

Nearly-Linear Time Seeded Extractors with Short Seeds

Dean Doron*

João Ribeiro†

Abstract

Seeded extractors are fundamental objects in pseudorandomness and cryptography, and a deep line of work has designed polynomial-time seeded extractors with nearly-optimal parameters. However, existing constructions of seeded extractors with short seed length and large output length run in time $\Omega(n \log(1/\varepsilon))$ and often slower, where n is the input source length and ε is the error of the extractor. Since cryptographic applications of extractors require ε to be small, the resulting runtime makes these extractors impractical.

Motivated by this, we explore constructions of strong seeded extractors with short seeds computable in nearly-linear time $O(n \log^c n)$, for any error ε . We show that an appropriate combination of modern condensers and classical approaches for constructing seeded extractors for high min-entropy sources yields such extractors. More precisely, we obtain strong extractors for n -bit sources with any min-entropy k and any target error ε with seed length $d = O(\log(n/\varepsilon))$ and output length $m = (1 - \eta)k$ for an arbitrarily small constant $\eta > 0$, running in nearly-linear time. When k or ε are very small, our construction requires a reasonable one-time preprocessing step. These extractors directly yield privacy amplification protocols with nearly-linear time complexity (possibly after a one-time preprocessing step), large output length, and low communication complexity. As a second contribution, we give an instantiation of Trevisan’s extractor that can be evaluated in *truly* linear time in the RAM model, as long as the number of output bits is at most $\frac{n}{\log(1/\varepsilon) \text{polylog}(n)}$. Previous fast implementations of Trevisan’s extractor ran in $\tilde{O}(n)$ time in this setting.

*Ben-Gurion University. deand@bgu.ac.il. Part of this work was done while visiting Instituto de Telecomunicações and the Simons Institute for the Theory of Computing.

†Instituto de Telecomunicações and Departamento de Matemática, Instituto Superior Técnico, Universidade de Lisboa. jribeiro@tecnico.ulisboa.pt. Part of this work was done while at NOVA LINCS and NOVA School of Science and Technology, and while visiting the Simons Institute for the Theory of Computing.

Contents

1	Introduction	3
1.1	Our Contributions	4
1.2	Other Related Work	5
1.3	Technical Overview	6
1.4	Future Work	8
1.5	Acknowledgements	9
2	Preliminaries	9
2.1	Notation	9
2.2	Model of Computation	9
2.3	Fast Finite Field Operations	9
2.4	Statistical Distance, Entropy	10
2.5	Extractors and Condensers	11
2.6	Averaging Samplers	12
2.7	Standard Composition Techniques for Extractors	14
3	Additional Building Blocks	15
3.1	Fast Generation of Small-Bias Sets	15
3.2	A Sampler from Bounded Independence	16
3.3	Nearly-Linear Time Condensers	19
3.3.1	The Lossless KT Condenser	20
3.3.2	The Lossy RS Condenser	22
3.3.3	Towards removing preprocessing?	24
4	Nearly-Linear Time Extractors with Order-Optimal Seed Length	26
4.1	A Non-Recursive Construction	26
4.1.1	Item 2: Generating the block source	27
4.1.2	Item 3: Subsampling from the block source	30
4.1.3	Item 4: Applying a block source extractor	32
4.1.4	Improving the output length	33
4.2	A Recursive Construction	34
4.2.1	The (extremely) low-error case	34
4.2.2	The (relatively) high-error case	38
5	A Faster Instantiation of Trevisan’s Extractor	47

1 Introduction

Seeded randomness extractors are central objects in the theory of pseudorandomness. A strong (k, ε) -seeded extractor, first introduced by Nisan and Zuckerman [NZ96], is a deterministic function $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ that receives as input an n -bit source of randomness X with k bits of min-entropy¹ and a d -bit independent and uniformly random seed Y , and outputs an m -bit string $\text{Ext}(X, Y)$ that is ε -close in statistical distance to the uniform distribution over $\{0, 1\}^m$, where ε is an error term, even when the seed Y is revealed. Besides their most direct application to the generation of nearly-perfect randomness from imperfect physical sources of randomness (and their early applications to derandomizing space-bounded computation [NZ96] and privacy amplification [BBR88, BBCM95]), seeded extractors have also found many other surprising applications throughout computer science, particularly in cryptography (specifically, in leakage-resilient cryptography [SV19, QWW21] and non-malleable cryptography [BGW19, CGL20, AKO⁺22]).

For most applications, it is important to minimize the *seed length* of the extractor. A standard application of the probabilistic method shows the existence of strong (k, ε) -seeded extractors with seed length $d = \log(n - k) + 2\log(1/\varepsilon) + O(1)$ and output length $m = k - 2\log(1/\varepsilon) - O(1)$, and we also know that these parameters are optimal up to the $O(1)$ terms [RT00]. This motivated a deep line of research devising explicit constructions of seeded extractors with seed length as small as possible spanning more than a decade (e.g., [NZ96, SZ99, NT99, Tre01, TZS06, SU05]) and culminating in extractors with essentially optimal seed length [LRVW03, GUV09]. In particular, the beautiful work of Guruswami, Umans, and Vadhan [GVU09] gives explicit strong extractors with order-optimal seed length $d = O(\log(n/\varepsilon))$ and output length $m = (1 - \eta)k$ for any constant $\eta > 0$, and follow-up work [DKSS13, TU12] further improved m to $(1 - o(1))k$ (at the expense of higher error). The extractors constructed in these works are explicit, in the sense that there is an algorithm that given x and y computes the corresponding output $\text{Ext}(x, y)$ in time polynomial in the input length.

A closer look shows that the short-seed constructions presented in the literature all run in time $\Omega(n \log(1/\varepsilon))$, and often significantly slower. In cryptographic applications of extractors we want the error guarantee ε to be small, which means that implementations running in time $\Omega(n \log(1/\varepsilon))$ are often impractical. If we insist on nearly-linear runtime for arbitrary error ε , we can use strong seeded extractors based on universal hash functions that can be implemented in $O(n \log n)$ time (e.g., see [HT16]) and have essentially optimal output length, but have the severe drawback of requiring a very large seed length $d = \Omega(m)$.

These limitations have been noted in a series of works studying concrete implementations of seeded extractors, with practical applications in quantum cryptography in mind [MPS12, FWE⁺23, FYEC25]. For example, Foreman, Yeung, Edgington, and Curchod [FYEC25] implement a version of Trevisan’s extractor [Tre01, RRV02] with its standard instantiation of Reed–Solomon codes concatenated with the Hadamard code, and emphasize its excessive running time as a major reason towards non-adoption.² Instead, they have to rely on extractors based on universal hash functions, which, as mentioned above, are fast but require very large seeds.

This state of affairs motivates the following question, which is the main focus of this work:

Can we construct strong (k, ε) -seeded extractors with seed length $d = O(\log(n/\varepsilon))$ and output length $m = (1 - \eta)k$ computable in nearly-linear time, for arbitrary error ε ?

¹A random variable X has k bits of min-entropy if $\Pr[X = x] \leq 2^{-k}$ for all x . Min-entropy has been the most common measure for the quality of a weak source of randomness since the work of Chor and Goldreich [CG88].

²The reason why these works focus on Trevisan’s extractor is that this is the best seeded extractor (in terms of asymptotic seed length) that is known to be secure against quantum adversaries [DPVR12].

Progress on this problem would immediately lead to faster implementations of many cryptographic protocols that use seeded extractors, like those mentioned above – the most significant performance gains would be in the context of privacy amplification.

1.1 Our Contributions

We make progress on the construction of nearly-linear time seeded extractors.

Seeded extractors with order-optimal seed length and large output length. We construct nearly-linear time strong seeded extractors with order-optimal seed length and large output length for any k and ε , with the caveat that they require a reasonable one-time preprocessing step whenever k or ε are small. More precisely, we have the following result.

Theorem 1. *For any constant $\eta > 0$ there exists a constant $C > 0$ such that the following holds. For any positive integers n and $k \leq n$ and any $\varepsilon > 0$ satisfying $k \geq C \log(n/\varepsilon)$ there exists a strong (k, ε) -seeded extractor*

$$\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$$

with seed length $d \leq C \log(n/\varepsilon)$ and output length $m \geq (1 - \eta)k$. Furthermore,

1. *If $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$, where $\tilde{O}(\cdot)$ hides polylogarithmic factors in its argument and \log^* denotes the iterated logarithm;*
2. *If $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon < 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step, corresponding to generating $O(\log^* n)$ primes $q \leq \text{poly}(n/\varepsilon)$;*
3. *If $k < 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step, corresponding to generating $O(\log \log n)$ primes $q \leq \text{poly}(n/\varepsilon)$ and a primitive element for each field \mathbb{F}_q .*

The one-time preprocessing steps above can be implemented in time $\text{polylog}(n/\varepsilon)$ using randomness.³

Theorem 1 follows from combining modern condensers with short seeds (namely, the lossless condenser of Kalev and Ta-Shma [KT22] and the lossy Reed-Solomon-based condenser of Guruswami, Umans, and Vadhan [GUV09]) with a careful combination and instantiation of classical recursive approaches developed by Srinivasan and Zuckerman [SZ99] and in [GUV09]. It readily implies, among other things, an $\tilde{O}(n)$ -time privacy amplification protocol where only $O(\log(n/\varepsilon))$ bits need to be communicated over the one-way authenticated public channel and almost all the min-entropy can be extracted (after a reasonable one-time preprocessing step if k or ε are very small).

Remark 1.1 (complexity of the preprocessing steps). Both of the one-time preprocessing steps in **Theorem 1** are well-studied, and can be implemented in time $\text{polylog}(n/\varepsilon)$ using randomness. They are related to the condensers we use in the construction, and we discuss their complexity in more detail in **Remarks 3.11** and **3.16**, and some approaches (and barriers) towards derandomizing them in **Section 3.3.3**.

³At least naively, deterministic algorithms for the above one-time preprocessing steps require time $\text{poly}(n/\varepsilon)$, by testing primality of all numbers in an interval of length $\text{poly}(n/\varepsilon)$. The preprocessing step in **Item 2** can be replaced by a different known procedure that can be implemented deterministically in time $\tilde{O}(n + \sqrt{n} \log(1/\varepsilon)^{4+\delta})$ for any constant $\delta > 0$. Note that this is $\tilde{O}(n)$ when, say, $\varepsilon \geq 2^{-Cn^{0.1}}$, meaning that in this case its runtime can be absorbed into the runtime of the extractor, yielding **Item 1**. This constraint can be weakened to roughly $\varepsilon \geq 2^{-Cn^{0.16}}$ by using an alternative approach that we sketch in **Section 3.3.3**. To avoid overloading **Theorem 1**, we leave these discussions to **Remark 3.11** and **Section 3.3.3**, respectively.

A new non-recursive construction. As a conceptual contribution which may be of independent interest, we present a new “non-recursive” construction of extractors with seed length $O(\log(n/\varepsilon))$ and output length $(1 - \eta)k$ that is computable in nearly-linear time when $k > \text{polylog}(1/\varepsilon)$ and avoids the complicated recursive procedures from [SZ99, GUV09] and [Theorem 1](#). We believe this to be a conceptually better approach towards constructing seeded extractors, and we discuss it in more detail in the technical overview and fully in [Section 4.1](#).

Faster instantiations of Trevisan’s extractor. One of the most widely-used explicit seeded extractors is Trevisan’s extractor [Tre01, RRV02]. While by now we have extractors with better parameters, one of its main advantages is that it is one of the few examples of extractors, and in a sense the best one, which are known to be *quantum-proof*.⁴

Trevisan’s extractor uses two basic primitives: combinatorial designs (when more than one output bit is desired), and binary list-decodable codes. A standard instantiation of such suitable codes goes by concatenating a Reed-Solomon code with a Hadamard code, and this is also what is considered in [FWE⁺23, FYEC25]. As they also observe, this gives a nearly-linear time construction when the output length $m = 1$. In fact, by leveraging fast multipoint evaluation, one can also get a nearly-linear time construction for any output length $m \leq \frac{n}{\log(1/\varepsilon)}$, although this was not noted in previous works.⁵

We present an alternative instantiation of Trevisan’s extractor that can be computed in truly linear time on a RAM in the logarithmic cost model, for any output length $m \leq \frac{n}{\log(1/\varepsilon) \cdot \text{polylog}(n)}$. While the underlying technical details are simple, we opt to present this result here because it may be of interest to some readers due to wide interest in efficient implementations of Trevisan’s extractor, and because it was not observed in prior works.

Theorem 2. *There exists an instantiation of Trevisan’s extractor, set to extract m bits with any error $\varepsilon > 0$, that is computable in:*

1. *Time $O(n) + m \log(1/\varepsilon) \cdot \text{polylog}(n)$ after a preprocessing step⁶ running in time $\tilde{O}(m \log(n/\varepsilon))$, on a RAM in the logarithmic cost model. In particular, there exists a universal constant c , such that whenever $m \leq \frac{n}{\log(1/\varepsilon) \cdot \log^c(n)}$, the instantiation runs in time $O(n)$, without the need for a preprocessing step.*
2. *Time $\tilde{O}(n + m \log(1/\varepsilon))$ in the Turing model.*

We note that one interesting instantiation of the above theorem is when Trevisan’s extractor is set to output $k^{\Omega(1)}$ bits for $k = n^{\Omega(1)}$. In this setting, Trevisan’s extractor requires a seed of length $O\left(\frac{\log^2(n/\varepsilon)}{\log(1/\varepsilon)}\right)$, and, as long as ε is not too tiny, we get truly-linear runtime.

1.2 Other Related Work

Besides the long line of work focusing on improved constructions of explicit seeded extractors and mentioned in the introduction above, other works have studied randomness extraction in a variety of restricted computational models. These include extractors computable by streaming algorithms [BRST02], local algorithms [Lu02, Vad04, BG13, CL18], AC⁰ circuits [GVW15, CL18,

⁴An extractor is quantum-proof if its output is close to uniform even in the presence of a quantum adversary that has some (bounded) correlation with X . A bit more formally, Ext is quantum-proof if for all classical-quantum states ρ_{XE} (where E is a quantum state correlated with X) with $H_\infty(X|E) \geq k$, and a uniform seed Y , it holds that $\rho_{\text{Ext}(X,Y)YE} \approx_\varepsilon \rho_{U_m} \otimes \rho_Y \otimes \rho_E$. See [DPVR12] for more details.

⁵For a rigorous statement on fast multipoint evaluation, see [Lemma 2.1](#).

⁶This preprocessing step corresponds to precomputing the design, and is *not* the same preprocessing step as in [Theorem 1](#).

CW24], AC^0 circuits with a layer of parity gates [HIV22], NC^1 circuits [CW24], and low-degree polynomials [ACG⁺22, AGMR25, GGH⁺24]. Moreover, some works have independently explored implementations of other fundamental pseudorandomness primitives in various restricted computational models. These include k -wise and ε -biased generators, which often play a key role in constructions of various types of extractors. See [HV06, Hea08, CRSW13, MRRR14] for a very partial list.

As mentioned briefly above, some works have also focused on constructing seeded extractors computable in time $O(n \log n)$, motivated by applications in privacy amplification for quantum key distribution. Such constructions are based on hash functions, and are thus far restricted to $\Omega(m)$ seed length. The work of Hayashi and Tsurumaru [HT16] presents an extensive discussion of such efforts. We also mention that nearly-linear time extractors with very short seed, in the regime $k = n^{\Omega(1)}$ and $\varepsilon = n^{-o(1)}$, were given in [DMOZ22], with applications in derandomization.⁷

The techniques in this paper build on, and extend, block-source extraction techniques [NZ96, Zuc96, Zuc97, SZ99, GUV09]. Another line of work, notably including [NT99, LRVW03, DW08, DKSS13, TU12], utilizes *mergers* to construct seeded extractors.⁸ However, when restricted to constructions that get optimal seed length, they generally do not support low error (let alone run in nearly-linear time). On the positive side, the state-of-the-art mergers-based constructions get sub-linear entropy loss [DKSS13, TU12], whereas the constructions in this paper do not. We stress that it is still a very interesting open problem to construct a low-error, optimal seed length extractor with sub-linear entropy loss.

1.3 Technical Overview

In a nutshell, we obtain [Theorem 1](#) by following two standard high-level steps:

1. We apply a randomness condenser with small seed length $O(\log(n/\varepsilon))$ to the original n -bit weak source X to obtain an output X' that is ε -close to a high min-entropy source.
2. We apply a seeded extractor tailored to high min-entropy sources with small seed length $O(\log(n/\varepsilon))$ to X' to obtain a long output that is ε -close to uniform.

To realize this approach, we need to implement each of these steps in nearly-linear time $\tilde{O}(n)$ (possibly after a reasonable one-time preprocessing step). We briefly discuss how we achieve this, and some pitfalls we encounter along the way.

Observations about nearly-linear time condensers. In order to implement [Item 1](#), we need to use *fast* condensers with short seeds. Luckily for us, some existing state-of-the-art constructions of condensers can already be computed in nearly-linear time, although, to the best of our knowledge, this has not been observed before. We argue this carefully in [Section 3.3](#).

For example, the “lossy Reed-Solomon condenser” from [GUV09] interprets the source as a polynomial $f \in \mathbb{F}_q[x]$ of degree $d \leq n/\log q$ and the seed y as an element of \mathbb{F}_q , and outputs $\text{RSCond}(f, y) = (f(y), f(\zeta y), \dots, f(\zeta^{m'-1}y))$, for an appropriate m' and field size q , with ζ a primitive element of \mathbb{F}_q . Evaluating $\text{RSCond}(f, y)$ corresponds to evaluating the same polynomial f on

⁷Our extractor can replace the one constructed in [DMOZ22], and it is indeed more efficient. However, due to other bottlenecks, they need to work with high error $\varepsilon = n^{-o(1)}$ and relatively large min-entropy, and so the difference between the two extractors is not significant.

⁸For the definition of mergers, see, e.g., [DKSS13]. We note that those works *do* often use block source conversion techniques, but usually in a different manner.

multiple points in \mathbb{F}_q . This is an instance of the classical problem of multipoint evaluation in computational algebra, for which we know fast and practical algorithms (e.g., see [vzGG13, Chapter 10] or [Lemma 2.1](#)) running in time $\tilde{O}((d + m') \log q) = \tilde{O}(n)$, since $d \leq n/\log q$, and if $m' \leq n/\log q$.

A downside of this condenser is that it requires knowing a primitive element ζ of \mathbb{F}_q with $q = \text{poly}(n/\varepsilon)$. But note that finding this primitive element only needs to be done once for a given set of parameters (n, k, ε, m) independently of the actual seed and input source, and so we leave it as a one-time preprocessing step. As discussed in [Remark 3.16](#), we have the freedom of choosing q to be prime, and in that case we can find this primitive element in randomized time $\text{polylog}(q) = \text{polylog}(n/\varepsilon)$.

The lossless “KT condenser” from [KT22] has a similar flavor. It interprets the source as a polynomial $f \in \mathbb{F}_q[x]$ and the seed y as an evaluation point, and outputs $\text{KTCond}(f, y) = (f(y), f'(y), \dots, f^{(m'-1)}(y))$, for some appropriate m' . The problem of evaluating several derivatives of the same polynomial f on the same point y (sometimes referred to as Hermite evaluation) is closely related to the multipoint evaluation problem above, and can also be solved in time $\tilde{O}(n)$.⁹ Furthermore, evaluating the KT condenser only requires preprocessing when ε is very small. On the other hand, it only works when the min-entropy $k \geq C \log^2(n/\varepsilon)$ for a large constant $C > 0$, where n is the source length and ε the target error of the condenser.

The “ideal” approach to seeded extraction from high min-entropy sources. We have seen that there are fast condensers with short seeds. It remains to realize [Item 2](#). Because of the initial condensing step, we may essentially assume that our n -bit weak source X has min-entropy $k \geq (1 - \delta)n$, for an arbitrarily small constant $\delta > 0$. In this case, we would like to realize in time $\tilde{O}(n)$ and with overall seed length $O(\log(n/\varepsilon))$ what we see as the most natural approach to seeded extraction from high min-entropy sources:

1. Use a fresh short seed to transform X into a *block source* $Z = (Z_1, Z_2, \dots, Z_t)$ with geometrically decreasing blocks. A block source has the property that each block Z_i has good min-entropy even conditioned on the values of blocks Z_1, \dots, Z_{i-1} .
2. Perform *block source extraction* on Z using another fresh short seed. Due to its special structure, we can extract a long random string from Z using only the (small) seed length associated with extracting randomness from the smallest block Z_t .

Similar approaches were taken in [NZ96, Zuc96], but they do not support logarithmic seed and low-error (see [Section 4.1](#)). The classical approach to [Item 2](#) where we iteratively apply extractors based on universal hash functions with increasing output lengths to the blocks of Z from right to left is easily seen to run in time $\tilde{O}(n)$ and requires a seed of length $O(\log(n/\varepsilon))$ if, e.g., we use the practical extractors of [TSSR11, HT16]. Therefore, we only need to worry about realizing [Item 1](#).

A standard approach to [Item 1](#) would be to use an *averaging sampler* to iteratively sample subsequences of X as the successive blocks of the block source Z , following a classical strategy of Nisan and Zuckerman [NZ96] (improved by [RSW06, Vad04]). We do know averaging samplers running in time $\tilde{O}(n)$ (such as those based on random walks on a carefully chosen expander graph). However, this approach requires a fresh seed of length $\Theta(\log(n/\varepsilon))$ *per block of Z* . Since Z will have roughly $\log n$ blocks, this leads to an overall seed of length $\Theta(\log^2 n + \log(1/\varepsilon))$, which is too much for us.

⁹Interestingly, recent works used other useful computational properties of the KT condenser. Cheng and Wu [CW24] crucially use the fact that the KT condenser can be computed in NC^1 . Doron and Tell [DT23] use the fact that the KT condenser is logspace computable for applications in space-bounded derandomization.

Instead, we provide a new analysis of a sampler based on bounded independence, that runs in time $\tilde{O}(n)$ and only requires a seed of length $O(\log(n/\varepsilon))$ to create the *entire* desired block source. However, this block source has blocks of *increasing* lengths, whereas we need decreasing blocks to perform the block source extraction. We remedy that by sub-sampling from each block using a standard expander random walk sampler.

We give the construction, which may be of independent interest, in [Section 3.2](#). The caveat of this construction is that it only works as desired when the target error $\varepsilon \geq 2^{-k^c}$ for some small constant $c > 0$. See [Section 4.1](#) for the formal analysis.

Getting around the limitation of the ideal approach. We saw above that combining the ideal approach to seeded extraction from high min-entropy sources with the new analysis of the bounded independence sampler yields a conceptually simple construction with the desired properties when the error is not too small. However, we would like to have $\tilde{O}(n)$ -time seeded extraction with $O(\log(n/\varepsilon))$ seed length and large output length for all ranges of parameters.

To get around this limitation of our first construction, it is natural to turn to other classical approaches for constructing nearly-optimal extractors for high min-entropy sources, such as those of Srinivasan and Zuckerman [[SZ99](#)] or Guruswami, Umans, and Vadhan [[GUV09](#)]. These approaches consist of intricate recursive procedures combining a variety of combinatorial objects, and require a careful analysis.¹⁰ However, we could not find such an approach that works as is, even when instantiated with $\tilde{O}(n)$ -time condensers and $\tilde{O}(n)$ -time hash-based extractors. In particular:

- The GUV approach [[GUV09](#)] gives explicit seeded extractors with large output length and order-optimal seed length for *any* min-entropy requirement k and error ε . However, its overall runtime is significantly larger than $\tilde{O}(n)$ whenever ε is not extremely small (for example, $\varepsilon = 2^{-k^\alpha}$ for some $\alpha \in (0, 1/2)$ is not small enough).
- The SZ approach [[SZ99](#)] can be made to run in time $\tilde{O}(n)$ and have large output length when instantiated with fast condensers, samplers, and hash-based extractors, but it is constrained to error $\varepsilon \geq 2^{-ck/\log^* n}$, where \log^* is the iterated logarithm.

Fortunately, the pros and cons of the GUV and SZ approaches complement each other. Therefore, we can obtain our desired result by applying appropriately instantiated versions of the GUV and SZ approaches depending on the regime of ε we are targeting.

1.4 Future Work

We list here some directions for future work:

- Remove the preprocessing step that our constructions behind [Theorem 1](#) require when k or ε are small. We expand on some promising approaches suggested by Jesse Goodman and also some barriers we face in [Section 3.3.3](#).
- On the practical side, develop software implementations of seeded extractors with near-optimal seed length and large output length. In particular, we think that our non-recursive construction in [Section 4.1](#) holds promise in this direction.

¹⁰In our view, these approaches are much less conceptually appealing than the “ideal” approach above. We believe that obtaining conceptually simpler constructions of fast nearly-optimal extractors that work for all errors is a worthwhile research direction, even if one does not improve on the best existing parameters.

1.5 Acknowledgements

We thank Jesse Goodman for many valuable comments and suggestions that greatly improved this work, and in particular for suggesting the approach outlined in [Section 3.3.3](#).

Part of this research was done while the authors were visiting the Simons Institute for the Theory of Computing, supported by DOE grant #DE-SC0024124. D. Doron’s research was also supported by Instituto de Telecomunicações (ref. UIDB/50008/2020) with the financial support of FCT - Fundação para a Ciência e a Tecnologia and by NSF-BSF grant #2022644. J. Ribeiro’s research was also supported by Instituto de Telecomunicações (ref. UIDB/50008/2020) and NOVA LINCS (ref. UIDB/04516/2020) with the financial support of FCT - Fundação para a Ciência e a Tecnologia.

2 Preliminaries

2.1 Notation

We often use uppercase Roman letters to denote sets and random variables – the distinction will be clear from context. We denote the support of a random variable X by $\text{supp}(X)$, and for a random variable X and set S , we also write $X \sim S$ to mean that X is supported on S . For a random variable X , we write $x \sim X$ to mean that x is sampled according to the distribution of X . We use U_d to denote a random variable that is uniformly distributed over $\{0, 1\}^d$. For two strings x and y , we may write (x, y) for their concatenation. Given two random variables X and Y , we denote their product distribution by $X \times Y$ (i.e., $\Pr[X \times Y = (x, y)] = \Pr[X = x] \cdot \Pr[Y = y]$). Given a positive integer n , we write $[n] = \{1, \dots, n\}$. For a prime power q , we denote the finite field of order q by \mathbb{F}_q . We denote the base-2 logarithm by \log .

2.2 Model of Computation

We work in the standard, multi-tape, Turing machine model with some fixed number of work tapes. In particular, there exists a constant C such that all our claimed time bounds hold whenever we work with at most C work tapes. This also implies that our results hold in the RAM model, wherein each machine word can store integers up to some fixed length, and standard word operations take constant time. In [Section 5](#) we will give, in addition to the standard Turing machine model bounds, an improved runtime bound that is dedicated to the logarithmic-cost RAM model.

2.3 Fast Finite Field Operations

Understanding the complexity of operations in finite fields will be useful for the analysis of the complexity of the condensers from [\[GUV09, KT22\]](#) in [Section 3.3](#). For a prime power $q = p^\ell$, we let $M_q(d)$ be the number of field operations required to multiply two univariate polynomials over \mathbb{F}_q of degree less than d , and $M_q^b(d)$ be the bit complexity of such a multiplication, so $M_q^b(d) \leq M_q(d) \cdot T(q)$, where we denote by $T(q)$ an upper bound on the bit complexity of arithmetic operations in \mathbb{F}_q . Harvey and van der Hoeven [\[HvdH19\]](#) (see also [\[HvdH22\]](#)) showed that

$$M_q^b(d) = O(d \log q \cdot \log(d \log q) \cdot 4^{\log^*(d \log q)}).$$

Overall, when $d \leq q$, we have that $M_q^b(d) = d \log d \cdot \tilde{O}(\log q)$.¹¹

¹¹A similar bound can be obtained using simpler methods. If \mathbb{F}_q contains a d -th root of unity, one can get $M_q(d) = d \log d$ from the classic FFT algorithm [\[CT65\]](#). For a simpler algorithm attaining the bound $M_q(d) = d \log d \log \log d$,

We will use fast multi-point evaluation and fast computation of derivatives (together with the preceding bounds on M_q^b).

Lemma 2.1 ([BM74], see also [vzGG13, Chapter 10]). *Let $d \in \mathbb{N}$, and let $q = p^r$ be a prime power. Then, given a polynomial $f \in \mathbb{F}_q[X]$ of degree at most d (together with a representation of \mathbb{F}_q via an irreducible polynomial over \mathbb{F}_p of degree r), the following holds.*

1. *Given a set $\{\alpha_1, \dots, \alpha_t\} \subseteq \mathbb{F}_q$, where $t \leq d$, one can compute $f(\alpha_1), \dots, f(\alpha_t)$ in time $O(M_q^b(d) \cdot \log d) = d \log^2 d \cdot \tilde{O}(\log q)$.*
2. *For $t \leq d$ and $\alpha \in \mathbb{F}_q$, one can compute the derivatives $f(\alpha), f'(\alpha), \dots, f^{(t)}(\alpha)$ in time $O(M_q^b(d) \cdot \log d) = d \log^2 d \cdot \tilde{O}(\log q)$.*

Note that when $q \leq 2^d$, we can bound $O(M_q(d) \cdot \log d)$ by $\tilde{O}(d) \cdot \log q$.

For a comprehensive discussion of fast polynomial arithmetic, see Von Zur Gathen and Gerhard's book [vzGG13] (and the more recent important developments [HvdH21]).

2.4 Statistical Distance, Entropy

We present some relevant definitions and lemmas about statistical distance and min-entropy.

Definition 2.2 (statistical distance). *The statistical distance between two random variables X and Y supported on \mathcal{S} , denoted by $\Delta(X, Y)$, is defined as*

$$\Delta(X, Y) = \max_{\mathcal{T} \subseteq \mathcal{S}} |\Pr[X \in \mathcal{T}] - \Pr[Y \in \mathcal{T}]| = \frac{1}{2} \sum_{x \in \mathcal{S}} |\Pr[X = x] - \Pr[Y = x]|.$$

We say that X and Y are ε -close, and write $X \approx_\varepsilon Y$, if $\Delta(X, Y) \leq \varepsilon$.

Definition 2.3 (min-entropy). *The min-entropy of a random variable X supported on \mathcal{X} , denoted by $\mathbf{H}_\infty(X)$, is defined as*

$$\mathbf{H}_\infty(X) = -\log \left(\max_{x \in \mathcal{X}} \Pr[X = x] \right).$$

The min-entropy rate of X is given by $\frac{\mathbf{H}_\infty(X)}{\log |\mathcal{X}|}$.

Definition 2.4 (average conditional min-entropy). *Let X and Y be two random variables supported on \mathcal{X} and \mathcal{Y} , respectively. The average conditional min-entropy of X given Y , denoted by $\tilde{\mathbf{H}}_\infty(X|Y)$, is defined as*

$$\tilde{\mathbf{H}}_\infty(X|Y) = -\log \left(\mathbb{E}_{y \sim Y} [2^{-\mathbf{H}_\infty(X|Y=y)}] \right).$$

The following standard lemma gives a chain rule for min-entropy.

Lemma 2.5 (see, e.g., [DORS08]). *Let X , Y , and Z be arbitrary random variables such that $|\text{supp}(Y)| \leq 2^\ell$. Then,*

$$\tilde{\mathbf{H}}_\infty(X|Y, Z) \geq \tilde{\mathbf{H}}_\infty(X|Z) - \ell.$$

We can turn the chain rule above into a high probability statement.

Lemma 2.6 (see, e.g., [MW97]). *Let X , Y , and Z be random variables such that $|\text{supp}(Y)| \leq 2^\ell$. Then,*

$$\Pr_{y \sim Y} [\tilde{\mathbf{H}}_\infty(X|Y = y, Z) \geq \tilde{\mathbf{H}}_\infty(X|Z) - \ell - \log(1/\delta)] \geq 1 - \delta$$

for any $\delta > 0$.

see [vzGG13, Sections 8, 10]. When $p = 2$, $M_q(d) = d \log d \log \log d$ also follows from Schönhage's algorithm [Sch77]. Now, since $M_q^b(d) = M_q(d) \cdot M_p(\ell) \cdot M_p^b(0)$, a bound of $d \log d \cdot \tilde{O}(\log q)$ also follows.

2.5 Extractors and Condensers

Definition 2.7 ($((n, k)$ -source). We say that a random variable X is an (n, k) -source if $X \sim \{0, 1\}^n$ and $\mathbf{H}_\infty(X) \geq k$.

Definition 2.8 (block source). A random variable X is an $((n_1, n_2, \dots, n_t), (k_1, k_2, \dots, k_t))$ -block source if we can write $X = (X_1, X_2, \dots, X_t)$, each $X_i \in \{0, 1\}^{n_i}$, where $\mathbf{H}_\infty(X_i | X_1, \dots, X_{i-1}) \geq k_i$ for all $i \in [t]$. In the special case where $k_i = \alpha n_i$ for all $i \in [t]$, we say that X is an $((n_1, n_2, \dots, n_t), \alpha)$ -block source.

We say that X is an exact block source if $\mathbf{H}_\infty(X_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1}) \geq k_i$ for any prefix x_1, \dots, x_{i-1} . **Lemma 2.6** tells us that any $((n_1, n_2, \dots, n_t), \alpha)$ -block-source is ε -close to an exact $((n_1, n_2, \dots, n_t), (1 - \zeta)\alpha)$ -block-source, where $\varepsilon = \sum_{i=1}^t 2^{-\alpha \zeta n_i}$.

Definition 2.9 (seeded extractor). A function $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is a (k, ε) seeded extractor if the following holds. For every (n, k) -source X ,

$$\text{Ext}(X, Y) \approx_\varepsilon U_m,$$

where Y is uniformly distributed over $\{0, 1\}^d$ and is independent of X and U_m is uniformly distributed over $\{0, 1\}^m$. We say that Ext is strong if $(\text{Ext}(X, Y), Y) \approx_\varepsilon U_{m+d}$.

Furthermore, Ext is said to be an average-case (k, ε) (strong seeded) extractor if for all correlated random variables X and W such that X is supported on $\{0, 1\}^n$ and $\mathbf{H}_\infty(X | W) \geq k$ we have

$$(\text{Ext}(X, Y), Y, W) \approx_\varepsilon (U_{m+d}, W),$$

where Y is uniformly distributed over $\{0, 1\}^d$ and is independent of X , and U_{m+d} is uniformly distributed over $\{0, 1\}^{m+d}$ and independent of W .

Remark 2.10. By **Lemma 2.6**, every strong (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is also an average-case strong $(k' = k + \log(1/\varepsilon), \varepsilon' = 2\varepsilon)$ -seeded extractor.

Definition 2.11 (condenser). A function $\text{Cond}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is a (k, k', ε) (seeded) condenser if the following holds. For every (n, k) -source X , it holds that $Z = \text{Cond}(X, Y)$ is ε -close to some Z' with $\mathbf{H}_\infty(Z') \geq k'$, where Y is uniformly distributed over $\{0, 1\}^d$ and is independent of X .

We say that Cond is strong if $(Y, \text{Cond}(X, Y))$ is ε -close to some distribution (Y, Z) with min-entropy k' (and note that here, necessarily, d bits of entropy come from the seed). Finally, we say that Cond is lossless if $k' = k + d$.

We also define seeded extractors tailored to block sources.

Definition 2.12 (block source extractor). A function $\text{BExt}: \{0, 1\}^{n_1} \times \dots \times \{0, 1\}^{n_t} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ is a $(k_1, \dots, k_t, \varepsilon)$ strong block-source extractor if for any $((n_1, \dots, n_t), (k_1, \dots, k_t))$ -block-source X ,

$$(\text{BExt}(X, Y), Y) \approx_\varepsilon U_{m+d},$$

where Y is uniformly distributed over $\{0, 1\}^d$ and is independent of X and U_{m+d} is uniformly distributed over $\{0, 1\}^{m+d}$.

We will also require the following extractors based on the leftover hash lemma and fast hash functions. We state a result from [TSSR11] which requires seed length $d \approx 2m$, where m is the output length.

Lemma 2.13 (fast hash-based extractors [TSSR11, Theorem 10], adapted. See also [HT16, Table I]). *For any positive integers n , k , and m and any $\varepsilon > 0$ such that $k \leq n$ and $m \leq k - 4\log(16/\varepsilon)$ there exists a (k, ε) -strong seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with seed length $d \leq 2(m + \log n + 2\log(1/\varepsilon) + 4)$. Moreover, Ext can be computed in time $O(n \log n)$.*

Note that by appending the seed to the output of the extractor, we can get the following: There exists a constant c such that for any constant $\theta \leq \frac{1}{3}$, $d \geq c \log(n/\varepsilon)$ and $k \geq \theta d + c \log(1/\varepsilon)$, there exists a (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{(1+\theta)d}$.

2.6 Averaging Samplers

In this section we define averaging samplers and state some useful related results and constructions.

Definition 2.14 (averaging sampler). *We say that $\text{Samp}: \{0, 1\}^r \rightarrow [n]^m$ is a (γ, θ) -averaging sampler if*

$$\Pr_{(i_1, \dots, i_m) \sim \text{Samp}(U_r)} \left[\left| \frac{1}{t} \sum_{j=1}^m f(i_j) - \mathbb{E}[f] \right| \geq \theta \right] < \gamma$$

for every function $f: [n] \rightarrow [0, 1]$, where $\mathbb{E}[f] = \frac{1}{n} \sum_{i \in [n]} f(i)$. We say that Samp has distinct samples if $\text{Samp}(x)$ outputs m distinct elements of $[n]$ for every input x . The parameter θ is often referred to as the accuracy of the sampler, and γ is its confidence parameter. Moreover, we sometimes refer to $\text{Samp}(U_r) \sim [n]^m$ as a (γ, θ) sampling distribution.

It is worth noting the equivalence between averaging samplers and extractors [Zuc97], where we think of the samples obtained by enumerating over the seeds of the extractor. Specifically, a (k, ε) -seeded extractor over inputs of length r is a $(2^{k-r+1}, \varepsilon)$ -averaging sampler, and a (γ, θ) -averaging sampler with input length r is a $(k, 2\theta)$ -seeded extractor for $k = n - \log(1/\gamma) + \log(1/\theta)$.

The following lemma gives guarantees on sub-sampling coordinates from a weak source using an averaging sampler.

Lemma 2.15 ([Vad04, Lemma 6.2]). *Let $\delta, \gamma, \tau \in (0, 1)$ be such that $\delta \geq 3\tau$ and let $\text{Samp}: \{0, 1\}^r \rightarrow [n]^m$ be a $(\gamma, \theta = \tau / \log(1/\tau))$ -averaging sampler with distinct samples. Then, for any $(n, k = \delta n)$ -source X and Y uniformly distributed over $\{0, 1\}^r$ we have that*

$$(Y, X_{\text{Samp}(Y)}) \approx_{\gamma+2^{-\Omega(\tau n)}} (Y, W),$$

where $(W|Y = y)$ is an $(m, k' = (\delta - 3\tau)m)$ -source for every y .

The “expander random walk” sampler. We will need the following well-known averaging sampler based on random walks on expanders (see, e.g., [Gil98, Zuc07]). Let G be a D -regular graph with vertex set $[n]$. We assume that the neighborhood of each vertex is ordered in some predetermined way. Then, the associated averaging sampler parses its input x as $(i_1, b_1, b_2, \dots, b_{t-1})$, where $i_1 \in [n]$ and $b_1, \dots, b_{t-1} \in [D]$, and outputs $\text{Samp}(x) = (i_1, \dots, i_t)$, where i_j is the b_{j-1} -th neighbor of i_{j-1} when $j > 1$.

The performance of Samp as an averaging sampler is determined by the spectral expansion of G .¹² In fact, if G has spectral expansion $\theta/2$ then a direct application of the expander Chernoff bound [Gil98] gives that Samp is a (γ, θ) -averaging sampler with $t = O(\log(1/\gamma)/\theta^2)$ and $r = \log n + O(t \log(1/\theta))$ [Vad04, Section 8.2].

¹²We say that an undirected graph G over n vertices has spectral expansion λ if $\lambda \leq \max_{i \geq 2} |\lambda_i|$, where $\lambda_n \leq \dots \leq \lambda_2 \leq \lambda_1 = 1$ are the eigenvalues of G 's random walk matrix.

To ensure distinct samples while maintaining accuracy, we follow [Vad04] and modify the standard random walk sampler as follows. As a “base” sampler, we use the above walk, but only take the first $(1 - \theta/2)t$ distinct vertices (if there are such). Letting $r_0 = \log n + O(t \log(1/\theta))$ be the corresponding randomness complexity, we then take a random walk of length $\ell = O(\log(1/\gamma))$ over an expander G_0 with 2^{r_0} vertices and constant spectral expansion. Each vertex of G_0 corresponds to a *random walk on G* , and out of those ℓ chosen walks, we take the first one that indeed has $(1 - \theta/2)t$ distinct vertices (and output some arbitrary value if the process failed). In [Vad04, Lemma 8.2], it is shown that setting the spectral expansion of G to be $\Theta(\theta)$ (and suitably choosing the constants inside the $O()$ notation), one still gets a (γ, θ) -averaging sampler.

We instantiate G and G_0 with the regular expander graphs from the following result of Alon [Alo21].

Lemma 2.16 ([Alo21, Theorem 1.2], adapted). *Fix any prime p such that $p \equiv 1 \pmod{4}$. Then, there is a constant C_p such that for every integer $n \geq C_p$ there exists a $(D = p + 2)$ -regular graph G_n on n vertices with spectral expansion $\lambda \leq \frac{(1+\sqrt{2})\sqrt{D-1+o(1)}}{D}$, where the $o(1)$ tends to 0 as $n \rightarrow \infty$. Furthermore, the family $(G_n)_n$ is strongly explicit.*

In particular, for any $\theta > 0$ there exist constants $C_\theta > 0$ and $D_\theta = O(\theta^{-2})$ and a strongly explicit family of D_θ -regular graphs $(G_n)_n$ with spectral expansion $\lambda \leq \theta$ for any $n \geq C_\theta$.

Given a graph G , its t -th power G^t is a graph over the same number of vertices, and each edge corresponds to a t -step walk over G . It is well-known that taking the t -th power of a λ -spectral expander improves its expansion to λ^t . This readily gives us the following corollary.

Corollary 2.17. *For every large enough n , and any $\lambda = \lambda(n) > 0$, there exists a D -regular graph $G = (V = [n], E)$ with spectral expansion λ , where $D = \text{poly}(1/\lambda)$, and given $x \in [n]$ and $i \in [D]$, the i -th neighbor of x can be computed in time $\log(1/\lambda) \cdot \text{polylog}(n) = \text{polylog}(n)$.*

Combining the discussion above with Lemma 2.16 (or Corollary 2.17) immediately yields the following, observing that the runtime is dominated by $\ell \cdot t \log(1/\lambda) \text{polylog}(n)$.

Lemma 2.18 ([Vad04, Lemma 8.2], appropriately instantiated). *For every large enough integer n and every $\theta, \gamma \in (0, 1)$, there exists a (γ, θ) -averaging sampler $\text{Samp}: \{0, 1\}^r \rightarrow [n]^t$ with distinct samples with $t = O(\log(1/\gamma)/\theta^2)$ and $r = \log n + O(t \log(1/\theta))$. Furthermore, Samp is computable in time $t \log(1/\gamma) \cdot \text{polylog } n$.*

We can extend Lemma 2.18 to output more distinct samples while not increasing r via the following simple lemma.

Lemma 2.19 ([Vad04, Lemma 8.3]). *Suppose that $\text{Samp}_0: \{0, 1\}^r \rightarrow [n]^t$ is a (γ, θ) -averaging sampler with distinct samples. Then, for every integer $m \geq 1$ there exists a (γ, θ) -averaging sampler $\text{Samp}: \{0, 1\}^r \rightarrow [m \cdot n]^{m \cdot t}$ with distinct samples.*

Lemma 2.19 follows easily by parsing $[m \cdot t] = [m] \times [t]$ and considering the sampler $\text{Samp}(x)_{i,j} = (i, \text{Samp}_0(x)_j)$ for $i \in [m]$ and $j \in [t]$. If we can compute $\text{Samp}_0(x)$ in time T , then we can compute $\text{Samp}(x)$ in time $T + O(mt \log(mn))$. The following is an easy consequence of Lemmas 2.18 and 2.19.

Lemma 2.20 ([Vad04, Lemma 8.4], with additional complexity claim). *There exists a constant $C > 0$ such that the following holds. For every large enough n and $\theta, \gamma \in (0, 1)$, there exists a (γ, θ) -averaging sampler $\text{Samp}: \{0, 1\}^r \rightarrow [n]^t$ with distinct samples for any $t \in [t_0, n]$ with $t_0 \leq C \log(1/\gamma)/\theta^2$ and $r = \log(n/t) + \log(1/\gamma) \cdot \text{poly}(1/\theta)$. Furthermore, Samp is computable in time $t_0 \log(1/\gamma) \cdot \text{polylog } n + O(t \log n)$.*

In particular, if θ is constant then $t_0 = O(\log(1/\gamma))$, $r = \log(n/t) + O(\log(1/\gamma))$, and Samp is computable in time $\log^2(1/\gamma) \cdot \text{polylog } n + O(t \log n)$.

2.7 Standard Composition Techniques for Extractors

We collect some useful classical techniques for composing seeded extractors.

Lemma 2.21 (boosting the output length [WZ99, RRV02]). *Suppose that for $i \in \{1, 2\}$ there exist strong (k_i, ε_i) -seeded extractors $\text{Ext}_i: \{0, 1\}^n \times \{0, 1\}^{d_i} \rightarrow \{0, 1\}^{m_i}$ running in time T_i , with $k_2 \leq k_1 - m_1 - g$. Then, $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^{d_1+d_2} \rightarrow \{0, 1\}^{m_1+m_2}$ given by $\text{Ext}(X, (Y_1, Y_2)) = (\text{Ext}_1(X, Y_1), \text{Ext}_2(X, Y_2))$ is a strong $(k_1, \frac{\varepsilon_1}{1-2^{-g}} + \varepsilon_2)$ -seeded extractor running in time $O(T_1 + T_2)$.*

Lemma 2.22 (block source extraction). *Let $X = (X_1, \dots, X_t)$ be an $((n_1, \dots, n_t), (k_1, \dots, k_t))$ -block-source, and let $\text{Ext}_i: \{0, 1\}^{n_i} \times \{0, 1\}^{d_i} \rightarrow \{0, 1\}^{m_i}$ be average-case (k_i, ε_i) -seeded extractors running in time T_i with output length $m_i \geq d_{i-1}$ for $i \geq 2$, that output their seed. Then, there exists a strong $(k_1, \dots, k_t, \varepsilon = \sum_{i \in [t]} \varepsilon_i)$ -block-source extractor $\text{BExt}: \{0, 1\}^{n_1} \times \dots \times \{0, 1\}^{n_t} \times \{0, 1\}^{d_t} \rightarrow \{0, 1\}^m$ with output length $m = m_1 - d_t$ that runs in time $O(\sum_{i \in [t]} T_i)$. If X is an exact block source, then the Ext_i 's do not need to be average-case.*

We discuss how the fast hash-based extractor from Lemma 2.13 can be used to construct a fast extractor with seed length any constant factor smaller than its output length for high min-entropy sources. We need the following lemma, which is an easy consequence of the chain rule for min-entropy.

Lemma 2.23 ([GUV09, Corollary 4.16]). *Let X be an $(n, k = n - \Delta)$ -source and let X_1, \dots, X_t correspond to disjoint subsets of coordinates of X , with each X_i of length $n_i \geq n'$. Then, (X_1, \dots, X_t) is $t\varepsilon$ -close to an exact $((n_1, \dots, n_t), (k', \dots, k'))$ -block-source for $k' = n' - \Delta - \log(1/\varepsilon)$.¹³*

The following appears in [GUV09] without the time complexity bound. We appropriately instantiate their approach and analyze the time complexity below.

Lemma 2.24 (fast extractors with seed shorter than output [GUV09, Lemma 4.11]). *For every integer $t \geq 1$ there exists a constant $C > 0$ such that for any positive integer n and $\varepsilon > 2^{-\frac{n}{50t}}$ there exists a strong $(k = (1 - \frac{1}{20t})n, \varepsilon)$ -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with $m \geq k/2$ and $d \leq k/t + C \log(n/\varepsilon)$ computable in time $O(tn \log n)$.*

Proof. Let X be an $(n, k = (1 - \frac{1}{20t})n)$ -source and $\varepsilon' = \frac{\varepsilon}{2t}$. Let X_1, \dots, X_t correspond to disjoint subsets of coordinates of X , with $|X_i| = \lfloor n/t \rfloor = n'$ for all i . Then, Lemma 2.23 guarantees that (X_1, \dots, X_t) is $(t\varepsilon')$ -close to an exact $((n_1 = n', \dots, n_t = n'), k' = n' - \frac{n}{20t} - \log(1/\varepsilon'))$ -block-source X' . Note that

$$k' = n' - \frac{n}{20t} - \log(1/\varepsilon') \geq \frac{19n}{20t} - 1 - \log(1/\varepsilon') \geq 0.9n',$$

since we have assumed that $\varepsilon > 2^{-\frac{n}{50t}}$. Now, let $\text{Ext}': \{0, 1\}^{n'} \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ be the strong (k', ε') -seeded extractor from Lemma 2.13 with output length $m = \lceil \frac{k'}{2t} \rceil \leq k - 4 \log(16/\varepsilon')$ and corresponding seed length $d \leq k/t + 4 \log(n'/\varepsilon') + 9 \leq k/t + C \log(n/\varepsilon)$ for a large enough constant $C > 0$ depending only on t . Then, we apply block source extraction (Lemma 2.22) to X' with $\text{Ext}_1 = \dots = \text{Ext}_t = \text{Ext}'$ to get the desired strong $(k, 2t\varepsilon' = \varepsilon)$ -extractor Ext with output length $t \cdot m \geq k/2$ and seed length d . Since Ext' is computable in time $O(n \log n)$, then Ext is computable in time $O(tn \log n)$. \square

¹³[GUV09, Corollary 4.16] is originally stated only for the special case where X_1, \dots, X_t partition the coordinates of X . This extends easily to the statement presented here by noting that the puncturing of X to its $\tilde{n} = \sum_{i=1}^t n_i$ bits corresponding to X_1, \dots, X_t is an $(\tilde{n}, \tilde{n} - \Delta)$ -source, and applying the original statement in [GUV09].

In addition to [Lemma 2.21](#), one can potentially boost the output length of a high min-entropy extractor by first treating the source as a block source and then performing a simple block source extraction. The next corollary appears in [[Vad12](#), Lemma 6.27].

Lemma 2.25. *Let $\text{Ext}_{\text{in}}: \{0,1\}^{n/2} \times \{0,1\}^\ell \rightarrow \{0,1\}^d$ and $\text{Ext}_{\text{out}}: \{0,1\}^{n/2} \times \{0,1\}^d \rightarrow \{0,1\}^m$ be (k', ε) -extractors. Then, for any $(n, k = \delta n)$ -source (X_1, X_2) , each $X_i \sim \{0,1\}^{n/2}$, and an independent uniform $Y \sim \{0,1\}^\ell$, we have that*

$$\text{Ext}((X_1, X_2), Y) = \text{Ext}_{\text{out}}(X_1, \text{Ext}_{\text{in}}(X_2, Y))$$

is 4ε -close to uniform, assuming that $k' \leq (\delta - \frac{3}{4})n$ and $\varepsilon \geq 2^{-n/4}$. In other words, Ext is a $(k, 4\varepsilon)$ -extractor. Moreover, if Ext_{in} is strong then Ext is also strong, and if Ext_{in} and Ext_{out} run in time T_1 and T_2 , respectively, then Ext runs in time $O(T_1 + T_2)$.

3 Additional Building Blocks

3.1 Fast Generation of Small-Bias Sets

A distribution $S \sim \{0,1\}^n$ is ε -biased if it is indistinguishable from uniform by every linear test. Namely, if for every nonempty $T \subseteq [n]$ it holds that $\Pr_{s \sim S}[\bigoplus_{i \in T} s_i = 1] \in [\frac{1-\varepsilon}{2}, \frac{1+\varepsilon}{2}]$. We say that a set $S \subseteq \{0,1\}^n$ is ε -biased if the flat distribution over its support is ε -biased. We say that a linear code $\mathcal{C} \subseteq \{0,1\}^n$ is ε -balanced if the Hamming weight of each nonzero codeword lies in $[\frac{1-\varepsilon}{2}n, \frac{1+\varepsilon}{2}n]$. It is known that these two objects are essentially the same: S is ε -biased if and only if the $|S| \times n$ matrix whose rows are the elements of S is a generator matrix of an ε -balanced code.

One prominent way of constructing ε -balanced codes is via *distance amplification*, namely, starting with a code of some bias $\varepsilon_0 \gg \varepsilon$ and, using a parity sampler, amplify its bias. We will use a specific, simple, instantiation of a parity sampler – the random walk sampler.

Lemma 3.1 (RWs amplify bias [[Ta-17](#), Theorem 3.1]¹⁴). *Let $\mathcal{C}_0 \subseteq \{0,1\}^n$ be an ε_0 -balanced code, and let $G = (V = [n], E)$ be a D -regular λ -spectral expander, and for an even $t \in \mathbb{N}$, let $\mathcal{W}_t = \{w_1, \dots, w_{\bar{n}}\} \subseteq [n]^t$ be the set of walks of length t on G , noting that $\bar{n} = n \cdot D^t$. Define $\mathcal{C} \subseteq \{0,1\}^{\bar{n}}$ such that*

$$\mathcal{C} = \{\text{dsum}_{\mathcal{W}_t}(c_0) : c_0 \in \mathcal{C}_0\},$$

where $y = \text{dsum}_{\mathcal{W}_t}(x)$ at location $i \in [\bar{n}]$ is given by $\bigoplus_{j \in w_i} x_j$.

Then, \mathcal{C} is ε -balanced, for

$$\varepsilon = (\varepsilon_0 + 2\lambda)^{t/2}.$$

For \mathcal{C}_0 , we will use the Justesen code.

Lemma 3.2 ([[Jus72](#)]). *There exist constants $R \in (0,1)$ and $\varepsilon_0 \in (0,1)$ such that there exists an explicit family of codes $\{\text{Jus}_n\}$ parameterized by block length n , with rate R and ε_0 -balanced. Moreover, given $x \in \{0,1\}^{k=Rn}$, $\text{Jus}_n(x)$ can be computed in $\tilde{O}(n)$.*

Proof. The parameters of the codes follow from the original construction (and specifically, the lemma holds, say, with $R = \frac{1}{8}$ and $\varepsilon_0 = \frac{37}{40}$), so we will just show that the code is efficiently computable. Given a message x , we first encode it with a full Reed–Solomon code of constant relative distance over a field \mathbb{F}_q of characteristic 2, where $q \log q = O(n)$. By [Lemma 2.1](#), this can be done in

¹⁴The argument for $t = 2$ was suggested already by Rozenman and Wigderson (see [[Bog12](#)]). Note that the goal of the RW is to *reduce* the bias, $\varepsilon \ll \varepsilon_0$, but we choose to use “amplify” and follow the existing nomenclature.

time $\tilde{O}(q) = \tilde{O}(n)$. Then, we replace each Reed–Solomon symbol $p_x(\alpha)$, for $\alpha \in \mathbb{F}_q$, with the binary representation of $(p(\alpha), \alpha \cdot p(\alpha))$. (In other words, we concatenate Reed–Solomon with the Wozencraft ensemble.) This takes $\tilde{O}(q)$ time as well. \square

Corollary 3.3. *There exist a constant $c > 1$, and an explicit family of balanced codes, such that for every $\bar{n} \in \mathbb{N}$ and any $\varepsilon > 0$, $\mathcal{C} \subseteq \mathbb{F}_2^{\bar{n}}$ is ε -balanced of rate $R = \varepsilon^c$, and given $x \in \mathbb{F}_2^{k=R\bar{n}}$, any m bits of $\mathcal{C}(x)$ can be computed in time $\tilde{O}(k) + m \log(1/\varepsilon) \cdot \text{polylog}(k)$.*

Moreover, for every $k \in \mathbb{N}$ and any $\varepsilon > 0$ there exists an explicit ε -biased set over \mathbb{F}_2^k generated by a function $\text{SmallBias}: [\bar{n}] \rightarrow \{0, 1\}^k$ computable in time $(k + \log(1/\varepsilon)) \cdot \text{polylog}(k)$.

Proof. Let $\mathcal{C}_0: \mathbb{F}_2^{k=\Theta(n)} \rightarrow \mathbb{F}_2^n$ be the ε_0 -balanced code guaranteed to us by [Lemma 3.2](#), and let $G = (V = [n], E)$ be the D -regular λ -spectral expander of [Corollary 2.17](#), instantiated with $\lambda = \frac{1-\varepsilon_0}{4}$ (so $D = D(\varepsilon_0)$). Letting $\mathcal{C}: \mathbb{F}_2^k \rightarrow \mathbb{F}_2^{\bar{n}}$ be the amplified code of [Lemma 3.1](#) set with

$$t = \frac{2 \log(1/\varepsilon)}{\log\left(\frac{2}{1+\varepsilon_0}\right)} = O(\log(1/\varepsilon)),$$

the lemma tells us that it is $(\varepsilon_0 + 2\lambda)^{t/2} \leq \varepsilon$ balanced. Given $x \in \mathbb{F}_2^n$ and $i \in [\bar{n}]$, computing $\mathcal{C}(x)_i$ amounts to XORing t coordinates of $\mathcal{C}_0(x)$ determined by $i = (v, i_1, \dots, i_t)$, which indexes a random walk over G . Computing $\mathcal{C}_0(x)$ takes $\tilde{O}(n)$ time, and computing a length- t walk over G takes $t \cdot \log(1/\lambda) \cdot \text{polylog}(n)$ time. The corollary then follows, observing that $\bar{n} = n \cdot D^t = n \cdot \text{poly}(1/\varepsilon)$.

For the “Moreover” part, recall that we can take the rows of the generator matrix of \mathcal{C} as our ε -biased set S . Thus, for any $i \in [\bar{n}]$, we can compute $\text{SmallBias}(i)$ as follows: Compute the corresponding random walk on G , and then, for any $j \in [k]$, $\text{SmallBias}(i)_j$ is obtained by XORing the bits of $\mathcal{C}_0(e_j)$ indexed by the i -th random walk. Observing that each bit of $\mathcal{C}_0(e_j)$ can be computed in time $\tilde{O}(\log n)$,¹⁵ the runtime of SmallBias is

$$t \cdot \log(1/\lambda) \cdot \text{polylog}(n) + k \cdot \tilde{O}(\log n) = (k + \log(1/\varepsilon)) \cdot \text{polylog}(k),$$

recalling that $k = \Theta(n)$. \square

Remark 3.4. Instead of using Justesen’s code from [Lemma 3.2](#) as our inner code \mathcal{C}_0 , we can instead use the linear-time encodable code of Spielman [\[Spi96\]](#). While not stated as *balanced* codes, but rather as constant relative distance codes, one can verify that the distance can also be bounded by above. The construction is more involved than Justesen’s one. However, in the logarithmic-cost RAM model, in which basic register operations over $O(\log n)$ bit registers count as a single time step, Spielman’s code can be implemented in $O(n)$ time.

3.2 A Sampler from Bounded Independence

Recall that $X_1, \dots, X_n \sim \Sigma^n$ is a b -wise independent distribution, if for every distinct $i_1, \dots, i_b \in [n]$ it holds that $(X_{i_1}, \dots, X_{i_b}) = U_{\Sigma^b}$, the uniform distribution over Σ^b .

Lemma 3.5. *For any prime power n , any $m \leq n$, and any $b \leq m$, there exists an explicit b -wise independent generator $\text{Bl}_b: \{0, 1\}^d \rightarrow [n]^m$ with $d = O(b \log n)$ ¹⁶ that satisfies the following.*

1. *Given $z \in \{0, 1\}^d$, $\text{Bl}_b(z)$ is computable in time $\tilde{O}(n)$.*

¹⁵Indeed, each coordinate of $\mathcal{C}_0(e_j)$ is a bit in the encoding of (α^j, α^{j+1}) for some $\alpha \in \mathbb{F}_q$, where $q \log q = O(n)$.

¹⁶That is, the distribution formed by picking $z \sim U_d$ and outputting $\text{Bl}_b(z)$ is b -wise independent over $[n]^m$.

2. For any $\theta > 0$ the following holds. With probability at least $1 - 2^{-\Omega(\theta b)}$ over $z \sim U_d$, $\text{Bl}_b(z)$ has at least $m - (1 + \theta)\frac{m^2}{n}$ distinct elements.

Proof. We use the standard polynomial-based construction of bounded independence sample spaces. Concretely, given $z \in \mathbb{F}_n^b$, we interpret it as a polynomial $f_z(x) = \sum_{i=1}^b z_i x^{i-1}$, and we let $\text{Bl}_b(z)$ output $(f_z(\alpha_1), \dots, f_z(\alpha_m))$, where the α_i -s are distinct elements in \mathbb{F}_n . This gives us a b -wise generator over $[n]^m$ (the correctness can be found, e.g., in [Vad12, Chapter 3]). By [Item 1 of Lemma 2.1](#), $\text{Bl}_b(z)$ is computable in time $\tilde{O}(n)$ (and this is true for any $b \leq n$). The seed length d is given by $O(b \log n)$.

Next, we argue that most samples contain mostly distinct elements. Towards this end, let X_1, \dots, X_m be our b -wise independent distribution $\text{Bl}_b(U_d)$, and let Z_i denote the indicator random variable that is 1 if and only if X_i is a duplicate element (namely, there exists $j < i$ such that $X_i = X_j$). We are looking to bound $\sum_{i \in [m]} Z_i$ with high probability. This will follow from the fact that X is b -wise independent.

Claim 3.6. Assume that $t \leq b/2$. Then, for any distinct $i_1, \dots, i_t \in [m]$, it holds that $\Pr[Z_{i_1} = \dots = Z_{i_t} = 1] \leq (m/n)^t$.

Proof. Fix indices j_1, \dots, j_t , where each $j_\ell < i_\ell$. The probability that $X_{i_\ell} = X_{j_\ell}$ for all $\ell \in [t]$ is at most n^{-t} , since this event depends on at most $2t \leq b$ random variables. Union-bounding over all choices of j -s incurs a multiplicative factor of $\prod_{\ell \in [t]} (i_\ell - 1) \leq m^t$, so overall, $\Pr[Z_{i_1} = \dots = Z_{i_t} = 1] \leq (m/n)^t$. \square

Now, [Claim 3.6](#) is sufficient to give us good tail bounds (see, e.g., [HH15, Section 3]¹⁷). In particular, denoting $\mu = \frac{m}{n}$ there exists a universal constant $c > 0$ such that

$$\Pr \left[\sum_{i \in [m]} Z_i \geq (1 + \theta)\mu m \right] \leq 2^{-c\theta b},$$

which implies [Item 2](#). \square

Towards introducing our sampler, we will need the following tail bound for b -wise independent random variables.

Lemma 3.7 (e.g., [Vad12, Chapter 3]). Let $X \sim \Sigma^m$ be a b -wise independent distribution, and fix some $\varepsilon > 0$. Then, X is also a (δ, ε) sampling distribution, where $\delta = \left(\frac{b}{2\varepsilon\sqrt{m}}\right)^b$.

While the error in [Item 2](#) above is small, it is not small enough for us to simply combine [Lemmas 3.5](#) and [3.7](#), and we will need to do a mild error reduction. We do this via random walks on expanders and discarding repeating symbols, as was also done in [Vad04, Section 8]. This gives us the following bounded-independence based sampler.

Lemma 3.8. For any positive integers $m \leq n$, any $\delta_\Gamma \in (0, 1)$, and any constant $\eta \in (0, 1)$ such that $m \leq \frac{\eta}{8}n$, there exists an explicit $(\delta_\Gamma, \varepsilon_\Gamma = 2\eta)$ -averaging sampler $\Gamma: \{0, 1\}^d \rightarrow [n]^m$ with $d = O\left(\frac{\log n}{\log m} \cdot \log \frac{1}{\delta_\Gamma}\right)$, that satisfies the following additional properties.

1. Every output of Γ contains distinct symbols of $[n]$, and

¹⁷In the notation of [HH15], the distribution Z is $(0, b)$ -growth-bounded. The tail bound then follows from [HH15, Theorem 3.2].

2. Given $y \in \{0, 1\}^d$, $\Gamma(y)$ is computable in time $\tilde{O}(n) + \text{poly}\left(\log \frac{1}{\delta_\Gamma} \cdot \frac{\log n}{\log m}\right)$.

Proof. Set b to be the smallest integer such that $b \log \frac{2\eta\sqrt{m'}}{b} \geq \log \frac{4}{\delta_\Gamma}$ and set $m' = (1+\eta)m$, $\theta = \eta/2$. Notice that $b = O\left(\frac{\log(1/\delta_\Gamma)}{\log m}\right)$. Assume first that n is a prime power, and let

$$\text{Bl}_b: \{0, 1\}^{d'} \rightarrow [n]^{m'}$$

be the b -wise independent generator guaranteed to us by [Lemma 3.5](#), with $d' = O(b \log n)$. By [Lemma 3.7](#), $X = \text{Bl}_b(U_{d'})$ is a (δ_b, η) sampling distribution, where

$$\delta_b = \left(\frac{b}{2\eta\sqrt{m'}}\right)^b \leq \frac{\delta_\Gamma}{4}.$$

Also, we know from [Lemma 3.5](#) that with probability at least $1 - 2^{-\Omega(\theta b)} \triangleq 1 - p$, each sample from X has at least $m' - (1 + \theta)\frac{m'^2}{n} \geq m$ distinct symbols, using the fact that $\frac{n}{m} \geq \frac{(1+\theta)(1+\eta)^2}{\eta}$. Conditioned on seeing at least m distinct symbols, X as a sampling distribution, when we remove the non-distinct elements, has confidence $\frac{\delta_\Gamma/4}{1-p} \leq \frac{\delta_\Gamma}{2}$ and accuracy 2η (where the second η comes from the fact that ηm symbols were removed).

Next, in order to improve the probability of sampling a good sequence to match the confidence, let $G = (V = \{0, 1\}^{d'}, E)$ be the D -regular λ -spectral expander of [Corollary 2.17](#), instantiated with $\lambda = p$, so $D \leq p^{-c}$ for some universal constant c . Write $d = d' + \ell'$ for $\ell' = \ell \cdot \log D$, where $\ell = c_\ell \cdot \frac{\log(1/\delta_\Gamma)}{b}$ for some constant c_ℓ soon to be determined. Given $y = (z, w) \in \{0, 1\}^{d'} \times [D]^\ell$, let $z = v_0, v_1, \dots, v_\ell$ denote the corresponding random walk over G . Our sampler Γ , on input y , computes $\text{Bl}_b(v_i)$ for every $i \in [\ell]$ and outputs the first m distinct symbols of the first sequence with at least m distinct symbols. If no such sequence was found, Γ simply outputs $(1, \dots, m)$ (in which case we say it *failed*). By the expander hitting property (see, e.g., [\[Vad12, Chapter 4\]](#)), Γ fails with probability at most

$$(p + \lambda)^\ell = (2p)^\ell \leq \frac{\delta_\Gamma}{2}$$

over $y \sim U_d$, upon choosing the appropriate constant $c_\ell = c_\ell(\eta)$. We then have that $\Gamma(U_d)$ is indeed a $(\delta_\Gamma, 2\eta)$ sampling distribution, that can be generated using a seed of length $d' + \ell' = O(\log(1/\gamma))$. In terms of runtime, computing v_1, \dots, v_ℓ can be done in time

$$\ell \cdot \log \frac{1}{p} \cdot \text{polylog}(d') = \text{poly}\left(\log \frac{1}{\delta_\Gamma} \cdot \frac{\log n}{\log m}\right),$$

and computing the sequences themselves takes $\ell \cdot \tilde{O}(n)$ time, and observe that $\ell = O(\log m)$.

Finally, we need to argue that we can also handle the case where n is not a prime power. One option to handle arbitrary n -s is to resort to almost b -wise independence. Specifically, we can start with a γ -biased distribution, with a small enough γ , over $n_b = \lceil \tau^{-1} \log n \rceil n$ bits, and map each consecutive $\lceil \tau^{-1} \log n \rceil$ bits to $[n]$, for a small enough τ , by simply taking the corresponding integer modulo n . We skip the details (to see the corresponding sampling property, see [\[XZ25\]](#)), and note that the result that uses the bounded-independence sampler, [Theorem 4.1](#), only needs to invoke the sampler with n being a prime power.¹⁸ \square

¹⁸In some more detail, in the proof of [Lemma 4.2](#) we use the b -wise distribution over $[n]$ in order to sub-sample from $X \sim \{0, 1\}^n$. In the context of [Theorem 4.1](#), we can assume that X has entropy rate $1 - \alpha$ for an arbitrarily small constant $\alpha > 0$, and the goal is to retain its high entropy rate when sub-sampling. Letting p be the largest prime smaller than or equal to n (which satisfies $p \geq n/2$ by Bertrand's postulate), we have that $X' = X_{[1,p]}$ has entropy rate at least $1 - 2\alpha$, and one can verify that sampling from X' works equally well, up to negligible loss in parameters. Moreover, we can find p in time $\tilde{O}(n)$ using, e.g., the deterministic AKS primality test [\[AKS04\]](#) that runs in time $\text{polylog } n$ applied to each integer in $[n/2, n]$.

We will need to somewhat extend [Lemma 3.8](#) and use the simple, yet crucial, property of our bounded independence sampling: A subset of the coordinates of a b -wise independent distribution with distinct samples is itself a b -wise independent distribution with distinct samples.¹⁹ Thus, if we wish to sample multiple times, say using $m_1 \leq \dots \leq m_t < n$ samples, we can use *one sample* from a sampler that outputs m_t coordinates, and truncate accordingly to create the other samples.

Lemma 3.9. *For any positive integers n and $m_1 < \dots < m_t \leq n$, any $\delta \in (0, 1)$ and any constant ε such that $m_t \leq \frac{\varepsilon}{16}n$, there exists an explicit function $\Gamma: \{0, 1\}^d \rightarrow [n]^{m_t}$ with $d = O\left(\frac{\log n}{\log m_1} \cdot \log \frac{1}{\delta}\right)$ that satisfies the following.*

1. *For any $i \in [t]$, the function Γ_i , that on input $y \in \{0, 1\}^d$ outputs $\Gamma(y)|_{[1, m_i]}$, is a (δ, ε) -averaging sampler, and each sample contains distinct symbols.*
2. *On input $y \in \{0, 1\}^d$, $\Gamma(y)$ can be computed in time $\tilde{O}(n) + \text{poly}(\log(1/\delta), (\log n)/(\log m_1))$.*

Proof Sketch. Let $\Gamma: \{0, 1\}^d \rightarrow [n]^{m_t}$ be the (δ, ε) -averaging sampler of [Lemma 3.8](#), set with $\delta_\Gamma = \delta$ and $\eta = \varepsilon/2$. Fix some $i \in [t]$, and let Γ_i be as described in the lemma's statement. Inspecting the proof of [Lemma 3.8](#), and using the same notation, we see that the output of Γ_i can be obtained by truncating the output of Bl_b to the first $m'_i = (1 + \eta)m_i$ bits and using the same expander random walk in order to get m_i distinct symbols with high probability. Note that:

1. (sampling) $X_{[1, m'_i]}$ is a b -wise independent distribution, whose sampling properties are determined by m'_i . We then need to make sure that the smallest m'_i is large enough, by setting $b = O\left(\frac{\log(1/\delta_\Gamma)}{\log m_1}\right)$, as in the proof of [Lemma 3.8](#).
2. (distinctness) In order for the probability that $X_{[1, m'_i]}$ has at least m_i distinct symbols to be large enough (recall that X itself is over m'_t symbols), $m_i \leq m_t$ needs to be small enough compared to n . And indeed, we take $m_i \leq \frac{\eta}{8}n = \frac{\varepsilon}{16}n$.

Once the two properties above are guaranteed, we can amplify the probability to sample a string with sufficiently many distinct symbols via expander random walks, exactly as in the proof of [Lemma 3.8](#). \square

3.3 Nearly-Linear Time Condensers

In this section we argue that modern constructions of condensers with nearly-optimal parameters [[GUV09](#), [KT22](#)] are computable in nearly-linear time, possibly after a reasonable one-time preprocessing step. We will repeatedly use the fact that these condensers allow us to transform, in time $\tilde{O}(n)$ and using nearly-optimal seed length $O(\log(n/\varepsilon))$, an arbitrary input (n, k) -source X into an output ε -close to a source X' of length $m \approx k$ and min-entropy rate $1 - \alpha$, for any constant $\alpha \in (0, 1)$ of our choice. We provide formal statements below, without being explicit about the precise relationship between the parameter α and the various constants in the seed length and entropy requirement for the sake of readability. More precise control over the constants in these condensers can be found, for example, in [[KT22](#), Theorems 1 and 2].

¹⁹We note that the distinct-samples sampler given in [[Vad04](#)], an instantiation of which we use in [Lemma 2.20](#), does not seem to enjoy a similar property. The advantage of [Lemma 2.20](#) over [Lemma 3.9](#) that will appear soon, is that it has better seed length.

3.3.1 The Lossless KT Condenser

We first give the lossless KT condenser based on multiplicity codes over \mathbb{F}_q , due to Kalev and Ta-Shma [KT22]. This condenser works when the min-entropy requirement $k = \Omega(\log^2(n/\varepsilon))$. Then, we discuss how it can be converted to a condenser over bits with only a negligible loss in parameters.

Theorem 3.10 (the lossless KT condenser over \mathbb{F}_q [KT22, adapted]). *For any constant $\alpha \in (0, 1)$ there exist constants $C_\alpha, C'_\alpha > 0$ such that the following holds for every $n \in \mathbb{N}$, $\varepsilon > 0$, $k \geq C_\alpha \log^2(n/\varepsilon)$, and a prime power $q = p^r$ with $p \geq n$ satisfying $\frac{C'_\alpha}{2} \log(n/\varepsilon) \leq \log q \leq C'_\alpha \log(n/\varepsilon)$. There exists a strong $(k, k' = k + \log q, \varepsilon)$ -condenser*

$$\text{KTCond}' : \mathbb{F}_q^{n'} \times \mathbb{F}_q \rightarrow \mathbb{F}_q^{m'}$$

where $n' = \left\lceil \frac{n}{\log q} \right\rceil$ and $m' \leq (1 + \alpha) \frac{k}{\log q}$. Moreover, given p , an irreducible polynomial over \mathbb{F}_p of degree r , $x \in \mathbb{F}_q^n$, and $y \in \mathbb{F}_q$, the output $\text{KTCond}'(x, y)$ can be computed in $\tilde{O}(n' \log q) = \tilde{O}(n)$ time.

Proof. We only need to establish the construction's runtime. Given $x \in \mathbb{F}_q^{n'}$ and $y \in \mathbb{F}_q$, interpret x as a polynomial $f \in \mathbb{F}_q[X]$ of degree at most $n' - 1$. The output $\text{KTCond}(x, y)$ is the sequence of derivatives

$$(f(y), f'(y), \dots, f^{(m'-1)}(y)).$$

By Lemma 2.1, computing the derivatives takes $\tilde{O}(n') \cdot \log q = \tilde{O}(n' \log q) = \tilde{O}(n)$ time since we are also given p and an irreducible polynomial over \mathbb{F}_p of degree r . The rest of the auxiliary operations are negligible compared to computing the derivatives. \square

Remark 3.11 (the KT condenser preprocessing step). The KT condenser must be instantiated with an appropriate field size $q = \text{poly}(n/\varepsilon)$ and requires performing operations over \mathbb{F}_q . Hence, as stated in Theorem 3.10, we need to know the characteristic p of \mathbb{F}_q and an irreducible polynomial over \mathbb{F}_p of the appropriate degree. Since these objects only need to be generated once for a given set of extractor parameters (n, k, ε, m) , we view this as a one-time preprocessing step. Our choice of q influences the complexity of constructing a model of \mathbb{F}_q . For example, if we aim for prime field size, then we need to generate a prime $q = \text{poly}(n/\varepsilon)$. We know how to do that in randomized time $\text{polylog}(n/\varepsilon)$ (e.g., see [Sho05, Section 9.4]). We believe that this is already reasonable when seen as a one-time preprocessing step, because generating large primes is a well-studied problem and practical randomized algorithms exist.

Nevertheless, we can do even better and avoid the preprocessing step if ε is not very small by exploiting the fact that the KT condenser works with any prime power $q = p^r = \text{poly}(n/\varepsilon)$ for any prime $p \geq n$ [KT22, Theorem 3]. First, a prime $p \in [n, 2n]$ is guaranteed to exist by Bertrand's postulate, and we can find it deterministically in time $\tilde{O}(n)$.²⁰ Second, a deterministic algorithm of Shoup [Sho90, Theorem 3.2] finds an irreducible polynomial of degree r over \mathbb{F}_p in time $\tilde{O}(\sqrt{p} \cdot r^{4+\delta})$ for any constant $\delta > 0$. Since $p \leq 2n$ and $r = \log_p(\text{poly}(n/\varepsilon)) = O(\log(n/\varepsilon))$, we conclude that Shoup's algorithm runs in time $\tilde{O}(n)$ provided that $\varepsilon \geq 2^{-Cn^\gamma}$ for any constant $\gamma < 1/8$ and a sufficiently large constant $C > 0$. In particular, it suffices to have, say, $\varepsilon \geq 2^{-Cn^{0.1}}$.

In sum, the preprocessing step for the KT condenser requires either (1) randomized $\text{polylog}(n/\varepsilon)$ time, or (2) deterministic $\tilde{O}(n + \sqrt{n} \cdot \log^{4+\delta}(1/\varepsilon))$ time. In particular, this leads to the distinction between Items 1 and 2 in Theorem 1.

²⁰The procedure that sequentially tests the primality of all integers in $[n, 2n]$ is already sufficiently quick. Each primality test takes time $\text{polylog}(n)$ using, for example, the AKS primality test [AKS04], and so the overall complexity is $\tilde{O}(n)$.

The condenser from [Theorem 3.10](#) receives and outputs vectors over \mathbb{F}_q . We would like to have a version of this condenser that works with vectors over bits. Towards that end, for completeness, we formally state and prove a (standard) transformation.

Lemma 3.12. *Suppose that $\text{Cond}_0: \mathbb{F}_q^{n'} \times \mathbb{F}_q \rightarrow \mathbb{F}_q^{m'}$ is a strong (k, k', ε) -condenser. Then, there exists $\text{Cond}: \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^m$ with $n = n' \cdot \lceil \log q \rceil$, $\ell = \lfloor \log q \rfloor$, and $m = m' \cdot \lceil \log q \rceil$ that is a strong $(k + \ell, k', 3\sqrt{\varepsilon})$ -condenser. Furthermore, if Cond_0 is computable in time T then Cond is computable in time $T + \tilde{O}(n)$.*

Proof. The condenser Cond works as follows. Given an input $x \in \{0, 1\}^n$, we interpret it as an element of $\mathbb{F}_q^{n'}$ by mapping each consecutive block of $\lfloor \log q \rfloor$ bits to an \mathbb{F}_q -element and discarding up to $\lceil \frac{n}{\lfloor \log q \rfloor} \rceil \cdot \lfloor \log q \rfloor - n \leq \lfloor \log q \rfloor$ bits at the end of x . Denote this mapping by $\psi: \{0, 1\}^n \rightarrow \mathbb{F}_q^{n'}$. Given a seed $y \in \{0, 1\}^\ell$, we interpret it as an \mathbb{F}_q -element, since $\ell = \lfloor \log q \rfloor$. Denoting this mapping by $\phi: \mathbb{F}_2^\ell \rightarrow \mathbb{F}_q$, we have that $\mathbf{H}_\infty(\phi(U_\ell)) \geq \ell$.

Equipped with an $\mathbb{F}_q^{n'}$ -element x , and an \mathbb{F}_q -element y , we are ready to compute $\text{Cond}_0(x, y) \in \mathbb{F}_q^{m'}$. Mapping the output into bits is done similarly, by writing the binary encoding of each \mathbb{F}_q -element using $\lceil \log q \rceil$ bits. There are a few possible losses along the way:

1. We get an (n, k) source X , but feed the condenser a source $X' \sim \mathbb{F}_q^{n'}$ with entropy at least $k - \lfloor \log q \rfloor = k - \ell$.
2. We do not use a uniform seed, but rather a seed that lacks at most 1 bit of entropy.
3. Mapping $\text{Cond}_0(\psi(x), \phi(y))$ to the binary $\text{Cond}(x, y)$ may increase the output by m' non-entropic bits. This simply increases the output length slightly, but not the output entropy.

To handle (1), we simply increase the entropy of the source. To handle (2), we prove that strong condensers work with entropy-deficient random seeds, as long as the error is good enough.

Claim 3.13. *Let $\text{Cond}: \mathbb{F}_q^n \times \mathbb{F}_q \rightarrow \mathbb{F}_q^m$ be a strong (k, k', ε) condenser, and let Y be a random variable with min-entropy $\log(q) - g$. Then, $(Y, \text{Cond}(X, Y))$ is ε' -close to a random variable with min-entropy k' , where $\varepsilon' = (2^g + 1)\sqrt{\varepsilon}$.*

Proof. Let X be a random variable over \mathbb{F}_q^n with min-entropy k . By an averaging argument, we know that there exists a set $\mathbf{B} \subseteq \mathbb{F}_q$ of size $|\mathbf{B}| \leq \sqrt{\varepsilon} \cdot q$ such that for any $y \notin \mathbf{B}$ it holds that $\text{Cond}(X, y)$ is $\sqrt{\varepsilon}$ -close to a random variable with min-entropy $k' - \log q$. But now,

$$\Pr[Y' \in \mathbf{B}] = \sum_{y \in \mathbf{B}} \Pr[Y = y] \leq |\mathbf{B}| \cdot 2^{-(\log(q) - g)} \leq \sqrt{\varepsilon} \cdot 2^g. \quad \square$$

In our case we invoke [Claim 3.13](#) with $g = 1$, so we can simply set the new error parameter ε' to be $\varepsilon' = \varepsilon^2/3$, where ε is the designated distance to high min-entropy. A bit more formally, guaranteeing that $(U_d, \text{Cond}(X, U_d))$ is ε' -close to entropy k' implies that $(\phi(U_d), \text{Cond}(X, \phi(U_d)))$ is ε -close to entropy k' as well. Finally, for the running time claim note that computing the mappings ψ and ϕ and converting the output in $\mathbb{F}_q^{m'}$ to $\{0, 1\}^m$ can be done in time $\tilde{O}(n)$. \square

Combining [Theorem 3.10](#) and [Lemma 3.12](#) yields the following.

Theorem 3.14 (the KT condenser over bits [[KT22](#)]). *For any constant $\alpha \in (0, 1)$ there exist constants $C_\alpha, C'_\alpha > 0$ such that the following holds for every $n \in \mathbb{N}$, $\varepsilon > 0$, and $k \geq C_\alpha \log^2(n/\varepsilon)$. There exists a strong (k, k', ε) -condenser*

$$\text{KTCond}: \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^m$$

where $\ell \leq C'_\alpha \log(n/\varepsilon)$, $m \leq (1 + \alpha)k$, and $k' = k \geq \ell + (1 - \alpha)m$. Moreover,

- Given $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^\ell$, a prime power $q = p^r$ in $[2^\ell, 2n \cdot 2^\ell]$ with $p \in [n, 2n]$ and an irreducible polynomial over \mathbb{F}_p of degree r , the output $\text{KTCond}(x, y)$ can be computed in $\tilde{O}(n)$ time.
- Given only $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^\ell$, $\text{KTCond}(x, y)$ can be computed in $\tilde{O}(n + \sqrt{n} \cdot \log(1/\varepsilon)^5)$ time. Note that this is $\tilde{O}(n)$ when $\varepsilon \geq 2^{-Cn^{0.1}}$.

In particular, if $\varepsilon' = \sqrt{\varepsilon}$ and C_α is chosen large enough compared to C'_α , then for all (n, k) -sources X and a $(1 - \varepsilon')$ -fraction of seeds y it holds that $\text{KTCond}(X, y) \approx_{\varepsilon'} Z_y$, where Z_y is an $(m, k' - \ell \geq (1 - \alpha)m)$ -source.

Proof. Fix a target $\alpha \in (0, 1)$, and let $\beta \in (0, 1)$ be a constant to be set appropriately small depending on α . We invoke [Theorem 3.10](#) with β in place of α and q a prime such that $\frac{C'_\beta}{2} \log(n/\varepsilon) \leq \log q \leq C'_\beta \log(n/\varepsilon)$ to get $\text{KTCond}' : \mathbb{F}_q^{n'} \times \mathbb{F}_q \rightarrow \mathbb{F}_q^{m'}$, a strong $(k - \log q, k' = k, \varepsilon^2/3)$ -condenser with $n' = \left\lceil \frac{n}{\log q} \right\rceil$ and $m' \leq (1 + \beta) \frac{k}{\log q}$. Then, we apply [Lemma 3.12](#) to KTCond' . This yields our $\text{KTCond} : \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^m$, a strong $(k, k' = k, \varepsilon)$ -condenser with $\ell = \lfloor \log q \rfloor = \Theta_\beta(\log(n/\varepsilon))$ and

$$m = m' \cdot \lceil \log q \rceil \leq (1 + \beta)k + m' \leq (1 + \beta)k + (1 + \beta)k/\ell \leq (1 + 2\beta)k \leq (1 + \alpha)k,$$

provided that $\beta \leq \alpha/2$. Furthermore, if C_β is set sufficiently larger than C'_β , then $k' - \ell = k - \ell \geq (1 - \beta)k$, and so

$$\frac{k' - \ell}{m} \geq \frac{(1 - \beta)k}{(1 + 2\beta)k} \geq 1 - \alpha,$$

provided that $\beta > 0$ is sufficiently smaller than α . The first part of the theorem statement follows if we set the new C_α and C'_α to be C_β and C'_β , respectively. For the running time, note that KTCond' is computable in time $\tilde{O}(n)$ given p and an irreducible polynomial over \mathbb{F}_p of degree r , and the transformation in [Lemma 3.12](#) still runs in time $\tilde{O}(n)$. The remaining claim about the running time given only x and y was discussed in [Remark 3.11](#).

To see the “In particular” part of the theorem statement, fix an (n, k) -source X and note that $(Y, \text{KTCond}(X, Y)) \approx_\varepsilon (Y, Z)$ for some Z such that $\mathbf{H}_\infty(Y, Z) \geq k'$. Let $Z_y = (Z|Y = y)$. Then, an averaging argument gives that for a $(1 - \sqrt{\varepsilon})$ -fraction of seeds y we have $\text{KTCond}(X, y) \approx_{\sqrt{\varepsilon}} Z_y$. Since Y is uniformly random over $\{0, 1\}^\ell$, we get that $\mathbf{H}_\infty(Z_y) \geq k' - \ell$, as desired. \square

3.3.2 The Lossy RS Condenser

The downside of [Theorem 3.14](#) is that it requires the entropy in the source to be $\Omega(\log^2(n/\varepsilon))$, instead of the optimal $\Omega(\log(n/\varepsilon))$. Instead, we can use a lossy condenser²¹ based on Reed–Solomon codes. Unfortunately, this comes at the expense of computing a primitive element of a field of size $\text{poly}(n/\varepsilon)$, which we do not know how to do in nearly-linear time in n for arbitrary ε -s. As in [Section 3.3.1](#), we consider it a one-time preprocessing step, as it does not depend on the inputs to the condenser. Luckily, we have freedom in choosing the field size, as long as it is large enough. Therefore, if we choose the field size to be prime (as opposed to a power of 2 as in [\[GUV09\]](#)), then we can implement this one-time preprocessing step in time $\text{polylog}(n/\varepsilon)$ using randomness, as discussed in [Remark 3.16](#).

²¹Our extractor will lose a small constant fraction of the entropy, so losing a small constant fraction of the entropy in the condensing step will not make much difference.

We first state the lossy condenser over a prime field \mathbb{F}_q and argue about its running time. This condenser can be converted to a condenser over bits with only a negligible loss in parameters as in [Section 3.3.1](#).

Theorem 3.15 (the lossy RS condenser over \mathbb{F}_q [[GUV09](#), adapted]). *For any constant $\alpha \in (0, 1)$ there exist constants $C_\alpha, C'_\alpha > 0$ such that the following holds for every $n \in \mathbb{N}$, $\varepsilon > 0$, $k \geq C_\alpha \log(n/\varepsilon)$, and q a prime satisfying $\frac{C'_\alpha}{2} \log(n/\varepsilon) \leq \log q \leq C'_\alpha \log(n/\varepsilon)$. There exists a strong (k, k', ε) -condenser*

$$\text{RSCond}' : \mathbb{F}_q^{n'} \times \mathbb{F}_q \rightarrow \mathbb{F}_q^{m'}$$

where $n' = \left\lceil \frac{n}{\log q} \right\rceil$, $m' = \lceil k/\log q \rceil$, and $k' \geq (1 - \alpha)k$. Moreover, given $x \in \mathbb{F}_q^{n'}$, $y \in \mathbb{F}_q$, and a primitive element ζ for \mathbb{F}_q , the output $\text{RSCond}'(x, y)$ can be computed in time $\tilde{O}(n)$.

Proof. Given $x \in \mathbb{F}_q^{n'}$ and $y \in \mathbb{F}_q$, similarly to [Theorem 3.14](#), interpret x as a univariate polynomial $f \in \mathbb{F}_q[X]$ of degree at most $n' - 1$. Let ζ be the primitive element of \mathbb{F}_q given to us. The output $\text{RSCond}'(x, y)$ is the sequence of evaluations

$$(f(y), f(\zeta y), \dots, f(\zeta^{m'-1}y)) \in \mathbb{F}_q^{m'}.$$

The correctness proof, as well as the exact choice of parameters, are given in [[GUV09](#), Section 6.1]. We focus on bounding the runtime. Computing the evaluation points $y, \zeta y, \dots, \zeta^{m'-1}y$ can be done naively in time $m' \cdot M_q^b(1) = \tilde{O}(n' \log q) = \tilde{O}(n)$. Then, using [Lemma 2.1](#), the evaluation can be done in time $\tilde{O}(n') \cdot \log q = \tilde{O}(n' \log q) = \tilde{O}(n)$ as well. \square

Remark 3.16 (complexity of the RS condenser preprocessing). We now discuss the complexity of the preprocessing step required by the RS condenser, which corresponds to finding a primitive element of a field \mathbb{F}_q for a prime $q \leq \text{poly}(n/\varepsilon)$. We do not know any algorithms for finding primitive elements of \mathbb{F}_q running in time $\text{polylog}(q)$, unless we have access to the prime factorization of $q-1$.²² However, the RS condenser from [[GUV09](#)] can be instantiated with *any* field \mathbb{F}_q of appropriately large order $q = \text{poly}(n/\varepsilon)$. We can leverage this to speed up the preprocessing considerably by moving away from the fields of characteristic 2 used in [[GUV09](#)]. More precisely, we can focus on prime q and follow an approach used in cryptography for generating public parameters (prime/primitive element pairs (q, g)) for cryptographic schemes based on discrete logarithms over \mathbb{Z}_q^* , outlined in Shoup's excellent book [[Sho05](#), Section 11.1]. This leads to a randomized algorithm running in time $\text{polylog}(n/\varepsilon)$ and succeeding with high probability, that only needs to be executed once for a given set of extractor parameters (n, m, k, ε) . More precisely, we can first efficiently sample the prime factorization of a uniformly random number r in the set $\{1, \dots, L\}$ for an appropriate upper bound $L = \text{poly}(n/\varepsilon)$, following approaches of Bach [[Bac88](#)] or Kalai [[Kal03](#)] (as discussed in [[Sho05](#), Section 9.6]). Then, we check whether $r \geq L/2$ and $q = r + 1$ is prime, and repeat if not. Since roughly a $\frac{1}{\text{polylog}(n/\varepsilon)}$ fraction of such q 's is prime, we will succeed with high probability after $\text{polylog}(n/\varepsilon)$ trials. After the process above we hold a suitably large prime q and the prime factorization of $q - 1$, which allows us to find a primitive element of \mathbb{F}_q in randomized time $\text{polylog}(q) = \text{polylog}(n/\varepsilon)$.

Finally, combining [Theorem 3.15](#) and [Lemma 3.12](#) allows us to get a version of the “prime q ” RS condenser over bits.

²²If we have access to the prime factorization of $q - 1$, then we can find a primitive element of \mathbb{F}_q in time $\text{polylog}(q)$ using randomness by repeatedly sampling a uniformly random element α of \mathbb{F}_q and checking whether it is primitive by seeing whether $\alpha^{\frac{q-1}{p}} \neq 1$ for every prime factor p of $q - 1$. An alternative algorithm is analyzed in [[Sho05](#), Section 11.1].

Theorem 3.17 (the lossy RS condenser over bits, [GUV09]). *For any constant $\alpha \in (0, 1)$ there exist constants $C_\alpha, C'_\alpha > 0$ such that the following holds for every $n \in \mathbb{N}$, $\varepsilon > 0$, and $k \geq C_\alpha \log(n/\varepsilon)$. There exists a strong (k, k', ε) -condenser*

$$\text{RSCond}: \{0, 1\}^n \times \{0, 1\}^\ell \rightarrow \{0, 1\}^m$$

where $\ell \leq C'_\alpha \log(n/\varepsilon)$, $k \leq m \leq (1 + \alpha)k$, and $k' \geq (1 - \alpha)m$. Moreover, given $x \in \{0, 1\}^n$, $y \in \{0, 1\}^\ell$, and a primitive element ζ for \mathbb{F}_q with q a prime in $[2^\ell, 2^{\ell+1}]$, the output $\text{RSCond}(x, y)$ can be computed in time $\tilde{O}(n)$.

In particular, if $\varepsilon' = \sqrt{\varepsilon}$ and $C_\alpha > 0$ is chosen large enough compared to C'_α , then for all (n, k) -sources X and a $(1 - \varepsilon')$ -fraction of seeds y it holds that $\text{RSCond}(X, y) \approx_{\varepsilon'} Z_y$, where Z_y is an $(m, k' - \ell \geq (1 - 2\alpha)m)$ -source.

Proof. We argue about the output length and the output min-entropy rate. The rest is analogous to the proof of [Theorem 3.14](#). Fix a target $\alpha \in (0, 1)$. We invoke [Theorem 3.15](#) with an appropriately small constant $\beta \in (0, 1)$ in place of α . Regarding the output length m , note that $m = m' \cdot \lceil \log q \rceil \geq \frac{k}{\log q} \cdot \log q = k$, and that

$$m = m' \lceil \log q \rceil \leq m' \log q + m' \leq k + \log q + m' \leq k + (\ell + 1) + (k/\ell + 1) = \left(1 + \frac{1}{\ell} + \frac{\ell + 2}{k}\right)k.$$

The quantity on the right hand side can be made at most $(1 + \beta)k \leq (1 + \alpha)k$ by setting $\beta < \alpha$ and C_β to be sufficiently larger than C'_β , and by the lower bound on $\ell = \lfloor \log q \rfloor$ in [Theorem 3.15](#). Regarding the output entropy k' , since [Lemma 3.12](#) implies a penalty of ℓ bits in the input min-entropy, from [Theorem 3.15](#) instantiated with β and [Lemma 3.12](#) we get the guarantee that

$$k' \geq (1 - \beta)(k - \ell) \geq (1 - \beta)k - \ell \geq (1 - 2\beta)k \geq \frac{(1 - 2\beta)m}{1 + \beta} \geq (1 - \alpha)m$$

provided that C_β is sufficiently larger than C'_β and that $\beta \leq \alpha/3$.

The “In particular” part of the theorem statement follows analogously to that of [Theorem 3.14](#), since $m \geq k$ and we can assume that $\ell/k \leq \alpha$ by setting C_α to be sufficiently large compared to C'_α . \square

3.3.3 Towards removing preprocessing?

We discuss an approach, suggested by Jesse Goodman, towards removing the preprocessing steps required for the KT and RS condensers, or at least expanding the range of parameters for which preprocessing is not required, and some barriers we face when trying to fully implement this strategy.

KT condenser. As discussed in [Remark 3.11](#), the KT condenser in [Theorem 3.14](#) requires a one-time preprocessing step independent of the source and the seed whenever, roughly, $\varepsilon \leq 2^{-Cn^{0.12}}$. If this holds, a sufficient preprocessing consists of generating a prime $q = \text{poly}(n/\varepsilon)$, which can be done in randomized time $\text{polylog}(n/\varepsilon)$. This raises the following possibility: perhaps we can derandomize this algorithm so that it only uses $O(\log(n/\varepsilon))$ bits of randomness without hurting the running time too much. If this is the case, then the randomness used by the algorithm can be absorbed into the seed, and we get a condenser running in time $\tilde{O}(n + \text{polylog}(n/\varepsilon))$, without preprocessing. When ε is not too small (depending on the exponent of the $\text{polylog}(n/\varepsilon)$ term), this is $O(n)$.

We can realize this approach by combining the randomized prime generation algorithm with an averaging sampler. A similar strategy was used in [BIW06]. We sketch how this can be done. Let $N = \text{poly}(n/\varepsilon)$. The randomized prime generation algorithm independently samples $O(\log N)$ uniformly random integers $q \in [N/2, N]$, and checks whether at least one of these numbers is prime. Primality can be tested in deterministic time $\tilde{O}(\log^6 N)$ using an improved variant of the AKS primality test [Len02], and, by the prime number theorem, with high enough probability there will be at least one prime number among the samples. One way to derandomize this algorithm is by using a (γ, θ) -averaging sampler $\text{Samp}: \{0, 1\}^r \rightarrow [N]^m$ with accuracy $\theta = \Theta(1/\log N) = \Theta(1/\log(n/\varepsilon))$ and confidence $\gamma = \varepsilon$ to generate m candidate primes, and then run the AKS primality test on each candidate. Denoting by T the runtime of Samp , this procedure runs in time $\tilde{O}(T + m \cdot \log^6(n/\varepsilon))$, uses r bits of randomness, and generates the desired prime except with probability at most ε .

With some hindsight, it turns out that this running time is not enough to beat the runtime of the deterministic preprocessing from Remark 3.11, even ignoring T and using a sampler with optimal sample complexity m . However, it is possible to improve on the runtime by replacing the AKS primality test with a faster probabilistic test, such as Miller-Rabin [Sho05, Section 10.2], and use a second sampling round to derandomize these probabilistic tests. More precisely, one iteration of the Rabin-Miller algorithm for testing primality of an integer q samples α uniformly at random from $[q]$ and checks whether α passes a certain test in time $\tilde{O}(\log^2 q)$.²³ When q is prime all α -s pass the test, while for q composite α fails the test with probability at least $3/4$. We use a (γ', θ') -averaging sampler $\text{Samp}': \{0, 1\}^{r'} \rightarrow [N]^{m'}$ with $\theta' = 1/4$ and $\gamma' = \varepsilon/m$ to generate $\alpha_1, \dots, \alpha_{m'}$, which will be used to run Miller-Rabin tests on the prime candidates q_1, \dots, q_m generated by the first sampler.²⁴ Note that for any fixed composite q_i , the probability that all Miller-Rabin tests $\alpha_1 \bmod q_i, \dots, \alpha_{m'} \bmod q_i$ pass is at most ε/m . By a union bound over q_1, \dots, q_m , the probability that this holds for at least one such q_i is at most $m \cdot \varepsilon/m = \varepsilon$. Therefore, if Samp and Samp' run in time T and T' , respectively, this procedure runs in time $\tilde{O}(T + T' + m \cdot m' \cdot \log^2(n/\varepsilon))$, uses $r + r'$ bits of randomness, and generates the desired prime except with probability at most 2ε .

We already know from Remark 3.11 that the KT condenser does not require preprocessing to run in $\tilde{O}(n)$ time when $\varepsilon > 2^{-Cn^{0.12}}$, so it is relevant to explicitly work out the best possible improvement afforded by the approach above, assuming we are aiming for $\tilde{O}(n)$ runtime. We can instantiate Samp and Samp' with the nearly-optimal efficient averaging samplers of Xun and Zuckerman [XZ25, Theorem 2]. Under the choices of N , (θ, γ) , and (θ', γ') above, for any constant $\delta > 0$, we can instantiate Samp with randomness complexity $r = O_\delta(\log(n/\varepsilon))$ and sample complexity $m = O(\log^{3+\delta}(n/\varepsilon))$, and Samp' with randomness complexity $r' = O_\delta(\log(n/\varepsilon))$ and sample complexity $m' = O(\log^{1+\delta}(n/\varepsilon))$.

Therefore, using these samplers, for any constant $\delta > 0$, the prime-generating procedure above uses $O_\delta(\log(n/\varepsilon))$ bits of randomness, which can be absorbed into the seed of the condenser, runs in time $\tilde{O}(T + T' + \log^6(n/\varepsilon))$, with T and T' the runtimes of Samp and Samp' , and fails with probability at most 2ε , which can be absorbed into the error of the condenser. One can verify that T and T' are $O(\log^6(n/\varepsilon))$ in this regime. Therefore, the time bound above is $\tilde{O}(n)$ when $\varepsilon > 2^{-Cn^\gamma}$ for any constant $\gamma < 1/6$, which improves on the simpler $\varepsilon > 2^{-Cn^{0.12}}$ bound from Remark 3.11.²⁵ We also note that we cannot hope to improve on this bound by picking a better averaging sampler,

²³Shoup [Sho05, Section 10.2] states that the test is computable in time $\tilde{O}(\log^3 q)$ via repeated squaring, but more sophisticated techniques yield the $\tilde{O}(\log^2 q)$ bound [Nar14].

²⁴Note that Samp' outputs samples from $[N]$, while the guarantees for the Miller-Rabin test on q hold for α uniformly random over $[q]$. But since each $q_i \in [N/2, N]$, taking the test to be $\alpha \bmod q_i$ for α uniformly random over $[N]$ only decreases the failure probability from at least $3/4$ to at least $1/2$.

²⁵In contrast, using the AKS primality test would lead to running time $\tilde{O}(\log^9(n/\varepsilon))$, which is only $\tilde{O}(n)$ when $\varepsilon > 2^{-Cn^{1/9}}$, and therefore worse than the bound from Remark 3.11.

since any (θ, γ) -averaging sampler must have sample complexity $m = \Omega(\log(1/\gamma)/\theta^2)$ [CEG95].

RS condenser. Unlike the KT condenser, the RS condenser from [Theorem 3.17](#) always requires preprocessing. Therefore, arguably the most interesting direction in this discussion would be to establish a statement of the form “if $\varepsilon \geq 2^{-n^\gamma}$ for some constant $\gamma > 0$, then the RS condenser does not require preprocessing to run in time $\tilde{O}(n)$ ”, in analogy with the statement we already have for the KT condenser. However, it is not clear to us how to implement a sampler-based strategy in this case. We elaborate on this below.

The preprocessing algorithm considered in [Remark 3.16](#) has two stages: (1) repeatedly sample an integer q alongside its prime factorization uniformly at random from $[N/2, N]$, with $N = \text{poly}(n/\varepsilon)$, until $q+1$ is prime, as in [Sho05, Section 9.6], and (2) find a primitive element of \mathbb{F}_q . We focus on the first stage. The first part of this stage requires sampling a “random non-increasing sequence” in $[1, N]$ (see [Sho05, Section 9.5 (Algorithm RS)]). However, this procedure requires too much randomness, which blows up the seed length of the sampler. Indeed, following the analysis in [Sho05, Section 9.5.2] gives that Algorithm RS requires $\Theta(\log^2 N) = \Theta(\log^2(n/\varepsilon))$ bits of randomness in expectation. More precisely, let O_i be the random variables denoting the number of times the integer i appears in the non-increasing sequence, as defined in [Sho05, Section 9.5.2]. There, it is shown that $\mathbb{E}[O_i] = \frac{1}{i-1}$ for all integers $i \in [2, N]$. Every time i is sampled requires using at least $\log i$ bits of randomness in that iteration of Algorithm RS. Therefore, the expected number of bits of randomness required is at least

$$\sum_{i=1}^N \log i \cdot \mathbb{E}[O_i] \geq \sum_{i=1}^N \frac{\log i}{i} \geq \int_1^N \frac{\log x}{x} dx - O(1) = \Theta(\log^2 N).$$

Therefore, the sampler must output samples of bitlength $\Omega(\log^2(n/\varepsilon))$, and so also requires a seed of length $\Omega(\log^2(n/\varepsilon))$.

We note that the barriers outlined above are specific to the preprocessing steps we considered in this work, and to the use of samplers. It may be possible to improve on the current results by considering alternative preprocessing steps and by using other pseudorandom objects. We leave this as a natural direction for future work.

4 Nearly-Linear Time Extractors with Order-Optimal Seed Length

4.1 A Non-Recursive Construction

In this section, we use the sampler based on bounded independence from [Section 3.2](#) and the nearly-linear time KT condenser from [Section 3.3](#) to construct a seeded extractor with order-optimal seed length $O(\log(n/\varepsilon))$ computable in time $\tilde{O}(n)$. We remark that the goal of this section is to give a low-error, relatively simple (and in particular, non-recursive) construction. Among the non-recursive “sample-then-extract” extractor constructions, two notable ones are [NZ96] and [Zuc96]: The [NZ96] construction uses poly-logarithmic seed (see [Footnote 26](#)); The [Zuc96] construction bears some resemblance to our construction and also utilizes sub-sampling, but only works in the high-error regime, namely has seed length $\Theta(\varepsilon^{-2} + \log n)$. Constructions in that framework that support low error and short seed include [SZ99] and followup works such as [GUV09, Zuc97]. We adapt those recursive constructions in [Section 4.2](#) to get our nearly linear time construction of [Theorem 1](#). Finally, we note that other block source conversion techniques have been used for constructing pseudorandomness primitives, such as in [Ta-02, DKSS13], but they are less relevant in our context.

In a nutshell, our extractor proceeds as follows on input an (n, k) -source X :

1. Using a fresh seed, apply the lossless KT condenser from [Theorem 3.14](#) to X . This yields an (n', k) -source X' of length $n' \approx k$ and constant entropy rate δ which can be arbitrarily close to 1. Note that in the parameter regime considered in this section the KT condenser does not require the one-time preprocessing step.
2. Using the fact that X' has high min-entropy rate, use the bounded-independence sampler from [Lemma 3.9](#) to sample subsources from X' using a fresh seed. Specific properties of the bounded-independence sampler allow us to obtain a block source $Z = (Z_1, Z_2, \dots, Z_t)$ with a seed of length only $O(\log(1/\varepsilon))$. The number of blocks is $t = O(\log n)$ and the blocks Z_i have geometrically *increasing* lengths, up to an n^α length threshold.
3. Now, to prepare for the hash-based iterative extraction, we need to make our blocks *decreasing*. Again, using a short seed, of length $O(\log(n/\varepsilon))$, we transform Z into $S = (S_1, \dots, S_t)$, where the blocks are now geometrically decreasing. The blocks lengths will vary from n^{β_1} to some n^{β_2} , for some constants $\beta_1 > \beta_2$. (Here, we do not use the “prefix samplers” property of [Lemma 3.9](#), so transforming Z into S can be done using an existing sampler, and specifically the one from [Lemma 2.20](#).)
4. Using a fresh seed, apply the fast hash-based extractor from [Lemma 2.13](#) to perform block source extraction from S . Noting that the first block has length $n^{\Omega(1)}$, the block source extraction only outputs $n^{\Omega(1)}$ bits.

While the seed length of [Lemma 2.13](#) requires at least m random bits, we are still able to use only $O(\log(n/\varepsilon))$ bits, since we do not output $n^{\Omega(1)}$ bits already at the beginning of the iterative extraction process, but instead first output logarithmically many bits, and then gradually increase the output length.

Finally, once we have extracted $n^{\Omega(1)}$ random bits, outputting almost all the entropy can be done using standard techniques (see [Section 4.1.4](#)). These steps will culminate in the following theorem.

Theorem 4.1 (non-recursive construction). *There exists a constant $c \in (0, 1)$ such that for every positive integers n and $k \leq n$, any $2^{-k^c} \leq \varepsilon \leq \frac{1}{n}$, and any constant $\eta \in (0, 1)$, there exists a strong (k, ε) extractor*

$$\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m,$$

where $d = O(\log(n/\varepsilon))$, and $m = (1 - \eta)k$. Moreover, given inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, we can compute $\text{Ext}(x, y)$ in time $\tilde{O}(n)$.

4.1.1 Item 2: Generating the block source

Because of the initial condensing step, we will assume from here onwards that our input source X is an $(n, k = \delta n)$ -source with constant δ . In order to generate the desired block source, we first use a fresh seed Y as input to an appropriate instantiation of the bounded-independence sampler Γ from [Lemma 3.9](#). This yields a tuple of coordinates $\Gamma(Y) = j_1, \dots, j_{m_t}$ from $[n]$, such that $\Gamma(Y)|_{[1, m_i]}$ is an appropriate averaging sampler for every i . Then, we use these coordinates to sample subsources from $X \sim \{0, 1\}^n$, and get a block source with *increasing* blocks. We recall that getting increasing blocks is only an intermediate step towards our final goal. Indeed, in this step, we sample from the source X , which will guarantee that a typical prefix leaves enough entropy in the next, larger, blocks. In [Section 4.1.2](#) we will need to subsample from those blocks and argue that each block still has entropy even after a typical fixing of all the blocks that precedes it. The latter property is the one needed for block-source extraction.

Lemma 4.2 (sampling a block source). *There exists a deterministic procedure that given an (n, k) -source X with $k \geq \delta n$, δ being constant, and:*

- A constant loss parameter $\zeta \in (0, 1)$,
- A closeness parameter $\varepsilon \in (0, 1)$,
- Number of desired blocks $t \in \mathbb{N}$,
- A final, maximal, block length $\Delta_t \leq c_t \cdot n$ where $c_t = c_t(\zeta, \delta)$ is constant, and,

takes an independent and uniform random seed $Y \sim \{0, 1\}^{d_{\text{samp}}}$ and outputs a random variable Z such that, for every y , $(Z|Y = y)$ is ε -close to an exact

$$((\Delta_1, \dots, \Delta_t), (1 - \zeta)\delta)$$

block-source, where each $\Delta_{i-1} = \alpha \cdot \Delta_i$ for $\alpha = \frac{\zeta\delta}{16}$ and assuming that $\varepsilon \geq 2^{-(c_\varepsilon \Delta_1 - t)}$ for some constant $c_\varepsilon = c_\varepsilon(\zeta, \delta)$. Moreover, the seed length $d = O\left(\frac{\log n}{\log \Delta_1} \cdot \log \frac{t}{\varepsilon}\right)$, and the procedure runs in time $\tilde{O}(n) + \text{polylog}(t/\varepsilon)$.

Note that for any constants $0 < \theta_1 < \theta_2 < 1$, and any $\varepsilon \geq 2^{-n^c}$ where $c > 0$ is a small enough constant, we can have $\Delta_t = n^{\theta_2}$ and $\Delta_1 = n^{\theta_1}$ for some $t = O(\log n)$, with seed length $O(\log(t/\varepsilon))$ and runtime $\tilde{O}(n)$.

Proof. Given our $\Delta_1, \dots, \Delta_t$, we let $m_i = \sum_{j=1}^i \Delta_j$ for $j \in [t]$. Note that for $i \in [t-1]$, each $m_i = \sum_{j=1}^i \Delta_j \leq \frac{\alpha}{1-\alpha} \Delta_{i+1}$, so in particular

$$m_t = m_{t-1} + \Delta_t \leq \frac{\alpha}{1-\alpha} \Delta_t + \Delta_t \leq n,$$

by choosing the constant c_t appropriately. Let $\Gamma: \{0, 1\}^{d_{\text{samp}}} \rightarrow [n]^{m_t}$ be the $(\gamma, \varepsilon_\Gamma)$ -averaging sampler of [Lemma 3.9](#), set with $\varepsilon_\Gamma = \frac{1}{\log(\frac{6}{\zeta\delta})} \cdot \frac{\zeta\delta}{6} = O(1)$ and $\gamma = \frac{\varepsilon}{8t}$. Note that then,

$$d_{\text{samp}} = O\left(\frac{\log n}{\log m_1} \cdot \log \frac{1}{\gamma}\right) = O\left(\frac{\log n}{\log \Delta_1} \cdot \log \frac{t}{\varepsilon}\right),$$

and indeed $m_t \leq \frac{\varepsilon_\Gamma}{16} \cdot n$ can be met by, again, setting the constant $c_t \in (0, 1)$ appropriately. Moreover, we have that for any $i \in [t]$,

$$W_i = \Gamma(Y)|_{[1, m_i]}$$

is a $(\gamma, \varepsilon_\Gamma)$ sampling distribution, where $w \sim W_i$ has distinct symbols. Set $\beta = \frac{\zeta}{2}$.

For each $i \in [t]$, let $A_i = X_{W_i}$. We may write $A_t = (Z_1, \dots, Z_t)$ with $Z_j = (A_t)_{[m_{j-1}+1, m_j]}$. Note that under this perspective we have $A_i = (Z_1, \dots, Z_i)$ for each $i \in [t]$. We claim that (Z_1, \dots, Z_t) is close to an appropriate (exact) block source $Z' = (Z'_1, \dots, Z'_t)$, even conditioned on the seed Y . This follows by an induction argument similar to that of [\[NZ96, proof of Lemma 17\]](#) which we detail below.

First, [Lemma 2.15](#) instantiated with $\tau = \frac{\beta\delta}{3}$ (notice that indeed $\varepsilon_\Gamma \leq \frac{\tau}{\log(1/\tau)}$) tells us that

$$(Y, Z_1, \dots, Z_i) = (Y, A_i) \approx_{\gamma+2^{-\Omega(\tau n)}} (Y, A'_i), \quad (1)$$

where $(A'_i|Y = y)$ has entropy rate at least $\delta - 3\tau \geq (1 - \beta)\delta$ for every y . [Equation \(1\)](#) with $i = 1$ and $Z'_1 = A'_1$ is our base case. Now, fix an arbitrary $i \geq 2$ and suppose that we already know that

$$(Y, Z_1, \dots, Z_{i-1}) \approx_{2(i-1)(\gamma+2^{-\Omega(\tau n)}+\xi)} (Y, Z'_1, \dots, Z'_{i-1}), \quad (2)$$

where $(Z'_j|Y = y, Z'_1 = z_1, \dots, Z'_{j-1} = z_{j-1})$ has entropy rate at least $(1 - \zeta)\delta$ for every $1 \leq j \leq i-1$ and every y, z_1, \dots, z_{j-1} , and $\xi = \frac{\varepsilon}{4t}$.

Write $A'_i = (Z''_1, \dots, Z''_i)$ with $|Z''_j| = |Z'_j|$. Applying [Lemma 2.6](#) to $(A'_i|Y = y)$ with $\delta = \xi$ yields

$$\begin{aligned} \mathbf{H}_\infty(Z''_i|Y = y, Z''_1 = z_1, \dots, Z''_{i-1} = z_{i-1}) &= \mathbf{H}_\infty(A'_i|Y = y, Z''_1 = z_1, \dots, Z''_{i-1} = z_{i-1}) \\ &\geq \mathbf{H}_\infty(A'_i) - \sum_{j=1}^{i-1} \Delta_j - \log(1/\xi) \\ &= \mathbf{H}_\infty(A'_i) - m_{i-1} - \log(1/\xi) \\ &\geq (1 - \beta)\delta m_i - \frac{\alpha}{1 - \alpha} \Delta_i - \log(1/\xi) \end{aligned} \quad (3)$$

except with probability at most ξ over the choice of (z_1, \dots, z_{i-1}) . Using the fact that $m_i \geq \Delta_i$, to get entropy rate at least $(1 - \zeta)\delta$ it is left to verify that

$$\left((\zeta - \beta)\delta - \frac{\alpha}{1 - \alpha} \right) \Delta_i = \left(\frac{\beta\delta}{2} - \frac{\alpha}{1 - \alpha} \right) \Delta_i \geq \log(1/\xi).$$

Using our bound on α , we get that $\frac{\alpha}{1 - \alpha} \leq \frac{\beta\delta}{4}$. Thus, $\frac{\beta\delta}{2} \Delta_i \geq \log(1/\xi)$ holds whenever $\log(1/\varepsilon) \leq c_\varepsilon \Delta_1 - t$, where both c_ε depend only on δ and ζ .

Call a vector $\vec{v} = (y, z_1, \dots, z_{i-1})$ *good* if [Equation \(3\)](#) holds. Suppose that we already have blocks B_1, \dots, B_{i-1} , arbitrarily distributed. We generate one more block B_i as follows. First, sample $\vec{v} \sim (Y, B_1, \dots, B_{i-1})$. If \vec{v} is good as defined above, we set $B_{i,\vec{v}}$ (the random variable B_i conditioned on $(Y, B_1, \dots, B_{i-1}) = \vec{v}$) to be $B_{i,\vec{v}} = (Z''_i|(Y, Z''_1, \dots, Z''_{i-1}) = \vec{v})$. Otherwise, including when \vec{v} is not in the support of $(Y, Z''_0, \dots, Z''_{i-1})$, we set $B_{i,\vec{v}}$ to be uniformly distributed over $\{0, 1\}^{\Delta_i}$ and independent of everything else. Note that by construction we have $\mathbf{H}_\infty(B_{i,\vec{v}}) \geq (1 - \zeta)\delta \Delta_i$ for all \vec{v} .

Then, it follows from [Equation \(1\)](#) that

$$(Y, Z_1, \dots, Z_{i-1}, B_i^{(1)}) \approx_{\gamma+2-\Omega(\tau n)} (Y, Z''_1, \dots, Z''_{i-1}, B_i^{(2)}), \quad (4)$$

where on the left-hand side we take $B_i^{(1)}$ to be sampled based on Y and $B_j = Z_j$ for $1 \leq j \leq i-1$ as described for B_i in the previous paragraph, and on the right-hand side we take $B_i^{(2)}$ to be sampled based on Y and $B_j = Z''_j$ for $0 \leq j \leq i-1$. Since $\vec{v} \sim (Y, Z''_1, \dots, Z''_{i-1})$ is good with probability at least $1 - \xi$, in which case $B_{i,\vec{w}}$ is sampled identically to $Z''_{i,\vec{w}}$, we get from [Equation \(4\)](#) and a triangle inequality that

$$(Y, Z_1, \dots, Z_{i-1}, B_i^{(1)}) \approx_{\gamma+2-\Omega(\tau n)+\xi} (Y, Z''_1, \dots, Z''_{i-1}, Z''_i). \quad (5)$$

By [Equation \(2\)](#), we also have that

$$(Y, Z_1, \dots, Z_{i-1}, B_i^{(1)}) \approx_{2(i-1)(\gamma+2-\Omega(\tau n)+\xi)} (Y, Z'_1, \dots, Z'_{i-1}, B_i^{(3)}), \quad (6)$$

where, again, $B_i^{(3)}$ is sampled based on Y and $B_j = Z'_j$ for $1 \leq j \leq i-1$ as described for B_i above. Combining [Equation \(1\)](#) (recall that $A'_i = (Z''_1, \dots, Z''_i)$) with [Equations \(5\)](#) and [\(6\)](#) via a triangle inequality, we get that

$$(Y, Z_1, \dots, Z_i) \approx_{2i(\gamma+2-\Omega(\tau n)+\xi)} (Y, Z'_1, \dots, Z'_{i-1}, B_i^{(3)}).$$

Note that the sampling of $B_i^{(3)}$ on the right-hand side of this equation guarantees that $\mathbf{H}_\infty(B_i^{(3)}|Y = y, Z'_1 = z_1, \dots, Z'_{i-1} = z_{i-1}) \geq (1 - \zeta)\delta\Delta_i$ for all (y, z_1, \dots, z_{i-1}) . Therefore, $(Z'_1, \dots, Z'_{i-1}, Z'_i = B_i^{(3)})$ is indeed the target block source with i blocks. Setting $i = t$, and by inspection of the sampling process for the Z'_j -s, gives that $(Z|Y = y)$ is

$$2t \cdot \left(\gamma + 2^{-\Omega(\tau n)} + \xi \right) \leq \varepsilon$$

close to an exact $((\Delta_1, \dots, \Delta_t), (1 - \zeta)\delta)$ block source for every y .

The bound on the runtime follows easily, recalling that Γ runs in time $\tilde{O}(n) + \text{polylog}(1/\gamma)$. \square

4.1.2 Item 3: Subsampling from the block source

To apply iterative extraction, we will need our block source to have *decreasing* blocks. Here, we will use a sampler to sample from each block, using the same seed across the blocks.

Lemma 4.3 (subsampling from a block source). *There exists a deterministic procedure that given a*

$$((\Delta_1, \dots, \Delta_t), \delta)$$

block-source $Z = (Z_1, \dots, Z_t)$, where $\Delta_1 \leq \dots \leq \Delta_t$ and δ is a constant,

- *A constant shrinkage parameter $\alpha \in (0, 1)$,*
- *A constant loss parameter $\zeta \in (0, 1)$,*
- *A closeness parameter $\varepsilon \in (0, 1)$,*
- *An initial, maximal, block length $\ell_1 \leq \Delta_1$, and,*
- *An independent and uniform random seed $Y \sim \{0, 1\}^{d_{\text{samp}}}$,*

satisfies the following. Assuming that $\ell_t \geq c_1 \log(t/\varepsilon)$ for some constant $c_1 = c_1(\zeta, \delta)$, it outputs a random variable S such that, for every y , $(S|Y = y)$ is ε -close to an exact

$$((\ell_1, \dots, \ell_t), (1 - \zeta)\delta)$$

block-source, where each $\ell_{i+1} = \alpha \cdot \ell_i$. Moreover, the seed length $d_{\text{samp}} = \log \frac{\Delta_t}{\ell_1} + O(t + \log \frac{1}{\varepsilon})$, and the procedure runs in time $t \cdot \text{polylog}(\Delta_t)(\ell_1 + \log^2(t/\varepsilon))$.

Note that when $\Delta_t = n^{\theta_1}$ and $\ell_1 = n^{\beta}$ for some constants $\theta_1, \beta \in (0, 1)$, $d_{\text{samp}} = O(\log(n/\varepsilon))$, the procedure runs in time $O(n)$, and we can take any $\varepsilon \geq 2^{-c \cdot \ell_t}$ for some constant c that depends on ζ and δ .

Proof. For $i \in [t]$, let $m_i = \sum_{j=1}^i \ell_j$, recalling that $\ell_i = \alpha^{i-1} \ell_1$. For each $i \in [t]$, let $\Gamma_i: \{0, 1\}^{d_i} \rightarrow [\Delta_i]^{\ell_i}$ be the $(\gamma, \varepsilon_\Gamma)$ distinct-samples sampler of Lemma 2.20, where $\gamma = \frac{\varepsilon}{2t}$ and $\varepsilon_\Gamma = \frac{1}{\log(\frac{6}{\zeta\delta})} \cdot \frac{\zeta\delta}{6} = O(1)$. We need to make sure that each $\ell_i \geq c \cdot \frac{\log(1/\gamma)}{\varepsilon_\Gamma^2}$ for some universal constant c , and indeed that is the case, by our constraint on ℓ_t . Also, $d_i = \log(\Delta_i/\ell_i) + O(\log \frac{1}{\gamma} \cdot \text{poly}(1/\varepsilon_\Gamma))$ and we set d_{samp} to be the maximum over the d_i -s, so

$$d_{\text{samp}} = d_t = \log \frac{\Delta_t}{\ell_1} + t \cdot \log \frac{1}{\alpha} + O\left(\log \frac{t}{\varepsilon}\right).$$

We denote the corresponding samples by $W_i = \Gamma_i(Y|_{[1,d_i]})$, and let $S_i = (Z_i)_{W_i}$. Setting $\varepsilon'_i = 2^{-(\zeta/2)\delta\Delta_i}$ and observing that $\delta\Delta_i = (1 - \frac{\zeta}{2})\delta\Delta_i + \log(1/\varepsilon'_i)$, we get that Z is $\varepsilon' = \sum_i \varepsilon'_i$ close to some Z' , an exact $((\Delta_1, \dots, \Delta_t), (1 - \zeta)\delta)$ -source. From here onwards, assume that Z is the exact block source, and aggregate the error.

Next, we invoke [Lemma 2.15](#) with $\tau = \frac{\zeta\delta}{6}$ (notice that indeed $\varepsilon_\Gamma \leq \frac{\tau}{\log(1/\tau)}$), and get that for every $i \in [t]$, and $z_{\text{pre}} \in \{0, 1\}^{\Delta_1 + \dots + \Delta_{i-1}}$,

$$(Y, S_{i,z_{\text{pre}}}) \approx_{\varepsilon''_i = \gamma + 2^{-\Omega(\tau\Delta_i)}} (Y, S'_{i,z_{\text{pre}}}),$$

where $S_{i,z_{\text{pre}}}$ denotes S_i conditioned on $(Z_1, \dots, Z_{i-1}) = z_{\text{pre}}$ and $S'_{i,z_{\text{pre}}}$ satisfies $\mathbf{H}_\infty(S'_{i,z_{\text{pre}}}|Y = y) \geq (1 - \frac{\zeta}{2})^2\delta \cdot \ell_i \geq (1 - \zeta)\delta \cdot \ell_i$ for all y . Thus, in particular, this holds if we condition on any sample from (S_1, \dots, S_{i-1}) , and so we have that for every $i \in [t]$,

$$(Y, S_1, \dots, S_{i-1}, S_i) \approx_{\varepsilon''_i} (Y, S_1, \dots, S_{i-1}, S'_i), \quad (7)$$

where $\mathbf{H}_\infty(S'_i|Y = y, S_1 = s_1, \dots, S_{i-1} = s_{i-1}) \geq (1 - \zeta)\delta \cdot \ell_i$ for all (y, s_1, \dots, s_{i-1}) .

This implies that, conditioned on the seed Y , (S_1, \dots, S_t) has distance

$$\varepsilon' + \sum_{i=1}^t \varepsilon''_i \leq t \cdot (\varepsilon'_1 + \varepsilon''_1) \leq \varepsilon$$

from an (exact) $((\ell_1, \dots, \ell_t), (1 - \zeta)\delta)$ block source, where we used the fact that the $2^{-\Omega(\tau\Delta_1)}$ and $2^{-(\zeta/2)\delta\Delta_1}$ terms are at most $\frac{\varepsilon}{4t}$, which follows from the fact that $c_1 \log(t/\varepsilon) \leq \Delta_1$ for a suitable choice of c_1 , and where ε' accounts for the assumption that Z is the exact block source above. This can be shown by induction on the number of blocks using [Equation \(7\)](#) analogously to (and even in a simpler way than) [\[NZ96, proof of Lemma 17\]](#), and similarly to the proof of [Lemma 4.2](#). Since we believe this proof is easier to follow than the proof of [Lemma 4.2](#), we give details here too for the sake of exposition.

First, the base case is given by [Equation \(7\)](#) for $i = 1$. Now, fix an arbitrary $i \geq 2$ and suppose that we already know that

$$(Y, S_1, \dots, S_{i-1}) \approx_{\sum_{j=1}^{i-1} \varepsilon''_j} (Y, S'_1, \dots, S'_{i-1}), \quad (8)$$

where $\mathbf{H}_\infty(S'_j|Y = y, S'_1 = s_1, \dots, S'_{j-1} = s_{j-1}) \geq (1 - \zeta)\delta\ell_j$ for every $1 \leq j \leq i-1$ and all (y, s_1, \dots, s_{j-1}) .

We now show how to extend this by one block. Generally speaking, suppose that we already have blocks B_1, \dots, B_{i-1} , arbitrarily distributed. Analogously to the proof of [Lemma 4.2](#), we generate one more block B_i by first sampling $\vec{v} \sim (Y, B_1, \dots, B_{i-1})$. If \vec{v} is in the support of (Y, S_1, \dots, S_{i-1}) , we set $B_{i,\vec{v}}$, the random variable B_i conditioned on $(Y, B_1, \dots, B_{i-1}) = \vec{v}$, to be $B_{i,\vec{v}} = (S'_i|(Y, S_1, \dots, S_{i-1}) = \vec{v})$. Otherwise, we set $B_{i,\vec{v}}$ to be uniformly distributed over $\{0, 1\}^{\ell_i}$ and independent of everything else. By construction, $\mathbf{H}_\infty(B_{i,\vec{v}}) \geq (1 - \zeta)\delta\ell_i$ for all \vec{v} .

From [Equation \(8\)](#), it follows that

$$(Y, S_1, \dots, S_{i-1}, B_i^{(1)}) \approx_{\sum_{j=1}^{i-1} \varepsilon''_j} (Y, S'_1, \dots, S'_{i-1}, B_i^{(2)}), \quad (9)$$

where $B_i^{(1)}$ is sampled by following the procedure in the paragraph above with $B_j = S_j$ for all $j \leq i-1$, and $B_i^{(2)}$ is sampled from $B_j = S'_j$ for all $j \leq i-1$. Now, note that $(Y, S_1, \dots, S_{i-1}, B_i^{(1)})$

is distributed exactly like $(Y, S_1, \dots, S_{i-1}, S'_i)$, because when $B_j = S_j$ for all $j \leq i-1$ we get that \vec{v} above is always in the correct support. Therefore, combining this observation with [Equations \(7\)](#) and [\(9\)](#) and a triangle inequality yields

$$(Y, S_1, \dots, S_{i-1}, S_i) \approx_{\varepsilon''_i + \sum_{j=1}^{i-1} \varepsilon''_j = \sum_{j=1}^i \varepsilon''_j} \left(Y, S'_1, \dots, S'_{i-1}, B_i^{(2)} \right).$$

To conclude the argument, it suffices to note that $(S'_1, \dots, S'_{i-1}, S'_i = B_i^{(2)} | Y = y)$ is the desired exact block source by inspection of the sampling process for the S'_j -s, and take $i = t$.

To establish the runtime, note that we simply apply Γ_i for each $i \in [t]$, which takes

$$\sum_{i=1}^t \log^2(1/\gamma) \cdot \text{polylog}(\Delta_i) + O(\ell_i \log \Delta_i) \leq t \cdot \text{polylog}(\Delta_t) (\ell_1 + \log^2(t/\varepsilon))$$

time. This concludes our lemma. \square

4.1.3 Item 4: Applying a block source extractor

We now wish to extract from our decreasing-blocks block source, and for that we combine [Lemmas 4.2](#) and [4.3](#) with the block source extraction of [Lemma 2.22](#). This will give us a nearly linear-time logarithmic-seed extractor that outputs $n^{\Omega(1)}$ bits. For the Ext_i -s in [Lemma 2.22](#), we will use the fast hash-based extractors from [Lemma 2.13](#).

Lemma 4.4. *There exists a small constant $c > 0$ such that the following holds. For every large enough n , any constant $\delta \in (0, 1)$, any $k \geq \delta n$, and any $\varepsilon \geq 2^{-n^c}$, there exists a strong (k, ε) extractor*

$$\text{Ext}_{\text{short}} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$$

where $d = O(\log(n/\varepsilon))$, and $m = n^c$. Moreover, given inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, we can compute $\text{Ext}_{\text{short}}(x, y)$ in time $\tilde{O}(n)$.

Proof. Let X be an $(n, k = \delta n)$ -source. Set $\varepsilon' = \varepsilon/3$, $\theta_1 = 8/10$, $\theta_2 = 9/10$, and $\zeta = 1/10$. We first apply [Lemma 4.2](#) with $\Delta_t = n^{\theta_2}$, $\Delta_1 = n^{\theta_1}$, and error ε' , where $t = O(\log n)$ is as guaranteed from [Lemma 4.2](#). This requires a seed of length $d_1 = O(\log(1/\varepsilon')) = O(\log(1/\varepsilon))$, and in time $\tilde{O}(n)$ we output a random variable Z_1 which is ε' -close to an exact $((\Delta_1, \dots, \Delta_t), (1 - \zeta)\delta)$ block source for every fixing of the seed.

Set $\beta = 7/10$, and $\gamma = 6/10 < \beta$. Set α to be the constant such that $n^\beta \cdot \alpha^{t-1} = n^\gamma$. We then apply [Lemma 4.3](#) on Z_1 with that α , the same ζ , closeness ε' and an initial block length $\ell_1 = n^\beta$. This gives us a random variable Z_2 that is $2\varepsilon'$ -close to a

$$\left((\ell_1 = n^\beta, \dots, \ell_t = n^\gamma), \delta' \triangleq (1 - \zeta)^2 \delta \right)$$

block source, requires a seed of length $d_2 = O(\log(n/\varepsilon')) = O(\log(n/\varepsilon))$, and runs in time $t \cdot \text{polylog}(\Delta_t) (\ell_1 + \log^2(t/\varepsilon)) = O(n)$, assuming c is small enough. Again, Z_2 is ε' -close to an exact block source for every fixing of the seed.

For our next and final step, of performing the block-source extraction itself, set $d_3 = c_E \log(\ell_t/\varepsilon_{\text{Ext}})$ where c_E is the constant guaranteed by [Lemma 2.13](#). Also, let $\varepsilon_{\text{Ext}} = \frac{\varepsilon'}{6t}$, and θ will be a constant whose value will be later determined. We will use the following extractors:

- Let $\text{Ext}_t : \{0, 1\}^{\ell_t} \times \{0, 1\}^{d_3} \rightarrow \{0, 1\}^{m_t = (1+\theta)d_3}$ be the $(k_t = \delta' \ell_t, \varepsilon_{\text{Ext}})$ extractor guaranteed to us by [Lemma 2.13](#). Notice that we need to satisfy $k_t \geq \theta d_3 + c_E \log(1/\varepsilon_{\text{Ext}})$. This can be satisfied making sure that ε is at most $2^{-\Omega(\ell_t)}$, where the hidden constant depends on c_E .

- For each $i \in [t-1]$, let

$$\text{Ext}_i: \{0, 1\}^{\ell_i} \times \{0, 1\}^{m_{i+1}} \rightarrow \{0, 1\}^{m_i}$$

be the $(k_i = \delta' \ell_i, \varepsilon_{\text{Ext}})$ extractor guaranteed to us by [Lemma 2.13](#), where $m_i = (1 + \theta)m_{i+1}$. We need to make sure that $m_{i+1} \geq c_E \log(\ell_i / \varepsilon_{\text{Ext}})$ and that $k_i \geq \theta m_{i+1} + c_E \log(1 / \varepsilon_{\text{Ext}})$. To see that the latter holds, note that $k_i = \delta' \cdot \ell_1 \alpha^{i-1} \geq n^{\gamma/2}$ and that $\theta m_{i+1} + c_E \log(1 / \varepsilon_{\text{Ext}}) = \theta(1 + \theta)^{t-i} d_3 + c_E \log(1 / \varepsilon_{\text{Ext}}) < n^{\gamma/2}$, if we choose θ to be a small enough constant (with respect to the constant $\frac{\log n}{t}$) and ε to be, again, at most $2^{-\Omega(\ell_t)}$.

Everything is in place to apply our block source extraction, [Lemma 2.22](#), on Z_2 and an independent and uniform seed of length d_3 . We get that BExt outputs Z_3 of length $m_1 = n^{\Omega(1)}$, which is $2t\varepsilon_{\text{Ext}} \leq \varepsilon'$ close to uniform, and runs in time $O(\sum_{i=1}^t \ell_i \log \ell_i) = O(n)$. Recall that indeed, as [Lemma 2.22](#) requires, all the Ext_i -s output their seed.

To conclude, note that the overall error of our extractor is at most $3\varepsilon' = \varepsilon$, and the seed length is $d_1 + d_2 + d_3 = O(\log(n/\varepsilon))$. \square

4.1.4 Improving the output length

The extractor $\text{Ext}_{\text{short}}$ from [Lemma 4.4](#) only outputs $n^{\Omega(1)}$ bits. Here, we will use an extractor Ext_{out} that outputs a linear fraction of the entropy but requires a (relatively) long seed, and use [Lemma 2.25](#) to boost the output length. For Ext_{out} , we will again use a sample-then-extract extractor, however this time, we can use *independent* samples to create a block source with exponentially decreasing blocks. This setting is easier, and we can simply use the original [\[NZ96\]](#) construction. Since a similar construction will be analyzed later in the paper (including a time complexity analysis), we choose to employ it instead of revisiting [\[NZ96\]](#). We state it formally as a corollary below.

Corollary 4.5. *There exists a constant $C \geq 1$ such that for any constants $\tau, c \in (0, 1)$, any large enough positive integer n and any $\varepsilon \geq 2^{-n^c}$, there exists a strong $(k = (1 - \tau)n, \varepsilon)$ extractor*

$$\text{Ext}_{\text{out}}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$$

where $d = O(\log n \cdot \log(n/\varepsilon))$, and $m = k/C$. Moreover, given inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, we can compute $\text{Ext}_{\text{out}}(x, y)$ in time $\tilde{O}(n)$.

The correctness of [Corollary 4.5](#) follows from [Corollary 4.10](#) applied with $i = 1$ (which is indeed non-recursive), without the need for a preliminary condensing step.²⁶

Plugging-in Ext_{out} and $\text{Ext}_{\text{short}}$ into [Lemma 2.25](#) readily gives the following result.

Lemma 4.6. *There exist constants $\tau, c \in (0, 1)$ such that for every positive integer n , and any $2^{-n^c} \leq \varepsilon \leq \frac{1}{n}$, there exists a $(k = (1 - \tau)n, \varepsilon)$ extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ where $d = O(\log(n/\varepsilon))$, and $m = ck$. Moreover, given inputs $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, we can compute $\text{Ext}(x, y)$ in time $\tilde{O}(n)$.*

To boost the output length in [Lemma 4.6](#) from $\Omega(k)$ to $(1 - \eta)k$ for any constant $\eta > 0$, we apply [Lemma 2.21](#) a constant number of times depending only on η (that is, we simply apply Ext with

²⁶ As mentioned earlier, [Corollary 4.5](#) can already be deduced from [\[NZ96\]](#) (modulo the tight runtime analysis), with a slightly worse seed of $d = O(\log^2 n \cdot \log(n/\varepsilon))$, which would not change the parameters of our overall construction. There, they first convert X into a block source Z using $\log n$ independent samples from a k -wise independent sample space, for $k \approx \log(n/\varepsilon)$. The block source Z has decreasing blocks, so the standard block source extraction scheme can then be employed. The fact that this procedure can be implemented in time $\tilde{O}(n)$ follows easily from the runtime analysis of primitives in our work (specifically, [Item 1](#) of [Lemma 2.1](#), and [Lemma 2.13](#)).

independent seeds and concatenate the outputs). To then go from any min-entropy requirement k to entropy rate $1 - \tau$, we first apply the KT condenser from [Theorem 3.14](#). Since $\varepsilon \geq 2^{-k^c}$ we also have that $\varepsilon \geq 2^{-n^{0.1}}$ if $c < 0.1$, and so the KT condenser does not require preprocessing. Furthermore, we can ensure that $k \geq C' \log^2(n/\varepsilon)$ with $C' > 0$ a sufficiently large constant so that the conditions for applying the KT condenser are satisfied whenever n is larger than some constant.

This finally gives us our main theorem for this section, [Theorem 4.1](#), apart from the strongness property, which we now discuss.

The non-recursive construction is strong. In what follows, we refer to the itemized list in the beginning of the section. The condensing step, [Item 1](#), is strong, since we use strong condensers. Next, the block source creators of [Lemmas 4.2](#) and [4.3](#) are strong, so [Items 2](#) and [3](#) hold in a strong manner as well. [Item 4](#) readily gives a strong extractor. For the output-extending phase, [Lemma 2.25](#) tells us that the extractor from [Lemma 4.6](#) is strong. Finally, we apply that extractor several times with independent seeds, and the strongness of that procedure is guaranteed from [Lemma 2.21](#).

4.2 A Recursive Construction

In this section, we prove the following.

Theorem 4.7 (recursive construction). *For any constant $\eta > 0$ there exists a constant $C > 0$ such that the following holds. For any positive integers n and $k \leq n$ and any $\varepsilon > 0$ satisfying $k \geq C \log(n/\varepsilon)$ there exists a strong (k, ε) -seeded extractor*

$$\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$$

with seed length $d \leq C \log(n/\varepsilon)$ and output length $m \geq (1 - \eta)k$. Furthermore,

1. if $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$;
2. if $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon < 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step, corresponding to generating $O(\log^* n)$ primes $q \leq \text{poly}(n/\varepsilon)$;
3. if $k < 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step, corresponding to generating $O(\log \log n)$ primes $q \leq \text{poly}(n/\varepsilon)$ and a primitive element for each field \mathbb{F}_q .

In a nutshell, our construction behind [Theorem 4.7](#) works by considering two cases. If $\varepsilon > Cn^3 \cdot 2^{-k/\log k}$, then we instantiate the recursive approach of Srinivasan and Zuckerman [[SZ99](#)] appropriately. Otherwise, we apply the recursive approach of Guruswami, Umans, and Vadhan [[GUV09](#)].

4.2.1 The (extremely) low-error case

In this section, we consider the lower error case of [Theorem 4.7](#) where $\varepsilon \leq Cn^3 \cdot 2^{-k/\log k}$. We instantiate the recursive approach from [[GUV09](#), Section 4.3.3] appropriately, and analyze its time complexity. Crucially, because of our upper bound on ε , we will only need to run $O(\log \log n)$ levels of their recursive approach.

In order to obtain the statement of [Theorem 4.7](#) for output length $m \geq (1 - \eta)k$ with η an arbitrarily small constant, it suffices to achieve output length $m = \Omega(k)$ and then apply [Lemma 2.21](#) a constant number of times depending only on η . Therefore, we focus on achieving output length $m = \Omega(k)$.

Theorem 4.8. *There exist constants $c, C > 0$ such that the following holds. For any positive integers n and $k \leq n$ and any $\varepsilon \in (0, Cn^3 \cdot 2^{-k/\log k}]$ further satisfying $k > C \log(n/\varepsilon)$, there exists a strong (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with seed length $d \leq C \log(n/\varepsilon)$ and output length $m \geq k/3$.*

Furthermore, Ext is computable in time $\tilde{O}(n)$ after a preprocessing step that corresponds to finding primitive elements of $O(\log \log n)$ fields \mathbb{F}_q with prime orders $q \leq \text{poly}(n/\varepsilon)$.

Proof. We discuss our instantiation of the recursive approach from [GUV09] in detail, as it will be relevant to the time complexity analysis. Let $\varepsilon_0 = \varepsilon / \log^C n$ and $d = C \log(n/\varepsilon_0) = O(\log(n/\varepsilon))$ for a large enough constant $C > 0$ to be determined later. For an integer $k \geq 0$, let $i(k) = \lceil \log(\frac{k}{8d}) \rceil$, which determines the number of levels in our recursion. It will be important for bounding the time complexity of this construction to observe that

$$i(k) = O(\log \log n) \quad (10)$$

because $\varepsilon \leq Cn^3 \cdot 2^{-k/\log k}$. For each k , we define a family of strong $(k, \varepsilon_{i(k)})$ -seeded extractors $\text{Ext}_{i(k)}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with $\varepsilon_{i(k)} \leq 9\varepsilon_{i(k/3)} + 63\varepsilon_0$ when $i(k) > 0$ by induction on $i(k)$. Solving this recursion yields $\varepsilon_{i(k)} = 2^{O(i(k))} \cdot \varepsilon_0 \leq \varepsilon$, provided that $\varepsilon_0 = \varepsilon / \log^C n$ for a sufficiently large constant $C > 0$.

Base case. For the base case $i(k) = 0$, which holds when $k \leq 8d$, we choose Ext_0 to be the (k, ε_0) -seeded extractor defined below. Before we define and analyze it formally, we informally discuss how the extractor works. Recall that we are aiming for seed length d , which in this base case satisfies $d \geq k/8$, output length $m \geq k/3$, and nearly-linear time complexity. Roughly speaking, on input an (n, k) -source X we first apply the fast RS strong condenser to obtain an output X' that is close to a source with high min-entropy rate. Then, we apply the fast hash-based extractor from Lemma 2.24, which can be made to require only seed length $\approx k/t$ for a large constant t , to X' with a fresh seed. More formally,

1. Apply the lossy RS strong condenser RSCond (Theorem 3.17) on X , instantiated with $\alpha = 1/400$ and error $\varepsilon'_0 = \varepsilon_0/2$. This requires a seed Y_1 of length $d_1 \leq C_0 \log(n/\varepsilon'_0)$, for some constant $C_0 > 0$, and is a valid invocation since $k \geq C \log(n/\varepsilon'_0)$ for a sufficiently large constant $C > 0$. The corresponding output X' satisfies $(Y_1, X') \approx_{\varepsilon'_0} (Y_1, Z)$, for some (n', k') -source Z with $k' \geq (1 - 2\alpha)n' = (1 - 1/200)n'$.
2. Let $\text{Ext}'_0: \{0, 1\}^{n'} \times \{0, 1\}^{d_2} \rightarrow \{0, 1\}^{m'}$ be the average-case strong (k', ε'_0) -seeded extractor from Lemma 2.24 instantiated with $t = 10$, which requires a seed Y_2 of length $d_2 \leq k'/10 + C'_0 \log(n'/\varepsilon'_0)$ for some constant $C'_0 > 0$ and has output length $m' \geq k'/2$. The conditions for the invocation of Lemma 2.24 with $t = 10$ are satisfied since $k' \geq (1 - 1/200)n' = (1 - \frac{1}{20t})n'$ and

$$2^{-n'/500} \leq 2^{-k/500} \leq (\varepsilon_0/n)^{C/500} \leq \varepsilon'_0,$$

where the second inequality uses the theorem's hypothesis that $k \geq C \log(n/\varepsilon)$ with $C > 0$ a sufficiently large constant.

We set $Y = (Y_1, Y_2)$ and define $\text{Ext}_0(X, Y) = \text{Ext}'_0(\text{RSCond}(X, Y_1), Y_2)$. We now argue that Ext_0 is an extractor with the desired properties. From the discussion above, we have

$$(Y, \text{Ext}_0(X, Y)) = (Y_1, Y_2, \text{Ext}'_0(\text{RSCond}(X, Y_1), Y_2)) \approx_{\varepsilon'_0} (Y_1, Y_2, \text{Ext}'_0(Z, Y_2)) \approx_{\varepsilon'_0} (Y_1, Y_2, U_{m'}).$$

Therefore, the triangle inequality implies that Ext_0 is an average-case strong $(k, 2\varepsilon'_0 = \varepsilon_0)$ -seeded extractor. It remains to argue about the seed length, output length, and time complexity of Ext_0 . The seed length of Ext_0 is

$$d_1 + d_2 \leq k'/10 + (C_0 + C'_0) \log(n'/\varepsilon'_0) \leq 0.8d + (C_0 + C'_0) \log(n'/\varepsilon'_0) \leq d,$$

provided that $d = C \log(n/\varepsilon)$ with C a sufficiently large constant. The output length of Ext_0 is $m' \geq k'/2 \geq k/3$, since $k' \geq (1 - 1/200)k$. Finally, both steps above take time $\tilde{O}(n)$, and so Ext_0 can be computed in time $\tilde{O}(n)$ after a one-time preprocessing step.

Induction step. When $i(k) > 0$, we assume the existence of the desired average-case strong extractors $\text{Ext}_{i(k')}$ for all $i(k') < i(k)$ as the induction hypothesis. More precisely, we assume that for all k' such that $i(k') < i(k)$ there exists a family of average-case strong $(k', \varepsilon_{i(k')})$ -seeded extractors $\text{Ext}_{i(k')} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{k'/3}$ parameterized by n computable in time $\tilde{O}(n)$ after a one-time preprocessing step.

Intuitively, we will first use these extractors to construct an extractor $\text{Ext}'_{i(k)}$ with all the desired properties (min-entropy requirement k , small error $\varepsilon' = \varepsilon_{i(k/3)} + 7\varepsilon_0$, small seed length $d' \approx d/16$) *except* that the output length is not large enough. Specifically, we will obtain output length $m = k/9$, but would like to get output length $m \geq k/3$ for the induction step. Roughly speaking, we proceed as follows on input an (n, k) -source X . First, we apply the fast RS strong condenser to X to obtain a source X' with high min-entropy rate. Then, we split X' in half to create a block source (X'_1, X'_2) with two blocks. This split decreases the entropies of X'_1 and X'_2 , so we apply the fast RS strong condenser to X'_2 to replace this second block by a block X''_2 with high min-entropy rate. Finally, we perform block source extraction on (X'_1, X''_2) . Recall that all we are missing after this is a sufficiently large output length. The output length can be boosted via standard techniques, at the expense of slightly larger error and seed length.

More formally,

1. Apply the lossy RS strong $(k, k', \varepsilon_1 = \varepsilon_0^2)$ -condenser RSCond ([Theorem 3.17](#)) on X with $\alpha = 1/20$ and a seed Y_{RS} of length $d_{\text{RS}} \leq C_{\text{RS}} \log(n/\varepsilon_0)$. This is valid since $k > 8d \geq C_\alpha \log(n/\varepsilon)$ with C_α the constant from [Theorem 3.17](#) with $\alpha = 1/20$ if the constant C in the theorem statement is large enough. By the second part of [Theorem 3.17](#) we get that with probability at least $1 - \varepsilon_0$ over the choice of $Y_{\text{RS}} = y$ it holds that the corresponding condenser output X' is ε_0 -close to some (n', k') -source Z with $k' \geq (1 - 2\alpha)n' = 0.9n'$. For the sake of exposition, from here onwards we work under such a good choice of the seed Y_{RS} , and we will add the ε_0 slack term to the final error.
2. Let (X'_1, X'_2) correspond to the first two blocks of $\lfloor n'/2 \rfloor \triangleq n''$ bits of X' . By [Lemma 2.23](#) instantiated with n'' and $\Delta = 0.1n'$ and the fact that X' is ε_0 -close to an (n', k') -source, we get that (X'_1, X'_2) is $(\varepsilon_0 + 2\varepsilon_0 = 3\varepsilon_0)$ -close to an exact $((n'', n''), k''/n'')$ -block-source (W_1, W_2) with

$$k'' \geq n'' - \Delta - \log(1/\varepsilon_0) \geq 0.4n' - \log(1/\varepsilon_0) - 1 \geq k/3, \quad (11)$$

since $n' \geq k > d = C \log(n/\varepsilon_0)$ for a sufficiently large constant $C > 0$.

3. Apply the lossy RS strong $(k'', k''', \varepsilon_1 = \varepsilon_0^2)$ -condenser RSCond' ([Theorem 3.17](#)) to X'_2 with $\alpha = 1/800$ and a seed Y'_{RS} of length at most $d'_{\text{RS}} = C'_{\text{RS}} \log(n''/\varepsilon_1) \leq C'_{\text{RS}} \log(n/\varepsilon_0)$ to get X''_2 . From [Item 2](#) and the data-processing inequality, we know that

$$(Y'_{\text{RS}}, X'_1, X''_2) = (Y'_{\text{RS}}, X'_1, \text{RSCond}(X'_2, Y'_{\text{RS}})) \approx_{3\varepsilon_0} (Y'_{\text{RS}}, W_1, \text{RSCond}(W_2, Y'_{\text{RS}})). \quad (12)$$

Since $(W_2|W_1 = w_1)$ is a k'' -source for any w_1 in the support of W_1 , we conclude from [Theorem 3.17](#) and [Equation \(12\)](#) that

$$(Y'_{\text{RS}}, W_1, \text{RSCond}(W_2, Y'_{\text{RS}})) \approx_{\varepsilon_1} (Y'_{\text{RS}}, W_1, \widetilde{W}_2),$$

where $\widetilde{W}_2 \sim \{0, 1\}^{n''''}$ and $\mathbf{H}_\infty(Y'_{\text{RS}}, \widetilde{W}_2|W_1 = w_1) \geq k''' + d'_{\text{RS}}$ for all w_1 in the support of W_1 , with $n'''' \geq k'' \geq k''' \geq (1 - 1/400)n''''$. This is a valid invocation since $k'' \geq k/3 > 8d/3 > C \log(n/\varepsilon)$ for a large enough constant $C > 0$ by [Equation \(11\)](#). Therefore, by the second part of [Theorem 3.17](#), with probability at least $1 - \varepsilon_0$ over the choice of $Y'_{\text{RS}} = y'$ we get that

$$(W_1, \widetilde{W}_2) | \{Y'_{\text{RS}} = y'\} \approx_{\varepsilon_0} (W_1, W'_2), \quad (13)$$

where $W'_2 \sim \{0, 1\}^{n''''}$ satisfies $\mathbf{H}_\infty(W'_2|W_1 = w_1) \geq k''' \geq (1 - 1/400)n''''$. Fix such a good fixing of Y'_{RS} from now onwards. As before, we will account for the probability ε_0 of fixing a bad seed in the final extractor error. Then, by combining [Equations \(12\)](#) and [\(13\)](#) we get that (X'_1, X''_2) is $(\varepsilon_{\text{BS}} = 4\varepsilon_0)$ -close to an $((n'', n'''), k'', k''')$ -block source.

4. We will now apply block source extraction to (X'_1, X''_2) , which we recall is $(\varepsilon_{\text{BS}} = 4\varepsilon_0)$ -close to an exact $((n'', n'''), k'', k''')$ -block source. We instantiate [Lemma 2.22](#) with Ext_2 being the strong extractor from [Lemma 2.24](#) with source input length n''' , min-entropy requirement k''' , error $\varepsilon_{\text{BExt}} = \varepsilon_0$, output length d , and $t = 16$. This requires a seed of length $d_{\text{BExt}} \leq d/16 + C'_0 \log(n/\varepsilon_0)$. This instantiation of [Lemma 2.24](#) is valid since $k''' \geq (1 - 1/400)n''' > (1 - \frac{1}{20t})n'''$ and

$$k''' \geq 0.95n''' \geq 0.95k'' \geq \frac{0.95k}{3} > \frac{0.95 \cdot 8d}{3} > 2d,$$

where we used the fact that $i(k) > 0$, and so $k > 8d$. For Ext_1 we choose the average-case strong extractor $\text{Ext}_{i(k/3)}$ (recall that $k'' \geq k/3$ and note that $i(k/3) < i(k)$) with input length n'' , entropy requirement $k/3$, error $\varepsilon_{i(k/3)}$, output length at least $(k/3)/3 = k/9$, and seed length d guaranteed by the induction hypothesis above.

[Items 1](#) to [4](#) above yield a strong seeded extractor $\text{Ext}'_{i(k)}: \{0, 1\}^n \times \{0, 1\}^{d'} \rightarrow \{0, 1\}^{m'}$ with min-entropy requirement k , error $\varepsilon' = \varepsilon_{i(k/3)} + \varepsilon_{\text{BExt}} + \varepsilon_{\text{BS}} + 2\varepsilon_0 = \varepsilon_{i(k/3)} + 7\varepsilon_0$ (where the $2\varepsilon_0$ term comes from the two fixings of the seeds in the two condensing steps in [Items 1](#) and [3](#)), seed length

$$d' = d_{\text{BExt}} + d'_{\text{RS}} + d_{\text{RS}} \leq d/16 + C' \log(n/\varepsilon_0),$$

for some constant $C' > 0$, and output length $m' = k/9$.

Boosting the output length of $\text{Ext}'_{i(k)}$. To conclude the definition of $\text{Ext}_{i(k)}$, we need to increase the output length of $\text{Ext}'_{i(k)}$ from $k/9$ to $k/3$. To that end, we use [Lemma 2.21](#). Applying [Lemma 2.21](#) once with $\text{Ext}_1 = \text{Ext}'_{i(k_1)}$ with $k_1 = k$ and $\text{Ext}_2 = \text{Ext}'_{i(k_2)}$ with $k_2 = k - k/9 - 1 = 8k/9 - 1$ and $g = 1$ yields a strong $(k, 3\varepsilon')$ -seeded extractor $\text{Ext}''_{i(k)}$ with output length $(k_1 + k_2)/9 \geq k(1 - (8/9)^2) - 1$ and seed length $2(d/16 + C' \log(n/\varepsilon_0)) = d/8 + 2C' \log(n/\varepsilon_0)$. Applying [Lemma 2.21](#) again with $\text{Ext}_1 = \text{Ext}''_{i(k_1)}$ for $k_1 = k$ and $\text{Ext}_2 = \text{Ext}''_{i(k_2)}$ for $k_2 = (8/9)^2 k$ and $g = 1$ yields a strong $(k, 9\varepsilon')$ -seeded extractor with output length $m \geq k(1 - (8/9)^4) - 1 \geq k/3$ and seed length $2(d/8 + 2C' \log(n/\varepsilon_0)) = d/4 + 4C' \log(n/\varepsilon_0) \leq d$, which we set as $\text{Ext}_{i(k)}$. This second invocation of [Lemma 2.21](#) is also valid, since $k_2 = (8/9)^2 k = k - (k(1 - (8/9)^2) - 1) - 1 = k_1 - m_1 - g$. Note that the error $\varepsilon_{i(k)} = 9\varepsilon' = 9\varepsilon_{i(k/3)} + 63\varepsilon_0$, as desired.

Time complexity and final error. It remains to analyze the time complexity and the overall error of the recursive procedure above. Evaluating $\text{Ext}_{i(k)}$ requires at most eight evaluations of the condenser from [Theorem 3.17](#), four evaluations of the fast hash-based extractor from [Lemma 2.24](#), four evaluations of $\text{Ext}_{i(k'')}$ for some $i(k'') < i(k)$, and simple operations that can be done in time $\tilde{O}(n)$. This means that the overall time complexity is $4^{i(k)} \cdot \tilde{O}(n) = \tilde{O}(n)$ after a one-time preprocessing step independent of the source and seed, since $4^{i(k)} = \text{poly}(\log n)$ by [Equation \(10\)](#). This preprocessing step corresponds to finding primitive elements for $O(\log \log n)$ fields \mathbb{F}_q with prime orders $q \leq \text{poly}(n/\varepsilon_0) = \text{poly}(n/\varepsilon)$. Furthermore, $\varepsilon_{i(k)} = O(\varepsilon_0 + \varepsilon_{i(k/3)})$ for all k , and so $\varepsilon_{i(k)} = 2^{O(i(k))} \varepsilon_0 = \text{poly}(\log n) \cdot \varepsilon_0 \leq \varepsilon$ provided that $\varepsilon_0 \leq \varepsilon / \log^C n$ for a large enough constant $C > 0$. \square

4.2.2 The (relatively) high-error case

In this section, we consider the higher error case where $\varepsilon \geq Cn^3 \cdot 2^{-k/\log k}$. We instantiate the recursive approach of Srinivasan and Zuckerman [[SZ99](#), Section 5.5] appropriately with the fast condensers from [Section 3.3](#), the sampler from [Lemma 2.20](#), and the fast hash-based seeded extractors from [Lemma 2.13](#), and analyze its complexity.

The next lemma shows how we can recursively decrease the seed length of an extractor. We complete the construction by instantiating the base extractor in this recursion appropriately, and then increasing its output length.

Lemma 4.9 (analogous to [[SZ99](#), Corollary 5.10], with different instantiation and additional complexity claim). *There exist constants $c, C > 0$ such that the following holds. Suppose that for any positive integers $n_0, k_0 = 0.7n_0$, and some $\varepsilon_0 = \varepsilon_0(n_0) \geq 2^{-ck_0}$ and $m_0 = m_0(n_0)$ there exists a strong (k_0, ε_0) -seeded extractor $\text{Ext}_0: \{0, 1\}^{n_0} \times \{0, 1\}^{d_0} \rightarrow \{0, 1\}^{m_0}$ with seed length $d_0 \leq u \cdot \log(n_0/\varepsilon_0) \leq k_0$. Then, for any positive integers n and $k \leq n$ there exists a family of strong (k, ε) -seeded extractors $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with error $\varepsilon \leq C \log u \cdot \varepsilon_0(ck)$, seed length $d \leq C \log u \cdot \log(n/\varepsilon_0(ck))$, and output length $m \geq m_0(ck)$. Furthermore,*

1. *If Ext_0 is computable in time $T(n_0)$ and $k \geq C \log^2(n/\varepsilon_0(ck))$, then Ext is computable in time $T(n) + \tilde{O}(n + \sqrt{n} \cdot \log(1/\varepsilon_0(ck)))^5$;*
2. *If Ext_0 is computable in time $T(n_0)$ after a preprocessing step, then Ext is computable in time $T(n) + \tilde{O}(n)$ after a preprocessing step.^{[27](#)}*

Proof. We begin by discussing the high-level approach in this proof. On input an arbitrary (n, k) -source X , Ext proceeds as follows. First, it applies a fast strong condenser to X to obtain a new source X' with high min-entropy rate. If $k \geq C \log^2(n/\varepsilon_0)$ then we can apply the KT condenser, which does not require preprocessing unless ε_0 is very small. Otherwise, we apply the RS condenser. Then, we use X' to generate a block source $Z = (Z_0, \dots, Z_t)$ with geometrically decreasing block lengths. The way we achieve this depends on the regime we are in. If we are in a regime where we must anyway resort to the RS condenser, then we generate each block by applying an appropriately instantiated RS condenser with a fresh seed to X' . Otherwise, if we are in a regime where we can use the KT condenser, then we use the expander random walks averaging sampler from [Lemma 2.20](#), which runs in time $\tilde{O}(n)$ in this regime.^{[28](#)} Finally, we apply block source extraction to Z . More concretely, we begin by applying the fast hash-based extractor from [Lemma 2.13](#) to the shorter

²⁷We discuss the precise preprocessing step in more detail in [Remark 4.11](#).

²⁸The sole reason for this case analysis is that by using the sampler instead of the RS condenser in the “KT regime” we can avoid a preprocessing step unless ε is tiny.

blocks at the end of Z , up until the second block Z_1 of Z . This generates a sufficiently large (but still short) seed that we can use to extract from the first block Z_0 using the base extractor Ext_0 .

We now formally analyze the approach above. We begin by setting up relevant parameters:

- Let $C_{\text{blocks}} \geq 1$ be a constant to be determined. Set $\ell_0 = \frac{k}{100 \cdot C_{\text{blocks}}}$ and $k_0 = 0.7\ell_0$. For $\varepsilon_0 = \varepsilon_0(\ell_0)$ and $m_0 = m_0(\ell_0)$, we define $\ell_1 = C_{\text{blocks}} \cdot u \log(\ell_0/\varepsilon_0)$. Then, we define $\ell_i = 0.9\ell_{i-1}$ for all $i \geq 2$. The ℓ_i 's will be block lengths for a block source Z . In particular, when performing block source extraction from Z we will instantiate Ext_0 with input length $n_0 = \ell_0$.
- Define $m_1 = u \cdot \log(\ell_0/\varepsilon_0)$ and $m_i = 0.9m_{i-1}$ for all $i \geq 2$. The m_i 's will be output lengths for block source extraction from Z .
- Set $t = 1 + \left\lceil \frac{\log(u/\log u)}{\log(1/0.9)} \right\rceil$. This will be the number of blocks of Z . We have $m_t = 0.9^{t-1}m_1 \in [0.9 \log u \cdot \log(\ell_0/\varepsilon_0), \log u \cdot \log(\ell_0/\varepsilon_0)]$. Furthermore, since $\ell_1 = C_{\text{blocks}} \cdot m_1$, we also have that $\ell_i = C_{\text{blocks}} \cdot m_i$ for all $i \geq 1$.

Let X be an arbitrary (n, k) -source. The extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ proceeds as follows on input X :

1. Using a fresh seed Y_{Cond} of length $C_{\text{Cond}} \log(n/\varepsilon_0)$, apply a strong (k, k', ε_0^2) -condenser Cond to X . If $k \geq C \log^2(n/\varepsilon_0)$ for an appropriately large constant $C > 0$, then we instantiate Cond with the KT strong $(k, k' = k, \varepsilon_0^2)$ -condenser ([Theorem 3.14](#)) instantiated with $\alpha = 0.05$. Otherwise, we instantiate Cond with the lossy RS $(k, k' \geq 0.975k, \varepsilon_0^2)$ -condenser ([Theorem 3.17](#)) instantiated with $\alpha = 0.025$. By the second part of either [Theorem 3.14](#) or [Theorem 3.17](#), we get that with probability at least $1 - \varepsilon_0$ over the choice of $Y_{\text{Cond}} = y$ it holds that $X' = \text{Cond}(X, y)$ is ε_0 -close to an (n', k') -source with $k' \geq 0.95n'$.

From here onwards we work under such a good fixing $Y_{\text{Cond}} = y$ and also assume that X' is an (n', k') -source. We account for the resulting $2\varepsilon_0$ error term in the final extractor error later.

2. We use X' to generate a block source $Z = (Z_0, Z_1, \dots, Z_t)$ with geometrically decreasing block lengths. Our procedure depends on the regime of parameters we are in:
 - (a) If $k \geq C \log^2(n/\varepsilon_0)$ for an appropriately large constant $C > 0$, then for each $i = 0, 1, \dots, t$ let $\text{Samp}_i: \{0, 1\}^{r_i} \rightarrow [n']^{\ell_i}$ be the $(\gamma = \varepsilon_0, \theta = 1/100)$ -averaging sampler from [Lemma 2.20](#) with input length $r_i = \gamma_{\text{Samp}} \log(n'/\varepsilon_0)$ for some constant $\gamma_{\text{Samp}} > 0$. We choose the constant C_{blocks} above to be large enough so that $n' \geq \ell_i \geq \ell_t \geq C_{\text{Samp}} \log(1/\varepsilon_0)/\theta^2$ for all $i \in [t]$, where C_{Samp} is the constant C from [Lemma 2.20](#). To see that $\ell_i \leq n'$ for $i = 0, 1, \dots, t$ (and so indeed [Lemma 2.20](#) can be applied to obtain ℓ_i samples), note that

$$\ell_0 + \sum_{i=1}^t \ell_i \leq \ell_0 + \sum_{i=1}^{\infty} \ell_i = \ell_0 + 10\ell_1 \leq k/9 < n'. \quad (14)$$

The second-to-last inequality uses the fact that

$$\ell_1 = C_{\text{blocks}} \cdot u \log(\ell_0/\varepsilon_0) \leq C_{\text{blocks}} \cdot k_0 \leq C_{\text{blocks}} \cdot \ell_0 = k/100,$$

where the first inequality holds since $u \log(\ell_0/\varepsilon_0) \leq k_0$ is an hypothesis in the lemma statement.

By [Lemma 2.15](#) instantiated with X' and Samp_0 , we conclude that

$$(Y_0, Z_0) \approx_{\varepsilon_0 + 2^{-\beta_{\text{Samp}} k}} (Y_0, Z'_0), \quad (15)$$

with $\beta_{\text{Samp}} > 0$ the constant guaranteed by [Lemma 2.15](#), where $(Z'_0|Y_0 = y_0)$ is an $(\ell_0, 0.9\ell_0)$ -source for every y_0 . We now argue how this guarantee extends to more blocks. For each Z_j , define $Z_{j,\vec{y}} = (Z_j|(Y_0, \dots, Y_{i-1}) = \vec{y})$. Consider any fixing $(Y_0, \dots, Y_{i-1}) = \vec{y}$. Then, [Lemma 2.6](#) with $\delta = 2^{-\beta_{\text{Samp}} k}$, where $\beta_{\text{Samp}} > 0$ is taken to be a small enough constant, and $\ell = k/9$ (from the upper bound in [Equation \(14\)](#)) implies that

$$\begin{aligned} \mathbf{H}_\infty(X'|Z_{0,\vec{y}} = z_0, \dots, Z_{i-1,\vec{y}} = z_{i-1}) &\geq k' - \ell - \beta_{\text{Samp}} k \\ &\geq 0.95n' - k/9 - \beta_{\text{Samp}} k \\ &\geq 0.95n' - n'/9 - \beta_{\text{Samp}} n' \\ &\geq 0.8n' \end{aligned} \quad (16)$$

except with probability at most $2^{-\beta_{\text{Samp}} k}$ over the choice $(z_0, \dots, z_{i-1}) \sim (Z_{0,\vec{y}}, \dots, Z_{i-1,\vec{y}})$. Call a fixing $\vec{v} = (y_0, z_0, \dots, y_{i-1}, z_{i-1})$ for which [Equation \(16\)](#) holds *good*. Define $X'_{\vec{v}} = (X'|(Y_0, Z_0, \dots, Y_{i-1}, Z_{i-1}) = \vec{v})$. Then, by [Equation \(16\)](#) and [Lemma 2.15](#) we know that for all good \vec{v} -s we have

$$(Y_i, Z_{i,\vec{v}} = (X'_{\vec{v}})_{\text{Samp}_i(Y_i)}) \approx_{\varepsilon_0 + 2^{-\beta_{\text{Samp}} k}} (Y_i, Z'_{i,\vec{v}}), \quad (17)$$

with $(Z'_{i,\vec{v}}|Y_i = y_i)$ an $(\ell_i, 0.7\ell_i)$ -source for all y_i .

Analogously to [NZ96, proof of Lemma 17] and the proof of [Lemma 4.4](#), we can use [Equation \(17\)](#) and the fact that $\vec{v} \sim (Y_0, Z_{0,\vec{y}}, \dots, Y_{i-1}, Z_{i-1,\vec{y}})$ is good with probability at least $1 - 2^{-\beta_{\text{Samp}} k}$ to show by induction on the number of blocks that

$$(Y_0, \dots, Y_t, Z) \approx_{\varepsilon_{\text{block}}} (Y_0, \dots, Y_t, Z''), \quad (18)$$

where for every (y_0, \dots, y_t) , $(Z''|Y_0 = y_0, \dots, Y_t = y_t)$ is an exact $((\ell_0, \dots, \ell_t), 0.7)$ -block-source, and $\varepsilon_{\text{block}} = (t+1)(\varepsilon_0 + 2 \cdot 2^{-\beta_{\text{Samp}} k})$.

- (b) If $k < C \log^2(n/\varepsilon_0)$, then for each $i = 0, 1, \dots, t$ let $\text{Cond}_i: \{0, 1\}^{n'} \rightarrow \{0, 1\}^{m_i}$ be the strong RS $(k_i = 0.9\ell_i, k'_i, \varepsilon_0^2)$ -condenser from [Theorem 3.17](#) instantiated with $\alpha = 0.01$. Note that $k_i \leq m_i \leq (1 + \alpha)k_i \leq \ell_i$ and $k'_i \geq (1 - \alpha)m_i \geq 0.99k_i$. Using a fresh seed Y_i of length at most $C'_\alpha \log(n'/\varepsilon_0^2) \leq 2C'_\alpha \log(n'/\varepsilon_0)$ with C'_α the constant guaranteed by [Theorem 3.17](#) for $\alpha = 0.01$, we compute $W_i = \text{Cond}_i(X', Y_i)$ and obtain Z_i by padding W_i to get length exactly ℓ_i , i.e., $Z_i = (W_i, 0^{\ell_i - m_i})$. This is valid since, as already pointed out, $m_i \leq \ell_i$. Later we argue that, despite this padding, Z_i will be statistically close to a source Z'_i with sufficiently large min-entropy rate.

The argument showing that Z_0, Z_1, \dots, Z_t is close to an exact $((\ell_0, \ell_1, \dots, \ell_t), 0.7)$ -block-source conditioned on the seeds Y_1, \dots, Y_t is very similar to that of the previous case. Nevertheless, we do need to check that the choices of ℓ_0, \dots, ℓ_t allow the desired applications of [Theorem 3.17](#).

First, to apply [Theorem 3.17](#) we need that $k_i \geq C_\alpha \log(n'/\varepsilon_0^2)$ for all i , with $C_\alpha > 0$ the constant from [Theorem 3.17](#) for $\alpha = 0.01$. Since $k_i = 0.9\ell_i \geq 0.9\ell_t = k_t$ for all i , it suffices to show that $k_t \geq C_\alpha \log(n'/\varepsilon_0^2)$. Since $n' \leq 2k$ and

$$k_t = 0.9\ell_t = 0.9^t \ell_1 \geq 0.9^2 C_{\text{blocks}} \log(\ell_0/\varepsilon_0) = 0.9^2 C_{\text{blocks}} \log\left(\frac{k}{100 C_{\text{blocks}} \varepsilon_0}\right),$$

it is enough to guarantee that

$$0.9^2 C_{\text{blocks}} \log\left(\frac{k}{100 C_{\text{blocks}} \varepsilon_0}\right) \geq 2C_\alpha \log(2k/\varepsilon_0).$$

If we take C_{blocks} to be large enough so that $C_{\text{blocks}} \geq 5C_\alpha$, the desired inequality holds provided that, say, $k \geq 2(100C_{\text{blocks}})^2$, which is a constant lower bound on k .

Analogously to [Equation \(16\)](#) in the previous case, for any $i = 0, 1, \dots, t$ and an arbitrary fixing $\vec{y} = (y_0, \dots, y_{i-1})$ we have that

$$\mathbf{H}_\infty(X' | Z_{0,\vec{y}} = z_0, \dots, Z_{i-1,\vec{y}} = z_{i-1}) \geq 0.8n'$$

except with probability at most $2^{-\beta k}$ over the choice of z_0, \dots, z_{i-1} , for a sufficiently small constant $\beta > 0$. Therefore, under such a good fixing $\vec{v} = (y_0, z_0, \dots, y_{i-1}, z_{i-1})$, and defining $W_{i,\vec{v}}$ to be W_i conditioned on $(Y_0, Z_0, \dots, Y_{i-1}, Z_{i-1}) = \vec{v}$, [Theorem 3.17](#) guarantees that $(Y_i, W_{i,\vec{v}}) \approx_{\varepsilon_0} (Y_i, W'_{i,\vec{v}})$ with $(W'_{i,\vec{v}} | Y_i = y_i)$ an (m_i, k'_i) -source for all y_i provided that $0.8n' \geq k_i = 0.9\ell_i$. This holds, since

$$0.8n' \geq 0.8k \geq \frac{k}{100C_{\text{blocks}}} = \ell_0 \geq 0.9\ell_i = k_i$$

for any $i = 0, 1, \dots, t$. By padding $W_{i,\vec{v}}$ and $W'_{i,\vec{v}}$ with $0^{\ell_i - m_i}$, we get that $(Y_i, Z_{i,\vec{v}}) \approx_{\varepsilon_0} (Y_i, Z'_{i,\vec{v}})$ with $(Z'_{i,\vec{v}} | Y_i = y_i)$ an (ℓ_i, k'_i) -source for all y_i . Furthermore, the min-entropy of $Z'_{i,\vec{v}}$ satisfies

$$k'_i \geq (1 - \alpha)k_i = 0.99k_i = 0.99 \cdot 0.9\ell_i \geq 0.7\ell_i,$$

and so $(Z'_{i,\vec{v}} | Y_i = y_i)$ is an $(\ell_i, 0.7\ell_i)$ -source for all y_i . As in the previous case, this can be used to conclude by induction that

$$(Y_0, \dots, Y_t, Z) \approx_{\varepsilon'_{\text{block}}} (Y_0, \dots, Y_t, Z''),$$

where for every fixing (y_0, \dots, y_t) we have that $(Z'' | Y_0 = y_0, \dots, Y_t = y_t)$ is an exact $((\ell_0, \dots, \ell_t), 0.7)$ -block-source, and $\varepsilon'_{\text{block}} = (t+1)(\varepsilon_0 + 2^{-\beta k})$. We choose $\beta = \beta_{\text{Samp}}$ as in [Item 2a](#), and so $\varepsilon'_{\text{block}} \leq \varepsilon_{\text{block}}$.

3. We apply block source extraction ([Lemma 2.22](#)) to $Z = (Z_0, Z_1, \dots, Z_t)$. More precisely, let $\text{BExt}: \{0, 1\}^{\ell_0} \times \dots \times \{0, 1\}^{\ell_t} \times \{0, 1\}^{d_t} \rightarrow \{0, 1\}^{m_0}$ be the strong $(k_0, k_1, \dots, k_t, (t+1)\varepsilon_0)$ -block-source extractor with $k_i = 0.7\ell_i$ obtained via [Lemma 2.22](#) as follows. We instantiate Ext_0 with the strong extractor promised by the lemma statement with seed length $d_0 \leq u \cdot \log(\ell_0/\varepsilon_0) = m_1$. For $i \in [t]$, we instantiate $\text{Ext}_i: \{0, 1\}^{\ell_i} \times \{0, 1\}^{d_i} \rightarrow \{0, 1\}^{m_i}$ as the strong $(k_i = 0.7\ell_i, \varepsilon_0)$ -seeded extractor from [Lemma 2.13](#) with seed length $d_i = 2m_i + 4\log(\ell_i/\varepsilon_0) + 8$. We choose the constant C_{blocks} to be large enough so that

$$m_i = \ell_i/C_{\text{blocks}} \leq 0.7\ell_i - 16\log(4/\varepsilon_0) = k_i - 16\log(4/\varepsilon_0),$$

as required by [Lemma 2.13](#). This is possible since by choosing C_{blocks} large enough we have

$$\ell_i \geq \ell_t = C_{\text{blocks}} \cdot m_t \geq 0.9 \cdot C_{\text{blocks}} \log u \cdot \log(\ell_0/\varepsilon_0) \geq 100 \log(4/\varepsilon_0)$$

for all $i \in [t]$, and so $0.7\ell_i - 16\log(4/\varepsilon_0) \geq \ell_i/2$ for all $i \in [t]$. Furthermore, for any $i \geq 2$ the output length m_i of Ext_i satisfies

$$d_i + m_i = 3m_i + 4\log(n/\varepsilon_0) + 8 \geq 2m_{i-1} + 4\log(n/\varepsilon_0) + 8 \geq d_{i-1},$$

where we recall that $m_i = 0.9m_{i-1}$ for $i \geq 2$. Finally, the output length of Ext_1 satisfies $d_1 + m_1 \geq m_1 \geq d_0$, where we recall that d_0 is the seed length of Ext_0 .

Let Y_{BExt} be a fresh seed of length d_t . With the desired upper bound on the seed length d from the lemma's statement in mind, we note that

$$d_t \leq 2m_t + 4\log(\ell_t/\varepsilon_0) + 8 \leq 2\log u \cdot \log(\ell_0/\varepsilon_0) + 4\log(\ell_0/\varepsilon_0) \leq 6\log u \cdot \log(n/\varepsilon_0), \quad (19)$$

since $\ell_0 \leq k \leq n$. By [Lemma 2.22](#), we get that

$$\begin{aligned} (Y_0, \dots, Y_t, Y_{\text{BExt}}, \text{BExt}(Z, Y_{\text{BExt}})) &\approx_{\varepsilon_{\text{block}}} (Y_0, \dots, Y_t, Y_{\text{BExt}}, \text{BExt}(Z'', Y_{\text{BExt}})) \\ &\approx_{(t+1)\varepsilon_0} (Y_0, \dots, Y_t, Y_{\text{BExt}}, U_{m_0}). \end{aligned}$$

Applying the triangle inequality, we conclude that

$$(Y_0, \dots, Y_t, Y_{\text{BExt}}, \text{BExt}(Z, Y_{\text{BExt}})) \approx_{\varepsilon_{\text{block}} + (t+1)\varepsilon_0} (Y_0, \dots, Y_t, Y_{\text{BExt}}, U_{m_0}).$$

We now define our final strong extractor $\text{Ext}: \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^{m_0}$ (recall that we abbreviate $m_0 = m_0(\ell_0)$). Choose our overall seed to be $Y = (Y_{\text{Cond}}, Y_0, Y_1, \dots, Y_t, Y_{\text{BExt}})$ and set $\text{Ext}(X, Y) = \text{BExt}(Z, Y_{\text{BExt}})$. By the discussion above, Ext is a strong (k, ε) -extractor with error (recall that we abbreviate $\varepsilon_0 = \varepsilon_0(\ell_0)$)

$$\varepsilon = 2\varepsilon_0 + \varepsilon_{\text{block}} + (t+1)\varepsilon_0 \leq (2t+4)(\varepsilon_0 + 2 \cdot 2^{-\beta_{\text{Samp}}k}).$$

As discussed above, the $2\varepsilon_0$ accounts for fixing the seed in the condensing step of [Item 1](#) and for assuming that X' is an (n', k') -source under this fixing. Since $t = O(\log u)$, if we pick $C > 0$ to be a sufficiently large constant and c to be smaller than β_{Samp} so that $\varepsilon_0 \geq 2^{-ck_0} \geq 2^{-ck} \geq 2^{-\beta_{\text{Samp}}k}$, we get

$$\varepsilon \leq C \log u \cdot \varepsilon_0.$$

The seed length is

$$d = |Y_{\text{Cond}}| + |Y_{\text{BExt}}| + \sum_{i=0}^t |Y_i| \leq C_{\text{Cond}} \log(n/\varepsilon_0) + d_t + t \cdot \gamma \log(n'/\varepsilon_0) \leq C \log u \cdot \log(n/\varepsilon_0),$$

where $\gamma = \max(C'_\alpha, \gamma_{\text{Samp}})$, provided that C is large enough (again since $t = O(\log u)$), as desired. We used [Equation \(19\)](#) to bound d_t and obtain the last inequality.

Time complexity. It remains to analyze the time complexity of Ext . We proceed by cases:

- If $k \geq C \log^2(n/\varepsilon_0)$ with C a sufficiently large constant, then by [Theorem 3.14](#) we get that [Item 1](#) either takes time $\tilde{O}(n + \sqrt{n} \cdot \log(1/\varepsilon_0)^5)$, or time $\tilde{O}(n)$ after a one-time preprocessing step. Regarding [Item 2](#), each averaging sampler Samp_i from [Lemma 2.20](#) runs in time $\log^2(1/\varepsilon_0) \cdot \text{polylog } n + O(\ell_i \log n)$. This is $\tilde{O}(n)$ when $\varepsilon_0 \geq 2^{-\tilde{O}(\sqrt{n})}$, which is implied by the constraint $k \geq C \log^2(n/\varepsilon_0)$. Since there are $t = O(\log u) = O(\log n)$ blocks, [Item 2](#) runs in $\tilde{O}(n)$ times. [Item 3](#) takes time $T(\ell_0) + t \cdot \tilde{O}(n) = T(\ell_0) + \tilde{O}(n) \leq T(n) + \tilde{O}(n)$, since Ext_0 is computable in time $T(\ell_0)$, each Ext_i from [Lemma 2.13](#) are computable in time $\tilde{O}(n)$, and $\ell_0 \leq n$. Therefore, in this case Ext is computable in overall time $T(n) + \tilde{O}(n + \sqrt{n} \cdot \log(1/\varepsilon_0)^5)$, or time $T(n) + \tilde{O}(n)$ after a preprocessing step.

- If $k < C \log^2(n/\varepsilon_0)$, then **Item 1** takes time $\tilde{O}(n)$ after a preprocessing step. In this case, **Item 2** amounts to $t = O(\log n)$ applications of **Theorem 3.17**, and so runs in time $\tilde{O}(n)$ after a preprocessing step. **Item 3** takes time $T(\ell_0) + \tilde{O}(n)$ after a preprocessing step, and so **Ext** is computable in overall time $T(\ell_0) + \tilde{O}(n) \leq T(n) + \tilde{O}(n)$ after a preprocessing step.

□

Denote by $\log^{(i)}$ the function that iteratively applies \log a total of i times (so $\log^{(1)}n = \log n$, $\log^{(2)}n = \log \log n$, and so on). Denote by \log^* the iterated logarithm. Then, we have the following corollary.

Corollary 4.10. *There exists a constant $C > 0$ such that the following holds. Let n be any positive integer and i any positive integer such that $\log^{(i)}n \geq 6C$. Then, for any $k \leq n$ and any $\varepsilon \geq n^3 \cdot 2^{-k/2^{C \cdot i}}$ there exists a strong (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with seed length $d \leq C \log^{(i)}n \cdot \log(n/\varepsilon)$ and output length $m \geq k/2^{C \cdot i}$. Furthermore,*

1. *if $k \geq 2^{C \cdot i} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$;*
2. *if $k < 2^{C \cdot i} \cdot \log^2(n/\varepsilon)$ or $\varepsilon < 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step.*

Consequently, if we choose i to be the largest integer such that $\log^{(i)}n \geq 6C$ (which satisfies $i \leq \log^*n$) we get a strong (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with seed length $d \leq 6C^2 \log(n/\varepsilon)$ and output length $m \geq k/2^{C \log^*n}$ for any error $\varepsilon \geq n^3 \cdot 2^{-k/2^{C \log^*n}}$. If $k \geq 2^{C \log^*n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq n^3 \cdot 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$. Otherwise, Ext is computable in time $\tilde{O}(n)$ after a preprocessing step.

Proof. This is a consequence of iteratively applying **Lemma 4.9** i times. Note that here part of the relevant condition for the preprocessing is $k \geq 2^{C \cdot i} \log^2(n/\varepsilon)$. The reason behind this is that each application of **Lemma 4.9** reduces the min-entropy requirement by a constant factor.

Let $c, C > 0$ be the constants guaranteed by **Lemma 4.9**. For the first application of the lemma, we take $\text{Ext}_0: \{0, 1\}^n \times \{0, 1\}^{d_0} \rightarrow \{0, 1\}^{m_0}$ to be the strong $(k_0 = 0.7n, \varepsilon_0)$ extractor from **Lemma 2.13** with $m_0 = k_0/20$ and $\varepsilon_0 \geq 2^{-ck_0/100}$ to be defined later. The corresponding seed length is $d_0 \leq 2m_0 + 4 \log(n/\varepsilon_0) + 4$, which satisfies $d_0 \leq k_0$, and so the initial value of u is $u_0 = d_0/\log(n/\varepsilon_0) \leq k_0$. Denote by Ext_1 the resulting strong seeded extractor. In the second application of **Lemma 4.9**, we instantiate Ext_0 with Ext_1 instead to obtain a new strong seeded extractor Ext_2 , and so on. For each $j \in [i]$, we obtain a family of strong (k, ε_j) -seeded extractors $\text{Ext}_j: \{0, 1\}^n \times \{0, 1\}^{d_j} \rightarrow \{0, 1\}^{m_j}$ parameterized by k with output length $m_j = m_{j-1}(ck)$, error

$$\varepsilon_j = C \log u_{j-1} \cdot \varepsilon_{j-1}(ck)$$

and seed length

$$d_j = C \log u_{j-1} \cdot \log(n/\varepsilon_{j-1}(ck)) = C \log u_{j-1} \cdot \log\left(\frac{n \cdot C \log u_{j-1}}{\varepsilon_j}\right),$$

where

$$\begin{aligned} u_j &= \frac{d_j}{\log(n/\varepsilon_j)} \\ &= C \log u_{j-1} \cdot \left(1 + \frac{\log C}{\log(n/\varepsilon_j)} + \frac{\log \log u_{j-1}}{\log(n/\varepsilon_j)}\right) \end{aligned}$$

$$\begin{aligned}
&\leq C \log u_{j-1} \cdot \left(1 + \frac{\log C}{\log n} + \frac{\log \log u_{j-1}}{\log n}\right) \\
&\leq 3C \log u_{j-1}.
\end{aligned}$$

The last inequality uses the fact that $u_{j-1} \leq u_0 \leq n$ for all j .

Recall that from the corollary statement that i is such that $\log^{(i)} n \geq 6C$. We show by induction that $u_j \leq 3C \log^{(j)} n + 3C \log(6C)$ for all $j = 0, \dots, i$. This is immediate for the base case $j = 0$, since $u_0 \leq k_0 \leq n$. For the induction step, note that

$$\begin{aligned}
u_{j+1} &\leq 3C \log u_j \leq 3C \log(3C \log^{(j)} n + 3C \log(6C)) \\
&\leq 3C \log(2 \cdot 3C \log^{(j)} n) = 3C \log^{(j+1)} n + 3C \log(6C),
\end{aligned}$$

as desired. This implies that

$$d_j = u_j \cdot \log(n/\varepsilon_j) \leq 6C \log^{(j)} n \cdot \log(n/\varepsilon_j)$$

and

$$\varepsilon_j = C \log u_{j-1} \cdot \varepsilon_{j-1}(ck) \leq (6C)^j \left(\prod_{j'=0}^{j-1} \log^{(j')} n \right) \cdot \varepsilon_0(c^j k)$$

for all $j \in [i]$. We may assume that C is large enough that $\log a \leq \sqrt{a}$ for all $a \geq C$, in which case $\prod_{j'=0}^{j-1} \log^{(j')} n \leq \prod_{j'=0}^{j-1} n^{2^{-j'}} \leq n^2$ since $\log^{(j')} n \geq C$ for all $j' \leq i$ by hypothesis. Therefore, we obtain final output length

$$m_i = m_0(c^i k) = k/2^{O(i)},$$

final error ε_i satisfying

$$\varepsilon_0(ck) \leq \varepsilon_i \leq (6C)^i \cdot n^2 \cdot \varepsilon_0(c^i k) \leq n^3 \cdot \varepsilon_0(c^i k),$$

where the last inequality uses that $\log^{(i)} n \geq 6C$, and final seed length

$$d_i \leq 6C \log^{(i)} n \cdot \log(n/\varepsilon_i).$$

We now instantiate $\varepsilon_0(c^i k) = \varepsilon/n^3$. Note that $\varepsilon_0(c^i k) \geq 2^{-0.7c^{i+1}k/100}$ as required for the choice of Ext_0 above so long as $\varepsilon \geq n^3 \cdot 2^{-0.7c^{i+1}k}$, which holds by the corollary's hypothesis if C is a large enough constant. With this choice of $\varepsilon_0(c^i k)$ we get final error $\varepsilon_i \leq n^3 \cdot \varepsilon_0(c^i k) = \varepsilon$. In fact, we can make ε_i larger so that $\varepsilon_i = \varepsilon$, in which case the final seed length satisfies

$$d_i \leq 6C \log^{(i)} n \cdot \log(n/\varepsilon),$$

as desired.

Time complexity. Finally, we discuss the time complexity of Ext . Note that the initial choice for Ext_0 is computable in time $\tilde{O}(n_0)$. Therefore, if $k \geq 2^{C \cdot i} \log^2(n/\varepsilon)$ then each application of [Lemma 4.9](#) runs in time $\tilde{O}(n + \sqrt{n} \log(1/\varepsilon_0(c^i k))^5)$. This uses the fact that the error increases in each application of [Lemma 4.9](#). Since $\varepsilon_0(c^i k) \geq \varepsilon/n^3$, then each application of [Lemma 4.9](#) runs in time $\tilde{O}(n)$ without preprocessing when $\varepsilon \geq 2^{-Cn^{0.1}}$. Otherwise, the condition in [Item 2](#) of [Lemma 4.9](#) holds and so Ext is computable in time $\tilde{O}(n)$ after a preprocessing step, since we always have $u \leq n$ in each application of the lemma. \square

Remark 4.11 (the preprocessing in [Corollary 4.10](#)). For the sake of readability we were not explicit about the precise preprocessing in the statement of Case 2 of [Corollary 4.10](#). We expand on that now. [Corollary 4.10](#) recursively invokes [Lemma 4.9](#) i times. All these recursive calls use the same type of condenser (either the KT condenser or the RS condenser), depending on the initial choices of k and ε . Therefore:

- If $k \geq 2^{C \cdot i} \cdot \log^2(n/\varepsilon)$ and $\varepsilon < 2^{-Cn^{0.1}}$, then the preprocessing corresponds to the preprocessing for i calls of the KT condenser – for example, generating i primes $q \leq \text{poly}(n/\varepsilon)$.
- If $k < 2^{C \cdot i} \cdot \log^2(n/\varepsilon)$ and $\varepsilon < 2^{-Cn^{0.1}}$, then each invocation of [Lemma 4.9](#) requires one preprocessing step for the RS condenser call in [Item 1](#), and, naively, t preprocessing steps for the $t = O(\log n)$ RS condenser calls in [Item 2b](#). We can further improve this by noting that all t calls of [Theorem 3.17](#) in [Item 2b](#) use the same input length, error, and α , and so we only need to run one preprocessing step that suffices for all t calls simultaneously. So, overall, the preprocessing corresponds to the preprocessing for $2i$ calls of the RS condenser – generating $2i$ primes $q \leq \text{poly}(n/\varepsilon)$ along with a primitive element for each \mathbb{F}_q .

Note that [Corollary 4.10](#) only guarantees output length $k/2^{C^i}$ for each i . In particular, when $i = \log^* n$ we get output length $k/2^{C \log^* n}$. This is slightly sublinear, and we would like to aim for output length ck for some constant $c > 0$. To obtain our final theorem, we use block source extraction to increase the output length of the extractor from [Corollary 4.10](#), following a strategy of Zuckerman [[Zuc97](#)].

Theorem 4.12. *There exist constants $c, C > 0$ such that the following holds. For any integers n and $k \leq n$ and any $\varepsilon \geq Cn^3 \cdot 2^{-k/\log k}$ there exists a strong (k, ε) -seeded extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ with seed length $d \leq C \log(n/\varepsilon)$ and output length $m \geq ck$. Furthermore,*

1. *if $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$;*
2. *if $k < 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ or $\varepsilon < 2^{-Cn^{0.1}}$, then Ext is computable in time $\tilde{O}(n)$ after a preprocessing step.*

Proof. We begin by providing an informal discussion of the proof. On input an arbitrary (n, k) -source X , we begin by applying a fast condenser to X to obtain another source X' with high min-entropy rate. Then, we split X' in half to obtain a block source (X_1, X_2) with two blocks. Then, we perform block source extraction on (X_1, X_2) . More concretely, we apply the extractor obtained by instantiating [Corollary 4.10](#) with $i = \log^* n$ to X_2 , and then use its output as the seed to extract from X_1 using the extractor obtained by instantiating [Corollary 4.10](#) with $i = 2$ (which has output length $\Omega(k)$).

More formally, define $\varepsilon' = \varepsilon/6$ and let X be an arbitrary (n, k) -source. The extractor Ext behaves as follows on input X :

1. Apply a strong $(k, k', (\varepsilon')^2)$ -condenser $\text{Cond}: \{0, 1\}^n \times \{0, 1\}^{d_{\text{Cond}}} \rightarrow \{0, 1\}^{n'}$ to X , with output min-entropy $k' \geq 0.95n'$ and seed length $d_{\text{Cond}} = C_{\text{Cond}} \log(n/\varepsilon')$. If $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$, we instantiate Cond with the KT strong (k, k', ε') -condenser ([Theorem 3.14](#) instantiated with $\alpha = 0.05$). Otherwise, we instantiate Cond with the RS strong (k, k', ε') -condenser ([Theorem 3.17](#) instantiated with $\alpha = 0.025$). By the second part of either [Theorem 3.14](#) or [Theorem 3.17](#), we get that with probability at least $1 - \varepsilon'$ over the choice of the seed y we obtain an output X' that is ε' -close to an (n', k') -source with $k' \geq 0.95n'$. As in previous arguments, we work under such a good fixing of y from here onwards and account for the probability ε' of selecting a bad seed in the final extractor error later on.

2. Let (X_1, X_2) correspond to the first two blocks of $\lfloor n'/2 \rfloor$ bits of X' . Choose the constant $c > 0$ in the theorem statement small enough so that $\log(1/\varepsilon') \leq \log(1/\varepsilon) + 3 \leq ck + 3 \leq 0.05k - 1$, which means that $\lfloor n'/2 \rfloor - 0.05k - \log(1/\varepsilon') \geq 0.4n'$. Then, combining [Item 1](#) with [Lemma 2.23](#) (instantiated with $t = 2$, $\Delta = 0.05k$, and $\varepsilon = \varepsilon'$) via the triangle inequality, (X_1, X_2) is $3\varepsilon'$ -close to an exact $((n_1 = \lfloor n'/2 \rfloor, n_2 = \lfloor n'/2 \rfloor), 0.8)$ -block-source.
3. Apply block source extraction to (X_1, X_2) . More precisely, let $\text{Ext}_1: \{0, 1\}^{n_1} \times \{0, 1\}^{d_1} \rightarrow \{0, 1\}^{m_1}$ be the strong $(k_1 = 0.8n_1, \varepsilon_1 = \varepsilon')$ -seeded extractor from [Corollary 4.10](#) instantiated with $i = 2$ and $n_1 = \lfloor n'/2 \rfloor$, which requires $\varepsilon_1 = \varepsilon' \geq n_1^3 \cdot 2^{-c_1 k_1}$ and guarantees $d_1 \leq C_1 \log \log k_1 \cdot \log(n'/\varepsilon)$ and $m_1 \geq c_1 k_1$, for constants $c_1, C_1 > 0$ guaranteed by [Corollary 4.10](#). Furthermore, let $\text{Ext}_2: \{0, 1\}^{n_2} \times \{0, 1\}^{d_2} \rightarrow \{0, 1\}^{m_2}$ be the strong $(k_2 = 0.8n_2, \varepsilon_2 = \varepsilon')$ -seeded extractor from the “Consequently” part of [Corollary 4.10](#) and $n_2 = \lfloor n'/2 \rfloor$, which requires $\varepsilon_2 \geq n_2^3 \cdot 2^{-k_2/2^{C_2 \log^* k_2}}$ and guarantees $d_2 \leq C_2 \log(n'/\varepsilon)$ and $m_2 \geq k_2/2^{C_2 \log^* k_2}$, for a constant $C_2 > 0$ guaranteed by [Corollary 4.10](#). This choice of parameters ensures that $m_2 \geq d_1$ and is valid by the lower bound on ε in the theorem statement, recalling that $\varepsilon' = \varepsilon/6$. Indeed, since $k \geq k_1 = k_2 \geq 0.4n'$, to see that $m_2 \geq d_1$ it suffices to check that

$$\frac{0.4k}{2^{C_2 \log^* k}} \geq d_1 = C_1 \log \log k \cdot \log(n'/\varepsilon_1).$$

Since $\varepsilon_1 = \varepsilon' = \varepsilon/6$ and $\log(n'/\varepsilon_1) = O(\log(k/\varepsilon')) = O(\log k + k/\log k) = O(k/\log k)$, it is enough that

$$k \geq C'_1 \cdot 2^{C_2 \log^* k} \log \log k \cdot \frac{k}{\log k}$$

for a sufficiently large constant $C'_1 > 0$, which holds whenever k is larger than some appropriate absolute constant. Instantiating [Lemma 2.22](#) with Ext_1 and Ext_2 above yields a strong $(k_1 = 0.8n_1, k_2 = 0.8n_2, \varepsilon_1 + \varepsilon_2)$ -block-source extractor $\text{BExt}: \{0, 1\}^{n_1} \times \{0, 1\}^{n_2} \times \{0, 1\}^{d_2} \rightarrow \{0, 1\}^{m_1}$.

Since X' is $3\varepsilon'$ -close to an exact $(n_1, n_2, 0.8)$ -block source, we conclude that

$$(Y_{\text{BExt}}, \text{BExt}(X', Y_{\text{BExt}})) \approx_{3\varepsilon' + \varepsilon_1 + \varepsilon_2} U_{d_2 + m_1}. \quad (20)$$

We define the output of our final strong extractor $\text{Ext}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{m_1}$ to be $\text{BExt}(X', Y_{\text{BExt}})$. Since $\varepsilon_1 = \varepsilon_2 = \varepsilon'$, [Equation \(20\)](#) implies that

$$(Y_{\text{Cond}}, Y_{\text{BExt}}, \text{Ext}(X, Y_{\text{Cond}}, Y_{\text{BExt}})) \approx_{5\varepsilon'} U_{d + m_1}.$$

This means that Ext is a strong $(k, \varepsilon' + 5\varepsilon' = \varepsilon)$ -seeded extractor with seed length $d = |Y_{\text{Cond}}| + |Y_{\text{BExt}}| = O(\log(n/\varepsilon))$ and output length $m_1 \geq c_1 k_1 \geq c'_1 k$ for an absolute constant $c'_1 > 0$, where one of the ε' terms in the error comes from fixing the seed in the condensing step of [Item 1](#).

Time complexity. Finally, we analyze the time complexity of Ext . If $k \geq 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$ and $\varepsilon \geq 2^{-Cn^{0.1}}$, then [Item 1](#) runs in time $\tilde{O}(n)$. In [Item 3](#), Ext_1 and Ext_2 are both computable in time $\tilde{O}(n)$ under these lower bounds on k and ε , and thus so is BExt . We conclude that Ext runs in time $\tilde{O}(n)$.

Otherwise, if $k < 2^{C \log^* n} \cdot \log^2(n/\varepsilon)$, then [Item 1](#) runs in time $\tilde{O}(n)$ after a preprocessing step. And if $\varepsilon < 2^{-Cn^{0.1}}$ then Ext_1 and Ext_2 in [Item 3](#) run in time $\tilde{O}(n)$ after a preprocessing step. Therefore, overall, Ext runs in time $\tilde{O}(n)$ after a preprocessing step in this case. \square

5 A Faster Instantiation of Trevisan's Extractor

We first recall Trevisan's extractor [Tre01, RRV02], $\text{Tre}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$, set to some designated error $\varepsilon > 0$. We will need the notion of weak designs, due to Raz, Reingold, and Vadhan [RRV02].

Definition 5.1 (weak design). *A collection of sets $S_1, \dots, S_m \subseteq [d]$ is an (ℓ, ρ) -weak design if for all $i \in [m]$ we have $|S_i| = \ell$ and*

$$\sum_{j < i} 2^{|S_i \cap S_j|} \leq \rho(m-1).$$

We will also need a δ -balanced code $\mathcal{C}: \{0, 1\}^n \rightarrow \{0, 1\}^{\bar{n}}$. The parameters of the weak design affect the extractor's parameters and can be set in a couple of different ways. The parameter ℓ is set to be $\log \bar{n}$, typically ρ is chosen according to m , ε , and the desired entropy k , and then d is chosen as a function of ℓ , m , and ρ according to the weak design (see [RRV02]). Given $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, Trevisan's extractor outputs

$$\text{Tre}(x, y) = (\bar{x}|_{y_{S_1}}, \dots, \bar{x}|_{y_{S_m}}), \quad (21)$$

where we denote $\bar{x} = \mathcal{C}(x)$ and interpret each length- $\log \bar{n}$ bit-string y_{S_i} as a location in $[\bar{n}]$. For the runtime analysis, it will be important to recall that δ is set to be $\frac{\varepsilon}{cm}$ for some universal constant c .

Theorem 5.2. *Trevisan's extractor of Equation (21), set to extract m bits with any error $\varepsilon > 0$, is computable in time $\tilde{O}(n + m \log(1/\varepsilon))$.*

On a RAM in the logarithmic cost model, Trevisan's extractor is computable in time $O(n) + m \log(1/\varepsilon) \cdot \text{polylog}(n)$ with a preprocessing time of $\tilde{O}(m \log(n/\varepsilon))$. In particular, there exists a universal constant c , such that whenever $m \leq \frac{n}{\log^c(n/\varepsilon)}$, it runs in time $O(n)$, without the need for a separate preprocessing step.

Proof. Looking at Equation (21), note that we only need to compute m coordinates of $\mathcal{C}(x)$. To compute those m coordinates, y_{S_1}, \dots, y_{S_m} , we first need to compute the weak design itself. Note that this can be seen as a preprocessing step, since it only depends on the parameters of the extractor, and not on x or y . We will use the following result.

Claim 5.3 ([FYEC25], Section A.5). *For every $\ell, m \in \mathbb{N}$ and $\rho > 1$, there exists an (ℓ, ρ) -weak design $S_1, \dots, S_m \subseteq [d]$ with $d = O(\frac{\ell^2}{\log \rho})$, computable in time $\tilde{O}(m\ell)$.*

Once we have our preprocessing step, we are left with computing the code. By Corollary 3.3, we can choose \bar{n} so that $n/\bar{n} = \delta^c$ for some universal constant c , and so $\bar{n} = n \cdot \text{poly}(m, 1/\varepsilon)$ and $\ell = \log \bar{n} = O(\log(n/\varepsilon))$. Generating the design can then be done in time $\tilde{O}(m \log(n/\varepsilon))$. Now, Corollary 3.3 tells us that any m bits of $\mathcal{C}(x)$ can be computed in time

$$\tilde{O}(n) + m \log(1/\delta) \cdot \text{polylog}(n) = \tilde{O}(n + m \log(1/\varepsilon)).$$

On a RAM in the logarithmic cost model, we can use the variant of \mathcal{C} that uses Spielman's code as a base code (see Remark 3.4) and get a runtime of $O(n) + m \log(1/\varepsilon) \cdot \text{polylog}(n)$. This gives a truly linear time construction whenever m is at most $\frac{n}{\log(1/\varepsilon) \text{polylog}(n)}$. \square

We conclude by noting that there is a natural setting of parameters under which Trevisan's extractor gives logarithmic seed and linear (or near-linear) time. When $m = k^{\Omega(1)}$, the parameters can be set so that $d = O\left(\frac{\log^2(n/\varepsilon)}{\log k}\right)$. We thus have the following corollary.

Corollary 5.4. *For every $n \in \mathbb{N}$, any constant $c > 1$, and any constants $\alpha, \beta \in (0, 1)$, Trevisan’s extractor $\text{Tre}: \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ can be instantiated as a ($k = n^\alpha, \varepsilon = n^{-c}$) extractor with $d = O(\log n)$, $m = k^\beta$, and given $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^d$, $\text{Tre}(x, y)$ is computable in time $\tilde{O}(n)$ (or $O(n)$ time, depending on the model).*

References

- [ACG⁺22] Omar Alrabiah, Eshan Chattopadhyay, Jesse Goodman, Xin Li, and João Ribeiro. Low-degree polynomials extract from local sources. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 10:1–10:20, 2022.
- [AGMR25] Omar Alrabiah, Jesse Goodman, Jonathan Mosheiff, and João Ribeiro. Low-degree polynomials are good extractors. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. To appear. Preprint available at <https://eccc.weizmann.ac.il/report/2024/093/>.
- [AKO⁺22] Divesh Aggarwal, Bhavana Kanukurthi, Sai Lakshmi Bhavana Obbattu, Maciej Obremski, and Sruthi Sekar. Rate one-third non-malleable codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 1364–1377, New York, NY, USA, 2022. Association for Computing Machinery.
- [AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, pages 781–793, 2004.
- [Alo21] Noga Alon. Explicit expanders of every degree and size. *Combinatorica*, pages 1–17, 2021.
- [Bac88] Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, 1988.
- [BBCM95] Charles H. Bennett, Gilles Brassard, Claude Crépeau, and Ueli M. Maurer. Generalized privacy amplification. *IEEE Transactions on Information Theory*, 41(6):1915–1923, 1995.
- [BBR88] Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM Journal on Computing*, 17(2):210–229, 1988.
- [BG13] Andrej Bogdanov and Siyao Guo. Sparse extractor families for all the entropy. In *Innovations in Theoretical Computer Science (ITCS)*, pages 553–560. ACM, 2013.
- [BGW19] Marshall Ball, Siyao Guo, and Daniel Wichs. Non-malleable codes for decision trees. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 413–434, Cham, 2019. Springer International Publishing.
- [BIW06] Boaz Barak, Russell Impagliazzo, and Avi Wigderson. Extracting randomness using few independent sources. *SIAM Journal on Computing*, 36(4):1095–1118, 2006.
- [BM74] Allan Borodin and Robert Moenck. Fast modular transforms. *Journal of Computer and System Sciences*, 8(3):366–386, 1974.

- [Bog12] Andrej Bogdanov. Topics in (and out) the theory of computing: Lecture notes. <https://andrejb.net/csc5060/notes/12L12.pdf>, 2012. [Online; accessed October 2024].
- [BRST02] Ziv Bar-Yossef, Omer Reingold, Ronen Shaltiel, and Luca Trevisan. Streaming computation of combinatorial objects. In *Annual Conference on Computational Complexity (CCC)*, pages 165–174. IEEE, 2002.
- [CEG95] Ran Canetti, Guy Even, and Oded Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995.
- [CG88] Benny Chor and Oded Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, 1988.
- [CGL20] Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Nonmalleable extractors and codes, with their many tampered extensions. *SIAM Journal on Computing*, 49(5):999–1040, 2020.
- [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)*, pages 37:1–37:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
- [CRSW13] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. *SIAM Journal on Computing*, 42(3):1030–1050, 2013.
- [CT65] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.
- [CW24] Kuan Cheng and Ruiyang Wu. Randomness extractors in AC⁰ and NC¹: Optimal up to constant factors. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 69:1–69:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [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, 2013.
- [DMOZ22] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. Nearly optimal pseudorandomness from hardness. *J. ACM*, 69(6), November 2022.
- [DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM Journal on Computing*, 38(1):97–139, 2008.
- [DPVR12] Anindya De, Christopher Portmann, Thomas Vidick, and Renato Renner. Trevisan’s extractor in the presence of quantum side information. *SIAM Journal on Computing*, 41(4):915–940, 2012.
- [DT23] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Computational Complexity Conference (CCC)*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

- [DW08] Zeev Dvir and Avi Wigderson. Kakeya sets, new mergers and old extractors. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 625–633, 2008.
- [FWE⁺23] Cameron Foreman, Sherilyn Wright, Alec Edgington, Mario Berta, and Florian J. Curchod. Practical randomness amplification and privatisation with implementations on quantum computers. *Quantum*, 7:969, March 2023.
- [FYEC25] Cameron Foreman, Richie Yeung, Alec Edgington, and Florian J. Curchod. Cryptomite: A versatile and user-friendly library of randomness extractors. *Quantum*, 9:1584, January 2025.
- [GGH⁺24] Alexander Golovnev, Zeyu Guo, Pooya Hatami, Satyajeet Nagargoje, and Chao Yan. Hilbert functions and low-degree randomness extractors. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 41:1–41:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [Gil98] David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.
- [GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), jul 2009.
- [GVW15] Oded Goldreich, Emanuele Viola, and Avi Wigderson. On randomness extraction in AC0. In *Conference on Computational Complexity (CCC)*, page 601–668. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.
- [Hea08] Alexander D. Healy. Randomness-efficient sampling within NC¹. *Computational Complexity*, 17:3–37, 2008.
- [HH15] Jan Håzla and Thomas Holenstein. Upper tail estimates with combinatorial proofs. In *Symposium on Theoretical Aspects of Computer Science (STACS)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- [HIV22] Xiangui Huang, Peter Ivanov, and Emanuele Viola. Affine extractors and AC0-parity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [HT16] Masahito Hayashi and Toyohiro Tsurumaru. More efficient privacy amplification with less random seeds via dual universal hash function. *IEEE Transactions on Information Theory*, 62(4):2213–2232, 2016.
- [HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 672–683. Springer, 2006.
- [HvdH19] David Harvey and Joris van der Hoeven. Faster polynomial multiplication over finite fields using cyclotomic coefficient rings. *Journal of Complexity*, 54:101404, 2019.
- [HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563 – 617, 2021.

- [HvdH22] David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time $O(n \log n)$. *J. ACM*, 69(2):1–40, 2022.
- [Jus72] Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.
- [Kal03] Adam Tauman Kalai. Generating random factored numbers, easily. *Journal of Cryptology*, 16(4):287–289, 2003.
- [KT22] Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [Len02] H. W. Lenstra. Primality testing with Gaussian periods. In *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science*, pages 1–1. Springer Berlin Heidelberg, 2002.
- [LRVW03] Chi-Jen Lu, Omer Reingold, Salil Vadhan, and Avi Wigderson. Extractors: Optimal up to constant factors. In *Symposium on Theory of Computing (STOC)*, pages 602–611. ACM, 2003.
- [Lu02] Chi-Jen Lu. Hyper-encryption against space-bounded adversaries from on-line strong extractors. In *Advances in Cryptology — CRYPTO*, pages 257–271. Springer, 2002.
- [MPS12] Wolfgang Maurer, Christopher Portmann, and Volkher B. Scholz. A modular framework for randomness extraction based on Trevisan’s construction. *arXiv e-prints*, December 2012. <https://arxiv.org/abs/1212.0520>.
- [MRRR14] Raghu Meka, Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Fast pseudorandomness for independence and load balancing. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 859–870. Springer, 2014.
- [MW97] Ueli Maurer and Stefan Wolf. Privacy amplification secure against active adversaries. In *Advances in Cryptology — CRYPTO*, pages 307–321. Springer, 1997.
- [Nar14] Shyam Narayanan. Improving the speed and accuracy of the Miller-Rabin primality test, 2014. Available at <https://math.mit.edu/research/highschool/primes/materials/2014/Narayanan.pdf>.
- [NT99] Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1):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.
- [QWW21] Willy Quach, Brent Waters, and Daniel Wichs. Targeted lossy functions and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 424–453, Cham, 2021. Springer International Publishing.
- [RRV02] Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in Trevisan’s extractors. *Journal of Computer and System Sciences*, 65(1):97–128, 2002.

- [RSW06] Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. *SIAM Journal on Computing*, 35(5):1185–1209, 2006.
- [RT00] Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, 2000.
- [Sch77] Arnold Schönhage. Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Informatica*, 7(4):395–398, 1977.
- [Sho90] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54(189):435–447, 1990.
- [Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005. Available at <https://shoup.net/ntb/>.
- [Spi96] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.
- [SU05] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.
- [SV19] Akshayaram Srinivasan and Prashant Nalini Vasudevan. Leakage resilient secret sharing and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 480–509, Cham, 2019. Springer International Publishing.
- [SZ99] Aravind Srinivasan and David Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28(4):1433–1459, 1999.
- [Ta-02] Amnon Ta-Shma. Almost optimal dispersers. *Combinatorica*, 22(1):123–145, 2002.
- [Ta-17] Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Symposium on Theory of Computing (STOC)*, page 238–251. ACM, 2017.
- [Tre01] Luca Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, jul 2001.
- [TSSR11] Marco Tomamichel, Christian Schaffner, Adam Smith, and Renato Renner. Leftover hashing against quantum side information. *IEEE Transactions on Information Theory*, 57(8):5524–5535, 2011.
- [TU12] Amnon Ta-Shma and Christopher Umans. Better condensers and new extractors from Parvaresh-Vardy codes. In *Conference on Computational Complexity (CCC)*, pages 309–315. IEEE, 2012.
- [TZS06] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed–Muller codes. *Journal of Computer and System Sciences*, 72(5):786–812, 2006.
- [Vad04] Salil Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *Journal of Cryptology*, 17:43–77, 2004.
- [Vad12] Salil Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2012.

- [vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2013.
- [WZ99] Avi Wigderson and David Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. *Combinatorica*, 19(1):125–138, 1999.
- [XZ25] Zhiyang Xun and David Zuckerman. Near-optimal averaging samplers and matrix samplers. In *40th Computational Complexity Conference (CCC)*, pages 6:1–6:28. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.
- [Zuc96] David Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16:367–391, 1996.
- [Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. *Random Structures & Algorithms*, 11(4):345–367, 1997.
- [Zuc07] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.