






# Randomness Extractors in $AC^0$ and $NC^1$ : Optimal up to Constant Factors

Kuan Cheng   

CFCS, School of CS, Peking University, China

Ruiyang Wu  

CFCS, School of CS, Peking University, China

---

## Abstract

We study randomness extractors in  $AC^0$  and  $NC^1$ . For the  $AC^0$  setting, we give a logspace-uniform construction such that for every  $k \geq n/\text{poly log } n, \varepsilon \geq 2^{-\text{poly log } n}$ , it can extract from an arbitrary  $(n, k)$  source, with a small constant fraction entropy loss, and the seed length is  $O(\log \frac{n}{\varepsilon})$ . The seed length and output length are optimal up to constant factors matching the parameters of the best polynomial time construction such as [13]. The range of  $k$  and  $\varepsilon$  almost meets the lower bound in [10] and [7]. We also generalize the main lower bound of [10] for extractors in  $AC^0$ , showing that when  $k < n/\text{poly log } n$ , even strong dispersers do not exist in non-uniform  $AC^0$ . For the  $NC^1$  setting, we also give a logspace-uniform extractor construction with seed length  $O(\log \frac{n}{\varepsilon})$  and a small constant fraction entropy loss in the output. It works for every  $k \geq O(\log^2 n), \varepsilon \geq 2^{-O(\sqrt{k})}$ .

Our main techniques include a new error reduction process and a new output stretch process, based on low-depth circuit implementations for mergers, condensers, and somewhere extractors.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Expander graphs and randomness extractors; Theory of computation  $\rightarrow$  Pseudorandomness and derandomization

**Keywords and phrases** randomness extractor, uniform  $AC^0$ , error reduction, uniform  $NC^1$ , disperser

**Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2024.69

**Category** RANDOM

**Related Version** Full Version: <https://ecc.weizmann.ac.il/report/2024/040/>

## 1 Introduction

Randomness extractors are functions that can transform weak random sources into distributions close to uniform. A typical definition of weak random sources is by min-entropy. A random variable (weak source)  $X$  has min-entropy  $k$  if for every  $x$  in the support of  $X$ ,  $\log \frac{1}{\Pr[X=x]} \geq k$ . To extract from an arbitrary weak source of a certain min-entropy, Nisan and Zuckerman [23] introduced the definition of seeded extractor, where the extractor has a short uniform random seed as an extra input. Specifically, a function  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  is defined to be a strong  $(k, \varepsilon)$ -extractor, if for every source  $X$  with min-entropy  $k$ ,

$$\|(U_d, \text{EXT}(X, U_d)) - U_{d+m}\| \leq \varepsilon,$$

where  $U_d$  and  $U_m$  are uniform distributions over  $\{0, 1\}^d$  and  $\{0, 1\}^m$  respectively, and  $\|\cdot\|$  is the statistical distance. The entropy loss of such a strong extractor is  $k - m$ . On the contrary, a weak  $(k, \varepsilon)$ -extractor has the same definition except we only require

$$\|\text{EXT}(X, U_d) - U_m\| \leq \varepsilon.$$

The entropy loss of such a weak extractor is  $k + d - m$ .



© Kuan Cheng and Ruiyang Wu;

licensed under Creative Commons License CC-BY 4.0

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024).

Editors: Amit Kumar and Noga Ron-Zewi; Article No. 69; pp. 69:1–69:22



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As a fundamental pseudorandom construction, extractors are closely related to other pseudorandom objects and also have various applications in computational complexity, combinatorics, algorithm design, information theory, and cryptography. See surveys [21, 29, 37, 30, 1, 38].

Optimizing extractor constructions aims to get, for every  $k$  and  $\varepsilon$ , an extractor with  $d$  as small as possible, and  $m$  as large as possible. An existential bound for strong extractors can be given by a probabilistic argument, which has  $d = \log(n - k) + 2\log(1/\varepsilon) + O(1)$ ,  $m = k - 2\log(1/\varepsilon) - O(1)$ . This is optimal up to some additive constants for  $k \leq n/2$ , due to the lower bound by [24]. After [23], a long line of work has been done to seek explicit extractors with parameters close to the existential bounds [40, 31, 11, 32, 41, 25, 21, 27, 36, 33, 26, 20, 13, 34, 9, 19]. Among them, [13] first achieves  $d = \log n + O(\log(k/\varepsilon))$  and an arbitrary constant factor entropy loss, and also achieves  $m = k - 2\log(1/\varepsilon) - O(1)$  with  $d = \log n + O(\log k \cdot \log(k/\varepsilon))$ . [34] and [19] can also achieve the same parameters by replacing the condenser in [13] with their condenser versions. On the other hand, [34] and [9] achieve subconstant entropy loss  $m = (1 - 1/\text{poly log } n)k$ ,  $d = O(\log n)$  when  $\varepsilon \geq 1/2^{\log^\beta n}$  for any constant  $\beta < 1$ .

In terms of computational complexity, an explicit construction is an algorithm that can compute the function in deterministic polynomial time on given parameters. A natural question is whether one can construct extractors in lower complexity classes, with matching parameters to the current best explicit ones. Some early work on extractors already pays attention to constructions in low-complexity models. For example, Zuckerman [41] showed that his construction is actually in  $NC$ . Also Bar-Yossef, Reingold, Shaltiel, and Vadhan [2] showed streaming constructions for several pseudorandom objects including extractors. Furthermore, extractors in low-complexity models have already been used in derandomization tasks for certain low-complexity classes, such as in [35, 8]. In this paper, we specifically focus on two low-complexity classes, i.e.  $AC^0$  and  $NC^1$ .  $AC^0$  is the class of all uniform circuit families of polynomial-size, constant depth, with NOT, AND, and OR gates, where AND and OR gates have unbounded fan-in.  $NC^1$  is the class of all uniform circuit families of polynomial-size,  $O(\log n)$  depth, with NOT, AND, and OR gates, where AND, OR gates have fan-in 2. Unless otherwise specified, our constructions are all logspace-uniform circuit families, i.e. there exists a logspace Turing machine that can output the description for each circuit in the family.

Viola [39] raised the question on extractor construction in  $AC^0$  and showed that for every constant  $D$ , there exists a polynomial  $p$  such that as long as  $k \leq n/p(\log n)$ , no extractor in  $AC^0$  with depth  $D$  extract even 1 bit with a constant error, no matter how long the seed is. Goldreich and Wigderson [10] extend the result for bit-fixing sources. This rules out the possibility for the case that  $k = n/\log^{\omega(1)} n$ . For the case  $k \geq n/\text{poly log } n$ , [10] gives a strong extractor in  $AC^0$  that has an output length linear to the seed length. Lately Cheng and Li [7] give a construction that significantly improves the parameters. For the case that  $\varepsilon = 1/\text{poly } n$ ,  $\delta = 1/\text{poly log } n$ , they achieve  $d = O(\log n)$ ,  $m = O(\delta n)$ . For the more general case that  $\varepsilon = 2^{-\text{poly log } n}$ ,  $\delta = 1/\text{poly log } n$ , they achieve  $d = O\left(\log n + \frac{\log(n/\varepsilon)\log(1/\varepsilon)}{\log n}\right)$ ,  $m = O(\delta n)$ . They also show that  $\varepsilon$  has to be at least  $2^{-\text{poly log } n}$  for  $AC^0$  extractors.

For extractors in  $NC^1$ , unlike the  $AC^0$  case, there are no known lower bounds for  $k$  or  $\varepsilon$ . Indeed the extractor based on universal hash functions [5], argued by the leftover hash lemma [16], can achieve an arbitrary  $\varepsilon$  and  $k$ . It can be realized in  $NC^1$  since there are simple linear function constructions for such hash functions. Trevisan's extractor [36], and its improved version [26] can also be realized in  $NC^1$ , since their main components, the average-case hard function based on local list-decodable codes can be computed in  $NC^1$ .

Extractors can also be derived from averaging samplers [41]. Healy [15] constructs a sampler in  $\text{NC}^1$ . However if one simply applies the transformation of [41] on it, then this can only give an extractor with a constant error. So it is still a question whether one can achieve extractors in  $\text{NC}^1$  with better parameters for arbitrary  $k$  and  $\varepsilon$ .

## 1.1 Our results

Our main positive result is an  $\text{AC}^0$  computable extractor with parameters optimal up to constant factors.

► **Theorem 1.** *For every constant  $a, c > 0, \gamma \in (0, 1)$ , every  $k \geq \frac{n}{\log^a(n)}, \varepsilon \geq 2^{-\log^c(n)}$ , there exists an explicit  $(k, \varepsilon)$ -strong extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  in  $\text{AC}^0$  with depth  $O((a + c + 1)^2)$ , such that  $d = O(\log \frac{n}{\varepsilon})$ , and  $m \geq (1 - \gamma)k$ .*

Notice that this is much better in seed length compared to the previous best  $\text{AC}^0$  constructions [7], which requires  $d = O\left(\left(\log n + \frac{\log(n/\varepsilon)\log(1/\varepsilon)}{\log n}\right) \log^a n\right)$  for such an output length. Also, notice that there are lower bounds for  $k$  and  $\varepsilon$  in the  $\text{AC}^0$  construction setting, i.e.  $k$  has to be at least  $n/\text{poly} \log n$  by [10] and  $\varepsilon$  has to be  $2^{-\text{poly} \log n}$  by [7]. Thus roughly in the plausible range for  $k$  and  $\varepsilon$ , we achieve parameters optimal up to constant factors.

Our method can also be used to give  $\text{NC}^1$  computable extractors.

► **Theorem 2.** *For every constant  $\gamma \in (0, 1)$  every  $k \geq \Omega(\log^2(n)), \varepsilon \geq 2^{-O(\sqrt{k})}$ , there exists a strong  $(k, \varepsilon)$  extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  computable in  $\text{NC}^1$ , with  $d = O(\log(n/\varepsilon))$ ,  $m = (1 - \gamma)k$ .*

To our knowledge, the previous best known  $\text{NC}^1$  construction is the improved Trevisan's extractor from [26], which has seed length  $O(\log^2 n \log \frac{1}{\varepsilon})$ , for all  $k, \varepsilon$ . Our parameters are optimal up to constant factors for ranges of  $k, \varepsilon$  as stated.

Our negative result generalizes the previous entropy parameter lower bound by [10] for strong extractors in  $\text{AC}^0$  to strong dispersers in  $\text{AC}^0$ .

► **Theorem 3.** *For every  $d, s > 0$ , every constant  $\delta \in (0, 1)$ , if  $C : \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}$  is a  $(k, \frac{1}{2} - \delta)$ -disperser that can be computed by a non-uniform  $\text{AC}$  circuit of size  $s$  and depth  $d$ , then  $k \geq \Theta\left(\frac{\delta n}{\log^{d-1} s}\right)$ .*

## 1.2 Technique Overview

### 1.2.1 Extractor in $\text{AC}^0$

Our  $\text{AC}^0$  computable extractor is constructed by three main parts.

#### 1.2.1.1 Merger in $\text{AC}^0$

In this part, we show that any somewhere high-entropy source  $X$  can be merged to be a high-entropy source in  $\text{AC}^0$  under a restricted setting of parameters. The merger is a crucial building block in the construction of our extractor.

Recall that  $X = (X_1, \dots, X_\Lambda)$  is a simple somewhere  $(n, k)$  source if there exists  $i \in [\Lambda]$ ,  $X_i$  is a  $(n, k)$  source. We call each  $X_i$  a segment. A somewhere  $(n, k)$  source is a convex combination of simple somewhere  $(n, k)$  sources. A  $(k, k', \varepsilon)$  merger is a function  $\text{Merge} : \{0, 1\}^{n\Lambda} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ , such that for any input somewhere  $(n, k)$  source  $X$ ,  $\text{Merge}(X, U)$  has entropy  $k'$ . [9] gives a fairly good merger for somewhere uniform sources, which has  $m = n = k, k' = (1 - \delta)k, d = \frac{1}{\delta}(\log \frac{2\Lambda}{\varepsilon})$ . Our key observation is that if the

number of segments in the somewhere uniform source is  $\text{poly log } n$ ,  $\delta$  is a small constant, and error  $\varepsilon = 2^{-\text{poly log } n}$ , then this merger can be computed in  $AC^0$ . To see this, note that the computation of [9] is over a finite field  $F_q$ , where  $q = 2^d = 2^{\text{poly log } n}$  in this setting. The computation only involves three operations: (1) the summation of  $\text{poly log } n$  elements; (2) the powering  $y^i$  where  $y \in F_q, i = \text{poly log } n$ ; (3) the product of a constant number of field elements. (1) is clearly in  $AC^0$  since it is actually the summation of  $\text{poly log } n$  bits, while (2) and (3) are shown to be in  $AC^0$  by [14]. Note that this can be straightforwardly generalized to a merger for somewhere high-entropy source by first applying an extractor to each segment and then merging them.

### 1.2.1.2 Error Reduction

In this part, we give a new error reduction that can be realized in a highly parallel way. The required seed length is optimal up to constant factors, significantly better than [7]. Our method takes the basic extractor from [7], applies error reduction and stretches the output length to  $\text{poly}(\log n)$  bits. The stretching is designed to satisfy the requirement in the next part.

Let  $X$  be an input  $(n, k)$ -source with  $k = n/\log^a n$  for some constant  $a$ . We start from an  $AC^0$  computable  $(k, \varepsilon_0)$  extractor  $\text{EXT}_0 : \{0, 1\}^n \times \{0, 1\}^{d_0} \rightarrow \{0, 1\}^{m_0}$  where  $\varepsilon_0 = 1/n$ ,  $d_0 = O(\log n)$ ,  $m_0 = O(k^2/n)$ , which is achieved in [7]. Then for every given constant  $c$ , the new error reduction can reduce the error to be as small as  $\varepsilon = 2^{-\log^c(n)}$ , with a seed length  $O(\log \frac{n}{\varepsilon})$ . We briefly describe the main steps of the procedure along with their arguments.

1. Apply  $\text{EXT}_0$  to  $X$  for  $t = \frac{\log(n/\varepsilon)}{\log n}$  times in parallel, using independent seeds, outputting  $Y_1, Y_2, \dots, Y_t$  respectively, each of length  $m_0$ .  
Notice that by the error reduction of [25], one can show that with probability at least  $1 - \varepsilon' \geq 1 - O(\varepsilon_0)^t$ , there exists  $i$  such that  $Y_i$  has min-entropy at least  $m_0 - O(\log t)$ , while the seed length used here is only  $td_0 = O(\log(n/\varepsilon))$ . Hence one can deduce that  $(Y_1, \dots, Y_t)$  is  $t\varepsilon'$  close to a somewhere  $(m_0, m_0 - O(\log t))$  source. We stress that this step is also the first step in the error reduction of [7]. But we differ from [7] after then.
2. For each  $i$ , cut  $Y_i$  into  $l = O(\log n)$  blocks such that their lengths form a geometric sequence. That is  $Y_i = (Y_{i,1}, Y_{i,2}, \dots, Y_{i,l})$ , where we let  $m_j = |Y_{i,j}| = m_0^{0.1} \cdot 3^j$ . Denote  $Y_{i,1\dots j}$  as the first  $j$  blocks of  $Y_i$ . Then for each  $j$ , let  $B_j = (Y_{1,1\dots j}, Y_{2,1\dots j}, \dots, Y_{t,1\dots j})$ , i.e. the  $i$ -th segment of  $B_j$  is the first  $j$  blocks from  $Y_i$ . Regard  $B_j$  as a somewhere high-entropy source and merge it by the merger from the previous part, attaining  $Z_j$ . Here we use the same seed for each  $j$ . Then we regard  $(Z_1, Z_2, \dots, Z_l)$  as a block source and extract in a standard way by using an extractor  $\text{EXT}_1$ . Here  $\text{EXT}_1$  is constructed by first sampling  $O(\log \frac{n}{\varepsilon})$  bits from the source and then applying universal hashing.  
Notice that since the high entropy segment of  $Y$  is a  $(m_0, m_0 - O(\log t))$  source, each  $B_j$  has to be a somewhere  $(M_j, M_j - O(\log t))$  source, where  $M_j = m_1 + m_2 + \dots + m_j$ . Also, as  $t = \text{poly log } n$ , the merger can be implemented in  $AC^0$ . As a result of merging,  $Z_j$  has a high constant entropy rate. Since  $m_j, j \in [l]$  forms a geometric sequence,  $Z_j$  is a constant times longer than  $Z_{j-1}$ . Thus  $(Z_1, Z_2, \dots, Z_l)$  is indeed very close to a block source that has a constant conditional entropy rate. The output length is  $\Omega(\log n \log \frac{n}{\varepsilon})$  since for each block we can sample  $O(\log \frac{n}{\varepsilon})$  bits and then apply an extractor from the left-over hash lemma. The seed length is  $O(\log \frac{n}{\varepsilon})$  since both the merger and the sample-then-extract have a seed length  $O(\log \frac{n}{\varepsilon})$ .
3. Assume the previous steps give an extractor  $\text{EXT}'$ . To increase the output length, we run the above steps again but instead use  $\text{EXT}'$  to replace  $\text{EXT}_1$  in the second step. This can increase the output length by a  $\Omega(\log n)$  factor. We do this for  $b$  times to finally get an extractor with output length  $\Omega(\log^b n \cdot \log \frac{n}{\varepsilon})$ , for a given arbitrary constant  $b$ .

Note that in this way the circuit depth has a factor  $b$  blow-up. The seed length also has a factor  $b$  blow-up. But as  $b$  is a constant, the construction is still in  $AC^0$  and the seed length is still  $O(\log \frac{n}{\epsilon})$ .

### 1.2.1.3 Output Stretch

The last part is a new output stretch procedure for  $AC^0$  computable extractors. Compared to the one in [7], the new method attains an output length  $(1 - \gamma)k$  with a seed length  $O(\log \frac{n}{\epsilon})$ .

Observe that if the input source already has a constant entropy rate, then this is an easy case. Because one can do sampling to get a two-block source with constant conditional entropy rates. Then one can use the extractor derived from the previous part to extract from the second source, attaining a  $\text{poly} \log \frac{n}{\epsilon}$  length output, and then use it to extract the first block by applying the main extractor from [7]. However, the hard case is when the entropy rate is sub-constant i.e.  $k = \frac{n}{\log^a n}$ . The above simple strategy does not work since we don't know how to argue that the block attained from sampling can keep a constant fraction of all entropy while conditioned on this block, the source still keeps a fairly large conditional entropy. To resolve this issue, we follow a general strategy used in [9]. We describe the following 3 steps to reduce the hard case to the easy case.

1. Use Ta-shma's somewhere-block-source converter [33] to convert the original source into a somewhere-two-block-source.

Recall that Ta-shma's converter tries every position of the input source. For each position, the source is cut into two substrings. To avoid having too many segments in the resulting somewhere-two-block-source, one can pick a cutting position after, for example, every  $n / \log^{2a} n$  consecutive positions. In this way, the number of segments is  $\Lambda = \log^{2a} n$ . [33] shows that for at least one of the position choices, the cutting can give a two-block source where the first block has entropy  $\Omega(k)$ , and the second has conditional entropy  $\Omega(k)$ .

2. For each segment, apply our extractor in the error reduction part for the second block and then use the output as a seed to extract the first block by the extractor in [7].

As at least one segment of the somewhere source is indeed a two-block source, the extraction for the second block can provide an output of length  $\text{poly} \log \frac{n}{\epsilon}$ . This is enough to extract a constant fraction of entropy i.e.  $\Omega(k)$  from the first block by [7]. Then what we get is very close to a somewhere uniform source.

3. Use the merger in  $AC^0$  from the previous part to get a source with a constant entropy rate and min-entropy  $\Omega(k)$ .

As we only have  $\text{poly} \log n$  segments,  $\epsilon = 2^{-\text{poly} \log n}$ , and the entropy rate attained is a constant, it holds that the merger is in  $AC^0$ , with a seed length  $O(\log \frac{n}{\epsilon})$ . Then after merging, the hard setting is reduced to the previously discussed easy setting, i.e. the constant entropy rate case.

### 1.2.2 Extractor in $NC^1$

Our construction for extractor in  $NC^1$  can be described by the following 3 steps.

1. First apply a condenser from [19]. Regard the output as  $(Y_1, Y_2)$  such that  $Y_1, Y_2$  have a equal length.

Compared to the condenser in [13], the condenser in [19] can only work for  $k \geq \Omega(\log^2(n)), \epsilon \geq 2^{-O(\sqrt{k(n)})}$ . However, the advantage is that it is computable in  $NC^1$ . Recall that the [19]  $(k, k + d, \epsilon)$  condenser can actually be viewed as  $\text{Cond} : \mathbb{F}_q^n \times \mathbb{F}_q \rightarrow \mathbb{F}_q^m$ . It views the input source as coefficients of a degree  $n - 1$  polynomial  $f(x) = \sum_{i=0}^{n-1} a_i x^i$

over field  $\mathbb{F}_q$ ,  $\log q = O(\log \frac{n}{\epsilon})$ . The seed is a random element of  $\mathbb{F}_q$ . The computation is actually  $\text{Cond}(f, u) = (u, f(u), f^{(1)}(u), \dots, f^{(m)}(u))$ . Where  $f^{(j)}(u) = \sum_{i=0}^d \frac{i!}{(i-j)!} a_i u^{i-j}$  is the  $j$ -th derivative of  $f$ . Notice that all these coefficients  $\frac{i!}{(i-j)!}$  can be precomputed and hardwired in the circuits. The polynomial evaluation consists of three operations: (1) the powering  $x^{i-j}$ , (2) the multiplication of two  $\mathbb{F}_q$  elements, and (3) the summation of a polynomial number of elements. The powering could be implemented with two steps: powering in  $\mathbb{N}$  and then divided by  $q$ , which is computable in  $NC^1$  by [4]. The multiplication and summation are both in  $NC^1$  by straightforward realizations. So after condensing, we get a source  $(Y_1, Y_2)$  with an entropy rate  $> 3/4$ . As  $Y_1$  and  $Y_2$  have an equal length, they form a two-block source with constant conditional entropy rates.

2. For  $Y_2$ , apply the extractor from our error reduction to get  $Z$  of length  $O(\log^2 n \log(n/\epsilon))$ . This step is basically the same as the  $AC^0$  case. We make sure the error reduction can also be done in  $NC^1$  under this parameter setting, and the seed length is still  $O(\log \frac{n}{\epsilon})$ .
3. Apply the improved Trevisan's extractor [26] to  $Y_1$  using  $Z$  as the seed.

Notice that this extracts  $O(k)$  bits with a desired error. It can be further stretched to  $(1 - \gamma)k$  by a standard parallel method. Also, notice that it is a folklore that Trevisan's extractor [36] and its improved version [26] can be realized in  $NC^1$ . So our whole construction is in  $NC^1$ . The required seed length for improved Trevisan's extractor is  $O(\log^2 n \log(n/\epsilon))$ , and the output from step 2 is enough to feed it. Hence the overall seed length is  $O(\log \frac{n}{\epsilon})$ .

### 1.2.3 A lower bound for $AC^0$ computable dispersers

Our lower bound follows from the improved switching lemma in [28]. Assume  $\text{Disp} : \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}$  is a strong  $(k, \frac{1}{2} - \delta)$ -disperser computable in  $AC^0$  with depth  $d$  and size  $s$ . Notice that we only need to consider the 1 bit output setting. Consider that for a fixed seed  $y \in \{0, 1\}^r$ , we apply a random restriction on  $C_y := \text{Disp}(\cdot, y)$ . Let the random restriction be  $R_p$  over  $\{0, 1, *\}^n$  such that for every  $i \in [n]$ , independently we have  $\Pr[R_p(i) = *] = p, \Pr[R_p(i) = 0] = \Pr[R_p(i) = 1] = \frac{1-p}{2}$ . For a restriction  $\rho$  sampled from  $R_p$ , the function  $C_y|_\rho$  is defined to be a function such that if  $\rho_i$  is 1 or 0 then fix the  $i$ -th input to be  $\rho_i$ , otherwise leave it unfixed, and then apply  $C_y$  on this modified input. The switching lemma from [28] basically shows that  $\Pr_{\rho \sim R_p}[C_y|_\rho \text{ is not constant}] \leq \delta$ , if  $p = \frac{\delta}{\Theta((\log s)^{d-1})}$ . Also notice that when  $\delta$  is a constant, with probability at least  $1 - 2^{-O(pn)} > 1 - \delta$ , the number of stars in  $\rho$  is at least  $p/2$  fraction. By a union bound and an averaging argument, one can show that there exists a  $\rho$  which has at least  $pn/2$  stars such that for  $> 1 - 2\delta$  fraction of  $y$ ,  $C_y|_\rho$  is a constant. Notice that if we take this  $\rho$  for a uniform input source, then it becomes a bit-fixing source of entropy  $k \geq pn/2 = \Theta(\frac{\delta n}{(\log^{d-1} s)})$ . Also notice that for every  $y$  such that  $C_y|_\rho$  is not fixed,  $\text{Supp}(C_y|_\rho(X)) \leq 2$  as  $C_y$  only has 1 bit output. This implies that  $|\text{Supp}(U, \text{Disp}(X, U))|$  is less than  $2\delta 2^r \cdot 2 + (1 - 2\delta)2^r \leq (\frac{1}{2} + \delta)2^{r+1}$ , a contradiction to the disperser definition.

## 1.3 Paper Organization

In Section 2 we prepare some basic tools used in the rest of the paper. In Section 3 we show that merger can be implemented in  $AC^0$ . In Section 4 we give our new error reduction. In Section 5 we give our new output stretch and show our  $AC^0$  computable extractor finally. In Section 6 we show our  $NC^1$  computable extractor. In Section 7 we give our lower bound for dispersers in  $AC^0$ . In Section 8 we describe some open questions.

## 2 Preliminaries

We use the following results from previous works. First, we review the extractors in  $\text{AC}^0$  from [7]. They are actually logspace-uniform constructions, though [7] did not explicitly mention this. We briefly explain the reason after exhibiting their results.

► **Theorem 4** ([7]). *For every constant  $a, c \geq 1$ , every  $k = \delta n = \Theta(n/\log^a n)$  there exists an explicit  $(k, 1/n^c)$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  computable in  $\text{AC}^0$  with depth  $O(a)$ , where  $d = O(\log n)$ ,  $m = k^{0.01}$ .*

► **Remark 5.** Theorem 4 uses several tools and all of them can be implemented by logspace-uniform  $\text{AC}^0$  circuits. Specifically they use hardness amplifications from [17] and [18] and the Nisan-Wigderson (NW) generator [22]. These tools only use 4 kinds of operations: 1) pairwise independent generator; 2) inner product in  $\mathbb{F}_2^{O(\log n)}$ ; 3) parity function on  $O(\log n)$  bits; 4) Construct a combinatorial design and run the NW generator. It is straightforward to see that Procedure 1), 2) and 3) are all logspace-uniform. Procedure 4) is also logspace-uniform by Lemma A.3 in [6].

For smaller errors, they have the following theorem.

► **Theorem 6** ([7] for small entropy). *For every constant  $\gamma \in (0, 1)$ ,  $a, c \geq 1$ ,  $k = \delta n = \Theta(n/\log^a n)$ ,  $\varepsilon = 2^{-\Theta(\log^c n)}$ , there exists an explicit  $(k, \varepsilon)$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  in  $\text{AC}^0$  with depth  $O(a+c)$ , where  $d = O\left(\left(\log n + \frac{\log(n/\varepsilon)\log(1/\varepsilon)}{\log n}\right)/\delta\right)$ ,  $m \geq (1-\gamma)k$ .*

Also, recall the sample-then-extract technique in  $\text{AC}^0$ .

► **Theorem 7** ([7] Sample-then-extract). *For every constant  $\delta \in (0, 1]$ ,  $c \geq 1$  and every  $\varepsilon = 2^{-\log^c n}$ , there exists an explicit strong  $(\delta n, \varepsilon)$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  in  $\text{AC}^0$  with depth  $O(c)$ , where  $d = O(\log(n/\varepsilon))$ ,  $m = \Theta(\log(n/\varepsilon))$ .*

► **Remark 8.** Theorem 7 has two main ingredients: 1) The  $\text{NC}^1$  sampler from [15]. 2) Transforming a circuit of input length  $l = \Theta(\log^c n)$ , depth  $O(\log l)$  and size  $\text{poly}(l)$  to a  $\text{AC}^0$  circuit, from [12] (See also Lemma 12). Both of them are indeed logspace-uniform.

Theorem 6 uses Theorem 4 together with an error reduction and output stretch procedure. Both the error reduction and output stretch only consist of some sample-then extract techniques and some utilities of the transformation from [12]. Hence it is also logspace-uniform.

Leftover hash lemma is also needed in our construction.

► **Lemma 9** (Leftover Hash Lemma [16]). *Let  $X$  be an  $(n', k = \delta n')$ -source. For any  $\Delta > 0$ , let  $H$  be a universal family of hash functions mapping  $n'$  bits to  $m = k - 2\Delta$  bits. The distribution  $U \circ \text{EXT}(X, U)$  is at distance at most  $1/2^\Delta$  to uniform distribution where the function  $\text{EXT} : \{0, 1\}^{n'} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  chooses the  $U$ 'th hash function  $h_U$  in  $H$  and outputs  $h_U(X)$ .*

*For universal hash functions, we use the construction from Toeplitz matrices. For every  $u$ , the hash function  $h_A(x)$  equals to  $Ax$  where  $A$  is a Toeplitz matrix.*

Error reduction for extractors has been extensively studied in previous works. We recall the following key ingredient in the classic error-reducing technique [25].

► **Lemma 10** ( $G_x$  Property [25]). *Let  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  be a  $(k, \varepsilon)$ -extractor with  $\varepsilon < 1/4$ . Let  $X$  be any  $(n, k+t)$ -source. For every  $x \in \{0, 1\}^n$ , there exists a set  $G_x$  such that the following holds.*



- For every  $x \in \{0, 1\}^n$ ,  $G_x \subset \{0, 1\}^d$  and  $|G_x|/2^d = 1 - 2\epsilon$ .
- If we draw a  $y$  from  $\text{EXT}(X, G_X)$  (draw an  $x$  from  $X$ , then draw  $g_x$  uniformly from the set  $G_x$ , take  $y = \text{EXT}(x, g_x)$ ), then with probability at least  $1 - 2^{-t}$  over this random drawing, the  $y$  we get can have the property that  $\Pr[\text{EXT}(X, G_X) = y] \leq 2^{-(m-1)}$ . Here  $\text{EXT}(X, G_X)$  is obtained by first sampling  $x$  according to  $X$ , then choosing  $r$  uniformly from  $G_x$ , and outputting  $\text{EXT}(x, r)$ .

We also need to use the following lemmas about low-depth circuits computing.

► **Lemma 11** (folklore, see also [7]). *Let  $a > 0$  be an absolute constant. Then  $\log^a(n)$ -bit parity can be computed by an  $AC^0$  circuit with  $O(a)$  depth and  $\text{poly}(n)$  size.*

► **Lemma 12** ([12]). *For every  $c \in \mathbb{N}$ , every integer  $l = \Theta(\log^c n)$ , if the function  $f_l : \{0, 1\}^l \rightarrow \{0, 1\}$  can be computed by circuits of depth  $O(\log l)$  and size  $\text{poly}(l)$ , then it can be computed by  $AC^0$  circuits of depth  $c + 1$ , size  $\text{poly}(n)$ .*

► **Remark 13.** The transformation from [12] mainly uses Barrington's Theorem [3] which provides a Dlogtime-uniform  $AC^0$  reduction from any  $NC^1$  circuit to a downward self-reducible  $NC^1$ -complete language. The self-reducible here is logspace-uniform  $NC^0$  reduction. Thus the  $NC^1$  complete language of input size  $l = \Theta(\log^c n)$  can be reduced to a language of input size  $O(\log n)$  and thus can be decided by logspace-uniform  $AC^0$  circuits.

Finally, we use some folklore facts about block sources. Proofs of them can be found in the full version.

► **Definition 14** (block source). *Let  $X = (X_1, \dots, X_l)$  such that each  $X_i$  is distributed on  $\{0, 1\}^{n_i}$ . We say  $X$  is a  $(n_1, k_1, n_2, k_2, \dots, n_l, k_l)$ -block source if for every  $i \in [l]$  and  $(x_1, \dots, x_{i-1}) \in \{0, 1\}^{n_1 + \dots + n_{i-1}}$ ,  $X_i|_{X_1=x_1, \dots, X_{i-1}=x_{i-1}}$  is a  $(n_i, k_i)$ -source.*

► **Lemma 15.** *Fix  $t \in \mathbb{N}$  and  $k, s, n, n_1, \dots, n_k \in \mathbb{N}$  such that  $n_1 + \dots + n_k = n$ . Let  $X = (X_1, \dots, X_l)$  be a  $(n, n - k)$ -source on  $\{0, 1\}^n$  such that  $X_i$  is distributed on  $\{0, 1\}^{n_i}$  for each  $i \in [t]$ . Then  $(X_1, \dots, X_l)$  is  $l \cdot 2^{-s}$ -close to a  $(n_1, n_1 - k, n_2, n_2 - k - s, \dots, n_l, n_l - k - s)$ -source.*

► **Lemma 16.** *Let  $X = (X_1, \dots, X_l)$  be a  $(n_1, k_1, n_2, k_2, \dots, n_l, k_l)$ -block source on  $\{0, 1\}^n$ . Suppose that  $\text{EXT}_i : \{0, 1\}^{n_i} \times \{0, 1\}^r \rightarrow \{0, 1\}^{m_i}$  is a strong  $(k_i, \epsilon)$ -extractor for each  $i \in [l]$ . Let  $Y$  be a uniformly random variable on  $\{0, 1\}^r$ . Take  $Z = (Z_1, \dots, Z_l)$  such that  $Z_i = \text{EXT}_i(X_i, Y)$ . Then  $(Y, Z)$  is  $l \cdot \epsilon$ -close to uniform.*

► **Definition 17** (strong two-block extractor). *We say a function  $\text{EXT} : \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^r \rightarrow \{0, 1\}^m$  is a strong  $(k_1, k_2, \epsilon)$ -two-block extractor, if for any  $(k_1, k_2)$ -block-source  $X = (X_1, X_2)$  and independent uniform random distribution  $U_r$  on  $\{0, 1\}^r$ , the joint distribution  $(U_r, \text{EXT}(X_1, X_2, U_r))$  is  $\epsilon$ -close to uniform distribution on  $\{0, 1\}^r \times \{0, 1\}^m$ .*

► **Lemma 18.** *Let  $\text{EXT}_1 : \{0, 1\}^{n_1} \times \{0, 1\}^{m_1} \rightarrow \{0, 1\}^{m_2}$  be a  $(k_1, \epsilon_1)$ -strong extractor, and  $\text{EXT}_2 : \{0, 1\}^{n_2} \times \{0, 1\}^r \rightarrow \{0, 1\}^{m_1}$  be a  $(k_2, \epsilon_2)$ -strong extractor. Then the construction*

$$\text{EXT}(X_1, X_2, U_r) = \text{EXT}_1(X_1, \text{EXT}_2(X_2, U_r)) \quad (1)$$

*is a strong  $(k_1, k_2, \epsilon_1 + \epsilon_2)$ -two-block extractor*



### 3 Merger in $AC^0$

In this section, we will examine the merger construction in [9] and show that the merger can indeed be implemented in  $AC^0$  for some specific setting of parameters.

We start by defining somewhere- $(n, k)$  sources.

► **Definition 19** (somewhere- $(n, k)$  source). *Let  $X = (X_1, \dots, X_\Lambda)$  such that each  $X_i$  is distributed on  $\{0, 1\}^n$ . We say  $X$  is a simple somewhere- $(n, k)$  source with  $\Lambda$  segments if there exists  $i \in [\Lambda]$  such that  $X_i$  is a  $(n, k)$ -source on  $\{0, 1\}^n$ . We say  $X$  is a somewhere-uniform source if  $X$  is a convex combination of simple somewhere- $(n, k)$  sources.*

*If  $n = k$  in the above definition, which means that  $X_i$  is uniform, we say  $X$  is a somewhere-uniform source.*

A merger is a function that takes a somewhere-uniform source and a uniform random seed as input and outputs a  $(m, k')$ -source. The remaining entropy  $k'$  is usually less than the original entropy  $k$ .

► **Definition 20** (merger and strong merger). *We say  $\text{Merge} : \{0, 1\}^{\Lambda \cdot n} \times \{0, 1\}^r \rightarrow \{0, 1\}^m$  is a  $(k, k', \varepsilon)$ -merger if for any somewhere- $(n, k)$  source  $X = (X_1, \dots, X_\Lambda)$ , the distribution  $\text{Merge}(X, U_r)$  is  $\varepsilon$ -close to a  $k'$ -source. Here  $U_r$  is a independent uniform random distribution on  $\{0, 1\}^r$*

*Furthermore, if  $(U_r, \text{Merge}(X, U_r))$  is  $\varepsilon$ -close to  $(U_r, W)$ , we say  $\text{Merge}$  is a strong  $(k, k', \varepsilon)$ -merger. Here  $W$  is a distribution such that for all  $a \in \{0, 1\}^r$ ,  $W|_{U_r=a}$  is a  $k'$ -source.*

We examine the merger introduced in [9], and find that the merger can be implemented in  $AC^0$  if the number of segments is not too large.

► **Theorem 21** (merger in [9]). *For any constant  $a, c > 0$ ,  $\delta \in (0, 1)$ , let  $\Lambda(n) \leq \log^a(n)$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ . Then there exists explicit  $(n, \delta n, \varepsilon(n))$ -mergers  $\text{Merge} : \{0, 1\}^{\Lambda(n) \cdot n} \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^n$ . Here  $r(n) = O(\log(\frac{1}{\varepsilon}))$ .*

*Furthermore, the mergers can be implemented in  $AC^0$  with  $O(a + c + 1)$  depth and  $\text{poly}(n)$  size,*

The merger in [9] is defined as follows:

Define  $q = 2^s$  be a power of two which is decided later. Let  $\mathbb{F}_q$  be the finite field of order  $q$ . Let  $X = (X_1, \dots, X_\Lambda)$  be a somewhere-uniform-source with  $\Lambda$  segments. Regard each  $X_i$  as distributed on  $\mathbb{F}_q^K$  with  $K = \frac{n}{s}$ . Then

$$X_i = (X_{i,1}, \dots, X_{i,K}), \quad X_{i,j} \in \mathbb{F}_q. \quad (2)$$

Note that the uniform distribution on  $\mathbb{F}_q^K$  is equivalent to the uniform distribution on  $\{0, 1\}^n$ .

Take  $\gamma_1, \dots, \gamma_\Lambda$  be  $\Lambda$  unique points in  $\mathbb{F}_q$ . Let  $C_1, \dots, C_\Lambda$  be  $\Lambda$  unique polynomials in  $\mathbb{F}_q[x]$  of degree at most  $\Lambda - 1$ , such that  $C_i(\gamma_j) = 1$  if  $i = j$  and  $C_i(\gamma_j) = 0$  if  $i \neq j$ . Then the merger is defined as:

$$\text{Merge}(X, y) = \left( \sum_{i=1}^{\Lambda} C_i(y) X_{i,1}, \dots, \sum_{i=1}^{\Lambda} C_i(y) X_{i,K} \right), \quad (3)$$

where  $y \in \mathbb{F}_q$ .

## 69:10 Randomness Extractors in $AC^0$ and $NC^1$ : Optimal up to Constant Factors

► **Lemma 22** (merger in [9]). *For any constant  $\delta > 0$ , let  $q \geq (\frac{2\Lambda}{\varepsilon})^{1/\delta}$ . Then the function  $\text{Merge} : \mathbb{F}_q^{K \cdot \Lambda} \times \mathbb{F}_q \rightarrow \mathbb{F}_q^K$  is a  $(K \log q, k, \varepsilon)$ -merger, where  $k = (1 - \delta) \cdot K \cdot \log q$ .*

The condition  $q \geq (\frac{2\Lambda}{\varepsilon})^{1/\delta}$  is equivalent to  $r \geq \frac{1}{\delta} \log(\frac{2\Lambda}{\varepsilon})$ . When  $\Lambda = \log^a(n)$ ,  $\varepsilon = 2^{-\log^c(n)}$ , this requires  $r \geq \frac{2}{\delta} \log^c(n)$ . So we can pick  $r(n) = \min\{s \in \mathbb{N} \mid s \geq \frac{2}{\delta} \log^c(n), \exists d \in \mathbb{N}, s = 3 \cdot 2^d\}$ . As  $\delta$  is a constant,  $r(n) = O(\log^c(n)) = O(\log(\frac{1}{\varepsilon}))$ .

► **Lemma 23.** *For any constant  $a, c, \delta \in (0, 1)$ , let  $\Lambda(n) \leq \log^a n$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ . Define  $r(n) = \min\{s \in \mathbb{N} \mid s \geq \frac{2}{\delta} \log^c(n), \exists d \in \mathbb{N}, s = 3 \cdot 2^d\}$ ,  $q(n) = 2^{r(n)}$ ,  $K(n) = \frac{n}{r(n)}$ . Then the  $(n, \delta n, \varepsilon)$ -merger  $\text{Merge} : \{0, 1\}^{\Lambda(n) \cdot n} \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^n$  can be implemented in uniform  $AC^0$  with  $O(a + c + 1)$  depth and  $\text{poly}(n)$  size.*

To prove the lemma, we can express the  $\Lambda$  polynomials  $C_1, \dots, C_\Lambda$  by their  $\Lambda^2$  coefficients. That is:

$$C_i(y) = \sum_{j=1}^{\Lambda} c_{i,j} y^{j-1}, \quad c_{i,j} \in \mathbb{F}_q, \quad i \in [\Lambda].$$

These coefficients are not necessarily computable in  $AC^0$ . Instead, they can be pre-determined and stored in the circuit. Note that  $\Lambda = \log^a(n)$  and  $r_2(n) = O(\log^c(n))$ . Therefore it requires  $O(\log^c(n))$  bits to store one coefficient, and  $O(\log^{2a+c}(n))$  bits to store all the coefficients.

Therefore, the  $AC^0$  circuit for the merger is only required to do three types of operations: powering, multiplication and summation. The parameters of these operations satisfies the following conditions:

1. The powering operation is to compute  $y^j$ , where  $j \leq \log^a(n)$ , and  $y \in \mathbb{F}_q$ . The order  $q = 2^s$  is a power of 2, and  $s = O(\log^c(n))$ .
2. The multiplication operation is to compute  $c_{i,j} y^{j-1} X_{i,k}$ , for each  $i \in [\Lambda]$ ,  $j \in [\Lambda]$ ,  $k \in [K]$ . All of the three multipliers are in  $\mathbb{F}_q$ .
3. The summation operation is to compute  $\sum_{i=1}^{\Lambda} \sum_{j=1}^{\Lambda} c_{i,j} y^{j-1} X_{i,k}$  for each  $k \in [K]$ . All the addends are in  $\mathbb{F}_q$ , and the total number of them is  $\log^{4a}(n)$ .

The following theorems in the work of Healy and Viola [14] show that the powering and multiplication are indeed in  $AC^0$ .

► **Lemma 24** ([14, Corollary 6(1)]). *Let  $a, c > 0$  be absolute constants. Let  $y \in \mathbb{F}_q$  where  $q = 2^s$  and  $s = 2 \cdot 3^d$  for some  $d \in \mathbb{N}$ . Suppose that  $j \leq \log^a(n)$  and  $s \leq \log^c(n)$ , then  $y^j$  can be computed by a logspace-uniform  $AC^0$  circuit with  $O(a + c)$  depth and  $\text{poly}(n)$  size.*

► **Lemma 25** ([14, Corollary 6(2)]). *Let  $a, c > 0$  be absolute constants. Let  $y_1, y_2 \in \mathbb{F}_q$  where  $q = 2^s$  and  $s = 2 \cdot 3^d$  for some  $d \in \mathbb{N}$ . Suppose that  $s \leq \log^c(n)$ , then  $y_1 \cdot y_2$  can be computed by a logspace-uniform  $AC^0$  circuit with  $O(c)$  depth and  $\text{poly}(n)$  size.*

The summation operation is also in  $AC^0$ , as the summation of elements in  $\mathbb{F}_q$  where  $q = 2^s$  is equivalent to bitwise parity of the binary representation of the elements if we implement  $\mathbb{F}_q$  by polynomial fields with coefficients in  $\mathbb{F}_2$ . When the number of addends is  $\text{poly} \log n$ , it is in  $AC^0$  by Lemma 11.

With these results, the merger can be implemented in  $AC^0$  with  $O(a + c)$  depth and  $\text{poly}(n)$  size.

**Proof of Lemma 23.** It is sufficient to prove that each  $\sum_{i=1}^{\Lambda} \sum_{j=1}^{\Lambda} c_{i,j} y^{j-1} X_{i,k}$  can be computed in  $\text{AC}^0$  with  $O(a+c)$  depth and  $\text{poly}(n)$  size. The powering could be computed in  $O(a+c)$  depth and  $\text{poly}(n)$  size by Lemma 24. The multiplication could be computed in  $O(c)$  depth and  $\text{poly}(n)$  size by Lemma 25. The summation could be computed in  $O(a)$  depth and  $\text{poly}(n)$  size by Lemma 11.  $\blacktriangleleft$

Theorem 21 follows directly from Lemma 22 and Lemma 23.

**Proof of Theorem 21.** Take  $r(n) = \min\{s \in \mathbb{N} \mid s \geq \frac{2^{\log^c(n)}}{\delta}, \exists d \in \mathbb{N}, s = 3 \cdot 2^d\}$ ,  $q(n) = 2^{r(n)}$ ,  $K(n) = \frac{n}{r(n)}$  as discussed above. By Lemma 22, we know that the merger is a  $(n, k(n), \varepsilon(n))$ -merger, where  $k(n) = (1-\delta)n$ . By Lemma 23, we know that the merger can be implemented in  $\text{AC}^0$  with  $O(a+c)$  depth and  $\text{poly}(n)$  size.  $\blacktriangleleft$

As noted in [9], their merger for somewhere uniform sources can be extended to handle somewhere high entropy sources. Following their idea, we also prepare a merger for somewhere high entropy sources, and furthermore, it is computable by low-depth circuits.

► **Corollary 26.** Let  $\delta \in (0, 1)$ ,  $\Lambda(n) \leq \text{poly}(n)$ ,  $\varepsilon(n) = 2^{-O(n)}$ ,  $\Delta(n) = O(\log(\frac{n}{\varepsilon}))$ . Then there exists a strong  $(n - \Delta(n), \delta m(n), \varepsilon(n))$ -merger  $\text{Merge} : \{0, 1\}^{\Lambda(n) \cdot n} \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$ . Here  $r(n) = O(\log(\frac{n}{\varepsilon}))$  and  $m(n) = \Omega(n)$ . The merger is computable in  $\text{logspace-uniform AC}^0[2]$ .

If  $\Lambda(n) \leq \log^a(n)$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$  for constant  $a, c > 0$ , then the merger can be implemented in  $\text{AC}^0$  with  $O(a+c+1)$  depth and  $\text{poly}(n)$  size.

## 4 Error Reduction

The main theorem of this section is the following:

► **Theorem 27.** For any constant  $a, c > 0$ ,  $b \in \mathbb{N}^+$ , every  $k(n) \geq n/\log^a(n)$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ , there exists a strong  $(k(n), \varepsilon(n))$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$ , where  $r(n) = O(\log(\frac{n}{\varepsilon(n)}))$ ,  $m(n) = \Theta(\log^b(n) \cdot \log(\frac{n}{\varepsilon(n)}))$ .

Furthermore, the extractor can be implemented in  $\text{AC}^0$  with  $O(b(a+c+1))$  depth.

We show this theorem by giving a new error reduction stated as the following. To describe it, we fix  $a > 0$  to be a constant and  $k(n) = \frac{n}{\log^a n}$ .

► **Lemma 28.** For any  $\varepsilon_0 \in (0, 1)$  every constant  $c > 0$  and  $\varepsilon = 2^{-\log^c n}$ , suppose there exists a  $(k, \varepsilon_0)$ -extractor  $\text{EXT}_0 : \{0, 1\}^n \times \{0, 1\}^{d_0} \rightarrow \{0, 1\}^{m_0}$  with  $m_0 \geq k^{0.01}$  and a family of strong  $(n_1/100, \varepsilon)$ -extractors  $\text{EXT}_1 : \{0, 1\}^{n_1} \times \{0, 1\}^{d_1} \rightarrow \{0, 1\}^{m_1}$  for every  $n_1 \in [m_0^{0.1}, m_0]$ . Then for any  $\varepsilon = 2^{-\log^c n}$ , there exists a strong  $(k, \varepsilon)$ -extractor  $\text{EXT}' : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ , where  $d = O(d_1 + d_0 \cdot \frac{\log \varepsilon}{\log \varepsilon_0})$ ,  $m = \Theta(m_1 \cdot \log n)$ .

If  $\text{EXT}_0$  and  $\text{EXT}_1$  can be realized by depth  $h$  and  $g$  AC circuits respectively, then  $\text{EXT}'$  can be realized by a depth  $O(h+g+c+1)$  AC circuit.

Now we describe the construction and analysis of Lemma 28.

### 4.1 Step 1: extracting in parallel

We apply  $\text{EXT}_0$  for  $t = \frac{\log(1/\varepsilon)}{\log(1/\varepsilon_0)}$  times in parallel, with independent seeds. Specifically, take  $U_{1,i}$  be independent uniform seeds in  $\{0, 1\}^{d_0}$  for every  $i \in [t]$ . Let  $Y = (Y_1, Y_2, \dots, Y_t)$ , where  $Y_i = \text{EXT}_0(X, U_{1,i})$ .

The step can be computed by depth  $h$  AC circuits because the extractor  $\text{EXT}_0$  has depth  $h$ , and the parallel extraction can be done without increasing the depth.

### Analysis

We now show that  $Y$  is close to a somewhere- $(m_0(n), m_0(n) - O(\log t))$ -source. The main idea is that by Lemma 10, we know that with high probability, at least one of the seeds  $U_i$  lands in  $G_x$ , which makes  $Y_i$  a good source with a high entropy rate. The following lemma states this formally:

► **Lemma 29.** *Let  $EXT_0 : \{0, 1\}^n \times \{0, 1\}^{d_0} \rightarrow \{0, 1\}^{m_0}$  be an  $(k, \varepsilon_0)$ -extractor and  $X$  be a  $(n, k + s)$ -source. Take independent seeds  $U_1, U_2, \dots, U_s \in \{0, 1\}^{d_0}$ . Let  $Y = (Y_1, Y_2, \dots, Y_t)$ , where  $Y_i = EXT_0(X, U_i)$ . Then  $Y$  is  $(2\varepsilon_0)^t + t \cdot 2^{-s}$ -close to a somewhere- $(m_0, m_0 - O(\log t))$ -source*

Take  $x$  from a fixed distribution  $X$  and fix extractor  $EXT$ . Let  $G_x$  be the set of good seeds from Lemma 10. We first denote event  $BAD_i = \{U_i \notin G_x\}$ . Note that these events are not necessarily independent. However, the probability that all of them happen is exponentially small, as the following claim shows.

▷ **Claim 30.**  $\Pr[BAD_1 \wedge BAD_2 \wedge \dots \wedge BAD_t] \leq (2\varepsilon_0)^t$ .

We define an indicator random variable  $I \in \{0, 1\}^{[t]}$  as follows:

$$\forall i \in [t], i \in I \iff U_i \in G_x. \quad (4)$$

With probability at least  $1 - (2\varepsilon_0)^t$ , The set  $I$  is not an empty set. Take  $Y_i = EXT(X, U_i)$ . By Lemma 10,  $Y_i|_{(BAD_i)^c} = Y_i|_{i \in I}$  is  $2^{-s}$ -close to a  $(m_0, m_0 - O(1))$  source.

We apply the technique from [20] to prove that  $(Y_1, Y_2, \dots, Y_t)$  is indeed close to a somewhere- $(m_0, m_0 - O(\log t))$ -source.

► **Lemma 31** ([20]). *Let  $Y = (Y_1, \dots, Y_t)$  be the random variable defined in Lemma 29. Let  $I$  be a random set subset of  $[t]$ . Assume  $I \neq \emptyset$ , and for every  $i \in [t]$ ,  $Y_i|_{i \in I}$  is  $\varepsilon$ -close to a  $(m, k)$ -source. Then  $Y$  is  $(t \cdot \varepsilon)$ -close to a somewhere- $(m, k - \log t)$  source.*

By Claim 30 and Lemma 31, we can prove Lemma 29:

**Proof of Lemma 29.** Take  $I$  as the random set indicator defined above. By Lemma 10,  $Y_i|_{(BAD_i)^c} = Y_i|_{i \in I}$  is  $2^{-s}$ -close to a  $(m_0, m_0 - O(1))$  source. By Claim 30, we know that with probability at least  $1 - (2\varepsilon_0)^t$ ,  $I$  is not an empty set. Conditioning on such events, Lemma 31 implies that  $Y|_{\{I \neq \emptyset\}}$  is  $t \cdot 2^{-s}$ -close to a somewhere- $(m_0, m_0 - O(\log t))$  source. The lemma follows. ◀

## 4.2 Step 2: divide and merge

Assume we have a somewhere- $(m_0, m_0 - O(\log t))$ -source. We divide each segment of the source into a sequence of blocks whose lengths form a geometric sequence. Specifically, take  $Y = (Y_1, Y_2, \dots, Y_t)$  to be a simple somewhere- $(m_0, m_0 - O(\log t))$ -source. We divide each  $Y_i$  into  $l + 1$  blocks of length  $m_1, m_2, \dots, m_{l+1}$  respectively, such that

$$Y_i = (Y_{i,1}, Y_{i,2}, \dots, Y_{i,l+1}) \text{ for every } i \in [t]. \quad (5)$$

The lengths satisfies

$$m_j = m_0^{0.1} \cdot 3^{j-1} \text{ for every } j \in [l]. \quad (6)$$

where  $l = \lceil \log_3 m_0^{0.9} \rceil$ . Denote  $Y_{i,1\dots j} = (Y_{i,1}, Y_{i,2}, \dots, Y_{i,j})$  for every  $i \in [t]$  and  $j \in [l]$ . Define  $B_j$  as:

$$B_j = (Y_{1,1\dots j}, Y_{2,1\dots j}, \dots, Y_{t,1\dots j}) \text{ for every } j \in [l]. \quad (7)$$

We denote  $M_j = m_1 + m_2 + \dots + m_j$  for every  $j \in [l]$ .

Let  $\text{Merge}_j : \{0, 1\}^{t \cdot M_j} \times \{0, 1\}^{d_2(n)} \rightarrow \{0, 1\}^{(1-\alpha)M_j}$  be a strong  $(M_j - \Delta, \frac{3}{4}(1-\alpha)M_j, \varepsilon(n)/l)$ -merger from Corollary 26 for every  $j \in [l]$ , where  $\alpha$  is a constant. The seed length of the merger is  $d_2(n) = O(\log(\frac{M_j}{\varepsilon(n)})) = O(\log(\frac{m(n)}{\varepsilon(n)}))$ . Let  $U_2$  be a uniform random variable on  $\{0, 1\}^{d_2(n)}$ . Define

$$Z_j = \text{Merge}_j(B_j, U_2) \text{ for every } j \in [l]. \quad (8)$$

The gap between source length and source entropy is  $\Delta = O(\log t) = O(\log \frac{1}{\varepsilon(n)})$ , which meets the requirement that  $\Delta = O(\log \frac{M_j}{\varepsilon(n)})$  in Corollary 26.

Next, we apply the strong extractor family  $\text{EXT}_1$  to extract from the block source. Let  $\text{EXT}_{1,j} : \{0, 1\}^{(1-\alpha)M_j} \times \{0, 1\}^{d_3(n)} \rightarrow \{0, 1\}^{m'(n)}$  be a strong  $((1-\alpha)M_j/100, \varepsilon(n)/l)$ -extractor for every  $j \in [l]$ . These  $\text{EXT}_{1,j}, j \in [l]$  with different input lengths, are all from the family  $\text{EXT}_1$ . Let  $U_3$  be a uniform random variable on  $\{0, 1\}^{d_3(n)}$ . Then

$$W_j = \text{EXT}_{1,j}(Z_j, U_3) \text{ for every } j \in [l]. \quad (9)$$

### Analysis

Now we give our analysis. Note that since  $Y$  is a simple somewhere high entropy source, by dividing it into blocks, each prefix  $B_j$  is a simple somewhere- $(M_j, M_j - O(\log t))$ -source. Through merging,  $Z_j$ 's are correlated high-entropy sources with different lengths. They are close to a block source.

► **Lemma 32.**  $Z_j$  is  $\varepsilon(n)/l$ -close to a  $((1-\alpha)M_j, \frac{3}{4}(1-\alpha)M_j)$ -source for every  $j \in [l]$ .

**Proof.** Let  $Y_i$  be a  $(m_0, m_0 - O(\log t))$ -source in  $Y$ . Then  $Y_{i,1\dots j}$  must have entropy at least  $m_j - O(\log t)$ . Therefore  $B_j$  is a somewhere- $(m_j, m_j - O(\log t))$ -source. By Corollary 26,  $Z_j$  is  $\varepsilon(n)/l$ -close to a  $((1-\alpha)M_j, \frac{3}{4}(1-\alpha)M_j)$ -source. The claim follows. ◀

Denote  $Z_0 = (U_1, U_2)$  as the seeds used in all previous steps to obtain  $Z_1, \dots, Z_j$ . We stress that the sequence  $Z_0, Z_1, \dots, Z_l$  is of exponentially increasing length and each contains  $|Z_j| - O(\log \frac{1}{\varepsilon(n)})$  bits of min-entropy. Therefore, even if all the randomness in  $(Z_0, \dots, Z_i)$  is contained in  $Z_{i+1}$ , there still must be  $\Omega(|Z_{i+1}|)$  bits of conditional min-entropy within  $Z_{i+1}$ . That makes the sequence a block source. We formalize the inspection into the following lemma.

► **Lemma 33.**  $(Z_0, Z_1, Z_2, \dots, Z_l)$  is  $2\varepsilon(n)$ -close to a block source  $(Z_0, Z'_1, Z'_2, \dots, Z'_l)$ . The conditional entropy of  $Z'_j$  is larger than  $(1-\alpha)M_j/100 = \Omega((1-\alpha)M_j)$  for each  $j \in [l]$

Then we can extract from the block-source  $(Z_0, Z_1, Z_2, \dots, Z_l)$  using standard methods, which gives  $(Z_0, U_3, W_1, W_2, \dots, W_l)$ :

► **Lemma 34.**  $(Z_0, U_3, W_1, W_2, \dots, W_l)$  is  $3\varepsilon(n)$ -close to  $(Z_0, U_3, V)$ , where  $V$  is a independent uniform distribution.

### 4.3 Wrap-up to prove Lemma 28 and Theorem 27

**Proof of Lemma 28.** Take  $X$  be the sources,  $U_1, U_2, U_3$  be the seeds. Let  $Y = (Y_1, Y_2, \dots, Y_t)$  such that  $Y_i = \text{EXT}_0(X, U_{1,i})$  for every  $i \in [t]$  as in the first step. By Lemma 29,  $Y$  is  $\varepsilon(n)$ -close to a somewhere- $(m(n), m(n) - O(\log t))$ -source. Let  $B_j$  be the source  $(Y_{1,1\dots j}, Y_{2,1\dots j}, \dots, Y_{t,1\dots j})$  for every  $j \in [l]$ . Then take  $Z_j = \text{Merge}_j(B_j, U_2)$  and  $W_j = \text{EXT}_{1,j}(Z_j, U_3)$  for every  $j \in [l]$  as in the second step. Here  $\text{EXT}_{1,j}$  is the strong extractor from family  $\text{EXT}_1$  with source length  $n_1 = m_j$ . By Lemma 34 and its remark,  $(U_1, U_2, U_3, W)$  is  $3\varepsilon(n)$ -close to uniform if  $Y$  is a somewhere- $(m(n), m(n) - O(\log t))$ -source. By the triangle inequality,  $W$  is  $4\varepsilon(n)$ -close to uniform.

Step 1 executes the extractor  $\text{EXT}_0$  in parallel, which costs depth  $h$ . Step 2 executes the merger  $\text{Merge}_j$  from Corollary 26 and the extractor  $\text{EXT}_{1,j}$ , for every  $j \in [l]$  in parallel. This takes depth  $O(c + g)$ . So the overall depth is as the lemma stated.

The seed length of the extractor is  $d(n) = |U_1| + |U_2| + |U_3|$ .  $U_1 = (U_{1,1}, U_{1,2}, \dots, U_{1,t})$  where  $|U_{1,i}| = d_0$  for every  $i \in [t]$  and  $t = \frac{\log \varepsilon(n)}{\log \varepsilon_0}$ .  $|U_2| = O(\log(\frac{n}{\varepsilon(n)}))$  and  $|U_3| = d_1$ . Therefore  $d = O(d_1 + d_0 \cdot \frac{\log \varepsilon}{\log \varepsilon_0})$ .

The output consists of  $\Theta(\log n)$  parts of length  $m_1$ . Therefore the output length is  $m = \Theta(m_1 \cdot \log n)$ .  $\blacktriangleleft$

We instantiate  $\text{EXT}_0$  as the extractor from Theorem 4 and  $\text{EXT}_1$  as the strong extractors from Theorem 7, which gives the following theorem:

► **Corollary 35.** *For any constant  $a, c > 0$ , every  $k(n) \geq n/\log^a(n)$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ , there exists a strong  $(k(n), \varepsilon(n))$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$ , where  $r(n) = O(\log(\frac{n}{\varepsilon(n)}))$ ,  $m(n) = \Theta(\log(n) \cdot \log(\frac{n}{\varepsilon(n)}))$ .*

*Furthermore, the extractor can be implemented in uniform  $AC^0$  with  $O(a + c + 1)$  depth.*

The only gap between Corollary 35 and Theorem 27 is that the output length of Corollary 35 is only  $\Theta(\log(n) \cdot \log(\frac{n}{\varepsilon}))$  instead of  $\Theta(\log^b(n) \cdot \log(\frac{n}{\varepsilon}))$ . We resolve the issue by repeatedly using Lemma 28, each time instantiating  $\text{EXT}_1$  in Lemma 28 as the strong extractor family provided by the immediate previous using of Lemma 28. After an iteration, the output length is multiplied by a  $\Theta(\log n)$  factor. Therefore we can achieve the parameter as in Theorem 27 after  $b$  iterations.

## 5 Output Stretch

In this section, we will use the framework introduced in [9], to further stretch the output length from  $O(\log^c(n))$  to a near-optimal  $O(k)$ . The main theorem of this section is the following:

► **Theorem 36.** *For any constant  $a, c > 0$  and  $\gamma \in (0, 1)$ , let  $k(n) \geq \frac{n}{\log^a(n)}$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ . Then there exists a  $(k(n), \varepsilon(n))$ -strong extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$ , such that  $r(n) = O(\log(\frac{n}{\varepsilon}))$ , and  $m(n) \geq (1 - \gamma) \cdot k(n)$ .*

*Furthermore, the extractor can be implemented in  $AC^0$ , with  $O(a + c + 1)^2$  depth and  $\text{poly}(n)$  size.*

We use a four-step method to extract randomness.

## 5.1 Step 1: Converting to a somewhere-block-source

In this subsection, we will convert the original  $k$ -source into a somewhere-block-source. First, we define the concept:

► **Definition 37** (somewhere-block-source). *Let  $X = (X_1, \dots, X_\Lambda)$  be a random variable with  $\Lambda$  segments, each  $X_i$  distributed on  $\{0, 1\}^{n_1} \times \{0, 1\}^{n_2}$ . We say  $X$  is a simple  $(k_1, k_2)$ -somewhere-block-source if there exists  $i \in [\Lambda]$  such that  $X_i$  is a  $(k_1, k_2)$ -block-source. We say  $X$  is a  $(k_1, k_2)$ -somewhere-block-source if  $X$  is a convex combination of simple  $(k_1, k_2)$ -somewhere-block-sources.*

Ta-shma's somewhere-block-source converter [33] is a deterministic function that converts a  $k_1 + k_2 + s$ -source into a  $(k_1 - O(n/\Lambda), k_2)$ -somewhere-block-source, which has  $\Lambda$  segments.

Take  $X_1 \in \{0, 1\}^n$  as the original source, assume  $n$  is divisible by  $\Lambda$ , otherwise pad  $X_1$  with 0's. Regard  $X_1$  as a source with  $\Lambda$  parts, each of length  $n/\Lambda$ :

$$X_1 = (X_{1,1}, \dots, X_{1,\Lambda}) \in \left(\{0, 1\}^{n/\Lambda}\right)^\Lambda. \quad (10)$$

Now define the following separation of these parts into  $(Y_i, Z_i)$ :

$$Y_i = (X_{1,1}, \dots, X_{1,i}, 0^{(\Lambda-i) \cdot (n/\Lambda)}), \quad (11)$$

$$Z_i = (0^{i \cdot (n/\Lambda)}, X_{1,i+1}, \dots, X_{1,\Lambda}). \quad (12)$$

Then  $(Y_i, Z_i) \in \{0, 1\}^{2n}$ . The Ta-shma's somewhere-block-source converter is defined as the collection of all  $(Y_i, Z_i)$ , for  $i \in [\Lambda]$ :

$$B_{TS}^\Lambda(X_1) = \{(Y_i, Z_i) \in \{0, 1\}^{2n} \mid i \in [\Lambda]\}. \quad (13)$$

► **Theorem 38** ([33]). *Let  $\Lambda$  be an integer and  $\Lambda$  divides  $n$ . Let  $B_{TS}^\Lambda$  be the Ta-shma's somewhere-block-source converter defined above. Fix  $k, k_1, k_2, s \in \mathbb{N}$  such that  $k = k_1 + k_2 + s$ . Then for any  $k$ -source  $X \in \{0, 1\}^n$ ,  $B_{TS}^\Lambda(X)$  is  $O(n \cdot 2^{-s/3})$ -close to a  $(k_1 - O(n/\Lambda), k_2)$ -somewhere-block-source.*

Now we summarize the first step:

Step 1: Set  $\Lambda = \log^{2a}(n)$ , Take  $X_2 = (X_{2,1}, \dots, X_{2,\Lambda}) = B_{TS}^\Lambda(X_1)$  as a somewhere-block-source.

► **Lemma 39.** *For any constant  $a \geq 0$ , let  $k \geq \frac{n}{\log^a(n)}$ . Then for any  $k$ -source  $X_1 \in \{0, 1\}^n$ , the somewhere-block-source  $X_2 = B_{TS}^\Lambda(X_1)$  is  $n \cdot 2^{-\frac{n}{\log^{2a} n}}$ -close to a  $(k - O(\frac{n}{\log^{2a} n}), \frac{n}{\log^{2a} n})$ -somewhere-block-source.*

The first step can be computed in  $AC^0$  with  $O(1)$  depth and  $\text{poly}(n)$  size, as it is only splitting the input into blocks.

## 5.2 Step 2: Extracting from a somewhere-block-source

In this subsection, we focus on the good block of the somewhere-block-source, and extract randomness from it. A two-block extractor is employed in this section. We use the block-extraction technique together with our extractors from Theorem 6 and Theorem 27 to extract  $O(\log^{a+c} n)$  randomness from the second block of the block source, then use it as seed for another extractor, in order to extract  $O(k)$  randomness from the first block of the block source.



## 69:16 Randomness Extractors in $AC^0$ and $NC^1$ : Optimal up to Constant Factors

For a somewhere-block-source, we may apply the two-block extractor to each segment such that the good segment is converted into a somewhere-close-to-uniform source. The source is defined as follows:

► **Lemma 40.** *Let  $X = (X_1, \dots, X_\Lambda)$  be a  $(k_1, k_2)$ -somewhere-block-source, where each segment is a source on  $\{0, 1\}^{n_1} \times \{0, 1\}^{n_2}$ . Let  $EXT: \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^r \rightarrow \{0, 1\}^m$  be a  $(k_1, k_2, \varepsilon)$ -strong-two-block extractor. Let  $U_r$  be a uniform random distribution on  $\{0, 1\}^r$ . Then*

$$(EXT(X_1, U_r), \dots, EXT(X_\Lambda, U_r))$$

is  $\varepsilon$ -close to a somewhere-uniform-source.

Take  $EXT_1$  from Theorem 6 and  $EXT_2$  from Theorem 27. Construct  $EXT(X_1, X_2, U_r) = EXT_1(X_1, EXT_2(X_2, U_r))$  as a strong-two-block extractor. We have the following theorem:

► **Theorem 41 (block-extraction in  $AC^0$ ).** *There exists a constant  $\gamma \in (0, 1)$ . For any constant  $a, c > 0$ , let  $k_1(n) \geq \frac{n}{\log^a(n)}$ ,  $k_2(n) \geq \frac{n}{\log^{2a}(n)}$ ,  $\varepsilon(n) \geq 2^{-\log^c(n)}$ , there exists a  $(k_1(n), k_2(n), \varepsilon(n))$ -strong-two-block extractor  $EXT: \{0, 1\}^n \times \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}^m$ , such that  $r(n) = O(\log(\frac{n}{\varepsilon}))$ , and  $m(n) \geq (1 - \gamma)k_1(n)$ .*

Furthermore, the extractor can be implemented in  $AC^0$ , with  $O(a + c + 1)^2$  depth and  $\text{poly}(n)$  size.

We summarize the second step here:

Step 2: Take  $EXT: \{0, 1\}^n \times \{0, 1\}^n \times \{0, 1\}^{r_1(n)} \rightarrow \{0, 1\}^{m(n)}$  as a  $(\frac{n}{\log^a(n)}, \frac{n}{\log^{2a}(n)}, \varepsilon(n))$ -strong-two-block extractor, where  $r_1(n) = O(\log(\frac{n}{\varepsilon}))$  and  $m(n) \geq (1 - \gamma)k(n)$ . Take  $X_3 = (EXT(X_{2,1}, U_{r_1}), \dots, EXT(X_{2,\Lambda}, U_{r_1}))$  be  $2 \cdot \varepsilon(n)$ -close to a somewhere-uniform-source.

This step can be implemented in  $AC^0$  with  $O(a + c)$  depth and  $\text{poly}(n)$  size, as it is applying  $AC^0$  functions to each block of the input.

The source  $X_3$  is now  $\varepsilon(n)$ -close to a somewhere-uniform-source. It has  $\Lambda = \log^{2a}(n)$  segments, each of length  $m(n) \geq (1 - \gamma)k(n)$ . The next step is using the merger introduced in [9] to merge the segments into one source.

### 5.3 Step 3: Merging the segments

We use the merger introduced in [9] to merge the segments of the somewhere-uniform-source into one source. The construction of the merger is discussed in Theorem 21.

Step 3: Take  $Merge: \{0, 1\}^{\Lambda \cdot m(n)} \times \{0, 1\}^{r_2(n)} \rightarrow \{0, 1\}^{m(n)}$  be the  $(m(n), \frac{3}{4}m(n), \varepsilon(n))$ -merger from Theorem 21. Then  $X_4 = Merge(X_3, U_{r_2})$ .

As a direct consequence of Theorem 21 we have the following lemma.

► **Lemma 42.**  *$X_4$  is  $3 \cdot \varepsilon(n)$ -close to a  $\frac{3}{4}m(n)$ -source.*

Also, notice that the computation in  $AC^0$  with depth  $O(a + c)$ , with seed length  $O(\log(n/\varepsilon(n)))$ .

## 5.4 Step 4: Second extraction

The final step is as the following.

Step 4: Take  $\text{EXT}_2 : \{0, 1\}^{m(n)/2} \times \{0, 1\}^{m(n)/2} \times \{0, 1\}^{r_3(n)} \rightarrow \{0, 1\}^{m'(n)}$  be the  $(\frac{1}{8}m(n), \frac{1}{8}m(n), \varepsilon(n))$ -strong-two-block extractor from Theorem 41, where  $r_3(n) = O(\log(\frac{n}{\varepsilon}))$  and  $m'(n) \geq \frac{1-\gamma}{6} \cdot m(n)$ . Take  $X_5 = \text{EXT}_2(X'_4, X''_4, U_{r_3})$ , where  $U_{r_3}$  is a uniform random distribution on  $\{0, 1\}^{r_3(n)}$ , where  $(X'_4, X''_4) = X_4$ .

► **Lemma 43.**  $X_5$  is  $5\varepsilon(n)$  close to uniform.

The circuit depth of  $\text{EXT}_2$  is  $O(a + c + 1)^2$  by Theorem 41.

Now we prove the main theorem of this section:

► **Theorem 44.** For any constant  $a, c > 0, \gamma' \in (0, 1)$ , let  $k(n) \geq \frac{n}{\log^a(n)}, \varepsilon(n) \geq 2^{-\log^c(n)}$ . Then there exists a  $(k(n), \varepsilon'(n))$ -strong extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$ , such that  $r(n) = O(\log(\frac{n}{\varepsilon(n)}))$ , and  $m(n) \geq (1 - \gamma') \cdot k(n)$ .

Furthermore, the extractor can be implemented in  $\text{AC}^0$ , with  $O(a + c + 1)^2$  depth and  $\text{poly}(n)$  size.

**Proof.** The extractor  $\text{EXT}$  is defined as  $\text{EXT}(X_1, U_{r_1}, U_{r_2}, U_{r_3}) = X_5$ , where  $X_5$  is defined through the four steps above. Detailed analysis could be found in the full version. ◀

## 6 Extractors in $\text{NC}^1$

Our method can also construct extractors in  $\text{NC}^1$  with improved parameters. The construction consists of 3 parts:

1. Apply a condenser from [19]. It behaves like the GUV condenser but is computable in  $\text{NC}^1$ . It condenses the source into a source with a constant entropy rate. We regard the output as a block source.
2. For the second block, apply our error reduction method which outputs a seed of length  $O(\log^2 n \log(n/\varepsilon))$ .
3. Apply the improved Trevisan's extractor [26] to the first block, which outputs  $\Omega(k)$  bits of randomness.

The main theorem is as follows:

► **Theorem 45.** For every constant  $\gamma \in (0, 1)$  every  $k = k(n) \geq \Omega(\log^2(n)), \varepsilon = \varepsilon(n) \geq 2^{-O(\sqrt{k(n)})}$ , there exists a strong  $(k, \varepsilon)$  extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$  computable in  $\text{NC}^1$ , with  $r(n) = O(\log(n/\varepsilon))$ ,  $m(n) = (1 - \gamma)k(n)$ .

### 6.1 Condenser in $\text{NC}^1$

The first component in our construction is the condenser from [19]. A simplified version of their result is as follows:

► **Lemma 46** (condenser from [19]). For every  $k = k(n) \geq \Omega(\log^2(n)), \varepsilon = \varepsilon(n) \geq n \cdot 2^{-\sqrt{k(n)}/1024}$ , There exists  $m(n) \leq \frac{3}{2}k(n)$  and a function  $\text{Cond} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$  with  $r \leq 4 \log(\frac{n}{\varepsilon})$  such that  $\text{Cond}$  is a  $(k, k + r, \varepsilon)$ -condenser.

## 69:18 Randomness Extractors in $\text{AC}^0$ and $\text{NC}^1$ : Optimal up to Constant Factors

The condenser takes the input  $x$  as the representation of a degree  $\leq d = O(\frac{n}{\log q})$  polynomial over  $\mathbb{F}_q$  for some prime  $q \geq d, \log q \geq r$ . Denote the degree  $\leq d$  polynomial as  $f$ .

The condenser takes the seed  $y$  as a point in  $\mathbb{F}_q$ . Then the output is defined as:

$$\text{Cond}(x, y) = (y, f(y), f^{(1)}(y), \dots, f^{(s)}(y)) \quad (14)$$

for some  $s = s(n) \leq \frac{m(n)}{r(n)}$ .  $f^{(i)}$  denotes the  $i$ -th formal derivative of  $f$ .

To apply the condenser, we need to transform a source on  $\{0, 1\}^n$  to a source on  $\mathbb{F}_q$  and transform it back for the output. We use division to do the transformation, which is computable in  $\text{NC}^1$ .

The condenser itself requires two sorts of operations: polynomial evaluation and formal derivative. Denote  $f(x) = \sum_{i=0}^d a_i x^i$ . Then  $f^{(j)}(x) = \sum_{i=0}^d \frac{i!}{(i-j)!} a_i x^{i-j}$ . There are at most  $d^2$  such coefficients  $\frac{i!}{(i-j)!}$ , which can be precomputed and stored in the circuit. The multiplication of  $a_i$  and  $\frac{i!}{(i-j)!}$  can be done in  $\text{NC}^1$ . Therefore, the formal derivative is computable in  $\text{NC}^1$ .

The polynomial evaluation consists of three operations: calculating the powering  $x^{i-j}$ , multiplication and summation. The powering can be implemented with two steps:  $O(n)$ -th powering and division by  $q$ , which are computable in  $\text{NC}^1$  according to [4]. The multiplication and iterated summation are both in  $\text{NC}^1$ .

Putting it together, we can obtain the following lemma:

► **Lemma 47.** *The condenser from Lemma 46 is computable in  $\text{NC}^1$ .*

Regard the output of the condenser as  $(X_1, X_2)$ ,  $|X_1| = |X_2| = \frac{1}{2}m(n)$ . By Lemma 15,  $(X_1, X_2)$  is  $\varepsilon(n)$ -close to a  $(\frac{1}{2}m(n), \frac{1}{6}m(n), \frac{1}{2}m(n), \frac{1}{6}m(n))$ -source.

### 6.2 Error Reduction in $\text{NC}^1$

After condensing, we only need to handle an input  $(n, k)$  source  $X$  over  $\{0, 1\}^n$  with constant entropy rate  $\delta = \frac{k}{n}$ . To extract a seed of length  $O(\log n \log(n/\varepsilon))$ , we use almost the same procedure as in Section 4 despite some minor changes.

For the first step to convert the source to a somewhere source, we use the same extractors as in Section 4. We apply the extractors in parallel for  $t = \frac{\log n}{\log(1/\varepsilon)} = O(\sqrt{k})$  times. Then the output is  $\varepsilon$ -close to a somewhere  $(m_0, m_0 - \log(t))$ -source, where  $m_0 = \Omega(k)$ .

For the second step, we still apply the  $(m_0 - \log(t), 0.9m_0, \varepsilon)$ -merger from Corollary 26 to the output of the first step as in Section 4. Since  $\varepsilon \geq 2^{-O(\sqrt{k})}$  and  $t = \text{poly}(k)$ , the merger is computable in  $\text{NC}^1$ .

After applying the merger, we obtain a block-source with exponentially increasing length. We require a modification to Theorem 7 for the  $\text{NC}^1$  setting. The main difference is that the error is now  $2^{-O(\sqrt{k})}$  instead of  $2^{-\text{poly}(\log n)}$ . Also we setup the block length  $m_j = 3^j \cdot 10 \log \frac{n}{\varepsilon}$ ,  $j \in [l]$ , where  $l$  can still be  $O(\log n)$ , since  $\varepsilon = 2^{-O(\sqrt{k})}$ .

To extract from the block source, we require the following extractor in  $\text{NC}^1$ .

► **Lemma 48.** *For every constant  $\delta \in (0, 1]$  and every  $\varepsilon = 2^{-O(n)}$ , there exists an explicit  $(\delta n, \varepsilon)$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$  in  $\text{NC}^1$ , where  $d = O(\log(n/\varepsilon))$ ,  $m = \Theta(\log(n/\varepsilon))$ .*

We use the sample-then-extract technique with leftover hash lemma to construct the extractor. Detailed analysis could be found in the full version.

Using the extractor to extract from the block source as in Section 4, we obtain a seed of length  $O(\log n \log(n/\varepsilon))$ .

One can use the iteration of Section 4 to stretch the output to  $O(\log^2 n \log(n/\varepsilon))$ .

This gives us the following lemma:

► **Lemma 49.** *For every  $\delta \in (0, 1)$ ,  $k = \delta n$ ,  $\varepsilon = \varepsilon(n) = 2^{-O(\sqrt{k})}$ , there exists a strong  $(k(n), \varepsilon(n))$ -extractor  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$  computable in  $\text{NC}^1$ , with  $r(n) = O(\log(n/\varepsilon))$ ,  $m(n) = O(\log^2(n) \log(n/\varepsilon))$ .*

### 6.3 Improved Trevisan's Extractor in $\text{NC}^1$

With the seed of length  $O(\log^2 n \log(n/\varepsilon))$ , We apply the extractor from [26] to the first block of the block source. Their extractor could be implemented in  $\text{NC}^1$

► **Theorem 50 (Improved Trevisan's Extractor [26]).** *For every  $k = k(n)$ ,  $\varepsilon = \varepsilon(n)$ , there are explicit  $(k(n), \varepsilon(n))$ -extractors  $\text{EXT}_{\text{Trevisan}} : \{0, 1\}^n \times \{0, 1\}^{r(n)} \rightarrow \{0, 1\}^{m(n)}$  with  $r(n) = O(\log^2(n) \log(n/\varepsilon))$  and  $m(n) = \Omega(k(n))$ .*

*Moreover, the extractor  $\text{EXT}_{\text{Trevisan}}$  is computable in  $\text{NC}^1$ .*

### 6.4 Putting it together

Now we can prove Theorem 45.

**Proof of Theorem 45.** Take  $X$  as the input source. Let  $\text{Cond} : \{0, 1\}^n \times \{0, 1\}^{r_1(n)} \rightarrow \{0, 1\}^{m(n)}$  be the  $(k, k + r_1, \varepsilon/4)$ -condenser from Lemma 46. Take  $(X_1, X_2) = \text{Cond}(X, U_1)$ , where  $U_1$  is the seed of length  $r_1 = O(\log(n/\varepsilon))$ . By Lemma 15,  $(X_1, X_2)$  is  $\varepsilon/2$ -close to a  $(\frac{1}{2}m(n), \frac{1}{6}m(n), \frac{1}{2}m(n), \frac{1}{6}m(n))$ -source.

For  $X_2$ , apply the  $(\frac{1}{6}m(n), \varepsilon/4)$ -strong extractor  $\text{EXT}_1$  from Lemma 49 with seed  $U_2$  of length  $r_2 = O(\log(n/\varepsilon))$ . The output is  $Y = \text{EXT}_1(X_2, U_2)$  of length  $O(\log^2(n) \log(n/\varepsilon))$ .

For  $X_1$ , apply the  $(\frac{1}{2}m(n), \varepsilon/4)$ -extractor  $\text{EXT}_{\text{Trevisan}}$  from Theorem 50 with seed  $Y$ , which outputs a distribution  $W$  of length  $\Omega(k)$ .

By the property of  $\text{EXT}_1$ ,  $(X_1, Y)$  is  $3\varepsilon/4$ -close to  $(X_1, Y')$  such that  $Y'$  is a independent uniform distribution. Therefore  $W = \text{EXT}_{\text{Trevisan}}(X_1, Y)$  is  $\varepsilon$ -close to uniform.

The extractor  $\text{EXT}$  is defined as  $\text{EXT}(X, U_1, U_2) = W$ .  $\text{Cond}, \text{EXT}_1, \text{EXT}_{\text{Trevisan}}$  are all computable in  $\text{NC}^1$ . Therefore,  $\text{EXT}$  is computable in  $\text{NC}^1$ . ◀

## 7 Entropy lower bound for $\text{AC}^0$ dispersers

In the context of  $\text{AC}^0$  computation, not all sources are extractable. A well-known result of [10] shows that extracting even one bit of randomness is impossible for sources with entropy less than  $\frac{n}{\text{poly}(\log n)}$ . Similar result from [7] shows that extracting randomness with error less than  $2^{-\text{poly}(\log n)}$  is impossible for  $\text{AC}^0$  extractors.

In this section, we will extend the bound from extractors to dispersers. Dispersers are functions that take a source and a seed and output a distribution like extractors. The only difference is that the output distribution is not necessarily uniform, but rather supported on all but a small fraction of the codomain. We will show that strong  $\text{AC}^0$  dispersers for sources with entropy less than  $\frac{n}{\text{poly}(\log n)}$  do not exist.

► **Definition 51 (Disperser).** *A function  $\text{Disp} : \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}^m$  is a  $(k, \varepsilon)$ -disperser if for every  $k$ -source  $X$  on  $\{0, 1\}^n$  and uniformly random variable  $Y$  on  $\{0, 1\}^r$ ,  $|\text{Supp}(\text{Disp}(X, Y))| \geq (1 - \varepsilon)2^m$ .*

*Furthermore,  $\text{Disp}$  is a strong  $(k, \varepsilon)$ -disperser if for every  $k$ -source  $X$  on  $\{0, 1\}^n$  and uniformly random variable  $Y$  on  $\{0, 1\}^r$ ,  $|\text{Supp}(Y, \text{Disp}(X, Y))| \geq (1 - \varepsilon)2^{r+m}$ .*

We remark that the requirement for  $X$  to have entropy  $\geq k$  can be replaced by a weaker requirement, which only requires  $\text{Supp}(X) \geq 2^k$ , without changing the definition.

Our proof is based on the new switching lemma for  $AC^0$  circuits by Rossman in [28]. Their original result says that every  $AC^0$  circuit can be reduced to a decision tree of arbitrary depth under a random restriction for all but a small fraction of the inputs. By restricting the inputs for the second time, it is reduced to a constant function.

► **Definition 52 (Restrictions).** A restriction  $\rho$  is a string on  $\{0, 1, *\}^n$ . We denote the application of  $\rho$  to  $x \in \{0, 1\}^n$  by  $\rho \circ x$ , which is defined as:

$$(\rho \circ x)_i = \begin{cases} \rho_i & \text{if } \rho_i \neq *, \\ x_i & \text{if } \rho_i = *. \end{cases} \quad (15)$$

The restriction on a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  is defined as:

$$f|_\rho(x) = f(\rho \circ x). \quad (16)$$

We use  $R_p$  to denote the independent uniform random restriction with star probability  $p$ . That is, for every  $i \in [n]$ ,  $\Pr[R_p(i) = *] = p$ ,  $\Pr[R_p(i) = 0] = \Pr[R_p(i) = 1] = \frac{1-p}{2}$ .

The switching lemma for  $AC^0$  circuits is stated as follows:

► **Lemma 53 (Switching Lemma for  $AC^0$  circuits [28]).** For every  $\delta \in (0, 1)$ ,  $d > 0$ ,  $s = s(n)$ , there exists  $p = \frac{\delta}{\Theta(\log s)^{d-1}}$  such that for every  $AC^0$  circuit  $C$  of size  $s$  and depth  $d$ ,

$$\Pr_{\rho \sim R_p} [C|_\rho \text{ is not constant}] \leq \delta. \quad (17)$$

The following negative result for strong dispersers directly follows from the switching lemma.

► **Theorem 54.** For every  $d > 0$ ,  $s = s(n)$ , every constant  $\delta \in (0, 1)$ , if  $C : \{0, 1\}^n \times \{0, 1\}^r \rightarrow \{0, 1\}$  is a  $(k, \frac{1}{2} - \delta)$ -disperser that can be computed by a non-uniform  $AC$  circuit of size  $s$  and depth  $d$ , then  $k \geq \Theta(\frac{\delta n}{\log^{d-1} s})$ .

## 8 Open Questions

We mention the following open questions.

- For extractors in  $AC^0$ , can we further improve the circuit depth? The current depth is  $O(a + c + 1)^2$ . Is it possible to be linear in  $a + c + 1$ , while maintaining other parameters to be roughly the same?
- For extractors in  $NC^1$ , can we improve the plausible range of  $k$  and  $\varepsilon$ ? For example is it possible to give an  $NC^1$  construction that can work for all  $k, \varepsilon$ , matching the parameters in [13]?
- Some components of our  $NC^1$  computable extractors are actually in  $AC^0$ [2]. Is it possible to give an extractor in  $AC^0$ [2], with parameters optimal up to constant factors?
- For weak dispersers, we do not have a similar negative result to that of Section 7. The reason is that a single good seed in the seed space can make the disperser good enough, regardless of other seeds. So it remains an open question whether weak dispersers can be constructed in  $AC^0$ , specifically for sources with entropy less than  $\frac{n}{\text{poly}(\log n)}$ .

## References

- 1 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- 2 Z. Bar-Yossef, O. Reingold, R. Shaltiel, and L. Trevisan. Streaming computation of combinatorial objects. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 165–174, 2002.
- 3 David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 1–5, 1986.
- 4 Paul Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986. doi:10.1137/0215070.
- 5 J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.
- 6 Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 125–136. IEEE, 2021. doi:10.1109/FOCS52979.2021.00021.
- 7 Kuan Cheng and Xin Li. Randomness extraction in ac0 and with small locality. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM) 2018*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 8 Gil Cohen, Dean Doron, Ori Sberlo, and Amnon Ta-Shma. Approximating iterated multiplication of stochastic matrices in small space. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 35–45. ACM, 2023. doi:10.1145/3564246.3585181.
- 9 Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. Extensions to the Method of Multiplicities, with Applications to Kakeya Sets and Mergers. *SIAM Journal on Computing*, 42(6):2305–2328, January 2013. doi:10/f5msx6.
- 10 Oded Goldreich, Emanuele Viola, and Avi Wigderson. On randomness extraction in AC0. In *Proceedings of the 30th Conference on Computational Complexity, CCC '15*, pages 601–668, Dagstuhl, DEU, June 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- 11 Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties (preliminary version) a quality-size trade-off for hashing. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 574–584, 1994.
- 12 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 440–449. ACM, 2007.
- 13 Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *Journal of the ACM*, 56(4):1–34, June 2009. doi:10/ctbzhm.
- 14 Alexander Healy and Emanuele Viola. Constant-Depth circuits for arithmetic in finite fields of characteristic two. In *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science, STACS'06*, pages 672–683, Berlin, Heidelberg, February 2006. Springer-Verlag. doi:10/df5dfs.
- 15 Alexander D Healy. Randomness-efficient sampling within nc1. *Computational Complexity*, 17(1):3–37, 2008.
- 16 Russel Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 12–24, 1989.
- 17 Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 538–545. IEEE, 1995.
- 18 Russell Impagliazzo and Avi Wigderson. P=BPP unless E has sub-exponential circuits: Derandomizing the xor lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.



- 19 Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM) 2022*, volume 245 of *LIPICs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.APPROX/RANDOM.2022.12.
- 20 Chi-Jen Lu, Omer Reingold, Salil Vadhan, and Avi Wigderson. Extractors: Optimal up to Constant Factors. *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, June 2003. doi:10/bw2j9d.
- 21 Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58:148–173, 1999.
- 22 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.
- 23 Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996.
- 24 Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors and depth-two superconcentrators. *Siam Journal on Discrete Mathematics*, 13:2–24, 2000.
- 25 Ran Raz, Omer Reingold, and Salil P. Vadhan. Error reduction for extractors. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 191–201. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814591.
- 26 Ran Raz, Omer Reingold, and Salil P. Vadhan. Extracting all the randomness and reducing the error in trevisan’s extractors. *J. Comput. Syst. Sci.*, 65(1):97–128, 2002. doi:10.1006/JCSS.2002.1824.
- 27 Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000*, pages 22–31. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892008.
- 28 Benjamin Rossman. An entropy proof of the switching lemma and tight bounds on the decision-tree size of  $AC^0$ , 2017. URL: <https://users.cs.duke.edu/~br148/logsize.pdf>.
- 29 Ronen Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of the European Association for Theoretical Computer Science*, 77:67–95, 2002.
- 30 Ronen Shaltiel. An introduction to randomness extractors. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming*, 2011.
- 31 A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28:1433–1459, 1999.
- 32 Amnon Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 276–285, 1996.
- 33 Amnon Ta-Shma. Almost optimal dispersers. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, 1998*, pages 196–202. ACM, 1998. doi:10.1145/276698.276736.
- 34 Amnon Ta-Shma and Christopher Umans. Better condensers and new extractors from parvaresh-vardy codes. In *2012 IEEE 27th Conference on Computational Complexity*, pages 309–315. IEEE, 2012.
- 35 Roei Tell. Improved bounds for quantified derandomization of constant-depth circuits and polynomials. *computational complexity*, 28(2):259–343, 2019.
- 36 Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001. doi:10.1145/502090.502099.
- 37 Salil Vadhan. The unified theory of pseudorandomness. *SIGACT News*, 38, 2007.
- 38 Salil Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.
- 39 Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. *computational complexity*, 13(3-4):147–188, 2005.
- 40 Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. *Combinatorica*, 19(1):125–138, 1999.
- 41 David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11(4):345–367, December 1997. doi:10/cr8kht.