Electronic Colloquium on Computational Complexity, Report No. 40 (2024)

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

Kuan Cheng * Ruiyang Wu †

Abstract

We study extractors computable in uniform AC^0 and uniform NC^1.

For the AC^0 setting, we give a construction such that for every $k \geq n/\operatorname{poly}\log n, \varepsilon \geq 2^{-\operatorname{poly}\log n}$, it can extract $(1-\gamma)k$ randomness from an (n,k) source for an arbitrary constant γ, with seed length $O(\log \frac{n}{\varepsilon})$. The output length and seed length are optimal up to constant factors matching the parameters of the best polynomial time construction such as [GUV09]. The range of k and ε almost meets the lower bound in [GVW15] and [CL18]. We also generalize the main lower bound of [GVW15] for extractors in AC^0, showing that when $k < n/\operatorname{poly}\log n$, even strong dispersers do not exist in AC^0.

For the NC^1 setting, we also give a construction with seed length $O(\log \frac{n}{\varepsilon})$ and a small constant fraction entropy loss in the output. The construction works for every $k \geq O(\log^2 n), \varepsilon \geq 2^{-O(\sqrt{k})}$. To our knowledge the previous best NC^1 construction is Trevisan's extractor [Tre01] and its improved version[RRV02] which have seed lengths $\operatorname{poly}\log \frac{n}{\varepsilon}$.

Our main techniques include a new error reduction process and a new output stretch process based on low depth circuits implementations for mergers from [DKSS13], condensers from [KT22] and somewhere extractors from [Ta-98].

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 rsource) 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 [NZ96] introduced the definition of seeded extractor, where the extractor has a short uniform random seed as an extra input. Specifically, a function $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is defined to be a strong (k,ε)-extractor, if for every source X with min-entropy k,

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

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

$$\|\textsc{Ext}(X, U_d) - U_m\| \leq \varepsilon.$$

As a fundamental pseudorandom construction, extractors are closely related to other pseudorandom objects and also have various applications in computational complexity, combinatorics,

*School of Computer Science, Peking University. ckkcdh@hotmail.com
†School of Computer Science, Peking University. 2301111967@stu.pku.edu.cn

ISSN 1433-8092

algorithm design, information theory, and cryptography. See surveys [NT99][Sha02] [Vad07] [Sha11] [AB09] [Vad12].

Optimizing extractor constructions aims to get, for every k and ε, 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 $k \le n/2$, due to the lower bound by [RTS00]. After [NZ96], a long line of work has been done to seek explicit extractors with parameters close to the existential bounds [WZ99, SZ99, GW94, Ta-96, Zuc97, RRV99, NT99, RSW00, Tre01, Ta-98, RRV02, LRVW03, GUV09, TSU12, DKSS13, KT22]. Among them, [GUV09] 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 $d = \log n + O(\log k \cdot \log(k/\varepsilon))$. [TSU12] and [KT22] can also achieve the same parameters by replacing the condenser in [GUV09] with their condenser versions. On the other hand, [TSU12] and [DKSS13] achieve subconstant entropy loss $m = (1-1/\operatorname{poly}\log n)k$, $d = O(\log n)$ when $\varepsilon \ge 1/2^{\log^\beta n}$ 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. We specifically focus on AC^0 and NC^1. AC^0 is the class of all uniform polynomial-size circuits of 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 polynomial-size circuits of $O(\log n)$ depth, with NOT, AND, and OR gates, where AND, OR gates have fan-in 2.

Viola [Vio05] raised the question on extractor construction in AC^0. Goldreich and Wigderson [GVW15] generalize the negative result of [Vio05], showing that for every constant D, there exists a polynomial p such that as long as $k \le n/p(\log n)$, no extractor in AC^0 with depth D extract even 1 bit with a constant error, no matter how long the seed is. This rules out the possibility for the case that $k = n/\log^{\omega(1)} n$. For the case $k \ge n/\operatorname{poly}\log n$, [GVW15] gives a strong extractor in AC^0 that has an output length linear to the seed length. Lately Cheng and Li [CL18] give a construction that significantly improves the parameters, achieving $d = O\left((\log n + \frac{\log(n/\varepsilon)\log(1/\varepsilon)}{\log n})\frac{n}{k}\right), m = (1-\gamma)\,k$, for any constant γ and any $\varepsilon \ge 2^{-\operatorname{poly}\log n}$. They also show that ε has to be at least $2^{-\operatorname{poly}\log n}$ for AC^0 extractors.

For extractors in NC^1, unlike the AC^0 case, there are no known lower bounds for k or ε. Indeed the extractor based on universal hash functions [CW79], argued by the leftover hash lemma [ILL89], can achieve an arbitrary ε and k. It can be realized in NC^1 since there are simple linear function constructions for such hash functions. Trevisan's extractor [Tre01], and its improved version [RRV02] 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 [Zuc97]. Healy [Hea08] constructs a sampler in NC^1. However if one simply applies the transformation of [Zuc97] 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 NC^1 with better parameters for arbitrary k and ε.

1.1 Our results

Our main positive result is an AC^0 computable extractor with parameters optimal up to constant factors.

Theorem 1.1. *For every constant $a, c > 0, \gamma \in (0,1)$, every $k \ge \frac{n}{\log^a(n)}, \varepsilon \ge 2^{-\log^c(n)}$, there exists an explicit (k,ε)-strong extractor* $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *in* 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 AC^0 constructions [CL18], which achieves $d = O\left(\left(\log n + \frac{\log(n/\varepsilon)\log(1/\varepsilon)}{\log n}\right)\log^a n\right)$. Also, notice that there are lower bounds for k and ε in the AC^0 construction setting, i.e. k has to be at least $n/\operatorname{poly}\log n$ by [GVW15] and ε has to be $2^{-\operatorname{poly}\log n}$ by [CL18]. Thus roughly in the plausible range for k and ε, we achieve parameters optimal up to constant factors.

Our method can also be used to give NC^1 computable extractors.

Theorem 1.2. *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,ε) *extractor* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *computable in* NC^1*, with* $d = O(\log(n/\varepsilon))$*,* $m = (1-\gamma)k$*.*

To our knowledge, the previous best known NC^1 construction is the improved Trevisan's extractor from [RRV02], which has seed length $O(\log^2 n \log \frac{n}{\varepsilon})$, for all k, ε. Our parameters are optimal up to constant factors for ranges of k, ε as stated.

Our negative result generalizes the previous entropy parameter lower bound by [GVW15] for strong extractors in AC^0 to strong dispersers in AC^0.

Theorem 1.3. *For every* $d, s > 0$*, every constant* $\delta \in (0,1)$*, if* $C : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}$ *is a* $(k, \frac{1}{2} - \delta)$*-disperser that can be computed by an* AC *circuit of size* s *and depth* d*, then* $k \geq \Theta(\frac{\delta n}{\log^{d-1} s})$*.*

1.2 Technique Overview

1.2.1 Extractor in AC^0

Our AC^0 computable extractor is constructed by three main parts.

Merger in AC^0 In this part, we show that any somewhere high-entropy source X can be merged to be a high-entropy source in AC^0 under a restricted setting of parameters. Recall that $X = (X_1, \ldots, X_\Lambda)$ 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', ε) merger is a function $\mathrm{Merge} : \{0,1\}^{n\Lambda} \times \{0,1\}^d \to \{0,1\}^m$, such that for any input somewhere (n,k) source X, $\mathrm{Merge}(X, U)$ has entropy k'. [DKSS13] 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 $\operatorname{poly}\log n$, δ is a small constant, and error $\varepsilon = 2^{-\operatorname{poly}\log n}$, then this merger can be computed in AC^0. To show this, we notice that under this parameter setting, the computation of [DKSS13] is over a finite field F_q, $q = 2^d = 2^{\operatorname{poly}\log n}$. The computation only involves three operations: (1) the summation of $\operatorname{poly}\log n$ elements; (2) the powering y^i where $y \in \mathbb{F}_q, i = \operatorname{poly}\log n$; (3) the product of a constant number of field elements. (1) is clearly in AC^0 since it is actually the summation of $\operatorname{poly}\log n$ bits, while (2) and (3) are shown to be in AC^0 by [HV06]. Notice 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.

Error Reduction This part gives 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 [CL18]. 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, ε_0) extractor $\mathrm{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ where $\varepsilon_0 = 1/n$, $d_0 = O(\log n), m_0 = O(k^2/n)$, which is achieved in [CL18]. 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 procedure in three steps along with their arguments:

1. Apply Ext_0 to X for $t = \frac{\log(n/\varepsilon)}{\log n}$ times, using independent seeds, outputting $Y_1, Y_2, \ldots, Y_t$ respectively.

   Notice that by the error reduction of [RRV99], 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 $(Y_1, \ldots, Y_t)$ 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 [CL18]. But we differ from [CL18] 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 $Y_i = (Y_{i,1}, Y_{i,2}, \ldots, Y_{i,l})$, where we let $m_j = |Y_{i,j}| = m_0^{0.1} \cdot 3^j$. Denote $Y_{i,1\ldots j}$ as the first j blocks of Y. Then for each j, let $B_j = (Y_{1,1\ldots j}, Y_{2,1\ldots j}, \ldots, Y_{t,1\ldots j})$, 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 $(Z_1, Z_2, \ldots, Z_l)$ as a block source and extract by a standard method.

   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 $M_j = m_1 + m_2 + \cdots + m_j$. Also, as $t = \operatorname{poly}\log n$, the merger can be implemented in AC^0. As the 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 $(Z_1, Z_2, \ldots, Z_l)$ is indeed very close to a block source that has a constant conditional entropy rate. The output length is $O(\log n \log \frac{n}{\varepsilon})$ 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. Use samplings to get a block source with a constant number of blocks. Apply a standard extraction, e.g. the method in [CL18], for the block source to get output length $\Omega(\log^b n \cdot \log \frac{n}{\varepsilon})$ for a given arbitrary constant b.

   Notice that we have to extract these blocks one by one from the last to the first, so the depth has a factor $O(b)$ blow-up. But as long as b is a constant, this is still in AC^0. The seed length is $O(\log \frac{n}{\varepsilon})$ as we only need to pay the seed for the sampling and the extraction of the last block.

Output Stretch The last part is a new output stretch procedure for AC^0 computable extractors. Compared to the one in [CL18], the new method attains an output length $(1-\gamma)k$ with a seed length $O(\log \frac{n}{\varepsilon})$. 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 $\operatorname{poly}\log \frac{n}{\varepsilon}$ length output, and then use it to extract the first block by applying the main extractor from [CL18]. 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 [DKSS13]. We describe the following 3 steps to reduce the hard case to the easy case:

1. Use Ta-shma's somewhere-block-source converter [Ta-98] 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 $\Lambda = \log^{2a} n$. [Ta-98] 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 part 2 for the second block and then use the output as a seed to extract the first block by the extractor in [CL18].

   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 $\operatorname{poly}\log \frac{n}{\varepsilon}$. This is enough to extract a constant fraction of entropy i.e. $\Omega(k)$ from the first block by [CL18]. 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 $\operatorname{poly}\log n$ segments, $\varepsilon = 2^{-\operatorname{poly}\log n}$, and the entropy rate attained is a constant, it holds that the merger is in AC^0, with a seed length $O(\log \frac{n}{\varepsilon})$. 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 [KT22]. Regard the output as (Y_1, Y_2) such that Y_1, Y_2 have a equal length.

   Compared to the condenser in [GUV09], the condenser in [KT22] can only work for $k \geq \Omega(\log^2(n)), \varepsilon \geq 2^{-O\left(\sqrt{k(n)}\right)}$, However, the advantage is that it is computable in NC^1. Recall that the [KT22] $(k, k+d, \varepsilon)$ condenser can actually be viewed as $\mathrm{Cond} : \mathbb{F}_q^n \times \mathbb{F}_q \to \mathbb{F}_q^m$. 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 $\mathbb{F}_q, \log q = O(\log \frac{n}{\varepsilon})$. The seed is a random element of $\mathbb{F}_q$. The computation is actually $\mathrm{Cond}(f, u) = (u, f(u), f^{(1)}(u), \ldots, f^{(m)}(u))$. 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 [BCH86]. 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 an output 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 $O(\log \frac{n}{\varepsilon})$.

3. Apply the improved Trevisan's extractor [RRV02] 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 [Tre01] and its improved version [RRV02] 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 $O(\log \frac{n}{\varepsilon})$.

1.2.3 A lower bound for AC^0 computable dispersers

Our lower bound follows from the improved switching lemma in [Ros]. Assume $\mathrm{Disp} : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}$ 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 $C_y := \mathrm{Disp}(\cdot, y)$. 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 ρ sampled from R_p, the function $C_y|_\rho$ is defined to be a function such that if ρ_i is 1 or 0 then fix the i-th input to be ρ_i, otherwise leave it unfixed, and then apply C_y on this modified input. The switching lemma from [Ros] 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 δ is a constant, with probability at least $1 - 2^{-O(pn)} > 1-\delta$, the number of stars in ρ is at least $p/2$ fraction. By a union bound and an averaging argument, one can show that there exists a ρ 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 ρ 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, $\mathrm{Supp}(C_y|_\rho(X)) \leq 2$ as C_y only has 1 bit output. This implies that $|\mathrm{Supp}(U, \mathrm{Disp}(X,U))|$ 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 main constructions for extractors in AC^0 from [CL18].

Theorem 2.1 ([CL18])**.** *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* $\mathrm{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *computable in* AC^0 *with depth $O(a)$, where $d = O(\log n), m \geq \Theta(\delta k)$.*

Theorem 2.2 ([CL18] for small entropy)**.** *For every constant $\gamma \in (0,1), a, c \geq 1$, every $k = \delta n = \Theta(n/\log^a n), \varepsilon = 2^{-\Theta(\log^c n)}$, there exists an explicit (k,ε)-extractor* $\mathrm{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *computable in* 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 AC^0.

Theorem 2.3 ([CL18] Sample-then-extract)**.** *For every constant $\delta \in (0,1], c \geq 1$ and every $\epsilon = 2^{-\log^c n}$, there exists an explicit $(\delta n, \epsilon)$-extractor* $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *in* AC^0 *with depth $O(c)$, where $d = O(\log(n/\varepsilon))$, $m = \Theta(\log(n/\varepsilon))$.*

The leftover hash lemma is also needed in our construction.

Lemma 2.4 (Leftover Hash Lemma [ILL89])**.** *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 $U \circ \textsc{Ext}(X, U)$ is at distance at most $1/2^{\Delta}$ to uniform distribution where the function* $\textsc{Ext}$: $\{0,1\}^{n'} \times \{0,1\}^d \to \{0,1\}^m$ *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.

Averaging sampler is also an important ingredient in our construction.

Definition 2.5 (Averaging Sampler)**.** *A (γ, ε)-averaging sampler is a function* $\mathrm{Samp} : \{0,1\}^r \to [n]^t$ *such that for every sequence of functions $f_i : [n] \to [0,1], i \in [t]$, $\mu_i = \mathbf{E}_{x \in [n]}[f_i(x)]$, it holds that*

$$\Pr_{s \leftarrow \mathrm{Samp}(U_r)}[\frac{1}{t} \sum_{i \in [t]} |f_i(s_i) - \mu_i| \geq \varepsilon] \leq \gamma.$$

Lemma 2.6. *[Zuc97] If there is an efficient $(\delta n, \varepsilon)$-extractor with seed length d, input length n, output length m, then there is an efficient $(2^{1-(1-\delta)n}, \varepsilon)$-sampler with input length n, length of each sample m, and 2^d number of samples.*

Now we give the following sampler.

Theorem 2.7. *For any n, any $\gamma = 1/\mathrm{poly}\, n$, $\varepsilon \geq 1/\mathrm{poly}\log n$, there exists an (γ, ε)-averaging sampler* $\mathrm{Samp} : \{0,1\}^r \to [n]^t$ *with seed length $r = \log n + O(\log(1/\gamma))$ and $t = \mathrm{poly}(r, \varepsilon)$ which can be computed by NC^1 circuits of size $\mathrm{poly}\log n$. Furthermore, this sampler can be computed by AC^0 circuits of $\mathrm{poly}\, n$.*

To prove the theorem we need to use Trevisan's extractor for the following version.

Theorem 2.8 ([Tre01] for polynomial small error)**.** *For every constant $\gamma \in (0,1)$, every $k \geq n^{\Omega(1)}$, $\varepsilon = 1/\mathrm{poly}\, n$, there exists an explicit (k, ε)-extractor* $\textsc{Ext}_0 : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *computable in $\mathrm{AC}^0[2]$, where $d = O(\log n), m \geq (1-\gamma)k$.*

Now we can prove Theorem 2.7.

Proof of Theorem 2.7. Let $\textsc{Ext}_0 : \{0,1\}^r \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ be a $(\delta r, \varepsilon)$-extractor from Theorem 2.8, where $m_0 = \log n$, $\delta = 1/2$, r be s.t. $\gamma = 2^{1-(1-\delta)r}$, $d_0 = O(\log r) = O(\log\log n)$.

By Lemma 2.6, this is a desired sampler. The furthermore part follows directly by Lemma 2.11. □

Error reduction for extractors has been extensively studied in previous works. We recall the following key ingredient in the classic error-reducing technique [RRV99].

Lemma 2.9 (G_x Property [RRV99])**.** *Let* $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a (k, ϵ)-extractor with $\epsilon < 1/4$. 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 drawn a y from* $\text{Ext}(X, G_X)$*, then with probability at least* $1-2^{-t}$, $\Pr[\text{Ext}(X, G_X) = y] \leq 2^{-(m-1)}$. *Here* $\text{Ext}(X, G_X)$ *is obtained by first sampling x according to X, then choosing r uniformly from G_x, and outputting* $\text{Ext}(x, r)$.
- $\text{Ext}(X, G_X)$ *is within distance at most 2^{-t} from an $(m, m - O(1))$-source. 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 2.10 (folklore)**.** *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* $\mathrm{poly}(n)$ *size.*

Proof. An AC^0 circuit with $O(1)$ depth can compute the parity of $\log(n)$ bits. Therefore, calculating the parity of $\log^a(n)$ bits is reducible to $\log^{a-1}(n)$ bits parity with $O(1)$ depth circuit. The lemma follows by induction. □

Lemma 2.11 ([GGH+07])**.** *For every $c \in \mathbb{N}$, every integer $l = \Theta(\log^c n)$, if the function $f_l : \{0,1\}^l \to \{0,1\}$ can be computed by circuits of depth $O(\log l)$ and size* $\mathrm{poly}(l)$*, then it can be computed by AC^0 circuits of depth $c+1$, size* $\mathrm{poly}(n)$.

A key part of our argument considers block sources. Recall the chain rule for min-entropy.

Definition 2.12 (block source)**.** *Let $X = (X_1, \ldots, X_l)$ such that each X_i is distributed on $\{0,1\}^{n_i}$. We say X is a $(n_1, k_1, n_2, k_2, \ldots, n_l, k_l)$-block source if for every $i \in [l]$ and $(x_1, \ldots, x_{i-1}) \in \{0,1\}^{n_1+\cdots+n_{i-1}}$, $X_i|_{X_1=x_1,\ldots,X_{i-1}=x_{i-1}}$ is a (n_i, k_i)-source.*

Lemma 2.13. *Fix $t \in \mathbb{N}$ and $k, s, n, n_1, \ldots, n_k \in \mathbb{N}$ such that $n_1+\cdots+n_k = n$. Let $X = (X_1, \ldots, X_l)$ 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 $(X_1, \ldots, X_l)$ is $l \cdot 2^{-s}$-close to a $(n_1, n_1 - k, n_2, n_2 - k - s, \ldots, n_l, n_l - k - s)$-source.*

Proof. We prove this by induction.

For $l = 2$, we have $X = (X_1, X_2)$. Assert that $\Pr[X_1 = x_1] \leq 2^{n_1-k}$ for every $x_1 \in \{0,1\}^{n_1}$. Suppose not, then there exists $x_1 \in \{0,1\}^{n_1}$ such that $\Pr[X_1 = x_1] > 2^{n_1-k}$. Then there exists $x_2 \in \{0,1\}^{n_2}$ such that $\Pr[X_1 = x_1, X_2 = x_2] > 2^{n_1+n_2-k}$. This contradicts the assumption that X is a k-source.

Fix any $x_1 \in \{0,1\}^{n_1}$, suppose that $X_2|_{X_1=x_1}$ is not a (n_2, n_2-k-s)-source. Then there exists $x_2 \in \{0,1\}^{n_2}$ such that $\Pr[X_2 = x_2 | X_1 = x_1] > 2^{-n_2+k+s}$. Since $\Pr[X_1 = x_1, X_2 = x_2] \leq 2^{-n+k}$, we have $\Pr[X_1 = x_1] \leq 2^{-n_1-s}$. Therefore $\Pr_{x_1 \leftarrow X_1}[X_2|_{X_1=x_1}$ is not a (n_2, n_2-k)-source$] \leq 2^{-s}$. The lemma follows.

For other l, we have $X = (X_1, \ldots, X_l)$. Let $X_2' = (X_1, X_2)$. By the induction hypothesis, we have that $X = (X_2', \ldots, X_l)$ is $(l-1)2^{-s}$-close to a $(n_1+n_2, n_1+n_2-k, \ldots, n_{l-1}, n_{l-1}-k-s)$-source. Denote that source by $Y = (Y_2, \ldots, Y_l)$. The $l = 2$ case shows that Y_2 is ε-close to a $(n_1, n_1-k, n_2, n_2-k-s)$-source (Y_1', Y_2'). Construct random variables $Y_3', \ldots, Y_l'$ such that $(Y_3', \ldots, Y_l')|_{Y_1'=y_1', Y_2'=y_2'}$ has the same distribution as $(Y_3, \ldots, Y_l)|_{Y_1=y_1', Y_2=y_2'}$. Then $(Y_1, Y_2, Y_3, \ldots, Y_l)$ is 2^{-s}-close to $(Y_1', Y_2', Y_3', \ldots, Y_l')$. The distribution $(Y_1', Y_2', Y_3', \ldots, Y_l')$ is a $(n_1, n_1-k, n_2, n_2-k-s, \ldots, n_l, n_l-k-s)$-source. The lemma follows. □

Recall a folklore extraction process for block sources.

Lemma 2.14. *Let $X = (X_1, \ldots, X_l)$ be a $(n_1, k_1, n_2, k_2, \ldots, n_l, k_l)$-block source on $\{0,1\}^n$. Suppose that $\textsc{Ext}_i : \{0,1\}^{n_i} \times \{0,1\}^r \to \{0,1\}^{m_i}$ is a strong (k_i, ε)-extractor for each $i \in [l]$. Let Y be a uniformly random variable on $\{0,1\}^r$. Take $Z = (Z_1, \ldots, Z_l)$ such that $Z_i = \textsc{Ext}_i(X_i, Y)$. Then Z is $l \cdot \varepsilon$-close to uniform.*

Proof. We prove this by induction.

Claim 2.15. *For every $i \in [l]$, $(Y, X_1, \ldots, X_{i-1}, Z_i, \ldots, Z_l)$ is $(l-i+1)\cdot\varepsilon$-close to $(Y, X_1, \ldots, X_{i-1}, U_i, \ldots, U_l)$ where U_j are independent uniformly random variables on $\{0,1\}^{m_j}$ for each $i \le j \le l$.*

Proof of Claim 2.15. We prove this by induction. For $i = l$, $X_l|_{X_1=x_1,\ldots,X_{l-1}=x_{l-1}}$ is a (n_l, k_l)-source. Therefore $(Y, Z_l)|_{X_1=x_1,\ldots,X_{l-1}=x_{l-1}}$ is ε-close to uniform. The claim follows.

For other i, by the induction hypothesis, we have that $(Y, X_1, \ldots, X_{i-1}, Z_{i+1}, \ldots, Z_l)$ is $(l-i)\cdot\varepsilon$-close to $(Y, X_1, \ldots, X_{i-1}, U_{i+1}, \ldots, U_l)$. Since U_j's are independent of $X_1, \ldots, X_{i-1}$, we only need to show that $(Y, X_1, \ldots, X_{i-1}, Z_i)$ is ε-close to $(Y, X_1, \ldots, X_{i-1}, U_i)$. This follows from case $i = l$. The claim follows. □

By Claim 2.15, we have that $(Y, Z_1, \ldots, Z_l)$ is $l \cdot \varepsilon$-close to $(Y, U_1, \ldots, U_l)$. Since $U_1, \ldots, U_l$ are independent uniformly random variables on $\{0,1\}^{m_1+\cdots+m_l}$, the lemma follows. □

Another folklore extraction process is used in stretching the output, we also state it here.

Theorem 2.16. *Let $\textsc{Ext}_1 : \{0,1\}^{n_1} \times \{0,1\}^{m_1} \to \{0,1\}^{m_2}$ be a (k_1, ε_1)-strong extractor, and $\textsc{Ext}_2 : \{0,1\}^{n_2} \times \{0,1\}^r \to \{0,1\}^{m_1}$ be a (k_2, ε_2)-strong extractor. Then the construction*

$$\textsc{Ext}(X_1, X_2, U_r) = \textsc{Ext}_1(X_1, \textsc{Ext}_2(X_2, U_r)) \tag{1}$$

is a $(k_1, k_2, \varepsilon_1 + \varepsilon_2)$-strong extractor.

Proof. Let $X = (X_1, X_2)$ be a (k_1, k_2)-block-source, and U_r be a uniform random distribution on $\{0,1\}^r$. Then $\textsc{Ext}_2(X_2, U_r)$ is ε_2-close to W. W is a uniform random distribution on $\{0,1\}^{m_1}$, independent of both X_1 and U_r. Then $\textsc{Ext}_1(X_1, W)$ is ε_1-close to uniform distribution V on $\{0,1\}^{m_2}$, where V is independent of W and U_r.

Therefore, (U_r, V) is ε_1-close to $(U_r, \textsc{Ext}_1(X_1, W))$. $(U_r, \textsc{Ext}_1(X_1, W))$ is ε_2-close to $(U_r, \textsc{Ext}_1(X_1, \textsc{Ext}_2(X_2, U_r)))$. Therefore, (U_r, V) is $\varepsilon_1 + \varepsilon_2$-close to $(U_r, \textsc{Ext}(X_1, X_2, U_r))$. □

3 Merger in AC^0

In this section, we will examine the merger construction in [DKSS13] to prove that the merger can indeed be implemented in AC^0. Some further modifications are discussed to construct strong mergers for non-uniform sources.

We start by defining the concept of somewhere-(n, k) source.

Definition 3.1 (somewhere-(n, k) source)**.** *Let $X = (X_1, \ldots, X_\Lambda)$ such that each X_i is distributed on $\{0,1\}^n$. We say X is a simple somewhere-(n, k) source with Λ 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.

The 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 3.2 (merger and strong merger)**.** *We say* $\mathrm{Merge} : \{0,1\}^{\Lambda \cdot n} \times \{0,1\}^r \to \{0,1\}^m$ *is a* (k, k', ε)*-merger if for any somewhere-*(n,k) *source* $X = (X_1, \ldots, X_\Lambda)$*, the distribution* $\mathrm{Merge}(X, U_r)$ *is* ε*-close to a* k'*-source. Here* U_r *is a independent uniform random distribution on* $\{0,1\}^r$

Furthermore, if $(U_r, \mathrm{Merge}(X, U_r))$ *is* ε*-close to* (U_r, W)*, we say* Merge *is a strong* (k, k', ε)*-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 [DKSS13], and find that the merger can be implemented in AC^0 if the number of segments is not too large.

Theorem 3.3 (merger in [DKSS13])**.** *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* $\mathrm{Merge} : \{0,1\}^{\Lambda(n) \cdot n} \times \{0,1\}^{r(n)} \to \{0,1\}^n$*. Here* $r(n) = O(\log(\frac{1}{\varepsilon}))$*.*

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

The merger in [DKSS13] 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 $X = (X_1, \ldots, X_\Lambda)$ be a somewhere-uniform-source with Λ segments. Regard each X_i as distributed on $\mathbb{F}_q^K$ with $K = \frac{n}{s}$. Then

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

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

Take $\gamma_1, \ldots, \gamma_\Lambda$ be Λ unique points in $\mathbb{F}_q$. Let $C_1, \ldots, C_\Lambda$ be Λ 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:

$$\mathrm{Merge}(X, y) = \left(\sum_{i=1}^{\Lambda} C_i(y) X_{i,1}, \ldots, \sum_{i=1}^{\Lambda} C_i(y) X_{i,K} \right), \tag{3}$$

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

Lemma 3.4 (merger in [DKSS13])**.** *For any constant* $\delta > 0$*, let* $q \geq \left(\frac{2\Lambda}{\varepsilon}\right)^{1/\delta}$*. Then the function* $\mathrm{Merge} : \mathbb{F}_q^{K \cdot \Lambda} \times \mathbb{F}_q \to \mathbb{F}_q^K$ *is a* $(K \log q, k, \varepsilon)$*-merger, where* $k = (1 - \delta) \cdot K \cdot \log q$*.*

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

Lemma 3.5. *For any constant* $a, c, \delta \in (0,1)$*, let* $\Lambda(n) \leq \log^a, \varepsilon(n) \geq 2^{-\log^c(n)}$*. Define* $r(n) = \min\{s \in \mathbb{N} | s \geq \frac{2}{\delta} \log^c(n), \exists d \in \mathbb{N}, s = 3 \cdot 2^d\}, q(n) = 2^{r(n)}, K(n) = \frac{n}{r(n)}$*. Then the* $(n, \delta n, \varepsilon)$*-merger* $\mathrm{Merge} : \{0,1\}^{\Lambda(n) \cdot n} \times \{0,1\}^{r(n)} \to \{0,1\}^n$ *can be implemented in* AC^0 *with* $O(a + c + 1)$ *depth and* $\mathrm{poly}(n)$ *size.*

To prove the lemma, we can express the Λ polynomials $C_1, \ldots, C_\Lambda$ by their Λ^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 suffice the following conditions:

1. The powering operation is to compute y^j, where $j \le \log^{2a}(n)$, and $y \in \mathbb{F}_q$. The order $q = 2^s$ is a power of 2, and $s = O\left(\log^c(n)\right)$.
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 [HV06] show that the powering and multiplication are indeed in AC^0.

Lemma 3.6 ([HV06, 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 $j \le \log^a(n)$ and $s \le \log^c(n)$, then y^j can be computed by an AC^0 circuit with $O(a+c)$ depth and $\mathrm{poly}(n)$ size.*

Lemma 3.7 ([HV06, 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 $s \le \log^c(n)$, then $y_1 \cdot y_2$ can be computed by an AC^0 circuit with $O(c)$ depth and $\mathrm{poly}(n)$ 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 $\mathrm{poly}\log n$, it is in AC^0 by Lemma 2.10.

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

Proof of Lemma 3.5. It is sufficient prove that each $\sum_{i=1}^{\Lambda} \sum_{j=1}^{\Lambda} c_{i,j} y^{j-1} X_{i,k}$ can be computed in AC^0 with $O(a+c)$ depth and $\mathrm{poly}(n)$ size. The powering could be computed in $O(a+c)$ depth and $\mathrm{poly}(n)$ size by Lemma 3.6. The multiplication could be computed in $O(c)$ depth and $\mathrm{poly}(n)$ size by Lemma 3.7. The summation could be computed in $O(a)$ depth and $\mathrm{poly}(n)$ size by Lemma 2.10. □

Theorem 3.3 follows directly from Lemma 3.4 and Lemma 3.5.

Proof of Theorem 3.3. Take $r(n) = \min\{s \in \mathbb{N} | s \ge \frac{2}{\delta} \log^c(n), \exists d \in \mathbb{N}, s = 3 \cdot 2^d\}, q(n) = 2^{r(n)}, K(n) = \frac{n}{r(n)}$ as discussed above. By Lemma 3.4, we know that the merger is a $(n, k(n), \varepsilon(n))$-merger, where $k(n) = (1-\delta)n$. By Lemma 3.5, we know that the merger can be implemented in AC^0 with $O(a+c)$ depth and $\mathrm{poly}(n)$ size. The theorem follows. □

3.1 Merger for high entropy sources

The original merger is only applicable to somewhere-uniform sources. We prepare a merger for somewhere-(n, k) source with high min-entropy by applying an extractor first, then merging them.

Theorem 3.8. *Let* $\Lambda(n) \le \mathrm{poly}(n), \varepsilon(n) = 2^{-O(n)}$, $\Delta(n) = O(\log(\frac{n}{\varepsilon}))$. *Then there exists a strong* $(n-\Delta(n), \frac{1}{2}m(n), \varepsilon(n))$*-merger* $\mathrm{Merge} : \{0,1\}^{\Lambda(n)\cdot n} \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$. *Here* $r(n) = O(\log(\frac{n}{\varepsilon}))$ *and* $m(n) = \Omega(n)$. *The merger is computable in* $\mathrm{AC}^0[2]$.

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

Proof. Assume that X is a simple somewhere-$(n, n-\Delta(n))$ source with Λ segments. Let $X_{i'}$ be a good segment. The construction of the merger is as follows:

1. Separate each X_i into $l = \frac{n}{\Delta(n)+5\log(\frac{1}{\varepsilon})}$ blocks of length $u(n) = \Delta(n) + 5\log(\frac{1}{\varepsilon})$, which are $X_{i,1}, \dots, X_{i,l}$. Take $s = 2\log(\frac{n}{\varepsilon})$, by Lemma 2.13, the good segment $X_{i'}$ satisfies that $(X_{i',1}, \dots, X_{i',n^{0.8}})$ is $l \cdot 2^s$-close to a $(n_1, k_1, n_2, k_2, \dots, n_l, k_l)$-block source. Here $n_j = u(n)$ and $k_j = 3\log(\frac{n}{\varepsilon})$ for each $j \in [l]$.

2. Since $3\log(\frac{n}{\varepsilon}) - 2\log(\frac{\varepsilon}{2l}) \ge \log(\frac{n}{\varepsilon})$, we take $\textsc{Ext}_1 : \{0,1\}^{u(n)} \times \{0,1\}^{r_1} \to \{0,1\}^{\log(\frac{n}{\varepsilon})}$ be a strong $(3\log(\frac{n}{\varepsilon}), \frac{\varepsilon}{2l})$-extractor using the leftover hash lemma from Lemma 2.4. Take U_1 be a uniformly random variable on $\{0,1\}^{r_1}$, and $Y_{i,j} = \textsc{Ext}_1(X_{i,j}, U_1)$ for each $i \in [\Lambda(n)], j \in [n^l]$. Each source $Y_i = (Y_{i,1}, Y_{i,2}, \dots, Y_{i,l})$ is of length $m(n) = l \cdot \log(\frac{n}{\varepsilon}) = \Omega(n)$. By Lemma 2.14, $Y_{i'} = (Y_{i',1}, \dots, Y_{i',l})$ is $\frac{\varepsilon}{2}$-close to uniform.

3. Take the merger $\mathrm{Merge}_1 : \{0,1\}^{\Lambda(n)\cdot m(n)} \times \{0,1\}^{r_2} \to \{0,1\}^{m(n)}$ be the $(m(n), \frac{1}{2}m(n), \frac{\varepsilon}{2})$-merger from Theorem 3.3. Take U_2 be a uniformly random variable on $\{0,1\}^{r_2}$, and $Z = \mathrm{Merge}_1(Y, U_2)$ which is the output.

The above merger is defined as $\mathrm{Merge}(X, U_1, U_2) = \mathrm{Merge}_1(Y, U_2)$ where $Y = (Y_{1,1}, \dots, Y_{\Lambda,l})$ and $Y_{i,j} = \textsc{Ext}_1(X_{i,j}, U_1)$ for each $i \in [\Lambda(n)], j \in [l]$.

For the $\mathrm{AC}^0[2]$ case, the extractor $\textsc{Ext}_1(x, y)$ can be realized as computing Ax on input x where $A = A(y)$ is a Toeplitz matrix of size $u(n) \cdot \log(\frac{n}{\varepsilon}) = \mathrm{poly}(n)$. So it is computable in $\mathrm{AC}^0[2]$.

The merger Merge_1 requires $\mathrm{poly}(n)$'th exponentiation of a $O(n)$-bit number in $\mathbb{F}$ of characristic 2, which is in $\mathrm{AC}^0[2]$ by [HV06, Theorem 4]. The multiplication and addition are both in $\mathrm{AC}^0[2]$. Therefore Merge_1 is computable in $\mathrm{AC}^0[2]$.

For the AC^0 case, notice that we set $\varepsilon(n) \ge 2^{-\log^c(n)}$. So the matrix size in $\textsc{Ext}_1$ is reduced to $O(\log^{2c}(n))$. Therefore it is computable in AC^0 by Lemma 2.10. For the merger, if $\Lambda(n) \le \log^a(n)$, then Theorem 3.3 shows that the merger is computable in AC^0.

The total seed length is $r_1 + r_2 = O(\log(\frac{n}{\varepsilon}))$.

The same arguments hold for the case that X is a somewhere-$(n, n-\Delta(n))$ source because a somewhere-$(n, n-\Delta(n))$ source is a convex combination of simple somewhere-$(n, n-\Delta(n))$ sources. The theorem follows. □

4 Error Reduction

In this section, we give a new error-reduction technique to transform an extractor with moderate error into an extractor with very small error. The main theorem of this section is the following:

Theorem 4.1. *For any constant* $a, c > 0, b \in \mathbb{N}^+$, *every* $k(n) \ge n/\log^a(n), \varepsilon(n) \ge 2^{-\log^c(n)}$, *there exists a strong* $(k(n), \varepsilon(n))$*-extractor* $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$, *where* $r(n) = O(\log(\frac{n}{\varepsilon(n)}))$, $m(n) = \Theta\left(\log^b(n) \cdot \log(\frac{n}{\varepsilon(n)})\right)$.

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

For the following discussion in the section, we will fix $a > 0$ to be a constant and $k(n) = \frac{n}{\log^a n}$. We mainly prove the following error reduction lemma.

Lemma 4.2. *For every $\varepsilon_0 \in (0,1)$, every constant $c > 0$, suppose there exists a (k, ε_0) extractor $\text{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ with $m_0 \geq k^{\Omega(1)}$. Then for any $\varepsilon = 2^{-\log^c n}$, there exists a (k, ε) extractor $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $d = O(d_0 \cdot \frac{\log \varepsilon}{\log \varepsilon_0})$, $m = \Theta\left(\log^b(n) \cdot \log(\frac{n}{\varepsilon})\right)$.*

If Ext_0 can be realized by a depth h AC circuit, then Ext can be realized by a depth $O(b(h+c+1))$ AC circuit.

Notice that Theorem 4.1 directly follows from Lemma 4.2 by instantiating $\text{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ as a (k, ε_0) extractor from Theorem 2.1, with $\varepsilon_0 = 1/n, m_0 = 0.9k, d_0 = O(\log n)$, depth $h = O(a)$.

For the rest of this section, we prove Lemma 4.2.

4.1 Step 1: extracting in parallel

In this section, we apply 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 $Y = (Y_1, Y_2, \ldots, Y_t)$, where $Y_i = \text{Ext}_0(X, U_{1,i})$. The step can be computed by depth h AC circuits because the extractor Ext_0 has depth h, and the parallel extraction can be done without increasing the depth.

Next, we show that the result is close to a somewhere-$(m_0(n), m_0(n) - O(\log t))$-source. The main idea is that by Lemma 2.9, 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 4.3. *Let $\text{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ be an (k, ε_0)-extractor and X be a $(n, k+s)$-source. Take independent seeds $U_1, U_2, \ldots, U_s \in \{0,1\}^{d_0}$. Let $Y = (Y_1, Y_2, \ldots, Y_t)$, where $Y_i = \text{Ext}_0(X, U_i)$. 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 2.9. We first denote event $\text{BAD}_i = \{U_i \notin G_X\}$. 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 4.4. $\Pr[\text{BAD}_1 \wedge \text{BAD}_2 \wedge \cdots \wedge \text{BAD}_t] \leq (2\varepsilon_0)^t$.

Proof. Fix any x from X. By Lemma 2.9, we know that $\Pr\{U_i \notin G_x\} \leq 2\varepsilon_0$ for every $i \in [t]$. By the independence of U_i's, we have $\Pr\{U_i \notin G_x, \forall i \in [t]\} \leq (2\varepsilon_0)^t$. Since this holds for every $x \in X$, we have $\Pr\{\text{BAD}_1 \wedge \text{BAD}_2 \wedge \cdots \wedge \text{BAD}_t\} \leq (2\varepsilon_0)^t$. □

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

$$\forall i \in [t], i \in I \iff U_i \in G_X. \tag{4}$$

With probability at least $1 - (2\varepsilon_0)^t$, The set I is not an empty set. Take $Y_i = \text{Ext}(X, U_i)$. By Lemma 2.9, $Y_i|_{(\text{BAD}_i)^c} = Y_i|_{i \in I}$ is 2^{-s}-close to a $(m_0, m_0 - O(1))$ source.

We apply the technique from [LRVW03] to prove that $(Y_1, Y_2, \ldots, Y_t)$ is indeed close to a somewhere-$(m_0, m_0 - O(\log t))$-source.

Lemma 4.5 ([LRVW03])**.** *Let $Y = (Y_1, \ldots, Y_t)$ be the random variable defined in Lemma 4.3. Let I be a random set subset of $[t]$. Assume $I \neq \varnothing$, and for every $i \in [t]$, $Y_i|_{i \in I}$ is ε-close to a (m, k)-source. Then Y is $(t \cdot \varepsilon)$-close to a somewhere-$(m, k - \log t)$ source.*

For completeness of the proof, we reprove this lemma.

Proof. Take I_0 to be the random selector variable over $[t]$, such that for every $S \subseteq [t]$, $I_0|_{I=S}$ uniformly randomly chooses one index from S. Fix $i \in [t]$, for every atomic state $(y_1, \dots, y_t, S)$ such that $i \in S$, define the atomic event $E = E(y_1, \dots, y_t, S) = \{Y_1 = y_1, \dots, Y_t = y_t, I = S\}$. Then for each event E,

$$\frac{\Pr(E \wedge I_0 = i)}{\Pr(E \wedge i \in I)} = \frac{\Pr(I_0 \text{ choose } i \text{ from } I|_E)\Pr[E]}{\Pr[E]} \in [1/t, 1]. \tag{5}$$

By summing over all such events, we have

$$\frac{\Pr(I_0 = i)}{\Pr(i \in I)} = \frac{\sum_{\{(y_1,\dots,y_t,S)|i\in S\}} \Pr(E(y_1, \dots, y_t,) \wedge I_0 = i)}{\sum_{\{(y_1,\dots,y_t,S)|i\in S\}} \Pr(E(y_1, \dots, y_t, S) \wedge i \in I)} \in [1/t, 1]. \tag{6}$$

By conditioning on the events respectively,

$$\frac{\Pr(E|_{I_0=i})}{\Pr(E|_{i\in I})} = \frac{\Pr(E \wedge I_0 = i)/\Pr(I_0 = i)}{\Pr(E \wedge i \in I)/\Pr(i \in I)} \in [1/t, t]. \tag{7}$$

Therefore, we have

$$\frac{\Pr\{Y_i = y|_{I_0=i}\}}{\Pr\{Y_i = y|_{i\in I}\}} = \frac{\sum_{\{(y_1,\dots,y_t,S)|i\in S, y_i=y\}} \Pr\{E(y_1, \dots, y_t, S)|_{I_0=i}\}}{\sum_{\{(y_1,\dots,y_t,S)|i\in S, y_i=y\}} \Pr\{E(y_1, \dots, y_t, S)|_{i\in I}\}} \in [1/t, t]. \tag{8}$$

By assumption, $Y_i|_{i\in I}$ is ε-close to a (m, k)-source. Equivalently,

$$\sum_{\{y|\Pr\{Y_i|_{i\in I}=y\}\geq 2^{-k}\}} \Pr\{Y_i|_{i\in I} = y\} - 2^{-k} \leq \varepsilon. \tag{9}$$

By applying the multiplicative relation between $\Pr\{Y_i|_{I_0=i} = y\}$ and $\Pr\{Y_i|_{i\in I} = y\}$, we have

$$\sum_{\{y|\Pr\{Y_i|_{I_0=i}=y\}\geq t\cdot 2^{-k}\}} \Pr\{Y_i|_{I_0=i} = y\} - t \cdot 2^{-k} \leq t \cdot \varepsilon. \tag{10}$$

The lemma follows. □

By Claim 4.4 and Lemma 4.5, we can prove Lemma 4.3:

Proof of Lemma 4.3. Take I as the random set indicator defined above. By Lemma 2.9, $Y_i|_{(BAD_i)^c} = Y_i|_{i\in I}$ is 2^{-s}-close to a $(m_0, m_0 - O(1))$ source. By Claim 4.4, we know that with probability at least $1 - (2\varepsilon_0)^t$, I is not an empty set. Conditioning on such events, Lemma 4.5 implies that $Y|_{\{I\neq\varnothing\}}$ 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

In this subsection, we first divide each segment of the somewhere-$(m_0, m_0 - O(\log t))$-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

$$Y_i = (Y_{i,1}, Y_{i,2}, \dots, Y_{i,l+1}) \text{ for every } i \in [t]. \tag{11}$$

The lengths satisfies

$$m_j = m_0^{0.1} \cdot 3^{j-1} \text{ for every } j \in [l]. \tag{12}$$

where $l = \lfloor \log_3 m_0 \rfloor$. 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:

$$B_j = (Y_{1,1\dots j}, Y_{2,1\dots j}, \dots, Y_{t,1\dots j}) \text{ for every } j \in [l]. \tag{13}$$

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

Let $\text{Merge}_j : \{0,1\}^{M_j} \times \{0,1\}^{d_2(n)} \to \{0,1\}^{(1-\alpha)M_j}$ be a strong $(M_j - M_j^{0.1}, \frac{3}{4}(1-\alpha)M_j, \varepsilon(n)/l)$-merger from Theorem 3.8 for every $j \in [l]$, where α is a constant. Let U_2 be a uniform random variable on $\{0,1\}^{d_2(n)}$. Define

$$Z_j = \text{Merge}_j(B_j, U_2) \text{ for every } j \in [l]. \tag{14}$$

The seed length of the merger is $d_2(n) = O(\log(\frac{M_j}{\varepsilon(n)})) = O(\log(\frac{m(n)}{\varepsilon(n)}))$.

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. We will extract from the block source to acquire the desired amount of entropy.

Lemma 4.6. *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 Theorem 3.8, Z_j is $\varepsilon(n)/l$-close to a $((1-\alpha)M_j, \frac{3}{4}(1-\alpha)M_j)$-source. The claim follows. □

The entropy argument immediately shows that Z_j's form a block source.

Lemma 4.7. *$(Z_1, Z_2, \dots, Z_l)$ is $2\varepsilon(n)$-close to a block source $(Z'_1, Z'_2, \dots, Z'_l)$. The conditional entropy of Z'_j is larger than $(1-\alpha)M_j/100 = \Omega((1-\alpha)M_j)$ for each $j \in [l]$*

Proof. We prove by induction that $(Z_1, Z_2, \dots, Z_j)$ is $\frac{2j}{l}\varepsilon(n)$-close to a block source $(Z'_1, Z'_2, \dots, Z'_j)$. The base case $j = 1$ is straightforward.

For the induction case, assume that the proposition holds for $j-1$. Consider the distribution $(Z'_1, Z'_2, \dots, Z'_{j-1}, Z^*_j)$, where $Z^*_j = T_I$. Here $I \in \{0,1\}$ is a selector random variable and T_0, T_1 are two different random variables. For simplicity, we denote $Z_{pref} = (Z_1, Z_2, \dots, Z_{j-1})$ and $Z'_{pref} = (Z'_1, Z'_2, \dots, Z_{j'-1})$. The conditional distribution $I|_{Z'_{pref}=z}$ is defined as $\Pr[I|_{Z'_{pref}=z} = 0] = \frac{\min(\Pr[Z_{pref}=z], \Pr[Z'_{pref}=z])}{\Pr[Z'_{pref}=z]}$. The distribution T_0 satisfies that $T_0|_{Z'_{pref}=z}$ has the same distribution as $Z_j|_{Z_{pref}=z}$.

Since $(Z_1, Z_2, \dots, Z_{j-1})$ is $\frac{2(j-1)}{l}\varepsilon(n)$-close to $(Z'_1, Z'_2, \dots, Z'_{j-1})$ by induction hypothesis, we have $\Pr[I = 0] \geq 1 - \frac{2(j-1)}{l}\varepsilon(n)$. Since T_0 has the same conditional distribution as Z_j, $(Z'_1, Z'_2, \dots, Z'_{j-1}, T_I)$ is $\frac{2(j-1)}{l}\varepsilon(n)$-close to $(Z_1, Z_2, \dots, Z_{j-1}, Z_j)$ regardless of how we choose T_1. Furthermore, $\Pr[Z_j = z] \geq \Pr[T_0 = z \wedge I = 0]$ for every point z in the co-domain.

We define T_1 such that $Z^*_j = T_I$ has the same distribution as Z_j. Specifically, T_1 is a distribution independent of Z_{pref}, Z'_{pref} such that $\Pr[T_1 = z] = \frac{\Pr[Z_j=z] - \Pr[T_0=z \wedge I=0]}{\sum_{w\in\{0,1\}^m}(\Pr[Z_j=w] - \Pr[T_0=w \wedge I=0])} = \frac{\Pr[Z_j=z] - \Pr[T_0=z \wedge I=0]}{\Pr[I=1]}$. Then $\Pr[T_I = z] = \Pr[T_0 = z \wedge I = 0] + \Pr[T_1 = z \wedge I = 1] = \Pr[Z_j = z]$.

The distribution $(Z'_1, Z'_2, \ldots, Z'_{j-1}, Z^*_j)$ is $\frac{2(j-1)}{l}\varepsilon(n)$-close to $(Z_1, Z_2, \ldots, Z_{j-1}, Z_j)$ and $Z^*_j = T_I$ has the same distribution as Z_j. By Lemma 4.6, there exists a $((1-\alpha)M_j, \frac{3}{4}(1-\alpha)M_j)$-source Z''_j such that Z^*_j is $\varepsilon(n)/l$-close to Z''_j.

Since $\frac{3}{4}(1-\alpha)M_j$ is larger that $\sum_{i=1}^{j-1}|Z_{j-1}| = \sum_{i=1}^{j-1}(1-a)M_i$, Lemma 2.13 implies that $(Z'_1, Z'_2, \ldots, Z'_{j-1}, Z''_j)$ is 2^{-s}-close to $(Z'_1, Z'_2, \ldots, Z'_{j-1}, Z'_j)$ such that $Z'_j|_{Z'_1=z'_1, Z'_2=z'_2, \ldots, Z'_{j-1}=z'_{j-1}}$ is a $((1-\alpha)M_j, \frac{3}{4}(1-\alpha)M_j - s - \sum_{i=1}^{j-1}(1-a)M_i)$ source. Take $s = (1-\alpha)M_j/100$. The min-entropy term is $(1-\alpha)M_j \cdot (\frac{3}{4} - \frac{1}{100} - \sum_{i=1}^{j-1} 3^{-i}) \geq (1-\alpha)M_j/100$. The statistical distance is $2^{-s} = 2^{-(1-\alpha)M_j/100} \leq \varepsilon(n)/l$.

By triangular inequality, $(Z_1, Z_2, \ldots, Z_{j-1}, Z_j)$ is $\frac{2j}{l}\varepsilon(n)$-close to a block source $(Z'_1, Z'_2, \ldots, Z'_j)$. ☐

Next, we apply the strong extractor from Theorem 2.3 to extract from the block source. Let $\text{Ext}_j : \{0,1\}^{(1-\alpha)M_j} \times \{0,1\}^{d_3(n)} \to \{0,1\}^{m'(n)}$ be strong $((1-\alpha)M_j/100, \varepsilon(n)/l)$-extractor for every $j \in [l]$. Let U_3 be a uniform random variable on $\{0,1\}^{d_3(n)}$. Then

$$W_j = \text{Ext}_j(Z_j, U_3) \text{ for every } j \in [l]. \tag{15}$$

The seed length of the extractor is $d_3(n) = O(\log(\frac{(1-\alpha)M_j}{\varepsilon(n)})) = O(\log(\frac{n}{\varepsilon(n)}))$.

Lemma 4.8. *$(W_1, W_2, \ldots, W_l)$ is $3\varepsilon(n)$-close to uniform.*

Proof. By the previous lemma, $(\text{Ext}_1(Z_1, U_3), \text{Ext}_2(Z_2, U_3), \ldots, \text{Ext}_j(Z_j, U_3))$ is $2\varepsilon(n)$-close to $(\text{Ext}_1(Z'_1, U_3), \text{Ext}_2(Z'_2, U_3), \ldots, \text{Ext}_{j-1}(Z'_{j-1}, U_3), V_j)$. The source $(Z'_1, Z'_2, \ldots, Z'_{j-1}, Z'_l)$ is a block source. By Lemma 2.14, the lemma holds. ☐

For every simple somewhere-$(m, m - O(\log t))$-source Y, the distribution W is $3\varepsilon(n)$-close to uniform. the same holds for general somewhere-$(m, m - O(\log t))$-source Y because it is a convex combination of simple somewhere-$(m, m - O(\log t))$-sources.

By composing the first and second steps above, we have a strong extractor which is computable in AC^0:

Lemma 4.9. *For any $\varepsilon_0 \in (0,1)$ every constant $c > 0$, suppose there exists a (k, ε_0) extractor $\text{Ext}_0 : \{0,1\}^n \times \{0,1\}^{d_0} \to \{0,1\}^{m_0}$ with $m_0 \geq k^{0.01}$. Then for any $\varepsilon = 2^{-\log^c n}$, there exists a (k, ε) extractor $\text{Ext}' : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $d = O(d_0 \cdot \frac{\log \varepsilon}{\log \varepsilon_0})$, $m = \Theta\left(\log(n) \cdot \log(\frac{n}{\varepsilon})\right)$.*

If Ext_0 can be realized by a depth h AC circuit, then Ext' can be realized by a depth $O(h+c+1)$ AC circuit.

Proof. Take X be the sources, U_1, U_2, U_3 be the seeds. Let $Y = (Y_1, Y_2, \ldots, Y_t)$ such that $Y_i = \text{Ext}(X, U_{1,i})$ for every $i \in [t]$ as in the first step. Then Y is $\varepsilon(n)$-close to a simple somewhere-$(m(n), m(n) - O(\log t))$-source conditioning on U_1. Let B_j be the source $(Y_{1,1\ldots j}, Y_{2,1\ldots j}, \ldots, Y_{t,1\ldots j})$ for every $j \in [l]$. Then take $Z_j = \text{Merge}_j(B_j, U_2)$ and $W_j = \text{Ext}_j(Z_j, U_3)$ for every $j \in [l]$ as in the second step. By Lemma 4.8, 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. The extractor can be made strong in a standard way since the output length is much longer than the seed length.

Step 1 executes the extractor Ext_0 in parallel, which costs depth h. Step 2 executes the merger Merge_j and the extractor Ext_j from Theorem 3.8 and Theorem 2.3 for every $j \in [l]$ in parallel which take depth $O(a+c)$. So the overall depth is as the lemma stated.

The seed length of the extractor is $r(n) = |U_1| + |U_2| + |U_3|$. $U_1 = (U_{1,1}, U_{1,2}, \ldots, U_{1,t})$ where $|U_{1,i}| = O(\log n)$ for every $i \in [t]$ and $t = \frac{\log(1/\varepsilon(n))}{\log n}$. $|U_2| = O(\log(\frac{n}{\varepsilon(n)}))$ and $|U_3| = O(\log(\frac{n}{\varepsilon(n)}))$. Therefore $r(n) = O(\log(\frac{n}{\varepsilon(n)}))$. ☐

4.3 Step 3: Sample-then-Extract

The final step of the construction is to stretch the length of the output from $\Theta\left(\log(n)\cdot\log(\frac{n}{\varepsilon(n)})\right)$ to $\Theta\left(\log^b(n)\cdot\log(\frac{n}{\varepsilon(n)})\right)$ for any given constant $b>0$. We use the Sample-then-Extract method to achieve this.

Assume X_0 is a $(n, 3\delta n)$-source. Let $\mathrm{Samp}_1,\dots,\mathrm{Samp}_b$ be sampler functions such that $\mathrm{Samp}_i : \{0,1\}^r_i \to [\delta n]^{n_{i+1}}$ samplers. Take $X_i = (X_{i-1})_{\mathrm{Samp}_i(U_i)}$ for every $i\in[b]$. Then the Sample-then-Extract method is the following:

Lemma 4.10 (Repeated Sample-then-Extract from [Vad03])**.** *For any constant $\delta\in(0,1), b\in\mathbb{N}$ Assume X_0 is a $(n,(2\delta+3b\varepsilon)n)$-source. Let $\mathrm{Samp}_i : \{0,1\}^r_i \to [\delta^{i-1}n]^{\delta^i n}$ be $(\gamma,\varepsilon/\log(1/\varepsilon))$ samplers. Let $U=(U_1,U_2,\dots,U_b)$ be a uniform random seed. Take $X_i=(X_{i-1})_{\mathrm{Samp}_i(U_i)}$ for every $i\in[b]$. Then $(X_0,X_1,\dots,X_b)$ is $b\cdot(\gamma+2^{-\Omega(\varepsilon n)})$-close to a source $(X'_0,X'_1,\dots,X'_b)$ such that $X'_i|_{U=u}$ is a $(\delta^{i-1}n, 2\delta^i n)$-source for every $i\in[b]$.*

Proof. We use induction to prove the lemma. For $b=2$, by the typical sample then extractor technique in [Vad03], (U,X_0,X_1) is $(\gamma+2^{-\Omega(\varepsilon n)})$-close to a source (U,X'_0,X'_1) such that $X'_1|_{U=u}$ is a $(\delta n,(2\delta+3(b-1)\varepsilon)\delta n)$-source.

For $b>2$, construct X'_1 as above. define $X'_i=(X'_{i-1})_{\mathrm{Samp}_i(U_i)}$ for every $i\in[b], i\geq 2$. By induction hypothesis, $(X'_1,X'_2,\dots,X'_b)$ is $(b-1)\cdot(\gamma+2^{-\Omega(\varepsilon n)})$-close to a source $(X''_1,X''_2,\dots,X''_b)$ such that $X''_i|_{U=u}$ is a $(\delta^{i-1}n,\delta^i n)$-source for every $i\in[b]$. Since $(X_1,X_2,\dots,X_b)$ is $(\gamma+2^{-\varepsilon n})$-close to $(X'_1,X'_2,\dots,X'_b)$, the lemma follows from the triangle inequality. □

To apply the extraction to the block sources $X_b, X_{b-1},\dots,X_1$, we need a seed Y_b and apply an extractor $\textsc{Ext}_b : \{0,1\}^{\delta^{b-1}n}\times\{0,1\}^{r_b}\to\{0,1\}^{r_{b-1}}$ to get $Y_{b-1}=\textsc{Ext}_b(Y_b,Y_b)$. We repeat this process for $i\in[b-1]$ to get Y_1, which is the output.

Now we prove the main lemma.

Proof of Lemma 4.2. Let X be a $(n,k(n))$ source. Define $\delta=\delta(n)=\frac{k(n)-3bn^{0.5}}{2n}$. Take $Samp_i$ be $(\gamma,1/(n^{0.5}\log n))$-samplers from Theorem 2.7, where $\gamma=\varepsilon/(4b)$. Take $X_i=(X_{i-1})_{\mathrm{Samp}_i(U_i)}$ for every $i\in[b]$. Then $(X_0,X_1,\dots,X_b)$ is $b\cdot(\gamma+2^{-\Omega(n^{0.5})})$-close to a source $(X'_0,X'_1,\dots,X'_b)$ such that $X'_i|_{U=u}$ is a $(\delta^{i-1}n,2\delta^i n)$-source for every $i\in[b]$. By Lemma 2.13, $X'_i|_{U=u,X'_{i+1}=x'_i}$ is a $(\delta^{i-1}n,\delta^i n)$-source for every $i\in[b]$.

For each $i\in[b-1]$, let $\textsc{Ext}_i : \{0,1\}^{\delta^{i-1}n}\times\{0,1\}^r\to\{0,1\}^m$ be $(\delta^i n/2,\varepsilon/(2bn))$ extractors from Lemma 4.9, such that $r=O(\log(\frac{n}{\varepsilon}))$ and $m=C\log n\cdot r=O(\log n\log(\frac{n}{\varepsilon}))$ for some constant $C>0$. Apply $\textsc{Ext}_i$ for $t_i=(C\log n)^{b-i-1}$ times, which gives

$$\textsc{Ext}'_i(X,U)=(\textsc{Ext}_i(X,U_1),\textsc{Ext}_i(X,U_2),\dots,\textsc{Ext}_i(X,U_{t_i})) \tag{16}$$

for $U=(U_1,U_2,\dots,U_{t_i})$. Since $t_i\cdot\varepsilon/(2bn)<\varepsilon/(2b)$, $\textsc{Ext}'_i : \{0,1\}^{\delta^{i-1}n}\times\{0,1\}^{r_i}\to\{0,1\}^{r_{i-1}}$ is a $(\delta^i n,\varepsilon/(2b))$-extractor. Here $r_i=(C\log n)^{b-i-1}r$.

Take Y_b be an independent uniform seed on $\{0,1\}^r$. Take $Y_{i-1}=\textsc{Ext}'_i(X_i,Y_i)$ for every $i\in[b-1]$. The final output is Y_0.

By the extractor property of $\textsc{Ext}'_i$, we know that $(X'_1,X'_2,\dots,X'_{i-1},\textsc{Ext}'_i(X'_i,U_{i+1}))$ is $\varepsilon/(2b)$-close to $(X'_1,X'_2,\dots,X'_{i-1},U_i)$ where U_i are uniform distributions on $\{0,1\}^{r_i}$. Denote $Y'_{i-1}=\textsc{Ext}'_i(X'_i,Y'_i)$ By the triangle inequality, $(X'_1,X'_2,\dots,X'_{j-1},Y'_j)$ is $\frac{b-j}{2b}\varepsilon$-close to a uniform distribution. Therefore Y'_0 is $\frac{1}{2}\varepsilon$-close to uniform.

From the result that $(X_0, X_1, \dots, X_b)$ is $b \cdot (\gamma + 2^{-\varepsilon})$-close to a source $(X'_0, X'_1, \dots, X'_b)$ where $b \cdot (\gamma + 2^{-\Omega(n^{0.5})}) \le \frac{1}{2}\varepsilon$, we have that Y_0 is $\frac{1}{2}\varepsilon$-close to Y'_0. Triangle inequality gives that Y_0 is ε-close to uniform.

The overall depth is $O((h + c + 1)b)$ since we only apply Ext in parallel for $O(b)$ rounds.

□

5 Output Stretch

In this section, we will use the framework introduced in [DKSS13], 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 5.1. *For any constant $a, c > 0$ and $\gamma \in (0, 1)$, let $k(n) \ge \frac{n}{\log^a(n)}, \varepsilon(n) \ge 2^{-\log^c(n)}$. Then there exists a $(k(n), \varepsilon(n))$-strong extractor* $\text{Ext} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$*, such that $r(n) = O(\log(\frac{n}{\varepsilon}))$, and $m(n) \ge (1 - \gamma) \cdot k(n)$.*

Furthermore, the extractor can be implemented in AC^0*, with $O(a + c + 1)^2$ depth and* $\mathrm{poly}(n)$ *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 5.2 (somewhere-block-source)**.** *Let $X = (X_1, \dots, X_\Lambda)$ be a random variable with Λ 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 [Ta-98] 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 Λ segments.

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

$$X_1 = (X_{1,1}, \dots, X_{1,\Lambda}) \in \left(\{0,1\}^{n/\Lambda}\right)^{\Lambda}. \tag{17}$$

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

$$Y_i = (X_{1,1}, \dots, X_{1,i}, 0^{(\Lambda - i)\cdot(n/\Lambda)}), \tag{18}$$
$$Z_i = (0^{i \cdot (n/\Lambda)}, X_{1,i+1}, \dots, X_{1,\Lambda}). \tag{19}$$

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 $i \in [\Lambda]$:

$$\mathrm{B}^{\Lambda}_{TS}(X_1) = \left\{ (Y_i, Z_i) \in \{0,1\}^{2n} \mid i \in [\Lambda] \right\}. \tag{20}$$

Theorem 5.3 ([Ta-98])**.** *Let Λ be an integer and Λ divides n. Let $\mathrm{B}^{\Lambda}_{TS}$ 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 $X \in \{0,1\}^n$, $\mathrm{B}^{\Lambda}_{TS}(X)$ 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 $X_2 = (X_{2,1}, \ldots, X_{2,\Lambda}) = \mathrm{B}_{TS}^{\Lambda}(X_1)$ be the somewhere-block-source.

Lemma 5.4. *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 $X_2 = \mathrm{B}_{TS}^{\Lambda}(X_1)$ 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 $\mathrm{poly}(n)$ 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 2.2 and Theorem 4.1 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.

Definition 5.5 (two-block extractor)**.** *We say a function* $\textsc{Ext} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^r \to \{0,1\}^m$ *is a (k_1, k_2, ε)-strong-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 $(U_r, \textsc{Ext}(X_1, X_2, U_r))$ is ε-close to uniform distribution on $\{0,1\}^r \times \{0,1\}^m$.*

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 5.6. *Let $X = (X_1, \ldots, X_\Lambda)$ be a (k_1, k_2)-somewhere-block-source, where each segments is a source on $\{0,1\}^{n_1} \times \{0,1\}^{n_2}$. Let* $\textsc{Ext} : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^r \to \{0,1\}^m$ *be a (k_1, k_2, ε)-strong-two-block extractor. Let U_r be a uniform random distribution on $\{0,1\}^r$. Then*

$$(\textsc{Ext}(X_1, U_r), \ldots, \textsc{Ext}(X_\Lambda, U_r))$$

is ε-close to a somewhere-uniform-source.

Proof. If X is a simple-somewhere-block-source, then there exists a good segment X_i such that X_i is a (k_1, k_2)-block-source. Then $(\textsc{Ext}(X_i, U_r))$ is ε-close to a uniform distribution on $\{0,1\}^m$. Therefore, $(\textsc{Ext}(X_1, U_r), \ldots, \textsc{Ext}(X_\Lambda, U_r))$ is ε-close to a somewhere-uniform-source.

Otherwise, X is a convex combination of simple-somewhere-block-sources. Each simple-somewhere-block-source is converted into a simple-somewhere-uniform-source. Therefore, X is converted into a somewhere-uniform-source. The lemma follows. □

For AC^0 implementation, we have the following theorem:

Theorem 5.7 (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* $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$, *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 $\mathrm{poly}(n)$ size.

Proof. Take $\text{Ext}_1 : \{0,1\}^n \times \{0,1\}^{m_1} \to \{0,1\}^{m_2}$ be the $(k_1, \varepsilon(n)/2)$-extractor from Theorem 2.2, where $m_1 = \log^{O(a+c)}(n)$ and $m_2 = (1-\gamma)k_1(n)$. Take $\text{Ext}_2 : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^{m_1}$ be the $(k_2, \varepsilon(n)/2)$-extractor from Theorem 4.1, where $r(n) = O(\log(\frac{n}{\varepsilon}))$ and $m_1 = \log^{O(a+c)}(n)$. By Theorem 2.16, $\text{Ext}(X_1, X_2, U_r) = Ext_1(X_1, \text{Ext}_2(X_2, U_r))$ is a $(k_1, k_2, \varepsilon(n))$-strong-two-block extractor.

The extractor is in AC^0 with depth $O(a+c+1)^2$, as Ext_1 is in AC^0 with depth $O(a+c+1)$ and Ext_2 is in AC^0 with depth $O(a+c+1)^2$ □

We summarize the second step here:

Step 2: Take $\text{Ext} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{r_1(n)} \to \{0,1\}^{m(n)}$ be the $(\frac{n}{\log^a(n)}, \frac{n}{\log^{2a}(n)}, \varepsilon(n))$-strong-two-block extractor, where $r_1(n) = O(\log(\frac{n}{\varepsilon}))$ and $m(n) \geq (1-\gamma)k(n)$. Take $X_3 = (\text{Ext}(X_{2,1}, U_{r_1}), \ldots, \text{Ext}(X_{2,\Lambda}, U_{r_1}))$ 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 [DKSS13] to merge the segments into one source.

5.3 Step 3: Merging the segments

We use the merger introduced in [DKSS13] to merge the segments of the somewhere-uniform-source into one source. The construction of the merger is discussed in Theorem 3.3.

Step 3: Take $\text{Merge} : \{0,1\}^{\Lambda \cdot m(n)} \times \{0,1\}^{r_2(n)} \to \{0,1\}^{m(n)}$ be the $(m(n), \frac{3}{4}m(n), \varepsilon(n))$-merger from Theorem 3.3. Then $X_4 = \text{Merge}(X_3, U_{r_2})$.

As a direct consequence of Theorem 3.3 we have the following lemma.

Lemma 5.8. *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 $\text{Ext}_2 : \{0,1\}^{m(n)/2} \times \{0,1\}^{m(n)/2} \times \{0,1\}^{r_3(n)} \to \{0,1\}^{m'(n)}$ be the $(\frac{1}{8}m(n), \frac{1}{8}m(n), \varepsilon(n))$-strong-two-block extractor from Theorem 5.7, 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 5.9. *X_5 is $5\varepsilon(n)$ close to uniform.*

Proof. We divide X_4 into 2 parts, $X_4 = (X_4', X_4'')$ on $\{0,1\}^{m(n)/2} \times \{0,1\}^{m(n)/2}$. By Lemma 5.8, X_4 is $3 \cdot \varepsilon(n)$-close to a $\frac{3}{4}m(n)$-source on $\{0,1\}^{m(n)}$. By Lemma 2.13, (X_4', X_4'') is $3\varepsilon(n) + 2^{-\frac{1}{24}m(n)}$-close to a $(\frac{1}{8}m(n), \frac{1}{8}m(n))$-block source. Here $2^{-\frac{1}{24}m(n)} \leq \varepsilon(n)$.

Now we apply the block extractor from Theorem 5.7 to extract randomness from the block source (X_4', X_4'').

Since (X_4', X_4'') is $4\varepsilon(n)$-close to a $(\frac{1}{8}m(n), \frac{1}{8}m(n))$-block source by Lemma 5.8, the final distribution X_5 is $5\varepsilon(n)$-close to a uniform distribution. □

The circuit depth of $\textsc{Ext}_2$ is $O(a+c=1)^2$ by Theorem 5.7.

Now we prove the main theorem of this section:

Theorem 5.10. *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 $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$, 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 AC^0, with $O(a+c+1)^2$ depth and $\mathrm{poly}(n)$ size.

Proof. The extractor $\textsc{Ext}$ is defined as $\textsc{Ext}(X_1, U_{r_1}, U_{r_2}, U_{r_3}) = X_5$, where X_5 is defined through the four steps above.

The extractor can be implemented in AC^0 with $O(a+c)^2$ depth and $\mathrm{poly}(n)$ size as each step is in AC^0 with corresponding parameters. The overall seed length is $r_1(n)+r_2(n)+r_3(n) = O(\log(\frac{n}{\varepsilon}))$. The output length is $m'(n) = \frac{1-\gamma}{6} \cdot m(n) = \frac{(1-\gamma)^2}{6} k(n)$. The error is $5\varepsilon(n)$ by Lemma 5.9.

By repeatedly extracting from the source X_1 in parallel for $(1-\gamma')/\frac{(1-\gamma)^2}{6}$ times with independent seeds, we could extract the desired amount of randomness with error $5\varepsilon(n) \cdot \frac{(1-\gamma)^2}{6} \cdot \frac{1}{1-\gamma'}$. The theorem follows by adjusting the error parameter by increasing the seed length. □

6 Extractors in NC^1

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

1. Apply a condenser from [KT22]. It behaves like the GUV condenser but is computable in 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 $O(\log^2 n \log(n/\epsilon))$.
3. Apply the improved Trevisan's extractor [RRV02] to the first block, which outputs $\Omega(k)$ bits of randomness.

The main theorem is as follows:

Theorem 6.1. *For every constant $\gamma \in (0,1)$ every $k = k(n) \geq \Omega(\log^2(n)), \varepsilon = \varepsilon(n) \geq 2^{-O\left(\sqrt{k(n)}\right)}$, there exists a strong (k, ε) extractor $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$ computable in NC^1, with $r(n) = O(\log(n/\varepsilon))$, $m(n) = (1-\gamma)k(n)$.*

6.1 Condenser in NC^1

The first component in our construction is the condenser from [KT22]. A simplified version of their result is as follows:

Lemma 6.2 (condenser from [KT22]). *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 $\mathrm{Cond} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$ with $r \leq 4\log(\frac{n}{\varepsilon})$ such that Cond is a $(k, k+r, \varepsilon)$-condenser.*

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:

$$\mathrm{Cond}(x, y) = (y, f(y), f^{(1)}(y), \ldots, f^{(s)}(y)) \tag{21}$$

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 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 NC^1. Therefore, the formal derivative is computable in NC^1.

The polynomial evaluation consists of three operations: calculating the powering x^{i-j}, multiplication and summation. The powering could be implemented with two steps: powering in $\mathbb{N}$ and division by q, which is computable in NC^1 according to [BCH86]. The multiplication and iterated summation are both in NC^1.

Putting it together, we can obtain the following lemma:

Lemma 6.3. *The condenser from Lemma 6.2 is computable in NC^1.*

Regard the output of the condenser as (X_1, X_2), $|X_1| = |X_2| = \frac{1}{2}m(n)$. By Lemma 2.13, (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 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 $O(\log n \log(n/\epsilon))$, 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 ε-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 Theorem 3.8 to the output of the first step as in Section 4. Since $\varepsilon \geq 2^{-O(\sqrt{k})}$ and $t = \mathrm{poly}(k)$, the merger is computable in NC^1.

After applying the merger, we obtain a block-source with exponentially increasing length. We require a modification to Theorem 2.3 for the NC^1 setting. The main difference is that the error is now $2^{-O(\sqrt{k})}$ instead of $2^{-\mathrm{poly}(\log n)}$. 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 $\varepsilon = 2^{-O(\sqrt{k})}$.

We use the following standard method to extract from the block source.

Lemma 6.4. *For every constant $\delta \in (0, 1]$ and every $\varepsilon = 2^{-O(n)}$, there exists an explicit $(\delta n, \varepsilon)$-extractor $\mathrm{EXT} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in NC^1, where $d = O(\log(n/\varepsilon))$, $m = \Theta(\log(n/\varepsilon))$.*

To prove the lemma, we use the sampler from [Hea08].

Lemma 6.5 (Sampler in [Hea08]). *For any $n \in \mathbb{N}$, any $\epsilon \in (0,1]$, there exists an (γ, ϵ)-averaging sampler* $\mathrm{Samp} : \{0,1\}^r \to [n]^m$ *with seed length $r = \log n + O\left(\frac{\log(1/\gamma)}{\epsilon^2}\right)$ and $m = O\left(\frac{\log(1/\gamma)}{\epsilon^2}\right)$ which can be computed by* NC^1 *circuits of size* $\mathrm{poly}(n, 1/\epsilon, \log(1/\gamma))$.

Proof of Lemma 6.4. Let $\gamma = 0.8\varepsilon$, ϵ be a small constant. We apply the sampler to the $(n, \delta n)$-source X with independent seed U_1. By Lemma 4.10, $X_1 = X_{\mathrm{Samp}(U_1)}$ is $\gamma + 2^{-\epsilon n}$-close to a $(m, (\delta - \epsilon)m)$ source.

Since $m = O(\log(n/\varepsilon))$, we can apply extractor $\textsc{Ext}_1$ from Lemma 2.4 to X_1 with independent seed U_2. For any $(m, (\delta - \epsilon)m)$-source X_1, $(U_2, \textsc{Ext}_1(X_1, U_2))$ is ε-close to uniform. Since $m = O(\log(n/\varepsilon))$, the seed length of $\textsc{Ext}_1$ is $O(\log(n/\varepsilon))$, The output length is $(\delta - \epsilon)m - 2\log(n/\varepsilon) = \Omega(\log(n/\varepsilon))$.

The final extractor is $\textsc{Ext}(X, U_1, U_2) = \textsc{Ext}_1(X_{\mathrm{Samp}(U_1)}, U_2)$. It satisfies the requirement of the lemma.

The extractor from leftover hash lemma performs a matrix multiplication, which is computable in NC^1. The sampler is also computable in NC^1. Therefore, the extractor $\textsc{Ext}$ is computable in NC^1. □

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 third step of Section 4 to stretch the output to $O(\log^2 n \log(n/\varepsilon))$. The analysis is exactly the same.

This gives us the following lemma:

Lemma 6.6. *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* $\textsc{Ext} : \{0,1\}^n \times \{0,1\}^r(n) \to \{0,1\}^{m(n)}$ *computable in* 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 NC^1

With the seed of length $O(\log^2 n \log(n/\varepsilon))$, We apply the extractor from [RRV02] to the first block of the block source. Their extractor is given as follows:

Theorem 6.7 (Improved Trevisan's Extractor [RRV02]). *For every $k = k(n), \varepsilon = \varepsilon(n)$, there are explicit $(k(n), \varepsilon(n))$-extractors* $\textsc{Ext}_{Trev} : \{0,1\}^n \times \{0,1\}^{r(n)} \to \{0,1\}^{m(n)}$ *with $r(n) = O(\log^2(n)\log(n/\varepsilon))$ and $m(n) = \Omega(k(n))$.*

The construction of their extractor is as follows:

Given k and ε, regard the source X and the seed U as distributions on alphabet F instead of $\{0,1\}$. F is a finite field such that $\log|F| = O(\log(1/\varepsilon))$. Take $n' = n/\log|F|, r' = r/\log|F|, m' = m/\log|F|$. Then the extractor is defined as

$$\textsc{Ext}_{Trev} : F^{n'} \times F^{r'} \to F^{m'}, \tag{22}$$

where $r' = O(\log^2(n))$, $m' = \Omega(k/\log(1/\varepsilon))$.

Define $l = \log n'$. The source X is regarded as a multilinear function $f : F^l \to F$, which has $n' = 2^l$ coefficients.

Given the above parameters, there exists a polynomial time constructible set of sets $(S_1, \ldots, S_{m'})$, which is called a (l, ρ)-weak design in [RRV02]. Each S_i is a subset of $[r']$ and $|S_i| = l$.

The first step of the extractor is to apply the function f on the seed U restricted to each S_i:

$$Y_i = f(U|_{S_i}), \tag{23}$$

which forms a block source $Y = (Y_1, \ldots, Y_{m'})$.

The second step of the extractor is to apply one universal hash function $h : F \to \{0,1\}^{O(\log(1/\varepsilon))}$ to each Y_i. The output is the concatenation of the hash values:

$$W = h(Y_1)h(Y_2)\ldots h(Y_{m'}). \tag{24}$$

The extractor is defined as $\textsc{Ext}_{Trev}(f, U, h) = h(f(U|_{S_1}))h(f(U|_{S_2}))\ldots h(f(U|_{S_{m'}}))$.

Lemma 6.8. *The extractor* $\textsc{Ext}_{Trev}$ *is computable in* NC^1*.*

Proof. Let F be a finite field of characteristic two satisfying $\log|F| = O(\log(1/\varepsilon))$. The weak design are m' subsets of $[r']$, which could be described using $O(m'r') = O(n^2)$ bits. Therefore, we can hardwire the weak design into the circuit and compute $U|_{S_i}$ in NC^1.

The function f is a multilinear function, which has $O(n)$ terms of degree $O(\log n)$. Since each element of F can be represented by $O(n)$ bits, the iterated multiplication of $O(\log n)$ and summation of $O(n)$ terms are computable in NC^1 by [HV06, Theorem 3]. Therefore, the evaluation $f(U|_{S_i})$ is computable in NC^1.

Using Toeplitz matrice as hash functions, the hash function h is computable in NC^1. Therefore, the extractor $\textsc{Ext}_{Trev}$ is computable in NC^1. □

6.4 Putting it together

Now we can prove Theorem 6.1.

Proof of Theorem 6.1. Take X as the input source. Let $\mathrm{Cond} : \{0,1\}^n \times \{0,1\}^{r_1(n)} \to \{0,1\}^{m(n)}$ be the $(k, k+r_1, \varepsilon/4)$-condenser from Lemma 6.2. Take $(X_1, X_2) = \mathrm{Cond}(X, U_1)$, where U_1 is the seed of length $r_1 = O(\log(n/\varepsilon))$. By Lemma 2.13, (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 $\textsc{Ext}_1$ from Lemma 6.6 with seed U_2 of length $r_2 = O(\log(n/\varepsilon))$. The output is $Y = \textsc{Ext}_1(X_2, U_2)$ of length $O(\log^2(n)\log(n/\varepsilon))$.

For X_1, apply the $(\frac{1}{2}m(n), \varepsilon/4)$-extractor $\textsc{Ext}_{Trev}$ from Theorem 6.7 with seed Y, which outputs a distribution W of length $\Omega(k)$.

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

The extractor $\textsc{Ext}$ is defined as $\textsc{Ext}(X, U_1, U_2) = W$. The three components $\mathrm{Cond}, \textsc{Ext}_1, \textsc{Ext}_{Trev}$ are all computable in NC^1. Therefore, $\textsc{Ext}$ is computable in NC^1.

□

7 Entropy lower bound for AC^0 dispersers

In the context of AC^0 computation, not all sources are extractable. A well-known result of [GVW15] shows that extracting even one bit of randomness is impossible for sources with entropy less than $\frac{n}{\mathrm{poly}(\log n)}$. Similar result from [CL18] shows that extracting randomness with error less than $2^{-\mathrm{poly}(\log n)}$ is impossible for 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 dispersers for AC^0 sources with entropy less than $\frac{n}{\mathrm{poly}(\log n)}$ do not exist.

Definition 7.1 (Disperser)**.** *A function* Disp $: \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$ *is a* (k,ε)*-disperser if for every* k*-source* X *on* $\{0,1\}^n$ *and uniformly random variable* Y *on* $\{0,1\}^r$*,* $|\operatorname{Supp}(\operatorname{Disp}(X,Y))| \geq (1-\varepsilon)2^m$*.*

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

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

Our proof is based on the new switching lemma for AC^0 circuits by Rossman in [Ros]. 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 7.2 (Restrictions)**.** *A restriction* ρ *is a string on* $\{0,1,*\}^n$*. We denote the application of* ρ *to* $x \in \{0,1\}^n$ *by* $\rho \circ x$*, which is defined as:*

$$(\rho \circ x)_i = \begin{cases} \rho_i & \text{if } \rho_i \neq *, \\ x_i & \text{if } \rho_i = *. \end{cases} \tag{25}$$

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

$$f|_\rho(x) = f(\rho \circ x). \tag{26}$$

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 7.3 (Switching Lemma for AC^0 circuits [Ros])**.** *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*,*

$$\Pr_{\rho \sim R_p}[C|_\rho \text{ is not constant}] \leq \delta. \tag{27}$$

We give the following negative result for strong dispersers using the switching lemma:

Theorem 7.4. *For every* $d > 0, s = s(n)$*, every constant* $\delta \in (0,1)$*, if* $C : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}$ *is a* $(k, \frac{1}{2} - \delta)$*-disperser that can be computed by an* AC *circuit of size* s *and depth* d*, then* $k \geq \Theta(\frac{\delta n}{\log^{d-1} s})$*.*

Proof. Define the sub-circuit $C_y(x) = C(x,y)$ for every $y \in \{0,1\}^r$. Let R'_p the random restriction that $R'_p = R_p|_{R_p \text{ assigns at least } \frac{p}{2} \text{ fraction of the inputs as } *}$. The event that R_p assigns at least $\frac{p}{2}$ fraction of the inputs as $*$ is less than $\binom{n}{pn} / (\sum_{i < \frac{pn}{2}} \binom{n}{i}) \leq (\sqrt{2e}p)^{pn} = 2^{-\Omega(n)}$. Therefore, R'_p is $2^{-\Omega(n)}$-close to R_p.

By Lemma 7.3, there exists $p = \frac{\delta}{\Theta(\log s)^{d-1}}$ such that $C_y|_{R_p}$ is constant with probability at least $1 - \delta$. Then $C|_{R'_p}$ is constant with probability at least $1 - 2\delta$. Define a restriction ρ to be bad for y if $C_y|_\rho$ is constant. Then for every y, $\Pr_{\rho \sim R'_p}[\rho \text{ is bad for } y] \geq 1 - 2\delta$. By averaging, we have

$$\Pr_{\rho \sim R'_p, y \sim \{0,1\}^r}[\rho \text{ is bad for } y] \geq 1 - 2\delta. \tag{28}$$

Therefore, there exists a restriction ρ from R'_p such that for at least $1 - \delta$ fraction of $y \in \{0,1\}^r$, ρ is bad for y.

Define a source X to be the bit-fixing source on $\{0,1\}^n$ such that $X = \rho \circ U$, where U is a uniformly random variable on $\{0,1\}^n$. Then X is a k-source for $k = \frac{2n}{p} = \Theta(\frac{\delta n}{\log^{d-1} s})$. Since ρ is bad for at least $1-2\delta$ fraction of $y \in \{0,1\}^r$, $C_y(X)$ is constant for at least $1-2\delta$ fraction of $y \in \{0,1\}^r$. Therefore $(Y, C(X,Y)) = (Y, C_Y(X))$ covers at most $2\delta(2 \cdot 2^r) + (1-2\delta)2^r = (\frac{1}{2}+\delta)2^{r+1}$ points in its sample space, a contradiction to the definition of the strong disperser. So the theorem follows. □

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 ε? For example is it possible to give an NC^1 construction that can work for all k, ε, matching the parameters in [GUV09]?
- 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 constructed in AC^0, specifically for sources with entropy less than $\frac{n}{\text{poly}(\log n)}$.

References

[AB09] Sanjeev Arora and Boaz Barak. Computational complexity: a modern approach. Cambridge University Press, 2009.

[BCH86] 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.

[CL18] 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 fuer Informatik, 2018.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. Journal of Computer and System Sciences, 18:143–154, 1979.

[DKSS13] 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.

[GGH+07] 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.

[GUV09] 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.

[GVW15] 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 fuer Informatik.

[GW94] 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.

[Hea08] Alexander D Healy. Randomness-efficient sampling within nc1. Computational Complexity, 17(1):3–37, 2008.

[HV06] 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.

[ILL89] 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.

[KT22] 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.

[LRVW03] 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.

[NT99] Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. Journal of Computer and System Sciences, 58:148–173, 1999.

[NZ96] Noam Nisan and David Zuckerman. Randomness is linear in space. Journal of Computer and System Sciences, 52(1):43–52, 1996.

[Ros] Benjamin Rossman. An entropy proof of the switching lemma and tight bounds on the decision-tree size of AC0.

[RRV99] 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.

[RRV02] 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.

[RSW00] 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.

[RTS00] Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors and depth-two superconcentrators. Siam Journal on Discrete Mathematics, 13:2–24, 2000.

[Sha02] Ronen Shaltiel. Recent developments in explicit constructions of extractors. Bulletin of the European Association for Theoretical Computer Science, 77:67–95, 2002.

[Sha11] Ronen Shaltiel. An introduction to randomness extractors. In Proceedings of the 38th International Colloquium on Automata, Languages, and Programming, 2011.

[SZ99] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. SIAM Journal on Computing, 28:1433–1459, 1999.

[Ta-96] 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.

[Ta-98] 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.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. Journal of the ACM, 48(4):860–879, 2001.

[TSU12] 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.

[Vad03] Salil P. Vadhan. On Constructing Locally Computable Extractors and Cryptosystems in the Bounded Storage Model. In Advances in Cryptology - CRYPTO 2003, volume 2729, pages 61–77. Springer Berlin Heidelberg, 2003.

[Vad07] Salil Vadhan. The unified theory of pseudorandomness. SIGACT News, 38, 2007.

[Vad12] Salil Vadhan. Pseudorandomness. Foundations and Trends® in Theoretical Computer Science, 7(1–3):1–336, 2012.

[Vio05] Emanuele Viola. The complexity of constructing pseudorandom generators from hard functions. computational complexity, 13(3-4):147–188, 2005.

[WZ99] Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. Combinatorica, 19(1):125–138, 1999.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. Random Structures and Algorithms, 11(4):345–367, December 1997.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il