# Improved Bounds on the Space Complexity of Circuit Evaluation

Yakov Shalunov University of Chicago yasha@uchicago.edu

April 30, 2025

#### Abstract

Williams (STOC 2025) recently proved that time-t multitape Turing machines can be simulated using  $O(\sqrt{t\log t})$  space using the Cook-Mertz (STOC 2024) tree evaluation procedure. As Williams notes, applying this result to fast algorithms for the circuit value problem implies an  $O(\sqrt{s} \cdot \text{polylog } s)$  algorithm for evaluating size s circuits.

In this work, we provide a direct reduction from circuit value to tree evaluation without passing through Turing machines, simultaneously improving the bound to  $O(\sqrt{s \log s})$  space and providing a proof with fewer abstraction layers.

This result can be thought of as a "sibling" result to Williams' for circuit complexity instead of time; in particular, using the fact that time-t Turing machines have size  $O(t \log t)$  circuits, we can recover a slightly weakened version of Williams' result, simulating time-t machines in space  $O(\sqrt{t} \log t)$ .

## 1 Introduction

Recently, Williams proved an extremely counterintuitive result: that all time t algorithms can be simulated using only  $\tilde{O}(\sqrt{t})$  space, no matter how they use the  $\Omega(t)$  cells of space they can touch in time t. This improved on the 50-year-old best-known simulation using space  $O(t/\log t)$  due to Hopcroft, Paul, and Valiant [7].

Since this is a matter of fine-grained time complexity, we remark that time t refers to specifically multitape Turing machine time and in this work, we attack an alternate model of fine-grained complexity: circuit size. Circuit size is closely related to Turing machine time, as shown below in theorems 1.3 and 1.4.

As a result, we believe circuit size to be a similarly motivated (and perhaps, more elegant) measure of fine-grained complexity which lends itself to a somewhat more direct proof for the analogous result:

**Theorem 1.1** (Main theorem). Circuits of size s can be evaluated in space  $O(\sqrt{s \log s})$ .

For completeness, Williams' result is stated precisely as:

**Theorem 1.2** (Williams [11]). Multitape Turing machines running in time  $t(n) \ge n$  can be simulated in space  $O(\sqrt{t \log t})$ .

As stated above, multitape Turing machine time and circuit size are closely related measures of complexity.

**Theorem 1.3** (Pippenger [8]). CIRCUITEVAL  $\in \text{TIME } [s \log^2 s]$ .

*Remark.* Pippenger's proof of this result does not appear to be available in digital form, nor were we able to locate any other expositions that proved the same bound. A rough description of the algorithm can be found in [11].

**Theorem 1.4.** For every  $t(n) \ge n$  and  $L \in \mathsf{TIME}[t]$ , the polylog-space-uniform<sup>1</sup> circuit complexity of L is  $O(t \log t)$ .

The latter result follows immediately from two facts about oblivious Turing machines. (An "oblivious" Turing machine is one whose head position at any given time step i depends only on the input length n and not the specific input x.) The first result is as seen in [11].

**Theorem 1.5** ([6, 9, 4]). For every deterministic, time t(n) multitape Turing machine M, there exists an equivalent oblivious Turing machine running in time  $O(t \log t)$ . Further, the head positions at time i can be computed in time polylog t.

**Theorem 1.6** ([10]). If L is decided by an oblivious Turing machine running in time t(n) whose head positions can be efficiently computed then L can be computed by uniform circuits of size O(t).

As noted, our analogous result is an incomparable sibling result to theorem 1.2:

- Williams' result (theorem 1.2) together with theorem 1.3 gives a space  $O(\sqrt{s}\log^{3/2} s)$  algorithm for the circuit value problem.
- Theorem 1.1 together with theorem 1.4 gives TIME  $[t] \subseteq \mathsf{SPACE}\left[\sqrt{t}\log t\right]$  (in this sense, our result generalizes the "warm-up" argument in Williams' paper to work on arbitrary circuits rather than only the implicit ones corresponding to oblivious Turing machines).

Our result is somewhat more straightforward, avoiding reasoning about the intricacies of the multitape Turing machine model or needing to "guess" the movements of nonoblivious machines and instead works with concrete modifications to the circuit being evaluated.

## 2 Preliminaries

**Definition 2.1** (Circuit Evaluation). Full Circuit Evaluation instances take the form (x, C) where x is an m-bit string and C is an s-gate circuit in the full binary basis on m inputs, provided as a list of triples of the form  $(\ell_i, r_i, \varphi_i)$  where  $\varphi_i : \{0, 1\}^2 \to \{0, 1\}$  is a binary function and each of  $\ell_i$  and  $r_i$  are either a gate index  $j \neq i$  or a variable index k for some  $k \leq m$ .

The objective is to output the value  $v_i$  of each gate. For simplicity, assume  $s \geq m$ .

A CIRCUITEVAL instance C is an s-gate circuit in the full binary basis provided as a list of triples of the form  $(\ell_i, r_i, \varphi_i)$  where  $\varphi_i : \{0, 1\}^2 \to \{0, 1\}$  is a binary function and each of  $\ell_i$  and  $r_i$  are either a gate index j < i (i.e., the input is in topologically sorted order) or a constant bit.

The objective is to output the value of the last gate.

<sup>&</sup>lt;sup>1</sup>Specifically, given two vertices, whether they are connected can be computed in polylog t time (and thus space), and correspondingly, the entire circuit can be produced using polylog t space.

<sup>&</sup>lt;sup>2</sup>The original result does not argue any uniformity, but the construction is a modified tableau which uses the knowledge of the head positions to add only constant number of gates (corresponding to the cell under the head) per time step; correspondingly, if the head positions are computable in space S, the circuit is constructible in space  $O(S + \log t)$ .

Remark. Formulations of the circuit evaluation/circuit value problem differ. However, all that we are aware of reduce trivially to the FULLCIRCUITEVAL formulation above. The following lemma justifies working with the simplified CIRCUITEVAL.

Lemma 2.1 (CIRCUITEVAL is good enough).

$$\texttt{CIRCUITEVAL} \in \mathsf{SPACE}\left[S\right] \implies \texttt{FULLCIRCUITEVAL} \in \mathsf{SPACE}\left[S + \log^2 s\right]$$

*Proof.* First, observe that given an instance of FullCircuitEval, we can replace every reference to variable k with the value  $x_k$  in space  $\log s$  by simply scanning over the gates of C and for each one, if  $\ell_i$  is a gate index, outputting it as is, and if it is a variable index k then storing the current gate in  $\log s$  bits and the variable index in  $\log s$  bits and scanning back to find  $x_k$ , outputting that value instead of k. Similarly for  $r_i$ .

Next, observe that it is possible to topologically sort a graph with a sink reachable from all vertices in  $O(\log^2 s)$  space. This observation is due to Hoza (private communication) and uses techniques from the proof of Savitch's theorem: we can compute whether there exists a path of length  $k \leq s$  between any two vertices using  $O(\log^2 s)$  space. Call this subroutine  $\mathsf{path}(u,v,k)$ . To topologically sort a graph, we pick a sink node r. Then for each  $k = s - 1, \ldots, 0$ , for each vertex v, we output v if

$$\operatorname{path}(v, r, k) \wedge \bigwedge_{k' < k} \neg \operatorname{path}(v, r, k')$$

In this way, we first output all vertices distance exactly s-1 from the sink, then all vertices distance s-2, etc., outputting a topological sort of the graph. (The iteration over k, v, and k' each require  $O(\log s)$  bits, so the space usage is dominated by the  $O(\log^2 s)$  space usage of the path routine.)

Finally, suppose M computes CIRCUITEVAL in space S. We use space-efficient composition and the above procedures to run M on gates  $\{0,\ldots,i\}$  for each i, sorting with respect to the sink i.<sup>3</sup> Between computations we only need to store which gate we're on using  $\log s$  bits and everything else can be reused.

# 3 Proof ingredients

A key ingredient, perhaps the central one, is the tree evaluation problem:

**Definition 3.1** (Tree Evaluation). TREEEVAL instances are full binary trees of height h where each leaf  $\ell$  is labeled with a b-bit string  $v_{\ell}$  and each internal node u is labeled with an explicit function (provided as a table of values)  $f_u: \{0,1\}^{2b} \to \{0,1\}^b$ .

We recursively define the value  $v_u$  at each internal node u in the natural way to be

$$v_u := f_u(v_{u0}, v_{u1})$$

where u0 and u1 are the left and right children of u. The objective is to evaluate  $v_{\text{root}} \in \{0,1\}^b$ , the value of the root.

The problem can also be generalized to the case where we consider d-ary trees instead of binary ones in the natural way.

 $<sup>^{3}</sup>$ this would omit any gates which don't have a path to gate i, but this means their value is unused in computing the value of gate i

(For the history and broader significance of the tree evaluation problem, see Cook and Mertz work [3]. We use the formulation of the tree evaluation problem exposited by Goldreich [5].)

The key fact is that though the problem appears to "inherently" require space  $\Omega(hb)$  (since the naive recursive approach "feels" fairly fundamental to the problem), it turns out that rather counterintuitively, it can be computed far more efficiently with the techniques of catalytic computation.

Theorem 3.1 (Cook-Mertz [3, 5]).

TREEEVAL 
$$\in$$
 SPACE  $[b + h \log b]$ 

(The d-ary version exposited by Williams (space  $O(d \cdot b + h \log(d \cdot b))$ ) is also useful for some intermediate motivation.)

Thus, Cook and Mertz are able to obtain a nearly-quadratic improvement over the original<sup>4</sup> "fundamental" usage of O(hb) space. It turns out, further, that this quadratic improvement can be carried to even more counterintuitive statements, as demonstrated in Williams' work.

Williams applies it to Turing machines by decomposing computation into time blocks which form a computation graph (with vertices representing connects to the last time block where a used tape block was modified) which can expanded into a tree. Naturally, the size of the blocks multiplies with the number of blocks (which bounds the depth) to form the original time—balancing the size of the blocks with the number of blocks yields the square root result.

Our result similarly relies on the idea of decomposing the computation, a circuit in this case, into blocks and bounding the depth of this quotient graph. In order to ensure that the resulting quotient graph is still a DAG, we group vertices in topologically sorted order.

Unlike with Turing machines, however, this block graph does not a priori have bounded fan-in. Our main technical contribution is a technique for reducing to a constant fan-in without increasing the depth. We introduce "forwarding" blocks which don't perform any actual computation but serve to simply "bring together" all the incoming wires gradually. The result on the underlying circuit can be thought of as a "semi-layered" construction with "thick layers" of depth  $\sqrt{s}$  where edges are only allowed within layers or going one layer back. (These terms should not be thought of as rigorous or precisely defined, but hopefully provide some intuition.)

Note that analogously to the process of blowing up a circuit into a formula (by creating a new graph with one vertex for every path from the root in the DAG), we can reduce such a DAG to TREEEVAL. This tree will have size exponential in the depth and thus potentially in the size of the DAG, but we will never actually write down the entire tree, simply computing it on-demand using the standard techniques of space-efficient composition.

Our theorem can be restated as:

**Theorem 3.2** (Main theorem, restated). Given a circuit of size s, for any  $b \in [s]$ , we can produce an equivalent Tree-Eval instance with  $h = O(\frac{s}{b})$  and b as chosen. Further, this reduction is computable in space  $O(b + \frac{s}{b})$ .

In particular, theorem 3.1 immediately gives theorem 1.1; further, a space O(h + b) procedure for TreeEval would immediately imply a space  $O(\sqrt{s})$  procedure for Circuiteval.

It is worth noting that using recursive invocations of this result, there is a local version of this reduction using space  $O(\sqrt{b \log b} + \log s)$  which on input (u, x, y) (where  $u \in \{0, 1\}^{\leq h}$  specifies a

<sup>&</sup>lt;sup>4</sup>It is worth noting that several years prior to their  $O(\log n \cdot \log \log n)$  result, Cook and Mertz first showed that the problem can be solved in space  $O(\log^2 n/\log \log n)$  [1, 2].

node and  $x, y \in \{0, 1\}^b$ ) outputs  $f_u(x, y)$ . However, this more fine-grained analysis does not provide any additional savings here.

This reduction involves a purely graph-theoretic construction:

**Lemma 3.3** (DAG partition). Given a directed acyclic graph G of size s and maximum in-degree d > 1 and any  $b \in [d, s]$ , there is a subdivision G' of G and a partition P of G' such that:

- Every part in P has size at most b.
- Every vertex in the quotient graph G'/P has in-degree at most d.
- Every directed path in G'/P has length at most  $\frac{d}{d-1} \cdot \frac{s}{b} = O(\frac{s}{b})$  (note that  $d/(d-1) \le 2$ ).
- Given G and b, the subdivision G' and partition P can be computed in  $O(\log s)$  space.

# 4 Technical implementation

Throughout, we will often use graph terminology and identify the circuit with the underlying DAG structure.

### The block graph

We partition the gates into roughly  $\sqrt{s} \approx s/b$  blocks of gates of size roughly  $b \approx \sqrt{s}$ . If  $C = (v_0, \ldots, v_{s-1})$  is in topologically sorted order (it is convenient to reindex from 0), we will let our blocks be  $B_i = (v_{ib}, \ldots, v_{(i+1)b-1})$ .

We create a new "block graph" by taking the quotient graph with respect to this partition. The edges in the block graph are "high bandwidth," carrying  $\sqrt{s}$  bits and representing all the wires between the blocks in a parallel bundle.

The new block graph has several desirable properties:

- 1. It is trivially still a DAG, since if i < j then all vertices in  $B_i$  come before any vertex in  $B_j$ .
- 2. Each block only contains  $\sqrt{s}$  gates, so if we evaluate the entire block, that's only  $\sqrt{s}$  bit output size. Looking ahead to the eventual TREEEVAL instance, this corresponds to  $b \approx \sqrt{s}$ .
- 3. There are a total of  $\sqrt{s}$  blocks in the block graph, so in particular, its depth is at most  $\sqrt{s}$ .

If we produce our TREEEVAL instance immediately, we correspondingly get space usage  $O(d \cdot \sqrt{s} + \sqrt{s} \log(d \cdot \sqrt{s}))$ , which looks a lot like what we want. Unfortunately, we still need to deal with the d. As mentioned above, there is no inherent reason for the fan-in to be constant, rather than  $\sqrt{s}$  (the number of incoming wires).<sup>5</sup> Plugging in a fan-in of  $\sqrt{s}$  makes the first summand,  $d \cdot b = \sqrt{s} \cdot \sqrt{s} = s$ , which is no better than the naive algorithm.

<sup>&</sup>lt;sup>5</sup>E.g., assuming for simplicity that s is a perfect square, consider the graph where  $\ell_i = i - 1$ ,  $r_i$  is arbitrary for  $i < s - \sqrt{s}$ , and  $r_i = (i - s + \sqrt{s}) \cdot \sqrt{s}$  for  $i \ge s - \sqrt{s}$ . That is, we have a chain of edges enforcing an exact topological ordering and then the other edges of the last block connect to a vertex in each of the previous blocks.

### Handling fan-in

The key idea is that we are not working with an arbitrary  $\sqrt{s}$  fan-in tree evaluation problem. An arbitrary one may actually rely on all b bits from all d of the input vertices; in this case, we only need  $2\sqrt{s}$  total input bits, even if they might be spread across all  $\sqrt{s}$  blocks. We can thus introduce additional forwarding blocks to "bring the bits together" with low fan-in.

Let  $b \approx \sqrt{s}$  (to be set exactly later). Our blocks will be of size at most b, but most will be smaller. In particular, some blocks may end up entirely empty with "phantom" connections where they ignore their child inputs and produce b 0's, which are then ignored by their parent. For simplicity, assume b is even.

First divide up into  $B_i$  similarly to before (with the change that the blocks are size b/2 rather than b), with  $B_i = (v_{ib/2}, \dots, v_{(i+1)b/2-1})$ . Let  $t = \lceil 2s/b \rceil - 1$  be the number of "primary" blocks. Note that the blocks are half the maximum allowed size, which ensures that the number of incoming edges is at most the allowed size.

For each block  $B_i$  (which we may refer to as  $B_{i,0}$ ), we will introduce a "tail" of additional blocks,  $B_{i,1}$  through  $B_{i,i}$ . The structure of the graph will be such that  $B_{i,j}$  has edges from  $B_{i-j-1}$  and  $B_{i,j+1}$  for  $j=0,\ldots,i-1$ . To describe these additional blocks, we can add additional gates to C. Each gate  $v=(\ell,r,\varphi)\in B_{i,0}$  at location  $k\in 0,\ldots,b/2-1$  (location in  $B_{i,0}$ ) has two inputs,  $\ell$  and r. For each input leading to a block  $B_0,\ldots,B_{i-2}$ , we will subdivide the edge (introducing an identity gate) and place the new vertex in the block  $B_{i,1}$  (we will say that the location of  $\ell$  in  $B_{i,1}$  is 2k and the location of r is 2k+1).

Observe that now no gate in  $B_i$  connects directly to a gate in  $B_0, \ldots, B_{i-2}$ . Thus,  $B_i$  is necessarily fan-in 2. Further, since this construction added at most two gates to  $B_{i,1}$  for each gate in  $B_i$ , the size of  $B_{i,1}$  is at most b as required.

For each  $B_{i,j}$  for 1 < j < i, the block is defined similarly by subdividing all edges from  $B_{i,j-1}$  to blocks  $B_0, \ldots, B_{i-j-1}$ , except that this time the previous block has size b (rather than b/2) but the in-degree of each vertex is 1. Correspondingly, if a gate has location k in  $B_{i,j-1}$  then the subdivision gate has location k in  $B_{i,j}$  (if it exists).

 $B_{i,i}$  will be an empty block and is added to simplify analysis. Other  $B_{i,j}$  may also be empty if no gates in  $B_i$  have inputs from any  $B_k$  for k < i - j.

It is this graph which we will "blow-up" into a binary tree and feed into TREEEVAL.

#### Functions and leaves

To finish the reduction to TREEEval, we let  $f_{B_{i,j}} = \{0,1\}^{2b} \to \{0,1\}^{b}$  be the function of evaluating the gates of  $B_{i,j}$ . In cases where  $B_{i,j}$  has size b' < b, we must embed the values of the b' gates into the b-bit output of  $f_{B_{i,j}}$ . For  $B_{i,0}$ , we will take the gates to be the first b' bits of the output. In  $B_{i,1}$  we say each subdivision gate has index either 2k if it's the left input to gate k and 2k+1 if it's the right. For  $B_{i,j}$ , j > 1, each gate corresponds to a gate in the previous block and we preserve its location. All indices not assigned this way will be set arbitrarily to 0.

All leaves will be either copies of  $B_0$  or  $B_{i,i}$  for some i—the former will simply be evaluated, since it has no dependencies on other blocks and the latter are identically 0.

<sup>&</sup>lt;sup>6</sup>In the case of an in-degree d graph, the size of each block would be  $\frac{d-1}{d}b$  instead of  $\frac{b}{2}$ .

<sup>&</sup>lt;sup>7</sup>In the case of an in-degree d graph, we would introduce d-1 tails, splitting the up to  $d \cdot \frac{d-1}{d}b = (d-1)b$  edges among them.

#### **Analysis**

Observe that even though we added blocks, we did not increase the depth of the block graph (indeed, this construction ensures that the length of every path from block  $B_i$  to  $B_j$  for i < j is exactly j - i).<sup>8</sup>

Then if the reduction is computable in space S, this gives us that

CIRCUITEVAL 
$$\in$$
 SPACE  $\left[S + b + \frac{s}{b} \log b\right]$ 

by plugging in b = b, and  $h = t = \frac{2s}{b}$ .

It remains to analyze the space complexity of the reduction itself, and choose our parameter b. In order to compute the reduction, we need a function which outputs a binary tree, where each node is a table of  $2^{2b}$  entries of size b.

First, observe that we can trivially compute the graph structure of the block graph in logarithmic space: an edge always exists to  $B_{i,j}$  from  $B_{i-j-1}$  and from  $B_{i,j+1}$  and no other edges exist.<sup>9</sup>

Correspondingly, given a  $\leq h$ -bit string of lefts and rights specifying a particular node in the tree, we can find which block  $B_{i,j}$  it refers to by (e.g., very inefficiently), at each step, simply iterating every pair of blocks to find which two have incoming edges to the current node and picking the appropriate one.

To compute bit k of  $f_{B_{i,j}}$  for  $j \geq 1$ , observe that we can find gate  $\lfloor k/2 \rfloor$  in  $B_i$  (selecting either the left or right input depending on whether k is even or odd. Assume here that it is  $\ell$ ). Then gate  $\ell$  belongs to block  $m = \lfloor 2\ell/b \rfloor$ . If  $m \geq i-j$ , bit k is 0 since this gate comes into the tail further up the chain. If m = i - j - 1 then bit k is simply bit  $\ell$  mod  $\frac{b}{2}$  of block  $B_m$ , which is the first input to  $B_{i,j}$ . Finally, if m < i - j - 1, then bit k is simply bit k of block  $B_{i,j+1}$ , which is the second input to  $B_{i,j}$ .

In all cases, the required operation is trivially logspace computable, and identifying the case is also logspace computable since it just requires simple numerical operations on indices. Evaluating  $f_{B_i}$  given  $v_{B_{i-1}}$  and  $v_{B_{i,1}}$  is simply the size b circuit evaluation problem which is trivially b-space computable (or, by recursion,  $\sqrt{b \log b}$  space computable).

Since all the node functions are computable in space O(b), we can write down their truth tables using O(b) bits by iterating over all possible inputs and computing the functions on each one.

Similarly, the leaf functions are either trivial  $(B_{i,i})$  or simply  $B_0$ , which we can evaluate trivially in space b (or, again, by recursion in space  $\sqrt{b \log b}$ ).

Thus, using  $h + O(\log s)$  bits to keep track of which node we're on and O(b) to output the truth table or leaf value for that node, we can compute the TREEEVAL instance in space  $S = O(s/b + b + \log s)$ . Thus:

CIRCUITEVAL 
$$\in$$
 SPACE  $\left[b + \frac{s}{b} \log b\right] \stackrel{b \leftarrow \sqrt{s \log s}}{=} \text{SPACE} \left[\sqrt{s \log s}\right]$ 

<sup>&</sup>lt;sup>8</sup>This follows from, e.g., an inductive argument: either all blocks in the path lie in the tail of  $B_j$ , in which case the result is trivial, or there exists some intermediate block i < k < j such that the path passes through  $B_k$ , in which case by strong induction the length of the path is (k-i) + (j-k) = j-i.

<sup>&</sup>lt;sup>9</sup>We can also compute the structure of the subdivided graph before partition similarly easily since an edge from a gate  $k_1 \in [0, s-1]$  (global position) in block  $i = \left\lfloor \frac{dk_1}{(d-1)b} \right\rfloor$  to gate  $k_2 \in [k_1+1, s-1]$  in block  $j = \left\lfloor \frac{dk_2}{(d-1)b} \right\rfloor$  is subdivided exactly j-i-1 times and this is the only difference from the original graph. This proves the DAG partition lemma (lemma 3.3).

# 5 Acknowledgments

I would like to thank William Hoza for suggesting this research direction, helping review this article, and general guidance.

## References

- [1] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 752–760, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384316.
- [2] James Cook and Ian Mertz. Encodings and the tree evaluation problem. *Electron. Colloquium Comput. Complex.*, TR21, 2021. URL: https://eccc.weizmann.ac.il/report/2021/054/.
- [3] James Cook and Ian Mertz. Tree evaluation is in space  $O(\log n \cdot \log \log n)$ . In Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, page 1268–1278, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649664.
- [4] Lance Fortnow, Richard Lipton, Dieter van Melkebeek, and Anastasios Viglas. Timespace lower bounds for satisfiability. *J. ACM*, 52(6):835–865, November 2005. doi:10.1145/1101821.1101822.
- [5] Oded Goldreich. On the Cook-Mertz tree evaluation procedure. *Electron. Colloquium Comput. Complex.*, TR24, 2024. URL: https://api.semanticscholar.org/CorpusID:271770168.
- [6] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4):533–546, October 1966. doi:10.1145/321356.321362.
- [7] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977. doi:10.1145/322003.322015.
- [8] Nicholas Pippenger. Fast simulation of combinational logic networks by machines without random-access storage. IBM Thomas J. Watson Research Division, 1977.
- [9] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, April 1979. doi:10.1145/322123.322138.
- [10] Heribert Vollmer. Introduction to Circuit Complexity: A Uniform Approach. Springer Publishing Company, Incorporated, 1st edition, 2010.
- [11] Ryan Williams. Simulating time with square-root space. arXiv preprint, 02 2025. doi:10.48550/arXiv.2502.17779.