

Unconditional Separation between LL and CLL

Shubham Bhardwaj*

11 October 2026

Abstract

Catalytic computation involves the study of Turing machines that are provided an additional catalytic tape (with arbitrary content) that the machine may use while ensuring that the tape contents are restored at the end of the computation. Although it is known that catalytic log-space machines are at least as powerful as non-deterministic log-space (and more), we do not have unconditional asymptotic separations.

In this work, we prove an unconditional separation between a deterministic space complexity class and its catalytic counterpart. More precisely, at the doubly logarithmic scale, we show that catalytic Turing machines can provably compute more languages than standard deterministic doubly-logarithmic space machines. The separation is witnessed by the unary language $\text{POW2} = \{1^n : n \text{ is a power of two}\}$. We construct a catalytic Turing machine that uses $O(\log \log n)$ bits of space and $O(\log n)$ bits of catalytic tape to decide POW2 , whereas any standard deterministic Turing machine requires $\Omega(\log n)$ space to decide it.

1 Introduction

Catalytic computation is a model of computation in which a Turing machine is provided with an additional *catalytic tape* that is initially in an arbitrary configuration. The machine may read from and write to the catalytic tape during its computation, but it must restore the tape to its original configuration before halting. This model was introduced by Buhrman, Cleve, Koucký, Loff, and Speelman [BCK⁺14]. We write CL for the class of languages decided by catalytic machines with $O(\log n)$ ordinary work space and polynomial catalytic space, and CLP for its polynomial-time restriction.

Buhrman et al. [BCK⁺14] also gave the surprising result that evaluating a logarithmic-depth threshold circuit can be done in catalytic log-space, that is TC^1 is contained in CL. Since TC^1 contains both NL and BPL, this gives $\text{NL} \subseteq \text{CL}$ and $\text{BPL} \subseteq \text{CL}$, or alternatively, that directed s - t reachability and estimation of random walks (complete problems for NL and BPL, respectively) can be solved in catalytic log-space.

*Tata Institute of Fundamental Research, Mumbai, India. Email: shubham.bhardwaj@tifr.res.in.

In particular, the conjecture $L \neq NL$ would imply the strict containment $L \subsetneq CL$. This gave a strong indication that catalytic space may strictly provide more power than ordinary space bounded computation. Thus, they gave strong, but conditional, evidence that catalytic log-space is more powerful than standard log-space. On the other hand, they also showed that $CL \subseteq ZPP$.

Recent work has also made progress to understand the connection between catalytic, non-deterministic, and randomized computation. Cook, Li, Mertz, and Pyne [CLMP25] proved the unconditional derandomization $CBPL = CL$, where $CBPL$ is the bounded-error randomized analogue of CL . On the upper bound side, they also showed that $CL \subseteq LOSSY$, improving the previous upper bound $CL \subseteq ZPP$. Koucký, Mertz, Pyne, and Sami [KMPS25] subsequently showed that nondeterminism also adds no power to catalytic log-space, proving $CNL = CL$. Consequently, the deterministic, nondeterministic, and bounded-error randomized versions of catalytic log-space collapse:

$$CNL = CBPL = CL.$$

These results show that catalytic space is the ultimate resource in the log-space regime, subsuming both nondeterminism and bounded-error randomness, and adding nondeterminism or bounded-error randomness to catalytic log-space does not yield any additional computational power.

There has also been substantial progress in placing natural problems in catalytic log-space. Agarwala and Mertz [AM25] showed that bipartite maximum matching can be solved in CL . More recently, general maximum matching was placed in CL by Chakraborty, Datta, Kusre, Mukhopadhyay, and Sinhababu [CDK⁺26]. Agarwala, Alekseev, and Vinciguerra [AAV26] proved the more general result that linear matroid intersection is in CL . These results show that CL contains central combinatorial problems that are not known to be in L , giving further evidence that catalytic log-space is more powerful than standard log-space.

Insights and techniques from catalytic computation also influenced the design of ordinary space-efficient algorithms. Cook and Mertz [CM20, CM24] crucially used the ideas from catalytic computation in their work on the Tree Evaluation problem, applying ideas from catalytic space even outside the catalytic model. This line of work culminated in their $O(\log n \log \log n)$ -space algorithm for Tree Evaluation [CM24], bringing this central candidate for separating L from P close to logarithmic space.

The Cook–Mertz Tree Evaluation procedure was subsequently used by Williams [Wil25] to obtain a breakthrough time–space simulation. By reducing the computation of a time-bounded multitape Turing machine to a sequence of implicitly defined Tree Evaluation instances, Williams proved that, for every $t(n) \geq n$,

$$\text{TIME}(t(n)) \subseteq \text{SPACE}\left(O\left(\sqrt{t(n) \log t(n)}\right)\right).$$

This substantially improves the previously known general simulation of time t in space $O(t/\log t)$.

Despite these advances, unconditional separations between catalytic and ordinary space bounded classes are not known at any level. The containments of NL and BPL in CL give a separation only under unproved standard complexity assumptions, while the matching and matroid-intersection results are not accompanied by log-space lower bounds. In this work, we give an unconditional separation between a deterministic space class and its catalytic counterpart. We obtain this separation at the doubly logarithmic scale, witnessed by the unary language POW2.

Theorem 1.1 (Main theorem). *There is a deterministic catalytic Turing machine that decides POW2 using $O(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space. On the other hand, no standard deterministic Turing machine decides POW2 using $o(\log n)$ work space. Consequently,*

$$\text{POW2} \in \text{CLL} \setminus \text{LL} \quad \text{and} \quad \text{LL} \subsetneq \text{CLL}.$$

The theorem follows from [Theorem 3.1](#) and [Theorem 4.3](#), which are proved in [Section 3](#) and [Section 4](#), respectively. The upper bound is obtained by constructing a catalytic Turing machine for POW2 that uses $O(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space. The lower bound is obtained by applying a theorem of Hartmanis and Berman [[HB76](#)] that shows that every infinite unary language that can be decided by a deterministic Turing machine using $o(\log n)$ work space contains an infinite regular sublanguage. We then show that POW2 contains no infinite regular sublanguage, which implies the lower bound.

2 Preliminaries

Throughout the paper, all logarithms are to base 2. We write

$$\text{POW2} = \{1^n : n = 2^k \text{ for some } k \geq 0\}.$$

We write REG for the class of regular languages. Recall that a unary language $R \subseteq \{1\}^*$ is regular if and only if its set of accepted lengths is eventually periodic. Equivalently, there exist integers $N \geq 0$ and $p \geq 1$ such that, for every $m \geq N$,

$$1^m \in R \iff 1^{m+p} \in R.$$

We use the standard off-line Turing machine model. A deterministic Turing machine has a two-way read-only input tape and a two-way read-write work tape. On input $x \in \{0,1\}^n$, the input is placed between two endmarkers, which the input head cannot cross. The work tape is initially blank. The work space of a computation is the number of distinct work-tape cells visited during the computation. All machines in this paper halt on every input.

Definition 2.1. For a function $s: \mathbb{N} \rightarrow \mathbb{N}$, let $\text{DSPACE}(s)$ denote the class of languages decided by deterministic Turing machines using $O(s(n))$ work space on inputs of length n . At the doubly logarithmic scale, we define

$$\text{LL} = \text{DSPACE}(\log \log n).$$

◇

Definition 2.2 (Buhrman et al. [BCK⁺14]). A deterministic catalytic Turing machine has, in addition to the input and ordinary work tapes, a two-way read-write catalytic tape that consists of $c(n)$ cells and is initially in an arbitrary configuration

$$\tau \in \{0, 1\}^{c(n)}.$$

The ordinary work tape remains initially blank and unmarked, exactly as in the standard model above. A catalytic machine must satisfy the following two conditions for every input x and every initial catalytic configuration τ :

1. its accept or reject decision depends only on x and not on τ ; and
2. when the machine halts, the catalytic tape again contains exactly τ .

Thus, the machine may read and modify the catalytic tape during its computation, but it must restore all of its initial contents before halting.

We say that such a machine uses ordinary work space $s(n)$ and catalytic space $c(n)$ if it visits at most $O(s(n))$ cells of the ordinary work tape and has a catalytic tape of length $O(c(n))$. We denote the corresponding class of languages by $\text{CSPACE}(s, c)$.

The catalytic analogue of LL considered in this paper is

$$\text{CLL} = \text{CSPACE}(\log \log n, \log n).$$

◇

Remark. It only makes sense to consider catalytic tapes of length at most $2^{O(s(n))}$, i.e., $c(n) \leq 2^{O(s(n))}$, since specifying an arbitrary position on a catalytic tape of length $c(n)$ requires $\log c(n)$ bits, which fits within the ordinary work space precisely in this regime. ◇

Notice that $\text{LL} \subseteq \text{CLL}$, since a catalytic machine may simply ignore its catalytic tape.

3 A Catalytic Algorithm for POW2

We first show that $\text{POW2} \in \text{CLL}$ by constructing a catalytic Turing machine that decides POW2 using $O(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space. The machine uses the catalytic

tape as a counter to store an arbitrary value L and performs arithmetic modulo 2^C , where $C = O(\log n)$ is the length of the catalytic tape. The machine first computes the largest power of two dividing n and then checks whether n is equal to that power of two.

Theorem 3.1. *There is a deterministic catalytic Turing machine that decides*

$$\text{POW2} = \{1^n : n \text{ is a power of two}\}$$

using $O(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space. Moreover, for every input and every initial content of the catalytic tape, the machine restores every catalytic cell that it visits before halting.

Proof. Let M be a deterministic catalytic Turing machine with a catalytic tape of length $C = 10 \log n$ that uses at most $W = 100 \log \log n$ ordinary work-tape cells.

We think of the catalytic tape as a binary counter holding some arbitrary value L and let $\text{enc}(L)$ be the encoding of this value that the catalytic tape holds. Let Inc be the routine that sweeps the input tape and increments the catalytic counter by 1 for each input symbol it encounters, and similarly, let Dec be the routine that sweeps the input tape and decrements the catalytic counter by 1 for each input symbol it encounters. Clearly, Inc and Dec are inverses of each other, and a single sweep of Inc followed by a single sweep of Dec restores the catalytic tape to its original content.

The machine has 2 phases. In the first phase, it finds the largest v such that 2^v divides n . We index the catalytic bits starting from 0, with bit 0 being the least significant bit. To find v , the machine starts with $b = 0$ and keeps increasing b until the b -th bit detects a change. In each iteration, the machine stores the current value of the b -th bit of the catalytic tape. It then performs Inc and checks the value of the b -th bit again. It then performs Dec to restore the catalytic tape. If the two values are different, then it sets $v = b$ and stops the first phase; otherwise it increases b by 1 and continues.

Algorithm 1: Phase 1: finding the largest power of two dividing n

Data: Input 1^n , catalytic counter initially encoding an arbitrary value L

Result: A work-tape value $v = \nu_2(n)$; the catalytic tape is restored

```
1  $b \leftarrow 0$ 
2 while true do
3    $x \leftarrow$  the  $b$ -th bit of the catalytic tape
4   Perform Inc  $\triangleright$  the counter changes from  $L$  to  $L + n$ 
5    $y \leftarrow$  the  $b$ -th bit of the catalytic tape
6   Perform Dec  $\triangleright$  the counter is restored to  $L$ 
7   if  $x \neq y$  then
8      $v \leftarrow b$ 
9     return  $v$ 
10   $b \leftarrow b + 1$ 
```

After the first phase, the machine has computed $v = \nu_2(n)$, the largest power of two dividing n . In the second phase, it checks whether $n = 2^v$. To do this, it compares the catalytic counter after adding n with the catalytic counter after adding 2^v . If the two counters are equal, then $n = 2^v$ and the machine accepts; otherwise it rejects. The second phase is described in [Algorithm 2](#). Let $\text{AddPow}_C(v)$ be the routine that adds 2^v to the catalytic counter modulo 2^C . This can be done by going to the v -th bit and adding 1 and propagating carries if any. Similarly, let $\text{SubPow}_C(v)$ be the routine that subtracts 2^v from the catalytic counter modulo 2^C .

In the second phase, for each bit position, the machine first runs Inc to add n to the catalytic counter, then remembers the value of the bit, and then runs Dec to restore the catalytic counter and then runs $\text{AddPow}_C(v)$ to add 2^v to the catalytic counter, then remembers the value of the bit, and then runs $\text{SubPow}_C(v)$ to restore the catalytic counter. If the two remembered values of the bit are different, then it rejects; otherwise it continues to the next bit position. If all bits agree, then it accepts.

Algorithm 2: Phase 2: checking whether $n = 2^v$

Data: Input 1^n , work-tape value $v = v_2(n)$, catalytic counter encoding L

Result: Accept iff $n = 2^v$; the catalytic tape is restored

```
1 for  $j = 0, 1, \dots, C - 1$  do
2   Perform Inc                                ▷ the counter changes from  $L$  to  $L + n$ 
3    $x \leftarrow$  the  $j$ -th bit of the catalytic tape
4   Perform Dec                                ▷ the counter is restored to  $L$ 
5   Perform AddPowC( $v$ )                        ▷ the counter changes from  $L$  to  $L + 2^v$ 
6    $y \leftarrow$  the  $j$ -th bit of the catalytic tape
7   Perform SubPowC( $v$ )                        ▷ the counter is restored to  $L$ 
8   if  $x \neq y$  then
9     return reject
10 return accept
```

We now prove correctness. We first analyze Phase 1. The basic observation is the following. For any $r \leq C$, the first r bits of the catalytic counter encode the value of the full counter modulo 2^r . Thus, after one execution of Inc, these first r bits change from

$$L \bmod 2^r \quad \text{to} \quad (L + n) \bmod 2^r.$$

In particular, if 2^r divides n , then the first r bits do not change.

Now consider the iteration of Phase 1 with index b . If $b < v_2(n)$, then 2^{b+1} divides n . By the observation with $r = b + 1$, bits $0, 1, \dots, b$ are unchanged after Inc, and hence bit b is unchanged. Therefore Phase 1 cannot stop before $b = v_2(n)$. On the other hand, when $b = v_2(n)$, we have

$$n \equiv 2^b \pmod{2^{b+1}}.$$

Adding n therefore flips bit b . Thus, we terminate with $b = v_2(n)$.

We next analyze Phase 2. At the beginning of each iteration over a bit position j , the catalytic counter again has its original value L . The first comparison value is bit j of

$$L + n \pmod{2^C},$$

and the second comparison value is bit j of

$$L + 2^v \pmod{2^C}.$$

Thus Phase 2 accepts exactly when these two C -bit strings are equal. This is equivalent to

$$L + n \equiv L + 2^v \pmod{2^C},$$

or, after cancelling L , to

$$n \equiv 2^v \pmod{2^C}.$$

Since $C = 10 \log n$, both n and 2^v are smaller than 2^C , so the above congruence holds if and only if $n = 2^v$. Since Phase 1 has computed $v = v_2(n)$, this happens exactly when n is a power of two. Therefore the machine accepts exactly the inputs in POW2.

Finally, restoration is built into the two phases. Every execution of Inc is followed by the matching execution of Dec before the machine uses the result of that iteration to halt or continue, and every execution of $\text{AddPow}_C(v)$ is followed by $\text{SubPow}_C(v)$. Hence every visited catalytic cell is restored for every initial catalytic content. The machine uses $C = O(\log n)$ catalytic cells.

We now show that the routines Inc, Dec, $\text{AddPow}_C(v)$, and $\text{SubPow}_C(v)$ can be implemented within $O(\log \log n)$ space. As for the input tape, we assume that the two ends of the catalytic tape are marked, so that the machine can detect when its catalytic head is on cell 0 or on cell $C - 1$. Between routine calls, the catalytic head is kept on cell 0.

Inc and Dec. To perform Inc, the machine moves its input head from the left endmarker to the right endmarker. Each time it reads a 1, it keeps the input head in place and adds 1 to the catalytic counter by a carry forward increment: starting at cell 0, it replaces each 1 by a 0 and moves right, until it reads a 0, which it replaces by a 1. If it replaces the 1 in cell $C - 1$ by a 0, the carry out of the last cell is discarded; this is exactly the reduction modulo 2^C . The catalytic head then returns to cell 0, and the input sweep continues. The carry and the progress of the increment are kept in the finite control, so Inc uses only $O(1)$ work-tape cells. The routine Dec is identical with the roles of 0 and 1 exchanged, so that a borrow propagates through the 0's, and it also uses $O(1)$ work-tape cells.

AddPow_C(v) and SubPow_C(v). The machine moves the catalytic head from cell 0 to cell v , counting its moves on a binary counter on the work tape, and then performs the carry forward increment described above starting from cell v instead of cell 0. This adds 2^v to the counter modulo 2^C . The catalytic head then returns to cell 0. The routine $\text{SubPow}_C(v)$ can also be implemented in a similar way. The binary counter never exceeds v , so both routines use $O(\log C)$ work-tape cells.

Reading a catalytic bit. To read bit b (in Phase 1) or bit j (in Phase 2), the machine moves the catalytic head to that cell using a binary counter in the same way, copies the bit to the work tape, and returns the catalytic head to cell 0. This uses $O(\log C)$ work-tape cells.

Altogether, the work tape holds the values b , v , and j , one binary counter for the position of the catalytic head, and the two bits x and y . Since $b \leq v_2(n) \leq \log n < C$, $v = v_2(n) < C$, and $j < C$, each of these values is at most $C = 10 \log n$ and is stored in $O(\log C) = O(\log \log n)$ cells. Hence the ordinary work space is $O(\log \log n)$. $\square$

4 A Lower Bound for POW2

We now show the space lower bound for POW2. We will use the following theorem of Hartmanis and Berman.

Theorem 4.1 (Hartmanis and Berman [HB76]). *Every infinite unary language that can be decided by a deterministic Turing machine using $o(\log n)$ work space contains an infinite regular sublanguage.*

We now show that POW2 contains no infinite regular sublanguage.

Lemma 4.2. *The language POW2 contains no infinite regular language.*

Proof. Suppose that there is an infinite regular language $R \subseteq \text{POW2}$. Since R is unary and regular, membership in R is eventually periodic. Thus, there are integers $N \geq 0$ and $p \geq 1$ such that, for every $m \geq N$,

$$1^m \in R \iff 1^{m+p} \in R.$$

Since R is infinite, we can choose $m \geq \max\{N, p+1\}$ for which $1^m \in R$. The containment $R \subseteq \text{POW2}$ implies that $m = 2^k$ for some k . Periodicity implies that $1^{m+p} \in R$, so $m+p$ must also be a power of two. On the other hand, $p < m$, and therefore

$$m < m+p < 2m.$$

There is no power of two strictly between the consecutive powers of two m and $2m$, which gives a contradiction. $\square$

From Theorem 4.1 and Lemma 4.2, we immediately obtain the following lower bound.

Theorem 4.3. *No deterministic Turing machine with an initially blank and unmarked work tape can decide POW2 using $o(\log n)$ space. Consequently,*

$$\text{POW2} \notin \text{LL}.$$

5 Conclusion

We have given an unconditional separation between a deterministic space class and its catalytic counterpart. The separation is witnessed by the unary language POW2. We constructed a catalytic

Turing machine that decides POW2 using $O(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space. On the other hand, the theorem of Hartmanis and Berman, together with the fact that POW2 has no infinite regular sublanguage, shows that no standard deterministic Turing machine decides POW2 using $o(\log n)$ work space. Consequently,

$$\text{LL} \subsetneq \text{CLL}.$$

Thus, catalytic space provides a genuine increase in computational power when the ordinary work tape has only doubly logarithmic size.

There is also a natural question below the doubly logarithmic threshold. In the ordinary space model, every language recognized using $o(\log \log n)$ work space is regular [SHI65]. It is therefore interesting to ask whether catalytic space can make useful computation possible below this classical threshold: does there exist a nonregular language that can be decided by a catalytic Turing machine using $o(\log \log n)$ ordinary work space and $O(\log n)$ catalytic space?

Another direction is to understand the role of nondeterminism at the doubly logarithmic scale. The containment $\text{NL} \subseteq \text{CL}$ shows that at logarithmic space, catalytic memory can simulate nondeterministic log-space. The corresponding question for the classes considered in this paper is open. Let NLL denote the class of languages decided by nondeterministic Turing machines using $O(\log \log n)$ work space. It is natural to ask whether

$$\text{NLL} \stackrel{?}{\subseteq} \text{CLL}.$$

The most prominent open question remains the analogous separation at the logarithmic scale. In particular, proving

$$\text{L} \subsetneq \text{CL}$$

unconditionally remains a major open problem. Even conditional separations or collapses based on weaker hardness assumptions would be of interest.

References

- [AAV26] Aryan Agarwala, Yaroslav Alekseev, and Antoine Vinciguerra. **Linear Matroid Intersection Is in Catalytic Logspace**. In *Proceedings of the 17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*, volume 362 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026.

- [AM25] Aryan Agarwala and Ian Mertz. **Bipartite Matching Is in Catalytic Logspace**. In *Proceedings of the 66th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2025)*, pages 360–372. IEEE, 2025.
- [BCK⁺14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. **Computing with a Full Memory: Catalytic Space**. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 857–866. ACM, 2014.
- [CDK⁺26] Srijan Chakraborty, Samir Datta, Aryan Kusre, Partha Mukhopadhyay, and Amit Sinhababu. **Maximum Matching and Related Problems in Catalytic Logspace**, 2026. Pre-print available at [arXiv:2604.24275](https://arxiv.org/abs/2604.24275).
- [CLMP25] James Cook, Jiayu Li, Ian Mertz, and Edward Pyne. **The Structure of Catalytic Space: Capturing Randomness and Time via Compression**. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025)*, pages 554–564. ACM, 2025.
- [CM20] James Cook and Ian Mertz. **Catalytic Approaches to the Tree Evaluation Problem**. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, pages 752–760. ACM, 2020.
- [CM24] 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)*, pages 1268–1278. ACM, 2024.
- [HB76] Juris Hartmanis and Leonard Berman. **On Tape Bounds for Single Letter Alphabet Language Processing**. *Theoretical Computer Science*, 3(2):213–224, 1976. An earlier version appears as Cornell University Technical Report TR 75-242, where the result used here is Lemma 6.
- [KMPS25] Michal Koucký, Ian Mertz, Edward Pyne, and Sasha Sami. **Collapsing Catalytic Classes**. In *Proceedings of the 66th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2025)*, pages 455–463. IEEE, 2025.
- [SHI65] Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II. **Hierarchies of Memory Limited Computations**. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 179–190. IEEE, 1965.
- [Wil25] R. Ryan Williams. **Simulating Time with Square-Root Space**. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025)*, pages 13–23. ACM, 2025.