Polynomial-Time Mistake-Bounded Language Generation

Héctor Jimenez
University of Chile
hjimenez@dim.uchile.cl

Alexander Kozachinskiy
CENIA
alexander.kozachinskyi@cenia.cl

Vicente Opazo
CENIA
vicente.opazo@cenia.cl

June 15, 2026

Abstract

In this note, we introduce a polynomial-time version of the mistake-bounded language generation (MBLG) framework due to Kleinberg, Peale, and Reingold (2026). We observe that the family of parities of variables, and the family of conjunctions of literals, are polynomial-time MBLG. Our main result states that the family of monotone Boolean functions with polynomially-many maxterms is polynomial-time MBLG. This family includes all monotone Boolean functions, computable by polynomial-size decision trees. Our technique can be presented as a new combinatorial game about writing numbers on a board.

1 Introduction

In the era of LLMs, a part of theoretical research in machine learning seeks to adapt its old models to the new reality. One example of this is a recent *language generation in the limit* model of Kleinberg and Mullainathan [8]. It builds upon a classical language identification in the limit model due to Gold [5], where a learner gets exposed to a stream of words of some unknown formal language from a language family $\mathcal{F}$. The task is to identify this language eventually, where the guess can be updated with each new word from the stream. In contrast, in the Kleinberg–Mullainathan model, the task is to simply output a word from the language not seen so far in the stream. For every language in the family and for every possible enumeration of it, there has to be a moment, starting from which we get this task right. This mild change, initially motivated by the practice of using LLMs as text generators, leads to a significantly richer mathematical theory. Families, identifiable in the limit, have been characterized by Angluin [2]. However, we are aware only of one "interesting" family, satisfying these conditions – namely, the family of pattern languages from Angluin's earlier paper [1]. Meanwhile, Kleinberg and Mullainathan have shown that *every* countable family is generatable in the limit. This result sparked a growing interest in the model, with subsequent works establishing the result for generators, satisfying additional desirable properties like breadth [10, 6] and representativity [12].

However, it was observed that if one measures success in the number of words needed to make sure that the generation is correct, the language generation model becomes rather infeasible. Imagine we have just two languages in the family with a finite but very large intersection. If the stream starts with some enumeration of this intersection, we have no way to avoid a mistake when we get the last word of the intersection. Motivated by this observation, Arenas et al. [3] show lower bounds on the size of the maximal finite intersection for a number of natural language families. For instance, they show there is no computable bound on the size of the intersection of two context-free languages whose intersection is finite (that is, there is no algorithm that, given two context-free languages L_1, L_2 such that $L_1 \cap L_2$ is finite, returns some upper bound on $|L_1 \cap L_2|$).

In a recent preprint, Kleinberg, Peale, and Reingold [9] suggest that a better way of measuring success is not the time until the last mistake, but the total number of mistakes, no matter how far the last one is. Observe that in the example, with two languages, it is easy to make at most 1 mistake. Indeed, if we see something outside the intersection, we already know the true language, and if we only see words from the intersection, we can always give another example from the intersection unless all of them are exhausted, which can happen at most once. Generalizing this example, Kleinberg, Peale, and Reingold give a beautiful algorithm that, for finite families of size s, makes at most $\lceil \log_2 s \rceil$ mistakes. Moreover, for countable families $\mathcal{F}$ they give an algorithm that for the i-th language of the family makes at most $\lceil \log_2 i \rceil$ mistakes.

In this note, we introduce a polynomial-time version of the mistake-bounded language generation (MBLG) framework of Kleinberg, Peale, and Reingold, in a style similar to Valiant's polynomial-time PAC learning [7] and Littlestone's online (a.k.a. mistake-bound) learning [11]. Let us first recall these models. Both start with a $2^{poly(n)}$-size family $\mathcal{F}$ of Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ (or equivalently, of languages of binary words of length n where these functions take value 1). In the PAC-model, the adversary secretly picks $f \in \mathcal{F}$ and a distribution D over $\{0,1\}^n$. The learner can sample pairs $(x, f(x))$ with $x \sim D$. It gets an input $y \in \{0,1\}^n$, sampled from D, with an unknown value of f, and its goal is to predict $f(y)$ with probability of error at most ε after working $poly(n/\varepsilon)$-time.

In contrast, the online learning model is iterative: the adversary first picks $f \in \mathcal{F}$, and then, in rounds, it picks $x \in \{0,1\}^n$ and asks the learner – "what is the value of $f(x)$?". After the learner takes its guess, the adversary reveals the correct value and proceeds to the next round. The goal of the learner is to make at most $poly(n)$ mistakes (independently of the number of rounds), working in polynomial time (in n and the number of rounds so far).

Similarly, in polynomial-time MBLG that we introduce here, there is a family

$$\mathcal{F} = \{L_1, \dots, L_s\}, \qquad L_1, \dots, L_s \subseteq \{0,1\}^n$$

of languages of binary words of length n, with $s = 2^{poly(n)}$. The adversary secretly picks a language $L \in \mathcal{F}$, and starts printing its words in some order (the words have to be distinct). For each $i = 0, \dots, |L|-1$, seeing the first i printed words $x^1, \dots, x^i$, our goal is to output some word $y^i \in L \setminus \{x^1, \dots, x^i\}$. We have to make at most $poly(n)$ mistakes, working in polynomial time – more precisely, the time to produce y^i from $x^1, \dots, x^i$ should be $poly(ni)$. Families for which this is possible will be called polynomial-time MBLG families. Note that in contrast to online learning, in the mistake-bounded generation model there is no feedback – we never get any separate confirmation whether y^i was indeed from L.

In all 3 models, if we forget about polynomial-time requirements, any $2^{poly(n)}$-size family becomes learnable. Indeed, in PAC learning, it is enough to sample $p(\varepsilon, \log_2 |\mathcal{F}|)$ pairs $(x, f(x))$ for some fixed polynomial p, pick any $\widehat{f} \in \mathcal{F}$ that is consistent with all these pairs, and predict according to this $\widehat{f}$ [13]. In turn, in the online learning, the "halving algorithm" of Littlestone will work – predict according to the majority of functions from $\mathcal{F}$ that were never wrong so far (each mistake decreases the number of potential functions by a factor of 2, and overall we get at most $\log_2 |\mathcal{F}|$ mistakes).

Note that these solutions require explicitly going through all functions in $\mathcal{F}$, and thus are not implementable in polynomial time. A similar situation is with the algorithm of Kleinberg, Peale, and Reingold [9] for the mistake-bounded language generation. Its overall structure is as follows: it assigns weights to every function (language) of $\mathcal{F}$, with all the weights being initially equal. Each time, among the words that have not been printed so far, we choose one that maximizes the sum of weights of languages that contain it. The weights of languages are updated after each new printed example from the adversary. Once again, although the algorithm guarantees at most $\log_2 |\mathcal{F}|$ mistakes, it requires explicitly going through all languages in the family, which is not implementable in polynomial time in our setting.

In this note, after formalizing polynomial-time MBLG, we give a few initial examples of families that are learnable in this model. We start by observing that the family of parities of variables, and the family of conjunctions of literals, are polynomial-time MBLG. We then establish our main result, which concerns monotone Boolean functions (a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ in our framework is identified with

the language of binary words of length n where f takes value 1). A *maxterm* of a Boolean function f is a maximal input where it takes value 0 (any further switching of an input bit from 0 to 1 leads to value 1). We show that for any polynomial $p(n)$, the family $\mathcal{F}$ of monotone Boolean functions with at most $p(n)$ maxterms is polynomial-time MBLG.

For instance, this result includes disjunctions of variables – these are monotone Boolean functions with exactly one maxterm. Further, one can notice that any monotone Boolean function, computable by a polynomial-size decision tree, has $poly(n)$ maxterms, namely, at most the number of 0-leaves of the tree. Indeed, every maxterm ends up in some 0-leaf, so it has to be consistent with the variables and the values that are asked on the path to this leaf, and on the variables that are not asked, it has to be equal to 1 – otherwise one can switch some input bit from 0 to 1 without changing the value.

We conclude the paper with some open questions.

2 Polynomial-time MBLG

Formally, we have to work with infinite sequences $\{\mathcal{F}_n\}_{n=1}^{\infty}$ of families,where $\mathcal{F}_n = \{L_1, L_2, \ldots, L_{s(n)}\}$ and $L_1, L_2, \ldots, L_{s(n)} \subseteq \{0,1\}^n$.

A *generator* is a function G that takes on input a natural number n (written in unary) and a sequence of distinct binary words of length n, and returns a binary word of length n.

Definition 1. *Let $\{\mathcal{F}_n\}_{n=1}^{\infty}$ be a sequence of families of languages, where $\mathcal{F}_n$ consists of languages of binary words of length n for all n. Further, let G be a generator and $e\colon \mathbb{N} \to \mathbb{N}$ be a function.*

We say that G makes at most $e(n)$ mistakes on $\{\mathcal{F}_n\}_{n=1}^{\infty}$ if for every n, for every $L \in \mathcal{F}$, and for every $x^1, \ldots, x^{|L|} \in \{0,1\}^n$ such that

$$L = \{x^1, \ldots, x^{|L|}\},$$

there exists at most $e(n)$ indices $i \in \{0, 1, \ldots, |L|-1\}$ such that:

$$G(1^n, x^1 x^2 \ldots x^i) \notin L \setminus \{x^1, \ldots, x^i\}.$$

Definition 2. *Let $\{\mathcal{F}_n\}_{n=1}^{\infty}$ be a sequence of families of languages, where $\mathcal{F}_n$ consists of languages of binary words of length n for all n, and $|\mathcal{F}_n| = 2^{poly(n)}$. If this sequence of families admits a polynomial-time generator, making $poly(n)$ mistakes, we say that the family is **polynomial-time MBLG**.*

In the results below, to increase readability, we no longer explicitly consider sequences of families $\{\mathcal{F}_n\}_{n=1}^{\infty}$. Instead, we simply say "$\mathcal{F}_n$ is polynomial-time MBLG", implicitly assuming that n is a parameter, going to infinity, yielding a sequence of families.

3 Conjunctions and Parities

For a Boolean variable x_i and $\alpha \in \{0,1\}$, denote:

$$x_i^{\alpha} = \begin{cases} x_i & \alpha = 1, \\ \neg x_i & \alpha = 0. \end{cases}$$

Proposition 3. *Let $Conj[(i_1, \alpha_1), \ldots, (i_k, \alpha_k)]$ be a language of all words $x \in \{0,1\}^n$ such that $x_{i_1}^{\alpha_1} \wedge \ldots \wedge x_{i_m}^{\alpha_m} = 1$. Then the family*

$$\mathcal{F} = \{Conj[(i_1, \alpha_1), \ldots, (i_k, \alpha_k)] : \qquad k \geq 0, \qquad 1 \leq i_1 < \ldots < i_k \leq n, \qquad \alpha_1 \ldots \alpha_k \in \{0,1\}^k\}$$

is polynomial-time MBLG.

Proof. Given an input sequence $(x^1, \ldots, x^i)$, we find the set P of all positions where all words $x^1, \ldots, x^i$ coincide. We know P includes the unknown positions $i_1, \ldots, i_k$, with the right values $\alpha_1, \ldots, \alpha_k$, but maybe some other positions too. We try to find another word that has the same values in positions from P as $(x^1, \ldots, x^i)$ – it would be a safe guess to make. What can happen, however, is that all such words are already within $x^1, \ldots, x^i$. But then the next word of the adversary will remove some position from P, so this can happen at most n times.

Note that this strategy is implementable in polynomial time in the length of $(x^1, \ldots, x^i)$. Constructing P is straightforward, and then we enumerate all words that have the required values in positions from P, in search of one that differs from all $x^1, \ldots, x^i$. This will take us at most i steps of enumeration.

□

Proposition 4. *Let $Par[i_1, \ldots, i_k]$ be the language of all words $x \in \{0,1\}^n$ such that $x_{i_1} \oplus \ldots \oplus x_{i_k} = 0$. The family $\mathcal{F} = \{Par[i_1, \ldots, i_k] \mid 1 \le i_1 < \ldots < i_k \le n\}$ is polynomial-time MBLG.*

Proof. When we have examples $x^1, \ldots, x^i$, it is safe to output any linear combinations (modulo 2) of these vectors. The only way it is not possible to output a fresh string like that is when the set $\{x^1, \ldots, x^i\}$ coincides with its own linear span. But then the next example of the adversary will increase the dimension of the span, so we will have at most n mistakes.

To implement this in polynomial time, we find a maximal independent subset of $\{x^1, \ldots, x^i\}$, and then start enumerating all words that can be obtained as its linear combinations. It takes at most i steps to find a new word because every word out of $x^1, \ldots, x^i$ can cause us trouble just once. □

A similar proof works when one considers the condition $x_{i_1} \oplus \ldots \oplus x_{i_k} = 1$, but one has to take the affine span.

4 Monotone Functions with Polynomially Many Maxterms

We formulate our main result in terms of languages. We consider the inclusion order on the set of subsets of $\{1, 2, \ldots, n\}$, viewing it as an order on $\{0,1\}^n$ and assuming that a string $x \in \{0,1\}^n$ is identified with the set of positions where it is equal to 1. A language $L \subseteq \{0,1\}^n$ is monotone if it is upwards-closed in this order. A maxterm of a monotone language L is a maximal element of $\{0,1\}^n \setminus L$.

Theorem 5. *For any fixed polynomial $p(n)$, the family of monotone languages $L \subseteq \{0,1\}^n$ that have at most $p(n)$ maxterms is polynomial-time MBLG.*

Proof. Imagine that words, printed by the adversary, get marked. Each time, we have to output a non-marked word. Since L is monotone, it is safe to output any non-marked word if below it (in the inclusion order) there is some marked word. So we only have to care about "crucial" moments when the set of marked words is upwards-closed.

In these moments, we will always output some maximal non-marked word. We make a mistake when we output a word from $\{0,1\}^n \setminus L$. Thus, we can only make a mistake when we output a maxterm of L. To make sure that we make at most $poly(n)$ mistakes, we will give an algorithm such that every word $x \in \{0,1\}^n$ gets outputted at most $O(n)$ times at crucial moments.

We will achieve this via the following simple strategy. We maintain the set M of maximal non-marked elements. Note that the size of M is polynomial in the bit-length of $(x^1, \ldots, x^i)$ because every its element has some string from $\{x^1, \ldots, x^i\}$ directly above it, but every string $\{x^1, \ldots, x^i\}$ can have at most n elements of M directly below it. Further, we maintain how many times each element of M has been given to the output at crucial moments. When there is a crucial moment, we choose any element of M that has been given the minimal number of times.

Let us study how the set M evolves at crucial moments. The key observation is that each time, at least one element of M goes away (and maybe some new elements are added). Moreover, elements that go away never come back. Indeed, in the crucial moment, all strings under any element of M are non-marked (because marked strings are upwards-closed). Hence, if all elements of M would have survived, the set of

non-marked elements would increase or stay the same. However, this set decreases over time. Moreover, an element $m \in M$ can go away only if m or something below it gets a mark, and then it can never enter M again.

Since strings never return to M, and since the size of M is polynomial, maintaining M together with the number of times its elements have been given as an output takes polynomial time.

Next, these numbers of times – how many times each element of M has been used at crucial moments – satisfy the rules of the following game with numbers written on a board:

- initially, one can write any number of 0's on the board (this corresponds to the first crucial moment when the set M is created, and no string has been given as the output yet).
- then you work in steps. At each step, you first increase the minimal number on the board by 1 (if there are multiple minimal numbers, you increase just one). Then at least one number has to be erased from the board, and then some new zeroes can be added.

In our application, the number of steps is bounded by the number of crucial moments, which is at most 2^n. It now suffices to establish the following lemma.

Lemma 6. *Let $n \geq 1$. Then the minimal number of steps for a number n to appear on the board in the game with numbers is at least 2^{n-1}.*

Proof. Imagine numbers as stacks of coins. Somewhat strangely, a number m will be represented as a stack of $m+1$ coins – a level-0 coin, a level-1 coin, ..., a level-m coin. Coins at different levels will have different costs. Namely, a level-0 coin has cost 1, and a level-i has cost 2^{i-1} for $i \geq 1$.

Look at the total cost of all the coins. When we add 0, the total cost increases by 1. Now, when we add $+1$ to a number m, we create a level-$(m+1)$ coin whose cost is 2^m. But then we either delete it right away, or we delete some other number that has to be at least m. The cost of all the coins that we delete is at least $1 + 1 + 2 + \ldots + 2^{m-1} = 2^m$, so the total cost does not increase.

Hence, to create a coin at level n, we have to put at least 2^{n-1} 0-level coins. Although we can put a lot of 0-level coins at a time, each 0-level coin can be converted into a 1-level coin just one at a time, so we require at least 2^{n-1} steps. □

5 Conclusion

It is interesting to see if our main result can be extended to non-monotone functions. For instance, so far it is not clear to us whether disjunctions of literals are polynomial-time MBLG. More generally, it is interesting to figure out whether polynomial-size decision trees (not necessarily monotone) are polynomial-time MBLG.

A question that does not seem clear is whether polynomial-time MBLG families are closed under union.

Finally, it is interesting to figure out the relationship of our notion with polynomial-time PAC and online learning. It is known that every polynomial-time online learnable family is polynomial-time PAC learnable, but the reverse direction is not always true [4].

□

References

[1] Angluin, D. Finding patterns common to a set of strings. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing* (1979), pp. 130–141.

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

[3] Arenas, M., Barceló, P., Cofré, L., and Kozachinskiy, A. Language generation: Complexity barriers and implications for learning. *arXiv preprint arXiv:2511.05759* (2025).

[4] Blum, A. L. Separating distribution-free and mistake-bound learning models over the boolean domain. *SIAM Journal on Computing 23*, 5 (1994), 990–1000.

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

[6] Kalavasis, A., Mehrotra, A., and Velegkas, G. On the limits of language generation: Trade-offs between hallucination and mode-collapse. In *Proceedings of the 57th annual ACM Symposium on Theory of Computing* (2025), pp. 1732–1743.

[7] Kearns, M., Li, M., Pitt, L., and Valiant, L. On the learnability of boolean formulae. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), pp. 285–295.

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

[9] Kleinberg, J., Peale, C., and Reingold, O. Mistake-bounded language generation. *arXiv preprint arXiv:2605.10809* (2026).

[10] Kleinberg, J., and Wei, F. Density measures for language generation. In *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)* (2025), pp. 620–658.

[11] Littlestone, N. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning 2*, 4 (1988), 285–318.

[12] Peale, C., Raman, V., and Reingold, O. Representative language generation. In *Proceedings of the 42nd International Conference on Machine Learning* (13–19 Jul 2025), A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, Eds., vol. 267 of *Proceedings of Machine Learning Research*, PMLR, pp. 48518–48541.

[13] Valiant, L. G. A theory of the learnable. *Communications of the ACM 27*, 11 (1984), 1134–1142.