Proceedings of Machine Learning Research vol 291:1–34, 2025 38th Annual Conference on Learning Theory

Exploring Facets of Language Generation in the Limit

Moses Charikar MOSES@CS.STANFORD.EDU
Stanford University

Chirag Pabbaraju CPABBARA@CS.STANFORD.EDU
Stanford University

Editors: Nika Haghtalab and Ankur Moitra

Abstract

The recent work of Kleinberg and Mullainathan (2024) provides a concrete model for language generation in the limit: given a sequence of examples from an unknown target language, the goal is to generate new examples from the target language such that no incorrect examples are generated beyond some point. In sharp contrast to strong negative results for the closely related problem of language identification, they establish positive results for language generation in the limit for all countable collections of languages. Follow-up work by Li, Raman, and Tewari (2024) studies bounds on the number of distinct inputs required by an algorithm before correct language generation is achieved — namely, whether this is a constant for all languages in the collection (uniform generation) or a language-dependent constant (non-uniform generation).

We show that every countable collection has a generator with the stronger property of non-uniform generation in the limit. However, while the generation algorithm of Kleinberg and Mullainathan (2024) can be implemented using membership queries, we show that any algorithm cannot non-uniformly generate even for collections of just two languages, using only membership queries.

We also formalize the tension between validity and breadth in the generation algorithm of Kleinberg and Mullainathan (2024) by introducing a definition of *exhaustive* generation, and show a strong negative result for exhaustive generation. Our result shows that a tradeoff between validity and breadth is inherent for generation in the limit. We also provide a precise characterization of the language collections for which exhaustive generation is possible. Finally, inspired by algorithms that can choose to obtain feedback, we consider a model of uniform generation with feedback, completely characterizing language collections for which such uniform generation with feedback is possible in terms of an abstract complexity measure of the collection.

Keywords: Language Generation in the Limit, Non-uniform Generation, Validity-breadth Tradeoff

1. Introduction

Consider the following algorithmic problem: given an infinite stream of strings from an unknown target language (one of a known collection), generate new and previously unseen strings also belonging to this target language, eventually making no mistakes. Kleinberg and Mullainathan (2024) recently formalized this, aiming to capture the core problem at the heart of large language models. The same problem, but with the goal of *identifying* the target language in the collection instead of simply generating from it, has been extensively studied in classical work on language identification in the limit (Gold, 1967; Angluin, 1979, 1980). In fact, we have a precise characterization (Angluin, 1979, 1980) of the collections of languages for which this problem is tractable. The main message of these works is that language identification is intractable for any interesting collection of formal languages (e.g., regular languages), even with unbounded computational power.

Despite such strong negative results for language identification, Kleinberg and Mullainathan (2024) showed a surprisingly strong positive result for language *generation*, i.e. that this is tractable for *every* countable collection $\mathcal{C}$ of languages! Their work gives a simple and elegant algorithm for this generation task, and further show that this algorithm can be implemented with only membership query oracle access to the collection $\mathcal{C}$. I.e., the algorithm simply needs to ask queries of the form "does w belong to L_i?" for any string w in the universe and language $L_i \in \mathcal{C}$.

How quickly can we hope to achieve this guarantee of generation in the limit? Kleinberg and Mullainathan (2024) give an algorithm that achieves a stronger guarantee for generation in the limit from *finite* collections: on seeing $t^* = t^*(\mathcal{C})$ many distinct strings, the algorithm correctly generates from the target language. (Here t^* depends neither on the target language nor its enumeration order). This algorithm is quite different from their all-purpose algorithm which achieves generation in the limit for all countable collections. Inspired by this, the very recent work of Li, Raman, and Tewari (2024) sought a precise characterization of the collections of languages with such stronger guarantees. Adopting their terminology, the collections for which $t^* = t^*(\mathcal{C}, L)$ may depend on the target language L, but not on its enumeration order, are said to be *non-uniformly generatable*, whereas the collections for which $t^* = t^*(\mathcal{C})$ is a function of the collection $\mathcal{C}$, and depends neither on the target language L nor its enumeration order, are said to be *uniformly generatable*. Li et al. (2024) show that uniformly generatable collections are exactly those that have bounded *closure dimension* – a complexity measure they introduce. They leave open the characterization of non-uniformly generatable collections.[1]

A different line of inquiry, also motivated from the work of Kleinberg and Mullainathan (2024), stems from a tradeoff between *validity* and *breadth* in their (general-purpose) generation algorithm The algorithm starts off generating invalid strings, refining its hallucinations, eventually settling to generate from an increasingly small subset of the language. En route to becoming a valid generator, it appears that the algorithm has to sacrifice on generating the entire bulk of the target language. This phenomenon also notoriously shows up in the form of *mode collapse* while training generative adversarial networks (Arjovsky and Bottou, 2022; Arjovsky et al., 2017). Kleinberg and Mullainathan (2024) asked if such a tradeoff is provably necessary for achieving generation in the limit.

1.1. Overview of Results

As our first result, we show that it is possible to *non-uniformly* generate in the limit from every countable language collection $\mathcal{C} = \{L_1, L_2, \ldots\}$. We give a simple algorithm (Section 3), which generates valid, unseen strings from the target language on seeing a fixed constant number of distinct strings in the enumeration, where this constant depends *only* on the target language and the collection, but *not* the enumeration order. Note that Kleinberg and Mullainathan (2024)'s algorithm for generation in the limit does not satisfy this latter property.

Theorem 1 (Non-uniform Generation for Countable Collections (see Theorem 12)) *There exists an algorithm that non-uniformly generates in the limit from every countable collection of languages.*

Li et al. (2024) asked if every countable collection can be non-uniformly generated in the limit; Theorem 1 answers this in the affirmative. The language-dependent constant for the number of

1. This was a question left open in the first version of their manuscript (Raman and Tewari, 2024); see Section 1.2.

distinct strings that our non-uniform generation algorithm must see is formalized in terms of a *non-uniform complexity* measure of the language (see Definition 11 and Theorem 12).

The algorithm of Kleinberg and Mullainathan (2024) has the attractive property that it can be implemented using only *membership queries*, I.e. their generation algorithm can be implemented using only queries of the form "$w \in L_i$?" for any string w and language $L_i \in \mathcal{C}$. One may ask if our non-uniform generation algorithm from above can also be implemented with sole access to such a membership query oracle. Our next result provides a surprisingly strong negative answer.

Theorem 2 (Non-uniform Generation Membership Query Lower Bound (see Theorem 13)) *Membership queries do not suffice to implement an algorithm that non-uniformly generates from countable collections.*

Our lower bound shows that non-uniform generation for collections of just *two* languages is impossible with only membership queries. We note that our result does not contradict the closure-based uniform generation algorithm of Kleinberg and Mullainathan (2024), which can be implemented using membership queries. The key detail is that their algorithm uses additional information about the collection; this information is critical (see Theorem 13 and the discussion at the end of Section 4).

Our next result addresses the open question in Kleinberg and Mullainathan (2024) regarding the tradeoff between validity and breadth in generation. Towards this, we propose a definition of *exhaustive* generation in the limit. This definition still requires an algorithm to eventually always generate from the target language (validity). We have an additional requirement concerning breadth of generation: beyond some finite time, it should be possible to *terminate* the input entirely and continue generating strings indefinitely. Then, the set of strings generated (past and future) when combined with the input seen so far should cover the target language. We formalize these requirements in Definition 10, and show a strong negative result for exhaustive generation.

Theorem 3 (Exhaustive Generation Lower Bound) *There exists a countable collection $\mathcal{C}$ (of regular languages) that cannot be exhaustively generated in the limit.*

Theorem 3 answers the open question of Kleinberg and Mullainathan (2024), showing that the validity-breadth tradeoff is necessary, adding to the evidence in the literature that language models with desirable properties must necessarily hallucinate (Kalai and Vempala, 2024; Xu, Jain, and Kankanhalli, 2024; Banerjee, Agarwal, and Singla, 2024; Wu, Grama, and Szpankowski, 2025).

Our next contribution is a *precise characterization* of the collections of languages that allow for exhaustive generation, similar to that given by Angluin (1980) for identifiability. We show that a *weaker* version of Angluin's condition, which we denote as "Weak Angluin's Condition with Existence" (see (2)), is both necessary and sufficient for exhaustive generation.

Theorem 4 (Exhaustive Generation Characterization (see Propositions 23, 25)) *A countable language collection $\mathcal{C}$ can be exhaustively generated if and only if it satisfies Weak Angluin's Condition with Existence.*

Lastly, we consider uniform language generation with feedback, where the algorithm has the additional ability of asking an oracle whether any string of its choice belongs to the *target* language. This is inspired by *informant* models in the classical work of Gold (1967) for language identification, modeling real-life learning scenarios where an algorithm gets feedback at every time step (e.g., as in RLHF (Christiano et al., 2017)). This is different from the membership query model where the algorithm can only ask the oracle queries regarding a language L_i of its choice, but does not know which language L is the target language. In this model, even *identification* becomes possible for a large class of countable collections (see Table 1 in Gold (1967)). From Theorem 1 above, all countable collections can be non-uniformly generated in the limit (without feedback). The interesting question is whether collections that cannot be uniformly generated without feedback become uniformly generatable with feedback—as Example 1 illustrates, this is indeed true. We completely characterize (Section 6) the collections that can be uniformly generated in this feedback model, in terms of an abstract complexity measure we term the Generation-Feedback (GF) dimension.

Theorem 5 (Uniform Generation with Feedback (see Lemmas 33, 34)) *A collection $\mathcal{C}$ of languages can be uniformly generated with feedback if and only if its* GF *dimension is finite.*

We supplement our results with interesting examples of language collections throughout the paper that provide intuition and elicit differences between different models.

1.2. Related Work

Our work directly builds off of the thought-provoking work of Kleinberg and Mullainathan (2024), which, in addition to introducing and studying language generation in the limit, also studies prompt completion in language models. There have since been a string of follow-ups to this work, including the very recent paper of Raman and Raman (2025) which studies generation in the limit with a noisy stream of examples. In particular, we highlight the relationship of our paper with **concurrent and independent** works on language generation by two sets of authors.

Non-uniform Generation. Our results on non-uniform generation are based on the definitions of uniform/non-uniform generation given originally by Raman and Tewari (2024), and their open question on characterizing the latter. In parallel and independent work, Raman and Tewari (2024) posted an updated version of their paper (Li, Raman, and Tewari, 2024), that, along with results on prompted generation, includes a result (Corollary 3.6, Li et al. (2024)) on non-uniform generation for countable collections; this resolves their open problem from earlier, like our Theorem 1.

Validity-breadth tradeoff. In the time we were preparing the original version of this manuscript and independently of our work, the very recent paper of Kalavasis, Mehrotra, and Velegkas (2025) also considers the validity-breadth tradeoff arising in Kleinberg and Mullainathan (2024). In particular, Kalavasis et al. (2025) also formalize a definition of *generation with breadth*, and obtain certain negative results similar to ours, providing evidence that the tradeoff between validity and breadth is necessary. While our definition of exhaustive generation is similar to their definition of generation with breadth, there are important differences. A generator that satisfies their definition also satisfies ours but not vice versa—thus ours is a *weaker* requirement of breadth than theirs. Furthermore, our lower bound is unconditional. Hence, our negative result for exhaustive generation allows us to also answer an open question asked by Kalavasis et al. (2025). We elaborate further on the connections to their work in Section 5.1. The work of Kalavasis et al. (2025) also studies the *statistical rates* at which generation in the limit can be achieved within the framework of universal learning (Bousquet et al., 2021), when the stream of input strings is drawn i.i.d. from a *distribution* supported on the target language. Some other notions of breadth (see Remark 19, Section C.2) are also considered by Kalavasis et al. (2025).

Characterizations of the validity-breadth tradeoff. Shortly after we made an initial version of this manuscript public, both we and the authors of Kalavasis et al. (2025) independently realized that we could build on our initial results to obtain tight characterizations (see Section 5.2). Their new manuscript (Kalavasis, Mehrotra, and Velegkas, 2024) also has characterizations of languages that satisfy exhaustive generation, as well as various other definitions of generation with breadth (exact/approximate breadth, etc.), mirroring some of the results we obtained in parallel. After we learned of their results from personal communication, we coordinated with them to make our definitions and terminology consistent. Appendix A has a detailed comparison between our results.

2. Preliminaries

We follow the problem setup of Kleinberg and Mullainathan (2024), who build upon the setup in Gold (1967). Let Σ be a finite alphabet set (e.g., $\{0,1\}$, $\{a, b, \ldots, z\}$), and let Σ^* denote the set of all strings of finite length formed by concatenating elements from Σ in any order. A language L is a countable subset of Σ^*, and we study generation from a **countable** collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ of languages. For generation in the limit to make sense, we will assume that $|L_i| = \infty$ for every i. The set of all integers is denoted by $\mathbb{Z}$.

2.1. Generation in the Limit

The setup assumes that a target language $K \in \mathcal{C}$ is chosen and fixed, and thereafter, strings from K are provided sequentially as input in the form of an enumeration $x_1, x_2, x_3, \ldots$. The choice of K and the order of enumeration can possibly be chosen by an adversary. Repetitions of strings are permitted in the enumeration—the only requirement is that every string in K appears at least once in the enumeration, i.e., for every $z \in K$, $z = x_t$ for some $t < \infty$. **We use the shorthand S_t to denote the set of all distinct strings seen in an enumeration $x_1, \ldots, x_t$ up until t.**

Definition 6 (Generation in the limit (Kleinberg and Mullainathan, 2024)) *An algorithm generates in the limit from languages in a collection $\mathcal{C}$, if for any language $K \in \mathcal{C}$ and any enumeration of K presented to the algorithm, there exists $t^* < \infty$ such that for all $t \geq t^*$, the string z_t generated by the algorithm at time step t belongs to $K \setminus S_t$.*

Remark 7 *Note that t^* above can depend on both K as well as the particular enumeration of it.*

In the above definition, the only requirement of the algorithm is that it is a *computable map* from the input seen so far to an output string (Kleinberg and Mullainathan (2024) refer to this setting as "generation in the limit via a function"). We will be explicit when we care about the computational power required by the algorithm to compute this map. For example, the way in which Kleinberg and Mullainathan (2024) account for the computational power required by a generating algorithm is via the *membership query* model. Here, at each step, an algorithm is allowed to make finitely many queries to an oracle of the form "$w \in L_i$?", for any $w \in \Sigma^*$ and any $L_i \in \mathcal{C}$ of its choice. Notably, the algorithm cannot make the query "$w \in K$?" corresponding to the unknown target language K. Kleinberg and Mullainathan (2024) showed that there exists an algorithm that successfully generates in the limit from any language in a countable collection using only membership queries.

2.2. Uniform/Non-uniform Generation

In the case that $\mathcal{C}$ is finite, Kleinberg and Mullainathan (2024) additionally showed that it is possible to construct an algorithm that *uniformly* generates from $\mathcal{C}$: namely, as soon as the algorithm sees $t^* = t^*(\mathcal{C})$ distinct strings from any $K \in \mathcal{C}$, *irrespective* of K and its enumeration order, it generates from $K \setminus S_t$ for every $t \geq t^*$. Inspired by this, the recent work of Li et al. (2024) formalizes the following distinctions of "non-uniform" and "uniform" generation of a language.

Definition 8 (Non-uniform generation in the limit (Definition 3 in Li et al. (2024))) *An algorithm non-uniformly generates in the limit from a collection $\mathcal{C}$, if for any language $K \in \mathcal{C}$, there exists a $t^* = t^*(\mathcal{C}, K)$ such that for any enumeration of K presented to the algorithm, the string z_t generated by the algorithm at time step t belongs to $K \setminus S_t$ for all t satisfying $|S_t| \geq t^*$.*

Definition 9 (Uniform generation in the limit (Definition 4 in Li et al. (2024))) *An algorithm uniformly generates in the limit from a collection $\mathcal{C}$, if there exists a $t^* = t^*(\mathcal{C})$ such that for any language $K \in \mathcal{C}$ and any enumeration of K presented to the algorithm, the string z_t generated by the algorithm at time step t belongs to $K \setminus S_t$ for all t satisfying $|S_t| \geq t^*$.*

Li et al. (2024) generalize the uniform generation result of Kleinberg and Mullainathan (2024) for finite collections to possibly infinite collections by showing that any collection $\mathcal{C}$ having *bounded complexity* (finite "closure dimension", see Definition 29) admits uniform generation. Furthermore, if a collection $\mathcal{C}$ has infinite closure dimension, then it is not possible to uniformly generate from $\mathcal{C}$.

2.3. Exhaustive Generation

The algorithm of Kleinberg and Mullainathan (2024) exhibits a tension between *validity* of outputs and *breadth* of generation, as also stated by them. Namely, their algorithm starts off by generating strings that possibly do not belong to the target language K, before eventually settling to generate from subsets of K that seemingly get smaller and smaller. They leave the problem of bridging this gap open, asking if it is possible to construct an algorithm that generates from K with breadth (i.e., does not miss out on generating any strings from K), or if there is a formal sense in which such a tradeoff is necessary. To model this tradeoff, we propose a definition of *exhaustive* generation.

For this, we consider a generating algorithm $\mathcal{A}$, which at any time t, maps S_t to a generator $\mathcal{G}_t : \mathbb{N} \to \Sigma^*$. We can imagine that the string generated by the algorithm at time step t is simply $\mathcal{G}_t(1)$. However, we want to also consider what happens if the input were to be *terminated* beyond time step t. In this case, we want to study the sequence of strings $\mathcal{G}_t(1), \mathcal{G}_t(2), \mathcal{G}_t(3), \dots$ that would be generated by $\mathcal{G}_t$—we can think of this latter scenario to be a form of "generate-only" mode that is implicitly defined by the generator $\mathcal{G}_t$. We use the shorthand $Z_{<t}$ to denote the set of distinct strings in the sequence $\mathcal{G}_1(1), \mathcal{G}_2(1), \dots, \mathcal{G}_{t-1}(1)$, and the shorthand $Z_{\geq t}$ to denote the set of distinct strings in the sequence $\mathcal{G}_t(1), \mathcal{G}_t(2), \dots$ generated by $\mathcal{G}_t$ were it to go into generate-only mode from time t.

Definition 10 (Exhaustive Generation) *A generating algorithm $\mathcal{A}$ exhaustively generates in the limit from languages in a collection $\mathcal{C}$, if for any $K \in \mathcal{C}$ and any enumeration of K, there exists $t^* < \infty$ such that for any $t \geq t^*$, it holds that (1) $|Z_{\geq t} \setminus K| < \infty$, and (2) $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq K$.*

We note that unlike uniform/non-uniform generation considered in Section 2.2, t^* in the above definition is allowed to depend on both K and its enumeration order, just as in Definition 6. Condition 1 above is in line with the requirement that the generator, if asked to go into generate-only mode, can only generate finitely many spurious strings, and thereafter, stops hallucinating. Condition 2 ensures coverage of the entire language. Since we want the generator to largely produce previously unseen examples, our definition effectively throws in the examples already presented in covering the target language K and does not penalize the generator for not regenerating those. We note that in Remark 24, Appendix D, we also comment on a slightly relaxed version of Definition 10.

While the notion of "generate-only" mode may seem unnatural at first, it is quite natural to terminate the input in the context of exhaustive generation. Recall that the input is a valid enumeration of the target language K; that is, every string in K eventually appears in it. Therefore, it is natural to terminate the input at some point and consider the output of the generator in "generate-only" mode.

3. Non-uniform Language Generation for Countable Collections

In this section, we show that every countable collection $\mathcal{C}$ of languages can be non-uniformly generated. We consider $\mathcal{C}$ to be specified in a given enumeration $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$.

Algorithm. Consider the algorithm, which at step t in the enumeration of its input, initializes $I_t = \Sigma^*$, and iterates through $L_1, \ldots, L_t$ in this order. Whenever it encounters a language L_i that contains S_t, it checks if $|I_t \cap L_i| = \infty$. If it is, then it updates I_t as $I_t = I_t \cap L_i$. Otherwise, it skips over L_i, and leaves I_t unaffected. Thus, throughout the algorithm's iteration over $L_1, \ldots, L_t$, the following invariants are maintained: (1) I_t is an infinite set, and (2) I_t is the intersection of a finite set of languages that contain S_t.[2] The algorithm then generates an arbitrary string from $I_t \setminus S_t$[3].

We will now show that the above algorithm non-uniformly generates from $\mathcal{C}$. We will specify the guarantee of the algorithm in terms of the *non-uniform complexity* of languages in $\mathcal{C}$.

Definition 11 (Non-uniform Complexity) *Given $\mathcal{C} = \{L_1, L_2, L_3, \ldots\}$, for any $i \in \mathbb{N}$, define the non-uniform complexity $m_{\mathcal{C}}(L_i)$ of L_i as*

$$m_{\mathcal{C}}(L_i) := \max\left\{\left|\bigcap_{L\in\mathcal{C}'} L\right| : \mathcal{C}' \subseteq \{L_1, \ldots, L_i\}, \mathcal{C}' \ni L_i, \left|\bigcap_{L\in\mathcal{C}'} L\right| < \infty\right\}. \tag{1}$$

Theorem 12 *For any language $L_{i^*} \in \mathcal{C}$, and any enumeration of L_{i^*} given as input to the above algorithm, the algorithm generates from $L_{i^*} \setminus S_t$ for all t satisfying $|S_t| \geq \max(i^*, m_{\mathcal{C}}(L_i^*) + 1)$.*

Proof Consider any t where $|S_t| \geq \max(i^*, m_{\mathcal{C}}(L_i^*) + 1)$. Note that such a t necessarily satisfies $t \geq i^*$, and hence, the target language L_{i^*} is under consideration by the algorithm at this step. Furthermore, L_{i^*} definitely contains S_t. So, the algorithm proceeds to iterate over $L_1, \ldots, L_t$, maintaining I_t along the way. We only need to argue that when the algorithm encounters L_{i^*} and considers $I_t \cap L_{i*}$, it finds that this set is infinite. If this is true, then I_t would be updated to be $I_t \cap L_{i^*}$, which is a subset of L_{i^*}. Thereafter, I_t will only become a smaller subset of L_{i^*} as the algorithm proceeds in its iteration. Finally, because the algorithm maintains I_t to be infinite, at the end of its iteration, I_t is an infinite subset of L_{i^*}. Thus, the algorithm can safely generate a string from $I_t \setminus S_t$, as it is guaranteed to also belong to $L_{i^*} \setminus S_t$.

So, we continue to argue that when the algorithm encounters L_{i^*} and considers $I_t \cap L_{i*}$, it finds that this set is infinite. Suppose this were not the case—then, $I_t \cap L_{i^*}$ is a finite set. Observe however, that $I_t \cap L_{i^*}$ is an intersection of languages that contain S_t. Therefore, since $|S_t| \geq m_{\mathcal{C}}(L_i^*) + 1$,

2. We use the convention that the intersection of zero languages is Σ^*.

3. If we do not want repetitions, we can generate an arbitrary ungenerated-as-yet string from the infinite $I_t \setminus S_t$.

we have that $|I_t \cap L_{i^*}| \geq m_{\mathcal{C}}(L_i^*) + 1$. But notice also that I_t is an intersection of languages that appear before L_{i^*} in the enumeration of $\mathcal{C}$. Thus, we have obtained a collection of languages that is a subcollection of $\{L_1, \dots, L_{i^*}\}$, contains L_{i^*}, and has a finite intersection of size at least $m_{\mathcal{C}}(L_i^*) + 1$: this contradicts the definition of $m_{\mathcal{C}}(L_i^*)$ (see (1)). Thus, $I_t \cap L_{i*}$ must be infinite. ■

At its heart, our algorithm is closer to the closure-based algorithm of Kleinberg and Mullainathan (2024) for finite collections. Namely, while that algorithm considers generating from the intersection of *all* languages in the collection consistent with the input (namely, the closure), our algorithm is more conservative among the languages it considers. Ultimately, it is able to greedily choose these languages, keeping track of a rather simple criterion—that of infinite intersection.

We make a concluding remark that our algorithm above is not collection-specific: it simultaneously works for all countable collections (defined on a fixed universe Σ^*), just like the algorithm of Kleinberg and Mullainathan (2024) for countable collections. Furthermore, our algorithm can be implemented with a membership query oracle, *and* an additional oracle, which, when queried with any finite collection of languages, responds with whether their intersection is finite or not.

4. Lower Bound for Non-uniform Language Generation Using Membership Queries

Since the generation algorithm of Kleinberg and Mullainathan (2024) can be implemented using membership queries, it is natural to ask if our non-uniform generation algorithm from above can also be implemented thus. We show that it is impossible for any algorithm to (simultaneously) non-uniformly generate for every finite collection (on a fixed Σ^*), with only membership queries. Surprisingly, our lower bound applies to collections with just *two* languages L_0 and L_1, with no a priori information about L_0 and L_1 other than access via membership queries.

Informally, the theorem says that we cannot have an algorithm with a non-uniform generation guarantee for every collection of two languages. A non-uniform guarantee for such a collection $\mathcal{C} = \{L_0, L_1\}$ means that, for any input that is a valid enumeration of L_0 (respectively L_1), there is a bound t_0 (respectively t_1) independent of the enumeration, such that the algorithm correctly generates from L_0 (respectively L_1) after time step t_0 (respectively t_1). The proof takes a supposed valid algorithm $\mathcal{A}$ and constructs a bad input, i.e., a specific collection of two languages where $\mathcal{A}$ fails to satisfy the non-uniform generation guarantee. This adversarial input is one that keeps the algorithm guessing at every step, i.e., the algorithm learns no information to determine whether the target languages is L_0 or L_1. For any time step t beyond $\max(t_0, t_1)$, the algorithm is supposed to make no mistakes. However, we can easily force a mistake at time t by picking the target language to be one of L_0, L_1, and revealing this to the algorithm after time step t.

Theorem 13 *There cannot exist a (deterministic) algorithm $\mathcal{A}$ that only makes membership queries, and satisfies the following property: for every collection $\mathcal{C} = \{L_0, L_1\}$ of two languages (on a fixed universe Σ^*), there exist $t^*(\mathcal{C}, L_0) < \infty$ and $t^*(\mathcal{C}, L_1) < \infty$ such that:*

1. *For every enumeration $x_1, x_2, \dots$ of L_0 and every $1 \leq t < \infty$, the algorithm makes a finite number of queries at step t, and if $|S_t| \geq t^*(\mathcal{C}, L_0)$, then $\mathcal{A}(S_t)$[4] $\in L_0 \setminus S_t$.*
2. *For every enumeration $x_1, x_2, \dots$ of L_1 and every $1 \leq t < \infty$, the algorithm makes a finite number of queries at step t, and if $|S_t| \geq t^*(\mathcal{C}, L_1)$, then $\mathcal{A}(S_t) \in L_1 \setminus S_t$.*

4. Here, we denote by $\mathcal{A}(S_t)$ the string z_t generated by the algorithm $\mathcal{A}$ at time step t.

For finite collections, uniform and non-uniform generation become equivalent, as one can simply assume the uniform bound to be the maximum non-uniform bound across all the languages. The proof constructs two languages L_0 and L_1 by simulating the given algorithm on an adaptively constructed input, over a sequence of rounds. In each round, we first add a string to both languages, feed it to the algorithm, observing the strings queried/generated, inserting these alternately into L_0/L_1, and answering accordingly. A key property is that L_0 and L_1 are fixed, infinite languages; all the strings presented to the algorithm belong to both. A non-uniform generation algorithm for this collection must have a bound n on the number of distinct examples it sees before it generates correctly. At a time step $m > n$, we can force the algorithm to make a mistake: suppose in our alternating strategy, we inserted the string that the algorithm generated in L_0. Then we declare L_1 as the true language and go on to enumerate all of L_1 (and vice versa). The full proof appears in the Appendix B.

One view of our proof is that two very different strategies generate from the target language, depending on whether the intersection of L_0 and L_1 is finite or infinite. For finite intersection of size d, from time step $d + 1$ onwards, only one of the two languages is consistent with the input since both the languages cannot contain the first $d + 1$ positive examples. Thus, the algorithm will have correctly identified the index of the target language at this point and can easily generate new strings from this language by performing membership queries until it finds a string in the language. On the other hand, if the intersection of the languages L_0 and L_1 is infinite, a very different strategy works: we simply perform membership queries until we find a new string x which belongs to both L_0 and L_1. Thus, successful strategies in the two cases of finite/infinite intersection both involve a potentially infinite loop guaranteed to terminate given a crucial piece of information, i.e., whether the intersection is finite or not. In the absence of this information *a priori*, we show that the algorithm must either loop forever, or a mistake can be forced at time step t for arbitrarily large t.

This does not contradict the uniform generation algorithm of Kleinberg and Mullainathan (2024) for finite collections because it crucially assumes *a priori* knowledge of whether the languages in the finite collection have infinite intersection or not, and uses very different strategies in the two cases of infinite/finite intersection.

5. Exhaustive Generation

In this section, we study the setting of exhaustive generation introduced in Section 2.3. Recall that in this setting, we think of generating algorithms $\mathcal{A}$ that output a generator $\mathcal{G}_t : \mathbb{N} \to \Sigma^*$ at every time step t. We denote by $Z_{<t}$ the set of distinct strings generated by the algorithm up until time $t-1$ (namely $\mathcal{G}_1(1), \ldots, \mathcal{G}_{t-1}(1)$), and by $Z_{\geq t}$ the set of distinct strings generated by the algorithm from time t onwards, as if it were to stop seeing any more input (namely $\mathcal{G}_t(1), \mathcal{G}_t(2), \ldots$). As given in Definition 10, we desire that for any language $K \in \mathcal{C}$ and enumeration of it, there exists a finite $t^* < \infty$ such that for every $t \geq t^*$, it holds that $|Z_{\geq t} \setminus K| < \infty$, and also that $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq K$.

We now restate and prove Theorem 3, showing that even simple collections cannot be exhaustively generated (see also Corollary 16, which rules out a still weaker form of exhaustive generation).

Theorem 3 (Exhaustive Generation Lower Bound) *There exists a countable collection $\mathcal{C}$ (of regular languages) that cannot be exhaustively generated in the limit.*

Proof Consider the collection $\mathcal{C} = L_\infty \cup \bigcup_{i \in \mathbb{Z}} L_i$, where $L_\infty = \mathbb{Z}$, and $L_i = \{-i, -i+1, -i+2, \ldots\}$ for $i \in \mathbb{Z}$. Each language is an arithmetic progression, and hence a regular language. Assume for the sake of contradiction that there exists an algorithm $\mathcal{A}$ that exhaustively generates in the limit from languages in $\mathcal{C}$. This means that for any $K \in \mathcal{C}$ and any enumeration of K, there exists a $t^* < \infty$ such that for any $t \geq t^*$, it holds that (1) $|Z_{\geq t} \setminus K| < \infty$, and (2) $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq K$.

Suppose that an adversary enumerates L_0 as $0, 1, 2, 3, \ldots$. Then, there must exist some time step $t_0^* < \infty$, such that for the generator $\mathcal{G}_{t_0^*}$ output by $\mathcal{A}$ at t_0^* (which is solely a function of $S_{t_0^*} = \{0, 1, 2, \ldots, t_0^*\}$, we have that $|Z_{\geq t_0^*} \setminus L_0| < \infty$. Let $t_0 = t_0^*$.

Next, consider the enumeration of L_1, given as $0, 1, 2, \ldots, t_0, -1, 0, 1, 2, 3, \ldots$. This is a valid enumeration of L_1, and hence there must exist a finite time step $t_1^* < \infty$, such that for any $t \geq t_1^*$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $|Z_{\geq t} \setminus L_1| < \infty$. Let $t_1 = \max(t_1^*, t_0 + 1)$, and observe that in this enumeration of L_1, t_1 appears at a timestep beyond t_1^*.

Now, consider the enumeration of L_2, given as $0, 1, 2, \ldots, t_0, -1, 0, 1, 2, 3, \ldots, t_1, -2, -1, 0, \ldots$. This is a valid enumeration of L_2, and hence there must exist a finite $t_2^* < \infty$, such that for any $t \geq t_2^*$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $|Z_{\geq t} \setminus L_2| < \infty$. Let $t_2 = \max(t_2^*, t_1 + 1)$, and observe again that in this enumeration of L_2, t_2 appears at a time step beyond t_2^*.

We can repeat the same argument indefinitely, which results in the enumeration $0, 1, 2, \ldots, t_0, -1, 0, 1, 2, \ldots, t_1, -2, -1, \ldots, t_2, -3, -2, \ldots, t_3, -4, -3, , \ldots$. Observe that this is a valid enumeration of L_∞: starting with $i = 0$, phase i of the above argument comprises of the sequence $-i, -i + 1, \ldots, 0, 1, \ldots, t_i$, and the overall enumeration is a concatenation of the sequences produced in phases $0, 1, 2, \ldots$. Hence, every negative integer appears at least once in this enumeration, and so does every positive integer, because $t_0 < t_1 < t_2 < \ldots$.

Now, if $\mathcal{A}$ were to exhaustively generate from L_∞, for the above enumeration of L_∞, there must be some finite time step t_∞ such that for every $t \geq t_\infty$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq L_\infty$. Consider then the smallest i for which t_i appears at a time step beyond t_∞ (such an i must exist, because the sequence $t_0, t_1, t_2, \ldots$ is increasing), and let this time step at which t_i appears be $t'_i \geq t_\infty$. By the previous reasoning, we have that $S_{t'_i} \cup Z_{<t'_i} \cup Z_{\geq t'_i} \supseteq L_\infty$. However, by construction, the time step t'_i is beyond t_i^*, and hence by the invariant that we have maintained (property (1) above), $|Z_{\geq t'_i} \setminus L_i| < \infty$. Hence, $S_{t'_i} \cup Z_{<_{t'_i}} \supseteq L_\infty \setminus (L_i \cup \{Z_{\geq t'_i} \setminus L_i\})$. But $S_{t'_i} \cup Z_{<t'_i}$ is a finite set, and $L_\infty \setminus (L_i \cup \{Z_{\geq t'_i} \setminus L_i\})$ is an infinite set, so this is impossible. ■

While it is not possible to exhaustively generate from the collection $\mathcal{C}$ in the proof of Theorem 3, it is also not possible to *identify* from this collection. Identifiability immediately implies exhaustive generation.In Section C.1, we separate these two notions, by showing a simple collection that is non-identifiable, but can be exhaustively generated. The collection (Example 2) simply consists of the set of all integers, together with sets that exclude a single integer.

5.1. Connections to Generation with Breadth (Kalavasis et al., 2025)

Our definition of exhaustive generation is related to the definition of generation with breadth given in the concurrent work of Kalavasis et al. (2025). While both these definitions seek to formalize the notion of *generating the entirety of the language*, there are important differences.

To elaborate on these, we first state the definition of generation with breadth from Kalavasis et al. (2025). While their model assumes that the generator output by the algorithm at time step t is a *distribution* over strings in Σ^*, and defines this notion in terms of the *support* of the distribution, we can equivalently state the definition in our formulation, where the generator $\mathcal{G}_t$ is a mapping from $\mathbb{N} \to \Sigma^*$ that enumerates the support of the distribution. Conversely, any distribution supported on the range $Z_{\geq t}$ of $\mathcal{G}_t$ is a generator in the sense of Kalavasis et al. (2025).

Definition 14 (Generation with breadth (Kalavasis et al., 2025)) *A generating algorithm $\mathcal{A}$ generates with breadth in the limit from languages in a collection $\mathcal{C}$, if for any $K \in \mathcal{C}$ and any enumeration of K, there exists $t^* < \infty$ such that for every $t \geq t^*$, it holds that $Z_{\geq t} = K$.*[5]

Observe that an algorithm that generates with breadth also satisfies our definition of exhaustive generation. This helps us answer an open question asked by Kalavasis et al. (2025) in the negative:

Open Question 1 in Kalavasis et al. (2025): *Is there a class of generative algorithms for which the induced generators can be modeled as Turing machines and which achieve breadth and consistency for all countable collections of languages?*

Here, "consistency" means the requirement of Definition 6. Note that if there was such a class of algorithms which achieves generation with breadth for all countable collections, it would also achieve exhaustive generation for these collections. This would contradict Theorem 3; thus, there cannot exist a class of algorithms that achieves breadth and consistency for all countable collections.

However, exhaustive generation *does not* imply generation with breadth. Notice that generation with breadth does not allow an algorithm to utilize the previous strings $Z_{\leq t}$ it generated, whereas our second condition for exhaustive generation (Definition 10) allows an algorithm to include an arbitrary subset of $Z_{\leq t}$ in order to cover the target language. Our definition of exhaustive generation was formulated independently of Kalavasis et al. (2025); hence the difference in the two definitions.

Before we go on, we briefly explain our rationale for including a subset of previously generated elements $Z_{\leq t}$ in covering the target language K. Requiring the generator to produce the entirety of the target language (i.e., the condition $Z_{\geq t} = K$) does appear rather close (although not obviously equivalent) to being able to identify the target language. The strong lower bounds on language identification motivated us to consider our definition of exhaustive generation, which allows for a notion of error in using previously generated elements to cover the target language K. Allowing such errors is already present in the original notion of generation in the limit from Kleinberg and Mullainathan (2024). The natural analog for exhaustive generation is to allow the algorithm to use a subset of the elements generated so far in covering K, leading to our definition.

This crucial distinction also allows us to formally separate exhaustive generation from generation with breadth. In Section C.2.1, we show that the collection mentioned above, that can be exhaustively generated but not identified, can also *not* be generated with breadth. Moreover, in Section C.2.2, we show that a necessary condition for identifiability is also necessary for generation with breadth, further illustrating how these two notions are closely tied together.

5.2. Characterization of Exhaustive Generation

Finally, similar to Angluin's characterization (Theorem 18) for identification in the limit, we fully characterize exhaustive generation. We introduce a weakening of the condition from Angluin's work, which we denote "Weak Angluin's Condition with Existence", and show that it precisely characterizes the language collections that can be exhaustively generated.

Weak Angluin's Condition with Existence:

A collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Weak Angluin's Condition with Existence, if for every language $L \in \mathcal{C}$, there exists a finite subset $T \subseteq L$, such that every $L' \in \mathcal{C}$ that contains T and is a proper subset of L satisfies $|L \setminus L'| < \infty$. (2)

5. Kalavasis et al. (2025) state two equivalent definitions of generation with breadth (Remark 2, Kalavasis et al. (2025)).

The proof of necessity of this condition (Proposition 23, Section D.1) follows an outline similar to the proof of Theorem 3. The proof of sufficiency (Proposition 25) shows how a modification of the algorithm by Kleinberg and Mullainathan (2024) exploits the existence of the sets T for each language L, so as to satisfy the conditions of exhaustive generation. However, this algorithm ends up requiring access to a powerful oracle; we remark that a slightly stronger condition (Section D.2) allows us to give an exhaustive generation algorithm that only uses the membership query oracle .

6. Uniform Generation with Feedback

In this section, we consider the setting of uniform generation from a collection *with feedback*, where an algorithm is allowed to query if any string w of choice belongs to the target language K. This model is different from the membership query model considered in Kleinberg and Mullainathan (2024); there, an algorithm can only query if a string w belongs to any language L_i in the collection $\mathcal{C}$. Directly querying membership in the target language K gives the algorithm more power.

We restrict our attention to countable collections; by Theorem 12, non-uniform generation is always possible for these, without feedback. When is *uniform* generation possible, with or without feedback? First, there exist language collections that cannot be uniformly generated without feedback, but can be uniformly generated with feedback (adopted from Lemma 3.9 in Li et al. (2024)).

Example 1 *Consider a partition $\{S_d\}_{d\in\mathbb{N}}$ of $\mathbb{N}$, where $|S_1| < |S_2| < \dots$. Let E be all negative even integers, and O be all negative odd integers. For $d \in \mathbb{N}$, let $L_d^E = E \cup S_d$, and let $L_d^O = O \cup S_d$, and consider $\mathcal{C}$ comprising of all languages L_d^E and L_d^O for $d \in \mathbb{N}$. This collection has infinite closure dimension, and hence cannot be uniformly generated from without feedback. With just one query, a generator can find out whether the target language belongs to the "even" or "odd" category. Then, it can generate indefinitely from either the set of even or odd negative integers.*

We want to identify a property of a given collection $\mathcal{C}$ which characterizes uniform generation with feedback. We formulate this in the form of a combinatorial dimension like the closure dimension (Definition 29) from Li et al. (2024) for uniform generation. We briefly give the intuition behind the definition here, and refer the reader to Appendix E for a detailed derivation.

We view language generation as the *transcript* of an interaction over multiple rounds, between an adversary enumerating a language L from a collection, and a generator trying to generate new strings from L. In each round, the adversary records a string from L followed by the generator recording its membership query to the adversary. The adversary responds (truthfully) to this query. Finally, the generator generates a string, and we move to the next round, where the process repeats. Each adversary or generator action can depend on the entire transcript so far.

With this view, the quantity of interest for the generator at any round is the *effective intersection* of consistent languages, i.e. the intersection of languages in the collection that are consistent with the transcript so far, after excluding the strings that the adversary has already provided. The GF dimension of a collection is the number of rounds that an adversary can proceed, while ensuring that the effective intersection at the last round is empty.

Definition 15 (GF dimension) *The GF dimension of a collection $\mathcal{C}$ is the supremum over $d \in \mathbb{N}$, for d satisfying the following property: for every generator strategy G, there exists a language $K \in \mathcal{C}$ and an adversary strategy A consistent with K, such that in the transcript $T = T(A, G)$, there exists a finite $r \geq d$ where $|S_r| \geq d$ and effective intersection $E_r(T) = \emptyset$.*

Lemmas 33, 34 argue that generation with feedback is possible uniformly, if and only if the GF dimension of the collection is finite (see Appendix E). We also show (see Section E.1 and Proposition 30) how our slightly different view of generation is expressive enough to capture the definition of closure dimension from Li et al. (2024). While Example 1 above separates uniform generation with and without feedback, we also present Example 3 that cannot be uniformly generated with or without feedback, but can still be *non-uniformly* generated.

Acknowledgments

The authors are supported by Moses Charikar's Simons Investigator Award. Chirag is additionally also supported by Gregory Valiant's Simons Investigator Award.

References

Dana Angluin. Finding patterns common to a set of strings. In *Proceedings of the eleventh annual ACM Symposium on Theory of Computing*, pages 130–141, 1979. URL https://www.sciencedirect.com/science/article/pii/0022000080900410.

Dana Angluin. Inductive inference of formal languages from positive data. *Information and control*, 45(2):117–135, 1980. URL https://www.sciencedirect.com/science/article/pii/S0019995880902855.

Martin Arjovsky and Leon Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/1701.04862.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017. URL https://proceedings.mlr.press/v70/arjovsky17a.html.

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. Llms will always hallucinate, and we need to live with this. *arXiv preprint arXiv:2409.05746*, 2024. URL https://arxiv.org/abs/2409.05746.

Olivier Bousquet, Steve Hanneke, Shay Moran, Ramon Van Handel, and Amir Yehudayoff. A theory of universal learning. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 532–541, 2021. URL https://dl.acm.org/doi/10.1145/3406325.3451087.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967. URL https://www.sciencedirect.com/science/article/pii/S0019995867911655.

Adam Tauman Kalai and Santosh S Vempala. Calibrated language models must hallucinate. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 160–171, 2024. URL https://dl.acm.org/doi/10.1145/3618260.3649777.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. Characterizations of language generation with breadth. *arXiv preprint arXiv:2412.18530*, 2024. URL https://arxiv.org/abs/2412.18530.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. 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 (To Appear)*, 2025. URL https://arxiv.org/abs/2411.09642.

Jon Kleinberg and Sendhil Mullainathan. Language generation in the limit. *Advances in Neural Information Processing Systems*, 37, 2024. URL https://arxiv.org/abs/2404.06757.

Jiaxun Li, Vinod Raman, and Ambuj Tewari. Generation through the lens of learning theory. *arXiv preprint arXiv:2410.13714v4*, 2024. URL https://arxiv.org/abs/2410.13714.

Ananth Raman and Vinod Raman. Generation from noisy examples. *arXiv preprint arXiv:2501.04179*, 2025.

Vinod Raman and Ambuj Tewari. Generation through the lens of learning theory. *arXiv preprint arXiv:2410.13714v3*, 2024. URL https://arxiv.org/abs/2410.13714v1.

Changlong Wu, Ananth Grama, and Wojciech Szpankowski. No free lunch: Fundamental limits of learning non-hallucinating generative models. In *International Conference on Learning Representations (To Appear)*, 2025. URL https://arxiv.org/abs/2410.19217.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*, 2024. URL https://arxiv.org/abs/2401.11817.

Appendix A. Comparison to Kalavasis et al. (2024)

To aid the reader, we map some of our results to the results in Kalavasis et al. (2024).

Exhaustive Generation: Our result showing that Weak Angluin's Condition with Existence (Proposition 23) is necessary for exhaustive generation is comparable to the similar result in Lemma 2.11 in Kalavasis et al. (2024). Our result showing the sufficiency of Weak Angluin's Condition with Existence (Proposition 25) for exhaustive generation is comparable to Lemma 2.9 in Kalavasis et al. (2024). Our result showing the sufficiency of Weak Angluin's Condition with Enumeration (Proposition 25) for exhaustive generation with only membership queries is comparable to Lemma 2.10 in Kalavasis et al. (2024).

Generation with (Exact) Breadth: Our result showing that Angluin's Condition with Existence (Proposition 21) is necessary for generation with (exact) breadth is comparable to Lemma 2.1 in Kalavasis et al. (2024).

We note also that Kalavasis et al. (2024) also establish characterizations for other notions of generation with breadth that Kalavasis et al. (2025) introduced, like generation with approximate breadth, unambiguous generation and stable generation (see also Remark 19).

Appendix B. Membership Query Lower Bound for Non-uniform Generation

Proof of Theorem 13 Assume for the sake of contradiction that $\mathcal{A}$ is a valid generating algorithm satisfying the property. We will adversarially construct two languages $L_0 = L_0(\mathcal{A})$ and $L_1 = L_1(\mathcal{A})$ such that $\mathcal{A}$ does not satisfy the property on the collection $\mathcal{C} = \{L_0, L_1\}$.

We will build up L_0 and L_1 in phases $\mathcal{P}_1, \mathcal{P}_2, \dots$, as a function of the execution of $\mathcal{A}$. The first time that we will consider any string $w \in \Sigma^*$, we will irrevocably decide whether w belongs to exactly one of L_0 or L_1, or both of them. Towards this, we will maintain a global variable a whose state is tracked across phases: it is initialized to $a = 0$ before the start of Phase $\mathcal{P}_1$, and maintains its value from the end of phase $\mathcal{P}_{m-1}$ to the start of phase $\mathcal{P}_m$. We also maintain a dictionary `membership`, whose keys are strings in Σ^*, and initialize $\texttt{membership}[w] = -1$ for every string $w \in \Sigma^*$. Each phase $\mathcal{P}_m$ operates as follows:

Description of Phase $\mathcal{P}_m$:

(1) //*prepare next string to be enumerated*

Insert a yet unseen, distinct string x_m in both L_0 and L_1,
and set $\texttt{membership}[x_m] = \{0, 1\}$.

(2) Feed x_m as the next input to $\mathcal{A}$.

(3) //*process membership queries made by the algorithm*

Initialize $j = 1$, and loop until $\mathcal{A}$ generates some z_m :

Suppose $\mathcal{A}$ queries "$y_j \in L_{a_j}$?" for $a_j \in \{0, 1\}$.

If $\texttt{membership}[y_j] \neq -1$:

$//y_j$ *has already been placed in* L_0, L_1 *earlier*

Answer "Yes"/"No" according to $\texttt{membership}[y_j]$.

Else:

$//$*place* y_j *in exactly one of* L_0, L_1

Insert y_j in L_a, set $\texttt{membership}[y_j] = a$.

Answer "Yes"/"No" based on $a_j \stackrel{?}{=} a$, and thereafter, set $a = 1 - a$.

(4) //*process string guessed by the algorithm*

If $\texttt{membership}[z_m] = -1$ for the generated string z_m:

Insert z_m in L_a, set $\texttt{membership}[z_m] = a$, and thereafter, set $a = 1 - a$.

This concludes the construction of the languages L_0 and L_1. Now, suppose there were some finite $t^*(\mathcal{C}, L_0)$ and $t^*(\mathcal{C}, L_1)$ for which the generations of $\mathcal{A}$ satisfied the property in the theorem statement. Let $n = \max(t^*(\mathcal{C}, L_0), t^*(\mathcal{C}, L_1))$. We note a few key properties of our construction that we exploit in the rest of the proof: (i) L_0 and L_1 are infinite languages. (ii) Each string enumerated in Step (2) as input belongs to both L_0 and L_1, thus the enumeration $x_1, x_2, \dots x_m$ (for $m \leq n$) is valid for both L_0 and L_1. (iii) The adversary can force the algorithm to make a mistake for the generated string z_n (and hence violate the correctness guarantee) by committing to one of L_0, L_1 as the true language.

Consider feeding the generator the input sequence $x_1, x_2, \dots, x_n$, where for $m \in [n]$, x_m is the string from Step (1) in Phase $\mathcal{P}_m$ above. Note that the number of distinct strings $|S_n| = n$. Suppose that $z_1, \ldots, z_n$ are the strings $\mathcal{A}$ generates at each step. Observe that by design, upon feeding x_m, $\mathcal{A}$ asks exactly the same sequence of queries that we considered in the loop in Step (3) above.

We first argue that $\mathcal{A}$ must only ask a finite number of queries, and then generate z_m, which is exactly the string generated by $\mathcal{A}$ at the culmination of the loop in Step (3). If this is not the case, then the loop in Step (3) would never have terminated. The infinite loop ensures that both L_0 and L_1 continue being populated to be infinite languages—hence, we have a legal collection $\mathcal{C}$. Note that the sequence $x_1, x_2, \ldots x_m$ produced so far can be extended to be a valid enumeration of either L_0 or L_1 (since each of $x_1, \ldots, x_m$ belongs to both L_0 and L_1, as ensured by Step (1)). We now simply declare L_0 to be the true (target) language. Since the sequence $x_1, x_2, \ldots x_m$ is the prefix of a valid enumeration of L_0, $\mathcal{A}$ must satisfy the correctness guarantee for a valid generation algorithm and cannot loop forever.

So, consider the string z_n generated by $\mathcal{A}$ after seeing the last input x_n—note that by the correctness guarantee, z_n must not belong to $S_n = \{x_1, x_2, \ldots x_n\}$. Consider the value of $\texttt{membership}[z_n]$ after Step (4) in Phase $\mathcal{P}_n$; suppose $\texttt{membership}[z_n] = 0$. We will then declare L_1 to be the true language, and thereafter, simply begin re-enumerating all of L_1 afresh. Similarly, if $\texttt{membership}[z_n] = 1$, we will declare L_0 to be the true language, and thereafter, simply begin re-enumerating all of L_0. Notice that either way, we produce a legal and complete enumeration of the true language. More importantly, notice that in either case, the string z_n that $\mathcal{A}$ generates at step n **does not** belong to the true language—this is because z_n is only contained in one of L_0 or L_1 after Step (4). This violates the guarantee required of $\mathcal{A}$, and hence $\mathcal{A}$ is not a valid generating algorithm.

■

Appendix C. Results on Exhaustive Generation

The proof of Theorem 3 also allows us to rule out algorithms that satisfy a slightly weaker notion of exhaustive generation. For example, consider the following randomized guarantee.

Corollary 16 (Randomized Exhaustive Generation Lower Bound) *For the collection $\mathcal{C}$ considered in the proof of Theorem 3, there cannot exist a randomized algorithm $\mathcal{A}$ which satisfies the following guarantee: for any $K \in \mathcal{C}$ and any enumeration of K, there exists $t^* < \infty$ such that for any $t \geq t^*$, it holds with probability $> 1/2$ that*

1. $|Z_{\geq t} \setminus K| < \infty$.
2. $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq K$.

Proof We need only modify the last paragraph in the proof of Theorem 18 as follows: for the constructed enumeration of L_∞, there must be some finite time step t_∞ such that for every $t \geq t_\infty$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that with probability $> 1/2$, $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq L_\infty$. Consider then the smallest i for which t_i appears at a time step beyond t_∞ (such an i must exist, because the sequence $t_0, t_1, t_2, \ldots$ is increasing), and let this time step at which t_i appears be t'_i. By the previous reasoning, we have that with probability $> 1/2$, $S_{t'_i} \cup Z_{<t'_i} \cup Z_{\geq t'_i} \supseteq L_\infty$. However, by construction, the time step t'_i is beyond t^*_i, and hence by property (1) above, with probability $> 1/2$, $|Z_{\geq t'_i} \setminus L_i| < \infty$. Since the two events $S_{t'_i} \cup Z_{<t'_i} \cup Z_{\geq t'_i} \supseteq L_\infty$ and $|Z_{\geq t'_i} \setminus L_i| < \infty$ individually occur with probability $> 1/2$, they both together occur with probability > 0. Thus, with probability > 0, $S_{t'_i} \cup Z_{<_{t'_i}} \supseteq L_\infty \setminus (L_i \cup \{Z_{\geq t'_i} \setminus L_i\})$. But $S_{t'_i} \cup Z_{<t'_i}$ is a finite set, and $L_\infty \setminus (L_i \cup \{Z_{\geq t'_i} \setminus L_i\})$ is an infinite set, meaning that $S_{t'_i} \cup Z_{<t'_i} \cup Z_{\geq t'_i} \supseteq L_\infty$ should happen with probability 0. This is a contradiction. ∎

C.1. Separation between Identifiability and Exhaustive Generation

One might observe that while it is not possible to exhaustively generate from the collection $\mathcal{C}$ considered in the proof of Theorem 3, it is additionally also not possible to *identify* from this collection. Furthermore, the proof we presented also has parallels to the way one might go about proving non-identifiability for this collection. A natural question to consider then is: are the notions of identifiability and exhaustive generation the same?

For clarity, we recall the definition of identifiability, wherein an algorithm is trying to figure out the *index* of the target language being enumerated, and state Angluin's characterization (Angluin, 1979, 1980) for collections of languages that are identifiable in the limit.

Definition 17 (Identification in the limit Gold (1967)) *An algorithm identifies in the limit from languages in a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$, if for any language $K \in \mathcal{C}$ and any enumeration of K presented to the algorithm, there exists $t^* < \infty$ such that for all $t \geq t^*$, the index i_t output by the algorithm at time step t satisfies $L_{i_t} = K$.*

Angluin's Condition with Enumeration (Condition 1 in Angluin (1980)):
A collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Angluin's Condition with Enumeration, if there exists a computable procedure, which for every language $L \in \mathcal{C}$, outputs an enumeration of a finite set T, such that $T \subseteq L$, and furthermore, every $L' \in \mathcal{C}$ that contains T satisfies that L' is not a proper subset of L. (3)

Theorem 18 (Angluin's characterization (Theorem 1 in Angluin (1980))) *A collection of languages $\mathcal{C} = \{L_1, L_2, \ldots\}$ is identifiable in the limit if and only if it satisfies Angluin's Condition with Enumeration.*

Note that identifiability immediately implies exhaustive generation—once an algorithm has identified the target language, it can simply enumerate all the strings from it thereafter. Does exhaustive enumeration also imply identifiability? The following example shows that there are collections that are non-identifiable, but can be exhaustively generated.

Example 2 *Let $\Sigma^* = \mathbb{Z}$. Let $L_\infty = \mathbb{Z}$, and for any $i \in \mathbb{Z}$, let $L_i = \mathbb{Z} \setminus \{i\}$. Consider the countable collection $\mathcal{C} = L_\infty \cup \bigcup_{i\in\mathbb{Z}} L_i$. We first claim that it is not possible to identify from $\mathcal{C}$. To see this, observe that for every finite subset T of L_∞, any language L_i for which $i \notin T$ contains T, but L_i is a proper subset of L_∞. Hence, this collection cannot satisfy Angluin's Condition with Enumeration, and by Theorem 18, $\mathcal{C}$ is not identifiable in the limit.*

We now argue that it is possible to exhaustively generate from $\mathcal{C}$ in a straightforward manner. Consider the algorithm $\mathcal{A}$, which, oblivious of the input sequence, simply generates the sequence $0, -1, 1, -2, 2, -3, 3, -4, 4, \ldots$ in this order; the generator $\mathcal{G}_t : \mathbb{N} \to \Sigma^*$ that $\mathcal{A}$ outputs at time step t may be appropriately deduced from this. We claim that this algorithm exhaustively generates from the collection. To see this, consider first that the target language is L_∞. Then, we can set $t^* = 1$, for which, we have that for all $t \geq t^*$, $Z_{\geq t} \subseteq L_\infty$, and also, $Z_{<t} \cup Z_{\geq t} = L_\infty$. Now, consider instead that the target language is L_i, for some $i \in \mathbb{Z}$. Then, observe crucially that once the algorithm generates i (an error), it makes no further errors. Namely, let t_i be the time step at which the algorithm generates i. We can set $t^* = t_i + 1$, which guarantees that for all $t \geq t^*$, $Z_{\geq t} \subseteq L_i$ and also $Z_{<t} \cup Z_{\geq t} \supseteq L_i$.

C.2. Connections to Generation with Breadth (Kalavasis et al., 2025)

We restate the definition of generation with breadth (Kalavasis et al., 2025) from the main body for clarity.

Definition 14 (Generation with breadth (Kalavasis et al., 2025)) *A generating algorithm $\mathcal{A}$ generates with breadth in the limit from languages in a collection $\mathcal{C}$, if for any $K \in \mathcal{C}$ and any enumeration of K, there exists $t^* < \infty$ such that for every $t \geq t^*$, it holds that $Z_{\geq t} = K$.*[6]

Remark 19 *Kalavasis et al. (2025) sometimes refer to the above as generation with exact breadth, to disambiguate it from some other relaxations that they consider, like generation with approximate breadth, unambiguous generation, stable generation, etc. For approximate breadth (Definition 21 in Kalavasis et al. (2025)), the generator is required to eventually only generate strings from the target language, but is allowed to miss out on finitely many strings from it. In unambiguous generation (Definition 8 in Kalavasis et al. (2025)), the generator is allowed to hallucinate (i.e., generate strings not belonging to the target language) infinitely often, but eventually, the set of strings it generates should have the smallest symmetric difference to the target language than any other language in the collection. For more details about how these, and other notions relate to one another, we refer the reader to the work by Kalavasis et al. (2024).*

6. Kalavasis et al. (2025) state two equivalent definitions of generation with breadth (Remark 2, Kalavasis et al. (2025)).

C.2.1. DISTINCTION BETWEEN GENERATION WITH BREADTH AND EXHAUSTIVE GENERATION

We elaborate on the relationship of the various notions: generation with breadth, exhaustive generation and language identification. One of the main results (Theorem 3.5) in Kalavasis et al. (2025) shows that, if there is a generating algorithm satisfying a certain "MOP" condition, which generates with breadth from a collection, then it can be used to construct an algorithm that can identify languages from the collection. The technical "MOP" condition, short for Membership Oracle Problem, is the following: for any generator that the algorithm may output at any time step (which, in the formulation of Kalavasis et al. (2025), is a *distribution* over strings), it should be possible to decide whether any string x belongs to the support of the generator.

But we can observe that the generators $\mathcal{G}_t$ output by the exhaustive generation algorithm from Example 2, viewed in the distributional sense of Kalavasis et al. (2025), satisfy the above MOP property—to check if some string x belongs to the support of $\mathcal{G}_t$, note that $Z_{\ge 2t+1} = \mathbb{Z} \setminus \{-t, \dots, t\}$ and $Z_{\ge 2t} = \mathbb{Z} \setminus \{-t, \dots, t-1\}$. But even so, the collection in the example is not identifiable! We can also observe that if the target language is, say L_1, at no time step $t \ge 1$ in the generated sequence does it hold that $Z_{\ge t} = L_1$. Hence, this algorithm does not generate with breadth. In fact, as we show ahead, there cannot exist an algorithm that generates with breadth for this collection.

Claim 20 *The collection $\mathcal{C}$ considered in Example 2 cannot be generated with breadth in the limit.*

Proof The proof steps are similar to the proof of Theorem 3. Assume for the sake of contradiction that there exists an algorithm $\mathcal{A}$ that generates the languages in $\mathcal{C}$ with breadth. This means that for any $K \in \mathcal{C}$ and any enumeration of K, there exists a $t^* < \infty$ such that for any $t \ge t^*$, it holds that $Z_{\ge t} = K$.

For the sake of convenience, for any $x \in \mathbb{Z}$, define

$$\text{next}(x) = \begin{cases} -x & \text{if } x < 0 \\ -(x+1) & \text{if } x > 0 \\ -1 & \text{if } x = 0. \end{cases}$$

That is, $\text{next}(x)$ is the example immediately following x in the sequence $0, -1, 1, -2, 2, -3, 3, \dots$. Suppose now that an adversary enumerates L_0 as $-1, 1, -2, 2, -3, 3, \dots$. Then, there must exist some time step $t_0^* < \infty$, such that for the generator $\mathcal{G}_{t_0^*}$ output by $\mathcal{A}$ at t_0^*, we have that $Z_{\ge t_0^*} = L_0$. Let $t_0 = t_0^*$, and let the example in the enumeration above at time step t_0 be n_1'.

Next, consider the enumeration of L_{n_1}, where $n_1 = \text{next}(n_1')$, given as

$$-1, 1, -2, 2, -3, 3, \dots, n_1', 0, -1, 1, -2, 2, \dots, n_1', \text{next}(n_1), \dots$$

where beyond n_1', observe that we inserted 0, and skipped including n_1. The reason we inserted 0 was because it was missing in the enumeration before n_1' (on account of the previous enumeration of L_0). This is a valid enumeration of L_{n_1}, and hence there must exist a finite time step $t_1^* < \infty$, such that for any $t \ge t_1^*$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $Z_{\ge t} = L_{n_1}$. In particular, choose a time step t_1 such that the example n_2' at t_1 satisfies that $|n_2'| > |n_1'|$.

Now, consider the enumeration of L_{n_2}, where $n_2 = \text{next}(n_2')$, given as

$$-1, 1, -2, 2, -3, 3, \dots, n_1', 0, -1, 1, -2, 2, \dots, n_2', n_1, 0, -1, 1, -2, 2, -3, 3, n_2', \text{next}(n_2), \dots$$

where beyond n'_2, observe that we again included n_1 (because it was missing in the enumeration till n'_2), and did not include n_2. This is a valid enumeration of L_{n_2}, and hence there must exist a finite time step $t^*_2 < \infty$, such that for any $t \geq t^*_2$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $Z_{\geq t} = L_{n_2}$. In particular, choose a time step t_2 such that the example n'_3 at t_2 satisfies that $|n'_3| > |n'_2|$.

We can repeat the same argument indefinitely, and observe that this results in an enumeration of L_∞. This is because the $|n'_1| < |n'_2| < |n'_3| < \ldots$, and the sequence between any $0, -1, 1, \ldots, n'_{i+1}, n_i$ comprises of all the elements in the enumeration $0, -1, 1, -2, 2, \ldots$ up until n'_{i+1}.

Now, if $\mathcal{A}$ were to successfully generate with breadth from L_∞, for the above enumeration of L_∞, there must be some finite time step t_∞ such that for every $t \geq t_\infty$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $Z_{\geq t} = L_\infty$. Consider then the smallest j such that n'_{j+1} appears at a time step beyond t_∞, and let t_j be the time step at which n'_{j+1} appears. By the invariant we have maintained above, it must be the case that $Z_{\geq t_j} = L_{n_j}$, which contradicts the requirement that $Z_{\geq t_j} = L_\infty$. ■

We make one final summarizing remark: our lower bounds from Claim 20 and Theorem 3 hold for *all* generators, but only for the specific collections that we consider, whereas the lower bound from (Kalavasis et al., 2025, Theorem 3.5) for generation with (exact) breadth holds for all non-identifiable collections, but only for a restricted class of generators (namely, those satisfying the MOP condition).

C.2.2. NECESSARY CONDITION FOR GENERATION WITH BREADTH

The collection in Example 2 exhibits a countable collection for which generation with breadth is not possible, but exhaustive generation is possible. Recall that we had argued above that this collection is not identifiable in the limit (the same adversary strategy employed in the proof above also foils any given identification algorithm). As we show next, a necessary condition for identifiability, which we denote as "Angluin's Condition with Existence", is also necessary for generation with breadth. This further illustrates how the notions of identification in the limit and generation with breadth are closely tied together.

Angluin's Condition with Existence (Condition 2 in Angluin (1980)):

A collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Angluin's Condition with Existence, if for every language $L \in \mathcal{C}$, there exists a finite subset $T \subseteq L$, such that every $L' \in \mathcal{C}$ that contains T satisfies that L' is not a proper subset of L. (4)

Corollary 1 in Angluin (1980) shows that the above condition is necessary for identification in the limit. The following proposition shows that it is also necessary for generation with breadth.

Proposition 21 *If a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ can be generated with breadth, then it satisfies Angluin's Condition with Existence.*

Proof Let us assume for the sake of contradiction that the collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ does not satisfy (4), but $\mathcal{A}$ is an algorithm that generates from languages in $\mathcal{C}$ with breadth. Then, there exists a language $L \in \mathcal{C}$, such that

$$\forall \text{ finite subsets } T \subseteq L, \exists\, L' \in \mathcal{C} \text{ that satisfies } T \subset L', \text{ and } L' \text{ is a proper subset of } L. \quad (5)$$

Fix an enumeration of all the strings in L, and let this be

$$L = \{x_1, x_2, \ldots\}. \tag{6}$$

Suppose that the adversary first presents x_1 as input. Then by (5), for $T_1 = \{x_1\}$, there exists $L_1 \in \mathcal{C}$ such that $T_1 \subset L_1$ and L_1 is a proper subset of L. Then, suppose that the adversary proceeds to enumerate strings in L_1 one by one in the order that they appear in the enumeration of L above, skipping over x_1. Since $\mathcal{A}$ generates with breadth, and the strings presented so far constitute a valid enumeration of L_1, there must exist some time $t_1^* < \infty$ such that for every $t \geq t_1^*$, it holds that $Z_{\geq t} = L_1$. Let $t_1 = t_1^*$. Suppose now that at time $t_1 + 1$, the adversary presents the string in $L \setminus L_1$ which has not been enumerated so far, and appears at the smallest index in the ordering in (6) above.

Now, consider the finite set S_{t_1+1} enumerated so far, and observe that $S_{t_1+1} \subset L$. So, let $T_2 = S_{t_1+1}$. Again by (5), there exists $L_2 \in \mathcal{C}$ such that $T_2 \subset L_2$ and L_2 is a proper subset of L. Since all the strings presented so far are contained in L_2, from time step $t_1 + 2$ onward, the adversary continues to enumerate all the strings in L_2 one by one in the order that they appear in (6), skipping over strings that have already been enumerated. Combined with the strings presented so far, this is a valid enumeration of L_2. Thus, there must exist some time $t_2^* < \infty$ such that for every $t \geq t_2^*$, it holds that $Z_{\geq t} = L_2$. Let $t_2 \geq t_2^*$ be such that $t_2 > t_1$. Now suppose that at time $t_2 + 1$, the adversary presents the string in $L \setminus L_2$ which has not been enumerated so far, and appears at the smallest index in the ordering in (6) above.

Now, consider the finite set S_{t_2+1} enumerated so far, and observe that $S_{t_2+1} \subset L$. So, let $T_3 = S_{t_2+1}$. Again by (5), there exists $L_3 \in \mathcal{C}$ such that $L_3 \supseteq T_3$, but also, L_3 is a proper subset of L. Thus, since all the strings presented so far are contained in L_3, from time step $t_2 + 2$ onward, the adversary continues to enumerate all the strings in L_3 one by one in the order that they appear in (6), skipping over strings that have already been enumerated. Combined with the strings presented so far, this is a valid enumeration of L_3. Thus, there must exist some time $t_3^* < \infty$ such that for every $t \geq t_3^*$, it holds that $Z_{\geq t} = L_3$. Let $t_3 \geq t_3^*$ be such that $t_3 > t_2$. Now suppose that at time $t_3 + 1$, the adversary inputs the string in $L \setminus L_3$ which has not been enumerated so far, and appears at the smallest index in the ordering in (6) above. ...

We can repeat this argument indefinitely, and observe that the adversary will have produced an enumeration of L. This is because $t_1 < t_2 < t_3 \ldots$, and at every time step $t_i + 1$, the adversary inputs the smallest indexed string in (6) that has not yet been enumerated. The condition that we maintain is that, when presented with the sequence up to time step t_i, the generator $\mathcal{G}_{t_i}$ output by $\mathcal{A}$ at t_i satisfies the generation with breadth guarantee for L_i; in particular, it maintains the invariant $Z_{\geq t_i} = L_i$.

Now, if $\mathcal{A}$ were to successfully generate with breadth from L, for the above enumeration of L, there must be some finite time step t^* such that for every $t \geq t^*$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $Z_{\geq t} = L$. Consider then the smallest j such that t_j is a time step beyond t^* in the above argument. By the invariant we have maintained, it must be the case that $Z_{\geq t_j} = L_j$. But note that L_j is a proper subset of L, and this contradicts the generation with breadth requirement for L that $Z_{\geq t_j} = L$. ∎

Remark 22 *We note that the only technical difference between Angluin's Condition with Existence* (4) *and Angluin's Condition with Enumeration* (3) *is the efficient computability (recursive enumerability) of the sets T for every language L. Angluin's Condition with Enumeration, which requires* *the T's to be computable, is also sufficient for generation with breadth, simply because it is sufficient for identification. However, we note that Angluin's Condition with Existence, which simply ensures existence of the T's, is not sufficient for identification in the limit (see Theorem 2 in Angluin (1980)). Therefore, if one were to show that Angluin's Condition with Enumeration (a stronger condition) is necessary for generation with breadth, then this would equate the two notions of identification in the limit and generation with breadth. On the other hand, showing that Angluin's Condition with Existence (a weaker condition) is sufficient for generation with breadth would separate the two notions.*

Appendix D. Characterization of Exhaustive Generation

Similar to Angluin's characterization (Theorem 18) for identification in the limit, in this section, we fully characterize exhaustive generation. First, in Section D.1, we introduce a weakening of Angluin's Condition with Existence, denoted as "Weak Angluin's Condition with Existence", and show that this condition characterizes the language collections that can be exhaustively generated. Namely, Proposition 23 shows that the condition is necessary for exhaustive generation to be possible at all. Proposition 25 shows that the condition is also sufficient for exhaustive generation, albeit with access to a somewhat strong oracle. Our next result (Proposition 27) in Section D.2 then shows that a slight strengthening of the condition, which we denote as "Weak Angluin's Condition with Enumeration", is sufficient for exhaustive generation with the standard membership query oracle.

D.1. Weakening of Angluin's Condition with Existence

We start by showing that the collections of languages that can be exhaustively generated from are *exactly* those collections that satisfy the following condition:

Weak Angluin's Condition with Existence:
A collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Weak Angluin's Condition with Existence, if for every language $L \in \mathcal{C}$, there exists a finite subset $T \subseteq L$, such that every $L' \in \mathcal{C}$ that contains T and is a proper subset of L satisfies $|L \setminus L'| < \infty$.

The following two propositions, respectively show that the above condition is necessary and sufficient for exhaustive generation, thereby establishing Theorem 4.

Proposition 23 (Exhaustive Generation Necessary Condition) *If a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ can be exhaustively generated, then it satisfies Weak Angluin's Condition with Existence.*

Proof Let us assume for the sake of contradiction that the collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ does not satisfy the condition, but $\mathcal{A}$ is an algorithm that exhaustively generates from languages in $\mathcal{C}$. Then, there exists a language $L \in \mathcal{C}$, such that:

$$\forall \text{ finite subsets } T \subset L, \exists\, L' \in \mathcal{C}, \text{ such that } T \subset L', L' \subsetneq L, \text{ but } |L \setminus L'| = \infty. \tag{7}$$

Fix an enumeration of all the strings in L, and let this be

$$L = \{x_1, x_2, \ldots\}. \tag{8}$$

Suppose that the adversary first presents x_1 as input. Then by (7), for $T_1 = \{x_1\}$, there exists $L_1 \in \mathcal{C}$ such that $T_1 \subset L_1$, L_1 is a proper subset of L, and $|L \setminus L_1| = \infty$. Then, suppose that the adversary proceeds to enumerate strings in L_1 one by one in the order that they appear in the enumeration of L above, skipping over x_1. Since $\mathcal{A}$ generates exhaustively for the collection, and the strings presented so far constitute a valid enumeration of L_1, there must exist some time $t_1^* < \infty$ such that for every $t \geq t_1^*$, it holds that $|Z_{\geq t} \setminus L_1| < \infty$. Let $t_1 = t_1^*$. Suppose now that at time $t_1 + 1$, the adversary presents the string in $L \setminus L_1$ that appears at the smallest index in the ordering in (8) within the set of strings that have not been enumerated so far.

Now, consider the finite set S_{t_1+1} enumerated so far, and observe that $S_{t_1+1} \subseteq L$. Let $T_2 = S_{t_1+1}$. Again by (7), there exists $L_2 \in \mathcal{C}$ such that $T_2 \subset L_2$, L_2 is a proper subset of L, and

$|L \setminus L_2| = \infty$. Since all the strings presented so far are contained in L_2, from time step $t_1 + 2$ onward, the adversary continues to enumerate all the strings in L_2 one by one in the order that they appear in (8), skipping over strings that have already been enumerated. Combined with the strings presented so far, this is a valid enumeration of L_2. Thus, there must exist some time $t_2^* < \infty$ such that for every $t \geq t_2^*$, it holds that $|Z_{\geq t} \setminus L_2| < \infty$. Let $t_2 \geq t_2^*$ be such that $t_2 > t_1$. Now suppose that at time $t_2 + 1$, the adversary presents the string in $L \setminus L_2$ that appears at the smallest index in the ordering in (8) within the set of strings which have not been enumerated so far.

Now, consider the finite set S_{t_2+1} enumerated so far, and observe that $S_{t_2+1} \subseteq L$. Let $T_3 = S_{t_2+1}$. Again by (7), there exists $L_3 \in \mathcal{C}$ such that $T_3 \subset L_3$, L_3 is a proper subset of L, and $|L \setminus L_3| = \infty$. Since all the strings presented so far are contained in L_3, from time step $t_2 + 2$ onward, the adversary continues to enumerate all the strings in L_3 one by one in the order that they appear in (8), skipping over strings that have already been enumerated. Combined with the strings presented so far, this is a valid enumeration of L_3. Thus, there must exist some time $t_3^* < \infty$ such that for every $t \geq t_3^*$, it holds that $|Z_{\geq t} \setminus L_3| < \infty$. Let $t_3 \geq t_3^*$ be such that $t_3 > t_2$. Now suppose that at time $t_3 + 1$, the adversary presents the string in $L \setminus L_3$ which appears at the smallest index in the ordering in (8) within the set of strings that have not been enumerated so far. ...

We can repeat this argument indefinitely, and observe that the adversary will have produced an enumeration of L. This is because $t_1 < t_2 < t_3 \ldots$, and at every time step $t_i + 1$, the adversary presents the string with smallest index in the ordering (8) that has not yet been enumerated. The condition that we maintain is that, when presented with the sequence up to time step t_i, the generator $\mathcal{G}_{t_i}$ output by $\mathcal{A}$ at t_i satisfies the exhaustive generation guarantee for L_i; in particular, it maintains the invariant $|Z_{\geq t_i} \setminus L_i| < \infty$.

Now, if $\mathcal{A}$ were to exhaustively generate from L, for the above enumeration of L, there must be some finite time step t^* such that for every $t \geq t^*$, the generator $\mathcal{G}_t$ output by $\mathcal{A}$ at t satisfies that $S_t \cup Z_{<t} \cup Z_{\geq t} \supseteq L$. Consider then the smallest j such that t_j is a time step beyond t^* in the above argument. By the exhaustive generation requirement for L, we must have that $S_{t_j} \cup Z_{<t_j} \cup Z_{\geq t_j} \supseteq L$. However, by the invariant we have maintained above, it is also the case that $|Z_{\geq t_j} \setminus L_j| < \infty$. Hence,

$$S_{t_j} \cup Z_{<t_j} \supseteq L \setminus Z_{\geq t_j} \supseteq L \setminus (L_j \cup Z_{\geq t_j}) = L \setminus (L_j \cup (Z_{\geq t_j} \setminus L_j)) = (L \setminus L_j) \setminus (Z_{\geq t_j} \setminus L_j). \tag{9}$$

But $S_{t_j} \cup Z_{t_j}$ and $Z_{\geq t_j} \setminus L_j$ are finite sets, while $L \setminus L_j$ is an infinite set, so (9) is impossible. ■

Remark 24 (Relaxed Exhaustive Generation) *We remark that essentially the same proof above also establishes that Weak Angluin's Condition with Existence is a necessary condition for a slightly more relaxed definition of exhaustive generation. Namely, consider replacing the two conditions in Definition 10, by the single condition $|Z_{\geq t} \,\Delta\, K| < \infty$, where Δ denotes the symmetric difference (i.e., $A \,\Delta\, B = (A \setminus B) \cup (B \setminus A)$). We can verify that the same contradiction in the proof above also goes through for this definition. However, we also observe that while Definition 10 implies this relaxed definition, the relaxed definition does not imply Definition 10 (namely, the relaxed definition does not necessarily satisfy the second condition in Definition 10).*

Proposition 25 (Exhaustive Generation Sufficient Condition) *If a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Weak Angluin's Condition with Existence, then it can be exhaustively generated, with access to an oracle which determines, for any i, j, whether $L_i \setminus L_j$ is finite.*

Proof We will show that the algorithm of Kleinberg and Mullainathan (2024), together with a slight modification, exhaustively generates in the limit. First, we recall their algorithm. Suppose that the target language being enumerated is L_z for $z \in \mathbb{N}$.

At time step t, the algorithm considers the languages in the subcollection $\mathcal{C}_t = \{L_i : 1 \le i \le t, L_i \supseteq S_t\}$, i.e., the languages among $L_1, \dots, L_t$ that are consistent with the input S_t enumerated so far. Let $\mathcal{C}_t = \{L_{i_1}, \dots, L_{i_{n_t}}\}$, where $i_1 < i_2 < \dots < i_{n_t}$. A language $L_{i_j} \in \mathcal{C}_t$ is termed *critical* if $L_{i_j} \subseteq L_{i_k}$ for every $k < j$. Consider the largest $j \le n_t$ such that the language $L_{i_j} \in \mathcal{C}_t$ is critical, and denote this j by $t^\star$—the algorithm generates a string from $L_{i_{t^\star}} \setminus S_t$.

We quickly review their proof of correctness for this algorithm. We can verify that there exists a large enough time step $t^+ \ge z$ such that for all $t \ge t^+$, language L_z is critical at time t (this is because L_z is always consistent with the input, and hence always belongs to $\mathcal{C}_t$ for $t \ge z$, and also, for every language L_i such that $i < z$ which satisfies that $L_z \not\subseteq L_i$, there is a string in $L_z \setminus L_i$ which eventually gets enumerated, and hence makes L_i inconsistent). Furthermore, we can also verify that for every $t \ge t^+$, the last critical language $L_{i_{t^\star}}$ that the algorithm chooses to generate from satisfies (by definition of criticality) that $L_{i_{t^\star}} \subseteq L_z$. This establishes correctness.

Our exhaustive generation algorithm closely follows this algorithm of Kleinberg and Mullainathan (2024). At time step t, let $L_{i_{t^\star}}$ be the language considered as above. Then, as we argued, for every $t \ge t^+$, we have that $L_{i_{t^\star}} \subseteq L_z$. Now, consider the finite set $T_z \subseteq L_z$ which is guaranteed to exist by (2). Observe crucially that there also exists a large enough t' such that for every $t \ge t'$, $T_z \subseteq S_t$. This is because T_z is a finite subset of L_z, and every string in L_z is guaranteed to show up in the input enumeration at some finite time. So, consider $t'' = \max(t', t^+)$.

We now claim that for every $t \ge t''$, the language $L_{i_{t^\star}}$, in addition to satisfying $L_{i_{t^\star}} \subseteq L_z$ (by the property of criticality), also additionally satisfies that $|L_z \setminus L_{i_{t^\star}}| < \infty$. To see this, suppose that $L_{i_{t^\star}} \subset L_z$ (if $L_{i_{t^\star}} = L_z$, then the claim is vacuously true). Because we have chosen a time step larger than t', the argument from the previous paragraph gives us that $T_z \subseteq S_t$. Furthermore, by definition of the algorithm, $S_t \subseteq L_{i_{t^\star}}$. Thus, $L_{i_{t^\star}}$ is a language satisfying $T_z \subseteq L_{i_{t^\star}}$, and also that $L_{i_{t^\star}} \subset L_z$. Therefore, (2) implies that $|L_z \setminus L_{i_{t^\star}}| < \infty$.

So, consider populating a language $Z_{\ge t}$ as follows. We initialize $Z_{\ge t} = L_{i_{t^\star}}$. In the collection of consistent languages $\mathcal{C}_t = \{L_{i_1}, \dots, L_{i_{t^\star}}, \dots, L_{n_t}\}$ considered at time step t, consider every $j \le t^\star$ such that

$$L_{i_j} \text{ is critical (and also a superset of } L_{i_{t^\star}}\text{), and also satisfies } |L_{i_j} \setminus L_{i_{t^\star}}| < \infty, \tag{10}$$

and update $Z_{\ge t} = Z_{\ge t} \cup (L_{i_j} \setminus L_{i_{t^\star}})$ for every such j.

Note that because $L_z \in \mathcal{C}_t$, and $z \le i_{t^\star}$, there is a j satisfying $i_j = z$ that will be considered. Furthermore, by the argument above, $|L_z \setminus L_{i_{t^\star}}| < \infty$, and hence this j will pass the condition (10). Because there are only finitely many $j \le t$, and for every j satisfying the condition, we only add finitely many strings to $Z_{\ge t}$, at the end of the procedure we ensure that 1) $Z_{\ge t} \supseteq L_z$, and also 2) $|Z_{\ge t} \setminus L_z| < \infty$.

The exhaustive generation algorithm outputs the generator $\mathcal{G}_t$, which simply enumerates the strings in $Z_{\ge t}$ that is constructed by the above procedure, if it is asked to go into generate-only mode at this time. By design, we have ensured that for all $t \ge t''$, both conditions required for exhaustive generation in Definition 10 are satisfied. ∎

Remark 26 *We note that the algorithm in the above proof of Proposition 25 also needs access to a* subset oracle *which, given indices i, j determines whether $L_i \subseteq L_j$. Kleinberg and Mullainathan*

(2024) show how to implement their algorithm using only membership queries, without needing such a subset oracle. The same ideas can be applied in our setting as well.

D.2. Weakening of Angluin's Condition with Enumeration

While the condition in (2) fully characterizes exhaustive generation, our algorithm above for collections satisfying this condition admittedly requires a more powerful oracle (for any i, j, it must be able to determine if $L_i \setminus L_j$ is finite) than the standard membership query oracle considered by Kleinberg and Mullainathan (2024). As our final result in this section, we show that a slight strengthening of (2) is sufficient for exhaustive generation with the standard membership query oracle. The strengthened condition requires the efficient computatability of the tell-tale sets in (2), and is yet another instantiation of the subtle difference between existence and enumerability of these sets.

Weak Angluin's Condition with Enumeration:

A collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Weak Angluin's Condition with Enumeration, if there exists a computable procedure, which for every language $L \in \mathcal{C}$, ouptuts an enumeration of a finite set T, such that $T \subseteq L$, and furthermore, every $L' \in \mathcal{C}$ that contains T and is a proper subset of L satisfies $|L \setminus L'| < \infty$. (11)

Proposition 27 *If a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ satisfies Weak Angluin's Condition with Enumeration, then it can be exhaustively generated with only membership oracle access to the language collection.*

Proof Suppose we have a collection $\mathcal{C} = \{L_1, L_2, \ldots\}$ that satisfies (11). We describe an algorithm for exhaustive generation with only membership query access to the language collection, inspired by Angluin's algorithm for language identification (in the proof of Theorem 1 in (Angluin, 1980)). Let $T_i^{(n)}$ denote the set of strings produced in the first n steps of the enumeration of T_i.

At time step n, the algorithm considers the languages in the subcollection $\mathcal{C}_n = \{L_i : 1 \le i \le n, L_i \supseteq S_n \supseteq T_i^{(n)}\}$, i.e., the languages L_i among $L_1, \ldots, L_n$ that are consistent with the input S_n enumerated so far, with the additional condition that all strings in $T_i^{(n)}$ have also appeared in the input. It is easy to check that $\mathcal{C}_n$ can be determined with only membership oracle access to the language collection. If $\mathcal{C}_n$ is empty, the algorithm sets $Z_{\ge n}$ arbitrarily. Otherwise, let g be the smallest index of a language in $\mathcal{C}_n$. The algorithm sets $Z_{\ge n}$ to be L_g.

We now establish correctness for this algorithm. Suppose that the target language being enumerated is L_z for $z \in \mathbb{N}$. For each $i \in \{1, \ldots, z\}$, we define n_i as follows: If $L_z \setminus L_i \neq \emptyset$, then n_i is the first time step when a string from $L_z \setminus L_i$ appears in the input. Otherwise, if $L_i \supseteq L_z$, n_i is the smallest value of n such that $T_i^{(n)} = T_i$. (Note that, in fact, $T_i^{(n)} = T_i$ for all $n \ge n_i$).

Consider any time step $n \ge \max\{n_i, 1 \le i \le z\}$. We will show that the algorithm satisfies the correctness guarantee for exhaustive generation. We claim that the subcollection $\mathcal{C}_n \cap \{L_1, \ldots L_z\}$ consists of precisely those languages $L_i, 1 \le i \le z$ such that $L_i \supseteq L_z$ and $L_i \setminus L_z$ is finite. Note that this also implies that $\mathcal{C}_n$ is non-empty since L_z satisfies this condition.

Suppose that $L_z \setminus L_i \neq \emptyset$. Since $n \ge n_i$, S_n contains a string in $L_z \setminus L_i$. Thus L_i is not a consistent language at time step n and is not included in $\mathcal{C}_n$. On the other hand, suppose $L_i \in \mathcal{C}_n$. By the previous argument, $L_i \supseteq L_z$. Since $n \ge n_i$, $T_i \subseteq T_i^{(n)} \subseteq S_n \subseteq L_z$. (The first set inclusion follows from the definition of n_i, the second from the definition of $\mathcal{C}_n$, and the third from the fact that L_z is the target language.) Applying (11) with $L' = L_z$, we conclude that $|L_i \setminus L_z| < \infty$.

Now consider the smallest index g of a language in $\mathcal{C}_n$. Recall that g exists since $\mathcal{C}_n$ is non-empty. By the claim we just established, $L_g \supseteq L_z$ and $L_g \setminus L_z$ is finite. Since the algorithm sets $Z_{\geq n} = L_g$, the algorithm satisfies the requirement for exhaustive generation. ■

Appendix E. Uniform Generation with Feedback

Towards this, we first propose an alternate combinatorial dimension which also characterizes uniform generation (without feedback) and show that this is equivalent to the closure dimension. We later generalize this combinatorial dimension and show that it characterizes uniform generation with feedback.

E.1. The GnF Dimension for Generation with no Feedback

First, we abstractly formalize some notions from the language generation setup, which will be particularly helpful in defining the dimension.

Transcript: A transcript is a record of interaction between an adversary (who is enumerating a language $K \in \mathcal{C}$) and a generator (who is trying to generate from the language). A transcript is an infinite sequence $x_1, z_1, x_2, z_2, x_3, z_3, \ldots$, where each $x_t \in K$; here, x_t is the input given to the generator at time step t, and z_t is the string generated by the generator at time step t. Note that the actions of the adversary and the generator are interleaved.

Adversary Strategy: An adversary strategy A is a mapping from prefixes of a transcript ending in an action by the generator (in this case, the last string generated by the generator) to the next action by the adversary (in this case, the next string from K to append to the enumeration). For any language $L \in \mathcal{C}$, an adversary strategy A is consistent with L if the sequence $x_1, x_2, \ldots$ is an enumeration of *all* the strings in L, i.e., for every $x \in L$, there is some finite index i such that $x_i = x$. Here again, we use the shorthand S_t to denote the set of distinct strings in the sequence $x_1, \ldots, x_t$.

Generator Strategy: A generator strategy G is a mapping from prefixes of a transcript ending in an action by the adversary to an action by the generator.

For an adversary strategy A and generator strategy G, the transcript of interaction between A and G, denoted $T(A, G)$, is the infinite sequence $x_1, z_1, x_2, z_2, \ldots$, where

$$\begin{aligned} &x_1 = A(\emptyset) \\ &z_1 = G(x_1) \\ &x_2 = A(x_1, z_1) \\ &z_2 = G(x_1, z_1, x_2) \\ &\vdots \end{aligned}$$

Consistent Languages: For a collection $\mathcal{C}$ of languages, generator strategy G, and adversary strategy A consistent with some language $K \in \mathcal{C}$, consider the transcript $T = T(A, G)$. For $r \in \mathbb{N}$, we say that a language $L \in \mathcal{C}$ is consistent with T upto round r if $x_t \in L$ for every $t \leq r$. Let $\mathcal{C}_r(T)$ denote the subset of $\mathcal{C}$ comprising of languages consistent with T upto round r. Note that $K \in \mathcal{C}_r(T)$ for every $r \in \mathbb{N}$, and hence $\mathcal{C}_r(T)$ is never empty.

Effective Intersection: We define the effective intersection at round r, denoted $E_r(T)$, as

$$E_r(T) = \left\{ \bigcap_{L \in \mathcal{C}_r(T)} L \right\} \setminus S_r. \tag{12}$$

We are now ready to define a complexity measure which we term the Generation-no-Feedback (GnF) dimension.

Definition 28 (GnF **dimension**) *The* GnF *dimension of a collection $\mathcal{C}$ is the supremum over $d \in \mathbb{N}$, for d satisfying the following property: for every generator strategy G, there exists a language $K \in \mathcal{C}$ and an adversary strategy A consistent with K such that, in the transcript $T = T(A, G)$, there exists a finite $r \geq d$ where $|S_r| \geq d$ and effective intersection $E_r(T) = \emptyset$.*

We first show that the GnF dimension characterizes uniform generation, by equating it to the closure dimension from the work of Li et al. (2024).

Definition 29 (Closure dimension) *The closure dimension of a collection $\mathcal{C}$ is the size of the largest set $S = \{x_1, \ldots, x_d\}$ such that the intersection of languages in $\mathcal{C}$ that contain S is finite.*

Proposition 30 *For any collection $\mathcal{C}$, the* GnF *dimension of $\mathcal{C}$ is equal to its closure dimension.*

Proof Consider any $1 \leq d < \infty$, and suppose that the GnF dimension of $\mathcal{C}$ is at least d. This means that there exists $d' \geq d$, such that for every generator strategy G, there exists a language $K \in \mathcal{C}$ and an adversary strategy A consistent with K such that in the transcript $T = T(A, G)$, there exists a finite $r \geq d'$ such that $|S_r| \geq d'$ and $E_r(T) = \emptyset$. Then, fix an arbitrary generator strategy, and consider the $K \in \mathcal{C}$ and adversary strategy A consistent with K that satisfy this property. In particular, consider the time step r in the transcript $T(A, G)$ at which $|S_r| \geq d'$ and $E_r(T) = \emptyset$. Recall that $\mathcal{C}_r(T)$ is not empty (it contains K), and furthermore, every language in $\mathcal{C}_r(T)$ contains S_r. Hence, since $E_r(T) = \emptyset$, the definition of $E_r(T)$ (see (12)) implies that

$$\bigcap_{L \in \mathcal{C}_r(T)} L = S_r.$$

But note that the languages in $\mathcal{C}_r(T)$ are exactly those languages that are consistent with S_r (i.e., they contain S_r), and furthermore, S_r is a finite set. Thus, we have that the languages in $\mathcal{C}$ containing S_r have finite intersection. Furthermore, S_r is a set of size at least $d' \geq d$. This implies that the closure dimension of $\mathcal{C}$ is at least $d' \geq d$.

Now, suppose that the closure dimension of $\mathcal{C}$ is at least d. This means that we can find a set S of size $d' \geq d$, such that the intersection of languages in $\mathcal{C}$ that contain S is finite. In particular, let

$$\bigcap_{L \in \mathcal{C}, L \supseteq S} L = \{x_1, \ldots, x_{d''}\}, \tag{13}$$

where $d' \leq d'' < \infty$. Now, fix any generator strategy G. Choose any $L \in \mathcal{C}$ that satisfies $L \supseteq S$ to be the target language K—we know that K contains $\{x_1, \ldots, x_{d''}\}$ from (13). Then, consider the adversary strategy A which, in the first d'' rounds, enumerates $x_1, \ldots, x_{d''}$, and then continues to enumerate the rest of the strings in K, irrespective of the the generator's actions. Then, we have that $S_{d''} = \{x_1, \ldots, x_{d''}\}$, and $|S_{d''}| = d'' \geq d' \geq d$. Furthermore, we claim that $\cap_{L \in \mathcal{C}_{d''}(T)} L = S_{d''}$, which would imply that $E_{d''}(T) = \emptyset$. To see this, observe that $\mathcal{C}_{d''}(T) = \{L \in \mathcal{C} : L \supseteq S_{d''}\}$. Let $\mathcal{C}_1 = \{L \in \mathcal{C} : L \supseteq S\}$. Observe that if $L \supseteq S$, $L \supseteq \{x_1, \ldots, x_{d''}\} = S_{d''}$ by (13). Thus, $\mathcal{C}_1 \subseteq \mathcal{C}_{d''}(T)$. Furthermore, if $L \supseteq S_{d''}$, then $L \supseteq S$, since $S_{d''} \supseteq S$. Thus, we also have that $\mathcal{C}_{d''}(T) \subseteq \mathcal{C}_1$, implying that $\mathcal{C}_{d''}(T) = \mathcal{C}_1$. This gives that

$$\bigcap_{L \in \mathcal{C}_{d''}(T)} L = \bigcap_{L \in \mathcal{C}_1} L = \{x_1, \ldots, x_{d''}\} = S_{d''},$$

where in the second equality, we used (13) again. Thus, we have obtained an $r = d'' \geq d$ such that $|S_r| \geq d'' \geq d$ and $E_r(T) = \emptyset$. Since this holds regardless of the generator strategy, we have that the GnF dimension of $\mathcal{C}$ is at least $d'' \geq d$.

The above argument shows that, for any finite $d \geq 1$, the closure dimension of $\mathcal{C}$ is at least d if and only if the GnF dimension of $\mathcal{C}$ is at least d. Thus, either both the closure and GnF dimension of $\mathcal{C}$ are finite and equal, or they are both unbounded. ■

Therefore, the GnF dimension is equivalent to the closure dimension and hence, characterizes uniform generation. However, this alternate formulation of the closure dimension in terms of abstract adversary and generator strategies allows us to generalize to the case where the generator is additionally allowed to query membership of a string in the target language at each time step.

E.2. The GF Dimension for Generation with Feedback

We systematically extend the notions defined in the previous section to account for queries issued by the generator.

Transcript: A transcript is now an infinite sequence $(x_t, y_t, a_t, z_t)_{t \in \mathbb{N}}$, here, at time step t, x_t is the input given by the adversary to the generator, $y_t \in \Sigma^*$ is the query issued by the generator, $a_t \in \{\text{Yes}, \text{No}\}$ is the response given by the adversary to the membership query, and z_t is the string generated by the generator. Note that the actions of the adversary and the generator are still interleaved.

Adversary Strategy: An adversary strategy A is still a mapping from prefixes of a transcript ending in an action by the generator (either the last string generated, or the membership query issued by the generator) to the next action by the adversary (either the next string to append to the enumeration, or a Yes/No response). For any language $L \in \mathcal{C}$, an adversary strategy A is consistent with L if (1) the sequence $x_1, x_2, \dots$ is an enumeration of *all* the strings in L, i.e., for every $x \in L$, there is some finite index t such that $x_t = x$, and (2) for all $t \in \mathbb{N}$, the response $a_t = \text{Yes}$ if $y_t \in L$, and No otherwise. The shorthand S_t still denotes the set of distinct strings in the input sequence $x_1, \dots, x_t$.

Generator Strategy: Similarly, a generator strategy G is still a mapping from prefixes of a transcript ending in an action by the adversary to an action by the generator.

For an adversary strategy A and generator strategy G, the transcript $T(A, G)$ of interaction between A and G is now the infinite sequence $(x_t, y_t, a_t, z_t)_{t \in \mathbb{N}}$, where

$$
\begin{aligned}
x_1 &= A(\emptyset) \\
y_1 &= G(x_1) \\
a_1 &= A(x_1, y_1) \\
z_1 &= G(x_1, y_1, a_1) \\
x_2 &= A(x_1, y_1, a_1, z_1) \\
y_2 &= G(x_1, y_1, a_1, z_1, x_2) \\
a_2 &= A(x_1, y_1, a_1, z_1, x_2, y_2) \\
z_2 &= G(x_1, y_1, a_1, z_1, x_2, y_2, a_2) \\
&\vdots
\end{aligned}
$$

We can now state the definition for uniform generation with feedback.

Definition 31 (Uniform Generation with Feedback) *A collection $\mathcal{C}$ can be uniformly generated from with feedback if there exists a generator strategy G and a constant $t^* = t^*(\mathcal{C})$, such that for every language $K \in \mathcal{C}$ and for every adversary strategy A consistent with K, in the transcript $T(A, G)$, $z_t \in K \setminus S_t$ for every t satisfying $|S_t| \geq t^*$.*

Consistent Languages: For a collection $\mathcal{C}$ of languages, generator strategy G, and adversary strategy A consistent with some language $K \in \mathcal{C}$, consider the transcript $T = T(A, G)$. For $r \in \mathbb{N}$, we say that a language $L \in \mathcal{C}$ is consistent with T upto round r if (1) $x_t \in L$ for every $t \leq r$, and (2) $a_t = \text{Yes}$ if $y_t \in L$, and No otherwise for every $t \leq r$. Let $\mathcal{C}_r(T)$ denote the subset of $\mathcal{C}$ comprising of languages consistent with T upto round r. Note that $K \in \mathcal{C}_r(T)$ for every $r \in \mathbb{N}$, and hence $\mathcal{C}_r(T)$ is never empty.

Effective Intersection: The definition of the effective intersection $E_r(T)$ at round r remains the same as given in (12).

With these changes to the notions of adversary and generator strategies, the GF dimension is defined similarly as in Definition 28. We restate it from the main body.

Definition 15 (GF **dimension**) *The* GF *dimension of a collection $\mathcal{C}$ is the supremum over $d \in \mathbb{N}$, for d satisfying the following property: for every generator strategy G, there exists a language $K \in \mathcal{C}$ and an adversary strategy A consistent with K, such that in the transcript $T = T(A, G)$, there exists a finite $r \geq d$ where $|S_r| \geq d$ and effective intersection $E_r(T) = \emptyset$.*

Remark 32 *One reason behind stating our definition of* GF *dimension directly in terms of adversary/generator strategies, and not in terms of a property about intersection of languages containing a set of strings as in Li et al. (2024), is because of the ability of the generator to make membership queries on previously unseen strings. Essentially, there is additional complexity in controlling the intersection of consistent languages, which may drastically change based on queries that the generator may ask.*

We showed that the GnF dimension characterizes uniform generation (without feedback), by relating it to the closure dimension. Here, we directly argue that the GF dimension characterizes uniform generation with feedback. This follows from the following two lemmas:

Lemma 33 (GF **Dimension Upper Bound**) *If the* GF *dimension of a collection $\mathcal{C}$ is finite, it can be uniformly generated with feedback.*

Proof Suppose the GF dimension of $\mathcal{C}$ is $d < \infty$. This means that for every $d' > d$, the property in Definition 15 is *not* satisfied for d'. In particular, substituting $d' = d + 1$, we get that: there exists a generator strategy G, such that for every language $K \in \mathcal{C}$ and adversary strategy A consistent with K, in the transcript $T = T(A, G)$, for every $r \geq d + 1$, either $|S_r| < d + 1$ or $E_r(T) \neq \emptyset$. But note that any adversary strategy consistent with K must eventually satisfy $|S_r| \geq d + 1$ for all large enough $r \geq d + 1$—this follows from the requirement that the adversary must enumerate all the strings in K. So, fix the first such $r' \geq d + 1$ where $|S_{r'}| = d + 1$. Then, for all $r \geq r'$, this generator strategy satisfies that $E_r(T) \neq \emptyset$, which means that there exists a string in $K \setminus S_r$ that the generator can generate. Thus, the collection $\mathcal{C}$ can be uniformly generated with feedback by this generator with $t^*(\mathcal{C}) = d + 1$. ■

Lemma 34 (GF Dimension Lower Bound) *If the* GF *dimension of a collection $\mathcal{C}$ is infinite, it cannot be uniformly generated with feedback.*

Proof Suppose a generator strategy G' claims to uniformly generate from languages in $\mathcal{C}$ (with feedback) as soon as it sees $t^* = t^*(\mathcal{C})$ distinct strings. We will show that there is a language $L \in \mathcal{C}$, and an adversary strategy A' consistent with L, such that in the transcript $T(A', G')$, the string generated by G' at some time step r where $|S_r| \geq t^*$ does not belong to $L \setminus S_r$. This would violate the uniform generation guarantee of G'.

Towards this, note that because the GF dimension of $\mathcal{C}$ is infinite, we can find some $d \geq t^*$, such that the following property holds: for every generator strategy G, there exists a language $K \in \mathcal{C}$ and an adversary strategy A consistent with K, such that in the transcript $T = T(A, G)$, there exists a finite $r \geq d$ where $|S_r| \geq d$ and $E_r(T) = \emptyset$. So, let us choose the language $K \in \mathcal{C}$ and adversary strategy A consistent with K for the particular generator G' from above. Then, we know that for some $r \geq d \geq t^*$, it is the case that $|S_r| \geq d \geq t^*$, and $E_r(T(A, G')) = \emptyset$. In particular, consider the collection $\mathcal{C}_r(T(A, G'))$—this collection is non-empty because K belongs to it. Let z_r be the string generated by G' at time step r. If $z_r \in S_r$, G' is not a valid generator. So, suppose that $z_r \notin S_r$. Because $E_r(T(A, G')) = \emptyset$, this means that

$$\bigcap_{L \in \mathcal{C}_r(T(A,G'))} L = S_r.$$

Thus, we also have that $z_r \notin \bigcap_{L \in \mathcal{C}_r(T(A,G'))} L$. In particular, this means that there exists some $L \in \mathcal{C}_r(T(A, G'))$ such that $z_r \notin L$. But now, consider the adversary strategy A', which makes the same actions as A up until time step r, but from time step $r+1$ onward, continues to arbitrarily complete the enumeration of L, responding to any queries according to membership in L. Note that A' is consistent with L, since $L \in \mathcal{C}_r(T(A, G'))$, which means that all the strings enumerated as well as the answers to any membership queries given by A up until r are consistent with every language in $\mathcal{C}_r(T(A, G'))$. Furthermore, the transcript $T(A', G')$ is identical to $T(A, G')$ in the first r rounds. However, $z_r \notin L \setminus S_r$. Thus, we have obtained a language L and an adversarial strategy A' consistent with L, such that at a time $r \geq t^*$ where $|S_r| \geq t^*$, in the transcript $T(A', G')$, the string z_r generated by G' does not belong to $L \setminus S_r$. This contradicts the uniform generation guarantee of G'. ■

Recall that Example 1 demonstrated a collection where uniform generation without feedback was not possible, but uniform generation with feedback was. We conclude this section with an example of a language collection that cannot be uniformly generated with or without feedback, but can still be non-uniformly generated.

Example 3 *Let $\Sigma^* = \mathbb{Z}$. For any $i \in \mathbb{Z}$, let $L_i = \mathbb{Z} \setminus \{i\}$, and for any $i, j \in \mathbb{Z}$ such that $i < j$, let $L_{ij} = \mathbb{Z} \setminus \{i, j\}$. Consider the (countable) collection $\mathcal{C} = \bigcup_{i \in \mathbb{Z}} L_i \cup \bigcup_{i<j \in \mathbb{Z}} L_{ij}$. We will argue that $\mathcal{C}$ cannot be uniformly generated, with or without feedback, but can be non-uniformly generated.*

$\mathcal{C}$ cannot be uniformly generated without feedback because $\mathcal{C}$ has infinite closure dimension—this can be seen, for example, by observing that for any $d \geq 1$, the intersection of languages in $\mathcal{C}$ containing the set $\{1, 2, \ldots, d\}$ is exactly $\{1, 2, \ldots, d\}$.

$\mathcal{C}$ cannot be uniformly generated even with feedback, because $\mathcal{C}$ has infinite GF *dimension. To see this, fix any $d \geq 1$, and generator strategy G. Consider the adversary strategy A that*

operates as follows: let x_t be the example input by the adversary, and y_t be the example queried by the generator at time step t. Furthermore, let S_t and Q_t denote the sets of distinct elements in $x_1, \dots, x_t$ and $y_1, \dots, y_t$ respectively. For $t = 1, \dots, d$, the input x_t is decided as follows: if $y_{t-1} \in S_{t-1}$, then x_t is a distinct number not contained in S_{t-1}. Else if $y_{t-1} \notin S_{t-1}$, then $x_t = y_t$. Next, for $t = 1, \dots, d-1$, the answers a_t to all queries are Yes. *However, the answer a_d to the last query is decided based on y_d: if $y_d \in S_d$, then $a_d =$ Yes, else if $y_d \notin S_d$, $a_d =$ No. This specifies the adversary strategy up until d rounds. Note that $|S_d| = d$ by construction. We must now choose a language $K \in \mathcal{C}$ consistent with this strategy, and then specify how the adversary strategy continues beyond round d.*

First, consider the case that the adversary answered the last query with a Yes. *Observe in this case that $Q_d \subseteq S_d$, and recall that we answered all queries in Q_d with a* Yes. *We choose K to be any language in $\mathcal{C}$ that contains S_d, and the adversary A continues to enumerate K (and answers any queries according to K) beyond round d. Then, in the transcript $T = T(A, G)$ generated, $\mathcal{C}_d(T)$ comprises of all the languages in $\mathcal{C}$ that do not exclude any element in S_d. Furthermore, the intersection of all these languages is exactly S_d, because any element outside of S_d is excluded by at least one language in $\mathcal{C}_d(T)$. This means that $E_d(T) = \emptyset$.*

Next, consider the case that the adversary answered the last query with a No. *This means that the last query y_d that the generator issued does not belong to S_d. Note however that $Q_{d-1} \subseteq S_d$, and A answered all the queries in Q_{d-1} with a* Yes. *Hence, we must choose a language in $\mathcal{C}$ that contains all of S_d, but excludes y_d. Choose the target language K to be any $L_{ij} \in \mathcal{C}$ that contains S_d, and for which $i = y_d$. The adversary continues to enumerate K (and answers any queries according to K) beyond round d. Then, in the transcript $T = T(A, G)$ generated, $\mathcal{C}_d(T)$ comprises of all the languages in $\mathcal{C}$ that contain S_d, but exclude y_d. In particular, $\mathcal{C}_d(T)$ comprises of all languages L_{ij} that contain S_d, for which one of i or j is equal to y_d, and the other is any number in $\mathbb{Z} \setminus S_d$. This means that the intersection of all languages in $\mathcal{C}_d(T)$ is again exactly S_d, which means that $E_d(T) = \emptyset$.*

Since the above argument holds for any $d \geq 1$, the GF *dimension of $\mathcal{C}$ is infinite.*

Finally, we argue that $\mathcal{C}$ can be non-uniformly generated (without any feedback). This follows from Theorem 12, and by the fact that $\mathcal{C}$ is countable. Even more directly, consider the generator that simply generates the sequence $0, -1, 1, -2, 2, -3, 3, -4, 4, \dots$, while always skipping any elements that have shown up in the input. Observe that there is a finite time step $t(\mathcal{C}, K)$ (which is $O(|i|)$ if the true language K is L_i, or $O(\max(|i|, |j|))$ if it is L_{ij}) beyond which the generator is sure to have "skipped past" the excluded elements in K, independent of its order of enumeration. Thus, such a generator non-uniformly generates from the collection.