Space-Efficient Language Generation in the Limit

Nicolas Flammarion NICOLAS.FLAMMARION@EPFL.CH
EPFL, Switzerland

Chirag Pabbaraju CPABBARA@STANFORD.EDU
Stanford University, USA

Hristo Papazov HRISTO.PAPAZOV@EPFL.CH
EPFL, Switzerland

Miltiadis Stouras MILTIADIS.STOURAS@EPFL.CH
EPFL, Switzerland

Ola Svensson OLA.SVENSSON@EPFL.CH
EPFL, Switzerland

Editors: Steve Hanneke and Tor Lattimore

Abstract

We initiate a resource-aware theory of *language generation in the limit* under the minimal constraint of space efficiency. In our framework, a learner observes an adversarial positive stream from a target language K and must eventually output a hallucination-free hypothesis language $L \subseteq K$ while omitting at most Δ strings of K. We focus on $\mathcal{C}_{s,k}$, the collection of languages recognized by DFAs with at most s states over an alphabet of size k, as the natural hypothesis class for memory-bounded learners. In the exponential-space regime, we prove that a learner can exactly identify the target K. Under a stricter memory budget, we characterize the strongest possible generation guarantees. In particular, we present a streaming algorithm using $\mathrm{poly}(s,k)$ space that converges to a hypothesis with generation gap $\Delta = O(k^{2s-2})$. Moreover, the learned hypothesis captures every string in K of length at least $2s-1$. We complement this result with a near-matching lower bound through a reduction from a standard communication complexity problem. Specifically, achieving generation gap $\Delta \leq k^{(1-\varepsilon)s}$ requires $k^{\Omega(\varepsilon s)}$ memory. Together, these results reveal a sharp transition between polynomial-space generation and exponential-space exact identification.

Keywords: Language Generation in the Limit, Finite-State Automata, Computational Efficiency

1. Introduction

Large language models (LLMs) have been shown to generate novel and grammatically well-formed text even after training on exclusively *positive* natural-language examples (Radford et al., 2019; Brown et al., 2020; Mahowald et al., 2024). This remarkable empirical success revives a classical question from computational learning theory: What kind of language acquisition becomes possible when a learner only observes valid strings from the target language with no explicit negative feedback?

Identification in the Limit. A canonical formalization of this positive-only setting comes from Gold's work on learning in the limit. Motivated by the now-contentious[1] claim from psycholinguistics that infants learn the grammar of a language solely from positive examples, Gold (1967) developed the learning-in-the-limit framework as a minimalist formalization of language learning from positive

1. Consider (Marcus, 1993) for arguments in favor of language acquisition solely from positive examples and see (Chouinard and Clark, 2003) for arguments for the presence of negative examples in parent-infant communication.

data. Gold's model fixes (i) a representation system for languages, (ii) a target language, and (iii) an infinite stream that eventually lists every string in the target language. After each received valid string from this language presentation, a learner outputs a hypothesis language. Gold defined a language family as *identifiable in the limit* when a learner eventually stabilizes on a hypothesis that matches the ground-truth exactly, regardless of the target language and the presentation order. Unfortunately, under this learning criterion, Gold (1967) proved that only highly restricted[2] language collections are learnable. In particular, one cannot even learn the class of regular languages. These impossibility results largely redirected mainstream learning theory toward distributional, sample-complexity, and efficiency guarantees, crystallized in PAC learning (Valiant, 1984; Kearns and Vazirani, 1994).

Generation in the Limit. Only recently, Kleinberg and Mullainathan (2024) revitalized Gold's positive presentation framework by proposing a different language-acquisition criterion that fits language modeling more closely. Specifically, the learner still receives an arbitrary enumeration of the target language, but success no longer requires eventual equality between hypothesis and target. Instead, after some finite time, the learner must begin to generate infinitely many unseen strings from the target language. Put informally, sufficient exposure to positive data should make hallucinations disappear while preserving an infinite learned subset of the target language. This shift replaces identification with generation by relaxing the objective from equality to containment. Surprisingly, the relaxed objective substantially alters the theory: Now, under oracle access to membership queries for the enumerated language family, generation in the limit becomes achievable for every countable collection of languages.[3] Kleinberg and Mullainathan (2024)'s positive result has since inspired a growing line of follow-up work examining different aspects of generation/learning in the limit (Kalavasis et al., 2024; Li et al., 2024; Charikar and Pabbaraju, 2024; Papazov and Flammarion, 2025; Charikar and Pabbaraju, 2025; Peale et al., 2025; Raman and Raman, 2025; Hanneke et al., 2025; Charikar et al., 2025). Interestingly, the current generation-in-the-limit literature largely treats computation as free. As a result, generation guarantees rest on models that place no bounds on computation and, in particular, allow unbounded memory. Such assumptions diverge from practice, where human learners operate without external storage and language models train and run under explicit memory budgets.

Space-Efficient Generation. Motivated by this explanatory gap, our paper initiates a resource-aware study of generation in the limit by imposing the minimal computational restriction of space efficiency. Under our proposed framework, (i) the learner receives a stream of positive examples through an online interface, (ii) processes each target string symbol-by-symbol while operating within a strict memory budget, and (iii) after each processed example, outputs a hypothesis representation that must eventually generate a subset of the target language. Now, the requirement of bounded memory implies that our learning algorithms can only occupy finitely many internal configurations. Therefore, membership verification for the streamed examples can only range over regular languages, and only over a finite set of such languages. Accordingly, space-efficient generation only admits a meaningful formulation over finite collections of regular languages. Despite this restriction, regular-language generation remains practically significant. Specifically, in Appendix A, we argue that the collection of *communicative languages*, which arise from the interaction of space-bounded agents such as humans (Miller and Chomsky, 1963), forms a strict subset of regular languages.

Concurrently and independently, Kleinberg et al. (2026) also study generation in the limit under bounded-memory restrictions. Their results are incomparable to ours: they consider broader language

2. See (Angluin, 1980, Theorem 1) for a precise characterization of identifiable collections in the limit.
3. Of course, if we insist on computable membership queries, then the scope of the generatable collections decreases.

collections with coarser memory models, where the learner can only retain a limited window or buffer of past examples. In contrast, we impose an explicit bit-space budget on the learners and focus on regular languages ($C_{s,k}$). This lets us prove quantitative space–breadth tradeoffs, including tight bounds on the generation gap achievable with polynomial space.

Overview and Contributions. The regularity arguments above justify restricting attention to regular hypothesis spaces parameterized by automaton complexity. For a fixed alphabet Σ with $|\Sigma| = k$ and a state bound s, let $\mathcal{C}_{s,k}$ denote the collection of languages recognizable by deterministic finite automata (DFAs) over Σ with at most s states. The remainder of the paper characterizes space-efficient generation within $\mathcal{C}_{s,k}$. Upper bounds give explicit streaming learners operating in $\mathrm{poly}(s, k)$ space, while lower bounds show that stronger generation guarantees require exponential memory. These results expose a sharp tradeoff between memory and generative breadth, yielding a resource-sensitive theory of generation in the limit.

Roadmap. Section 2 formalizes the space-efficient generation model and describes our main results. Section 3 presents a space-efficient generation algorithm for $\mathcal{C}_{s,k}$. Section 4 proves matching lower bounds via communication complexity reductions. Section 5 concludes with implications for resource-bounded language acquisition and open directions.

2. Space-Efficiency Framework and Main Results

As described in Section 1, we study the problem of space-efficient generation in the limit where an adversary enumerates $w_1, w_2, w_3, \ldots$ and presents a target language $K \in \Sigma^*$ in a *streaming* fashion. Each string w_t arrives one symbol at a time, with a designated delimiter separating consecutive strings. After each symbol, the learning algorithm $\mathcal{A}$ performs computation under a polynomial space budget, and after each processed string $w_t \in K$, $\mathcal{A}$ outputs a hypothesis representation of a regular language.

Concretely, for the rest of the paper, we fix an alphabet Σ of size k and let $\mathcal{C}_{s,k}$ denote the collection of all regular languages over Σ recognizable by DFAs of at most s states. As a quick reminder, we recall the textbook definition of a DFA (Sipser, 1996; Hopcroft et al., 2001).

Definition 1 (Deterministic Finite-State Automaton) *A DFA A over the finite alphabet Σ is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of accepting (final) states. We extend the transition function δ to act on Σ^* by setting $\delta(q, \varepsilon) = q$ (where ε is the empty string) and recursively defining $\delta(q, \sigma w) := \delta(\delta(q, \sigma), w)$, for every $q \in Q, \sigma \in \Sigma$ and $w \in \Sigma^*$. We use the notation $L(A)$ to denote the regular language accepted by the DFA A, i.e., $L(A) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$, and we define the size of the DFA A as $|A| = |Q|$.*

We now state our main definition of space-efficient language generation in the limit.

Definition 2 (Space-Efficient Generation in the Limit) *A space-efficient algorithm $\mathcal{A}$ generates in the limit from $\mathcal{C}_{s,k}$ with a generation gap $\Delta_{\mathcal{A}}(s, k)$ if (i) $\mathcal{A}$ uses at most $\mathrm{poly}(s, k)$ bits of working memory, and (ii) for any target regular language $K \in \mathcal{C}_{s,k}$ and any surjective enumeration $w : \mathbb{N} \twoheadrightarrow K$*

presented to $\mathcal{A}$ in a streaming fashion, there exists a finite time $t^\star$ such that for all $t \geq t^\star$, $\mathcal{A}$ outputs a representation of a DFA $\mathcal{A}(t)$[4] *satisfying $L(\mathcal{A}(t)) \subseteq K$* [5] *and $|K \setminus L(\mathcal{A}(t))| \leq \Delta_{\mathcal{A}}(s,k)$.*

The literature refers to the requirement of outputting a language representation satisfying $L(\mathcal{A}(t)) \subseteq K$ as *index-based generation in the limit* (Kleinberg and Wei, 2025). While index-based generation with an unbounded generation gap admits a trivial solution, for example by repeatedly outputting the singleton language w for some short string $w \in K$, such a strategy fails to approximate K in any rich or informative sense. In particular, constructing an infinite sublanguage of K already poses a nontrivial challenge. This paper studies a substantially stronger objective: generation in the limit under a bounded generation gap, where the output language differs from K on only finitely many strings, thereby capturing almost the entire target language.

Without an *a priori* bound s on the size of a DFA recognizing the target language, bounded-gap generation in the limit remains impossible, even without memory constraints. More formally, Kalavasis et al. (2024, Theorem 3.10) show that any collection admitting bounded-gap[6] generation in the limit must satisfy the so-called *Weak Angluin's Condition*. In particular, regular languages violate the weak Angluin's condition. Consequently, bounded-gap generation in the limit fails over the full class of regular languages (without any *a priori* state bound), even with unlimited memory.

Interestingly, Angluin (1980) proved that any finite collection of languages allows identification in the limit. Hence, without the polynomial-space requirement, we can completely remove the generation gap and *identify* $\mathcal{C}_{s,k}$ in the limit. We describe such an algorithm in Section 3; here, we simply note that identification crucially relies on using exponential space. Then, how does the picture change with polynomial space constraints? Perhaps surprisingly, our first result shows that there exists a space-efficient algorithm that generates from $\mathcal{C}_{s,k}$ in the limit with a bounded generation gap.

Theorem 3 (Space-Efficient Generation in the Limit) *There exists a space-efficient learning algorithm $\mathcal{A}$ that generates in the limit from $\mathcal{C}_{s,k}$ with a generation gap $\Delta_{\mathcal{A}}(s,k) \leq O(k^{2s-2})$.*

In fact, the algorithm satisfies a stronger guarantee: The output language omits only target strings of length at most $2s-2$. At a high-level, our algorithm traverses $\mathcal{C}_{s,k}$ according to a predefined topological order and outputs the first language not proven inconsistent with the most recently observed input string. At the next input, the learner continues traversing $\mathcal{C}_{s,k}$ starting from the previously outputted language. The topological order ensures that the algorithm converges to a language with a finite symmetric difference with the target language. We make the traversal space-efficient by using a recursion technique inspired by Savitch's famous theorem in complexity theory (Savitch, 1970). Finally, to achieve generation in the limit while minimizing the generation gap, we invoke results from the automata-minimization literature (Gawrychowski et al., 2011) that bound the size of the finite symmetric difference between the learner's hypothesis and the target.

Our space-efficient algorithm nonetheless incurs a nonzero generation gap of $O(k^{2s-2})$. This bound raises a natural question: Does polynomial-space learning permit a zero generation gap, or, equivalently, identification in the limit over $\mathcal{C}_{s,k}$? The next theorem rules out this possibility. In fact, we prove something much stronger: Any learner that operates with sub-exponential space must incur an exponential generation gap.

4. We use $\mathcal{A}(t)$ as a shorthand for $\mathcal{A}(w_1, \ldots, w_t)$.
5. One can also formulate the objective without the hard no-hallucination requirement by asking directly for a bounded symmetric difference $|L(\mathcal{A}(t)) \triangle K| \leq \Delta_{\mathcal{A}}(s,k)$. The upper and lower bounds in this paper remain the same under that relaxed formulation as we show in Sections 3 and 4.
6. Kalavasis et al. (2024) refer to this notion as generation with "approximate breadth".

Theorem 4 (Space–Breadth Tradeoff) *For any $\varepsilon > 0$, any algorithm $\mathcal{A}$ generating in the limit from $\mathcal{C}_{s,k}$ with a generation gap $\Delta_{\mathcal{A}}(s,k) \leq k^{(1-\varepsilon)s}$ must use $k^{\Omega(\varepsilon s)}$ memory bits. Moreover, any algorithm $\mathcal{A}$ achieving the relaxed condition $|L(\mathcal{A}(t^\star)) \triangle K| \leq k^{(1-\varepsilon)s}$ must still use $k^{\Omega(\varepsilon s)}$ bits.*

Most existing lower bounds in the identification/generation-in-the-limit literature involve some form of a diagonalization argument: The adversary constructs an enumeration of a problematic language by endlessly switching back-and-forth between enumerating different languages and causing the algorithm to fail at the objective at each switch in the process. These diagonalization arguments crucially rely on a special nesting structure present among the languages in the collection.

In contrast, our lower bound follows from a reduction to the *Index* problem from communication complexity: a standard route to space lower bounds for streaming algorithms. The Index problem defines a one-way communication game between Alice and Bob: Alice receives a length-n vector $x \in [k]^n$, Bob receives an index $i \in \{1, \ldots, n\}$, and Bob must output x_i after receiving a single message from Alice. A standard pigeonhole-principle bound implies that any protocol deterministically solving Index requires communication of $n \log_2 k$ bits from Alice to Bob[7]. To prove the streaming lower bound, we show that a space-efficient learner using only $\mathrm{poly}(s,k)$ bits of memory and achieving sufficiently small generation gap induces a one-way protocol for Index with communication strictly below $n \log_2 k$ bits, yielding a contradiction.

The lower bound also shows that the generation gap achieved by the algorithm matches the optimal rate up to constant factors in the exponent. Together, Theorem 3 and Theorem 4 establish that the classical separation between identification and generation in the limit persists under polynomial space bounds. This separation vanishes once the space budget increases from $\mathrm{poly}(s,k)$ to $\exp(s,k)$. Under $\exp(s,k)$ space, an algorithm can identify $\mathcal{C}_{s,k}$ in the limit. Space-bounded generation in the limit therefore exhibits a sharp phase transition: Sub-exponential space forces an exponential generation gap, while exponential space permits exact identification.

3. Upper Bounds

In this section, we derive our space-efficient algorithm $\mathcal{A}$, which, for all sufficiently large t, outputs a DFA $\mathcal{A}(t)$ that satisfies $L(\mathcal{A}(t)) \subseteq K$, where K denotes the unknown target language assumed to require a DFA of size at most s for recognition. Furthermore, $\mathcal{A}(t)$ satisfies that $|K \setminus L(\mathcal{A}(t))| \leq O(k^{2s-2})$. In particular, $L(\mathcal{A}(t))$ contains all strings in K that have length at least $2s-1$. Before describing our learning procedure, we first recall Angluin's algorithm for identification in the limit to illustrate the challenges for achieving space-efficiency.

Recalling Angluin's identification algorithm for finite collections. Let us arbitrarily enumerate the finitely many DFAs of size at most s as $A_1, A_2, \ldots, A_N$.[8] Here, $N \leq 2^s \cdot s^{ks}$ (2^s ways of selecting an accepting subset of at most s states, followed by specifying the transition on each symbol for every state). By assumption, there exists some index $i^\star$ for which $L(A_{i^\star}) = K$. As established by Angluin (1980), all finite collections are identifiable in the limit (without space constraints). The algorithm operates as follows: It first topologically sorts the DFAs $A_1, A_2, \ldots, A_N$ according to the

7. Related strong lower bounds, with quantitatively similar rates, also hold for randomized one-way communication protocols; see, for example, Bar-Yossef et al. (2002); Jayram et al. (2008).
8. Our enumeration will contain many isomorphic DFAs recognizing the same language. This redundancy is harmless. We only require that each regular language in $\mathcal{C}_{s,k}$ receives some representation in the enumeration.

strict partial order $\prec_1$ induced by *strict inclusion*:

$$A \prec_1 A' \iff L(A) \subset L(A'). \tag{1}$$

Namely, if $L(A)$ is a strict subset of $L(A')$, then A appears before A' in the topological sort.[9] Indeed, since the collection is finite, a topological sort corresponding to any partial order exists, and can be constructed by a standard depth-first search over a directed graph whose nodes correspond to the DFAs A_i. So, abusing notation slightly, suppose that the DFAs $A_1, \ldots, A_N$ are already ordered according to the topological sort induced by $\prec_1$. At time step t, after having seen input $w_1, \ldots, w_t$, the algorithm outputs the DFA $A_{i(t)}$, where $i(t)$ is the smallest index of a DFA that accepts each of $w_1, \ldots, w_t$[10]. Concretely, for every $j < i(t)$, it holds that $\{w_1, \ldots, w_t\} \nsubseteq L(A_j)$. Let $i^\star$ be the index of the DFA that recognizes K. Note that $\{w_1, \ldots, w_t\} \subseteq L(A_{i^\star})$ for every t. Therefore, $i(t) \leq i^\star$ for every t; since $i(t)$ is non-decreasing in t, $i(t)$ must eventually converge. Furthermore, the DFA $A_{i(t)}$ at which the algorithm converges must satisfy $L(A_{i^\star}) \subseteq L(A_{i(t)})$, since otherwise, there exists some $w \in L(A_{i^\star}) \setminus L(A_{i(t)})$, which is guaranteed to show up eventually in the input, causing $A_{i(t)}$ to be invalidated. Finally, since $i(t) \leq i^\star$, and $L(A_{i^\star}) \subseteq L(A_{i(t)})$, it must necessarily be the case that $L(A_{i^\star}) = L(A_{i(t)})$, by the definition of $\prec_1$.

An important feature of the algorithm above is that it keeps track of the entire history of inputs seen so far at every time step. This is not feasible in our framework of space efficiency where we insist on maintaining only $\mathrm{poly}(s, k)$ bits of working memory across time steps. As we shall see later, it is actually sufficient to remember only those input strings that have length at most $O(s)$ for exact identification; however, even this requires $\exp(s, k)$ memory.

There is also the separate issue of traversing the DFAs $A_1, \ldots, A_N$ in an order topologically sorted according to $\prec_1$ in a space-efficient manner. We address this latter issue first. For any automata C, D, checking $C \prec_1 D$ amounts to testing $L(C) \setminus L(D) = \emptyset$ and $L(D) \setminus L(C) \neq \emptyset$. Both conditions can be checked via the standard product constructions (see Appendix B) of DFAs (on at most s^2 states) that recognize $L(C) \setminus L(D)$ and $L(D) \setminus L(C)$, which can be done in $\mathrm{poly}(s, k)$ time (and hence also space). Thereafter, testing whether the product automaton recognizes an empty language amounts to checking if there exists a directed path beginning from the initial state and reaching an accepting state, which can again be done in $\mathrm{poly}(s, k)$ time (and hence space). Thus, one can verify $C \prec_1 D$ in $\mathrm{poly}(s, k)$ time. However, implementing the full topological sort of the exponentially many DFAs using only polynomial space requires further attention.

3.1. A space-efficient topological sort

Recall that the standard algorithm for topological sorting performs a depth-first search over a directed graph whose nodes consists of the DFAs in the collection $\mathcal{C}_{s,k}$. In the worst case, this approach requires maintaining a history of exponentially many already-enumerated DFAs, which clashes with our memory constraints. To remedy this, we consider a *Rank Iteration Strategy* for enumerating $\mathcal{C}_{s,k}$ in a space-efficient manner.

Concretely, given an arbitrary strict partial order $\prec$, for any DFA A in the collection $\mathcal{C}_{s,k} = \{A_1, \ldots, A_N\}$, define its *rank* with respect to $\prec$, denoted $\mathrm{rank}(A, \prec)$, as follows:

$$\mathrm{rank}(A, \prec) := \max\{\ell : \exists A_{i_1}, \ldots, A_{i_\ell} \in \mathcal{C}_{s,k} \text{ such that } A_{i_1} \prec \cdots \prec A_{i_\ell} \prec A\}. \tag{2}$$

9. It can be readily verified that the relation in (1) is a strict partial order: It is irreflexive, asymmetric and transitive.
10. Such an $i(t)$ always exists, since the target language K is recognized by some DFA in the collection.

Algorithm 1 Rank Iteration Strategy (recursive doubling for rank)

Input: DFA size s, strict partial order $\prec$, position $j \leq N$
Output: DFA A_j that is j^{th} in a topological sorting of all DFAs of size $\leq s$ under $\prec$

Procedure `rankIteration` $(s, \prec, j)$ **:**
  Initialize count $\leftarrow 0$
  for $r = 0, \ldots, N-1$ **do**
    Let $B_1, \ldots, B_N$ be any enumeration of all DFAs of size at most s
    for $i = 1, \ldots, N$ **do**
      if `computeRank`$(B_i, \prec) = r$ **then**
        count $\leftarrow$ count $+ 1$
        if count $= j$ **then return** B_i;

Input: DFA D, strict partial order $\prec$
Output: $\text{rank}(D, \prec)$ as in (2)

Procedure `computeRank` $(D, \prec)$ **:**
  for $\ell = N-1, N-2, \ldots, 1$ **do**
    Let $B_1, \ldots, B_N$ be any enumeration of all DFAs of size at most s
    for $i = 1, \ldots, N$ **do**
      if `path`$(B_i, D, \ell, \prec)$ **then return** ℓ;
  return 0

Input: DFAs C, D, path length $\ell \geq 1$, strict partial order $\prec$
Output: True iff there exists a $\prec$-chain of length ℓ from C to D

Procedure `path` $(C, D, \ell, \prec)$ **:**
  if $\ell = 1$ ***and*** $C \prec D$ **then return** True;
  Let $B_1, \ldots, B_N$ be any enumeration of all DFAs of size at most s; let $m \leftarrow \lceil \ell/2 \rceil$
  for $i = 1, \ldots, N$ **do**
    if `path`$(C, B_i, m, \prec)$ ***and*** `path`$(B_i, D, \ell - m, \prec)$ **then**
      return True
  return False

Here, $\text{rank}(A, \prec) = 0$ if there does not exist $A_{i_1} \in \mathcal{C}_{s,k}$ satisfying $A_{i_1} \prec A$. The rank of any $A \in \mathcal{C}_{s,k}$ can range from 0 to $N-1$. The following elementary observation relates rank to the partial order $\prec$:

Observation 5 (Relating rank to $\prec$) *If $A' \prec A$, then $\text{rank}(A, \prec) \geq \text{rank}(A', \prec) + 1$.*

This observation implies that, in order to traverse the DFAs topologically sorted according to $\prec$, it suffices to iterate through them in increasing order of their rank. This gives rise to the Rank Iteration Strategy encapsulated in Algorithm 1. The strategy processes DFAs in increasing order of their ranks. The rank of each DFA is computed space-efficiently using a technique inspired by the "middle-first search" approach used in Savitch's theorem (Savitch, 1970). Savitch's theorem gives a space-efficient algorithm for checking if two nodes are connected by a path of length ℓ in a directed graph, by recursively checking for the existence of a "middle node" that lies at distance $\ell/2$ from each of the nodes. The crucial point is that these recursive checks can reuse space. A similar idea applies for our purpose of computing rank, where a path of length ℓ between two DFAs C and D corresponds to a chain $C \prec B_1 \prec \cdots \prec B_{\ell-1} \prec D$ in the partial order. In this case, if the relation $\prec$ can be checked using polynomial space, then the entire Rank Iteration Strategy uses only polynomial space. This leads to the following proposition, whose proof is given in Appendix C.

Proposition 6 (Rank Iteration Space Complexity) `rankIteration`$(s, \prec, j)$ *given in Algorithm 1 requires* $\text{poly}(s, k)$ *bits of memory, provided one can check "$C \prec D$?" using* $\text{poly}(s, k)$ *space for any two DFAs C and D with at most s states.*

3.2. A natural space-efficient modification and the main challenge

We now tackle the issue of storing input history. The subroutine `SpaceEfficientTraversal`($\prec$) below modifies Angluin's algorithm to avoid storing the full input history $\{w_1, \ldots, w_t\}$ up to time t. However, this modification will necessitate changing the partial order $\prec_1$ considered by Angluin's algorithm, as the proceeding analysis reveals.

> `SpaceEfficientTraversal`($\prec$): Let $A_1, \ldots, A_N$ denote all the DFAs of size at most s topologically sorted according to $\prec$. The previous analysis shows that we can traverse the DFAs in this order space-efficiently. Let $i(0) = 1$. At any time step $t \geq 1$, the algorithm first initializes $i(t) = i(t-1)$. After receiving the streamed input string w_t, the algorithm checks if $w_t \in L(A_{i(t)})$, which only requires tracing w_t through the DFA $A_{i(t)}$, and hence requires constant memory. Here, $A_{i(t)}$ is retrieved by invoking `rankIteration`$(s, \prec, i(t))$. If this check does not pass, the algorithm increments $i(t) \leftarrow i(t) + 1$ and outputs the DFA $A_{i(t)}$. Note that the outputted $A_{i(t)}$ might not accept w_t. Indeed, requiring the algorithm to retest membership of w_t in $L(A_{i(t)})$ could necessitate remembering a prohibitively long string w_t.

We can instantiate `SpaceEfficientTraversal` with the partial order $\prec_1$, and by the preceding discussion, this procedure uses only $\mathrm{poly}(s, k)$ memory. The remaining question is whether it identifies K in the limit; i.e., whether it eventually outputs $A_{i(t)}$ with $L(A_{i(t)}) = K$ for all $t \geq t^\star$.

Recall that there exists an index $i^\star$ such that $L(A_{i^\star}) = K$. Now, if the algorithm ever arrives at index $i^\star$, it never proceeds beyond it, since every $w_t \in K$. Thus, in the limit, the algorithm converges to some index $i(t) \leq i^\star$. Then, it must be the case that $|K \setminus L(A_{i(t)})| < \infty$; otherwise, we are guaranteed to see some string $w \in K \setminus L(A_{i(t)})$ in the future, contradicting convergence at $i(t)$.

If $i(t) = i^\star$, then we identify K. Otherwise, $i(t) < i^\star$. We already argued above that $|K \setminus L(A_{i(t)})| < \infty$. If $|K \setminus L(A_{i(t)})| = 0$, then $K \subseteq L(A_{i(t)})$. In that case, we are guaranteed that $K = L(A_{i(t)})$. Indeed, $K \subset L(A_{i(t)})$ would contradict the definition of $\prec_1$. So, we achieve the desired objective in this case as well.

The only case that remains is when $i(t) < i^\star$ and $|K \setminus L(A_{i(t)})| < \infty \neq 0$. Unfortunately, in this case, it is possible that $L(A_{i(t)}) \setminus K$ is non-empty as well. In fact, $L(A_{i(t)})$ could even contain infinitely many strings outside K. So, in this case, not only do we fail to achieve the desired objective of identification, but the language that we converge to may contain infinitely many hallucinations.

This unwanted case is a direct consequence of the memory constraint. Indeed, when $K \setminus L(A_{i(t)})$ is nonempty, and if we were to store the entire history of inputs at every time step (as Angluin's algorithm does), the history would eventually contain some string in $K \setminus L(A_{i(t)})$, causing us to proceed beyond $A_{i(t)}$. However, consider the possibility where $K \setminus L(A_{i(t)})$ has very few strings which all appeared in the distant past without showing up again. Since we do not store past inputs in memory, and only make our decisions based on the most recent string, we never see any evidence again that would cause us to move beyond $A_{i(t)}$.

As a final remark, note that `SpaceEfficientTraversal`($\prec_1$) *does* succeed with exact identification under the additional assumption that every string in K appears infinitely often in the input stream. In this setting, any missing string $w \in K \setminus L(A_{i(t)})$ would eventually reappear and invalidate the currently wrong hypothesis.

3.3. A slightly more robust partial order

The analysis above isolates the main limitation of the partial order $\prec_1$: If there exist two DFAs A and A' such that $|L(A) \setminus L(A')| < \infty$ but $|L(A') \setminus L(A)|$ is huge, then topologically sorting according to $\prec_1$ allows A' to be placed before A. Consequently, if $L(A)$ is the target language, we might converge at $L(A')$, which contains a large number of hallucinations outside $L(A)$. Our final solution arises as a direct remedy to this issue.

Concretely, consider an ordering of the DFAs which enforces the following property: For any two DFAs A and A', if $|L(A) \setminus L(A')| < \infty$ and $|L(A) \setminus L(A')| < |L(A') \setminus L(A)|$, then A appears before A'. Namely, we would like to consider the relation

$$A \prec_2 A' \iff |L(A) \setminus L(A')| < \infty \text{ and } |L(A) \setminus L(A')| < |L(A') \setminus L(A)|. \tag{3}$$

If this relation were a strict partial order, we could topologically sort the DFAs according to $\prec_2$ and run the space-efficient algorithm described above on this ordering. As it turns out, the relation is indeed a strict partial order; the elementary proof appears in Appendix C.

Proposition 7 ($\prec_2$ Strict Partial Order) *The relation $\prec_2$ defined in (3) is a strict partial order.*

Next, we observe that checking $A \prec_2 A'$ can also be done with $\mathrm{poly}(s, k)$ space. For this, we once again construct the product automata that recognize $L(A) \setminus L(A')$ and $L(A') \setminus L(A)$, respectively (each of these have at most s^2 states). Then, we use the fact that a regular language is infinite if and only if an automaton recognizing it has a cycle that can be reached from the initial state, and can then go on to reach an accepting state. We can check this using standard depth-first search on the directed graph underlying the automaton. For our purposes, this requires $\mathrm{poly}(s, k)$ time, and hence space. Finally, if both the languages $L(A) \setminus L(A')$ and $L(A') \setminus L(A)$ are determined to be finite, we require comparing their sizes. In this case, we observe that both languages must contain strings of length at most s^2. Otherwise, if either contained a string of length larger than s^2, this string would induce a cycle while traversing its product automaton, which could be pumped infinitely. So, the size of both languages can be computed exactly by iterating over all strings of length at most s^2, and checking membership in the automaton. This step can also be executed in $\mathrm{poly}(s, k)$ space.

Using all the ingredients established so far, we can then show that `SpaceEfficientTraversal`, invoked with the partial order $\prec_2$, converges to a DFA that has a finite symmetric difference with the target language K — this notion has been referred to as *hyper-equivalence* in the automata literature (Badr et al., 2009; Gawrychowski et al., 2011; Maletti and Quernheim, 2011).

Proposition 8 ($\prec_2$ Converges to Finite $\triangle$) `SpaceEfficientTraversal`$(\prec_2)$ *converges to an index $i(t)$ satisfying $|L(A_{i(t)}) \triangle K| < \infty$, while using only $\mathrm{poly}(s, k)$ memory. Furthermore, if every string in K shows up infinitely often in the input stream, then $L(A_{i(t)}) = K$ in the limit.*

Proof Since it is guaranteed that $K = L(A_{i^\star})$ for some index $i^\star$, the algorithm never proceeds beyond $i^\star$, and hence, $i(t) \leq i^\star$ in the limit. Furthermore, $|K \setminus L(A_{i(t)})| < \infty$, since otherwise, the algorithm eventually sees a string that causes it to move past $i(t)$. So, either $i(t) = i^\star$, or $i(t) < i^\star$. In the latter case, it must hold that $|L(A_{i(t)}) \setminus K| \leq |K \setminus L(A_{i(t)})| < \infty$, since we topologically sorted according to $\prec_2$. Either way, $|L(A_{i(t)}) \triangle K| < \infty$ holds. Since we argued above that checking $\prec_2$ uses $\mathrm{poly}(s, k)$ space, by Proposition 6, the entire algorithm uses only $\mathrm{poly}(s, k)$ space.

Finally, if every string in K shows up in the input infinitely often, then the algorithm converges to an $i(t)$ satisfying $L(A_{i(t)}) \supseteq L(A_{i^\star})$. Otherwise, a string in $L(A_{i^\star}) \setminus L(A_{i(t)})$ is guaranteed to show

up in the input, causing the algorithm to move beyond $i(t)$. Now, suppose that $L(A_{i(t)}) \supset L(A_{i^\star})$. Then, we have that $|L(A_{i^\star}) \setminus L(A_{i(t)})| = 0 < |L(A_{i(t)}) \setminus L(A_{i^\star})|$, which contradicts sorting according to $\prec_2$ since $i(t) < i^\star$. Thus, $L(A_{i(t)}) = L(A_{i^\star}) = K$. ■

3.4. The final algorithm

We now have all the ingredients necessary to state our final algorithm. By Proposition 8, we know that $\texttt{SpaceEfficientTraversal}(\prec_2)$ converges to a DFA $A_{i(t)}$ that satisfies $|L(A_{i(t)}) \triangle K| < \infty$. If we can now output a DFA that accepts a large subset of $L(A_{i(t)})$ but avoids the strings in the finite symmetric difference $L(A_{i(t)}) \triangle K$, our objective of language generation with finite gap would be accomplished.

Towards this, observe that $L(A_{i(t)}) \triangle K$ is yet another regular language. Moreover, the product DFA C recognizing $L(A_{i(t)}) \triangle K$ has at most s^2 states. Hence, given that this language is finite, it must not include any string of length $\geq s^2$; otherwise, this string would visit some state in the DFA C twice before reaching an accepting state, yielding a cycle, which can be pumped arbitrarily often to generate infinitely many accepted strings. In fact, we can prove a stronger bound, using a prior result by Gawrychowski et al. (2011), which shows that $L(A_{i(t)}) \triangle K$ can only contain strings of length at most $2s - 2$. The proof of this result appears in Appendix C.

Lemma 9 (Symmetric Difference Contains Length-$O(s)$ Strings) *Let $A = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ denote DFAs having at most s states over a common alphabet Σ, such that $|L(A) \triangle L(B)| < \infty$. Then, for any string $w \in \Sigma^*$, if $|w| \geq 2s - 1$, then $w \notin L(A) \triangle L(B)$.*

Thus, if we output a DFA A' that only accepts strings of length at least $2s - 1$ in $L(A_{i(t)})$, we ensure that $L(A') \subseteq L(A_{i(t)}) \cap K \subseteq K$ as required. This leads to the following theorem, whose precise proof is given in Appendix C.

Theorem 10 (Space-Efficient Language Generation) *Let K denote the unknown target language recognized by a DFA of size at most s. There exists an algorithm $\mathcal{A}$ which, for all sufficiently large t, outputs a DFA $A' = \mathcal{A}(t)$ that satisfies $L(A') \subseteq K$, and uses only $\mathrm{poly}(s, k)$ space. Furthermore, the algorithm misses at most $|K \setminus L(A')| \leq O(k^{2s-2})$ target strings.*

Let us also remark here that by the conclusion of Lemma 9, our learning algorithm $\mathcal{A}$ is guaranteed to output an automaton A' such that every string in $K \setminus L(A')$ has length at most $2s - 2$. Thus, if the algorithm additionally keeps track of all the inputs of length at most $2s - 2$ (which requires $\exp(s, k)$ space), $\mathcal{A}$ can output a DFA that exactly recognizes K. In other words, with $\exp(s, k)$ memory, we can identify $\mathcal{C}_{s,k}$ in the limit.

Finally, observe that our algorithm above ensures no hallucinations, but misses out on a finite amount of breadth (namely $O(k^{2s-2})$ strings) in the limit. Instead, we could have the algorithm output a different product DFA A' (again on $\leq 2s^2$ states), which accepts $L(A_{i(t)}) \cup \{w \in \Sigma^* : |w| \leq 2s - 2\}$. This would ensure full breadth (i.e., $L(A') \supseteq K$), but a finite amount of hallucinations (again $O(k^{2s-2})$ strings). We thus have to necessarily incur a misspecification cost on $O(k^{2s-2})$ strings, either in hallucinations, or breadth. Our next result shows that this compromise is indeed unavoidable.

4. Lower Bounds

In the previous section, we showed that polynomial space suffices to generate in the limit from $\mathcal{C}_{s,k}$ with a generation gap of $O(k^{2s-2})$. A natural question is whether this loss is merely an artifact of our algorithm, or whether it is fundamentally unavoidable under space constraints. In this section, we prove that the latter is actually the case.

Our result establishes that any algorithm that operates with sub-exponential memory must incur an exponential generation gap. Moreover, the same lower bound holds under the relaxed symmetric-difference objective from Definition 2, where hallucinations and missed strings are counted together through $|L(A(t))\triangle K|$. Thus, hallucinations cannot be traded for substantially smaller missed breadth: Any sub-exponential-memory learner still incurs an exponential total error. This demonstrates a sharp transition between what is achievable with polynomial space versus exponential space: Exponential memory permits full identification, whereas any polynomial-space learner is fundamentally forced to miss an exponential number of strings.

Our proof departs from the diagonalization-based arguments commonly used in the identification-in-the-limit literature. Instead, we adopt a communication-complexity perspective and reduce from the INDEX$_n$ problem. At a high level, we show that a space-efficient learner with a sufficiently small generation gap could be used to derive an efficient communication protocol for the INDEX$_n$ problem, which contradicts known lower bounds. We begin by recalling the relevant communication problem and its standard lower bound, and then we present the reduction that translates a small generation gap into an efficient Index protocol.

Lemma 11 (Communication Lower Bound for INDEX$_n$**, Kushilevitz (1997))** *Let* INDEX$_n$ *be the one-way communication problem where Alice is given a vector $x \in [k]^n$, Bob is given an index $i \in [n]$, and Bob must output x_i after a single message from Alice. Any deterministic protocol for* INDEX$_n$ *requires at least $n \log_2 k$ bits of communication.*

The proof of Lemma 11 is a standard pigeonhole argument; for completeness we include it in Appendix D. The next lemma connects INDEX$_n$ to generation in the limit over $\mathcal{C}s, k$, yielding a generation-gap lower bound in terms of the learner's space.

Lemma 12 *Any algorithm $\mathcal{A}$ that uses $m(s)$ bits of space and generates in the limit from $\mathcal{C}_{s,k}$ must incur a generation gap of $\Delta_{\mathcal{A}}(s,k) \geq k^{s-2\log_k((m(s)+1)/\log_2 k)-3}$.*

Proof Fix s and let $m(s)$ be the number of bits that algorithm $\mathcal{A}$ uses. As we discussed above our proof goes through the INDEX$_n$ problem. We will set n, i.e. the input size of INDEX$_n$, later in the proof as a function of $m(s)$ and k.

Alphabet and building blocks. Let $\Sigma \coloneqq [k]$ be an alphabet of size k. Throughout the proof we use $[k] = \{0, 1, \ldots, k-1\}$. We define U to be the set of all strings that start with a 1 and continue with some finite number of symbols from Σ. We use the first, say p, symbols after 1 to encode a pair (i, q) for all $i \in [n]$ and all $q \in \Sigma$, and we partition the universe U into subsets based on these p symbols. Then, every string will continue with some free symbols, say ℓ of them, which means that each of the subsets corresponding to a pair (i, q) has k^ℓ strings. Since there are kn different pairs (i, q) and we are encoding in base-k, we need $p \coloneqq 1 + \log_k n$ symbols to encode them. Formally, for each $j \in \{0, 1, \ldots, kn-1\}$, let $\text{enc}_k(j) \in \Sigma^p$ denote the base-k encoding of j with length p (allowing leading zeros). We define the languages $L_0, \ldots, L_{kn-1}$, which partition U, as

$$L_j \coloneqq \{1 \circ \text{enc}_k(j) \circ u \mid u \in \Sigma^\ell\}.$$

Every language above has size k^ℓ, as they have exactly one string for every member of Σ^ℓ. We create a bijective mapping, map : $[n] \times [k] \to [kn]$, that maps every combination of a dimension $i \in [n]$ and symbol $q \in [k]$ to a unique language in the above partition. Any such mapping can work for our reduction, for simplicity let $\text{map}(i, q) \coloneqq i \cdot k + q$.

Furthermore, for every $i \in [n]$ and every $q \in [k]$ we define the language $\text{Base}(i, q)$ to contain all above partitions of U except for $L_{\text{map}(i,q)}$. For technical reasons that will become clear later, we also make $\text{Base}(i, q)$ include an infinite component, namely all strings that start with a 0. Formally,

$$\text{Base}(i, q) \coloneqq \{0 \circ u \mid u \in \Sigma^*\} \cup \{1 \circ u \mid u \in \Sigma^{p+\ell}\} \setminus L_{\text{map}(i,q)}.$$

Notice that all partitions of U, i.e. $L_0, \ldots, L_{kn-1}$, can be recognized by a DFA with at most $p + \ell + 2$ states, which simply recognizes the encoding bits and then counts ℓ more characters. Also, for each language $\text{Base}(i, q)$, we can construct a DFA that recognizes it as follows:

- If the first symbol is 0, transition to an accepting state that self-loops on all symbols in Σ (this branch accepts $\{0 \circ u \mid u \in \Sigma^*\}$).
- If the first symbol is 1, the DFA needs to accept exactly $U \setminus L_{\text{map}(i,q)}$. Equivalently, among strings of length $1 + p + \ell$ that start with 1, it rejects exactly those whose next p symbols encode the number $\text{map}(i, q)$, i.e. $\text{enc}_k(\text{map}(i, q))$. This can easily be done by 2 chains of length p that track whether or not the sequence encodes the target number. After this part, the chain that recognized this sequence leads to a rejecting state, while the chain that did not recognize the sequence reads ℓ more symbols and accepts.

In Appendix D, we give a figure (Figure 1) that shows the above construction. Such a DFA uses at most $2p + \ell + 5$ states. We now set $s \coloneqq 2p + \ell + 5$ so that all above languages, i.e. all $L_{\text{map}(i,q)}$ and $\text{Base}(i, q)$ for $i \in [n]$ and $q \in \Sigma$, are recognized by DFAs of size at most s.

Reduction from INDEX_n**.** We now build a deterministic one-way protocol for INDEX_n using algorithm $\mathcal{A}$. As a reminder, Alice receives $x = (x_0, x_1, \ldots, x_{n-1}) \in \Sigma^n$, while Bob receives $i \in [n]$ and must output x_i. Alice simulates $\mathcal{A}$ on a finite input stream consisting of all the strings in the languages $L_{\text{map}(j,q)}$ for every coordinate $j \in [n]$ and every symbol $q \neq x_j$. Conceptually, the only partitions of U that are **not** fed to the algorithm are exactly the combinations (j, x_j) of Alice's input. Formally, she feeds $\mathcal{A}$ the set

$$S_A = \bigcup_{j \in [n]} \bigcup_{q \in \Sigma \setminus \{x_j\}} L_{\text{map}(j,q)}.$$

Let σ be the internal memory state of $\mathcal{A}$ after reading this prefix. Alice sends the state σ to Bob, which by definition is at most $m(s)$ bits. Bob resumes the execution of $\mathcal{A}$ from state σ and continues by feeding it all the strings in the languages $L_{\text{map}(j,q)}$ for $j \neq i$, i.e., the set

$$S_B = \bigcup_{\substack{j \in [n], \\ j \neq i}} \bigcup_{q \in \Sigma} L_{\text{map}(j,q)}.$$

Notice that at this point, the learner $\mathcal{A}$ has seen all partitions of U except for the one corresponding to (i, x_i). After feeding these strings, Bob continues running the algorithm $\mathcal{A}$ and gives it strings from $\{0 \circ u \mid u \in \Sigma^*\}$, via some surjective enumeration. This part is simply done so that the algorithm $\mathcal{A}$ can keep running until it converges to an answer.

Consider now the union of the strings presented to $\mathcal{A}$, i.e. $S_A \cup S_B \cup \{0 \circ u \mid u \in \Sigma^*\}$. For all $j \neq i$, Bob has given $\mathcal{A}$ all the strings in the languages $L_{\text{map}(j,q)}$. Furthermore, for the index $j = i$, Alice has given $\mathcal{A}$ all the strings in the languages $L_{\text{map}(i,q)}$ for all $q \neq x_i$. Thus, the algorithm has seen all languages $L_{\text{map}(j,q)}$ except for $L_{\text{map}(i,x_i)}$. Formally, the stream is a surjective enumeration of

$$S_A \cup S_B \cup \{0 \circ u \mid u \in \Sigma^*\} = \{0 \circ u \mid u \in \Sigma^*\} \cup \{1 \circ u \mid u \in \Sigma^{p+\ell}\} \setminus L_{\text{map}(i,x_i)} = \text{Base}(i, x_i).$$

Decoding x_i from algorithm $\mathcal{A}$. By the definition of generation in the limit, with gap $\Delta_{\mathcal{A}}(s,k)$, applied to $L^* = \text{Base}(i, x_i)$, the limiting output of $\mathcal{A}$, let it be H, satisfies

$$L(H) \subseteq \text{Base}(i, x_i) \qquad \text{and} \qquad \big|\text{Base}(i, x_i) \setminus L(H)\big| \leq \Delta_{\mathcal{A}}(s,k).$$

Let's assume at this point that the generation gap of the learner $\mathcal{A}$ is $\Delta_{\mathcal{A}}(s,k) < k^\ell$. Then, for any $q \neq x_i$, if $L(H) \cap L_{\text{map}(i,q)} = \emptyset$, then $\mathcal{A}$ would miss all k^ℓ strings in $L_{\text{map}(i,q)} \subseteq \text{Base}(i, x_i)$, which would mean that its generation gap is at least $|\text{Base}(i, x_i) \setminus L(H)| \geq k^\ell$, contradicting our assumption. Therefore, for every $q \neq x_i$ we must have $L(H) \cap L_{\text{map}(i,q)} \neq \emptyset$. On the other hand, $L_{\text{map}(i,x_i)}$ is disjoint from $\text{Base}(i, x_i)$, and since $L(H) \subseteq \text{Base}(i, x_i)$, it holds that $L(H) \cap L_{\text{map}(i,x_i)} = \emptyset$.

As a result, x_i is the unique symbol $q \in \Sigma$ such that $L(H) \cap L_{\text{map}(i,q)} = \emptyset$. Bob can, thus, determine x_i by testing the emptiness of the intersection of $L(H)$ with the languages $L_{\text{map}(i,q)}$ (e.g. by constructing the product DFAs and checking if there is any path from the starting state to an accepting state). Notice that Bob does not necessarily need to know when the learner $\mathcal{A}$ has converged. The reduction from INDEX_n is purely information-theoretic and therefore it suffices that there exists a t^* after which Bob could test the above conditions and recover x_i. See Appendix D for further discussion.

We have, therefore, constructed a deterministic one-way protocol for INDEX_n in which Alice's message σ has at most $m(s)$ bits. Choosing $m(s) = n \log_2 k - 1$ leads to a contradiction of the INDEX_n lower bound (Lemma 11), which means that our assumption of $\Delta_{\mathcal{A}}(s,k) < k^\ell$ is false in the case where $m(s) = n \log_2 k - 1$. Recall that we used $s = 2p + \ell + 5$ and $p = 1 + \log_k n$, which yields $\ell = s - 2p - 5$ and $p = 1 + \log_k((m(s)+1)/\log_2 k)$. The proof of the lemma is concluded by combining the former identities $\Delta_{\mathcal{A}}(s,k) \geq k^\ell = k^{s - 2\log_k((m(s)+1)/\log_2 k) - 3}$. ■

Finally, the main result of this section, Theorem 4, follows from Lemma 12, as in the case where $\Delta_{\mathcal{A}}(s,k) \leq k^{(1-\epsilon)s}$, we have that $2\log_k((m(s)+1)/\log_2 k) \geq \varepsilon s - 3$, i.e. $m(s) = k^{\varepsilon s/2 - 3 + \log_k(\log_2 k)} = k^{\Omega(\varepsilon s)}$.

5. Conclusion

This paper develops a space-efficient model of generation in the limit for the DFA family $\mathcal{C}_{s,k}$ and proves a tight memory–breadth tradeoff: $\text{poly}(s,k)$ space guarantees hallucination-free generation up to an exponential gap, while achieving substantially smaller gaps (and, in particular, exact identification) forces exponential memory.

Several directions remain open, including extending the framework to non-uniform learners, and developing analogous tradeoffs for other resources such as time and sample complexities.

Acknowledgments

This work was partially funded by the Swiss National Science Foundation, grant number 212111. Chirag Pabbaraju is supported by Gregory Valiant's and Moses Charikar's Simons Investigator Awards, and a Google PhD Fellowship. Miltiadis Stouras and Ola Svensson are supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number MB22.00054.

References

Charu C Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

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

Andrew Badr, Viliam Geffert, and Ian Shipman. Hyper-minimizing minimized deterministic finite state automata. *RAIRO-Theoretical Informatics and Applications*, 43(1):69–94, 2009.

Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 93–102. IEEE Computer Society, 2002. doi: 10.1109/CCC.2002.1004344.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Moses Charikar and Chirag Pabbaraju. Exploring facets of language generation in the limit. *arXiv preprint arXiv:2411.15364*, 2024.

Moses Charikar and Chirag Pabbaraju. Pareto-optimal non-uniform language generation. *arXiv preprint arXiv:2510.02795*, 2025.

Moses Charikar, Chirag Pabbaraju, and Ambuj Tewari. A characterization of list language identification in the limit. *arXiv preprint arXiv:2511.04103*, 2025.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Michelle M Chouinard and Eve V Clark. Adult reformulations of child errors as negative evidence. *Journal of child language*, 30(3):637–669, 2003.

B. Jack Copeland. The Church-Turing Thesis. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2026 edition, 2026.

Javier Esparza and Michael Blondin. *Automata theory: An algorithmic approach*. MIT Press, 2023.

Pawel Gawrychowski, Artur Jez, and Andreas Maletti. On minimising automata with errors. In *International Symposium on Mathematical Foundations of Computer Science*, pages 327–338. Springer, 2011.

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

David Gries. Describing an algorithm by hopcroft. *Acta Informatica*, 2(2):97–109, 1973.

Steve Hanneke, Amin Karbasi, Anay Mehrotra, and Grigoris Velegkas. On union-closedness of language generation. *arXiv preprint arXiv:2506.18642*, 2025.

John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory Comput.*, 4(1):129–135, 2008. doi: 10.4086/TOC.2008.V004A006.

Alkis Kalavasis, Anay Mehrotra, and Grigoris Velegkas. Characterizations of language generation with breadth. *arXiv preprint arXiv:2412.18530*, page 3, 2024.

Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

Jon Kleinberg and Sendhil Mullainathan. Language generation in the limit. *Advances in Neural Information Processing Systems*, 37:66058–66079, 2024.

Jon Kleinberg and Fan Wei. Density measures for language generation. *arXiv preprint arXiv:2504.14370*, 2025.

Jon Kleinberg, Anay Mehrotra, Amin Saberi, and Grigoris Velegkas. On language generation in the limit with bounded memory, 2026.

Eyal Kushilevitz. Communication complexity. In *Advances in Computers*, volume 44, pages 331–360. Elsevier, 1997.

Jiaxun Li, Vinod Raman, and Ambuj Tewari. Generation through the lens of learning theory. *arXiv preprint arXiv:2410.13714*, 2024.

Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. Dissociating language and thought in large language models. *Trends in cognitive sciences*, 28(6):517–540, 2024.

Andreas Maletti and Daniel Quernheim. Optimal hyper-minimization. *International Journal of Foundations of Computer Science*, 22(08):1877–1891, 2011.

Gary F Marcus. Negative evidence in language acquisition. *Cognition*, 46(1):53–85, 1993.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

George A Miller and Noam Chomsky. Finitary models of language users. 1963.

Hristo Papazov and Nicolas Flammarion. Learning algorithms in the limit. *arXiv preprint arXiv:2506.15543*, 2025.

Charlotte Peale, Vinod Raman, and Omer Reingold. Representative language generation. *arXiv preprint arXiv:2505.21819*, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

Anej Svete and Ryan Cotterell. Recurrent neural language models as probabilistic finite-state automata. *arXiv preprint arXiv:2310.05161*, 2023.

Anej Svete and Ryan Cotterell. Transformers can represent n-gram language models. *arXiv preprint arXiv:2404.14994*, 2024.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Organization of the Appendix

- ➢ Appendix A motivates the focus on finite regular hypothesis spaces by developing the connection to communicative languages induced by interaction among space-bounded agents.
- ➢ Appendix B provides standard results on DFAs.
- ➢ Appendix C contains deferred proofs from Section 3.
- ➢ Appendix D contains deferred proofs from Section 4.

Appendix A. Resource-Bound Communication

Here, we expand on our point from Section 1 that space-efficient learning in the limit admits a meaningful formulation only over finite hypothesis spaces of regular languages. We also argue that this language-acquisition regime captures the most relevant and informative setting.

Space-Efficient Generation. Under our proposed framework, the learner $\mathcal{A}$ receives an adversarial enumeration $w_1, w_2, w_3, \ldots$ of a target language K through an online streaming interface. Each string $w_t \in K$ arrives one symbol at a time, with a designated delimiter separating consecutive strings. After each read symbol, the learning algorithm can perform computation and storage within a specified memory budget, dependent on the complexity of the target language. Furthermore, after processing each w_t, the learner outputs a hypothesis representation. The generation objective requires that, after some finite time, every subsequent output of $\mathcal{A}$ must represent a language $L \subseteq K$.

As a consequence of the Church-Turing Thesis (Copeland, 2026), we can model space-bounded learners as *streaming Turing machines* (STM) (Aggarwal, 2007) with bounded-size work tapes. An STM features a one-way read-only input and output tapes and several bidirectional work tapes. Clearly, a space-bounded STM admits only finitely many reachable configurations, so any decision procedure implemented by such a model necessarily recognizes a regular language. Now, the space-bounded learner $\mathcal{A}$ who searches for the target K within a hypothesis collection $\mathcal{C}_\mathcal{A}$ should have the ability to verify whether a given input stream $w \in K$ belongs to the current hypothesis $L \in \mathcal{C}_\mathcal{A}$. Hence, since the finite-memory STM $\mathcal{A}$ decides membership queries for every hypothesis language, $\mathcal{C}_\mathcal{A}$ must contain only regular languages. Moreover, since $\mathcal{A}$ can occupy finitely many internal configurations, $\mathcal{A}$ can only decide membership for finitely many languages, leading to $|C_\mathcal{A}| < \infty$. Thus, space-efficient generation in the limit naturally confines analysis to finite collections of regular hypotheses.

Although the restriction to finite regular hypothesis spaces may appear limiting, this language-acquisition regime provides the central case for understanding realistic learners. In particular, we demonstrate that the collection $\mathcal{K}$ of *communicative languages*, which arise from the interaction of space-bounded agents such as humans (Miller and Chomsky, 1963), forms a strict subset of regular languages.

Regularity of Communicative Languages. We now show from the dual perspectives of space-bounded language users and learners that the set of communicative languages $\mathcal{K}$ can only contain regular languages.

First, we posit that a competent user U of a language $K \in \mathcal{K}$ has the ability to decide whether $w \in K$ for any input stream $w \in \Sigma^*$. Together with the finite memory assumption for U, this assertion implies the regularity of K. A potential objection arises from Chomsky's classic observation that the possibility of unbounded center embedding makes English at least context-free (Chomsky,

1956). However, in practice, an English speaker will need access to a pen and paper to reliably judge the grammaticality of deeply center-embedded sentences such as "The mouse that the cat that the dog chased bit ran." Indeed, for any input stream $w \in \Sigma^*$, humans respond in one of three ways: (i) accept w as grammatical, (ii) reject w as ungrammatical, or (iii) determine that processing w requires additional resources and reject w as an unnatural conversational sentence. In this sense, the set of grammatically well-formed English strings $\bar{K}$ might sit high in the Chomsky hierarchy, but the communicative English sub-language $K \subset \bar{K}$ remains regular.

Second, we posit that any communicative language $K \in \mathcal{K}$ admits learning by a space-bounded learner $\mathcal{A}$. Indeed, human infants acquire grammar without external memory aids, just as large transformers successfully train with finite context windows. Hence, as argued above, the hypothesis space $\mathcal{C}_\mathcal{A}$ contains finitely many regular languages – a conclusion corroborated by the search spaces of LLMs.[11] Therefore, due to the realizability assumption $K \in \mathcal{C}_\mathcal{A}$, K becomes regular.

Importantly, our arguments for regularity do not imply that communicative languages have finite size or strings of bounded length. Throughout, we model strings as concatenations of grammatically well-formed sentences similar to token sequences produced by LLMs, which may consist of many sentences terminated by an end-of-sequence marker. Under this interpretation, a communicative language may contain strings of unbounded length, provided that those strings remain locally parseable by a finite-memory process. As an analogy, a proficient human reader can judge an entire book as grammatically correct while using only bounded working memory. Finally, we emphasize that we treat languages purely as sets of grammatically well-formed strings and deliberately ignore semantic verification, which may require substantially greater resources.

Appendix B. Standard DFA Results

In this section, we briefly state some foundational and some more recent results about DFAs. For a more detailed presentation, we refer the reader to the standard books on Automata Theory (Sipser, 1996; Hopcroft et al., 2001; Esparza and Blondin, 2023).

For a DFA $A = (Q, \Sigma, \delta, q_0, F)$, we denote by $L(A) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$ the regular language accepted by A and let

$$L_A^-(q) = \{w \in \Sigma^* : \delta(q_0, w) = q\} \quad \text{and} \quad L_A^+(q) = \{w \ \in \Sigma^* : \delta(q, w) \in F\}$$

stand for the left (or prefix) and right (or residual) languages of state $q \in Q$. Throughout the paper, we use the term *automaton* to refer to a DFA.

Regular languages possess an interesting algebraic property described by the Myhill-Nerode theorem (Esparza and Blondin, 2023, Chapter 2.4). For a subset $L \subseteq \Sigma^*$, we can define the Nerode equivalence $\equiv_L$ between strings in Σ^* such that $u \equiv_L v \iff \{w \in \Sigma^* : u \cdot w \in L\} = \{w \in \Sigma^* : u \cdot w \in L\}$. Now, the theorem states that L is a regular language if and only if $\equiv_L$ splits Σ^* into a finite number of equivalence classes, indicated as $|\equiv_L|$. Moreover, when $|\equiv_L| < \infty$, there exists a unique minimum-state DFA for L with the following description: $q_0 = [\varepsilon]_{\equiv_L}$, $Q = \{[w]_{\equiv_L} : w \in \Sigma^*\}$, $F = \{[w]_{\equiv_L} : w \in L\}$, $\delta([w]\equiv_L, \sigma) = [w \cdot \sigma]_{\equiv_L}$, $\forall w \in \Sigma^*, \sigma \in \Sigma$.

One can also consider equivalence of DFAs in terms of accepting languages: $A \equiv B \iff L(A) = L(B)$. This equivalence extends to automata states. Indeed, for a DFA $A = (Q, \Sigma, \delta, q_0, F)$,

11. Transformers, RNNs, and related NTP architectures behave like large probabilistic deterministic finite-state automata (Svete and Cotterell, 2023, 2024; Merrill et al., 2024) and therefore generate from a regular support.

we define states $p, q \in Q$ as equivalent ($p \equiv q$) if $L_A^+(p) = L_A^+(q)$. Given the automaton A, Hopcroft's algorithm (Hopcroft, 1971; Gries, 1973) allows us to find the minimal automaton for $L(A)$ in $O(|\Sigma||Q| \log |Q|)$ time by cleverly merging equivalent states. As a consequence of Hopcroft's algorithm, if $p \not\equiv q$, then there exists a distinguishing string $w \in L_A^+(p) \triangle L_A^+(q)$ of length at most $|Q| - 1$.

A relaxed variant of equivalence, referred to as hyper-equivalence (Badr et al., 2009; Gawrychowski et al., 2011; Maletti and Quernheim, 2011), underpins our results. We define two automata A and B as hyper-equivalent, denoted $A \sim B$, if $L(A)$ and $L(B)$ disagree on finitely many strings: $|L(A) \triangle L(B)| < \infty$. Clearly, $\sim$ is an equivalence relation, which we can again extend to automata states by writing $p \sim q$ if and only if $|L_A^+(p) \triangle L_A^+(q)| < \infty$.

Now, we describe the standard product DFA constructions. In the notation of Definition 1, for two DFAs $C = (Q_1, \Sigma, \delta_1, q_0^{(1)}, F_1)$ and $D = (Q_2, \Sigma, \delta_2, q_0^{(2)}, F_2)$ accepting regular languages $L(C)$ and $L(D)$, the DFAs recognizing $L(C) \cap L(D)$, $L(C) \cup L(D)$ and $L(D) \setminus L(C)$ can be realized as a suitable product automaton $C \times D$. The state space in this automaton is $Q_1 \times Q_2$ (so that the number of states is $|Q_1| \cdot |Q_2|$), and the initial state is the pair $(q_0^{(1)}, q_0^{(2)})$. The transition function $\delta_1 \times \delta_2$ is defined as

$$(\delta_1 \times \delta_2)((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)).$$

For $L(C) \cap L(D)$, the accepting states correspond to (q_1, q_2) pairs such that both $q_1 \in F_1$ and $q_2 \in F_2$. For $L(C) \cup L(D)$, the accepting states correspond to (q_1, q_2) pairs such that either $q_1 \in F_1$ or $q_2 \in F_2$. For $L(D) \setminus L(C)$, the accepting states correspond to (q_1, q_2) pairs such that $q_1 \notin F_1$ and $q_2 \in F_2$.

Appendix C. Deferred Proofs from Section 3

Proposition 6 (Rank Iteration Space Complexity) $\texttt{rankIteration}(s, \prec, j)$ *given in Algorithm 1 requires* $\mathrm{poly}(s, k)$ *bits of memory, provided one can check "$C \prec D$?" using* $\mathrm{poly}(s, k)$ *space for any two DFAs C and D with at most s states.*

Proof Within $\texttt{rankIteration}(s, \prec, j)$, keeping track of count, and the current value of r in the outer for loop requires $\log(N) = \mathrm{poly}(s, k)$ space. The inner loop, which traverses the collection of DFAs, also requires keeping track of only $\mathrm{poly}(s, k)$ bits of memory. For example, this can be implemented by keeping track of s bits indicating the states present in the DFA and $sk \log s$ bits indicating the transitions at each state and alphabet symbol, followed by elementary checks to ensure that the implied DFA is valid. Thereafter, observe that we can reuse the space required by $\texttt{computeRank}(\cdot, \prec)$ in every iteration of the nested for loop. Thus, the total space complexity of Algorithm 1 is $\mathrm{poly}(s, k)$ *plus* the space required to implement $\texttt{computeRank}(\cdot, \prec)$.

We now argue that $\texttt{computeRank}(\cdot, \prec)$ can be implemented efficiently. Again, the outer for loop requires keeping track of ℓ which requires $\log(N) = \mathrm{poly}(s, k)$ space, and the inner for loop which traverses the collection of DFAs also requires keeping track of $\mathrm{poly}(s, k)$ bits, as argued above. Thereafter, every iteration of the nested for loop can reuse the space required by $\texttt{path}(\cdot, \cdot, \ell, \prec)$.

It thus remains to argue that $\texttt{path}(\cdot, \cdot, \ell, \prec)$ can be implemented using $\mathrm{poly}(s, k)$ space. Here, we realize that the recursive invocation to $\texttt{path}(\cdot, \cdot, \ell - \lceil \ell/2 \rceil, \prec)$ can reuse the space used by the recursive invocation to $\texttt{path}(\cdot, \cdot, \lceil \ell/2 \rceil, \prec)$. Furthermore, each invocation needs to store only $\mathrm{poly}(s, k)$ bits corresponding to the values of C, D and ℓ it is invoked with before recursing. The

recursion depth until the base case is reached is $\log(\ell) \leq \log(N) = \mathrm{poly}(s,k)$. Finally, the base case for $\ell = 1$ requires checking if two DFAs C and D satisfy $C \prec D$, which, by assumption, requires $\mathrm{poly}(s,k)$ space. In total, we obtain that $\texttt{path}(\cdot,\cdot,\ell,\prec)$ requires $\mathrm{poly}(s,k)$ space. ■

Proposition 7 ($\prec_2$ Strict Partial Order) *The relation $\prec_2$ defined in (3) is a strict partial order.*

Proof Recall the definition of $\prec_2$:

$$A \prec_2 A' \iff |L(A) \setminus L(A')| < \infty \text{ and } |L(A) \setminus L(A')| < |L(A') \setminus L(A)|.$$

We will argue that $\prec_2$ is irreflexive ($A \not\prec_2 A$), asymmetric ($A \prec_2 B \implies B \not\prec_2 A$) and transitive ($A \prec_2 B$ and $B \prec_2 C \implies A \prec_2 C$).

1. Irreflexivity: $A \not\prec_2 A$, since $|L(A) \setminus L(A)| = 0$, and $0 \not< 0$.

2. Asymmetry: Suppose $A \prec_2 B$, which means that $|L(A) \setminus L(B)| < \infty$ and $|L(A) \setminus L(B)| < |L(B) \setminus L(A)|$. Assume for the sake of contradiction that $B \prec_2 A$. This means that $|L(B) \setminus L(A)| < \infty$ and $|L(B) \setminus L(A)| < |L(A) \setminus L(B)|$. But when both $|L(A) \setminus L(B)|$ and $|L(B) \setminus L(A)|$ are finite, $|L(A) \setminus L(B)| < |L(B) \setminus L(A)|$ and $|L(B) \setminus L(A)| < |L(A) \setminus L(B)|$ cannot simultaneously hold.

3. Transitivity: Suppose $A \prec_2 B$ and $B \prec_2 C$; we want to show that $A \prec_2 C$. Since $A \prec_2 B$ and $B \prec_2 C$, we have that $|L(A) \setminus L(B)| < \infty$ and $|L(B) \setminus L(C)| < \infty$. Then,

$$\begin{aligned} & L(A) \setminus L(C) \subseteq (L(A) \setminus L(B)) \cup (L(B) \setminus L(C)) \\ \implies \quad & |L(A) \setminus L(C)| \leq |L(A) \setminus L(B)| + |L(B) \setminus L(C)| < \infty. \qquad \text{(union bound)} \end{aligned}$$

   Thus, $|L(A) \setminus L(C)| < \infty$. Now, we wish to further show that $|L(A) \setminus L(C)| < |L(C) \setminus L(A)|$. Towards this, we will use the following key identity, which expresses $L(A) \setminus L(B)$ as a disjoint union:

$$\begin{aligned} & L(A) \setminus L(B) = (L(A) \setminus (L(B) \cup L(C))) \sqcup ((L(A) \cap (L(C)) \setminus L(B)), \\ \implies \quad & |L(A) \setminus L(B)| = |L(A) \setminus (L(B) \cup L(C))| + |(L(A) \cap (L(C)) \setminus L(B)|. \end{aligned}$$

   Expressing all remaining pairwise set differences in this fashion, we have

$$\begin{aligned} |L(B) \setminus L(A)| &= |L(B) \setminus (L(A) \cup L(C))| + |(L(B) \cap (L(C)) \setminus L(A)|, \\ |L(B) \setminus L(C)| &= |L(B) \setminus (L(C) \cup L(A))| + |(L(B) \cap (L(A)) \setminus L(C)|, \\ |L(C) \setminus L(B)| &= |L(C) \setminus (L(B) \cup L(A))| + |(L(C) \cap (L(A)) \setminus L(B)|, \\ |L(A) \setminus L(C)| &= |L(A) \setminus (L(C) \cup L(B))| + |(L(A) \cap (L(B)) \setminus L(C)|, \\ |L(C) \setminus L(A)| &= |L(C) \setminus (L(A) \cup L(B))| + |(L(C) \cap (L(B)) \setminus L(A)|. \end{aligned}$$

   Since $A \prec_2 B$ and $B \prec_2 C$, we also have that $|L(A) \setminus L(B)| < |L(B) \setminus L(A)|$ and $|L(B) \setminus L(C)| < |L(C) \setminus L(B)|$. Using the above identities, this means that

$$\begin{aligned} & |L(A) \setminus (L(B) \cup L(C))| + |(L(A) \cap (L(C)) \setminus L(B)| \\ & \qquad < |L(B) \setminus (L(A) \cup L(C))| + |(L(B) \cap (L(C)) \setminus L(A)|, \qquad (4) \\ & |L(B) \setminus (L(C) \cup L(A))| + |(L(B) \cap (L(A)) \setminus L(C)| \\ & \qquad < |L(C) \setminus (L(B) \cup L(A))| + |(L(C) \cap (L(A)) \setminus L(B)|. \qquad (5) \end{aligned}$$

All the quantities in the LHS of both (4) and (5) are finite, since both $|L(A) \setminus L(B)|$ and $|L(B) \setminus L(C)|$ are finite. Furthermore, this also implies that the term $|L(B) \setminus (L(A) \cup L(C))|$ in the RHS of (4), and the term $|(L(C) \cap (L(A)) \setminus L(B)|$ in the RHS of (5) are both finite.

Now recall: we wish to show that $|L(A) \setminus L(C)| < |L(C) \setminus L(A)|$. Since we have shown that $|L(A) \setminus L(C)| < \infty$, if $|L(C) \setminus L(A)| = \infty$, then the inequality already holds. So, suppose that $|L(C) \setminus L(A)| < \infty$. Using the identity for $|L(C) \setminus L(A)|$ above, this means that both $|L(C) \setminus (L(A) \cup L(B))|$ and $|(L(C) \cap (L(B)) \setminus L(A)|$ are finite. But then, all the terms in the inequalities (4) and (5) are finite. Adding the two inequalities, and canceling the common terms on both sides, we get

$$\begin{aligned} |L(A) \setminus (L(C) \cup L(B))| + |L(A) \cap (L(B) \setminus L(C))| \\ < |L(C) \setminus (L(A) \cup L(B))| + |L(C) \cap (L(B) \setminus L(A))|. \end{aligned} \tag{6}$$

But then, using the identities above one final time, this means that $|L(A) \setminus L(C)| < |L(C) \setminus L(A)|$, which is the desired inequality. Thus, we have shown that $A \prec_2 C$.

■

Lemma 9 (Symmetric Difference Contains Length-$O(s)$ Strings) *Let $A = (Q_A, \Sigma, \delta_A, q_0^A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_0^B, F_B)$ denote DFAs having at most s states over a common alphabet Σ, such that $|L(A) \triangle L(B)| < \infty$. Then, for any string $w \in \Sigma^*$, if $|w| \geq 2s - 1$, then $w \notin L(A) \triangle L(B)$.*

Proof The result essentially follows from Lemma 5 in Gawrychowski et al. (2011); we flesh out a detailed proof for completeness.

Consider a "disjoint union automaton" M, whose state space $Q_M = Q_A \sqcup Q_B$, so that $|Q_M| = |Q_A| + |Q_B| \leq 2s$. In M, we retain transitions originally within states in Q_A and Q_B, but there are no transitions across states in Q_A and Q_B. That is, for any alphabet $\sigma \in \Sigma$, for any $q \in Q_A$, we have $\delta_M(q, \sigma) = \delta_A(q, \sigma)$, and for any $q \in Q_B$, we have $\delta_M(q, \sigma) = \delta_B(q, \sigma)$. M has two distinct initial states q_0^A and q_0^B, and accepting states $F = F_A \sqcup F_B$. We note that M is not exactly a valid automaton, and we only construct it for the purpose of analysis.

Let us the define the "right language" of a state $q \in M$ as follows:

$$L_M^+(q) = \{w \in \Sigma^* : \delta_M(q, w) \in F\}$$

We immediately have that $L_M^+(q_0^A) = L(A)$ and $L_M^+(q_0^B) = L(B)$. Now, for any $p, q \in Q_M$, define the following (pseudo)distance $d_M(p, q)$:

$$d_M(p, q) = \min\{\ell \geq 0 : L_M^+(p) \cap \Sigma^{\geq \ell} = L_M^+(q) \cap \Sigma^{\geq \ell}\}. \tag{7}$$

where $\Sigma^{\geq \ell} = \{w \in \Sigma^* : |w| \geq \ell\}$. In words, $d_M(p, q)$ is the smallest ℓ, such that the right languages $L_M^+(p)$ and $L_M^+(q)$, restricted to strings of length at least ℓ, are equal. We can verify that $d_M(p, q)$ satisfies the following recursive formula:

$$d_M(p, q) = \begin{cases} 0 & \text{if } L_M^+(p) = L_M^+(q) \\ 1 + \max_{\sigma \in \Sigma}\{d_M\left(\delta_M(p, \sigma), \delta_M(q, \sigma)\right)\} & \text{otherwise.} \end{cases} \tag{8}$$

In fact, $d_M(p,q)$ is a so-called "ultrapseudometric" over the states in Q_M: we can immediately verify that $d_M(p,p) = 0$, $d_M(p,q) \geq 0$ always and $d_M(p,q) = d_M(q,p)$; more importantly, $d_M(p,q)$ also satisfies the strong triangle inequality:

$$d_M(p,q) \leq \max(d_M(p,r), d_M(r,q)). \tag{9}$$

To see this, note that the inequality holds immediately if either $d_M(p,r) = \infty$ or $d_M(r,q) = \infty$. Otherwise, let $d_M(p,r) = a < \infty$ and $d_M(r,q) = b < \infty$. By definition of $d_M(\cdot,\cdot)$, we have that

$$\begin{aligned} & L_M^+(p) \cap \Sigma^{\geq a} = L_M^+(r) \cap \Sigma^{\geq a} \quad \text{and} \quad L_M^+(r) \cap \Sigma^{\geq b} = L_M^+(q) \cap \Sigma^{\geq b} \\ \implies \quad & L_M^+(p) \cap \Sigma^{\geq \max(a,b)} = L_M^+(q) \cap \Sigma^{\geq \max(a,b)} \\ \implies \quad & d_M(p,q) \leq \max(a,b) = \max(d_M(p,r), d_M(r,q)), \end{aligned}$$

as required.

We now claim that, for any two states $p, q \in Q_M$, if $d_M(p,q) < \infty$, then $d_M(p,q) < |Q_M| \leq 2s$. Before we go ahead and prove this, let us first see how it implies the lemma. For this, consider $d_M(q_0^A, q_0^B)$. Recall that $L_M^+(q_0^A) = L(A)$ and $L_M^+(q_0^B) = L(B)$, and by assumption, $|L(A) \triangle L(B)| < \infty$. But this necessarily means that $d_M(q_0^A, q_0^B) < \infty$ (otherwise, there would be arbitrarily long strings in the symmetric difference, making it an infinite set). So, by our claim, it holds that $d_M(q_0^A, q_0^B) \leq 2s$. By definition of $d_M(\cdot,\cdot)$, this implies that any string in Σ^* of length at least $2s-1$ is either in both of $L(A)$ and $L(B)$ or in neither, and hence proves the lemma.

We now proceed to proving the promised claim: if $d_M(p,q) < \infty$, then $d_M(p,q) < |Q_M|$. Towards this, for any $i \geq 0$, consider the equivalence relation D_i over the set Q_M defined as follows:

$$(p,q) \in D_i \iff d_M(p,q) \leq i. \tag{10}$$

To see that this is an equivalence relation for every i, note that $(p,p) \in D_i$ always since $d_M(p,p) = 0$ (reflexivity), and also, $(p,q) \in D_i \implies (q,p) \in D_i$ since $d_M(p,q) = d_M(q,p)$ (symmetry). For the transitive property, we use the strong triangle inequality (9) from above: if $(p,r) \in D_i$ and $(r,q) \in D_i$, meaning that $d_M(p,r) \leq i, d_M(r,q) \leq i$, then

$$d_M(p,q) \leq \max(d_M(p,r), d_M(r,q)) \leq i,$$

meaning that $(p,q) \in D_i$.

Now, let n_i be the number of equivalence classes that D_i partitions Q_M into. We have that $n_0 \leq |Q_M|$, and every $n_i \geq 1$. Observe now that by definition of D_i, it is also the case that $D_0 \subseteq D_1 \subseteq D_2 \subseteq \ldots$. This immediately implies that

$$n_0 \geq n_1 \geq n_2 \ldots.$$

Since $n_0 \leq |Q_M| < \infty$ and each $n_i \geq 1$, there must be some finite i for which $n_i = n_{i+1}$. We now claim that if $n_i = n_{i+1}$ for any $i \geq 0$, then both: (1) for any two states $p, q \in Q_M$, $d_M(p,q) > i \implies d_M(p,q) = \infty$, and (2) $n_j = n_i$ for every $j > i$.

For (1), first note that $n_i = n_{i+1}$ implies that $D_i = D_{i+1}$. This follows simply because $D_i \subseteq D_{i+1}$, and both are equivalence relations on Q_M. Namely, if there is any pair $(p,q) \in D_{i+1}$ which is not in D_i, then this pair would merge two equivalence classes in D_i, contradicting $n_i = n_{i+1}$.

Now, fix $p, q \in Q_M$, and suppose that $d_M(p,q) > i$. We will first argue that $d_M(p,q) \neq i+1$. Suppose not, meaning that $d_M(p,q) = i+1$. Then, by definition, $(p,q) \in D_{i+1}$. But $(p,q) \notin D_i$, contradicting that $D_i = D_{i+1}$. Thus, $d_M(p,q) \neq i+1$.

Now, we argue that $d_M(p,q) \neq j$ for every finite $j > i+1$. Suppose not, meaning that $d_M(p,q) = j$ for some finite $j > i+1$. Since $j > i+1 \geq 1$, by the recursive definition in (8),

$$j = d_M(p,q) = 1 + \max_{\sigma \in \Sigma} \{d_M\left(\delta_M(p,\sigma), \delta_M(q,\sigma)\right)\}.$$

This means that for $\sigma \in \Sigma$ which is the maximizer above,

$$d_M\left(\delta_M(p,\sigma), \delta_M(q,\sigma)\right) = j - 1.$$

Denoting $\delta_M(p,\sigma) := p_1$ and $\delta_M(q,\sigma) := q_1$, we have that $\delta_M(p_1, q_1) = j-1$. Let $t = j - (i+1)$. Since j is finite, t is finite; so repeating the argument above t times, we will obtain that

$$\delta_M(p_t, q_t) = j - t = i + 1.$$

But this means that $d_M(p_t, q_t) > i$, which, as we established above, necessarily means that $\delta_M(p_t, q_t) \neq i+1$. This is a contradiction, and hence $d_M(p,q) \neq j$. This proves (1).

We now turn towards proving (2). Consider any $j > i$. For any two states $p, q \in Q_M$, if $d_M(p,q) \leq j < \infty$, then (1) above implies that $d_M(p,q) \leq i$. We also trivially have that $d_M(p,q) \leq i \implies d_M(p,q) \leq j$. That is, for any two states $p, q \in Q_M$, it holds that

$$(p,q) \in D_i \iff d_M(p,q) \leq i \iff d_M(p,q) \leq j \iff (p,q) \in D_j.$$

Thus, $D_i = D_j$ and hence $n_j = n_i$ for every $j > i$, which proves (2).

Finally, putting (2) together with the fact that $|Q_M| \geq n_0 \geq n_1 \geq \ldots$, where each $n_i \geq 1$, we necessarily have that by $i = |Q_M| - 1$, it holds that $n_i = n_{i+1}$. But then the contrapositive of (1) above implies that for any $p, q \in Q_M$, $d_M(p,q) < \infty \implies d_M(p,q) \leq |Q_M| - 1$, which completes the entire proof. ■

Theorem 10 (Space-Efficient Language Generation) *Let K denote the unknown target language recognized by a DFA of size at most s. There exists an algorithm $\mathcal{A}$ which, for all sufficiently large t, outputs a DFA $A' = \mathcal{A}(t)$ that satisfies $L(A') \subseteq K$, and uses only $\mathrm{poly}(s,k)$ space. Furthermore, the algorithm misses at most $|K \setminus L(A')| \leq O(k^{2s-2})$ target strings.*

Proof The algorithm $\mathcal{A}$ runs `SpaceEfficientTraversal`$(\prec_2)$, which outputs, in the limit, a DFA $A_{i(t)}$ that satisfies $|L(A_{i(t)}) \triangle K| < \infty$ (by Proposition 8). Let B be a DFA that accepts all strings of length at least $2s-1$, and rejects all strings of smaller length; B can be constructed with $2s$ states $q_0, \ldots, q_{2s-1}$ where every q_i always transitions to q_{i+1} for $0 \leq i < 2s-1$. The initial state is q_0, every q_i for $1 \leq i < 2s-1$ is a rejecting sink state, and q_{2s-1} is the only accepting sink state. By Lemma 9, we have that any $w \in L(B)$ is either in $L(A_{i(t)}) \cap K$ or in $\Sigma^* \setminus (L(A_{i(t)}) \cup K)$.

So, consider the DFA A' which is the product DFA recognizing $L(A_{i(t)}) \cap L(B)$. The DFA A' has at most $s \cdot 2s = 2s^2$ states, and satisfies that $L(A') \subseteq L(A_{i(t)}) \cap K \subseteq K$. Furthermore, since any $w \in K \setminus A_{i(t)}$ must necessarily have length smaller than $2s-1$, $K \setminus A_{i(t)} \subseteq K \setminus L(B)$, and hence

$$K \setminus L(A') \subseteq (K \setminus A_{i(t)}) \cup (K \setminus L(B)) = K \setminus L(B). \tag{11}$$

Thus, $L(A')$ is only deficient of the strings in K that have length at most $2s-2$. If $|\Sigma| = k = 1$, we can have the algorithm $\mathcal{A}$ additionally keep track of a set S of all the strings seen in the input that have length at most $2s-2$. Since there are at most $2s-1$ such strings (including the empty string), this requires only $\mathrm{poly}(s)$ memory. Furthermore, since all the strings in K are eventually revealed, the set S eventually contains all strings in K that have length at most $2s-2$. Finally, since S can be recognized by a DFA having at most $2s-1$ states, the algorithm can output $\mathcal{A}(t)$ to be the product DFA that recognizes $L(A') \cup S$, which has at most $2s^2 \cdot (2s-1) = O(s^3)$ states. In this case, we are guaranteed that $L(\mathcal{A}(t)) = K$ eventually.

If $k > 1$, we simply have the algorithm output the DFA $\mathcal{A}(t) = A'$. In this case, (11) gives that $|K \setminus L(A')| \leq |K \setminus L(B)| = \frac{k^{2s-1}-1}{k-1} = O(k^{2s-2})$. ■

Let us revisit the trick we used above for the case where $|\Sigma| = 1$; here, we had the algorithm keep track of the set S of input strings that had length at most $2s-2$. With a unary alphabet, S could have at most $O(s)$ strings, and hence the algorithm could store all of these in its $\mathrm{poly}(s,k)$ memory budget. In fact, this allowed the algorithm to achieve the *stronger* guarantee that $\mathcal{A}(t) = K$; i.e., $\mathcal{A}$ identified the target K in the limit! When $k > 1$, storing the set S requires $\exp(s,k)$ memory, which is why we simply returned A' in this case. Nevertheless, we note that with $\exp(s,k)$ memory, we can use the same trick as in the $|\Sigma| = 1$ case, and achieve identification in the limit.

Appendix D. Deferred Proofs from Section 4

Proof [Proof of Lemma 11] Suppose that a deterministic protocol for INDEX_n uses $c < n \log_2 k$ bits of communication. Then, there are at most $2^c < k^n$ possible messages that the protocol can send, meaning that there exist two vectors $x, y \in [k]^n$, with $x \neq y$ on which the protocol sends the same message. Let i be one coordinate in which x and y differ. In both instances of the problem (namely, (x,i) and (y,i)), Bob receives the same message and produces the same answer, even though $x_i \neq y_i$, which contradicts the correctness of the protocol. ■

Relaxed symmetric-difference lower bound. We also spell out the proof of the second part of Theorem 4, corresponding to the relaxed objective mentioned after Definition 2. In this formulation, the learner is not required to satisfy $L(\mathcal{A}(t)) \subseteq K$; instead, for every target $K \in \mathcal{C}_{s,k}$ and every surjective enumeration of K, the limiting hypothesis must satisfy

$$|L(\mathcal{A}(t)) \triangle K| \leq \widetilde{\Delta}_{\mathcal{A}}(s,k)$$

for all sufficiently large t.

We use exactly the same INDEX_n reduction and the same languages as in the proof of Lemma 12. Fix Bob's index i, and write $B_q := \mathrm{Base}(i,q)$ for $q \in \Sigma$. If Alice's input is x, then after Alice's prefix and Bob's continuation, the target language presented to the learner is B_{x_i}. Let H be the limiting DFA output by the learner on this stream. Assume for contradiction that

$$\widetilde{\Delta}_{\mathcal{A}}(s,k) < k^{\ell}.$$

Under the new definition, this means that

$$|L(H) \triangle B_{x_i}| < k^{\ell}.$$

For any $q \neq x_i$, the two candidate target languages B_q and B_{x_i} differ exactly on the two disjoint blocks $L_{\text{map}(i,q)}$ and $L_{\text{map}(i,x_i)}$. Therefore,

$$|B_q \triangle B_{x_i}| = 2k^\ell.$$

By the triangle inequality for symmetric difference,

$$|L(H)\triangle B_q| \geq |B_q \triangle B_{x_i}| - |L(H)\triangle B_{x_i}| > 2k^\ell - k^\ell = k^\ell.$$

On the other hand, $|L(H)\triangle B_{x_i}| < k^\ell$. Hence $q = x_i$ is the unique minimizer of $|L(H)\triangle \text{Base}(i,q)|$ over $q \in \Sigma$.

Bob can therefore decode x_i by computing, for every $q \in \Sigma$, the size of the regular language $L(H)\triangle\text{Base}(i,q)$, and outputting the unique minimizer. Concretely, Bob constructs the product DFA for this symmetric difference and counts its accepted language when it is finite; an infinite value may be treated as $+\infty$. Under the promise above, the true candidate is finite and uniquely smallest. This gives the same one-way INDEX$_n$ protocol as in Lemma 12 with Alice's message equal to the memory state of the learner. The INDEX$_n$ lower bound is contradicted whenever that memory state has fewer than $n \log_2 k$ bits. Thus $\widetilde{\Delta}_{\mathcal{A}}(s,k) \geq k^\ell$, and substituting the same values of p, ℓ, n as in Lemma 12 gives the same quantitative lower bound.

Further remarks for the Proof of Lemma 12. As we can see in the proof of INDEX$_n$ above, the lower bound is purely information-theoretic and contains no requirements about the computation power of Alice and Bob. This means that even the existence of two functions, an encoding function of x to $\ell < n \log_2 k$ bits, let it be $f : [k]^n \to [2]^\ell$, and a decoding function $g : [n] \times [2]^\ell \to [k]$ such that for all $x \in [k]^n$ and all $i \in [n]$ it holds that $g(i, f(x)) = x_i$, violates the lower bound. In our construction, you can view $f(x)$ as the memory state σ of the learner after Alice has given it her part of the input. Now, let $t^*(i,q)$ be the timestep after which the learner $\mathcal{A}$ has converged to its limiting DFA, on the enumeration of our construction when the target language is $\text{Base}(i,q)$. Let also T^* be the maximum over $t^*(i,q)$ for all $i \in [n]$ and $q \in [k]$. Then, the decoding function g can be defined as the output of Bob when he waits for $(T^* + 1)$ time steps and then performs the corresponding finite DFA test. Of course, the existence of g contradicts the INDEX$_n$ lower bound.

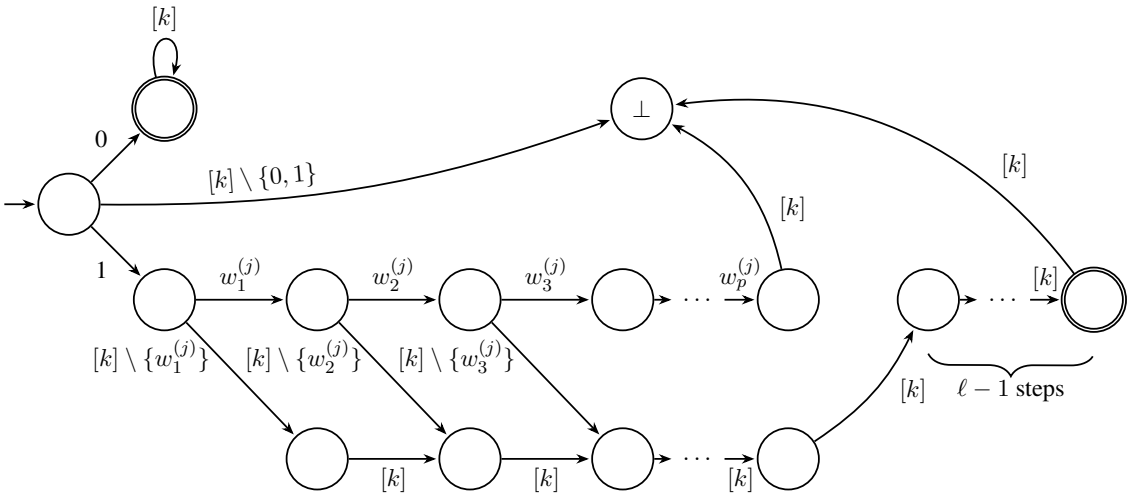

Figure 1: A Deterministic Finite Automaton (DFA) accepting Base(j, q) for some $j \in [n]$ and $q \in \Sigma$. We use the notation $\text{enc}_k(j) = w_1^{(j)} \circ w_2^{(j)} \circ w_3^{(j)} \circ \cdots \circ w_p^{(j)}$