evaluation metrics and benchmarks have been proposed to assess desirable properties of model behavior, such as truthfulness and the avoidance of toxicity.

An outstanding challenge in this space is that evaluation is multifaceted: there is an ongoing proliferation of benchmarks and metrics, each reflecting its own goals and assumptions [10; 4]. Attempts at measuring rule-following or compliance raise additional questions about what should count as successful model behavior and how it should be measured. Designing appropriate measures for safe model performance, for example, can have significant ramifications for the trajectory of AI development [5].

DFA-based rules. Here we offer a formal framework for understanding rule-following in generative models. For our analysis here, we will focus on the set of rules that can be described by a Deterministic Finite Automaton (DFA). DFAs are a useful formal proxy for rules. They offer a clean way of determining whether a given sequence of tokens *passes* (i.e., meets the rules), and this conclusion can be made at every token transition and using finite memory. DFAs, of course, offer just one formalism for reasoning about rule-compliance, and we offer some analysis that attempts to generalize our framework to rules and languages that are not describable by DFA.

The present work. Our main contribution is to provide a framework for measuring rule-compliance in language models. Our framework consists of a measure of rule-compliance for any rule that can be expressed as a DFA, and also for some rules that cannot be expressed this way.[1] We operationalize our proposal in an experimental setting where we train transformers from scratch on known languages and evaluate the model's output using our framework. Empirically, we observe that transformers are remarkably successful when measured by naive methods of DFA compliance but exhibit certain tendencies that fall short of DFA learning using our metrics. In summary:

- We introduce the notion of "(ϵ, δ)-matching" a DFA to measure whether a generative model has learned the rules of a language (Section 2). Using an example of

[1]Cornell Tech, New York, NY, USA [2]Cornell University, Ithaca, NY, USA. Correspondence to: Benjamin Laufer <bdl56@cornell.edu>.

ICML 2025 Workshop on Assessing World Models.

[1]Languages adhering to these rules would be described as *non-regular* in formal language theory.

a simple language, we demonstrate how to evaluate a model for ϵ and δ in closed-form (Section 3).

- We introduce an experimental framework where we can specify languages as distributions over next tokens and evaluate the ability of generative models to emulate a simple language. Experimentally, using simple languages and small models, we observe the strengths and shortcomings of transformers at learning certain classes of rules (Section 4).

Related work. This paper builds on a line of work attempting to evaluate learning in generative models [3; 2; 1] and, especially, world model evaluation [6; 8; 9]. For example, Liu et al. [7] find the theoretically, transformers should be capable of learning an arbitrary DFA. The present work builds off the basic structure introduced by Vafa et al. [9] for understanding a language model's world model. Their analysis focuses on exact sequence-level distinctions via precision and recall metrics. Our contribution generalizes this to a probabilistic setting, accounting for varying likelihoods.

2. Conceptual Framework

We conceive of a language as a distribution over next tokens (e.g., letters). Our setting involves learning certain languages whose rules can be expressed as a DFA. We start with an overview of this set-up before introducing our main contribution: a metric for evaluating rule-following and a framework for using it.

Distributions vs Rules. Even though a language can be conceived of as a distribution over next tokens, people typically encounter only individual sequences, or *draws* from the distribution. When we conclude that a particular output breaks the *rules* of a language, we are not necessarily claiming something about the distribution of outputs, but instead about an individual sequence. One natural starting point is to conceive of the rules of a language as being broken when a sequence of tokens includes a next token that the language places 0 probability on. This would align with the views of Vafa et al. [9], who introduce the DFA as a model for capturing the world model implicit in an LLM. However, in many ways, we *want* information about the distribution of sequences to understand how frequently a language model violates rules. There are many models, including transformers, which place non-zero probability on *every* next token, so characterizing a model's compliance should take into account its frequency of violation and other skews. Thus, this framework attempts to provide a notion of rule-following with an eye towards probability distributions and aggregation.

2.1. DFA-based Language Rules

Here we introduce our model for learning the rules of a language. We define language rules using a DFA $A = (Q, \Sigma, \Delta, q_0, F)$ where Q is the finite set of states in the language, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the starting state, and $F \subseteq Q$ is the set of accepting states.

A sting is given by a sequence of letters (or, interchangeably, "tokens") in the alphabet, $x = x_1x_2x_3...x_n$ where each $x_i \in \Sigma$. Each letter corresponds to the transition over states $q \in Q$, via the transition function Δ. We denote the overall transition over a sequence using $q(q_0, x_i) = \Delta(\Delta(...\Delta(q_0, x_1), ...), x_n)$. Sometimes we will treat q_0 as given and simply write $q(x)$. For any sequence x, the DFA accepts x if the final state reached (q_f) is in the set F. Thus, we can say the language L "recognized" by the DFA A is given by: $L(A) = \left\{x \in \Sigma^{|x|} : q(q_0, x) \in F\right\}$.

2.2. DFA Matching

We now introduce formal definitions for understanding the extent to which a language generation model m complies with a DFA A. We are after a measure of distributional divergence: How much probability mass does the model place on legal and illegal sequences, and does the distribution of this mass depend predictably on the DFA state? To arrive at a measure, we start by defining needed concepts inspired by formal language theory to identify and compare legal and illegal sequences.

Distributions over the Myhill Nerode Boundary Informally, the Myhill-Nerode boundary for a model given two prefixes s_1, s_2 is the set of suffixes accepted by the DFA after one of the prefixes and rejected by the other.[2]

Definition 2.1 (Myhill-Nerode Boundary)**.** Given a generative model m, a DFA A and a compliance function $l(x) := x \in L(A)$. For two prefixes s_1, s_2 with $q(s_1) \neq q(s_2)$, the **Myhill-Nerode boundary** B is given by:

$$B(s_1, s_2) = \left\{x \in \Sigma^k : l(x \mid s_1) \neq l(x \mid s_2)\right\},$$

where $k > 0$ is the maximum suffix length.

With this definition, we can define probability distributions over the set of suffixes B. Within this set, we hope that the distribution of suffixes following state s_1 is completely non-overlapping with the distribution of suffixes following state s_2. With this intuition, we define the *compliance over the boundary* as the compliance rate of the model over distinguishing suffixes.

[2] Vafa et al. [9] offer an alternative definition of this same term. Theirs is asymmetrical in that it defined the Myhill-Nerode boundary as the set of states accepted after s_1 but not s_2. In our definition, it is always the case that $B(s_1, s_2) = B(s_2, s_1)$.

Definition 2.2 (Boundary-regularized compliance)**.** Given a generative model m, a DFA A with compliance function $l(x) := x \in L(A)$, and a divergence measure D. For two prefixes s_1, s_2 with $q(s_1) \neq q(s_2)$, the boundary-regularized compliance is given by:

$$C_r(s_1, s_2) = \frac{\sum_{x \in B(s_1,s_2)} P(x|s_1)l(x|s_1) + P(x|s2)l(x|s_2)}{\sum_{x \in B(s_1,s_2)} P(x|s_1) + P(x|s_1)}.$$

where $P_m(x|s_i)$ is the model's probability distribution over suffixes. If the model places zero mass on the Myhill-Nerode boundary or no such boundary exists for suffix length k, define $C_r(s_1, s_2) = 1$ (the model is fully compliant).

DFA Matching We are now in a position to define our notion of DFA matching, which broadly aims to capture a model's ability to distinguish the rules of a DFA and track the state over a sequence of tokens.

Definition 2.3 ((ϵ, δ)-matching)**.** Generative model m (ϵ, δ)-**matches** a DFA A (with respect to a divergence D) if:

1. For all prefixes s_1, s_2 such that $q(s_1) = q(s_2)$, $D(P_m(.|s_1), P_m(.|s_2)) \leq \epsilon$,
2. For all prefixes s_1, s_2 such that $q(s_1) \neq q(s_2)$, the boundary-regularized compliance $C_r(s_1, s_2) \geq \delta$.

This definition is advantageous because it covers both prefix equivalence (the idea that language models should produce outputs as a function of the state, not the path) and suffix distinguishing (the idea that language models should be able to differentiate suffixes accepted by the DFA from suffixes rejected by the DFA).

3. Analysis of an Example

In this section, we analyze an example of a regular language to illustrate our framework. Our language is given by a DFA that accepts strings of the form 0^n1^m – that is, any number of the character '0' followed by any number of the character '1'. We assume there is a *true generating process* and we will use an (imperfect) *language model* to model the generating process.

True generating process. The true sequence-generating process produces samples sequentially using a simple distribution. Before any instance of 1 has occurred (state s_0), the sequence generates tokens from the alphabet uniformly at random: $P(0|s_0) = P(1|s_0) = 0.5$. After the first 1 has occurred (state s_1), the sequence generates only 1 tokens from the alphabet to remain compliant with the DFA: $P(0|s_1) = 0, P(1|s_1) = 1$. The DFA specifying the acceptable sequences is visualized in Figure 1.

Language models. We evaluate a possibly *imperfect* model m that generates sequences one token at a time. We will use two models: `penultimate` and `bigram`.

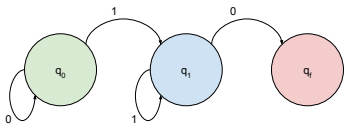

Figure 1. Markov process corresponding to the language's rules. The language is regular because its acceptable sequences can be computed deterministically and with finite memory via the state diagram.

`penultimate` uses the *second to last* token in the prefix to determine the next token, and `bigram` uses the *last token* that appears in the prefix to determine the next token. We evaluate the model on all valid prefixes of length 2 ('00', '01', '11') and all suffixes of length 2 ('00', '01', '10', '11'). We use the Total Variation (TV) distance as our divergence metric.

Model family 1: `penultimate`. Let's start with a particular language model to attempt to emulate the given language. In this family of language models, the second to last token is used to predict the next token. The model's full probability distribution is parameterized as follows:

$$P(0|*0\,.) = 1 - c, \quad P(1|*0\,.) = c,$$
$$P(0|*1\,.) = d, \quad P(1|*1\,.) = 1 - d.$$

The boundary-regularized compliance rate is computed between any pair of prefixes that land in different states on the DFA. In the current case, it is easy to observe that there are two prefix pairs of length 2 that arrive at different states: ('00', '01') and ('00', '11').

This setting is sufficiently simple that we can arrive at closed-form expressions for the δ and ϵ values for different parameter values (c, d). The expressions are given below and are visualized in Figure 3.

Proposition 3.1. *The value of ϵ for the `penultimate` model is given by:*

$$\epsilon = |1 - c - d|.$$

Proposition 3.2. *The value of δ for the `penultimate` model is given by:*

$$\delta = \min\left\{\frac{1}{2}, \frac{1-c}{1-c+d}\right\}.$$

Model family 2: `bigram`. Given that our first model (`penultimate`) was unable to match the DFA in a satisfactory manner, we introduce a new model here that we hope will perform better. In our new model, the conditional probabilities of next tokens are given by the following pa-

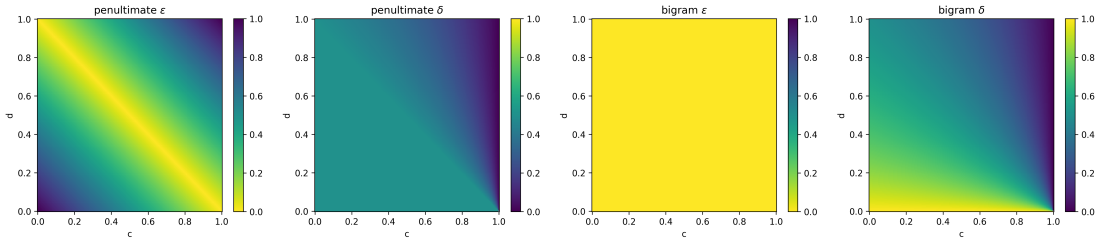

Figure 2. The values of ϵ and δ for the `penultimate` models (left two) and the `bigram` models (right two), for all feasible parameters c, d in each model family. In the `penultimate` model, there is no non-trivial combination of parameters for which $\epsilon = 0, \delta = 1$, meaning that the there is no model that perfectly matches the DFA. Further, the values that maximize δ diverge from the values that minimize ϵ, meaning there exists a trade-off between achieving between-state compliance and within-state consistency. The `bigram` model with $c = 0, d = 0$, however, does achieve $\epsilon = 0, \delta = 1$, and no such trade-off exists.

rameterization:

$$P(0|*0) = 1 - c, \quad P(1|*0) = c,$$
$$P(0|*1) = d, \quad P(1|*1) = 1 - d.$$

Again, we are able to solve the values of (ϵ, δ) as a closed-form function of the model parameters (c, d).

Proposition 3.3. *For the* `bigram` *model,*

$$\epsilon = 0.$$

This result is intuitive because the bigram model has the correct state model for the DFA – the distribution of next token depends on the state, which has a direct mapping to the most recent token.

Proposition 3.4. *For the* `bigram` *model,*

$$\delta = \frac{1 - c}{1 - c + d}.$$

This model is intuitively the better fit given the true data-generating process, and so naturally, we see that there exist more favorable combinations of DFA-matching values ϵ and δ. If we can use an empirical process to arrive at the optimal combinations of $(c = 0, d = 0)$, we could say that our bigram model (ϵ, δ)-matches the DFA with values $\epsilon = 0, \delta = 1$.

4. Empirical Tests

Here we report the results of empirical tests where we train real language models on known distributions, and measure their outputs' comportment with the rules of the language. We start by introducing four languages which we will use to generate sequences for training and evaluating state-of-the-art language models.

True generating processes. Like the previous section, we consider a set of data-generating processes which produce sequences of tokens in a two-token alphabet $\Sigma = (0, 1)$. Generally, under 'normal' circumstances, the language draws uniformly at random from these values. However, depending on the rules of the particular language, there are certain instances where we expect the sequence to exhibit certain behavior, which we describe in detail below.

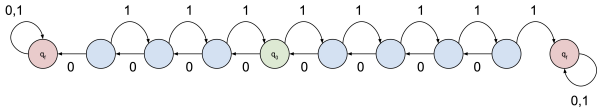

Figure 3. A DFA specifying the rules of a language with a 2-token alphabet ('0','1'), where the starting state can be represented as the integer 3, and each token corresponds to navigating -1 or $+1$ from the current position. The DFA rejects sequences that pass the boundaries of $\{0, 7\}$.

Constrained random walk. Imagine the sequences generated by our language(s) are a set of instructions for an agent that finds herself on the number line. Assume she can only hop between the integers. She starts at some position (call it q_0, and assume $q_0 = 3$ unless otherwise specified). If the language has no rules, she is simply doing a random walk on the number line. As a simple set of rules, however, we can imagine that there are one or two *boundaries* over the set of integers that constrain the agent's path. We will analyze two cases – a *one-sided* bound at 0 and a *two-sided* bound at $\{0, 7\}$. The DFA corresponding to the two-sided constrained random walk is given in Figure 3.

Stopping conditions. We will test two different forms of this model. In the first, `reflect`, the stopping condition between sequences is given by a maximum sequence length, n, which we will assume is 32. In the true data-generating process with this stopping condition, if the random walk reaches the boundary, it will 'reflect' and always return the token corresponding to navigating (legally) away from the boundary, rather than crossing it. In the second of our stopping conditions, `absorb`, the sequence ends only when

the sequence reaches the boundary, and so sequences can reach long lengths before terminating.

	Bounds	Boundary Behavior	Max. Length	Language
1	$\{0, 7\}$	Reflect	32	Regular (DFA)
2	$\{0, 7\}$	Absorb	∞	Regular (DFA)
3	$\{0, \infty\}$	Reflect	32	Regular (DFA)
4	$\{0, \infty\}$	Absorb	∞	Non-regular

Table 1. Overview of the four language settings used to evaluate language model compliance. Settings vary in the type of boundaries imposed on the token-based random walk, the boundary behavior at those limits, and the maximum sequence length. The regularity of each language determines whether its rules can be captured by a finite-state machine. Though we set up scenarios 2 and 4 to be possibly infinite, when we implement these languages programatically, for practical reasons we put a high limit (500) on the sequence size. Thus, technically, scenario 4 is regular in the implementation because it can be represented by a massive DFA.

Training transformer models. Now that we have specified a set of basic languages that we can treat as the true distribution over next tokens, we can train transformer models to emulate these distributions and evaluate the extent to which the trained models comply with the rules of our languages. Further details are provided in Appendix B. For each scenario, we generate datasets of 100,000 tokens emulating the true distribution, and train a 0.2M parameter transformer.

Output Compliance. In Figure 6, we depict the absolute compliance rate for each of our four training scenarios. The results suggest that the trained models reach high levels of compliance after about 1000 iterations, in the first three settings listed in Table 1. The final setting (4), which uses a language that cannot be expressed as a DFA, does not reach the same level of compliance. Of course, these absolute compliance rates represent coarse understandings of the extent to which a model has learned a DFA, since they depend on how sparse the rejected suffixes are in the total output space. As we develop our experimental setting further, we hope to formally characterize values of ϵ and δ to see how these transformers match the DFA underlying the rules of their language.

5. Conclusion

In this workshop paper, we have described ongoing work which aims to provide a framework for evaluating the extent to which a language model adheres to the rules of a language. Leveraging formal language theory, we consider rules that can be expressed as a DFA, and define the notions of (ϵ, δ)-matching and boundary-regularized compliance. We also set up an experimental framework where we can reason about rule-adherence in transformer models.

References

[1] Angluin, D. Inductive inference of formal languages from positive data. *Information and control*, 45(2): 117–135, 1980.

[2] Gold, E. M. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[3] Kleinberg, J. and Mullainathan, S. Language generation in the limit. *Advances in Neural Information Processing Systems*, 37:66058–66079, 2024.

[4] Laufer, B., Gilbert, T., and Nissenbaum, H. Optimization's neglected normative commitments. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pp. 50–63, 2023.

[5] Laufer, B., Kleinberg, J., and Heidari, H. The backfiring effect of weak ai safety regulation. *arXiv preprint arXiv:2503.20848*, 2025.

[6] Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

[7] Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

[8] Toshniwal, S., Wiseman, S., Livescu, K., and Gimpel, K. Chess as a testbed for language model state tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 11385–11393, 2022.

[9] Vafa, K., Chen, J., Rambachan, A., Kleinberg, J., and Mullainathan, S. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37:26941–26975, 2024.

[10] Wallach, H., Desai, M., Pangakis, N., Cooper, A. F., Wang, A., Barocas, S., Chouldechova, A., Atalla, C., Blodgett, S. L., Corvi, E., et al. Evaluating generative ai systems is a social science measurement challenge. *arXiv preprint arXiv:2411.10939*, 2024.

A. Algorithms for Arriving at ϵ, δ

A.1. Computing ϵ

Algorithm 1 Computing ϵ: Max divergence between prefixes reaching the same DFA state

Require: Language model m, DFA state function $q(s)$, prefix length n, suffix length k

Generate all prefixes s of length n over the alphabet Σ
For each prefix s, compute its DFA state $q(s)$
Initialize $\epsilon \leftarrow 0$
for all groups of prefixes that share the same state **do**
  for all prefix pairs (s_1, s_2) in the group **do**
    Query the model for $P_m(x \mid s_1)$ and $P_m(x \mid s_2)$ over all $x \in \Sigma^k$
    Compute the divergence: $D \leftarrow D(P_m(. \mid s_1), P_m(. \mid s_2))$
    Update: $\epsilon \leftarrow \max(\epsilon, D)$
  end for
end for
Return ϵ

A.2. Computing δ

Algorithm 2 Computing δ: Min Boundary-Regularized Compliance between distinct DFA states

Require: Language model m, DFA state function $q(s)$, legality function $l(x \mid s)$, prefix length n, suffix length k

Generate all prefixes s of length n over the alphabet Σ
For each prefix s, compute its DFA state $q(s)$
Initialize $\delta \leftarrow 1$
for all pairs of prefixes (s_1, s_2) such that $q(s_1) \neq q(s_2)$ **do**
  Query the model for $P_m(x \mid s_1)$ and $P_m(x \mid s_2)$ over all $x \in \Sigma^k$
  Initialize Numerator $\leftarrow 0$, Denominator $\leftarrow 0$
  for all suffixes $x \in \Sigma^k$ **do**
    if $l(x \mid s_1) \neq l(x \mid s_2)$ **then**
      Numerator $+= P_m(x \mid s_1) \cdot l(x \mid s_1) + P_m(x \mid s_2) \cdot l(x \mid s_2)$
      Denominator $+= P_m(x \mid s_1) + P_m(x \mid s_2)$
    end if
  end for
  $\mathrm{C}_r \leftarrow$ Numerator/Denominator
  Update: $\delta \leftarrow \min(\delta, \mathrm{C}_r)$
end for
Return δ

B. Details of Transformer Implementation

Here we provide additional details about the transformers we trained. We used Karpathy's minGPT implementation with the following hyperparameters:

- Training dataset size: 50000
- Size of sequences: 20
- batch size = 16
- block size = 32
- max iters = 5000

- eval interval = 100
- learning rate = 1e-3
- eval iters = 200
- n_embd = 64
- n_head = 4
- n_layer = 4
- dropout = 0.0

C. Examples of output paths

Here we provide some examples of output paths to give a sense of the distribution of outputs for a transformer trained in setting 1. The results are depicted in Figure 4. Another example is depicted in Figure 5.

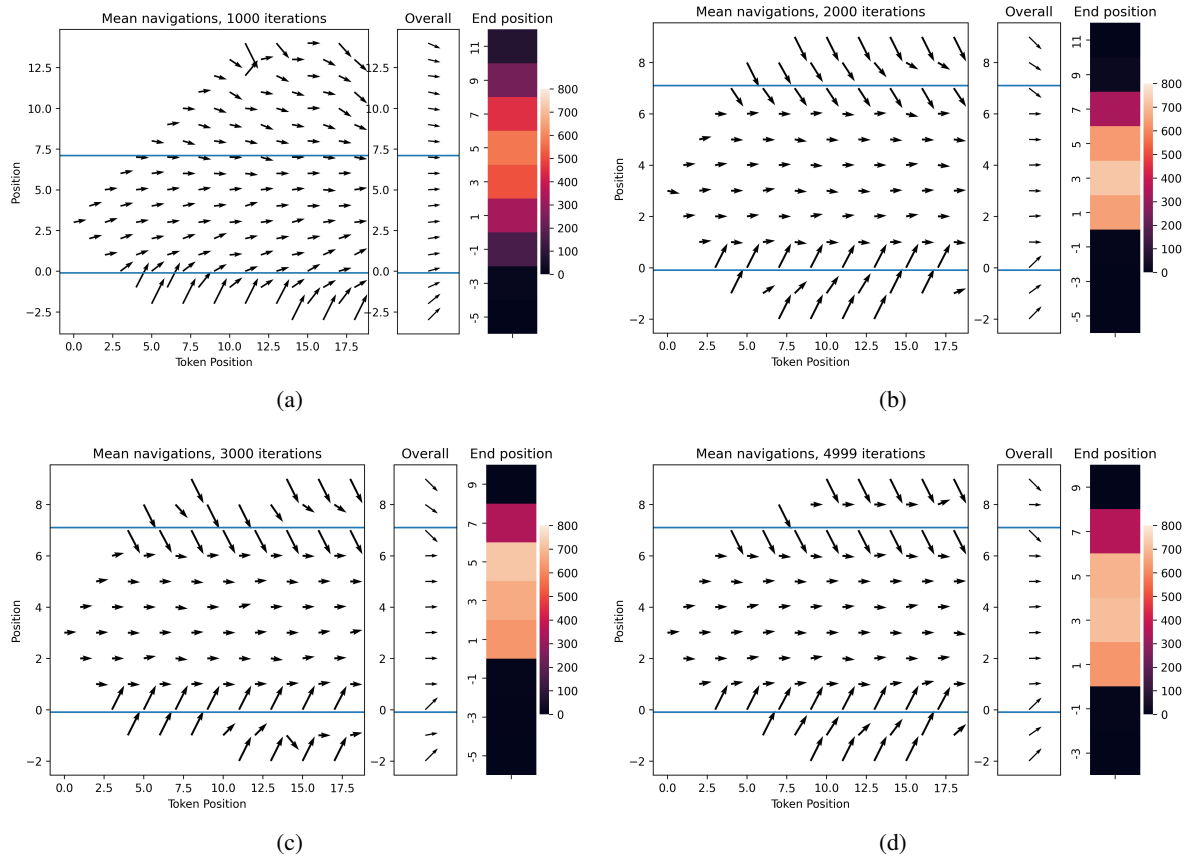

Figure 4. Output paths for testing scenario 1. After a small number of training iterations, the model breaks the rule often and seems to perform a more general 'mean reversion.' After more training iterations, the model more closely resembles the data-generating process, but still exhibits some 'safe' behavior where at position 6 the average behavior is to move towards the mean, even though the true data-generating process produces 0 and 1 with equal probabilities.

D. Compliance Results

Here we provide results on the rate of compliance in the output generated by our trained transformer models. The results are depicted in Figure 6.

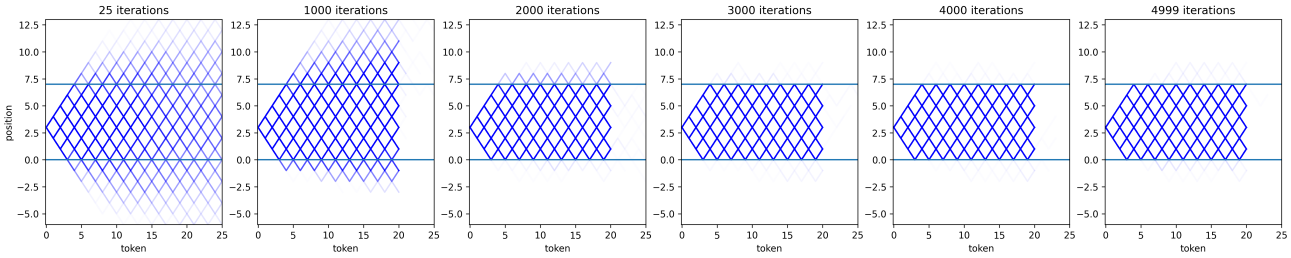

Figure 5. Sequences output by transformer model at different training stages.

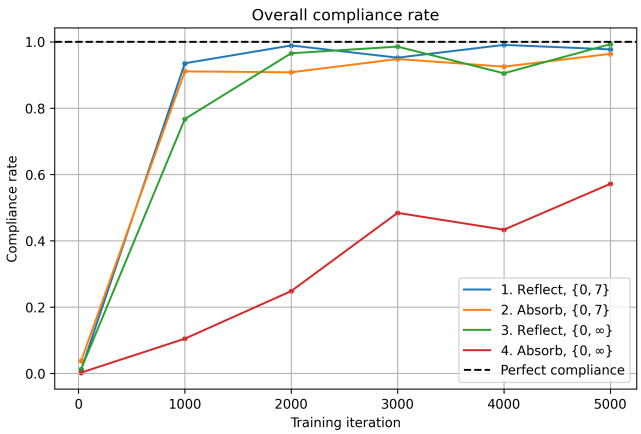

Figure 6. Compliance rates for different trained transformers, in our four training scenarios (listed in Table 1).