REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
04-06-2022	Final Report	19-Jan-2018 - 18-Jan-2022

4. TITLE AND SUBTITLE
Final Report: Native DNA-Based Data Storage and Computing

5a. CONTRACT NUMBER
W911NF-18-2-0032

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
111111

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
University of Illinois - Urbana - Champaign
c/o Office of Sponsored Programs
1901 S. First Street, Suite A
Champaign, IL 61820 -7406

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
73125-CH-DRP.2

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Olgica Milenkovic
UU	UU	UU	UU		19b. TELEPHONE NUMBER 217-244-7358

Standard Form 298 (Rev 8/98)
Prescribed by ANSI Std. Z39.18

RPPR Final Report
as of 17-Jun-2022

Agency Code: 21XD

Proposal Number: 73125CHDRP **Agreement Number: W911NF-18-2-0032**
INVESTIGATOR(S):

Name: Olgica Milenkovic
Email: milenkov@illinois.edu
Phone Number: 2172447358
Principal: Y

Organization: **University of Illinois - Urbana - Champaign (UIUC)**
Address: c/o Office of Sponsored Programs, Champaign, IL 618207406
Country: USA
DUNS Number: 041544081 EIN: 376000511

Report Date: 18-Apr-2022 Date Received: 04-Jun-2022
Final Report for Period Beginning 19-Jan-2018 and Ending 18-Jan-2022
Title: Native DNA-Based Data Storage and Computing
Begin Performance Period: 19-Jan-2018 **End Performance Period:** 18-Jan-2022
Report Term: 0-Other

Submitted By: Olgica Milenkovic Email: milenkov@illinois.edu
Phone: (217) 244-7358

Distribution Statement: 1-Approved for public release; distribution is unlimited.

STEM Degrees: **STEM Participants:**

Major Goals: 1. Develop new DNA-based data storage system that use native DNA, or combinations of native and synthetic DNA for massive DNA storage of heterogeneous data in order to enable parallel writing, fast random access and single-bit random access, reduce the cost and latency of the reading/writing process.
2. Enable multidimensional storage both in the DNA content and backbone via a combination of synthetic DNA and superimposed nicks for metadata. Extend the capability of molecular recorders to allow for easy privacy-preserving information erasure and rewriting.
3. Develop new coding and machine learning methods specialized for the DNA storage media that increase the reliability of the recording system.
4. Develop watermarking and other authentication protocols for DNA-encoded data da and implement multilevel flash memory storage systems via enzymatic synthesis of length-modulated tails at the sites of nicks.
5. Develop the first known parallel, in-memory computing paradigm that operates directly on data stored in DNA. In particular, demonstrate massively parallel single instruction set register incrementing, Rule 110 and sorting in memory on nicked data.

Accomplishments: 1. Developed the theoretical underpinnings of the first DNA-based data storage paradigm that uses native (bacterial) DNA as the storage media (Part I of the report).
2. Experimentally implemented the system by encoding both text and imaging data. The key idea is to record information in the form of nicks in the sugar-phosphate backbone, with a nick representing either 1,2 (sense, antisense) or 0 (no nick) (Part I of the report).
3. Optimized a new class of nicking enzymes (PfAgo) to operate in parallel fashion with negligible off-target activity (Part I of the report).
4. Performed the first nanopore sequencing experiments for the purpose of detecting nicks, toeholds and tails (Part I of the report).
5. Enabled the first bitwise random access protocol for DNA-based data storage (Part I of the report).
6. Developed new coding methods for error-correction (e.g., coded trace reconstruction), constrained coding (e.g., set codes) and image inpainting methods for data reconstruction.
7. Devised the first in-memory computing method that operates directly on information stored in DNA nicks. The paradigm, termed SIMDNA, combines strand displacement with unique in memory protocols for register incrementing, Rule 110 and sorting.
8. Work in progress includes writing out the results for DNA flash memories.

Training Opportunities: Nothing to Report

RPPR Final Report
as of 17-Jun-2022

Results Dissemination: STORAGE THRUST

1. Chao Pan, S Kasra Tabatabaei, SM Hossein Tabatabaei Yazdi, Alvaro G Hernandez, Charles M Schroeder, Olgica Milenkovic, “Rewritable Two-Dimensional DNA-Based Data Storage with Machine Learning Reconstruction,” to appear in Nature Communications (accepted in 2020, delayed due to copyright issues for two images), 2022.
2. Nagendra Athreya, Olgica Milenkovic, Jean-Pierre Leburton, “Interaction dynamics and site-specific electronic recognition of DNA-nicks with 2D solid-state nanopores,” npj 2D Materials and Applications, vol. 4, no. 1, pp. 1-8, Nature Publishing Group, 2020.
3. M Cheraghchi, R Gabrys, O Milenkovic, J Ribeiro, “Coded trace reconstruction,” IEEE Transactions on Information Theory 66 (10), 6084-6103, 2020.
4. S Kasra Tabatabaei, Boya Wang, Nagendra Bala Murali Athreya, Behnam Enghiad, Alvaro Gonzalo Hernandez, Christopher J Fields, Jean-Pierre Leburton, David Soloveichik, Huimin Zhao, Olgica Milenkovic, “DNA punch cards for storing data on native DNA sequences via enzymatic nicking,” Nature Communications, vol. 11, no. 1, pp. 1-10, 2020. (NOTABLE COVERAGE: Nature, New Scientist, Scientific American, Editor’s Highlights)
5. Abhishek Agarwal, Olgica Milenkovic, Srilakshmi Pattabiraman, Joao Ribeiro, “Group Testing with Runlength Constraints for Topological Molecular Storage,” International Symposium on Information Theory, ISIT 2020.
6. Ryan Gabrys, Hoang Dau, Charles Colbourn, Olgica Milenkovic, “Set-codes with small intersections and small discrepancies,” SIAM Journal on Discrete Mathematics 34 (2), 1148-1171 pp. 132-137, 2019.
7. Ke Liu, Chao Pan, Alexandre Kuhn, Adrian Pascal Nievergelt, Georg E Fantner, Olgica Milenkovic, Aleksandra Radenovic, “Detecting topological variations of DNA at single-molecule level,” Nature Communications, vol. 12, no. 1, pp. 1-9, 2019.

COMPUTING THRUST

1. B Wang, C Chalk, D Soloveichik, “SIMD||DNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades,” DNA25: International Conference on DNA Computing and Molecular Programming, Lecture Notes in Computer Science 11648: 219-235 (2019).
2. B Wang, S Wang, C Chalk, A Ellington, D Soloveichik, "Parallel molecular computation on digital data stored in DNA," in preparation.
3. Tonglin Chen, Arnav Solanki, and Marc Riedel, “Parallel Pairwise Operations on Data Stored in DNA: Sorting, Shifting, and Searching” LIPI DNA Computing, Vol. 205, pp. 11:1 – 11:21, 2021.
4. Amlan Ganguly, Sergi Abadal, Ishan Thakkar, Natalie Enright Jerger, Marc Riedel, Masoud Babaie, Rajeev Balasubramonian, Abu Sebastian, Sudeep Pasricha, and Baris Taskin “Interconnects for DNA, Quantum, In-Memory and Optical Computing” IEEE Micro, doi: 10.1109/MM.2022.3150684, Feb. 14, 2022.

5. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Performing Math from Truth Tables with DNA”, under review, PloS One, 2022.
6. Patrick Holec, Weikang Qian, Marc Riedel, and Ivo Rosenberg, “Conditionally Improved Synthesis of Polynomial Arithmetic Through Stochastic Logic” Journal of Multivalued Logic and Soft Computing, to appear, 2022.
7. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Sorting, Shifting and Searching in DNA”, Journal of Natural Computing (under review).
8. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Cascadable Stochastic Logic for DNA Storage” IEEE International Conference on Visual Communications and Image Processing, 2021.

Honors and Awards: Nothing to Report

Protocol Activity Status:

RPPR Final Report
as of 17-Jun-2022

Technology Transfer: DNA-based image storage and retrieval
Inventors: Chao Pan, SM Hossein Tabatabaei Yazdi, SeyedKasra Tabatabaei, Alvaro G Hernandez, Charles M Schroeder, Olgica Milenkovic
2021/6/24, US 17102143

Coded Trace Reconstruction
Inventors: Olgica Milenkovic, Ryan Gabrys, João Ribeiro, Mahdi Cheraghchi
2021/4/29, US 17069247

Nick-Based Data Storage in Native Nucleic Acids
Inventors: Olgica Milenkovic, Alvaro Gonzalo Hernandez, Seyed Kasra Tabatabaei, Huimin Zhao
2020 No. 16, US 136066

PARTICIPANTS:

Participant Type: PD/PI
Participant: Olgica Milenkovic
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co PD/PI
Participant: Mark Riedel
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co PD/PI
Participant: David Soloveichik
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co PD/PI
Participant: Alvaro Gonzalo Hernandez
Person Months Worked: 6.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co PD/PI
Participant: Mark Riedel
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co-Investigator
Participant: Charles Schroeder
Person Months Worked: 6.00 **Funding Support:**

Project Contribution:
National Academy Member: N

Participant Type: Co-Investigator
Participant: Andrew Ellington
Person Months Worked: 6.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Co-Investigator
Participant: Huimin Zhao
Person Months Worked: 6.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Kasra Tabatabaei
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Chao Pan
Person Months Worked: 10.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Boya Wang
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Tonglin Chen
Person Months Worked: 15.00 **Funding Support:**
Project Contribution:
National Academy Member: N

Participant Type: Graduate Student (research assistant)
Participant: Chris Chalk
Person Months Worked: 6.00 **Funding Support:**
Project Contribution:
National Academy Member: N

RPPR Final Report
as of 17-Jun-2022

Partners

,

I certify that the information in the report is complete and accurate:
Signature: Olgica Milenkovic
Signature Date: 6/4/22 3:30PM

FINAL REPORT FOR GRANT AF894 DARPA

W911NF-18-2-0032

NATIVE DNA-BASED DATA STORAGE AND COMPUTING

Submitted by University of Illinois, Urbana-Champaign

- **PROJECT GOALS:**

1. **Develop new DNA-based data storage system that use native DNA, or combinations of native and synthetic DNA for massive DNA storage of heterogeneous data in order to enable parallel writing, fast random access and single-bit random access, reduce the cost and latency of the reading/writing process.**
2. **Enable multidimensional storage both in the DNA content and backbone via a combination of synthetic DNA and superimposed nicks for metadata. Extend the capability of molecular recorders to allow for easy privacy-preserving information erasure and rewriting.**
3. **Develop new coding and machine learning methods specialized for the DNA storage media that increase the reliability of the recording system.**
4. **Develop watermarking and other authentication protocols for DNA-encoded data da and implement multilevel flash memory storage systems via enzymatic synthesis of length-modulated tails at the sites of nicks.**
5. **Develop the first known parallel, in-memory computing paradigm that operates directly on data stored in DNA. In particular, demonstrate massively parallel single instruction set register incrementing, Rule 110 and sorting in memory on nicked data.**

The role of team members from the University of Illinois was to address Project Goals 1-4. Subawardees from University of Texas, Austin and University of Minnesota were tasked with addressing Project Goal 5.

- **SUMMARY OF CONTRIBUTIONS:**

1. **Developed the theoretical underpinnings of the first DNA-based data storage paradigm that uses native (bacterial) DNA as the storage media (Part I of the report).**
2. **Experimentally implemented the system by encoding both text and imaging data. The key idea is to record information in the form of nicks in the sugar-phosphate backbone, with a nick representing either 1,2 (sense,antisense) or 0 (no nick) (Part I of the report).**
3. **Optimized a new class of nicking enzymes (PfAgo) to operate in parallel fashion with negligible off-target activity (Part I of the report).**
4. **Performed the first nanopore sequencing experiments for the purpose of detecting nicks, toeholds and tails (Part I of the report).**
5. **Enabled the first bitwise random access protocol for DNA-based data storage (Part I of the report).**
6. **Developed new coding methods for error-correction (e.g., coded trace reconstruction), constrained coding (e.g., set codes) and image inpainting methods for data reconstruction.**
7. **Devised the first in-memory computing method that operates directly on information stored in DNA nicks. The paradigm, termed SIMDNA, combines strand displacement with unique in memory protocols for register incrementing, Rule 110 and sorting.**
8. **Work in progress includes writing out the results for DNA flash memories.**

Publications:

Storage thrust

1. Chao Pan, S Kasra Tabatabaei, SM Hossein Tabatabaei Yazdi, Alvaro G Hernandez, Charles M Schroeder, Olgica Milenkovic, “Rewritable Two-Dimensional DNA-Based Data Storage with Machine Learning Reconstruction,” to appear in *Nature Communications* (accepted in 2020, delayed due to copyright issues for two images), 2022.
2. Nagendra Athreya, Olgica Milenkovic, Jean-Pierre Leburton, “Interaction dynamics and site-specific electronic recognition of DNA-nicks with 2D solid-state nanopores,” *npj 2D Materials and Applications*, vol. 4, no. 1, pp. 1-8, Nature Publishing Group, 2020.
3. M Cheraghchi, R Gabrys, O Milenkovic, J Ribeiro, “Coded trace reconstruction,” *IEEE Transactions on Information Theory* 66 (10), 6084-6103, 2020.
4. S Kasra Tabatabaei, Boya Wang, Nagendra Bala Murali Athreya, Behnam Enghiad, Alvaro Gonzalo Hernandez, Christopher J Fields, Jean-Pierre Leburton, David Soloveichik, Huimin Zhao, Olgica Milenkovic, “DNA punch cards for storing data on native DNA sequences via enzymatic nicking,” Nature Communications, vol. 11, no. 1, pp. 1-10, 2020. **(NOTABLE COVERAGE: Nature, New Scientist, Scientific American, Editor’s Highlights)**
5. Abhishek Agarwal, Olgica Milenkovic, Srilakshmi Pattabiraman, Joao Ribeiro, “Group Testing with Runlength Constraints for Topological Molecular Storage,” International Symposium on Information Theory, ISIT 2020.
6. Ryan Gabrys, Hoang Dau, Charles Colbourn, Olgica Milenkovic, “Set-codes with small intersections and small discrepancies,” *SIAM Journal on Discrete Mathematics* 34 (2), 1148-1171 pp. 132-137, 2019.
7. Ke Liu, Chao Pan, Alexandre Kuhn, Adrian Pascal Nievergelt, Georg E Fantner, Olgica Milenkovic, Aleksandra Radenovic, “Detecting topological variations of DNA at single-molecule level,” *Nature Communications*, vol. 12, no. 1, pp. 1-9, 2019.

Patents:

DNA-based image storage and retrieval

Inventors: Chao Pan, SM Hossein Tabatabaei Yazdi, SeyedKasra Tabatabaei, Alvaro G Hernandez, Charles M Schroeder, Olgica Milenkovic

2021/6/24, US 17102143

Coded Trace Reconstruction

Inventors: Olgica Milenkovic, Ryan Gabrys, João Ribeiro, Mahdi Cheraghchi

2021/4/29, US 17069247

Nick-Based Data Storage in Native Nucleic Acids

Inventors: Olgica Milenkovic, Alvaro Gonzalo Hernandez, Seyed Kasra Tabatabaei, Huimin Zhao

2020 No. 16, US 136066

Computing thrust

1. B Wang, C Chalk, D Soloveichik, “SIMD||DNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades,” DNA25: International Conference on DNA Computing and Molecular Programming, Lecture Notes in Computer Science 11648: 219-235 (2019).
2. B Wang, S Wang, C Chalk, A Ellington, D Soloveichik, "Parallel molecular computation on digital data stored in DNA," in preparation.
3. Tonglin Chen, Arnav Solanki, and Marc Riedel, “Parallel Pairwise Operations on Data Stored in DNA: Sorting, Shifting, and Searching” LIPI DNA Computing, Vol. 205, pp. 11:1 – 11:21, 2021.
4. Amlan Ganguly, Sergi Abadal, Ishan Thakkar, Natalie Enright Jerger, Marc Riedel, Masoud Babaie, Rajeev Balasubramonian, Abu Sebastian, Sudeep Pasricha, and Baris Taskin “Interconnects for DNA, Quantum, In-Memory and Optical Computing” *IEEE Micro,* doi: 10.1109/MM.2022.3150684, Feb. 14, 2022.
5. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Performing Math from Truth Tables with DNA”, under review, *PloS One*, 2022.
6. Patrick Holec, Weikang Qian, Marc Riedel, and Ivo Rosenberg, “Conditionally Improved Synthesis of Polynomial Arithmetic Through Stochastic Logic” *Journal of Multivalued Logic and Soft Computing*, to appear, 2022.
7. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Sorting, Shifting and Searching in DNA”, Journal of Natural Computing (under review).

8. Arnav Solanki, Tonglin Chen, and Marc Riedel, “Cascadable Stochastic Logic for DNA Storage” *IEEE International Conference on Visual Communications and Image Processing*, 2021.

PART I: STORING INFORMATION IN NATIVE DNA VIA NICKS AND 2D-DNA STORAGE

Contents

1 System Architecture of DNA Punchcards

Synthetic DNA-based data storage systems (1-11) have received significant attention due to the promise of ultrahigh storage density. However, all proposed systems suffer from high cost, read-write latency and error-rates that render them impractical. One means to avoid synthesizing DNA is to use readily available native DNA. As native DNA content is fixed, one may adopt an alternative recording strategy that modifies the DNA topology to encode desired information. We developed the first macromolecular storage paradigm in which data is written in the form of “nicks (punches)” at predetermined positions on the sugar-phosphate backbone of native dsDNA. The platform accommodates parallel nicking on multiple “orthogonal” genomic DNA fragments, paired nicking and disassociation for creating “toehold” regions that enable single-bit random access and strand displacement. As a proof of concept, we used the multiple-turnover programmable restriction enzyme *Pyrococcus furiosus* Argonaute (12) to punch files into the PCR products of *Escherichia coli* genomic DNA. The encoded data is reliably reconstructed through simple read alignment.

All existing DNA-based data recording architectures store user content in synthetic DNA oligos (1-11) and retrieve desired information via next-generation (e.g., HiSeq and MiSeq) or nanopore sequencing technologies (5). Although DNA sequencing can be performed routinely and at low cost, *de novo* synthesis of DNA strands with a predetermined content is a major bottleneck (14); DNA synthesis protocols add one nucleotide per cycle and are inherently slow and prohibitively expensive compared to existing optical and magnetic writing mechanisms. To address these limitations of DNA-based data storage systems and reduce their cost, we developed a new storage paradigm that represents information via *in vitro* topological modifications on native DNA sequences (e.g., genomic DNA or its cloned or PCR-amplified products).

In the write component of the proposed system (Figure 1, top), binary user information is converted into a positional code that describes where native DNA sequence is to be topologically modified, i.e. nicked. A nick is a cut in the sugar-phosphate backbone between two adjacent nucleotides in double-stranded DNA, and each nick encodes either $\log_2 2 = 1$ bit (if only one strand is allowed to be nicked or left unchanged) or $\log_2 3 = 1.58$ bits (if either of the two strands is allowed to be nicked or both left unchanged). As bacterial cells are easy to handle and grow, the native DNA nicking substrates of choice are the PCR products of one or multiple regions of the bacterial genomic DNA, that can be easily isolated via simple and inexpensive commercially available protocols. Native DNA is organized into *orthogonal registers*, with each register represented by multiple replicas of one isolated genomic region; two registers are termed orthogonal if their sequence edit distance is sufficiently large (>55%). Each register is nicked in a combinatorial fashion, determined by the information content to be stored. To enable fast and efficient data recording, a library of registers with desired nicking site patterns is created in parallel. Registers or orthogonal registers are subsequently placed into grids of microplates that enable random access to registers and spatially organize the data, similar to tracks and sectors on disks and tapes.

In the read component of the proposed system (Figure 1, bottom), nicked DNA may be processed using different protocols: next-generation sequencing (NGS), solid-state nanopore sequencing and strand displacement (16-18). As demonstrated by our molecular dynamics simulations, MoS_2 nanopore sequencers can operate directly on the nicked DNA by correlating ionic and transverse sheet currents (Figures S12-S14).

The register chosen for experimental verification is a DNA fragment of length 450 bps that was PCR-amplified from the genomic DNA of *E. coli* K12 MG1655. The register contains ten

designated nicking positions. Although registers as long as 10 Kbps can be easily accommodated, they are harder to read; hence, multiple orthogonal registers are preferred to long registers. The nicking positions are determined based on four straightforward to accommodate sequence composition constraints that enable precise nicking. To prevent disassociation of the two strands at room temperature, the nicking sites are placed at a conservative distance of at least 25 bps apart. The user file is parsed into 10-bit strings which are converted into nicking positions of spatially arranged registers, according to the rule that a '1' corresponds to a nick, while a '0' corresponds to the absence of a nick (the number of bits recorded is chosen based on the density of nicks and the length of the register). As an example, the string 0110000100 is converted into the positional code 238, indicating that nicking needs to be performed at the 2^{nd}, 3^{rd} and 8^{th} positions (Figure 2A). Note that recording the bit '0' does not require any reactions, as it corresponds to the "no nick" ground state. Therefore, nick-based recording effectively reduces the size of the file to be actually recorded by half. This property of native DNA storage resembles that of compact disk (CD) and other recorders.

2 Writing Mechanisms

As the writing tool, we needed to choose a nicking enzyme with optimized programmability and nicking activity. Nicking endonucleases (natural/engineered) are only able to detect specific sequences in DNA strands; they can bind certain nucleotide sequences. Also, *Streptococcus pyogenes* Cas9 nickase (*Sp*Cas9n), as a widely used tool for genetic engineering applications, requires the presence of a protospacer adjacent motif (PAM) sequence (NGG) at the 3' site of the target DNA. The NGG motif constraint limits the nicking space to 1/16 of the available positions. The *Sp*Cas9n complex uses RNA guides (gRNAs) to bind the target, which makes it unstable and hard to handle. Furthermore, *Sp*Cas9n is a single turnover enzyme (15), i.e., one molecule of the enzyme can generate one nick per DNA molecule only. These make *Sp*Cas9n exhibit low efficiency and versatility for storage applications. To address these problems, we used the programmable restriction enzyme *Pyrococcus furiosus* Argonaute (*Pf*Ago) (12) as our writing tool. *Pf*Ago has significantly larger flexibility in double-stranded DNA cleaving than *Sp*Cas9n and, most importantly, has a high turnover rate (one enzyme molecule can be used to create a large number of nicks). *Pf*Ago also uses 16 nt 5'-phosphorylated DNA guides (gDNAs) that are more stable and easier to handle *in vitro*. We experimentally demonstrated that under proper reaction conditions, *Pf*Ago can successfully perform simultaneous nicking of multiple prescribed sites with high efficiency and precision within 40 min. A comparison of the nicking performance of *Sp*Cas9n and *Pf*Ago may be found in Table S2 and Figure S3-4.

To facilitate writing multiple user files in parallel, we designed *Pf*Ago guides for all ten nicking positions in the chosen register and created registers bearing all 2^{10} = 1024 nicking combinations (Table S3). Registers were placed in microplates in an order dictated by the content to be encoded.

Since the length, sequence composition and nicking sites of a register are all known beforehand, reading amounts to detecting the positions of the nicks. The nicked registers are first denatured, resulting in ssDNA fragments of variable length dictated by the nicked positions. These length-modulated ssDNA fragments are subsequently converted into a dsDNA library, sequenced on Illumina MiSeq, and the resulting reads are aligned to the known reference register sequence. The positions of the nicks are determined based on read coverage analysis, the insert size distributions and through alignment with the reference sequence; potential nicking sites that are not covered are declared to be '0's (Figure 2A-C).

3 Reading Mechanisms

We reported write-read results for a 272-word file of size 0.4 KB containing Lincoln's Gettysburg Address (LGA) and a JPEG image of the Lincoln Memorial of size 14 KB (Figure S6). Both files were compressed and converted into ASCII and retrieved with perfect accuracy. Given the inherent redundancy of the sequencing process and the careful selection of the nicking sites and register sequences, no error-correction redundancy was needed (Figure 2B, C and Figure S5B-D).

A potentially more efficient, portable and cost-effective approach to read the nicked DNA registers is via two-dimensional (2D) solid-state nanopore membranes. One approach is to use toeholds, short single-stranded regions on dsDNA created through two closely placed nicks, instead of single nicks. Toeholds can be accurately read using solid-state SiN_x and MoS_2 nanopores, as recently reported in (13). The cost of creating toeholds is twice as high as that of nicks, since one needs two different nicking guides. To mitigate this problem, one may attempt to detect nicks directly. To illustrate the feasibility of this approach, we performed Molecular Dynamics (MD) simulations based on quantum transport calculations. These revealed a strong inverse correlation between the ionic and electronic sheet current signals along the membrane induced by nicks in MoS_2 nanopores (Figure 2D, Figures S12-S14 & Video S1). The regions of

strong negative “extremal” correlations between the ionic current and transverse sheet conductance strongly associate with the positions of the nicks.

In addition to allowing for nanopore-based reading, toeholds also enable complex in-memory computations. Toehold-mediated DNA strand displacement is a versatile tool for engineering dynamic molecular systems and performing molecular computations (16). Information is processed through releasing strands in a controlled fashion, with toeholds serving as initiation sites to which these input strands bind to displace a previously bound output strand.

Toeholds are usually generated by binding two regions of synthetic ssDNA and leaving a short fragment unbound. However, with *Pf*Ago, one can easily create toeholds in native DNA. To form a toehold, two nicks are generated within 14 bps. Under appropriate buffer and temperature conditions, in a single reaction the 14 nt strand between the two nicks disassociates, leaving a toehold on the double-stranded DNA (Figure S15).

Fluorescence-based methods can detect the existence of a toehold and the concentration of registers bearing a toehold without modifying the DNA registers. We illustrate this process on a register encoding 0010000000, with a toehold of length 14 nts at the nicking position 3. As shown in Figure 3A, a fluorophore and quencher labelled reporter strand with a sequence complementary to the toehold can hybridize to the toehold segment, producing a fluorescence signal resulting from an increase of the distance between the fluorophore and the quencher. We were also able to reliably measure different ratios of native DNA fragments with and without toeholds within 20 mins (Figure 3B). Since the reporter has a short single stranded overhang, it can be pulled off from the register upon hybridization, making the readout process non-destructive (Polyacrylamide gel electrophoresis analysis, Figure 3C). This feature equips our proposed storage system with unique nondestructive bitwise random access, since one is able to design specific reporters to detect any desired toehold sequence which accompanies a nick. It also enables computations on data encoded in nicks, as described in two recent papers (19,20).

In summary, by reprogramming *Pf*Ago as a universal nickase and using *E. coli* native DNA sequences, *we* have implemented the first DNA-based storage system that mitigates the use of costly long synthetic DNA strands for storing user information. Our platform utilizes a parallel writing mechanism that combines an inexpensive nicking enzyme and a small number of short and inexpensive synthetic DNA guides. In addition, this approach enables enzyme driven toehold creation, allowing for bitwise random access and in memory computing via strand displacement.

Nick-based storage outperforms known synthetic DNA technologies in all relevant performance categories except for recording density; but the roughly one order of magnitude loss is insignificant for a system that already compacts petabytes in grams and overcompensated by

the four-fold reduction of cost in our proposed system (Table 1). It also allows for cost-efficient scaling as a) long registers and mixtures of orthogonal registers may be nicked simultaneously; b) most uncompressed data files do not contain all possible 10-mers or compositions of orthogonal *k*-mers; c) genomic DNA and *Pf*Ago, as the writing tool, are readily available, and the mass of the created DNA products by far exceeds that of synthetic DNA, significantly increasing the number of readout cycles with NGS devices. This storage system may also be used to superimpose, erase and rewrite categorical and metadata on synthetic DNA oligos, in which case bitwise random access enables efficient non-destructive search and concentration sensing.

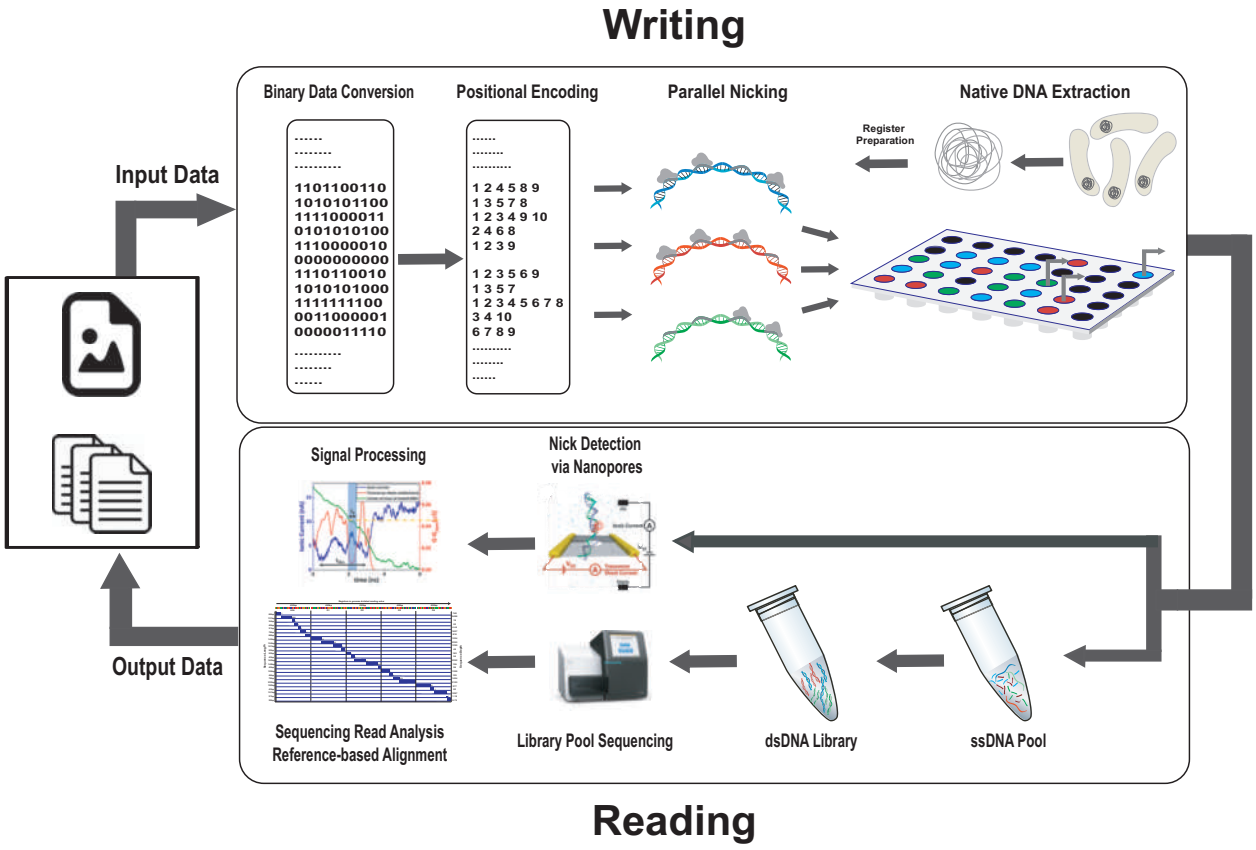
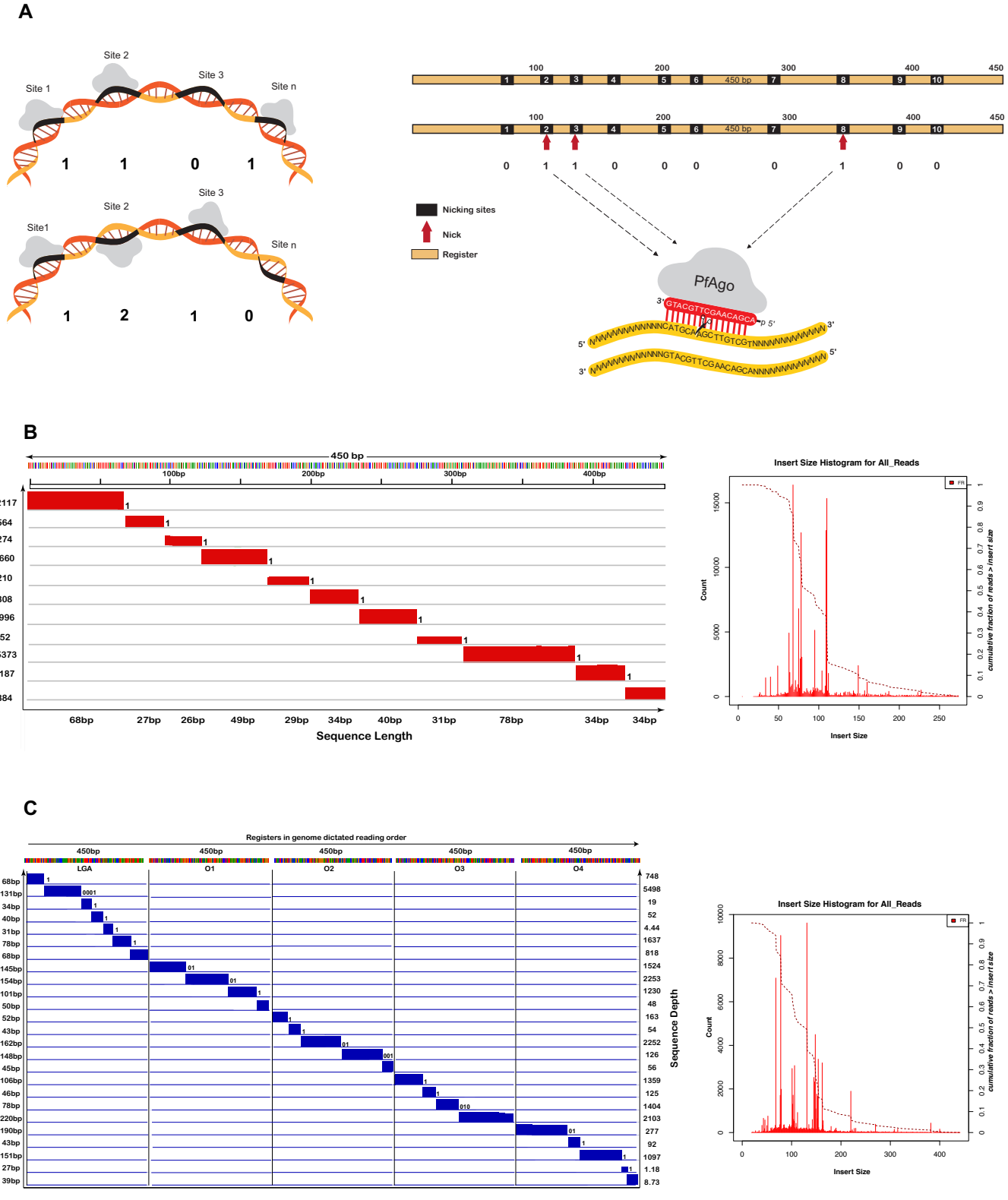


Figure 1 | **The native DNA-based data storage platform**. In the **Write component**, arbitrary user content is converted into a binary message. The message is then parsed into blocks of *m* bits, where *m* corresponds to the number of nicking positions on the register (for the running example, *m* = 10). Subsequently, binary information is translated into positional information indicating where to nick. Nicking reactions are performed in parallel via combinations of *Pf*Ago and guides. In the **Read component**, nicked products are purified and denatured to obtain a pool of ssDNAs of different lengths. The pool of ssDNAs is sequenced via MiSeq. The output reads are processed by first performing reference-based alignment of the reads, and then using read coverages to determine the nicked positions.


A
Site 1
Site 2
Site 3
Site n
1 1 0 1
Site1
Site 2
Site 3
Site n
1 2 1 0
100
200
300
400
450
450 bp
0 1 1 0 0 0 0 1 0 0
Nicking sites
Nick
Register
PfAgo
B
450 bp
100bp
200bp
300bp
400bp
Sequence Depth
32117
564
274
4660
210
808
2996
52
15373
1187
884
68bp 27bp 26bp 49bp 29bp 34bp 40bp 31bp 78bp 34bp 34bp
Sequence Length
Insert Size Histogram for All_Reads
Count
Insert Size
cumulative fraction of reads > insert size
C
Registers in genome dictated reading order
450bp
LGA
O1
O2
O3
O4
Sequence Length
Sequence Depth
68bp 748
131bp 5498
34bp 19
40bp 52
31bp 4.44
78bp 1637
68bp 818
145bp 1524
154bp 2253
101bp 1230
50bp 48
52bp 163
43bp 54
162bp 2252
148bp 126
45bp 56
106bp 1359
46bp 125
78bp 1404
220bp 2103
190bp 277
43bp 92
151bp 1097
27bp 1.18
39bp 8.73
Insert Size Histogram for All_Reads
Count
Insert Size
cumulative fraction of reads > insert size

D

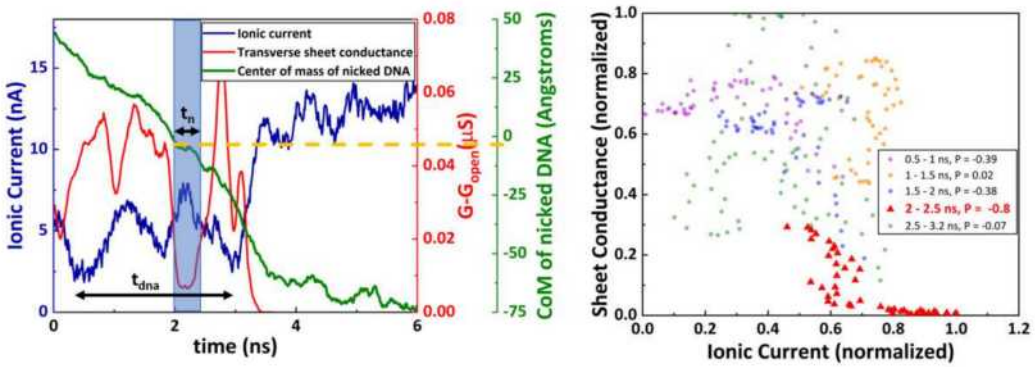


Figure 2 | Writing and reading the encoded data. **A**) *Pf*Ago can nick several pre-designated locations on only one strand (left, **top**) or both strands (left, **bottom**), simultaneously. In the first register, the stored content is 110…1, while in the second register, the content is 1210…0. The chosen register is a PCR product of a 450 bp *E. coli* genomic DNA fragment with 10 pre-designated non-uniformly spaced nicking positions (right, **top**). The positional code 238 corresponds to the binary vector 0110000100 (right**, bottom**). **B**) The MiSeq sequencing reads were aligned to the reference register to determine the positions of the nicks. The size distribution histogram (right) and coverage plots (left) are then generated based on the frequency and coverage depth of the reads. Coverage plots allow for straightforward detection of nicked and unnicked sites. In the example shown, all the ten positions were nicked, resulting in eleven aligned fragments. **C**) Five orthogonal registers used instead of one single register. Each vertical section represents one register in genome dictated reading order, and each row shows the read lengths retrieved after sequencing analysis. Read lengths are recorded on the left and sequencing depths on the right axis. **D) Reading via solid-state nanopores**. Plot of the calculated ionic current (blue), differential transverse sheet conductance (red) and center of mass of the nicked site (green) versus time (t_{dna} represents the translocation time of the entire dsDNA, while t_n represents the dwell time of the nick in the pore (left)). Scatter plot of normalized sheet conductance versus normalized ionic current over t_{dna} (right). A strong inverse correlation with P = -0.8 between the ionic current and transverse sheet conductance is observed at 2 – 2.5 ns, during which time the ion current reaches its global maximum, while the sheet current reaches its global minimum in t_{dna}. This time interval corresponds to the nick translocation event.

4 Bitwise Random Access

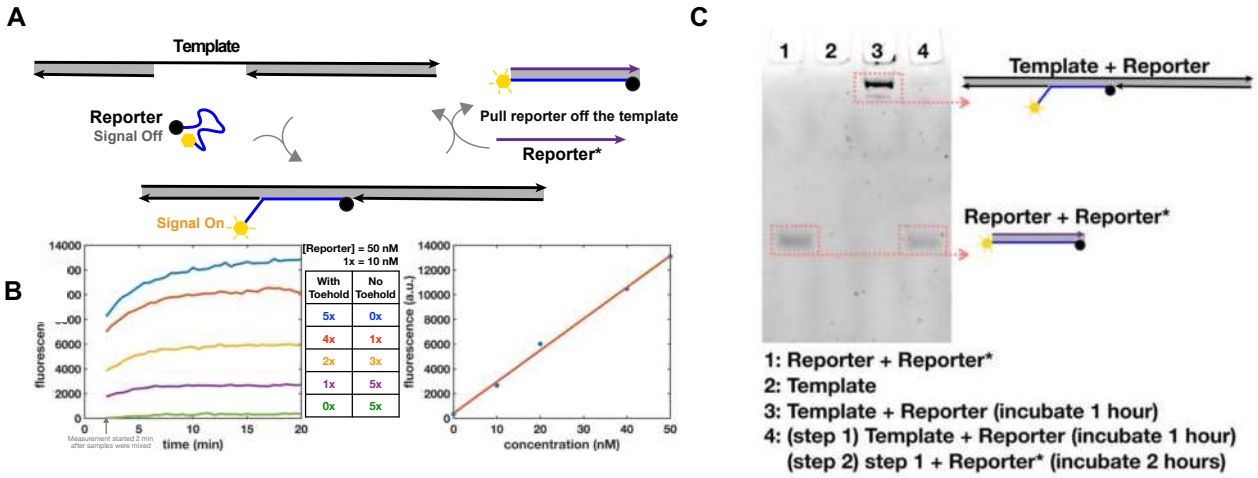


Figure 3 | **Non-destructive detection of a toehold**. **A)** Non-destructive detection of toeholds through a fluorophore and quencher labelled Reporter strand. Once the Reporter hybridizes with the toehold on the register strand, a fluorescence signal is observed due to the increase of the distance between the fluorophore and quencher. The Reporter strand can be pulled off from the register once the Reporter* strand hybridizes with the Reporter. **B)** Kinetics of detecting the concentrations of registers with and without toeholds in a mixtures **(left).** The fluorescence signals saturate within 20 minutes. The samples were mixed no more than 2 min before measurement. The concentration of toehold-ed DNA can be accurately quantified through fluorescence intensity **(right),** as it increases linearly with the concentration of the registers with toehold. **C)** PAGE gel results for non-destructive detection of a toehold. The gel was not stained with other fluorescence dyes, thus only the species with self-fluorescence is observed. After adding the Reporter, a large size complex appears in lane 3, indicating hybridization of the Reporter and the register. After the Reporter* is added, as seen in lane 4, the large size complex in lane 3 no longer exhibits self-fluorescence, indicating that the Reporter strand is pulled off from the register.

Table 1 | Comparison of synthetic and native DNA-based data storage platforms. Native DNA-based platforms outperform synthetic DNA-based approaches in all performance categories, except for storage density.

DNA-based Storage Method	Price per Bit Replica	Writing Latency	Reading Latency	Enables Computation?	Bit-wise Random Access	Maximum achievable physical Density	Information Density	(Optimal) Coding Loss (10)
Synthesis - based (1-11)	>$0.06 <$0.12 (10)	Sequential de novo synthesis/ Hours	NGS/hours	×	×	200 Ebytes/g (9)	< 2 bits/bp (to account for coding loss, usually ~1.5 bits/bp)	21% (9,10)
This work	<$\$1.5\times10^{-6}$	Parallel Nicking/ < 40 min	NGS followed by reference alignment/ hours	✓	✓	4 Ebytes/g	0.036 bits/bp	0%

References

1. G. M. Church, Y. Gao, S. Kosuri. *Science* **337,** 1628-1628 (2012).
2. N. Goldman *et al. Nature* **494,** 77-80 (2013).
3. S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, O. Milenkovic. *Sci. Rep.* **5,** 14138 (2015).
4. R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, W. J. Stark, *Angew. Chem. Int. Ed.* **54,** 2552–2555 (2015).
5. S. H. T. Yazdi, R. Gabrys, and O. Milenkovic. *Scientific reports* 7.1 (2017).
6. S.L. Shipman, J. Nivala, J.D. Macklis, G.M. Church. *Nature* **547,** 345–349 (2017).
7. O. Milenkovic, R. Gabrys, H.M. Kiah, S.H.T. Yazdi. *IEEE Spectrum*, *55*(5), 40-45 (2018).
8. V. Zhirnov, R.M. Zadegan, G.S. Sandhu, G.M. Church, W.L. Hughes. *Nat. Mater.* **15,** 366-370 (2016).
9. Y. Erlich, D. Zielinski, D. *Science*, **355**, 950-954 (2017).
10. S.H.T. Yazdi, H.M. Kiah, E. Ruiz-Garcia, J. Ma, H. Zhao, O. Milenkovic. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications,* **1, 3,** 230-248, (2015).
11. C. Laure, D. Karamessini, O. Milenkovic, L. Charles, J.F. Lutz. *Angewandte Chemie International Edition*, ***55***(36), pp.10722-10725 (2016).
12. B. Enghiad, H. Zhao, ACS Synth. Biol., **6**, 752−757 (2017).
13. K. Liu, C. Pan, A. Kuhn, A.P. Nievergelt, G. Fantner, O. Milenkovic, A. Radenovic. *Nature Communications*, **10**, 3 (2019).
14. S. Palluk, D.H. Arlow, T. De Rond, S. Barthel, J.S. Kang, R. Bector,H.M. Baghdassarian, A.N. Truong, P.W. Kim, A.K. Singh, N.J. Hillson, J.D. Keasling. *Nat Biotechnol.*, **36**, 645-650 (2018).
15. C. Andres, M. Jinek. *Methods Enzymol.* **546**, 1-20 (2016)

16. B. Yurke, A.J. Turberfield, A.P. Mills, F.C. Simmel, J.L. Neumann. *Nature*, **406**:605–608 (2000).
17. D.Y. Zhang, G. Seelig. *Nat Chem.,* **3**:103–113 (2011).
18. B. Wang, C. Thachuk, A. Ellington, E. Winfree, D. Soloveichik. *PNAS*, **115** (52), E12182-E12191 (2018).
19. B. Wang, C. Chalk, D. Soloveichik, *DNA 25 Conference,* Seattle, WA, U.S.A. (2019).
20. T. Chen, M. Riedel, *11th International Workshop on Bio-Design Automation (IWBDA),* Cambridge, England, U.K. (2019).

5 2DDNA

Traditional DNA-based data recording architectures store user information in the sequence content of synthetic DNA oligos within large pools that lack an inherent ordering, and user information is retrieved via next-generation or nanopore sequencing[6]. Despite recent progress, several issues continue to hinder the practical implementation of molecular information storage models, including the high cost of synthetic DNA, lack of straightforward rewriting mechanisms, large write-read latencies, and missing oligo errors incurred by solid-phase synthesis.

Image data is typically compressed before being recorded, and even a single mismatch can cause catastrophic error-propagation during decompression and lead to unrecognizable reproductions[6,11,12]. Moreover, the rate of synthesis and sequencing errors may vary an order of magnitude from one platform to another, while PCR reactions and topological data rewriting may cause additional gradual increases in sequencing errors. Therefore, to ensure accurate reconstruction, one needs to account for the worst-case scenario and perform extensive write-read-rewrite experiments to estimate the error rates before adding redundancy[13–15]. Moreover, the estimated error rates have to be accurate enough for efficient error correction due to the mismatched decoding parameter problem[16,17]. The mismatched-decoder problem is an issue mostly overlooked in prior works and it asserts that powerful error-correction schemes such as low-density parity-check (LDPC) codes[18] require good estimates of the channel error probability to operate properly. This is clearly hard to achieve for traditional DNA-based data storage systems due to the highly stochastic nature of the PCR, sequencing and rewriting process.

Here, we develop and experimentally test a hybrid DNA-based data storage system termed 2DDNA, to address the issue of rewriting and avoid the use of worst-case error-correcting redundancy needed to combat random and missing oligo errors that may accumulate in time and due to content changes. 2DDNA uses two different information dimensions and

combines desirable features of both synthetic and nick-based recorders[19]. This is achieved by superimposing metadata (such as ownership information, dates, clinical status descriptions) stored via nicks onto images encoded in the sequence. Sequence content carries large amounts of information, but rewriting is difficult; information stored in nicks[19] is usually of smaller volume but highly amenable for efficient, permanent and privacy-preserving erasing and rewriting. Importantly, information in both dimensions can be read simultaneously, as locations of nicks are determined using the nick-free strand as reference. Our approach is based on a simple compression scheme for images that operates separately on three different color channels and combines newly developed and existing machine learning (ML) and computer vision (CV) techniques for image reconstruction and enhancement to create high-quality replicas of the original data. For some images with highly granular details, we also propose unequal error protection methods[20] based on LDPC codes[18] that only introduce redundancy for sensitive facial features. The 2DDNA paradigm eliminates the need for worst-case coding redundancy and avoids problems with mismatched decoding parameters. It offers the possibility for users to retrieve images of quality dictated by their channel error rates, which may be seen as a form of multiresolution coding. It also offers high information density and simultaneously enables rewriting of data recorded in the backbone via ligation followed by enzymatic nicking, lending itself for use in applications with both synthetic and native DNA substrates for the sequence content[19].

The encoding framework of 2DDNA is shown in Fig. 1. In the sequence dimension, we perform aggressive quantization and specialized lossless compression that leads to two-fold file size reductions. Compression is known to cause significant losses in image quality when errors are present, so it is common practice to include up to 30% error-correction redundancy[4,7] which ultimately increases the cost of the storage system. We avoid error-correction redundancy and instead tailor our compression algorithm to accommodate image processing techniques from ML and CV to restore the image to its original quality. The specialized encoding procedure involves two steps, depicted in Fig. 1a. First, RGB channel separation is followed by 3-bit quantization and separate lossless compression of the three color channels. The latter process is performed using the Hilbert space-filling curve[22] which preserves local 2D image similarity and smoothness, thereby resulting in linear strings with small differences between adjacent string entries. Moreover,

we further employ differential encoding[23] that involves taking differences of adjacent string values to create new strings with a high probability of small symbols. Differential encoding is followed by Huffman encoding[24] which exploits the bias towards small symbol values. Together, these operations are performed separately on strings partitioned into eight subsets according to their quantized intensity (brightness) levels. Note that in our ML-based image reconstruction approach, we do not try to optimize the compression scheme: One may also use a basic 3-bit quantization scheme without lossless compression, at the cost of slightly increased file sizes.

Our encoding involves a second step that translates the binary strings into DNA oligo sequences. Here, DNA oligos of length 196nts are parsed into the following three subsequences (Fig. 1a): (1) a pair of primer sequences, each of length 20nts, used as prefix and suffix, (2) an address sequence of length 10nts, and (3) 11 information-bearing sequences of length 13nts. Primer and address sequences are used for PCR amplification and random access[5]. In addition, a block of three nucleotides is prepended to the address sequence to represent the RGB color information. When converting binary data into DNA sequence content, we use two additional constrained mappings to ensure that the maximum run length of G symbols is limited to three (to avoid G quadruplexes), and that the GC content is in the range of $40-60\%$. Overall, the mapping scheme converts blocks of 16 bits into blocks of 10nts for the address sequences, and blocks of 22 information bits into blocks of 13nts.

In the topological dimension, we record the metadata in nicks created on the backbone of the synthetic DNA molecules by transforming and generalizing our Punch-Cards system[19] that was also used for specialized in-memory molecular computing[25]. The main modifications consist in disposing of nicking enzymes that require the additional synthesis of specific guide sequences; native nicking endonucleases are used instead by employing ON-OFF encoding across different intensity pools. Short binary strings are converted into combinations of native nicking endonucleases that determine the composition of nicked/unnicked sites.

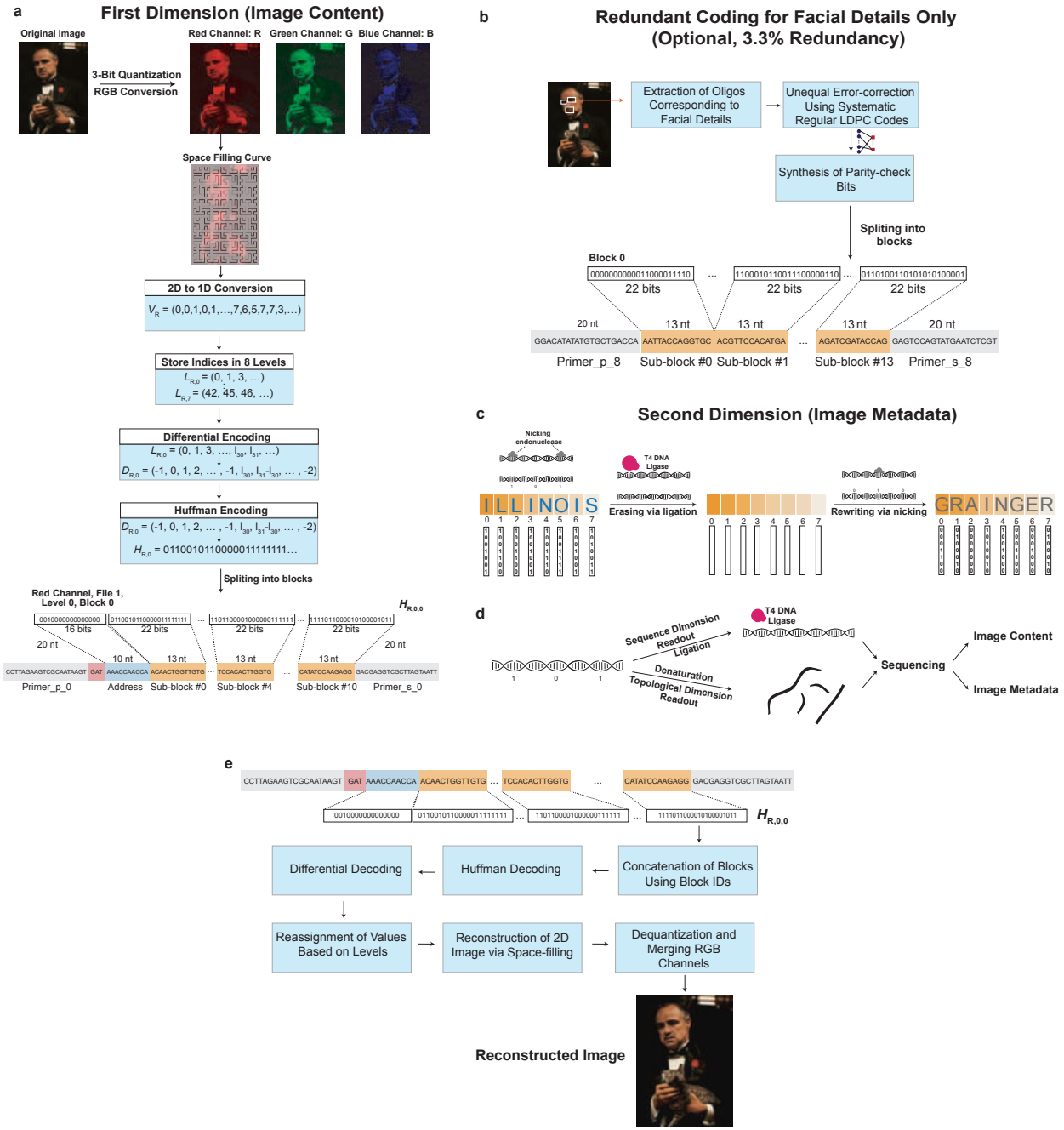


Figure 1. Schematic of the encoding and decoding procedure of our 2DDNA system. **a**) The encoding procedure in the first dimension (sequence content) entails splitting the color image into the Red (R), Green (G) and Blue (B) channel; aggressively quantizing the RGB channels from 256 to 8 intensity levels; performing lossless compression of individual channels through a combination of 2D to 1D conversion of the image data via space-filling curves followed by differential and Huffman encoding. Note that the encoding procedure is separately applied to each intensity level, and the generated binary vector is further augmented by channel information and addresses used to access the oligos. The scheme does not include

error-correction redundancy. **b)** For images with granular and highly relevant image features, one can optionally use unequal error-correction coding based on low-density parity-check (LDPC) codes with only 3.3% redundancy compared to the scheme without redundancy. **c)** The encoding procedure in the second dimension (topological content) entails representing letters of the English alphabet in ASCII format and designating one nicking endonuclease to each of the seven bits in the format. Information is encoded using mixtures of endonucleases for which the ASCII bit is equal to 1. Rewriting is performed by sealing the nicks using the T4 DNA ligase and repeating the previously outlined procedure with different data. **d)** 2D data readout through the use of two subpools, one for each storage dimension. **e**) Image decoding is performed by reversing the steps of the encoding process in the first dimension. The image[26] used in this figure is courtesy of Paramount Pictures.

More precisely, a set of complementary nicking endonucleases is used as the writing tool and selected based on two main criteria: (1) endonucleases must be highly site-specific to prevent non-specific cleavage of the DNA template and hence preserve DNA integrity; and (2) recognition sequences should be selected with sufficiently large Hamming distances between them to prevent undesired cross-nicking (i.e., an enzyme nicking an undesired target site). The mixture composition determines which letter is stored based on the corresponding ASCII code, with the caveat that a '1' is encoded through the presence of the enzyme in the mixture (ON), whereas a '0' is encoded through the absence of the enzyme (OFF). This method enables superimposing information on top of data stored in the DNA sequence content, with no need to change the synthetic platform, as shown in Fig. 1c. Nevertheless, it introduces readout challenges as the nicks break the structure of the strands and may hence lead to assembly ambiguities. We address this problem via an algorithmic solution that involves searching for potential prefix-suffix substrings in the nicked pool.

To demonstrate a proof-of-concept, we experimentally tested the storage platform on eight Marlon Brando movie stills, shown in Fig. 2a. The original files were of total size 8,654,400bits, but after the two-step encoding procedure (Fig. 2b), they reduced to 2,317,896nts. The corresponding 11,826 DNA oligos were synthesized by Integrated DNA Technologies (IDT). One pool was reserved for each of the eight levels. The oPools were sequenced on an Illumina MiSeq device following standard protocols described in the Methods. Individual sequence reads may

contain errors, so we first construct a consensus sequence by aligning reads with error-free addresses, following the approach described in our prior work[6]. This process led to 11,726 perfectly recovered sequences and 22 sequences that contain errors but do not significantly compromise the image quality; 78 oligos were either highly corrupted or completely missing from the pool.

The images generated from this procedure are depicted in Fig. 2c. Upon close inspection, it is apparent that the encoded images suffer from visible degradation, and in particular, large blocks of discolorations. These artifacts can be removed by applying a carefully designed combination of ML and CV image processing techniques (Fig. 4), tailor-made to operate on images compressed according to our method.

To correct for image discolorations, we implement a three-step post-processing procedure that has no matching counterpart in the digital domain and heavily relies of using the color channels as a natural source of redundancy. The first step includes detecting the locations with discolorations and masking them out, as shown in Fig. 4a. To pinpoint the discolored regions without direct visual inspection (i.e., in an automated fashion), as already pointed out, we leverage the separate information content in the three distinct RGB color channels. Due to the random nature of errors, it is highly unlikely to have correlated errors in multiple channels for the same pixel. Hence, the three-color decomposition acts as a 3-repetition code, because at least two of the three color channels are likely to be unperturbed. The second step involves using an existing deep learning technique known as image inpainting[28-30] to replace the masked pixels with values close to the original. Neural networks are well-suited for inpainting because they can be trained on massive datasets. For our system, we use the state-of-the-art GatedConvolution[28] and EdgeConnect[30] methods. EdgeConnect results after applying discoloration detection and image inpainting are shown in Fig. 2d. Finally, the third step involves smoothing the image to reduce blocking effects caused by quantization and blending mismatched inpainted pixels. Here, we use bilateral[31] and adaptive median smoothing[32] on the coarsely inpainted images, and we include additional image enhancement features[33] to further improve image quality. The image post-processing procedure relies on storing R, G and B color channels in different oligos and using the channels as "proxies" for repetition codes. This ensures that it is highly unlikely to have correlated errors in multiple channels for the same pixel and that discolorations can be detected through

majority rules. As a result, our scheme can be used with any other type of recorder that splits images into R, G, B subimages and stores them separately.

The results of image smoothing are depicted in Fig. 2e, and the enhanced images are shown in Fig. 2f. As shown in Figs. 2e, f, some facial details in highly granular images remain blurred even after applying the learning methods. To address these issues, we further propose the use of unequal error-protection for such images, which implies adding highly limited redundancy only to oligos bearing facial features (e.g., eyes, lips), as shown in Fig. 1b. Redundancy is added through a regular systematic LDPC codes of rate 0.75, resulting in 391 additional oligos and an overall overhead of 3.3%. Images generated from this redundant pool are shown in Fig. 2g, whereas Figs. 2h, i, j parallel the results of Figs. 2d, e, f for the case of no unequal error-correction redundancy. Note that there exist no other approaches to performing the same task in the signal processing and computer vision community. Applying state-of-the-art image enhancement method[33] directly on images generated from error-bearing DNA oligos without error-correction results in poor quality reconstructions because classical image enhancement methods cannot automatically correct discolorations (Fig. 2k and Fig. 3k). Both quantitative metrics (Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM)) as well as visual inspection of the recovered images show that our method offers significantly better performance than direct image recovery and enhancement of the corrupted DNA-encoded images. Processed images with corresponding quality values are plotted in Figs. 2a, f, j, k and Fig. 3.

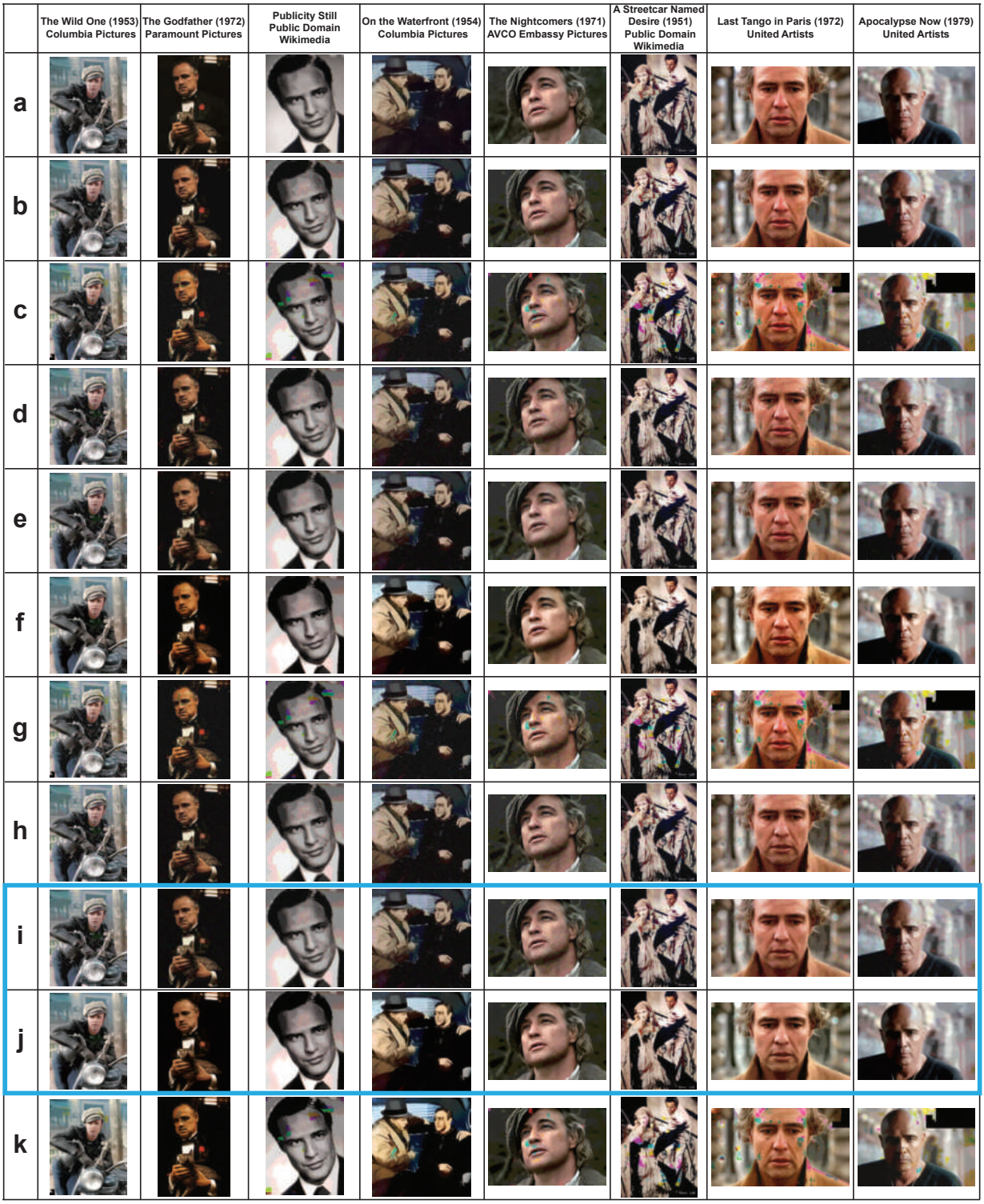


Figure 2. Write-Read results for encoding information content in the sequence dimension. **a**) Original images with 256 RGB intensity levels, encoded by 8 bits each. **b**) Quantized images with 8 RGB intensity levels, encoded by 3 bits each. **c**) Images generated directly from the information encoded in DNA oligos

without error-correction redundancy. **d**) Images reconstructed after applying a combination of discoloration detection and image inpainting on the results in **c**. **e**) Images refined via smoothing of the results depicted in **d**. **f**) Image enhancement results for images shown in **e. g**) Images reconstructed using unequal error-correcting coding for facial features. **h**) Images reconstructed after applying a combination of discoloration detection and image inpainting on the results in **g**. **i**) Images refined via smoothing of the results depicted in **h**. **j**) Image enhancement results for images shown in **i**. **k**) Image enhancement results for images shown in **c**. In summary, the best quality results – obtained using our image processing techniques – are given in **i** and **j** (boxed). The images in this figure are courtesy of: Paramount Pictures, Sony Pictures, MGM Studios, StudioCanal, American Zoetrope (© 1979 Zoetrope Corp. All Rights Reserved.), the Marlon Brando and Rod Steiger estates. The black and white public domain still of "A Streetcar Named Desire" was colorized using the software Hotpot.ai.

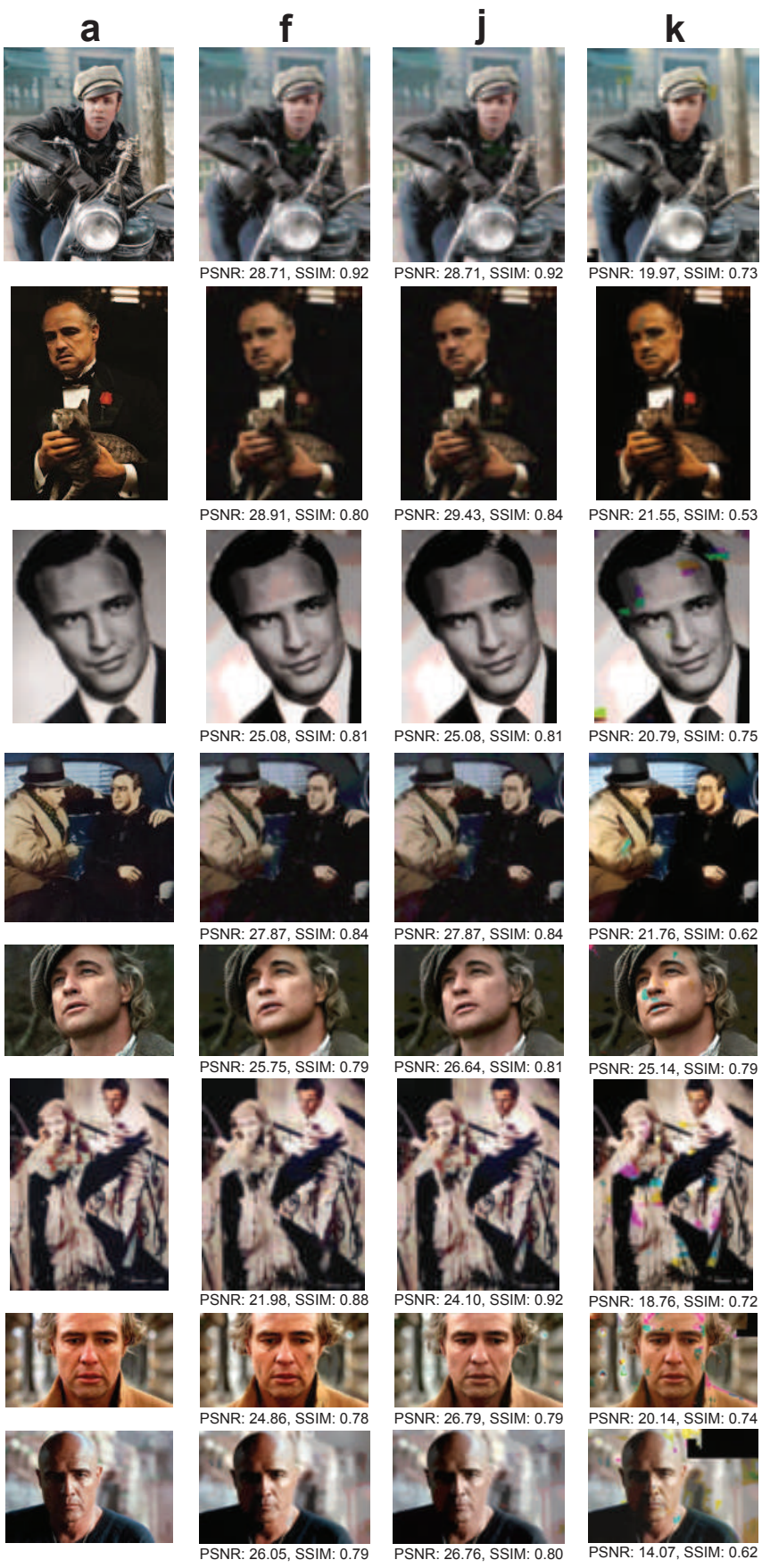

a
f
j
k
PSNR: 28.71, SSIM: 0.92
PSNR: 28.71, SSIM: 0.92
PSNR: 19.97, SSIM: 0.73
PSNR: 28.91, SSIM: 0.80
PSNR: 29.43, SSIM: 0.84
PSNR: 21.55, SSIM: 0.53
PSNR: 25.08, SSIM: 0.81
PSNR: 25.08, SSIM: 0.81
PSNR: 20.79, SSIM: 0.75
PSNR: 27.87, SSIM: 0.84
PSNR: 27.87, SSIM: 0.84
PSNR: 21.76, SSIM: 0.62
PSNR: 25.75, SSIM: 0.79
PSNR: 26.64, SSIM: 0.81
PSNR: 25.14, SSIM: 0.79
PSNR: 21.98, SSIM: 0.88
PSNR: 24.10, SSIM: 0.92
PSNR: 18.76, SSIM: 0.72
PSNR: 24.86, SSIM: 0.78
PSNR: 26.79, SSIM: 0.79
PSNR: 20.14, SSIM: 0.74
PSNR: 26.05, SSIM: 0.79
PSNR: 26.76, SSIM: 0.80
PSNR: 14.07, SSIM: 0.62

Figure 3. Comparison of our automatic discoloration detection and inpainting approach with the state-of-the-art image enhancement technique[33]. The results shown include quantitative performance metrics computed with respect to **a**. The column labels refer to the corresponding rows in Fig. 2. Column **a**): The original, uncompressed images. Column **f**): The images reconstructed using our method, without unequal protection redundancy for facial features. Column **j**): The images reconstructed using our method, with roughly 3.3% redundancy for facial features. Column **k**): Results obtained after image enhancement, applied directly to the decoded DNA oligo images with errors. The pictures in this figure are courtesy of: Paramount Pictures, Sony Pictures, MGM Studios, StudioCanal, American Zoetrope (© 1979 Zoetrope Corp. All Rights Reserved.), the Marlon Brando and Rod Steiger estates. The black and white public domain still of "A Streetcar Named Desire" was colorized using the software Hotpot.ai. The original data are provided in the Source Data file.

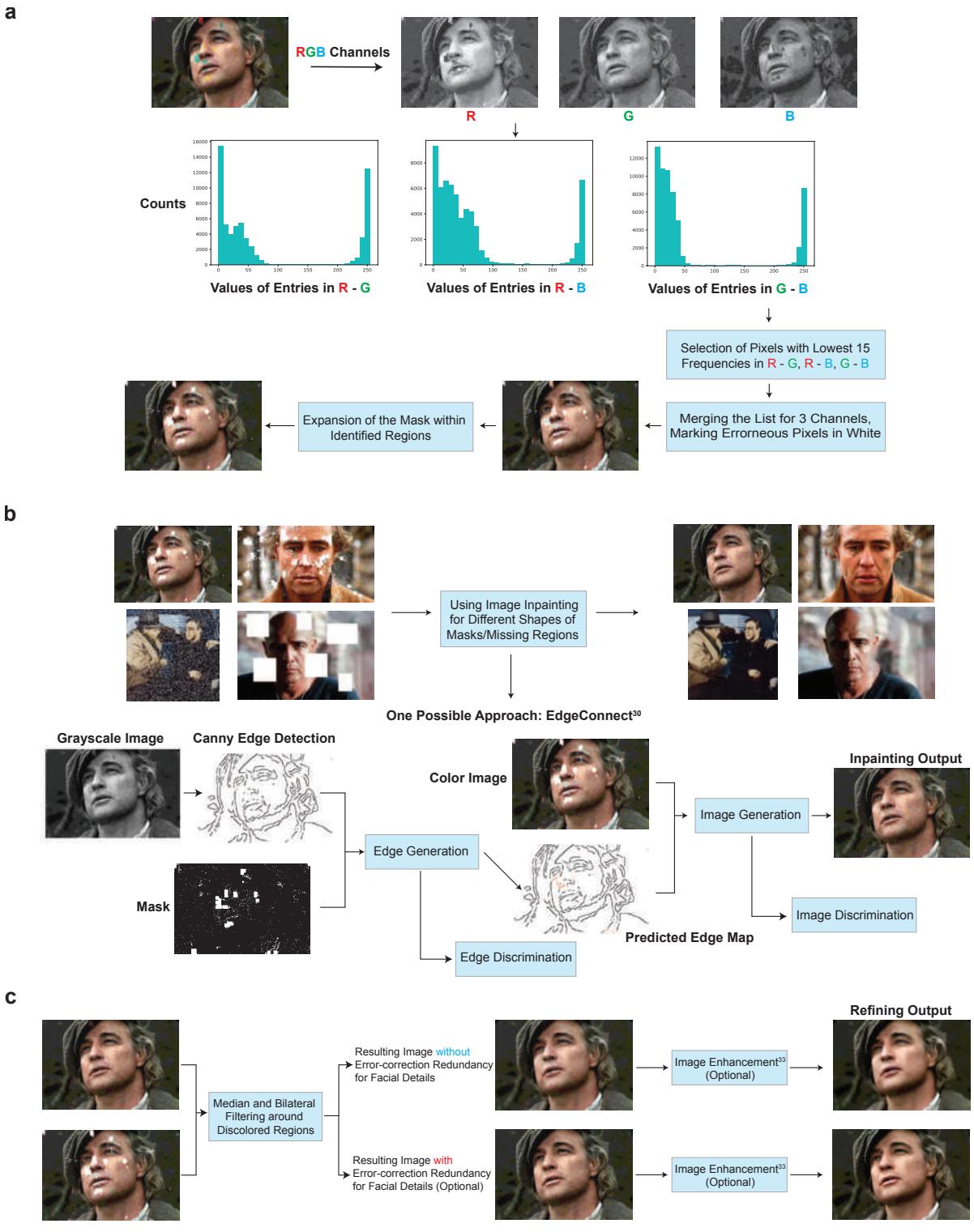


Figure 4. Diagram of the ML postprocessing techniques used to reconstruct images encoded in oPools. **a)** Automatic discoloration detection based on the natural redundancy in the three RGB color channels. The histograms reflect the frequency counts of the pairwise differences in channel intensity levels which are

used to assess which color channel may contain errors. **b**) Pixel masking and inpainting via deep-learning architectures. **c)** The smoothing and image enhancement procedures. The pictures in this figure are courtesy of: Paramount Pictures, Sony Pictures, MGM Studios, StudioCanal, American Zoetrope (© 1979 Zoetrope Corp. All Rights Reserved.), the Marlon Brando and Rod Steiger estates. The original data are provided in the Source Data file.

Note that our compression scheme mitigates the effects of catastrophic error-propagation which may be otherwise present when using a JPEG compressor. As JPEG formats are highly sensitive to errors, they result in poor-quality reconstructions if one does not use a coding overhead that guarantees exact reconstruction. Furthermore, alternative methods based on joint source-channel coding[41-42] still require introducing error-control redundancy which we are aiming to dispose of in our learning-based approach. To demonstrate this point, we performed extensive simulations with six combinations of JPEG image compression qualities and matching error-control coding schemes. For JPEG-compressed files with different quality parameters (as defined in the Python Pillow Package for all image formats, JPEG included, on a scale from 1 (worst) to 95 (best)), we added LDPC redundancy to the compressed data for error-correction to ensure that the resulting number of oligos (file size) is as close as possible to that used in our experiment. The base substitution error is set to 0.8%, while the missing oligo error is set to 0.7%, matching the numbers obtained experimentally, leading to an overall bit error of 1.9%. We decoded the binary information from the erroneous DNA oligos using LDPC codes, followed by JPEG reconstruction. Part of the results are shown in Fig. 5. Note that since JPEG has very specific formatting rules, missing or erroneous critical identifiers in JPEG files leads to system errors, such as OSError in Python.

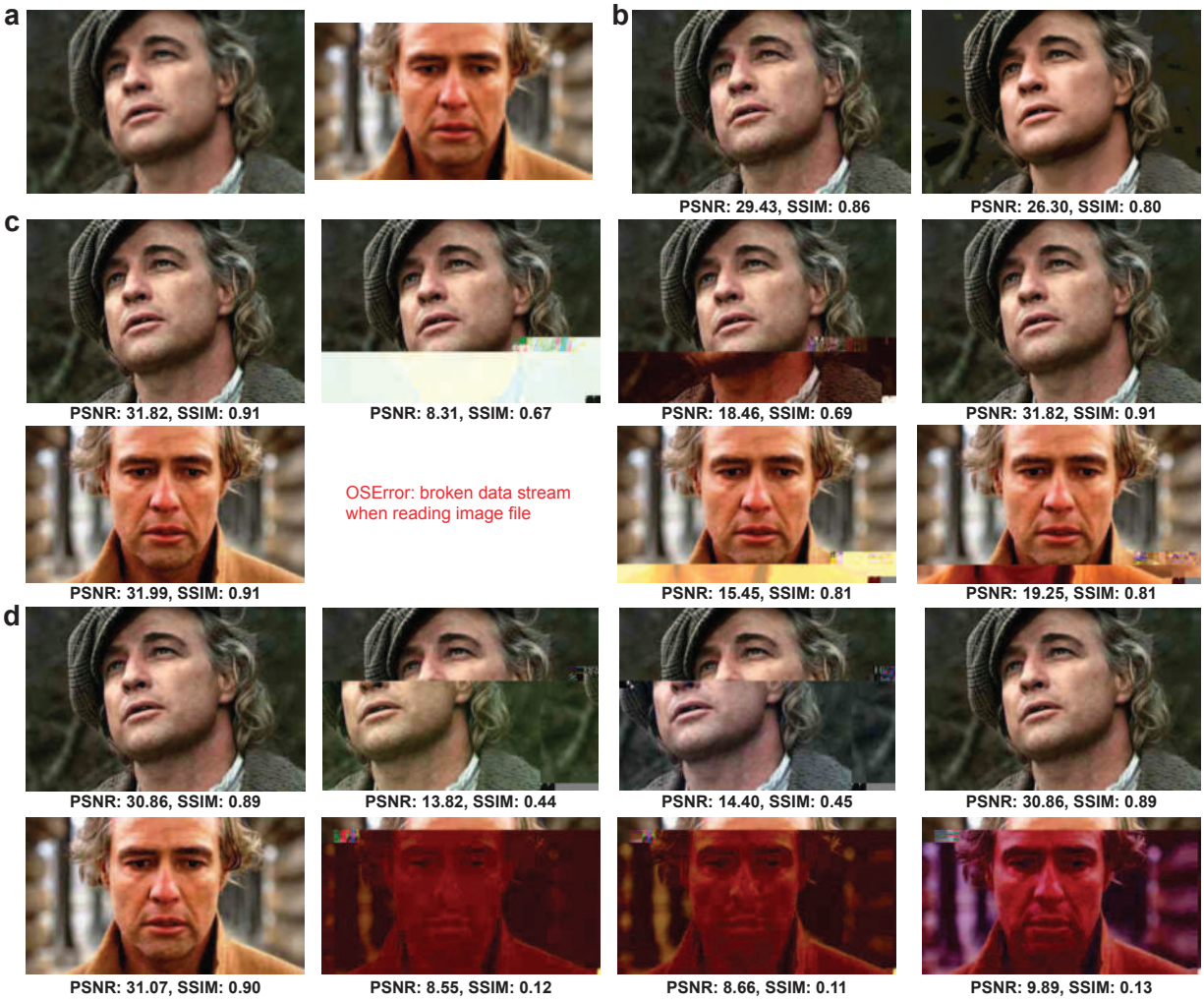


Figure 5. a) The original, uncompressed images. **b**) An example illustrating why single-value metrics for assessing image quality are inadequate. Left: An image compressed by JPEG with quality parameter 20. Right: The same image, after simple 3-bit quantization and image enhancement[33]. The image on the right is visually superior yet has consistently worse numerical quality metrics compared to the image on the left. **c**) Images are compressed with JPEG quality parameter 40 and encoded with an LDPC code of rate R=0.125. The two images in the first column: No errors are added, so the decoding procedure is successful. Images in the remaining three columns represent pairs of decoded images with LDPC channel error rate parameters (probabilities) 0.5%, 1%, and 5%, respectively; in these cases, the base substitution error equals 0.8%, the missing oligo error rate equals 0.7%, resulting in an overall bit error rate of 1.9%. The channel parameter for LDPC decoding is assumed not to be known beforehand. **d**) Images compressed with JPEG quality parameter set to 30 and encoded with an LDPC code of rate R=0.108. The two images in the first column: No errors are added, so the decoding procedure is successful. Images in the remaining three

columns presenting pairs of decoded images with LDPC channel error rate parameters (probabilities) 0.5%, 1%, and 5%, respectively; in these cases, the base substitution error equals 0.8%, the missing oligo error rate equals 0.7%, resulting in an overall bit error rate of 1.9%. The pictures in this figure are courtesy of: Paramount Pictures, Sony Pictures, MGM Studios, StudioCanal, the Marlon Brando and Rod Steiger estates. The original data are provided in the Source Data file.

For LDPC codes, it is crucial to have good estimates of the channel error probability: LDPC belief propagation decoding performs well in practice but is highly sensitive to incorrect initial log-likelihood ratios, which are functions of the channel error rate[16,17]. Therefore, when using mismatched channel parameters, LDPC decoders can fail to correct all errors, which in turn can lead to corrupted JPEG decoding, as seen in Fig. 5. It is worth pointing out that correlations amongst errors may cause some oligos to be disproportionally affected and others to have barely any errors. To further mitigate this issue, oligo-level redundancy was used[4] before, but here it is replaced by a concatenation of an interleaver and LDPC codes, as interleaving renders errors uncorrelated and helps with missing oligo content reconstruction.

As a proof of concept for storage in the topological dimension, we superimposed information on the same Marlon Brando images (Fig. 7). In the writing experiment, we recorded the word "ILLINOIS," comprising 56 bits in ASCII code, across eight different intensity-level DNA pools. We selected seven nicking endonucleases, each representing one bit of the 7-bit ASCII code. These enzymes have recognition sites that exist in at least one oligo of each of the eight pools, and the sites are used as recording positions. In the ASCII code, '1' translates into inclusion, whereas '0' translates into exclusion of the corresponding enzyme. Upon nicking, the pools are sequenced using the procedure described in Fig. 1d. In this way, the nicked oligos were denatured, resulting in ssDNA fragments of various lengths dictated by the position of the nicks. The fragments were subsequently converted into a dsDNA library and sequenced via Illumina MiSeq. To verify the existence of short-length fragments capped at both ends by enzyme recognition sites, we developed a detection algorithm with a flowchart depicted in Fig. 6. The gist of the algorithm is to detect if a nick was created or not based on a search for two fragments corresponding to the prefix and suffix of the sequences recognized by the enzyme. Note that our algorithm counts the number of appearances of all possible (potential) nicking events for the sets of enzymes used. The decision regarding which enzymes are included in a certain pool is based on the counts of each

prefix-suffix pair. To rewrite the data, we performed the process outlined in Fig. 1c, which involves treatment of the nicked DNA with the T4 DNA ligase. This erasure method completely removed the recorded metadata. Note that the ligase was perfectly effective in so far that each original oligo was accounted for in the sequenced pool. We then rewrote the word "GRAINGER" using the same topological nicking process with error-free reconstruction.

As outlined above, decoding the information stored in the two dimensions requires nontrivial approaches, involving new pattern search algorithms. To hence read the content stored in both dimensions, two separate subpools are retrieved for each level. The sequence content is reconstructed by first sealing the nicks in one of the two subpools via ligation, as done during rewriting, followed by sequencing. Alternatively, to avoid ligation for the sequence content readouts, one may choose to only record the topological information on a subpool of oligos. This resolves the problem of sorting the nicked oligo fragments. The content in the nicks is retrieved using the second subpool. After sequencing, the reads are aligned to the now known full-length reads obtained from the first subpool in which the nicks were sealed. The results of the alignment are used in the algorithmic procedure to determine which enzymes were used for nicking and consequently, for reconstruction of the ownership metadata (Fig. 7).

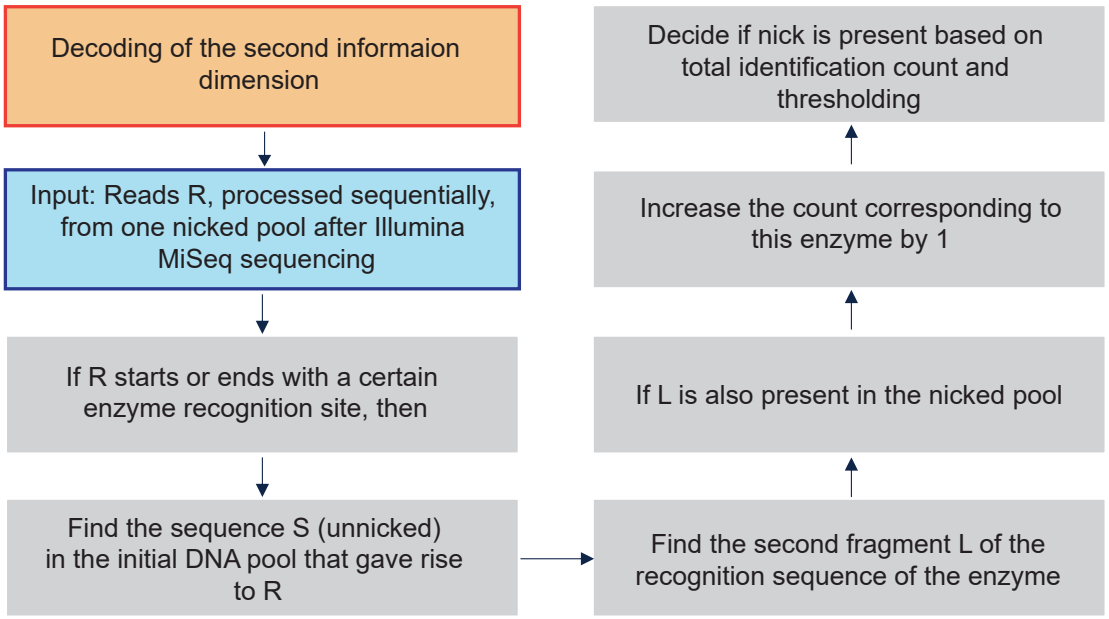


Figure 6. A prefix-suffix pattern search method for decoding the information in the topological dimension. Each fragment obtained from the nicked pool is searched for the presence of a prefix-suffix substring pair that can indicate that a specific enzyme was included in the combinatorial mixture used for the given intensity pool. Since undesired nicking reactions may occur, some counts corresponding to recognition sites of enzymes that were not actually present in the pool may be nonzero; in this case, we make the decision based on how large the counts are relative to others (i.e., we use thresholding with a threshold determined based on the largest count values for the pool).

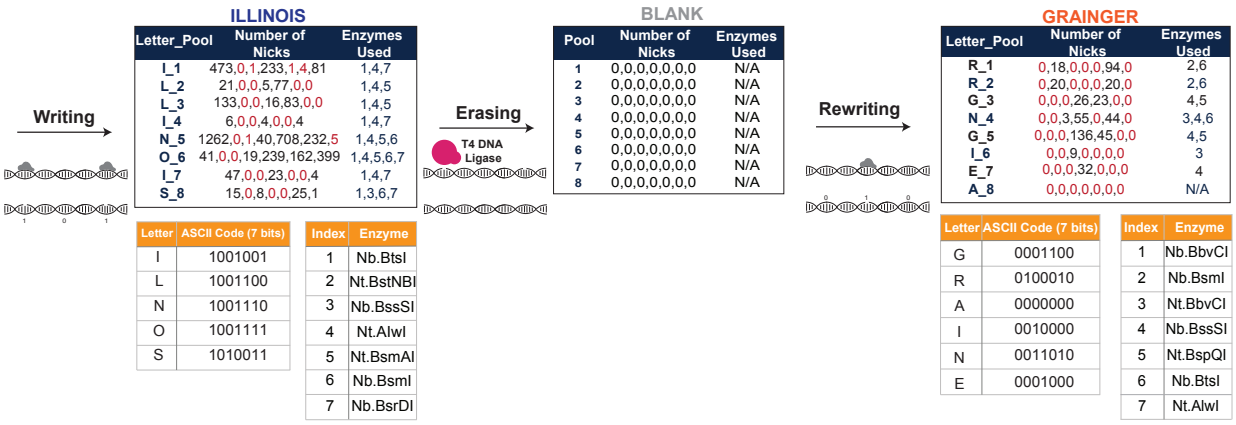


Figure 7. Schematic of metadata encoding and identification using DNA nicking. The numbers in the middle column of the leftmost and rightmost tables represent the number of oligos in the sequenced pool capped by the recognition sequences of the nicking enzymes. The numbers listed in red correspond to the labels of nicking enzymes not used in the encoding of the letter to the left. As may be seen from both tables, the largest red numerical value is significantly smaller than the smallest black value for all encodings (e.g., 4<<81, 5<<40 in the leftmost table) and the second round of writing resulted in no spurious nicks whatsoever. The quality of the results in the rewriting experiment may be attributed to a more suitable choice of nicking enzymes determined upon inspection of the results of the first round. Hence it is recommendable to use the second collection of enzymes for recording purposes. Also note that we shuffled the symbol encodings for the rewriting experiment in order to test more combinations of nicking enzymes. In the erasure step, the T4 DNA ligase was used in a single step reaction to seal all the nicks. No nicks were found after the ligation reaction, showing that the ligase perfectly erased the data (middle table). Note that when recording “ILLINOIS” and “GRAINGER” only six and five enzymes were effectively used for the ASCII code, respectively, due to the choice of the letters in the words.

Existing technologies for DNA synthesis, editing, and sequencing allow for writing and reading diverse information in multiple dimensions or molecular features. Our 2DDNA platform

exploits these tools to enable recording data in two DNA dimensions, including sequence context and backbone structure, thereby opening the door for multidimensional macromolecular storage systems that can use multiple molecular properties (including molecular concentration). Our results show that the 2DDNA system takes advantage of our automatic discoloration detection approach and powerful state-of-the-art deep learning methods for image inpainting and enhancement to substantially improve the quality of the stored images without error-control redundancy. This represents a fundamental advancement in molecular storage which departs from prior techniques in the field and reduces the cost of data storage by greatly minimizing or eliminating the need for synthesizing redundant oligos. The tailor-made learning methods also overcome reliability issues that cannot be addressed by off-the-shelf JPEG compression and joint source-channel coding methods.

Our storage system also offers a simple means for permanently erasing metadata information. The ligation-based approach differs substantially from the existing rewriting methods[5,43]. In the first setting, overlap-extension PCR is used to rewrite blocks of texts corresponding to words. This is a tedious, multi-step approach and much more complex to perform than ligation. In the second approach, one requires additional DNA synthesis and multiple hybridization and strand displacement steps to rewrite the content. Note that in our system metadata is automatically sequenced during the sequencing of the actual image – no separate sequencing for the nick-based information is needed. This is the case since we can always use the strand that is free of nicks as reference for sequence alignment to determine the locations (positions) of the nicks.

For selective amplification and PCR-based random access, the oligos we used to store image content contain carefully designed primers. The primers satisfy Hamming distance, sequence correlation, sequence balance and so-called primer-dimer constraints[44]. Note that once nicks are added to the sugar-phosphate backbone, one cannot run PCR reactions on the oligos directly. To randomly access an image, a certain amount of DNA from the oPools has to be isolated, sealed using the T4 ligase and then amplified via PCR. Consequently, metadata is removed from the selected subpool to enable random access to the image itself, but it remains intact in the global pool of oligos. In order to avoid first sealing the nicks and then running the

PCR, one can also use other methods for random access, involving magnetic beads with attached primers corresponding to the address sequences of the image of interest[45].

Our 2DDNA platform was tested on eight images of total size 1.082MB. The only oligo content that does not correspond to actual raw image information includes primers, pixel/color/image identifiers, constrained redundancy for balancing the GC content and removing long runs of Gs as needed for synthesis. The average sequencing coverage used is 112x, which is small compared to the 3000x coverage reported in[2] and the 370x coverage from[4]. It is higher than the coverage of 5x reported in[15] but in that case, error-control coding redundancy is used. We did not try to optimize the sequencing coverage - our coverage values are dictated by the sequencing protocol used and are not needed for high-quality reconstruction.

The information density of our platform equals the number of bits stored divided by the number of nucleotides used for encoding. Since quantization is used during the encoding procedure, there are two ways to compute this density: If calculated with respect to the number of bits in the raw image files, the information density equals 3.73bits/nt. Clearly, this exceeds the maximum 2bits per nucleotide density dictated by the 4-alphabet size, but may be seen as a consequence of the fact that we get a distorted image back, which allows for an increase from 2 to 3.73bits/nt. If the information density is calculated with respect to the number of bits of the quantized image files, the information density equals 1.40bits/nt. The reason why this value is smaller than 1.57bits/bp reported in[5] and 1.72bits/bp reported in[6], is that in the latter two works gBlocks of length 1000bps were used, while in this work we used oPools of length 196nts. To allow for random access, one has to include primers and address sequences which amount to 53nts per oligo, i.e., per 196 nucleotides – an overhead of 27%. This is to be compared to roughly 50bps per 1000bps[5,6], resulting in a significantly smaller overhead of 5%. When converted into bytes/gram, the two reported densities theoretically equal 0.91 zettabytes/gram and 0.34 zettabytes/gram.

The oPools and corresponding primers were ordered from Integrated DNA Technologies (IDT):
https://www.idtdna.com/pages/products/custom-dna-rna/dna-oligos/custom-dna-oligos/opools-oligo-pools .All oPools were diluted to 5ng/ul. The primers were diluted to 10uM. Each oPool was

amplified in separate reactions using forward and reverse primers for each of the 8 levels. Reactions were set up with 5ng of oPool, 1ul of each forward and reverse primer diluted to 10uM, 22ul of water and 25ul of Kapa HiFi DNA Polymerase (Roche, CA) with the following PCR cycling conditions: denaturation at 98ºC for 45s, 8 cycles of 98ºC for 15s, annealing at 51ºC for 30s and extension at 72ºC for 30s, followed by a final extension at 72ºC for 1min and hold to 4ºC. After PCR, the individual reactions were cleaned up with 50ul of AMPure beads (Agilent, CA) and eluted in 20ul of 10mM Tris. The PCR products were quantitated with the Qubit 3.0 fluorometer and run on a Fragment Analyzer (Agilent, CA) to determine the presence of a band of the correct size and the absence of free primers or primer-dimers. The PCR products from each level were pooled in equimolar concentration and the pool was converted into a sequence-ready library with the Kapa Hyper Library Construction kit (Roche, CA) with no PCR amplification. The final library was quantitated with Qubit and evaluated in a Fragment analyzer and further quantitated by qPCR. The library was loaded on a MiSeq (Illumina, CA) and sequenced for 250 cycles from each end of the library fragments with a Nano V2 500 cycles kit (Illumina). The raw fastq files were generated and demultiplexed with the bcl2fastq v2.20 Conversion Software (Illumina).

All nicked products were purified using the Qiaquick PCR purification kit (QIAGEN) and eluted in ddH_2O. They were then denatured at 98°C for 5min and immediately cooled down to 4°C. The ssDNA samples were first quantified via the Qubit 3.0 fluorometer. Next, the Accel-NGS® 1S plus DNA library kit (Swift Biosciences) was used for library preparation following the manufacturer's recommended protocol. Prepared libraries were quantitated using Qubit and then run on a DNA Fragment Analyzer (Agilent, CA) to determine fragment sizes, pooled in equimolar concentration. The pool was further quantitated by qPCR. All steps were performed for each sample separately and no nicked DNA samples were mixed. The pooled libraries were loaded on an MiSeq device and sequenced for 250 cycles from each end of the library fragments with a Nano V2 500 cycles kit (Illumina). The raw fastq files were generated and demultiplexed with the bcl2fastq v2.20 Conversion Software (Illumina).

References

1. Goda, K. & Kitsuregawa, M. The History of Storage Systems. *Proc. IEEE* **100**, 1433–1440 (2012).

2. Church, G. M., Gao, Y. & Kosuri, S. Next-Generation Digital Information Storage in DNA. *Science* **337**, 1628–1628 (2012).
3. Goldman, N. *et al.* Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* **494**, 77–80 (2013).
4. Grass, R. N., Heckel, R., Puddu, M., Paunescu, D. & Stark, W. J. Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes. *Angew. Chem. Int. Ed.* **54**, 2552–2555 (2015).
5. Tabatabaei Yazdi, S. M. H., Yuan, Y., Ma, J., Zhao, H. & Milenkovic, O. A Rewritable, Random-Access DNA-Based Storage System. *Sci. Rep.* **5**, 14138 (2015).
6. Yazdi, S. M. H. T., Gabrys, R. & Milenkovic, O. Portable and Error-Free DNA-Based Data Storage. *Sci. Rep.* **7**, 5011 (2017).
7. Zhirnov, V., Zadegan, R. M., Sandhu, G. S., Church, G. M. & Hughes, W. L. Nucleic acid memory. *Nat. Mater.* **15**, 366–370 (2016).
8. Cao, C. *et al.* Aerolysin nanopores decode digital information stored in tailored macromolecular analytes. *Sci. Adv.* **6**, eabc2661 (2020).
9. Arcadia, C. E. *et al.* Multicomponent molecular memory. *Nat. Commun.* **11**, 691 (2020).
10. Rosenstein, J. K. *et al.* Principles of Information Storage in Small-Molecule Mixtures. *IEEE Trans. NanoBioscience* **19**, 378–384 (2020).
11. Dimopoulou, M., Antonini, M., Barbry, P. & Appuswamy, R. A biologically constrained encoding solution for long-term storage of images onto synthetic DNA. in *2019 27th European Signal Processing Conference (EUSIPCO)* 1–5 (IEEE, 2019). doi:10.23919/EUSIPCO.2019.8902583.
12. Dimopoulou, M. & Antonini, M. Image storage in DNA using Vector Quantization. in *2020 28th European Signal Processing Conference (EUSIPCO)* 516–520 (IEEE, 2021). doi:10.23919/Eusipco47968.2020.9287470.
13. Cheraghchi, M., Gabrys, R., Milenkovic, O. & Ribeiro, J. Coded Trace Reconstruction. *IEEE Trans. Inf. Theory* **66**, 6084–6103 (2020).
14. Gabrys, R., Kiah, H. M. & Milenkovic, O. Asymmetric Lee Distance Codes for DNA-Based Storage. *IEEE Trans. Inf. Theory* **63**, 4982–4995 (2017).
15. Chandak, S. *et al.* Improved read/write cost tradeoff in DNA-based data storage using LDPC codes. in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* 147–156 (IEEE, 2019). doi:10.1109/ALLERTON.2019.8919890.
16. Savin, V. Self-corrected Min-Sum decoding of LDPC codes. in *2008 IEEE International Symposium on Information Theory* 146–150 (IEEE, 2008). doi:10.1109/ISIT.2008.4594965.
17. Summers, T. A. & Wilson, S. G. SNR mismatch and online estimation in turbo decoding. *IEEE Trans. Commun.* **46**, 421–423 (1998).
18. Gallager, R. Low-density parity-check codes. *IEEE Trans. Inf. Theory* **8**, 21–28 (1962).
19. Tabatabaei, S. K. *et al.* DNA punch cards for storing data on native DNA sequences via enzymatic nicking. *Nat. Commun.* **11**, 1742 (2020).
20. Kumar, V. & Milenkovic, O. On Unequal Error Protection LDPC Codes Based on Plotkin-Type Constructions. *IEEE Trans. Commun.* **54**, 994–1005 (2006).
21. Laver, T. *et al.* Assessing the performance of the Oxford Nanopore Technologies MinION. *Biomol. Detect. Quantif.* **3**, 1–8 (2015).

22. Hilbert, D. Über die stetige Abbildung einer Linie auf ein Flächenstück. in *Dritter Band: Analysis · Grundlagen der Mathematik · Physik Verschiedenes* 1–2 (Springer Berlin Heidelberg, 1935). doi:10.1007/978-3-662-38452-7_1.
23. Gray, R. M. *Source Coding Theory*. vol. 83 (Springer US, 1989).
24. Huffman, D. A Method for the Construction of Minimum-Redundancy Codes. *Proc. IRE* **40**, 1098–1101 (1952).
25. Wang, B., Chalk, C. & Soloveichik, D. SIMD||DNA: Single Instruction, Multiple Data Computation with DNA Strand Displacement Cascades. in *DNA Computing and Molecular Programming* (eds. Thachuk, C. & Liu, Y.) vol. 11648 219–235 (Springer International Publishing, 2019).
26. The Godfather. https://www.pinterest.com/pin/188025353183000993/.
27. Pan, C. *et al.* Image Processing in DNA. in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 8831–8835 (IEEE, 2020). doi:10.1109/ICASSP40776.2020.9054262.
28. Yu, J. *et al.* Free-Form Image Inpainting with Gated Convolution. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4470-4479, doi: 10.1109/ICCV.2019.00457.
29. Yeh, R. A. *et al.* Semantic Image Inpainting with Deep Generative Models. in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 6882–6890 (IEEE, 2017). doi:10.1109/CVPR.2017.728.
30. Nazeri, K., Ng, E., Joseph, T., Qureshi, F. & Ebrahimi, M. EdgeConnect: Structure Guided Image Inpainting using Edge Prediction. in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* 3265–3274 (IEEE, 2019). doi:10.1109/ICCVW.2019.00408.
31. Tomasi, C. & Manduchi, R. Bilateral filtering for gray and color images. in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)* 839–846 (Narosa Publishing House, 1998). doi:10.1109/ICCV.1998.710815.
32. Narendra, P. M. A Separable Median Filter for Image Noise Smoothing. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-3**, 20–29 (1981).
33. Wan, Ziyu, et al. Bringing old photos back to life. in proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020.
34. The Wild One. https://www.pinterest.com/pin/479985272788240268/.
35. Marlon Brando Publicity Still. https://commons.wikimedia.org/wiki/File:Marlon_Brando_by_Edward_Cronenweth,_1955_(b_w).jpg.
36. On the Waterfront. https://2.bp.blogspot.com/-NKpsNd26yJk/TWqUncf4jrI/AAAAAAAAIa4/RQDHhTEtBnk/s1600/aaaaon8.jpg.
37. The Nightcomers. https://www.pinterest.com/pin/755619643712244163/.
38. A Streetcar Named Desire. https://commons.wikimedia.org/wiki/File:Brando_-_Leigh_-_1951.jpg.
39. Last Tango in Paris. http://thecinemaarchives.com/wp-content/uploads/2018/07/marlon-brando-the-last-tango-in-paris.jpg.

40. Apocalypse Now. https://i2.wp.com/macguff.in/wp-content/uploads/2017/09/Apocalypse-Now-Movie-Still-2.jpg.
41. Chandak, S. *et al.* Overcoming High Nanopore Basecaller Error Rates for DNA Storage via Basecaller-Decoder Integration and Convolutional Codes. in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 8822–8826 (IEEE, 2020). doi:10.1109/ICASSP40776.2020.9053441.
42. Fei, P. & Wang, Z. LDPC Codes for Portable DNA Storage. in *2019 IEEE International Symposium on Information Theory (ISIT)* 76–80 (IEEE, 2019). doi:10.1109/ISIT.2019.8849814.
43. Chen, K., Zhu, J., Bošković, F. & Keyser, U. F. Nanopore-Based DNA Hard Drives for Rewritable and Secure Data Storage. *Nano Lett.* **20**, 3754–3760 (2020).
44. Tabatabaei Yazdi, S. M. H., Kiah, H. M., Gabrys, R. & Milenkovic, O. Mutually Uncorrelated Primers for DNA-Based Data Storage. *IEEE Trans. Inf. Theory* **64**, 6283–6296 (2018).
45. Kojima, T. PCR amplification from single DNA molecules on magnetic beads in emulsion: application for high-throughput screening of transcription factor targets. *Nucleic Acids Res.* **33**, e150–e150 (2005).

Part II: In-Memory Computing on Data stored in DNA

May 17, 2022

Contents

1 Single-Instruction Multiple-Data: SIMD-DNA

Ever since Watson and Crick first described the molecular structure of DNA, its information-bearing potential has been apparent to computer scientists. With each nucleotide in the sequence drawn from the four-valued alphabet of $\{A, T, C, G\}$, a molecule of DNA with n nucleotides stores 4^n bits of data. Indeed, this information storage underpins life as we know it: all the instructions on how to build and operate a life form are stored in its DNA, honed over eons of evolutionary time. In Part I, we discussed our work on DNA *storage* systems. Storage is, of course, only half the equation. The transformative aspect of DNA is that it is *programmable* – meaning that one can effect computation of data stored in this medium. Here we discuss our approach to computation "in-memory", that is to say modifying the data stored in DNA directly.

Beginning with the seminal work of Adelman in 1994 [1], DNA computing has promised the benefits of massive parallelism in operations. However, it is fair to say that in the three decades since, the practical impact of research in this field has been modest. Operations are typically performed on the *concentration* of DNA strands in solution. For instance, with DNA strand displacement cascades, single strands displace parts of double strands, releasing single strands that can then participate in further operations [2, 3, 4]. The inputs and outputs are the concentration values of specific strands. With a focus on concentration, the application space for this research has been limited to computing that is embedded in chemical systems.

A practical DNA storage system, particularly one that is inherently programmable such as our platform, changes this. The scheme that we discussed in Part I operates on data stored not in the sequence of nucleotides, but rather in topological modifications to the strands: breaks in the phosphodiester backbone of DNA that we call "nicks" and gaps in the backbone that we call "toeholds."

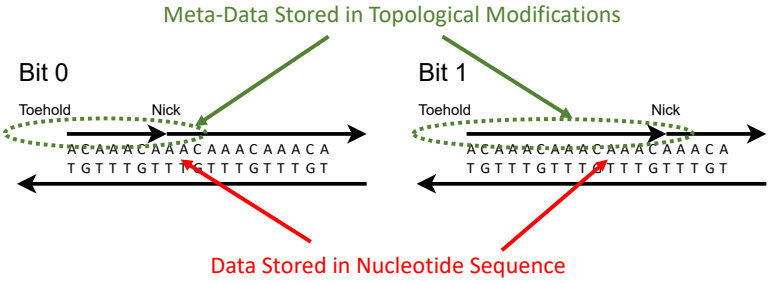


Figure 1: Data is stored in multiple dimensions. The sequence of nucleotides stores data in the form of the A's, C's, T's, and G, with 2 bits per letter. Superimposed on this, we store data via topological modifications to the DNA, in the form of nicks and exposed toeholds. This data is **rewritable**, with techniques developed for DNA computation.

Note that the data that we operate on with this form of DNA computing is encoded in a different dimension than the data encoded in the sequence data of the DNA. The **underlying data** – perhaps terabyte's worth of it – is stored as the sequence of A's, C's, T's, and G's in synthesized strands. Superimposed on this, we store **metadata** via topological modifications. This is illustrated in Fig. 1. This metadata is rewritable with the techniques that we describe here. Accordingly, it fits the paradigm of "in-memory" computing [5]. The paradigm that we will deploy, dubbed "SIMD||DNA", was recently introduced by PIs Soloveichik [6], with follow-on work by PI Riedel [7].

1.1 SIMD||DNA structure

Funded by this DARPA grant, we proposed a new paradigm called SIMD||DNA (Single Instruction Multiple Data[1] DNA) which integrates DNA storage with in-memory computation. As shown in Figure 4A, unlike traditional DNA data storage where information is encoded in the nucleotide sequence, SIMD||DNA encodes information in a register, a multi-stranded DNA complex with a unique pattern of nicks and exposed single-stranded regions. The paradigm of storing information in nicks rather than in the DNA sequence itself was proposed in [9]. Besides enabling computation, since different data can be stored using different subsets of the same DNA strands, this data representation scheme can potentially also reduce the cost [9]. Although storing information in the location of nicks rather than the sequence reduces the storage density somewhat (factor of about 30, see Discussion), the density remains many orders of magnitude higher than competing magnetic and optical technologies.

Our implementation of SIMD provides a means to transform stored data, perhaps large amounts of it, with a single parallel instruction. We divide stretches of double-stranded DNA into "domains", where each domain is a contiguous sequence of nucleotides of some small specified length (typically 5 to 20). A sequence of several (typically 5 to 7) domains maps to a "cell" storing one binary bit. Whether a cell stores a 0 or a 1 depends upon topological variations, specifically the location of "nicks", i.e., breaks in the DNA backbone. The nicks always occur on one strand of a double-stranded complex (generally the top strand in our examples); the other remains untouched. We create these nicks in the assembly process.

Strand displacement is used to implement computation on the stored values. It is predicated on the encoding scheme for data, shown in Figure 2. Each cell stores a single binary value (a "bit"). Each cell consists of 7 domains. We do not specify the actual nucleotide sequence of the domains here for simplicity. While preparing this cell, the top DNA strand must be nicked before and after domain 1. This strand can then be displaced by denaturing, creating an exposed toehold. Domain 1 is always exposed as a toehold in this representation. Domains 2 through 7 are covered. When storing a bit 0, we will nick the top strand between domains 3 and 4; when storing a bit 1, we will nick between domains 5 and 6.

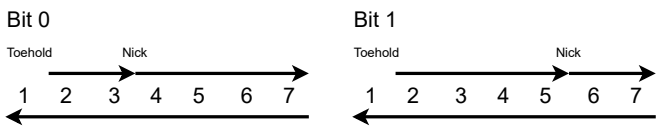


Figure 2: Bit representation in the encoding scheme. Horizontal lines represent DNA strands. Integers represent "domains": specific sequences of nucleotides. Arrowheads represent nicked positions: places where the phosphodiester bond in the backbone of the DNA strand has been broken, via gene-editing techniques. Cells store binary values. Each cell consist of 7 domains. Domain 1 is always exposed, forming a toehold.

The computation is carried out by a sequence of "instructions", where each instruction implements DNA strand displacement reactions on cells. Instructions are initiated by single-

[1]Single instruction, multiple data (SIMD) is one of the four classifications in Flynn's taxonomy [8]. The taxonomy captures computer architecture designs and their parallelism. The four classifications are the four choices of combining single instruction (SI) or multiple instruction (MI) with single data (SD) or multiple data (MD). SI versus MI captures the number of processors/instructions modifying the data at a given time. SD versus MD captures the number of data registers being modified at a given time, each of which can store different information. Our scheme falls under SIMD, since many registers, each with different data, are affected by the same instruction.

stranded "instruction strands" added to the solution. After the strand displacement cascades complete, any single-strand fragments in the solution are washed away; the original strand is kept and separated via a magnetic bead. After a sequence of instructions, the data is transformed to its final state. The readout can be performed via fluorescence or with Oxford nanopore devices [10, 11].

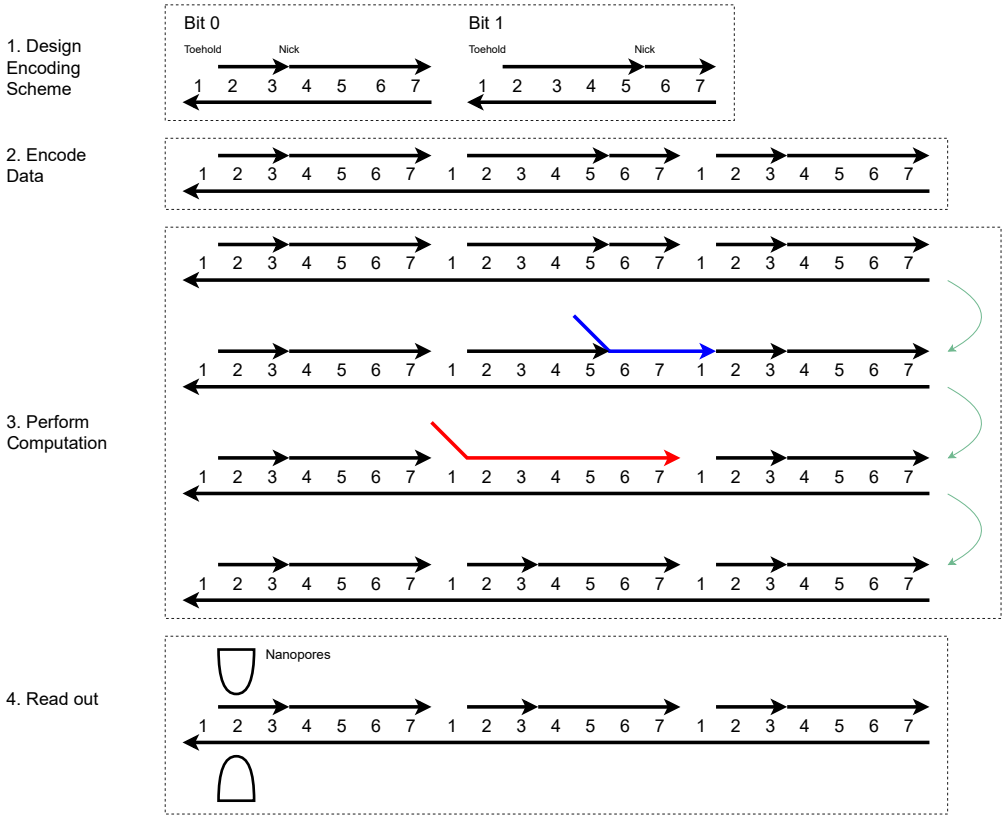


Figure 3: General Outline of SIMD||DNA Computations. Arrowheads represent "nicks": breaks in the DNA backbone, performed with gene editing techniques. Integers represent "domains": contiguous sequences of nucleotides of some small, specified length. For convenience, we use the numbers 1 through 7 repeatedly; however, each copy of a number represents a distinct domain, consisting of a unique nucleotide sequence. Stage 1 shows the encoding of binary bits 0 and 1, based of different locations of toeholds and nicks. Note that domain 1 is always "exposed": the DNA backbone of the top strand is nicked, and the DNA is gently denatured until this segment falls off, exposing a toehold at this domain. Stage 2 shows an example of encoding the bits 010. Stage 3 illustrates the step in which computation is performed with strand displacement, in a general sense. Details of this step will be provided for specific algorithms in later sections. Note that, in this generic example, the location of nick in the second cell has changed at the end of stage 3. Stage 4 illustrates how nanopore sequencing could be used to perform readout.

The general flow of SIMD||DNA computation is summarized as follows and illustrated in Figure 3.

1. Design an encoding structure that best suits the algorithm.

2. Encode the data at specific locations, using enzymes to nick corresponding targets.

3. Gently denature the DNA, allowing segments between adjacent nicks to detach, exposing toeholds.

4. Execute instructions, implemented as strand-displacement operations.

5. Finally, read out data using fluorescence or with nanopore sequencing.

All the registers share the same sequence space, but each is capable of storing and manipulating a different value. To manipulate information, an instruction (a set of DNA strands) is applied in parallel to all registers. The strand composition of a register updates if the applied instruction strands trigger strand displacement reactions [12] within that register; thus, the outcome of the instruction could differ between the registers. Through strand displacement mechanism, the strand composition, patterns of nicks, and exposed single-stranded regions in the registers are changed. Instruction strands are synthesized independently of the data stored in the registers, so that executing an instruction does not require reading the data. After the non-reacted instruction strands and reaction waste are washed away, subsequent instructions can be performed. In the sense that the same set of instruction strands simultaneously operates on all the different registers each storing different information, our scheme computes in a massively parallel manner (Figure 4B). Additionally, SIMD||DNA programs are "doubly-parallel" in the sense that multiple sites within a register can also undergo displacement in parallel.

We built the theoretical framework for SIMD||DNA and experimentally realized two different algorithms. The molecular programs of these algorithms—binary counting and cellular automaton Rule 110—were developed in a conference paper [6]. Binary counting is a fundamental function in computer programming, and Rule 110 is Turing-universal. Additional algorithms were later developed for more efficient Turing machine simulation [13], and specific operations of sorting, shifting, and searching [14] showing that diverse algorithms can be fit to SIMD||DNA.

We subsequently experimentally implemented these programs and demonstrated correct computation for in-memory and parallel computation for a pool of 16 registers encoding all possible 4-bit binary values. To scale up the computational power, we show that the registers can be repeatedly processed prior to read out by conducting multiple rounds of computation. In addition, like a computer's memory, information stored in the SIMD||DNA paradigm can be specifically queried (random access [15]) or erased. A publication of the experimental results is in preparation.

Registers can be constructed using both chemically synthesized DNA and naturally-occurring DNA (i.e. non-genetically modified sequences), further reducing the dependency on custom oligonucleotide synthesis. We show that unmodified kilobase-length M13 phage plasmid provides a large storage space that allows information size to be scaled up, by constructing multiple sub-registers for parallel computation. So far this is the largest strand displacement system using naturally-occurring DNA sequences: Using SIMD||DNA, we implemented 18 distinct strand displacement reactions in solution at the same time, and

in total 122 distinct strand displacement reactions. An important reason for the stunning achievements of DNA origami is that a native M13 template strand is used [16]. We believe that exploring DNA strand displacement on native DNA similarly has profound implications for *dynamic* DNA nanotechnology.

Figure 4 shows another overview of SIMD||DNA. Every register contains a long "bottom" strand and multiple short strands, called *top strands*, bound to the bottom strand. We use *domain* to represent consecutive nucleotides that act as a functional unit. Complementary domains are represented by a star ($*$). The length of the domains is chosen so that: (1) each domain can initiate strand displacement (i.e. can act as a toehold), (2) strands bound by a single domain readily dissociate, and (3) strands bound by two or more domains cannot dissociate. Each bottom strand is partitioned into sets of consecutive domains called *cells* (Figure 4C). Each cell contains the same number of domains. Cells encode information with the binding configuration of their top strands (e.g. lengths, presence or absence of toeholds). For the programs we designed, we used a binary encoding with each cell representing one bit.

Each *instruction* of a program corresponds to the addition of a set of DNA strands at high concentration to a solution containing the registers. The registers are attached to magnetic beads, allowing washing away of beadless non-reacted instruction strands and reaction waste. Registers and instruction strands are allowed to react for a short amount of time before washing such that the high concentration instruction strands interact with the registers, but the low concentration waste products do not. The instruction strands can cause three different types of events (Figure 4D). **Attachment** reactions preserve all the strands originally bound to the register and attach new strands (as long as the new strand binds strongly enough—by two or more domains). The attachment of an instruction strand can lead to a partial displacement of a pre-existing strand on the register. **Displacement** reactions introduce new strands to the register and detach some pre-existing strands. Upon binding to a toehold on the register, the instruction strand displaces pre-existing strands through 3-way branch migration. Toehold exchange reactions are favored towards displacement by the instruction strand since they are added at high concentration. Two instruction strands can also cooperatively displace strands on the register. **Detachment** reactions detach pre-existing strands without introducing new strands to the registers. An instruction strand that is complementary to a pre-existing strand with an open overhang can use the overhang as a toehold and pull the strand off the register.

To experimentally implement SIMD||DNA, considerations for readout need to be incorporated in the design (Figure 4E). To read out parallel computation results where registers share the same sequence space, registers with different initial values are given their own barcode sequences. Since information is encoded in the pattern of the top strands, direct readout requires obtaining the location of nicks. To read out the stored information through sequencing while preserving the desired computation logic, we modified the encoding by introducing mismatches between the top strands and the register in a manner that can coexist with the nick-based encoding. Since mismatches can affect strand displacement kinetics [17], the mismatch locations are carefully chosen to ensure that the desired strand displacement reaction is favorable. This secondary encoding allows us to read out the data stored in a heterogeneous pool of registers after ligating the nicks, PCR amplifying the products, and applying next generation sequencing (NGS). The resulting NGS reads, which correspond to

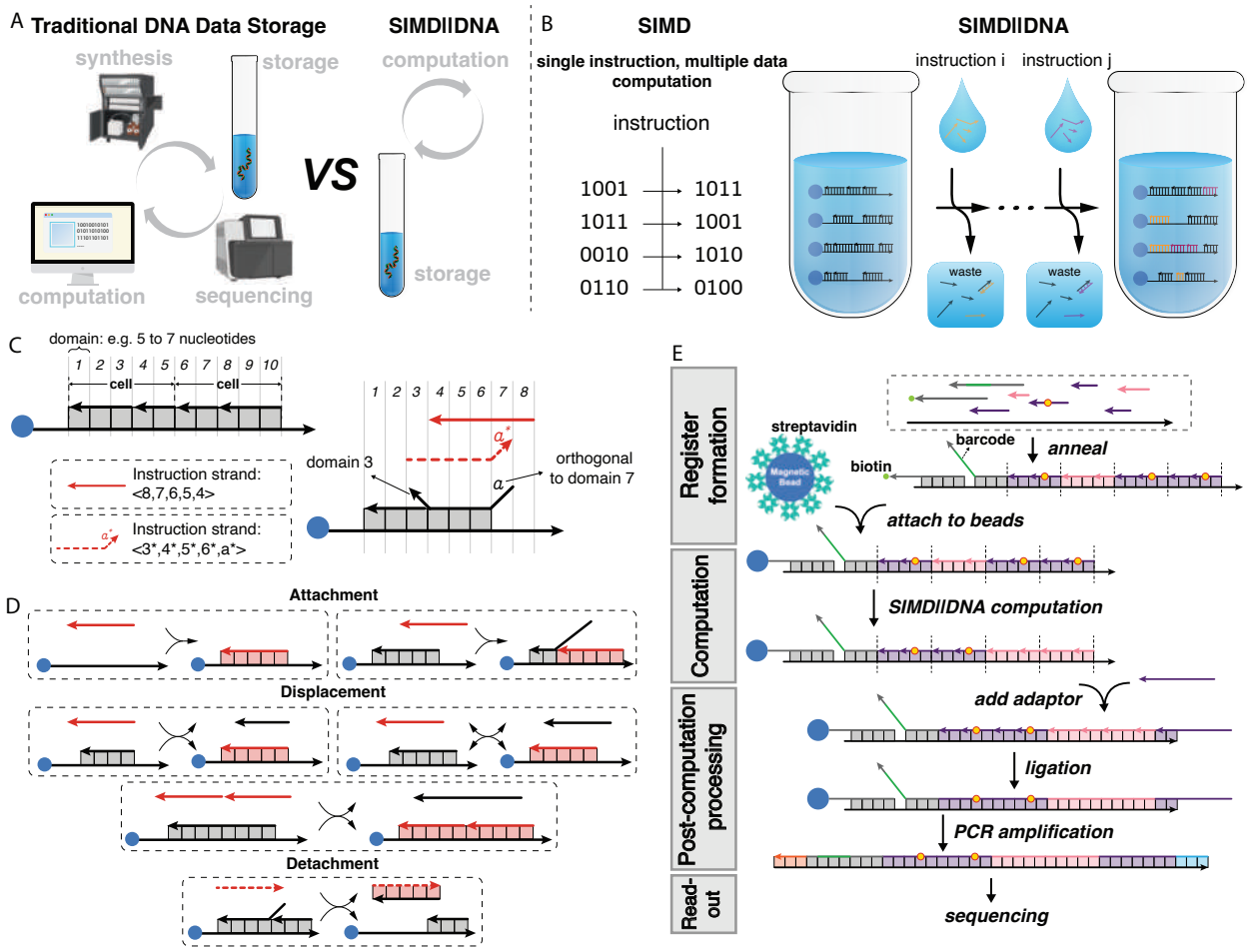


Figure 4: Overview of SIMD||DNA. (A) Computation in traditional DNA storage paradigm relies on outsourcing computation process on classical computer with additional required steps of sequencing and synthesis. The SIMD||DNA paradigm allows in-memory computation performed by DNA itself. (B) Analog to the single instruction multiple data (SIMD) computation in classical computer which enables processing multiple data by one single instruction, the SIMD||DNA paradigm can also perform parallel computation on multiple registers simultaneously. Each DNA register is a multi-stranded complex. Different information is encoded in the pattern of nicks and exposed single-stranded regions in the register. Registers are attached to magnetic beads (blue). At each instruction, a set of instruction strands is added to the solution to react with all the registers in parallel. Then waste species (unreacted instruction strands and displaced reaction products) are washed away. After a series of sequential reaction and washing steps, registers update their information accordingly. (C) The notations for SIMD||DNA. Domains are represented by square boxes. We indicate complementarity of instruction strands to register domains by vertical alignment. If a domain label is given explicitly (e.g. a and a^*), the domain is orthogonal to the other vertically aligned domains A strand can be described by listing the constituent domains in a bracket <> from 5'-end to 3'-end. Strands with solid lines are complementary to the corresponding domains in the bottom strand. Strands with dashed lines are complementary to the corresponding domains in the top strand. A dashed instruction strand indicates the domains in the instruction strand are complementary to other vertically aligned domains. (D) Three types of events can occur when registers react with instruction strands: attachment, displacement and detachment. (E) Experimental workflow. Registers are first assembled, and then undergo computation. The post-computation process including ligation and PCR amplification allows computation products to be further sequenced. Mismatches are labeled as yellow dots.

proportionally amplified computation products, each encodes a 4-bit value and collectively represent the output of the computation. Like in regular DNA storage, this readout method is destructive; however, a small sample can be taken, leaving most of the solution intact.

1.2 Binary Counting Program

We first start with the binary counting program: beginning from arbitrary initial counts stored in different registers, each computation step increments all the registers in parallel. Compared to counting in electrical circuits at the hardware level, where complicated modules are required (a full adder requires at least 2 *XOR* gates, 2 *AND* gates and an *OR* gate—18 transistors total), binary counting in SIMD||DNA requires only 7 instruction steps independent of the size of input. Binary counting requires changing all 1s to 0 starting from the least significant (rightmost) bit to more significant bits until the first 0, and changing that 0 to 1. All bits to the left of the rightmost 0 remain the same. As shown in Figure 5, the SIMD||DNA program encodes states 0 and 1 by two different sets of top strands. One extra domain is included to the right of the rightmost cell which is used to initiate displacement. Starting from the rightmost domain, the program erases all 1's in between the rightmost cell and the rightmost state-0 cell (Instructions 1 and 2), and changes those cells to 0 at Instructions 4 and 5. The rightmost state-0 cell is first marked (Instruction 3), and then changed to state 1 (Instructions 6 and 7). Note that the binary counting program requires a strand displacement cascade (Instructions 1) and the depth of the cascade is dependent on the number of consecutive 1's to the right of the rightmost 0.

To further reduce SIMD||DNA's dependence on artificially designed long oligonucleotides as bottom strands, we chose to assemble registers using the M13mp18 single-stranded DNA plasmid from the M13 bacteriophage, without modifications to the original M13 sequence. Phosphoramidite synthesis, currently the golden standard for *de novo* synthesis of single-stranded oligonucleotides, becomes increasingly error-prone as a function of strand length. On the other hand, naturally-derived DNA is ensured to have both high fidelity and high quality DNA as a result of biological error-correcting mechanisms. The single-stranded M13 bacteriophage plasmid is a staple of DNA nanotechnology that has been widely used as scaffolds for DNA origami [16]; similarly, it could potentially accommodate computation with SIMD||DNA on several hundreds of bits. Despite these advantages, naturally-occurring DNA is typically not used in strand displacement due to the potential for undesired sequence complementarity. While artificially designed sequences can be optimized to minimize secondary structure (e.g., using a 3-letter alphabet [18], computational tools like NUPACK sequence designer [19], or other tools [18, 20]), naturally-occurring DNA may contain thermodynamically stable secondary structures that trigger undesired spurious interactions or prevent desired displacement from completing and ultimately producing incorrect computation results.

Rather than designing the sequence, we pursue the use the naturally-occurring DNA without a heavy load of sequence optimization: We screened different regions on the M13mp18 plasmid for viability by first eliminating areas with undesirable secondary structures (specifically, G-quadruplexes and hairpins [A] [16]) from consideration and then selecting 9 random addresses as candidates at which we encoded sub-registers (Figure 5B). We tuned the domain strength and categorized the encoded registers according to the binding strengths of some

domains: weak (sub-registers 1 through 3), medium (sub-registers 4 through 6) and strong (sub-registers 7 through 9). Each category is expected to react at different experimental conditions as a result of the domain strength; for example, the registers with strong binding strength are expected to require higher temperature or longer reaction time. We tested initial values 0010 and 0011 with different reaction temperatures on these 9 sub-registers, and then picked 5 for further experiments.

We first performed SISD (single instruction single data) computation on sub-register 8 for each of the 16 4-bit initial values. Each test tube contained only one initial value of sub-register 8. After NGS sequencing, sequencing reads were organized according to the barcode sequences associated with their encoded initial values, and the percentage of reads representing the correct value was calculated. More than 90% of the registers can be successfully assembled, processed, and sequenced. After a round of binary counting computation, sub-registers affiliated with all 16 initial values show the correct output as the dominant output (Figure 5C), with the minimum correct percent at 68%. We observed similar results for sub-register 3. We then performed SIMD computation on sub-register 8 by pooling registers with all 16 initial values in the same test tube for computation . Figure 5D shows that all the initial values were updated correctly, with the minimum correct ratio at 60%. After testing different incubation temperatures, we achieved similar computation results on sub-register 7, and 9 at a higher temperature.

We then investigated the ability to store and compute data on multiple registers simultaneously with SIMD||DNA. We tested parallel computation on multiple sub-registers assembled on M13. Each M13 molecule was assembled with both sub-registers 7 and 9 at the temperature compatible to both. For each step of the computation, instruction strands for both sub-registers were applied simultaneously. As shown in Figure 5E, most registers produced the highest readcount for the correct output, with the minimum correct ratio at 15%. This reduction of yield could be due to spurious cross-talk between the instruction strands for sub-registers 7 and 9. Additionally, as the success of computation is dependent on experimental conditions, this reduced accuracy may also stem from operating at a sub-optimal temperature for each register as a compromise for compatiblity.

1.3 Rule 110 Program

In addition to binary counting, we also implemented a program that simulates elementary cellular automaton (CA) Rule 110. An elemental cellular automaton [21], one of simplest models of computation, consists of an infinite set of cells with two states, 0 or 1. At each time step, updates to a cell depend on the states of its left and right neighbors. A simple two-rule characterization of Rule 110's transition rule is as follows: 0 updates to 1 if and only if the state to its right is a 1, and 1 updates to 0 if and only if both neighbors are 1. Critically, Rule 110 has been shown to be Turing universal [22].

Theoretically, SIMD||DNA's in-memory computation model is as powerful as any other space-bounded computing technique [6]. In other words, our space-bounded simulation of Rule 110 immediately gives that any computable function can be computed by a SIMD||DNA program, if the required space is known beforehand. Note that the Rule 110 simulation

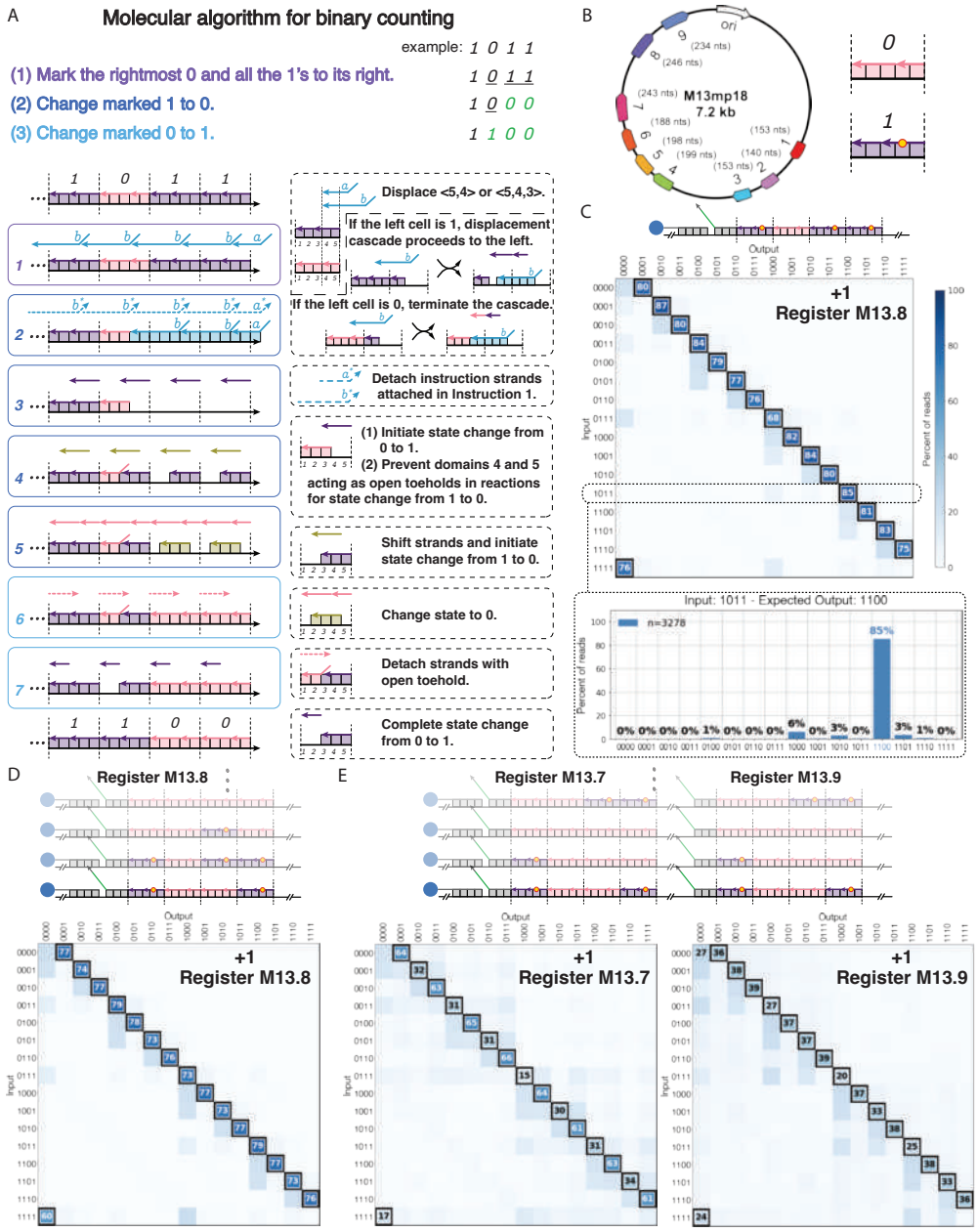


Figure 5: Binary counting program on naturally-occurring sequences. (A) Molecular program implementing addition by 1 of a binary string on an example register. The top register shows the initial state of each cell. After 7 instructions, the register updates to the state shown at the bottom. Strand colors have three information categories: state 1 (purple), state 0 (pink), intermediates (other colors). Solid boxes show the instruction strands and the state of the register before the strands are applied. Dashed boxes explain the logical meaning of the instructions. The overhang domains a and b are orthogonal to their vertically aligned domains. (B) (left) Locations of registers on the M13mp18 phagemid. (right) Mismatches (labeled as yellow dot) are introduced at one of strands representing state 1. (C) Single data binary counting on register M13.8. For each initial value, the distribution of the output values are represented in the heat-map matrix. Lower bar plot shows an example of the data in one row of the heat map: the distribution of output values on reads associated with initial value "1011". (D) Multiple data Binary Counting on register M13.8. (E) Multiple data Binary Counting on register M13.7 and M13.9 in parallel. In all the heat maps, the correct output value is indicated by a white and black border; values that appear in $> 25\%$ of all reads for a given sample are marked by text.

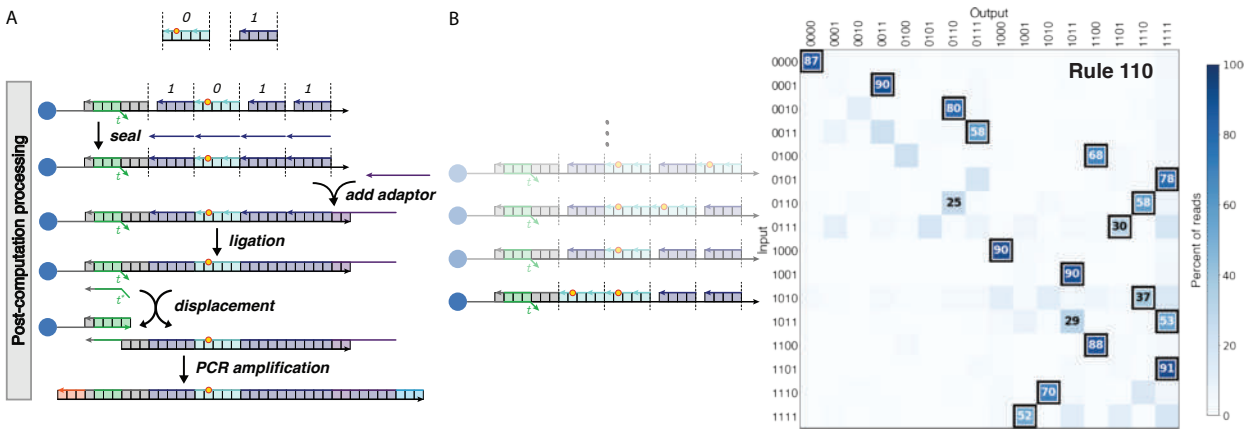


Figure 6: Rule 110 computation with chemically synthesized DNA. (A) Post-computation process for the Rule 110 program with chemically synthesized DNA. After computation, a set of "seal" strands are added to the register to fill in the gap for cells representing bit 1 for the following ligation step. (B) Multiple data Rule 110 computation on 16 registers with unique initial values. The correct output value is indicated by a white and black border; values that appear in $> 25\%$ of all reads for a given sample are marked by text.

invokes two sources of parallelism: instruction strands are applied to all registers in parallel, and every cell within a register can update concurrently. This contrasts with binary counting where instruction 1 requires a cascade of strand displacement reactions across multiple cells.

To experimentally implement the Rule 110 program, we used M13 sequences as well as artificially designed sequences. Since the encoding of information 1 contains an exposed region, to enable ligation and sequencing, a set of "seal" strands were applied to all the registers after performing parallel computation on all 16 initial values to fill in the gaps on the patterns of the top strands (Figure 6A). We confirmed that the Rule 110 program updated correctly for the 16 registers encoded with artificially designed sequences–the correct values are the dominant output (Figure 6B). We achieved similar results using the native M13 sequence as well (not shown).

1.4 Random Access

A related desired functionality for DNA data storage is to be able to selectively address or read out a specific subset of data registers, a process commonly referred to as random access. Random access avoids reading out everything at once, thereby destroying all data. Traditional DNA storage uses PCR to selectively amplify data [23] or selectively pull out information by tuning the binding affinity between sequences [24]. However, designing sequences or multiplexed orthogonal PCR probes with high specificity can be challenging. Additionally, it is necessary to reconstruct the database for information update after if a single piece of data is read. On the other hand, strand displacement achieves specificity through kinetically and energetically favorable reactions that displace a pre-existing strand. In SIMD||DNA, every register is prepared with unique barcode sequences corresponding to different initial values; these sequences can serve as a point of access for specific registers. Another feature of random access is that it allows selective erasure. Accessing data can selectively destroy a subset of the database (data erasure) but leaves the remainder available for further computation.

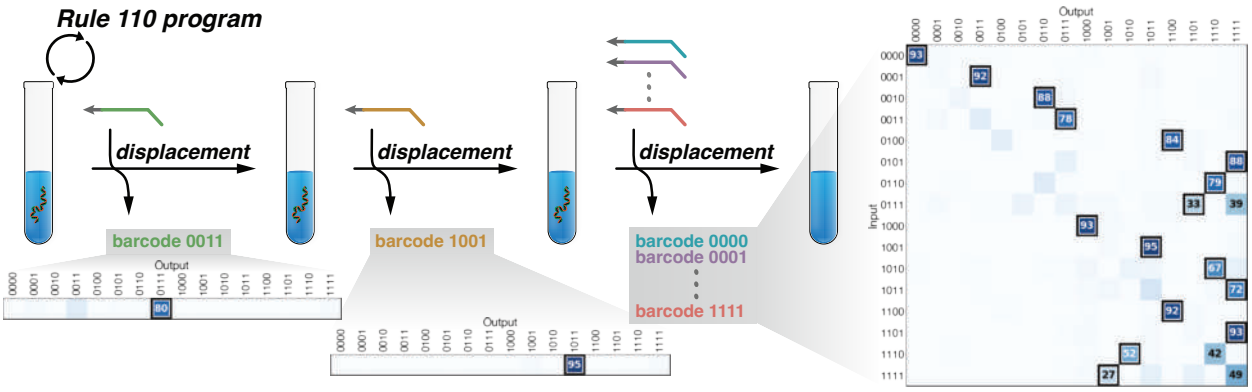


Figure 7: Random access with chemically synthesized DNA. Sequential random access for the Rule 110 algorithm. Following Rule 110 computation, registers with initial value "0011" were accessed first (top), "1001" second (middle), and all remaining values last (bottom).

Instead of reconstructing the database, a new, edited register can simply be added to a previously-accessed database as an update. In principle, in SIMD||DNA programs, after computation on multiple registers, displacement strands with unique barcode sequences can be added to the solution to release registers with the matching barcodes from magnetic beads. Thus, every register can be queried separately for read out from the register mix.

We experimentally demonstrated parallel computation and random access of both the Rule 110 and the binary counting programs. We show that registers can be sequentially accessed by adding a series of different displacement strands with distinct barcodes (Figure 7A). We mixed all 16 registers to perform Rule 110 computation. After computation, we first added a displacement strand with a barcode corresponding to 0011 and processed the displaced registers (ligation, PCR amplification, sequencing). Next, we added another displacement strand with a barcode corresponding to 1001 to query the second register. Finally, we added all 16 different displacement strands (corresponding to all 16 barcodes to access all of the information. The sequencing results confirmed that, for the first and second queries, the desired register is the dominating register among the registers displaced from the mix.

Registers can be accessed in parallel by adding different displacement strands to different register mixes at the same time. All the queries were successful and at least 23% of registers show the correct value. Accessing a register also performs selective erasure of the data. Following displacement of one specific register, we added all 14 displacement strands to displace the remaining data from the register mix. We observed that reads corresponding to the displaced register were notably less abundant compared to reads corresponding to all other registers.

1.5 Sequential computation

Finally, we scaled up the computational power of SIMD||DNA through sequential computation. We began with the Rule 110 program (Figure 8A) and prepared 4 sets of register mix containing 5 distinct registers, each encoding a unique initial value. Each set went through one of the

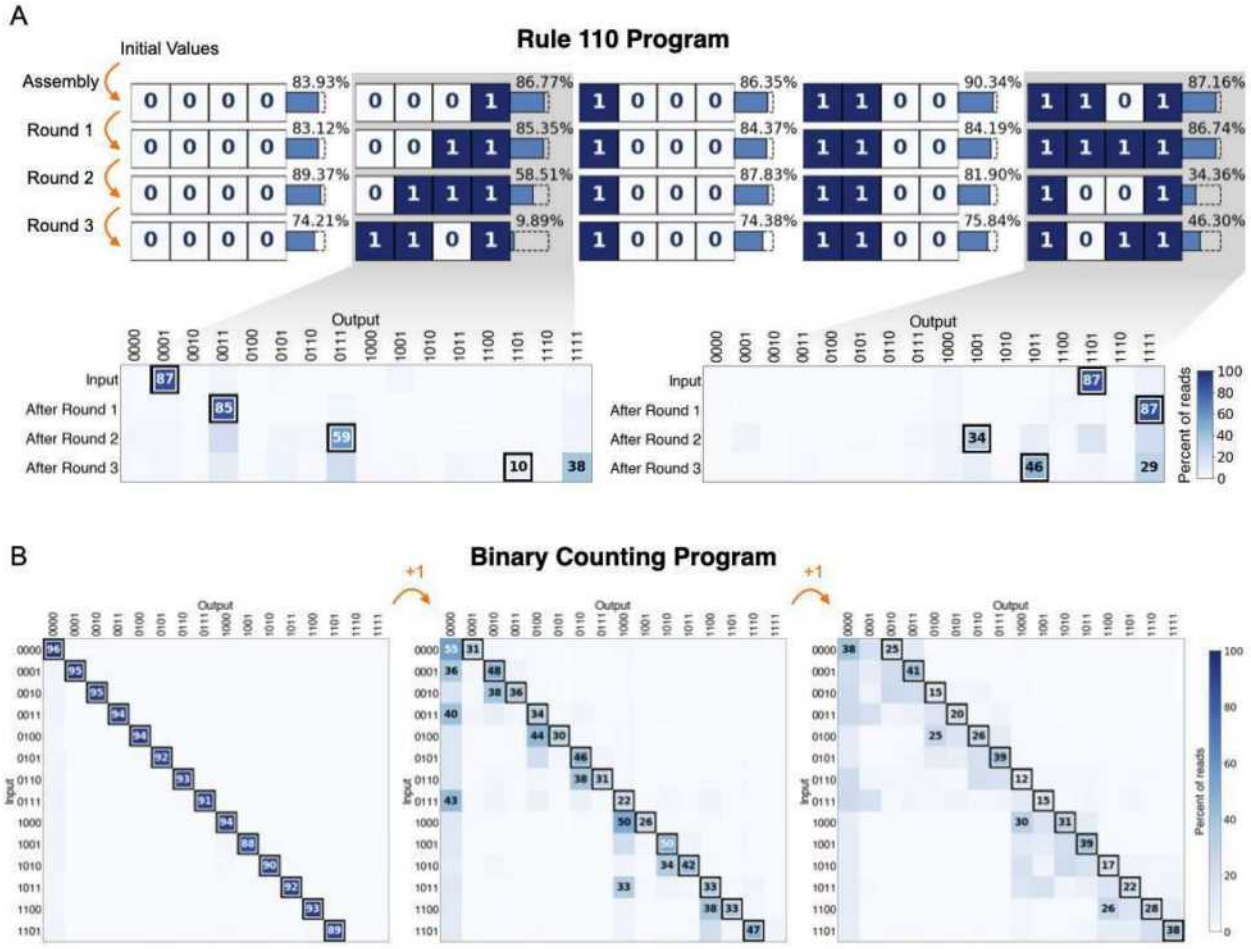


Figure 8: Multiple rounds of sequential computation with chemically synthesized DNA. (A) Sequential computation of the Rule 110 program. Results are normalized to the total read count for each sample. Reads with one or more indeterminate digits were excluded. Lower panels show the distribution of outputs values for initial values 0001 and 1101. (B) Sequential computation of the binary counting program. In both the lower panels of (A) and (B), the correct value is indicated by a white and black border; values that were either the correct value or observed in $> 25\%$ of all reads are labeled.

following processes: no computation, one round of computation, two rounds of computation, and three rounds of computation. After these processes, all registers from the register mix were ligated, displaced from magnetic beads, PCR amplified, and sequenced. In the first round of computation, we confirm that all initial values included in the register mix produced the correct value as the dominant output, with the correct value encompassing at least 83% of all reads. In the second round of computation, all initial values again achieved the correct value as the dominant output, with the correct value represented in at least 34% of all reads. In the final round, all but one initial value produced the correct value as the dominant output; for this initial value, the correct value was observed in approximately 10% of all reads.

For the binary counting program, we first prepared 7 sets of register mix containing all 16 registers (Figure 8B, left panel). One set did not go through any computation and served as a control. The other 6 sets initially went through one round of computation. As part of another experiment, a different register was random accessed (and therefore erased) from each set (results in Figure 7). For 3 of the 6 sets, all remaining registers were displaced and sequenced, and the analyzed results were pooled together to account for the missing registers (Figure 8B, middle panel). The other 3 sets were subjected to another round of computation, followed by access of all remaining registers, post-computation processing, and sequencing. Likewise, the two-round computational results of these 3 registers were pooled in our analysis (Figure 8B, right panel). After the first round of computation, the correct value was represented in at least 22% of all reads for each initial value; following the second round of computation, the correct value was present in at least 12% of all reads.

We investigated the limit of multi-round computation by quantifying product loss after each round of computation using both qPCR and electrophoretic techniques. One round of computation (washing, strand displacement, displacement from magnetic beads, ligation) yields about 38%, while washing alone (without instruction strands added for computation) leaves about 56% of product. Approximately 70% of the product loss is estimated to result from bead loss during washing and 30% by imperfect ligation of the top strands and strand displacement. Theoretically, using the same protocol, we can perform up to 38 rounds when storing registers with 16 different values and up to 26 rounds when storing registers with 16,000 different values. This indicates that an improved washing techniques may drastically increase the yield and consequently the maximum rounds of computation.

1.6 Identifying Bit Pairs

Before describing the implementation of specific algorithms for sorting, shifting, and searching, we will present some general algorithmic steps useful in implementing all of these.

A common task in our algorithms is "identifying" pairs of adjacent bits, i.e., recognizing the specific pair of cells at a location of interest. We will exploit the fact that domain 1 is always exposed to identify these specific pairs. Figure 9 illustrates our approach on the string 11001, which contains all 4 possible adjacent pairs: $00, 01, 10$ and 11.

Identification is performed with three instructions. In instruction 1, the strands (S_1 6 7 1 2 3) are issued to all pairs of bits. Through the toehold at domain 1 between each pair, the strand S_1 binds to domains 6, 7, 1 in the pair $(1, 1)$, leaving domains S_1, 2, 3 open. In the pair $(0, 0)$, the strand S_1 binds to domains 1, 2, 3, leaving domains S_1, 6, 7 open. The

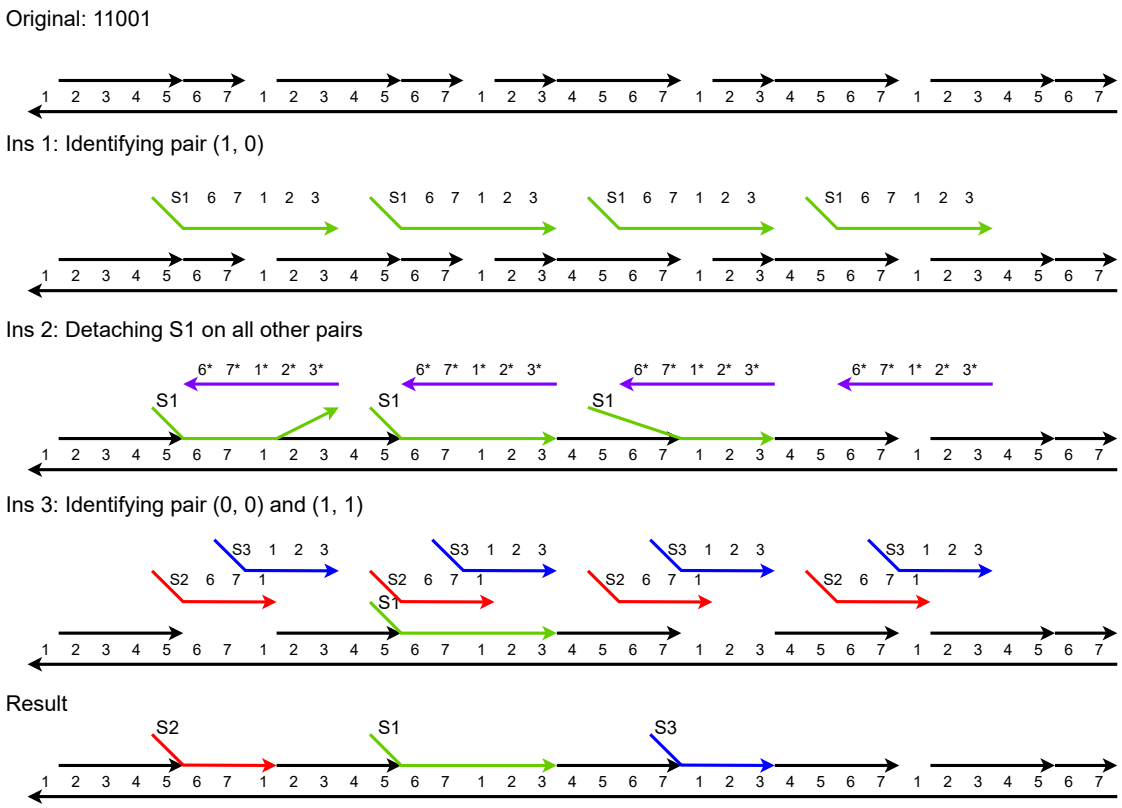


Figure 9: Example of Identifying Different Pairs of Adjacent Bits.

strand S_1 binds to domains 6, 7, 1, 2, 3, in the pair $(1, 0)$. The strand S_1 does not bind to the pair $(0, 1)$ since the only exposed toehold is domain 1. We can then distinguish the pair $(1, 0)$ from the open domains on strand S_1.

In instruction 2, using the complementary strands (6* 7* 1* 2* 3*), the strand S_1 that attaches to the pairs $(0, 0)$ and $(1, 1)$ is pulled out. This is done through the open domains 2, 3 in the pair $(0, 0)$ and the open domains 6, 7 in the pair $(1, 1)$ on strand S_1. After this instruction, strand S_1 remains only in the pair $(1, 0)$.

In instruction 3, two instruction strands are issued at the same time: (S_2 6 7 1) and (S_3 1 2 3). Here (S_2 6 7 1) will bind to the pair $(1, 1)$ and (S_3 1 2 3) will bind to the pair $(0, 0)$. They will not bind with any other pairs since the only exposed toehold for binding would be domain 1; they will prefer the locations with more exposed domains.

The result is that the adjacent bit pairs $(1, 1)$, $(1, 0)$ and $(0, 0)$ are each *labeled* with strands S_2, S_1 and S_3 respectively. Pairs $(0, 1)$ are labelled with an exposed toehold at domain 1. This toehold could be replaced by a strand (S_x 4 5 6 7 1) or a strand (S_x 1 2 3 4 5); the choice would be made depending on the use case.

1.7 Rewriting a cell

By exposing toeholds across domains 2 through 7 in a cell, we can rewrite the content of that cell – so change a 1 to 0 or a 0 to 1 – with three instructions. The idea is that, since there

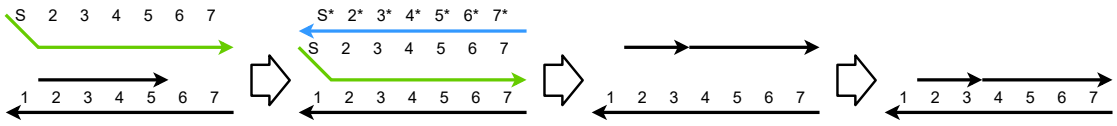


Figure 10: Example of Rewriting in Three Steps

are exposed domains, we can displace the content of the cell with a single strand covering all these domains. Then we can remove the covering strand through the exposed "tag" domain (S in Figure 10) using a complementary strand. The cell is now completely exposed. We can write a new bit to it by hybridizing the strands according to our encoding scheme, leaving domain 1 as a toehold and placing the nick at the desired location.

1.8 Parallel Binary Bubble Sorting

Sorting is a simple yet fundamental operation in computer science. Here we consider sorting binary values.[2] Sorting can be used to determine the "weight" of a vector of 0's and 1's: the count of the number of 1's relative to the length of the vector. It can also be used to compute the majority function: whether there are more 1's than 0's or not in the input set. Majority is a fundamental operation for many machine-learning algorithms.

Our SIMD DNA implementation performs parallel bubble sorting on binary bits [25]. It can be expressed as a pairwise operation in the form of $f(a, b) = (c, d)$, where (a, b) is the value of the input bit pair, and (c, d), the outputs, represent the action we take, whether to rewrite or to leave it as it is. The outputs can be 0 or 1, which means that we can arbitrarily change the value of the cell. They can also be X, meaning they remain unchanged. We discuss what kind of pairwise operations can be performed on our encoding in Section 1.15.

The sorting operation can be expressed in the following pairwise operation,

$$f(0,0) = (X,0) f(0,1) = (X,X) f(1,0) = (0,1) f(1,1) = (1,X).$$

Algorithmically, the following "bit swapping" is performed:

- If the current bit is 1, it changes it to 0 if and only if its right neighbor is 0.
- If the current bit is 0, it changes it to 1 if and only if its left neighbor is 1.

We argue that repeatedly performing such bit swapping will sort the entire sequence of binary values.

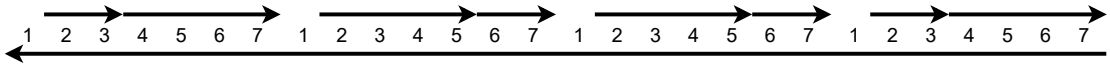
Claim: Bit swapping will never happen more than once for any consecutive sequence of three bits. Such a sequence consists of two consecutive pairs, sharing the middle bit.

Proof" The only pair of consecutive bits that ever gets rewritten is the pair $(1, 0)$ to $(0, 1)$. It is impossible to have two consecutive, overlapping pairs $(1, 0)$ sharing a common middle bit.

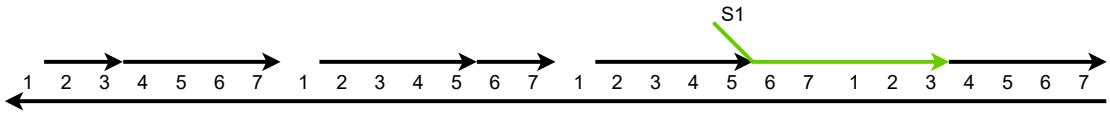
Accordingly, bubble sorting binary values in parallel does not require an odd and even index addressing scheme, as does bubble sorting arbitrary values. **Claim:** Sorting completes in at most $(N - 1)$ parallel steps where N is the total number of bits.

[2]Perhaps counter-intuitively, sorting binary values in hardware is as difficult algorithmically as sorting arbitrary values such as integers or real numbers [25]

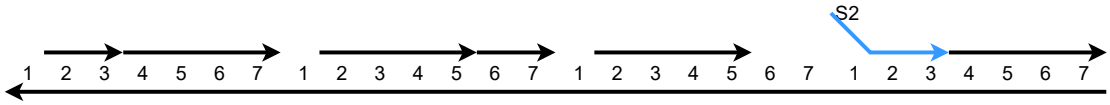
Proof: Suppose we have a sequence of binary bits of length N, in which all bits except the first are 0. When applying the algorithm, the 1 located at the start will be pushed back one position at a time with the $f(1,0) = (0,1)$ bit swap operation. Fully sorting the sequence, i.e., moving the 1 to the last position, requires $N-1$ total swaps. Now suppose we are sorting an arbitrary bit sequence. We argue that, after $N-1$ swaps, all the 1's will be at the end of the sequence. To see why, note that an $f(1,0) = (0,1)$ operations moves a 1 forward, while an $f(1,1) = (1,1)$ operation does not affect adjacent 1's. Thus, in $N-1$ steps, all 1's will have moved to end of the sequence.



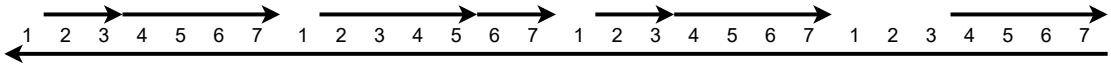
(a) Initial Sequence 0110



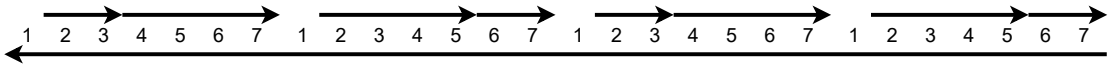
(b) After Recognizing $(1,0)$



(c) Protection on Bit 0



(d) Flipped third bit to 0, Protection Removed



(e) Flipped fourth bit to 1, Result 0101

Figure 11: Outline of the SIMD DNA parallel binary sorting algorithm.

1.9 Implementation

Here we give an instruction set for performing parallel binary bubble sort with SIMD DNA, using the encoding in Figure 2. It consists of 12 individual instructions. These are summarized as follows.

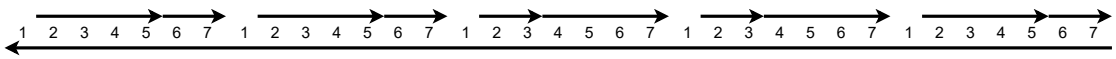
1. Label pairs $(1,0)$.
2. Uncover these, leaving domains 6 and 7 for the bits 1 and domains 2 and 3 for the bits 0 open in these pairs.
3. Protect the bits 0 of these pairs by covering the corresponding toehold at domains 2 and 3.

4. Flip the bits 1 to 0 in these pairs.

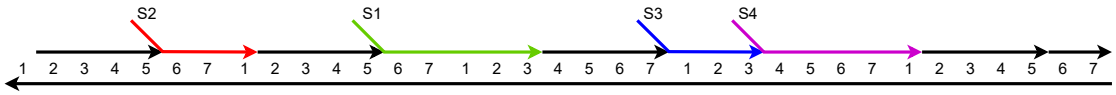
5. Release the protective covers; flip the bits 0 to 1 in these pairs.

For the initialization, we can use the first two instructions described in Section 1.6, with an additional instruction to fix open domains for bits that do not change. We can use the rewriting method described in Section 1.7 to flip the bits. A full description of the implementation of sorting is provided in Appendix 1.13.1.

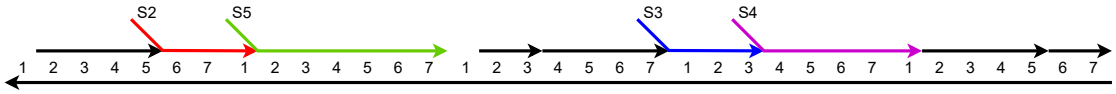
1.10 Parallel Left Shifting



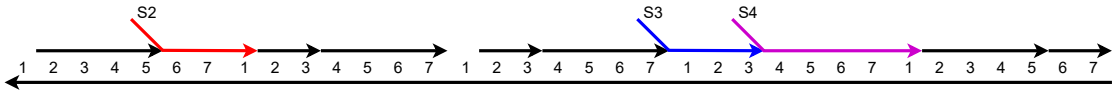
(a) Initial Sequence 11001



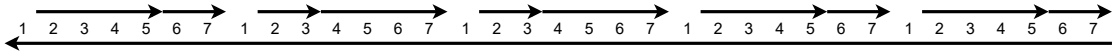
(b) After identifying all pairs



(c) Release S_1 from Pair $(1, 0)$



(d) Rewrite bit 1 in the previous pair with 0



(e) Release S_2, S_3 and S_4 then write 1, Result 10011

Figure 12: Outline of the SIMD DNA parallel left shift operations. The initial sequence S is 11001 and the result sequence T is 10011. The operation shift each bit to left one position (T[5:1]=S[4:0]), while keeping the Least Significant Bit unchanged.

We propose a SIMD DNA implementation of shifting, another fundamental operation in computer science. Shifting left corresponds to multiplying a binary number by 2; shifting right corresponds to dividing it by 2. It is a useful operation in general for aligning data in a variety of algorithms [25]. We present a left shift algorithm, one that shifts all N binary bits one position to the left, with the Least Significant Bit (LSB) remaining unchanged. This operation is, of course, a parallel left shift, moving all bits simultaneously in lockstep. Our implementation requires 11 instructions per shift. Note that unlike usual arithmetic or logical left shift that inserts a bit 0 to the LSB, the left shift operation described here keeps the

original LSB, thereby duplicating the LSB. The usual left shift could be implemented by adding instructions rewriting the LSB to 0 after the instructions we provide here.

We describe the shift operation using the following pairwise operation as:

$$f(0,0) = (0,X) f(0,1) = (1,X) f(1,0) = (0,X) f(1,1) = (1,X)$$

Here X means a value that does not change. For each bit pair, the operation writes the value of the right bit to the left bit. Since only the value of the left bit is changed in each bit pair, the operation is non-overlapping and can be implemented using the encoding scheme we propose. We illustrate with the example of shifting 11001 to 10011, shown in Figure 12.

1. Label all the bit pairs. Cover the toeholds for the pairs $(0,0)$ and $(1,1)$.
2. For the pairs $(1,0)$, flip the bits 1 to 0.
3. For the pairs $(0,1)$, flip the bits 0 to 1.
4. Finally, uncover all the toeholds for the pairs $(0,0)$ and $(1,1)$.

A full description of the implementation of shifting is given in Appendix C.

1.11 Parallel Search Algorithm

Searching is fundamental to all branches of computer science that involve data storage and retrieval. We consider the problem of deciding whether a given substring exists in a stored string of bits. We first discuss a general algorithm that returns an answer to such a question in $\log(n)$ parallel steps, where n is the substring length. We then propose an implementation in SIMD DNA. Due to practical constraints, the time complexity of the implementation is not $O(\log(n))$; it is closer to $O(n)$, depending on the problem size and implementation details. We note that a requirement of our algorithm is that the length of the query string is a power of 2. We discuss the time complexity and constraints in detail in Section 1.17.

1.11.1 Algorithm

Suppose we have a *query* substring Q of a length n and we would like to search whether it appears in a much longer *target* string A. Pseudo-code for our approach is given as Algorithm 1. We will elucidate the pseudo-code by stepping through examples.

1.11.2 Parallel search procedure

We illustrate searching for a query string $Q = 1101$ in the following target string A:

$$\begin{aligned} A_0 &= 10101010\mathbf{1101}0100011\mathbf{1101}01000100 \\ A_1 &= a_2a_2a_2a_2a_3a_1a_2a_2a_0a_3a_3a_1a_1a_0a_1a_0 \\ A_2 &= b_0b_0\mathbf{b_1}b_0b_2\mathbf{b_1}b_3b_3 \end{aligned} \tag{1}$$

The original string is A_0. In each step, two consecutive symbols are read and replaced with a single symbol. Here $a_0 = 00, a_1 = 01, a_2 = 10, a_3 = 11, b_0 = a_2a_2, b_1 = a_3a_1, b_2 = a_0a_3, b_3 = a_1a_0$. Note that $Q = 1101 = a_3a_1 = b_1$. After three steps, we conclude that the query string exists in the target string, since there are two matches in the string A_2.

Algorithm 1: Pseudo-code for Parallel Search Algorithm. Note that the operations inside the two **foreach** loops can be performed in parallel since they are independent. The **merge** operation here is to find a corresponding symbol that replaces the two symbols in the lookup table, and the **identify** operation is to look up the symbol that represents the target string.

```
Data: S ←Query String, T ←Target String
Result: A set of offsets consists of successful search with offset
n ← length of S;
results ← ∅ ;
foreach i in [0, n − 1] do
    T_i ← T;
    truncate first i characters of T_i;
    L_i ← length(T_i);
    p ← 1;
     while p ≤ length(S) do
        foreach consecutive pair of bits a, b in T_i do
            c ← pair(a, b);
            if c = identify(S) then
                return True;
            end
            replace a, b in T_i with c;
        end
        p ← 2 · p;
    end
    return False
end
```

1.12 Implementation

To implement the algorithm in SIMD DNA, we do not issue instruction strands to each pair of overlapping bits. Instead, we consider the non-overlapping bit pairs. In the example shown in Figure 13, for the bit sequence 1011, we would consider operations on bit pair 10 and 11, but not on bit pair 01.

Figure 13 shows the critical steps on searching a target sequence 1011. It provides an example of a successful search and also the potential outcome of two failed searches. To implement the search operation with an offset, we can simply skip the number of bits according to the offset. We use the word *symbol* to represent the consecutive cells that we search for on a certain level. For example, in the first level, the symbols are 10 and 11. We can use the bit identifying steps described in Section 1.6 to recognize these symbols. We use identifiers $A_0 = 00, A_1 = 01, A_2 = 10, A_3 = 11$ to represent symbols in this level. We then move on to the next level, searching for consecutive symbols A_2A_3, which corresponds to the target string 1011.

In the first step of the second level, we first rewrite the topological structure at symbols that appear to be a query result. In this example, A_2 should be found as the left symbol, and A_3 should be found as the second symbol. We pull identifier A_2 out from every *odd* symbol (we only look at the first, third, fifth, etc.) and rewrite the entire symbol with the technique described in Section 1.7. After rewriting, we have the identifier A_2' that covers domains (5 6 7) in the *right most* cell, as seen in Figure 13c. For the second symbol A_3, we repeat the step described, except we pull the identifier out from every *even* symbol and the new identifier A_3' covers domains (2 3 4) in the *left most* cell. Through these steps, we have essentially "moved" the identifier of the matching symbols to the middle. In the final step, we issue the new identifier strand (B_{11} 5 6 7 1 2 3 4) to the location between every two symbols. It will result in a perfect binding only if there is a match at the current symbol level. Figure 13e shows the example of a matching result. Figure 13f and 13g show two potential examples of imperfect binding, indicating a non-matching result. We can pull them out through the open domains either on the identifier itself or a nearby open domain on the base strand. Therefore, the presence of the identifier B_{11} indicates a successful match.

We can repeat the process to recognize multiple symbols at the same level. When we move to the next level $l + 1$, we can use the identifiers from this level l as a starting point for rewriting. To identify a symbol $S_{l+1,c} = S_{l,a}S_{l,b}$ at level $l + 1$, we simply pull out identifiers for $S_{l,a}$ at odd symbols and $S_{l,b}$ at even symbols at level l. Then we "move" the identifier to the middle. Finally, we give identifiers for $S_{l+1,c}$ to the middle of each pair and identify the symbol.

A possible weakness of our implementation is that the strand used for rewriting could potentially be very long. This could cause problems when performing these operations *in vitro* due to branch migration complications. Lastly, this search operation can handle multiple overlapping queries within the reference string, but this requires careful consideration of the base-pair sequence of the cells in designing identifier strands.

1.13 Instructions for Converting to Another Scheme

Instruction 1 identifies and distinguishes the two different bits. In instruction 1, strand (S_1 1 2 3) is issued. In bit 0, the strand will displace the short strand over domains 2 and 3 but does not edit bit 1 since domain 1 is the only open domain for binding. In instruction 2, all domains in bit 1 are replaced by a single strand covering all domains with identifier S_a. Then in instruction 3, the strand S_1 is detached, so domains 1, 2, and 3 on bit 0 are exposed. In Instruction 4, all domains in bit 0 are replaced by a single strand covering all the domains with the identifier S_b. Then any encoding scheme with 7 domains in 1 cell could be written to the bits by first detaching strand S_a and writing the encoding for bit 1, then detaching strand S_b and writing the encoding for bit 0.

1.13.1 Detailed Implementation of Each Step for Parallel Sorting

Here we give an instruction set for parallel binary bubble sort with the previously defined encoding scheme. We can implement each step of the sorting algorithm in 12 individual operations. Details of the design are shown in Figure 15.

The 12 instruction falls to 2 stages. The first stage is "identifying." During instructions 1-4, all the pairs (0, 1) are identified, and in both bit 0 and 1, a toehold is exposed for writing new data. More specifically, Instructions 1 and 2 identify the combination of $(1, 0)$. In instruction 1, (S_1 6 7 1 2 3) is issued to each pair of bits. In pair $(0, 0)$, S_1 and domains 6, 7 are exposed. In pair $(0, 1)$, since the only open domain is 1, it will not form a strong enough bind. In pair $(1, 0)$, only S_1 is exposed. In pair $(1, 1)$, S_1 and domains 2, 3 are exposed. In instruction 2, strand (6* 7* 1* 2* 3*) is issued to each pair of bits. Since pair $(1, 0)$ is the only pair that does not have exposure 5 or 2, this strand will detach strand S_1 in each pair except pair $(1, 0)$. After Instruction 2, the toehold between a bit value of 1 and a bit value of 0 in the pair $(1, 0)$ is replaced by a strand with an identifier of S_1. Instruction 3 seals off the domain exposed in the other pairs during Instruction 1 and 2 so that it will not be edited later. In instruction 4, the strand with identifier S_1 is detached, exposing domains 6 and 7 in the left cell containing bit 1, or domains 2 and 3, in the right cell containing bit 0. After this instruction, toeholds are exposed only in the 1s and 0s in pair $(1, 0)$. Other bits are not affected.

The second stage is flipping the bits in the pair $(1, 0)$. In instruction 5, in the case of a bit value of 0, domains 2 and 3 are temporarily covered by a strand with identifier S_2 so that the writing process will not interfere with the identified 0s at this moment. In instruction 6, a bit value of 1 is replaced by a strand with identifier S_3 via the open toehold at domains 6 and 7. The strand is then detached in instruction 8, exposing all the domains of that bit. Then, the bit value of 0 is written to the location of a bit value of 1 in instruction 8. In instruction 9, the temporary cover for a bit 0 is lifted. Then, in instructions 10 through 12, a bit 1 is written to the location of a bit value of 0 using the same scheme as instructions 6 through 8. Throughout the process, only bits identified in the first stage with toeholds exposed are affected.

1.13.2 Detailed Implementation of Each Step for Parallel Left Shift cell

The instructions are shown as followed, with an example of shifting 11001 to 10011.

The first three instructions are exactly the same as those for identifying bit pairs in Section 1.6. In instruction 1, the strand (S_1 6 7 1 2 3), which identifies the different patterns of two bits, is issued to each pair of bits. In instruction 2, strand (6* 7* 1* 2* 3*) is issued, detaching strands with open domains 6 and 7, or 2 and 3. After this instruction, strands with identifier S_1 only remain at pair (1, 0). In instruction 3, we issue two species of strands at the same time: (S_2 6 7 1) and (S_3 1 2 3). (S_2 6 7 1) will bind with pair $(1, 1)$ and (S_3 1 2 3) will bind with pair $(0, 0)$. S_2 will not form a stable binding with pair $(0, 0)$ or $(0, 1)$ because the binding area is only one domain. Same goes with S_3 and pair $(1, 1)$ or $(0, 1)$. After this instruction, only domain 1 between pair $(0, 1)$ is still exposed. In instruction 4, strand (S_4 4 5 6 7 1) is issued. Through the open domain 1 between pair $(0, 1)$, the strand in bit 0 is replaced by S_4. After this step, the first bit in pair $(1, 0)$ is identified with the strand S_1, and the first bit in pair (0, 1) is replaced with the strand S_4.

Instructions 5 to 9 rewrite the first bit in pair $(1, 0)$ to 0. In instruction 5, the strand S_1 is detached, exposing domains 6, 7, 1, 2 and 3. The exposed domains 2 and 3 are sealed off in instruction 6 to not interfere with subsequent instructions. In instruction 7, strand (S_5 2 3 4 5 6 7) is issued through the open toehold on domains 6 and 7 in the bit 1 in pair $(1, 0)$, and displaces the strand in that bit. Since domains 2 and 3 are sealed off, bit 0 will not be modified in this instruction. In instruction 8, strand S_5 is detached, leaving the domains in the bit open. In instruction 9, strands (2 3) and (4 5 6 7), which represent 0, are written to the bit containing open domains.

In the final two instructions, we write 1 to the first bit in pair $(0, 1)$. In instruction 10, 3 strands are issued to each pair of bits: (S_2* 6* 7* 1*), (S_3* 1* 2* 3*) and (S_4* 4* 5* 6* 7* 1*). S_2, S_3 and S_4 are detached through these strands. Since S_4 covers the bit 0 in pair (0, 1), after this step, domain 3 and 4 are exposed in these bits, ready to be written to 1. In the final step, strands (2 3), (2 3 4 5), and (6 7) are issued to each cell. Strand (2 3) and (6 7) will fix the exposed domains from strand S_2 or S_3, and strand (2 3 4 5) will write bit 1 to the bit with domain 3 and 4 exposed. Details of the design are shown in Figure 16.

For all the pairs of $(0, 0)$ and $(1, 1)$, the first bit in those pairs will not be modified since the toehold 1 will be covered with S_2 or S_3 in the process.

1.13.3 Detailed Implementation of the Second Level in Parallel Search

Here we discuss the *second* level of the parallel search operation. The first level of search operation uses the instructions that were described in Section 1.6, except we now only issue strands to non-overlapping bit pairs. We use identifiers $A_0 = 00, A_1 = 01, A_2 = 10, A_3 = 11$ to represent symbols in this level. For instance, to search for the target string 1011, we search for the symbol A_2 in odd symbols and A_3 in even symbols. The cases of A_2 in even symbols and A_3 in odd symbols are covered by searching with offset.

In the first instruction of the second level, we uncover the A_2 in the odd symbols, creating an open region. In instruction 2, we use a long strand to cover the entire right half of

the symbol, from the start of identifier A_2 to the rightmost cell. This strand is pulled out in instruction 3. In instruction 4, we use an identifier A_2' to cover domains 5, 6, 7 in the rightmost cell while covering all other domains.

Instructions 5 to 8 are essentially the same as instructions 1 to 4, but with two significant differences. Firstly, since A_3 is the second symbol in the current level of query, we only search for even-numbered symbols (2, 4, 6, etc.). Secondly, instead of rewriting the right half of the symbol, we write the left half. We make the new identifier A_3' to cover domains 2, 3, 4 in the left-most cell. In instruction 9, we use identifier ($B_1$1 5 6 7 1 2 3 4) to recognize the two consecutive symbols A_2 and A_3. Since, in the regular encoding, no strand starts from domain 5 or ends at domain 4, it will only form a perfect binding with a matched result.

After the identifier B_1 1 binds, we also need to clean up the imperfect bindings in case of a mismatch. Figure 17 shows the instructions for the cleanup process. In instruction 10, we first use the complementary strand (5* 6* 7* 1* 2* 3* 4*) to pull out the imperfect bond identifier $B_1$1. Then we issue strands covering the exposed domain. We first issue strands covering fewer domains, then in following instructions, we issue strands covering more domains. As a result, we always obtain a perfect fit; the strands will not be pulled out in potential unrelated rewriting processes.

1.14 Discussion

We proposed and implemented the in-memory and parallel computation architecture SIMD||DNA as a new DNA data storage paradigm. We theoretically proved the correctness of two programs binary counting and cellular automaton Rule 110, the latter of which has been shown to be Turing universal. In practice, we performed in-memory and parallel computation of these two programs on 4-bit registers, which can be constructed using both naturally existing sequences and artificially designed sequences. To scale up the computational power, we implemented random access memory and multiple rounds of sequential computation.

We investigated the completion level and the robustness of some of the instructions. Leak can come from fraying at the nicks in the registers and undesired opening of domains, both of which may lead to strands being mistakenly displaced or binding incorrectly. We mitigated leak by allowing registers and instructions strands to react for a short amount of time before washing. This favors the faster desired strand displacement events while slower leak reactions are unfavored. However, in situations where undesired reactions are fast, leak can be a major source of error. In electrical and computer engineering, a common error correction strategy includes adding redundancy to the system design: making one mistake in the overall computation requires multiple leak events to occur [26]. The recent "leakless" design principles have shown that introducing redundancy can also reduce leak in chemical systems [27, 28]. This raises the question of whether leakless design principles can be imposed on SIMD||DNA constructions. In addition to preventing errors at the experimental level, it remains open to address errors at the "software level" by employing error correction codes in the data and employing error correction schemes in the instructions.

Our method of storing information in DNA is motivated by recent developments in DNA storage employing topological modifications of DNA to encode data [9]. Although we use chemically synthesized strands to assemble registers, it is possible to programmatically cut naturally existing DNA and form strand breaks at desired locations as a high-throughput

method of writing information into registers. In contrast to storing data in the DNA sequence itself, encoding data in nicks could reduce the cost of large-scale *de novo* DNA synthesis by repurposing biologically-derived DNA. Other than the approach we have taken to adapt SIMD||DNA for sequencing (i.e. including a secondary sequence encoding with mismatches and performing ligation), recently developed Nanopore sequencing methods could potentially read information encoded in nicks and single-stranded gaps directly in double stranded DNA in a high-throughput manner [11]. Registers can also be affixed to the surface of a microfluidic chip to achieve autonomous control of reacting with instruction strands and elution, which could increase both the yield and scale of computation.

Compared to information stored in the DNA sequence which could be stable for thousands of years [29], information stored in the pattern of nicks may be more prone to change since the patterns of nicks is more readily disrupted than DNA sequence itself (e.g. via undesired 4-way branch migration between different registers). In addition to the methods used in traditional DNA data storage to increase the longevity [30, 31], it is possible to seal the nicks reversibly through light-induced photochemical ligation [32].

Data storage in the nicking patterns trades computability for data density. Our current encodings in SIMD||DNA store data at a density of approximately 0.03 bit per nucleotide, a decrease from traditional storage schemes that encode information in the DNA sequence itself for a theoretical maximum data density of 2 bits per nucleotide. In principle, data density can be increased by using different encoding schemes, such as allowing overhangs on the top strands to encode information. In our current implementation of reading out SIMD||DNA products, we use mismatches to differentiate bit information, which is orthogonal to the logic encoding. It may be possible to increase data density by encoding logic information through mismatches so that the effect of an instruction depends on the difference in binding stability or kinetics between mismatched and perfectly matched sequences.

We engineered SIMD||DNA programs with strand displacement reactions happening on naturally-occurring sequences. Although some DNA strand displacement systems have been adapted to naturally-occurring sequences such as diagnostic applications, it is still a challenge to design systems that can be widely applicable. There are several advantages to using naturally-occurring DNA over artificially designed and chemically-synthesized DNA. First, the length and fidelity of biologically-produced DNA far exceeds those attainable by chemical synthesis [33]. With phosphoramidite synthesis, currently the standard technique for *de novo* production of oligonucleotides, oligonucleotides longer than 100 nucleotides (such as those required for SIMD||DNA registers) are likely to be truncated and consequently result in error in register assembly. Further, if the displacing strand is truncated it may not be able to fully displace the intended target, resulting in low completion. Thus, current chemical synthesis techniques have an upper bound of oligonucleotide length under reasonable yield requirements, which limits the design of DNA architectures. Most schemes therefore avoid using oligonucleotides of lengths longer than ≈ 70 bases, because longer strands require higher levels of purification and a different, more expensive synthesis architecture (e.g. IDT Ultramers™ [34]). In contrast, bacteriophage DNA is typically on the order of kilobases in length and, importantly, single-stranded; as a result, it has been a popular choice as scaffolds for DNA origami [16], and could potentially allow SIMD||DNA to compute on several hundreds of bits. Second, the cost to produce natural DNA biologically is far lower than that of producing custom DNA synthetically. M13mp18 plasmid can be easily cultured

and harvested using minimal equipment, in contrast to custom oligonucleotide that require specialized synthesizers. Special synthesis architecture and additional purification steps are often needed to produce a similar yield compared to shorter oligonucleotides, adding to both the time and financial cost of production. At time of writing, M13 plasmid can be commercially purchased at less than \$5 for 1 μg at leading suppliers, whereas a typical oligonucleotide sequence of 200 nt costs around \$40 per 1 nmole. Further, recent technology developed for DNA origami can produce *both* short single stranded staples and the long M13 to achieve production costs of around \$0.025 per μg of folded DNA origami [35]. The same technology may be potentially applicable to SIMD||DNA, with instruction strands synthesized in the same manner as the staple strands for origami.

We discuss the features and implementation constraints of the proposed algorithms.

1.15 Ability to compute any non-conflicting pairwise operation

In Section 1.8 and Section 1.10, we presented examples of algorithms that perform pairwise operations, namely sorting and shifting, respectively. Given the ability to identify pairs of bits and a universal way to rewrite a cell, we can readily implement any algorithm that performs non-conflicting pairwise operations. Such operations only entail rewriting pairs of adjacent bits. The result of the operation on a specific sequence should always be the same, irrespective of the execution order. To illustrate, consider the following operation:

$$f(0,0) = (X,X) f(0,1) = (X,1) f(1,0) = (X,X) f(1,1) = (0,X)$$

Here X indicates a value that does not change. This operation *is* conflicting. To see why, consider its effect on the sequence 011. The second bit should change to 1 when the operation is applied to the first pair. However, this bit should change to 0 when the operation is applied to the second pair. Depending on the order of execution, the final result will be different. To ensure an operation is non-conflicting, for every three adjacent bits that two operations are performed on, the middle bit should be set to the same value.

Non-conflicting operations can be performed in parallel on all bit pairs. In the first step, we identify the four bit pairs described in 1.6. After this step, we supply strands with four labels covering the four bit pairs. Then, we release strands with specific labels one at a time to obtain write access to specific bit pairs. (Write access refers to a domain being exposed.) We rewrite these cells with the operation described in Section 1.7. The full operation requires rewriting all four bit pairs.

We conclude that our encoding scheme and design method are generally applicable to parallel bitwise algorithms, provided that they can be expressed in terms of such non-conflicted pairwise operations.

1.16 Converting to Different Encoding Schemes

A benefit of the encoding scheme that we are proposing is that it can easily be converted to any other similar scheme since each cell always has an exposed domain 1. In the original SIMD DNA scheme proposed in [36], the authors designed two specific encoding schemes for the two applications proposed (rule 110 and a binary counter). We suggest that our

encoding scheme could be used as an intermediate form when converting to other encoding schemes, designed for particular algorithms. Figure 19 illustrates how we can use a single strand (S_1 1 2 3) to differentiate bit values of 0 from bit values of 1. We can use the technique discussed in 1.7 to re-write the data with a different encoding scheme, so long as the scheme also encodes each bit with 7 domains. Complete instructions for performing such encoding changes are given in Appendix 1.13.

1.17 Time Complexity of Parallel Search

While the time complexity of the proposed parallel search is $O(\log(n))$ in principle, where n is the query substring length, the time complexity of our SIMD DNA implementation is somewhat worse. While the abstract search algorithm finds the query in the reference string by pairing individual characters in parallel, and thus completes in $O(\log(n))$ steps, our implementation searches for and identifies distinct symbols sequentially, that is to say, it first searches for a specific symbol across all possible locations at once, then it searches for the next symbol across all locations at once, and so on.

The abstract algorithm assumes all symbols are identified in one pass to allow for further pairing. If we consider all the different symbols in a query string, counting repeated symbols, $\frac{n}{2^i}$ symbols must be searched sequentially at level i in our implementation. Accordingly, the total number of sequential search steps could be as high as $O(n)$. However, at each level, all the occurrences of a specific symbol are identified simultaneously. At level i, each symbol represents a binary string with a length of 2^i, so there are at most 2^{2^i} distinct symbols at level i. For example, in the first level, instead of searching for $\frac{n}{2}$ symbols, we only search for four distinct symbols. In the second level, there are only 16 distinct symbols. Since we only search for distinct symbols, the number of steps in the first few levels will be greatly reduced.

Our parallel search algorithm currently only works on query strings having a length that is a power of two. However, we believe that our implementation could be modified to allow for arbitrary-length query strings. We do not provide details here, as they are cumbersome, but we outline the method as follows.

Note that, in parallel search, the query string is searched reductively: at each level, two symbols are reduced to one symbol. When working with query strings having any arbitrary length, there might be an odd number of symbols in the current level, meaning that the last symbol cannot be reduced for the next level. In this case, we can add a method to identify the trailing odd symbol at the current level and replace it in the next level. The reduction can still be completed in a logarithmic number of levels.

The range of algorithms discussed here suggest that the SIMD||DNA paradigm is broadly applicable to computation. It is open to investigate the power of SIMD||DNA: what algorithms can be directly designed (instead of implemented through Turing machine) and what algorithms in principle cannot be achieved in SIMD||DNA paradigm. For all the designs available, they all have unique data encodings. For the future design, it is natural to ask whether there exists a universal encoding that is compatible with different programs, or if it is possible to convert one encoding to the other. Can we design more efficient programs with regard to the number of instructions or the number of unique instruction strands? More complicated strand displacement mechanisms [37, 38, 39] may sacrifice speed and increase design complexity in exchange for extra computational power in this architecture.

SIMD||DNA can potentially revolutionize DNA storage architecture for future applications. Given the current challenges in attaining high-quality, large-scale *de novo* long custom strand synthesis [40, 41] and the urgent, growing need for archival data storage worldwide, SIMD||DNA presents an intermediate solution that facilitates DNA storage for practical settings. In a longer-term context, SIMD||DNA could remain relevant as an interface between DNA computation modules that process molecular inputs and a semi-permanent record of the output of those computations, thereby both scaling up strand displacement-based DNA nanotechnology and adding a "wet" computation component to otherwise "cold" data storage. Towards this end, one could for instance envision a database of personal medical records that is collected through molecular detection programs taking daily samples from the patient as input and updating corresponding registers for later readout.

1.18 Conclusions

We have presented algorithms for basic parallel operations within the SIMD DNA framework. We note that there are, in fact, two layers of parallelism possible:

1. Bit-level Parallelism: instructions applied to all bits in an array at once.
2. Data-level Parallelism: the same instructions applied to *multiple* arrays at once.

While operations on DNA are slow and error-prone, with these levels of parallelism, perhaps DNA computation could scale to a truly impressive regime. Consider the following back-of-an-envelop estimates. Suppose:

- we have 10^{12} independent cells in parallel in a single test tube;
- a single operation takes approximately 10 minutes to complete.
- different cells use the same DNA sequence. Using distinct sequences for different cells, as in our search operation, can result in a solution with multiple competing DNA molecules. At larger scales, this would result in an increase in reagent volume and could diminish reaction rates.

This means that we can perform approximately 10^9 operations per second in a single test tube, already impressive. Now suppose that:

- we have 100 test tubes.

This means we can compute at 100,000 MIPS (million instructions per second). This is comparable to what very respectable existing silicon systems can achieve. The key conceptual difference between the SIMD DNA approach and other forms of DNA computing is that it exploits a substrate on which data is stored. This enables the SIMD parallelism.

Many experimental hurdles remain in demonstrating and deploying this paradigm. DNA synthesis remains prohibitively expensive. A possible alternative is to use gene-editing techniques to encode data on naturally occurring DNA [42].

2 Fractional Computing on Concentrations

This report presents a novel strategy for computing mathematical functions with molecular reactions, based on theory from the realm of digital design. It demonstrates how to design chemical reaction networks based on truth tables that specify analog functions, computed by stochastic logic. The theory of stochastic logic entails the use of random streams of zeros and ones to represent probabilistic values. A link is made between the representation of random variables with stochastic logic on the one hand, and the representation of variables in molecular systems as the concentration of molecular species, on the other. Research in stochastic logic has demonstrated that many mathematical functions of interest can be computed with simple circuits built with logic gates. This report presents a general and efficient methodology for translating mathematical functions computed by stochastic logic circuits into a chemical reaction networks. The computation performed by the reaction networks is accurate and robust to variations in the reaction rates, to within a log-order constraint. Reaction networks are given that compute functions for applications such as image and signal processing, as well as machine learning: arctan, exponential, *Bessel*, and sinc. An implementation of the reaction networks is proposed with a specific experimental chassis: DNA strand displacement with units called DNA concatemers.

2.1 Introduction

In recent years, the topic of stochastic logic has been advertised as a possible design paradigm for emerging technologies that promise scaling beyond complementary metal–oxide–semiconductor (CMOS), as well as the basis of non-von Neumann architectures [43, 44]. While the term can mean many things, ranging from randomized algorithms to probabilistic analysis, in our context "stochastic computing" or "stochastic logic" has a specific meaning: it refers to logic-level computation on randomized bitstreams. Instead of the traditional values of 1 and 0 that form the basis of binary computing systems, in stochastic computing a real value x is represented as a stream of random bits. In this stream, the probability of a randomly chosen bit being 1 is x, and the probability of it being 0 is $1 - x$.

The original ideas for this form of stochastic computation are generally attributed to research by Gaines and Poppelbaum in the late 1960s [45, 46], as well as to work by Brown and Card in the 1990s [47]. Beginning in the late 2000s, there has been a renewed interest, with too many publications to enumerate. We point to some influential papers as well as surveys: [48, 49, 50, 51, 52, 53]. In [54, 55], Qian et al. presented a general synthesis methodology for stochastic logic. Our exposition is based on that framework.

The main appeal of stochastic logic is that a wide variety of functions can be computed with simple structures. For instance, multiplication can be implemented with a single AND gate. More complicated functions such as the exponential, absolute value, square roots, and hyperbolic tangent can each be computed with a very small number of gates [56]. Simplicity is a compelling advantage for the task that we confront in this report computing with molecular reactions.

The idea of molecular computing dates back to seminal work by Len Adleman, who discussed solutions to combinatorial problems such Boolean satisfiability and Hamiltonian paths with DNA [57]. There has been a broad range of research since. We point to a small

subset: [3, 58, 59, 60, 61, 62, 63].

This report explores a link between the two fields. Specifically, it presents a strategy for computing mathematical functions with molecular reactions by applying concepts from stochastic logic. We preview with an example. Suppose we want a chemical reaction network that computes the function

$$f(a, b) = 1 - a - b + ab,$$

where a and b are real-valued variables. The corresponding digital function for stochastic logic can be obtained using the methods discussed in Section 2.1. In this case, it is $f(a, b) = \mathrm{NOR}(a, b)$, expressed in the following truth table:

a	b	$\mathrm{NOR}(a, b)$
0	0	1
0	1	0
1	0	0
1	1	0

To represent a stochastic variable x that ranges from $[0, 1]$ in a molecular format, we use a *pair* of chemical species X_0 and X_1. As will be discussed in Section 2.2, we use a *fractional* representation:

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0] + [\mathbf{X}_1]}. \tag{2}$$

Here $[X_1]$ denotes the concentration of the molecular species X_1. Using this representation, we obtain a chemical reaction network (CRN) from the truth table above:

$$\begin{aligned} \mathbf{A}_0 + \mathbf{B}_0 &\rightarrow \mathbf{C}_1 \\ \mathbf{A}_0 + \mathbf{B}_1 &\rightarrow \mathbf{C}_0 \\ \mathbf{A}_1 + \mathbf{B}_0 &\rightarrow \mathbf{C}_0 \\ \mathbf{A}_1 + \mathbf{B}_1 &\rightarrow \mathbf{C}_0 \end{aligned} \tag{3}$$

Note that the subscripts of the species match the entries of the truth table above. This CRN computes the target function, $c = 1 - a - b + ab$, in terms of the fractional variables a, b and c. Each of these corresponds to a pair of chemical species, $\{A_0, A_1\}, \{B_0, B_1\}$ and $\{C_0, C_1\}$, respectively. The central result of this report, presented in Section 2.4, is a proof that we can implement any polynomial function, specified by a truth table, with a CRN matching its truth table template.

This report builds upon our prior work, both generalizing and simplifying it. We use the same formalism, namely a fractional representation of values, in this report as in [66] and [67].

- In [66], we proposed a technique for computing functions based on a decomposition with *Bernstein polynomials* [68]. The technique can implement a broad class of functions, namely all univariate polynomials, but is quite abstruse. A target polynomial is first repackaged in Bernstein form [69]. This form is implemented in a logic circuit using a form of generalized *multiplexing* [55]. Finally, the logic circuit is translated into a CRN.

- In [67], we proposed an alternate technique based on a factoring of polynomials with *Horner's rule*. The factored form is implemented with a cascade of 2-input logic gates. Finally, the logic gate circuit is translated into a CRN. Although conceptually simpler than working with Bernstein polynomials, this approach is not quite so general: only a small subset of polynomials can be decomposed in the requisite way with Horner's rule.

We admit that a significant limitation of the prior work is the complexity of the mathematical packaging.

The approach in this report is conceptually much cleaner. As with the NOR function example above, a target polynomial function is first mapped to a truth table. This can be done using fairly standard techniques – at least for people familiar with the theory of stochastic logic – and the results are intuitive. Then a CRN is constructed that matches the template of the truth table.

This approach is also more general. Whereas the method in [66] is limited to univariate polynomials, the method in this report can implement any multivariate polynomial. Stochastic logic operates on functions where the domain and codomain are in the interval $[0, 1]$, i.e., the inputs and the output are probabilities. Common transcendental functions can be computed via polynomial approximations. In the supplementary information, we provide CRNs for stochastic functions such as arctan, exponential, Bessel, and sinc to demonstrate our approach in detail. These functions have practical applications in fields such as machine learning, signal processing, and image processing. We discuss the implementation of these abstract chemical reaction networks with DNA strand displacement, with units called DNA concatemers.

This report is organized as follows. Section 2.2 presents background information on chemical reaction networks and stochastic logic. Section 2.3 describes our methodology for translating any function computed by a stochastic logic circuit into a set of chemical reactions. Section 2.4 provides mathematical proof that the proposed methodology is mathematically sound, based on an analysis of the chemical kinetics. Section 2.6 analyzes sources of error stemming from differences in reaction rates. Section 2.7 discusses the implementation with DNA strand displacement and DNA concatemers. Finally, Section 2.9 provides concluding remarks and discusses future research directions.

2.2 Background

2.2.1 Chemical Reaction Networks

A chemical reaction network (CRN) consists of a set of *reactions* operating on a set of *molecular species*. When a reaction fires, *reactant* molecules are transformed into *product* molecules. Tor instance, consider the reaction:

$$\mathbf{X}_1 + \mathbf{X}_2 \xrightarrow{\mathrm{k}} \mathbf{X}_3.$$

Here one molecule of reactant $\mathbf{X}_1$ combines with one molecule of reactant $\mathbf{X}_2$, resulting in one molecule of the product $\mathbf{X}_3$. The parameter k is called the *rate constant*. A CRN consists of multiple reactions occurring simultaneously, where different reactions may involve different combinations of reactants and yield different products. Consider a toy example of a CRN

with three reactions operating on the molecule species set $\{\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4\}$:

$$\begin{aligned} \mathbf{X}_1 + \mathbf{X}_2 &\rightarrow \mathbf{X}_3, \\ \mathbf{X}_2 + \mathbf{X}_3 &\rightarrow 2\mathbf{X}_4, \\ 3\mathbf{X}_3 + \mathbf{X}_4 &\rightarrow \mathbf{X}_1. \end{aligned}$$

Here we assume that all three reactions have the same rate constant, k, an arbitrary value. To quantify the changes in concentration of all the molecular species involved in a CRN over time we can apply the theory of *mass-action kinetics* [70]: reaction rates are proportional to both the concentrations of the reactants and to their rate constants. Given a CRN, one can derive a set of nonlinear differential equations for the concentrations of all molecular species. For instance, for the first reaction above, the rate of change of the concentrations of $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ is

$$-\frac{d[\mathbf{X}_1]}{dt} = -\frac{d[\mathbf{X}_2]}{dt} = \frac{d[\mathbf{X}_3]}{dt} = k[\mathbf{X}_1][\mathbf{X}_2], \tag{4}$$

where $[\mathbf{X}]$ denotes the concentration of the chemical species $\mathbf{X}$. (We omit the equations for the second and third reactions for brevity.) Given the initial concentration of the different molecular species, one can predict the behavior of the CRN by simulating the differential equations.

2.2.2 Digital Logic

We give some basic definitions that we will need pertaining to digital logic.
Definition: Combinational Logic Function. An n-input combinational logic function is a function $F(X_1, X_2, \ldots, X_n) = Y$, where all inputs and outputs are Boolean values. That is, $\forall\, 1 \leq i \leq n, X_i \in \{0, 1\}, Y \in \{0, 1\}$.
Definition: Truth table. The truth table of a combinational logic function lists all the possible combinations of its inputs and the corresponding outputs. Each combination of inputs is called a **minterm**. Table 1 in the next section gives an example of the truth table of a combinational logic function. (We also provided the truth table for the NOR function in Section 2.1.)

2.2.3 Stochastic Logic

Stochastic logic is an active topic of research in digital design, with applications to emerging technologies [45, 55]. Computation is performed with familiar digital constructs, such as AND, OR, and NOT gates. However, instead of having specific Boolean values of 0 and 1, the inputs are random bitstreams. A number x ($0 \leq x \leq 1$) corresponds to a sequence of random bits. Each bit has *probability* x of being one and probability $1 - x$ of being zero. Computation is recast in terms of the probabilities observed in these streams.

Consider basic logic gates. Given a stochastic input x, a NOT gate implements the function

$$\mathrm{NOT}(x) = 1 - x. \tag{5}$$

This means that while an individual input of 1 results in an output of 0 for the NOT gate (and vice versa), statistically, for a random bitstream that encodes the stochastic value x, the NOT gate output is a new bitstream that encodes $1 - x$.
The output of an AND gate is 1 only if all the inputs are simultaneously 1. The probability of the output being 1 is thus the probability of all the outputs being 1. Therefore, an AND gate implements the stochastic function:

$$\mathrm{AND}(x, y) = xy, \tag{6}$$

that is to say, multiplication. The output of an OR gate is 0 only if all the inputs are 0. Therefore, an OR gate implements the stochastic function:

$$\mathrm{OR}(x, y) = 1 - (1 - x)(1 - y) = x + y - xy. \tag{7}$$

The output of an XOR gate is 1 only if the two inputs x, y are different. Therefore, an XOR gate implements the stochastic function:

$$\mathrm{XOR}(x, y) = (1 - x)y + x(1 - y) = x + y - 2xy. \tag{8}$$

The NAND, NOR, and XNOR gates can be derived by composing the AND, OR, and XOR gates each with a NOT gate, respectively. Please refer to Table 2 for a full list of the algebraic expressions of these gates. An important assumption in stochastic computation is that all inputs are independent of each other, i.e., the random bitstreams are uncorrelated.

We formalize the definition of stochastic logic functions as follows.
Definition: Stochastic Logic Function. An n-input stochastic logic function $y = f(x_1, x_2, \ldots, x_n)$, where $\forall\, x_i \in [0, 1]$ and $y \in [0, 1]$, is obtained from a combinational logic function $Y = F(X_1, X_2, \ldots, X_n)$, by the setting corresponding inputs to be independent random variables X_i with $\Pr(X_i = 1) = x_i$.
For a given Boolean circuit, its stochastic function can be computed as follows.

Theorem 1 (Output of a Stochastic Logic Function[71])**.** *Given input sequences generated by independent Bernoulli random variables, the output of a stochastic logic function will also be a sequence generated by a Bernoulli random variable. The probability of the output of a stochastic logic function f being* 1 *is the sum of all the probabilities of the minterms that evaluate to* 1 *in the corresponding combination logic function F. That is,*

$$\Pr(Y = 1) = \sum_{J \in S} \left(\prod_{h=1}^{n} [\Pr(X_h = j_h)] \right) \tag{9}$$

where $J = (j_1, j_2, \ldots, j_n), j_i \in \{0, 1\}$ is a minterm, and $S = \{J | F(J) = 1\}$ is the set of minterms that evaluate to 1.

To elucidate Theorem 1, we step through the implementation of a stochastic logic function from a truth table. Consider a combinational circuit computing a function $F(X_1, X_2, X_3)$ with the truth table shown in Table 1. Let $f(x_1, x_2, x_3)$ be the stochastic function computed

by this circuit, with real-valued inputs $x_1, x_2, x_3 \in [0, 1]$. Assuming each input is independent of the others, set

$$[\Pr(X_1) = 1] = \quad x_1, \tag{10}$$
$$[\Pr(X_2) = 1] = \quad x_2, \tag{11}$$
$$[\Pr(X_3) = 1] = \quad x_3. \tag{12}$$

Table 1: Truth table for a combinational circuit, and the corresponding probability of each row.

X_1	X_2	X_3	$F(X_1, X_2, X_3)$	Probability of row
0	0	0	0	$(1 - x_1) \cdot (1 - x_2) \cdot (1 - x_3)$
0	0	1	1	$(1 - x_1) \cdot (1 - x_2) \cdot x_3$
0	1	0	0	$(1 - x_1) \cdot x_2 \cdot (1 - x_3)$
0	1	1	1	$(1 - x_1) \cdot x_2 \cdot x_3$
1	0	0	0	$x_1 \cdot (1 - x_2) \cdot (1 - x_3)$
1	0	1	1	$x_1 \cdot (1 - x_2) \cdot x_3$
1	1	0	1	$x_1 \cdot x_2 \cdot (1 - x_3)$
1	1	1	1	$x_1 \cdot x_2 \cdot x_3$

The probability that the function f evaluates to 1 is equal to the sum of the probabilities of occurrence of each row that evaluates to 1. The probability of occurrence of each row, in turn, is obtained from the assignments to the variables, as shown in Table 1: x_i if the corresponding variable X_i is 1 and $(1 - x_i)$ if it is 0. Thus, we filter the rows in Table 1 where $F(X_1, X_2, X_3) = 1$ and add their probabilities together to obtain the expression for the stochastic function:

$$\begin{aligned} f(x_1, x_2, x_3) =& (1 - x_1)(1 - x_2)x_3 \quad + \\ & (1 - x_1)x_2x_3 \quad + \\ & x_1(1 - x_2)x_3 \quad + \\ & x_1x_2(1 - x_3) \quad + \\ & x_1x_2x_3 \\ =& (1 - x_2)x_3 + x_2x_3 + x_1x_2(1 - x_3). \end{aligned} \tag{13}$$

The procedure shown for this example can be generalized to any combinational circuit to evaluate its stochastic function. Such probabilistic analysis of networks of logic gates is not new. As early as 1975, the circuit testing community had begun analyzing errors in a similar way [72, 73]. Similar techniques have also been applied to tasks such as timing and power analysis [74, 75]. However, characterizing the *outputs* of the computation this way, as probabilistic functions, is specific to the field of stochastic logic. We point to some of our prior work in this field. In [69] we proved that any multivariate polynomial function with its domain and codomain in the unit interval $[0, 1]$ can be implemented using stochastic logic.

In [55], we provide an efficient and general synthesis procedure for stochastic logic, the first in the field. In [50], we provided a method for transforming probabilities values with digital logic. Finally, in [76, 77] we demonstrated how stochastic computation can be performed deterministically.

2.3 Implementing Stochastic Logic with Chemical Reactions

In the introduction, we gave a brief example of translating a simple polynomial function, the NOR function, into a CRN. In this section, we step through the details of this process.

2.3.1 Fractional Representation in Solution

To represent a stochastic value x in a chemical system, we use two distinct molecular species $\mathbf{X}_0$ and $\mathbf{X}_1$ such that

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0] + [\mathbf{X}_1]}. \tag{14}$$

Here we use the notation $[\mathbf{X}]$ to refer to the concentration of a molecular species $\mathbf{X}$. We introduced this *fractional* representation in our prior work [66, 67]: the value x equals the ratio of the concentration of $\mathbf{X_1}$ to the total concentration of $\mathbf{X_0}$ and $\mathbf{X_1}$. As with probabilities in stochastic logic, such a fractional value can represent any real number in the unit interval $[0, 1]$. Indeed, we will demonstrate how this fractional encoding can be used to compute stochastic functions. We present a potential experimental implementation using DNA strand displacement in Section 2.7.

2.3.2 Building a Chemical Reaction Network from a Truth Table

Consider the truth table for the Boolean AND operation:

A	B	C =AND(A, B)
0	0	0
0	1	0
1	0	0
1	1	1

Given the fractional representation described above, let us design a CRN that performs multiplication with an AND operation on two stochastic inputs a and b, producing an output c. The network consists of the following reactions:

$$\begin{aligned} \mathbf{A}_0 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0, \\ \mathbf{A}_0 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_0, \\ \mathbf{A}_1 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0, \\ \mathbf{A}_1 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_1. \end{aligned} \tag{15}$$

Here k is the rate constant, an arbitrary value, equal for all the reactions. Notice that there is a one-to-one mapping from the Boolean truth table of the AND gate to the indices of the

chemical species. Note that, given the two inputs a and b in the fractional encoding,

$$a = \frac{[\mathbf{A}_1]}{[\mathbf{A}_0] + [\mathbf{A}_1]} \quad \text{and} \quad b = \frac{[\mathbf{B}_1]}{[\mathbf{B}_0] + [\mathbf{B}_1]}. \tag{16}$$

If we simulate this CRN, we observe that

$$c = \frac{[\mathbf{C}_1]}{[\mathbf{C}_0] + [\mathbf{C}_1]} = a \times b. \tag{17}$$

This strategy for implementing stochastic functions with CRN works for an arbitrary number of inputs, provided the reaction rates are the same for all reactions. We will prove this assertion in Section 2.4. Table 2 lists CRNs that implement the stochastic functions of all the basic logic gates. Again, note that the indices that appear in each CRN match the truth table of the corresponding gate.

The rate constants for all reactions in these CRNs must be equal for the computation to proceed correctly. Consider a different situation: for the CRN presented in Eq. 15, suppose that the rate constant of the fourth reaction is $2k$, while all the other rate constants are k (where k is an arbitrary value). Given stochastic inputs $a = 0.7$ and $b = 0.6$, simulation shows that the output is $c = 0.462$ instead of the expected value $a \times b = 0.42$. We analyze the effects of varying rate constants on the accuracy of the computation in Section 2.6.

2.4 Proof of the Proposed Method

Here we prove the correctness of the method of implementing stochastic functions with CRNs discussed in Section 2.3:

Theorem 2. *Assume an n-input stochastic function $y = f(x_1, x_2, \ldots, x_n)$ is implemented by a combinational Boolean function $Y = F(X_1, X_2, \ldots, X_n)$. The stochastic function can then be implemented with a CRN with $2n + 2$ different molecular species, in which pairs of molecular species store the input values $x_1, x_2, \ldots, x_n$ as well as the output value y, according to the fractional representation in Eq 14. The CRN consists of 2^n reactions, each of the form,*

$$\mathbf{X}_{1,v_1} + \mathbf{X}_{2,v_2} + \cdots + \mathbf{X}_{n,v_n} \xrightarrow{k} \mathbf{Y}_{F(V)}, \tag{18}$$

where $v_1, v_2, \ldots, v_n : F(V)$ is a row of the truth table for the combinational function F, and $V = (v_1, v_2, \ldots v_n)$ denotes a minterm for the function . Note that the rate constants for all reactions are equal to k, an arbitrary value.

Let S_1 be the set of all minterms V such that $F(V) = 1$, and let S_0 be the set of all minterms V such that $F(V) = 0$. Also, we denote $c_{i,j}$ as,

$$c_{i,j} = \Pr(X_i = j) = \begin{cases} 1 - x_i & \text{if } j = 0 \\ x_i & \text{if } j = 1 \end{cases} \tag{19}$$

where x_i is a stochastic input, and i is the index of the input x_i in function $y = f(x_1, x_2, \ldots, x_n)$.

Table 2: Chemical Reaction Networks for Basic Logic Gates. Note that the indices of molecules match the truth table implementing the logic gate.

gate	inputs	function	CRN
NOT	a	$b = 1 - a$	$\mathbf{A}_0 \to \mathbf{B}_1$ $\mathbf{A}_1 \to \mathbf{B}_0$
AND	a, b	$c = ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$
OR	a, b	$c = a + b - ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$
NAND	a, b	$c = 1 - ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
NOR	a, b	$c = 1 - a - b + ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
XOR	a, b	$c = a + b - 2ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_0$
XNOR	a, b	$c = 1 - a - b + 2ab$	$\mathbf{A}_0 + \mathbf{B}_0 \to \mathbf{C}_1$ $\mathbf{A}_0 + \mathbf{B}_1 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_0 \to \mathbf{C}_0$ $\mathbf{A}_1 + \mathbf{B}_1 \to \mathbf{C}_1$

To prove the theorem, we need to show that, for the given initial values of the stochastic value x_i at time $t = 0$,

$$x_i = \left.\frac{[\mathbf{X}_{i,1}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]}\right|_{t=0}, \tag{20}$$

the output of the CRN should match the output of the stochastic function stated in Theorem 1,

$$\lim_{t\to\infty} y = \lim_{t\to\infty} \frac{[\mathbf{Y}_1]}{[\mathbf{Y}_0] + [\mathbf{Y}_1]} = \sum_{V\in S_1} \left(\prod_{h=1}^{n} c_{h,v_h} \right). \tag{21}$$

In fact, we prove a stronger result: Eq. 21 applies for all $t > 0$.
Proof Given the CRN described in Theorem 2, the rate equations for each input are

$$\begin{aligned} \frac{d[\mathbf{X}_{i,j}]}{dt} &= -k \cdot [\mathbf{X}_{i,j}] \cdot \prod_{h=1,h\neq i}^{n} \left([\mathbf{X}_{h,0}] + [\mathbf{X}_{h,1}] \right), j \in \{0,1\} && (22) \\ &= -k \cdot \frac{[\mathbf{X}_{i,j}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]} \cdot \prod_{h=1}^{n} \left([\mathbf{X}_{h,0}] + [\mathbf{X}_{h,1}] \right). && (23) \end{aligned}$$

Note that k, an arbitrary value, is the rate constant for each reaction. The rate equations for the output species are,

$$\frac{d[\mathbf{Y}_j]}{dt} = k \sum_{V\in S_j} \left(\prod_{h=1}^{n} [\mathbf{X}_{h,v_h}] \right), j \in \{0,1\}. \tag{24}$$

We define the following new variables,

$$\begin{aligned} p_i &= \frac{[\mathbf{X}_{i,1}]}{[\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}]} && (25) \\ q_i &= [\mathbf{X}_{i,0}] + [\mathbf{X}_{i,1}] && (26) \\ r_{i,j} &= \begin{cases} 1 - p_i & \text{if } j = 0 \\ p_i & \text{if } j = 1 \end{cases} && (27) \end{aligned}$$

We substitute these variables into the expressions for the concentrations:

$$\begin{aligned} [\mathbf{X}_{i,0}] &= q_i(1 - p_i), && (28) \\ [\mathbf{X}_{i,1}] &= q_i p_i, && (29) \\ \therefore [\mathbf{X}_{i,j}] &= q_i r_{i,j}. && (30) \end{aligned}$$

These substitutions are introduced into Eqs. 23 and 24:

$$\begin{aligned} \frac{d[\mathbf{X}_{i,j}]}{dt} &= -k \cdot r_{i,j} \prod_{h=1}^{n} q_h, && (31) \\ \frac{d[\mathbf{Y}_j]}{dt} &= k \left(\prod_{h=1}^{n} q_h \right) \sum_{V\in S_j} \left(\prod_{h=1}^{n} r_{h,v_h} \right). && (32) \end{aligned}$$

As the concentrations $[\mathbf{X}_{i,j}]$ are functions of time, all p, q, and r are also functions of time. Consider the following two expressions derived from Eq. 31,

$$\frac{d[\mathbf{X}_{i,0}]}{dt} = -k(1-p_i)\prod_{h=1}^{n} q_h \tag{33}$$

$$\frac{d[\mathbf{X}_{i,1}]}{dt} = -k \cdot p_i \prod_{h=1}^{n} q_h \tag{34}$$

$$\therefore \frac{dq_i}{dt} = \frac{d[\mathbf{X}_{i,0}]}{dt} + \frac{d[\mathbf{X}_{i,1}]}{dt} = -k\prod_{h=1}^{n} q_h. \tag{35}$$

We also have

$$[\mathbf{X}_{i,1}] = p_i \cdot q_i \tag{36}$$

$$\therefore \frac{d[\mathbf{X}_{i,1}]}{dt} = p_i\frac{dq_i}{dt} + q_i\frac{dp_i}{dt} \tag{37}$$

$$= p_i\left(-k\prod_{h=1}^{n} q_i\right) + q_i\frac{dp_i}{dt} \tag{38}$$

$$= \frac{d[\mathbf{X}_{i,1}]}{dt} + q_i\frac{dp_i}{dt}. \tag{39}$$

As $q_i \neq 0$, we conclude that

$$\frac{dp_i}{dt} = 0, \tag{40}$$

that is, p_i is invariant to time. Consequently, $r_{i,j}$ is also invariant to time. This means that the stochastic value encoded by each pair of input species remains the same throughout the reaction. Therefore, for $t > 0$, we have

$$p_i = x_i \tag{41}$$

$$r_{i,j} = c_{i,j} \tag{42}$$

We assign the new symbol

$$l = \left(\prod_{h=1}^{n} q_i\right) \tag{43}$$

$$\therefore \frac{d[\mathbf{Y}_j]}{dt} = k \cdot l \sum_{V \in S_j}\left(\prod_{h=1}^{n} r_{h,v_h}\right). \tag{44}$$

Finally, we can calculate the stochastic output y as

$$y = \frac{\int_0^t \frac{d[\mathbf{Y}_1]}{dt} dt}{\int_0^t \frac{d[\mathbf{Y}_0]}{dt} dt + \int_0^t \frac{d[\mathbf{Y}_1]}{dt} dt} \tag{45}$$

$$= \frac{\sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h} \right) \int_0^t k \cdot l \cdot dt}{\sum_{V \in S_0} \left(\prod_{h=1}^{n} r_{h,v_h} \right) \int_0^t k \cdot l \cdot dt + \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h} \right) \int_0^t k \cdot l \cdot dt} \tag{46}$$

$$= \frac{\sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h} \right)}{\sum_{V \in S_0} \left(\prod_{h=1}^{n} r_{h,v_h} \right) + \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h} \right)} \tag{47}$$

$$= \sum_{V \in S_1} \left(\prod_{h=1}^{n} r_{h,v_h} \right). \tag{48}$$

The numerator in Eq. 47 corresponds to the sum of the minterms of all rows of the truth table F that evaluate 1, while the denominator corresponds to the sum of all minterms. As $r_{i,j}$ is only dependent on the initial input value, the denominator must sum up to 1 since it includes all the minterms. Therefore, we conclude that a CRN constructed this way, corresponding to an arbitrary Boolean truth table F will implement the stochastic function f of that truth table. The only requirement is that the rate constants of all the reactions must be equal.

In what follows, we elucidate the proof with detailed examples. In the Supporting Information, we give CRN implementations of a variety of functions that are of practical interest.

2.5 A demonstrative example

Let us go back to the two-input AND gate from Section 2.3.

$$\begin{aligned} \mathbf{A}_0 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0 \\ \mathbf{A}_0 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_0 \\ \mathbf{A}_1 + \mathbf{B}_0 &\xrightarrow{k} \mathbf{C}_0 \\ \mathbf{A}_1 + \mathbf{B}_1 &\xrightarrow{k} \mathbf{C}_1 \end{aligned} \tag{49}$$

The rate equations for the input and output species are:

$$\begin{aligned}
\frac{d[\mathbf{A}_0]}{dt} &= -k[\mathbf{A}_0]([\mathbf{B}_0]+[\mathbf{B}_1]) \\
\frac{d[\mathbf{A}_1]}{dt} &= -k[\mathbf{A}_1]([\mathbf{B}_0]+[\mathbf{B}_1]) \\
\frac{d[\mathbf{B}_0]}{dt} &= -k[\mathbf{B}_0]([\mathbf{A}_0]+[\mathbf{A}_1]) \\
\frac{d[\mathbf{B}_1]}{dt} &= -k[\mathbf{B}_1]([\mathbf{A}_0]+[\mathbf{A}_1]) \\
\frac{d[\mathbf{C}_0]}{dt} &= k([\mathbf{A}_0][\mathbf{B}_0]+[\mathbf{A}_0][\mathbf{B}_1]+[\mathbf{A}_1][\mathbf{B}_0]) \\
\frac{d[\mathbf{C}_1]}{dt} &= k[\mathbf{A}_1][\mathbf{B}_1].
\end{aligned} \tag{50}$$

We introduce some variables to represent the stochastic values,

$$a = \frac{[\mathbf{A}_1]}{[\mathbf{A}_0]+[\mathbf{A}_1]},\ \ b = \frac{[\mathbf{B}_1]}{[\mathbf{B}_0]+[\mathbf{B}_1]},\ \ c = \frac{[\mathbf{C}_1]}{[\mathbf{C}_0]+[\mathbf{C}_1]},$$

as well as the sum of concentrations of each pair of input species,

$$[\mathbf{A}_0]+[\mathbf{A}_1] = q_a,\ \ [\mathbf{B}_0]+[\mathbf{B}_1] = q_b.$$

With these variables, Eq. 50 becomes:

$$\begin{aligned}
\frac{d[\mathbf{A}_0]}{dt} &= -kq_aq_b\cdot(1-a) \\
\frac{d[\mathbf{A}_1]}{dt} &= -kq_aq_b\cdot a \\
\frac{d[\mathbf{B}_0]}{dt} &= -kq_aq_b\cdot(1-b) \\
\frac{d[\mathbf{B}_1]}{dt} &= -kq_aq_b\cdot b \\
\frac{d[\mathbf{C}_0]}{dt} &= kq_aq_b\cdot[(1-a)(1-b)+(1-a)b+a(1-b)] \\
\frac{d[\mathbf{C}_1]}{dt} &= kq_aq_b\cdot ab.
\end{aligned} \tag{51}$$

Let us prove the time invariance of a and b. We can express $[\mathbf{A}_1]$ as $a\cdot q_a$, therefore according to the chain rule for derivatives,

$$\frac{d[\mathbf{A}_1]}{dt} = q_a\frac{da}{dt} + a\frac{dq_a}{dt}. \tag{52}$$

According to Eq 51,

$$\frac{dq_a}{dt} = \frac{d[\mathbf{A}_1]}{dt} + \frac{d[\mathbf{A}_1]}{dt} = -kq_aq_b. \tag{53}$$

From Eq. 51, 52 and 53, we conclude that,

$$q_a \frac{da}{dt} = 0. \tag{54}$$

Since, during the process, q_a is not a constant equal to 0, we conclude that $\frac{da}{dt} = 0$. This proves the time invariance of a, that is to say, during the process, the fractional value encoded by $[\mathbf{A}_0]$ and $[\mathbf{A}_1]$ remains the same. Similarly, we can prove that b is time-invariant.

From here, we can calculate c for $t > 0$. Assume the initial concentration of $[\mathbf{C}_0]$ and $[\mathbf{C}_1]$ are 0, then

$$\begin{aligned} c &= \frac{[\mathbf{C}_1]}{[\mathbf{C}_0] + [\mathbf{C}_1]} \\ &= \frac{\int_0^t \frac{d[\mathbf{C}_1]}{dt} dt}{\int_0^t \frac{d[\mathbf{C}_0]}{dt} dt + \int_0^t \frac{d[\mathbf{C}_1]}{dt} dt} \\ &= \frac{\int_0^t k q_a q_b \cdot ab \cdot dt}{\int_0^t k q_a q_b dt} \\ &= \frac{ab \int_0^t k q_a q_b dt}{\int_0^t k q_a q_b dt} (\text{since } a, b \text{ are constant}) \\ &= ab. \end{aligned} \tag{55}$$

This proves than an AND gate implements multiplication.

2.6 Error Analysis

We performed simulations to test the robustness of CRNs implementing stochastic functions with the program *Mathematica* [78]. We generated differential equations corresponding to the reaction kinetics for CRNs, and investigated the impact of varying reaction rates. Here we present a detailed analysis for a specific CRN, one that implements the polynomial:

$$f(x, y, z) = x + y + z - 2xy - 2xz - 2yz + 4xyz. \tag{56}$$

This polynomial is generated by the following truth table:

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

To see this, take the sum of the expressions for the minterms, i.e., the rows that evaluate to one. Recall that the expression for each row is formed by multiplying together factors corresponding to the input variables: x if the variable x is equal to 1 or $1-x$ f the variable x is equal to 0:

$$\begin{aligned} f(x,y,z) &= (1-x)(1-y)z + \\ &\quad (1-x)y(1-z) + \\ &\quad (1-x)y(1-z) + \\ &\quad x(1-y)(1-z) + \\ &\quad xyz \\ &= x+y+z-2xy-2xz-2yz+4xyz. \end{aligned} \tag{57}$$

According to the method discussed in Section 2.3, we can translate this truth table into a CRN as follows:

$$\begin{aligned} \mathbf{X}_0 + \mathbf{Y}_0 + \mathbf{Z}_0 &\xrightarrow{k_1} \mathbf{F}_0 \\ \mathbf{X}_0 + \mathbf{Y}_0 + \mathbf{Z}_1 &\xrightarrow{k_2} \mathbf{F}_1 \\ \mathbf{X}_0 + \mathbf{Y}_1 + \mathbf{Z}_0 &\xrightarrow{k_3} \mathbf{F}_1 \\ \mathbf{X}_0 + \mathbf{Y}_1 + \mathbf{Z}_1 &\xrightarrow{k_4} \mathbf{F}_0 \\ \mathbf{X}_1 + \mathbf{Y}_0 + \mathbf{Z}_0 &\xrightarrow{k_5} \mathbf{F}_1 \\ \mathbf{X}_1 + \mathbf{Y}_0 + \mathbf{Z}_1 &\xrightarrow{k_6} \mathbf{F}_0 \\ \mathbf{X}_1 + \mathbf{Y}_1 + \mathbf{Z}_0 &\xrightarrow{k_7} \mathbf{F}_0 \\ \mathbf{X}_1 + \mathbf{Y}_1 + \mathbf{Z}_1 &\xrightarrow{k_8} \mathbf{F}_1 \end{aligned} \tag{58}$$

Note that the indices of the molecular species match the entries in the truth table above. Since we will be exploring the consequences of non-uniform rate constants, note that here we have assigned the eight reactions unique rate constants: $k_1, k_2, \ldots k_8$, respectively. We can verify that this CRN implements the function in Eq. 56 through the differential equations. We define the following stochastic variables:

$$x = \frac{[\mathbf{X}_1]}{[\mathbf{X}_0]+[\mathbf{X}_1]},\ y = \frac{[\mathbf{Y}_1]}{[\mathbf{Y}_0]+[\mathbf{Y}_1]},\ z = \frac{[\mathbf{Z}_1]}{[\mathbf{Z}_0]+[\mathbf{Z}_1]},\ f = \frac{[\mathbf{F}_1]}{[\mathbf{F}_0]+[\mathbf{F}_1]}. \tag{59}$$

We used the procedure `NDSolveValue` in *Mathematica* to simulate the differential equations corresponding to CRN in Eq. 2.6. We varied the rate constants as well as the initial concentrations. We compare the value of f computed by the CRN, in terms of $[F_0], [F_1]$ to the expected value of f from Eq. 56. Here is a summary of the trials:

2.6.1 Trials for Error Analysis

1. With all $k_i = 100$ except for $k_1 = 1000$, i.e., one rate constant being an order of magnitude higher than the others: the highest error observed was 0.31, with 38.1% of the input combinations having an error greater than 0.1.

2. With all $k_i = 100$ except for $k_1 = 10$, i.e., one rate constant being an order of magnitude lower than the others: the highest error observed was 0.12, with 15.7% of the input combinations having an error greater than 0.1.

3. With all all $k_i = 100$ except for $k_1 = 10000$, i.e., one rate constant being two orders of magnitude higher than the others: the highest error observed was 0.45, with 45.8% of the input combinations having an error greater than 0.1.

4. With all $k_i = 100$ except for $k_1 = 1$, i.e., one rate being two orders of magnitude lower than the others: the highest error recorded was 0.12, with 22.7% of the input combinations having an error greater than 0.1.

5. With all k_i randomly generated, from a normal distribution with a mean of 100 and a low standard deviation of 10: the highest error recorded was 0.06, with no input combinations having an error greater than 0.1.

6. With all k_i randomly generated, from a normal distribution with a mean of 100 and a high standard deviation of 70 (negative values were not allowed): the highest error recorded was 0.25, with 14.4% of the input combinations having an error greater than 0.1.

The absolute difference between the output value of f, calculated with Eq. 59, compared to the expected value of f from Eq. 56 was calculated for a wide range of input concentrations. These are graphed in Figure 21. The inputs x, y, and z, calculated with Eq. 59, were set to values in the interval $[0, 1]$ forming a cube mesh input. All input chemical species were initialized such that $[\mathbf{X}_0] + [\mathbf{X}_1] = 100$. The maximum error difference and the number of input combinations for which the error differential exceeded 0.1 were recorded. The purpose of this simulation was not to account for all possible values of the rate constants, but rather to understand the design constraints and the error margins. The key observations from our simulations are:

1. In a network with many reactions, one rate constant being slower than the others by an order of magnitude or two has a lower impact on error than if it were faster by a similar amount.

2. Error rates are low if all the rate constants are within the same order of magnitude and are distributed normally with a small standard deviation.

3. Error rates are also low when some of the fractional inputs are close to 0 or to 1. This translates to very slow or very fast reactions, respectively.

4. Even when the rate constants differ by orders of magnitude, not all inputs result in high errors. Simulation is a valuable guide.

2.7 Implementation using DNA

2.7.1 DNA Strand-Displacement

DNA strand displacement is a well-established technique for implementing molecular computation [2, 79]. Prior work has shown that such a system can emulate *any* abstract set of chemical reactions. The reader is referred to Soloveichik et al. and Zhang et al. for further details [3, 12]. Here we illustrate a simple, generic example. In Section 2.7.1, we discuss how to map our models to such DNA strand-displacement systems.

We begin by first defining a few basic concepts. DNA strands are linear sequences of four different nucleotides $\{A, T, C, G\}$. Nucleotide can bind to another following *Watson-Crick* base-pairing: A binds to T, C binds to G. A pair of single DNA strands will bind to each other, a process called *hybridization*, if their sequences are complementary according to the base-pairing rule, that is to say, whereever there is an A in one, there is a T in the other, and vice versa; and whenever there is a C in one, there is a G in the other and vice-versa. The binding strength depends on the length of the complementary regions. Longer regions will bind strongly, smaller ones weakly. Reaction rates match binding strength: hybridization completes quickly if the complementary regions are long and slowly if they are short. If the complementary regions are very short, hybridization might not occur at all. (In this discussion, for simplicity, we are omitting many relevant details such temperature, concentration, and the distribution of nucleotide types – in particular, the fraction of paired bases that are A-T versus C-G, since C-G pairs are stronger. All of these parameters must be accounted for in realistic simulation runs.)

Figure 22 illustrates strand displacement with a set of reversible reactions. The entire reaction occurs as reactant molecules A and B form products E and F, with each intermediate stages operating on molecules C and D. In the figure, A and F are single strands of DNA, while B, C, D, and E are double-stranded complexes. Each single-strand DNA molecule is divided, conceptually, into subsequences that we call **domains**, denoted as 1, 2, and 3 in the figure. The complementary sequences for these domains are $1^*, 2^*$ and 3^*. (We will use this notation for complementarity throughout.) All distinct domains are assumed to be *orthogonal* to each other, meaning that these domains do not hybridize. **Toeholds** are a specific kind of domain in a double-stranded DNA complex where a single strand is exposed. For instance, the molecule B contains a toehold domain at 1^* in Figure 22. Toeholds are usually 6 to 10 nucleotides long, while the lengths of regular domains are typically 20 nucleotides. The exposed strand of a toehold domain can bind the complementary domain from a longer single DNA strand, and thus toeholds can trigger the binding and displacement of DNA strands. The small length of the toehold makes this hybridization reversible.

In the first reaction in Figure 22, the open toehold 1^* in molecule B binds with domain 1 from strand A. This forms the molecule C where the duplicate 2 domain section from molecule A forms an overhanging flap. This reaction exhibits how a toehold triggers the binding of

DNA strands. In molecule C, the overhanging flap can stick onto the complementary domain 2^*, thus displacing the previously bound strand. This type of branch migration is shown in the second reaction, where the displacement of one flap to the other forms the molecules D. This reaction is reversible, and the molecules C and D exist in a dynamic equilibrium. The process of branch migration of the flap is essentially a random walk: at any time when part of the strand from molecule A hybridizes with strand B, more of A might bind and displace a part of F, or more of F might bind and displace a part of A. Therefore, this reaction is reversible. The third reaction is the exact opposite of reaction 1 – the new flap in molecule D can peel off from the complex and thus create the single strand molecule F and leave a new double-stranded complex E. Molecule E is similar to molecule B, but the toehold has migrated from 1^* to 3^*. The reaction rate of this reaction depends on the length of the toehold 3^*. If we reduce the length of the toehold, the rate of reaction 3 becomes so small that the reaction can be treated as a forward-only reaction. This bias in the direction of the reaction means that we can model the entire set of reactions as a single DNA strand displacement event, where reactants A and B react to produce E and F. Note that the strand F can now participate in further toehold-mediated reactions, allowing for cascading of such these DNA strand displacement systems.

2.8 DNA Concatemers

DNA Concatemers are long strands of DNA that contain repeated base-pair sequences. These are formed when a single smaller DNA unit is capable of hybridizing with other copies of itself. Specifically, to form a DNA strand of the form $\mathbf{ABABAB}\ldots$, the 1-mer unit must have the following 3 regions:

1. A leading sticky end (single-stranded region) on the 1st strand with the sequence $\mathbf{A}$.
2. A middle double-stranded section with the sequence $\mathbf{B}$.
3. A trailing sticky end on the 2nd strand with the complement sequence $\mathbf{A}'$ such that it can bind to a leading sticky end for $\mathbf{A}$.

We propose designing our molecules for fractional representation as DNA concatemers [80] that can interact via strand displacement, as detailed in the next subsection. For a fractional variable a, the molecules $\mathbf{A}_0$ and $\mathbf{A}_1$ needed for the reaction network can be designed as concatemer units such that the double-stranded section for each unit is distinct, but the sticky ends for both of them are the same. This allows the two species to cross-polymerize and forms a linear chain of DNA of randomly arranged $\mathbf{A}_0$ and $\mathbf{A}_1$ units. This is similar to the randomized digital bitstreams used in stochastic computing in which a random stream of 0's and 1's forms the basic data unit [45, 55]: $\mathbf{A}_0$ and $\mathbf{A}_1$ correspond to 0 and 1, respectively. Thus a single fractional variable can be stored as a long DNA strand that can be amplified to improve readout [81]; or the reactant concentration and can easily be broken up using artificial restriction enzymes – or natural restriction enzymes, if the sticky ends are designed purposefully. Furthermore, this concatemer design allows the use of RNA-seq [82] in the readout process to measure the fractional value stored by a DNA strand. For this purpose, a

long DNA concatemer must be broken into its constituent units using a restriction enzyme, and then these smaller DNA units can be used instead of the standard complementary DNA in RNA-seq to determine the expression level of each unit. From this quantitative readout, the relative amount of $\mathbf{A}_1$ unit to $\mathbf{A}_0 + \mathbf{A}_1$ can be determined .

2.8.1 Procedure

Figure 23 illustrates the reaction $\mathbf{A}_i + \mathbf{B}_j \rightarrow \mathbf{C}_k$ implemented with DNA strand displacement and cleaving enzymes. Two species of concatemer units are transformed into another concatemer unit. The implementation consists of three stages:

1. Extracting single strands: Consider the two input concatemers $\mathbf{A}_i$ and $\mathbf{B}_j$ shown in the figure. We design the concatemers in such a way that the sticky ends of a concatemer unit can act like open toeholds in DNA strand displacement. As a result, we can extract a single strand from a concatemer. For example, concatemer $\mathbf{A}_i$ is formed with two single strands $[T_i, A_i]$, $[A_i^*, T_1^*]$. We can add strand $[A_i, T_1]$ so that strand $[T_1, A_i]$ is displaced. Similarly, we can extract strand $[T_2, B_j, T_3]$ from concatemer $\mathbf{B}_j$ with strand $[B_j, T_3, T_2]$.

2. This is the strand displacement reaction that implements the main reaction. It receives two single-strand DNA molecules, $[T_1, A_i]$ and $[T_2, B_j, T_3]$ as reactants. The product is a complex containing the output concatemer. The reaction is divided into two parts. In the first part, strand $[T_1, A_i]$ displaces strand $[A_i, T_2]$ from the auxiliary complex $\mathbf{G}_1$ and forms $\mathbf{G}_2$ through a reversible reaction. Then the strand $[T_2, B_j, T_3]$ displaces the output complex which is formed by strand $[B_j, T_3, C_k]$ and $[C_k^*, T_3^*]$. This step is irreversible since the output complex cannot bind to the resulting auxiliary complex $\mathbf{G}_3$ after this step.

3. Cleaving. The output complex from the previous step contains the domain B_j in addition to the part that could form concatemer $\mathbf{C}_k$. The domain B_j is cleaved from the complex. After this step, we get a concatemer $\mathbf{C}_k$ with T_3 sticky end. Cleaving can be achieved by using DNA editing enzymes such as CRISPR-Cas9 and PfAgo [9].

We assume that the concentration of the initial auxiliary complex $\mathbf{G}_1$ is much larger than the concentration of the concatemers. With this assumption, the concentration of the auxiliary complex can be treated invariant through the reaction. Thus, the reaction rate only depends on the concentration of the single strands extracted from the concatemers. As there are four reactions to implement the two-input network shown in this example, four species of the auxiliary complex representing each reaction should be used. This ensures that the mixture of different species of $\mathbf{A}_0$ and $\mathbf{A}_1$, or $\mathbf{B}_0$ and $\mathbf{B}_1$, can react competitively. During the cleaving step, each reactant participates in only one reaction. Therefore, it should not affect the reaction rate or the fractional encoding of the output by the two product species.

The reaction itself can be extended to a multimolecular reaction by extending the chain of toehold exchange reactions. For example, in the complex $\mathbf{G}_1$, domains $[T_4, D_l]$ and their complementary domains can be added between the domains B_j and T_3. In this way, $\mathbf{G}_1$ would be capable of receiving an additional strand $[T_4, D_l]$ before displacing the final product.

2.9 Conclusion

This report proposed a strategy for computing mathematical functions with molecular systems based on a fractional representation, using a pair of molecular species to represent each mathematical variable. With this representation, we are able to apply the theory of stochastic logic to develop very efficient chemical reaction networks for computing functions. In particular, we showed how to translate the truth tables for stochastic functions into chemical reaction networks. We demonstrated how to implement the reaction networks with DNA strand displacement.

Stochastic logic is an intriguing paradigm for digital computation. Instead of computing definite outputs from definite inputs – say Boolean values from Boolean values, or integers from integers – it entails computing probabilities from probabilities. There is randomness and yet the computation is robust. The computation is effected by transforming the *statistical distribution* of random bitstreams. The paradigm has been applied in a variety of domains, particularly for emerging technologies [44, 84, 85, 86]. It has been most successful for applications that entail computing mathematical functions: for instance, `arctan` for nonlinear activation functions in machine learning; *Bessel* functions for differential system models; and the `sinc` function for image and signal processing. We give examples of CRN implementations of these functions in the Supporting Information. Of course, we cannot point to real-world applications that call for the molecular computation of such functions. For now, the ideas in this report should be taken as a proof of concept.

Over the past two decades, computing has moved from desktops and data centers into the wild. Embedded microchips – found in our gadgets, our tools, our buildings, our soils and even our bodies – are transforming our lives. And yet, there are limits to where silicon can go and where it can compute effectively. It operates based on voltage and so requires a power source. Even miniaturized to the microscale or smaller, an electronic system is often a foreign object inserted into a material, substrate, or environment. This sort of computation discussed in this report could find application in a novel class of computing system that is not foreign, but rather an integral part of its physical and chemical environment: a system that computes *with* its constituent molecules. In such a system, sensing, computing, and actuating occur at the molecular level, with no interfacing at all with external electronics. Futuristic, yes, but we can point to the field of *soft robotics* where such systems are being developed [87].

2.10 Supporting information

S1 Examples of CRNs for polynomial approximations of nonlinear functions. For the examples in this section, we consider polynomial approximations: 4th order Maclaurin series at $x = 0$.

ArcTan Function

Here we show an example of $\arctan(x)$. It can be approximated as:

$$\arctan(x) \approx x - \frac{1}{3}x^3 = x(1 - \frac{1}{3}x^2)$$

We assign the stochastic variables $x_1 = x, x_2 = \frac{1}{3}, x_3 = x_4 = x$. Then the stochastic logic

function is:

$$\text{AND}(x_1, \text{NAND}(x_2, \text{AND}(x_3, x_4)))$$

According to the truth table, the corresponding CRN is:

$$\begin{aligned}
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} &\to \mathbf{Y}_0
\end{aligned}$$

Exponential Function

Here we show an example of $\exp(-x)$. Its approximation is:

$$\exp(-x) \approx 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 = 1 - x(1 - \frac{1}{2}x(1 - \frac{1}{3}x))$$

We assign the stochastic variables $x_1 = x, x_2 = \frac{1}{2}, x_3 = x, x_4 = \frac{1}{3}, x_5 = x$. Then the stochastic logic function is:

$$\text{NAND}(x_1, \text{NAND}(x_2, \text{AND}(x_3, \text{NAND}(x_4, x_5))))$$

According to the truth table, the corresponding CRN is:

$$\begin{aligned}
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_1
\end{aligned}$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$

Bessel Function

Here we show an example of the Bessel function of the first kind with parameter $\alpha = 1$. Its approximation is:

$$J_1(x) \approx \frac{1}{2}x - \frac{1}{16}x^3 = \frac{1}{2}x(1 - \frac{1}{8}x^2)$$

We assign the stochastic variables $x_1 = \frac{1}{2}, x_2 = x, x_3 = \frac{1}{8}, x_4 = x_5 = x$. Then the stochastic logic function is:

$$\mathrm{AND}(x_1, \mathrm{AND}(x_2, \mathrm{NAND}(x_3, \mathrm{AND}(x_4, x_5))))$$

According to the truth table, the corresponding CRN is:

$$\begin{aligned}
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} &\to \mathbf{Y}_0 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} &\to \mathbf{Y}_1 \\
\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} &\to \mathbf{Y}_1
\end{aligned}$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} \to \mathbf{Y}_0$$

Sinc Function

The mathematical expression of the sinc function is:

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$

And its approximation is:

$$\text{sinc}(x) \approx 1 - \frac{1}{6}x^2 + \frac{1}{120}x^4 = 1 - \frac{1}{6}x^2(1 - \frac{1}{20}x^2)$$

We assign the stochastic variables $x_1 = x_2 = x, x_3 = \frac{1}{6}, x_4 = \frac{1}{20}, x_5 = x_6 = x$. Then the stochastic logic function is:

$$\text{NAND}(\text{AND}(x_1, x_2), \text{AND}(x_3, \text{NAND}(x_4, \text{AND}(x_5, x_6))))$$

According to the truth table, the corresponding CRN is:

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,0} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,0} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,0} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,0} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,0} + \mathbf{X}_{6,1} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,0} \to \mathbf{Y}_0$$
$$\mathbf{X}_{1,1} + \mathbf{X}_{2,1} + \mathbf{X}_{3,1} + \mathbf{X}_{4,1} + \mathbf{X}_{5,1} + \mathbf{X}_{6,1} \to \mathbf{Y}_1$$

References

[1] L. M. Adleman, *Science* **266**, 1021 (1994).

[2] B. Yurke, *Nature* **406** (2000).

[3] D. Soloveichik, G. Seelig, E. Winfree, *Proceedings of the National Academy of Sciences* **107**, 5393 (2010).

[4] L. Qian, E. Winfree, *Journal of the Royal Society Interface* (2011).

[5] D. Ielmini, H.-S. P. Wong, *Nature electronics* **1**, 333 (2018).

[6] B. Wang, C. Chalk, D. Soloveichik, *DNA25: International Conference on DNA Computing and Molecular Programming*, Springer (LNCS, 2019), vol. 11648, pp. 219–235.

[7] T. Chen, A. Solanki, M. Riedel, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, M. R. Lakin, P. Šulc, eds. (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021), vol. 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 11:1–11:21.

[8] M. J. Flynn, *IEEE Transactions on Computers* **C-21**, 948 (1972).

[9] S. K. Tabatabaei, *et al.*, *Nature communications* **11**, 1 (2020).

[10] N. Athreya, O. Milenkovic, J.-P. Leburton, *Biophysical Journal* **116**, 292a (2019).

[11] K. Liu, *et al.*, *Nature Communications* **10**, 3 (2019).

[12] D. Y. Zhang, G. Seelig, *Nature Chemistry* **3**, 103 (2011).

[13] D. Doty, A. Ong (2021).

[14] T. Chen, A. Solanki, M. Riedel, *27th International Conference on DNA Computing and Molecular Programming (DNA 27)*, M. R. Lakin, P. Šulc, eds. (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021), vol. 205 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 11:1–11:21.

[15] S. Tabatabaei Yazdi, Y. Yuan, J. Ma, H. Zhao, O. Milenkovic, *Scientific reports* **5**, 1 (2015).

[16] P. W. Rothemund, *Nature* **440**, 297 (2006).

[17] R. R. F. Machinek, T. E. Ouldridge, N. E. C. Haley, J. Bath, A. J. Turberfield, *Nature Communications* **5**, 5324 (2014).

[18] D. Y. Zhang, *DNA Computing and Molecular Programming*, Y. Sakakibara, Y. Mi, eds. (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011), vol. 6518, pp. 162–175.

[19] B. R. Wolfe, N. J. Porubsky, J. N. Zadeh, R. M. Dirks, N. A. Pierce, *Journal of the American Chemical Society* **139**, 3134 (2017).

[20] C. G. Evans, E. Winfree, *DNA Computing and Molecular Programming*, D. Soloveichik, B. Yurke, eds. (Springer International Publishing, Cham, 2013), vol. 8141, pp. 61–75.

[21] S. Wolfram, *Reviews of Modern Physics* **55**, 601 (1983).

[22] M. Cook, *Complex Systems* **15**, 1 (2004).

[23] L. Organick, *et al.*, *Nature Biotechnology* **36**, 242 (2018).

[24] C. Bee, *et al.*, *Nature Communications* **12**, 4764 (2021).

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms, Third Edition* (The MIT Press, 2009), third edn.

[26] J. von Neumann, *Automata Studies. (AM-34)*, C. E. Shannon, J. McCarthy, eds. (Princeton University Press, 1956), pp. 43–98.

[27] C. Thachuk, E. Winfree, D. Soloveichik, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Springer Verlag, 2015), vol. 9211, pp. 133–153.

[28] B. Wang, C. Thachuk, A. D. Ellington, E. Winfree, D. Soloveichik, *Proceedings of the National Academy of Sciences of the United States of America* **115**, E12182 (2018).

[29] G. M. Church, Y. Gao, S. Kosuri, *Science* **337**, 1628 (2012).

[30] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, W. J. Stark, *Angewandte Chemie International Edition* **54**, 2552 (2015).

[31] N. V. Ivanova, M. L. Kuzmina, *Molecular Ecology Resources* **13**, 890 (2013).

[32] A. F. De Fazio, *et al.*, *ACS Nano* **13**, 5771 (2019).

[33] R. A. Hughes, A. D. Ellington, *Cold Spring Harbor Perspectives in Biology* **9**, a023812 (2017).

[34] Ultramer™ DNA Oligonucleotides, https://www.idtdna.com/pages/products/custom-dna-rna/dna-oligos/ultramer-dna-oligos.

[35] F. Praetorius, *et al.*, *Nature* **552**, 84 (2017).

[36] B. Wang, C. Chalk, D. Soloveichik, *DNA Computing and Molecular Programming*, C. Thachuk, Y. Liu, eds. (Springer International Publishing, Cham, 2019), pp. 219–235.

[37] N. L. Dabby, Synthetic Molecular Machines for Active Self-Assembly: Prototype Algorithms, Designs, and Experimental Study, Ph.D. thesis, California Institute of Technology (2013).

[38] X. Chen, *Journal of the American Chemical Society* **134**, 263 (2012).

[39] A. J. Genot, D. Y. Zhang, J. Bath, A. J. Turberfield, *Journal of the American Chemical Society* **133**, 2177 (2011).

[40] S. Palluk, *et al.*, *Nature Biotechnology* **36**, 645 (2018).

[41] H. H. Lee, R. Kalhor, N. Goela, J. Bolot, G. M. Church, *Nature Communications* **10**, 2383 (2019).

[42] S. Tabatabaei, *et al.*, *Nature Communications* **11** (2020).

[43] J. Von Neumann, *Automata studies* **34**, 43 (1956).

[44] N. R. Shanbhag, R. A. Abdallah, R. Kumar, D. L. Jones, *Proceedings of the 47th Design Automation Conference* (2010), pp. 859–864.

[45] B. Gaines, *Advances in Information Systems Science* (Plenum Press, 1969), vol. 2, chap. 2, pp. 37–172.

[46] W. J. Poppelbaum, A. Dollas, J. B. Glickman, C. Otoole, *Advances in Computers*, M. C. Yovits, ed. (1976), vol. 17, pp. 187–230.

[47] B. Brown, H. Card, *IEEE Transactions on Computers* **50**, 891 (2001).

[48] W. Qian, **Marc Riedel**, *Design Automation Conference* (2008), pp. 648–653.

[49] A. Alaghi, J. P. Hayes, *ACM Transaction on Embedded Computing* **12** (2013).

[50] W. Qian, **Marc Riedel**, H. Zhou, J. Bruck, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (to appear)* (2011).

[51] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, W. J. Gross, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**, 2688 (2017).

[52] K. Cushon, C. Leroux, S. Hemati, S. Mannor, W. J. Gross, *IEEE Transactions on Circuits and Systems II: Express Briefs* **57**, 893 (2010).

[53] M. H. Najafi, D. Jenson, D. J. Lilja, M. D. Riedel, *IEEE Tran. on Very Large Scale Integration (VLSI) Systems* (2019).

[54] W. Qian, Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams, Ph.D. thesis, University of Minnesota (2011).

[55] W. Qian, X. Li, **Marc Riedel**, K. Bazargan, D. J. Lilja, *IEEE Transactions on Computers* **60**, 93 (2011).

[56] **M. Hassan Najafi**, *et al.*, *J. Emerg. Technol. Comput. Syst.* **13**, 57:1 (2017).

[57] L. Adleman, *Science* **266**, 1021 (1994).

[58] M. Cook, D. Soloveichik, E. Winfree, J. Bruck, *Algorithmic Bioprocesses*, A. Condon, D. Harel, J. N. Kok, A. Salomaa, E. Winfree, eds. (Springer, 2009), pp. 543–584.

[59] D. Soloveichik, M. Cook, E. Winfree, J. Bruck, *Natural Computing* **7** (2008).

[60] L. Qian, E. Winfree, *DNA Computing* (2009), pp. 70–89.

[61] H. Jiang, M. D. Riedel, K. K. Parhi, *IEEE Design & Test of Computers* **29**, 21 (2012).

[62] H. Jiang, , S. Salehi, M. D. Riedel, K. K. Parhi, *ACS Synthetic Biology* **2**, 245 (2013).

[63] M. N. Stojanovic, D. Stefanovic, S. Rudchenko, *Accounts of chemical research* **47**, 1845 (2014).

[64] J. C. Anderson, E. J. Clarke, A. P. Arkin, C. A. Voigt, *Journal of Molecular Biology* **355**, 619 (2006).

[65] S. Venkataramana, R. M. Dirks, C. T. Ueda, N. A. Pierce, *Proceedings of the National Academy of Sciences* (2010 (in press)).

[66] S. A. Salehi, M. Riedel, K. Parhi, *ACS Synthetic Biology* **6** (2017).

[67] S. A. Salehi, X. Liu, M. Riedel, K. Parhi, *Nature Scientific Reports* **8** (2018).

[68] S. N. Bernstein, *Communications of the Kharkov Mathematical Society* **13**, 1 (1912).

[69] W. Qian, **Marc Riedel**, I. Rosenberg, *European Journal of Combinatorics* **32**, 448 (2011).

[70] F. Horn, R. Jackson, *Archive for rational mechanics and analysis* **47**, 81 (1972).

[71] W. Qian, **Marc Riedel**, *DAC'08*, pp. 648–653.

[72] K. P. Parker, E. J. McCluskey, *IEEE Transactions on Computers* **24**, 668 (1975).

[73] J. Savir, G. Ditlow, P. H. Bardell, *IEEE Transactions on Computers* **33**, 79 (1984).

[74] J.-J. Liou, K.-T. Cheng, S. Kundu, A. Krstic, *Design Automation Conference* (2001), pp. 661–666.

[75] R. Marculescu, D. Marculescu, M. Pedram, *International Conference on Computer-Aided Design* (1994), pp. 294–299.

[76] D. Jenson, **Marc Riedel**, *International Conferences on Computer-Aided Design* (2016).

[77] M. H. Najafi, D. Jenson, D. J. Lilja, M. D. Riedel, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **27**, 2925 (2019).

[78] S. Wolfram, *Mathematica: a system for doing mathematics by computer* (Addison Wesley Longman Publishing Co., Inc., 1991).

[79] G. Seelig, D. Soloveichik, D. Y. Zhang, E. Winfree, *Science* (2006), vol. 314, pp. 1585–1588.

[80] L. Sun, B. Åkerman, *Computational and structural biotechnology journal* **11**, 66 (2014).

[81] U. Schlecht, J. Mok, C. Dallett, J. Berka, *Scientific reports* **7**, 1 (2017).

[82] Z. Wang, M. Gerstein, M. Snyder, *Nature reviews genetics* **10**, 57 (2009).

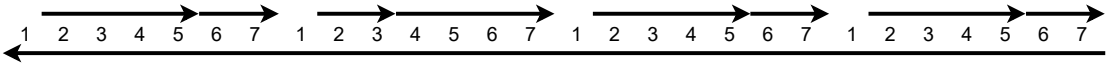
[83] M. Zuckermann, *et al.*, *Scientific reports* **8**, 1 (2018).

[84] R. Venkatesan, S. Venkataramani, X. Fong, K. Roy, A. Raghunathan, *Proceedings of the 2015 Design, Automation and Test in Europe Conference*, DATE '15 (EDA Consortium, San Jose, CA, USA, 2015), p. 1575–1578.

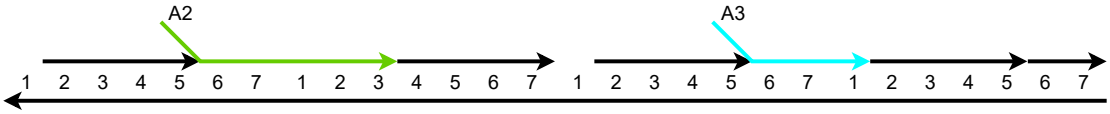
[85] X. Jia, *et al.*, *Spintronic Solutions for Stochastic Computing* (2019), pp. 165–183.

[86] **M. Hassan Najafi**, D. J. Lilja, *ASP-DAC 2017, 22nd Asia and South Pacific Design Automation Conference* (2017).

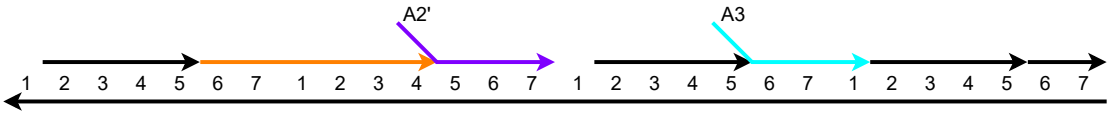
[87] D. Rus, M. T. Tolley, *Nature* **521**, 467 (2015).



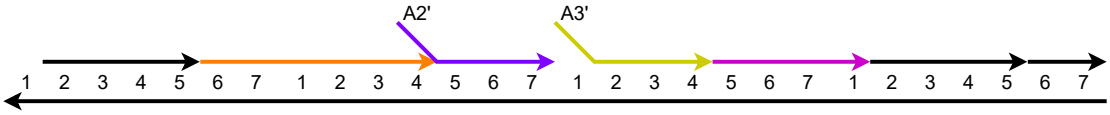
(a) Initial Sequence 1011



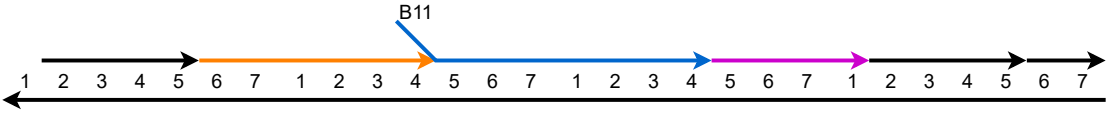
(b) Identifier A_2 captures first pair 10, A_3 captures second pair 11



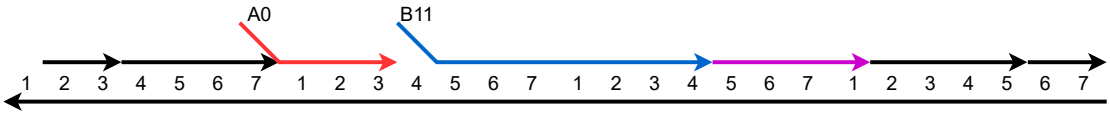
(c) covering the domain 1 between the two bit pairs



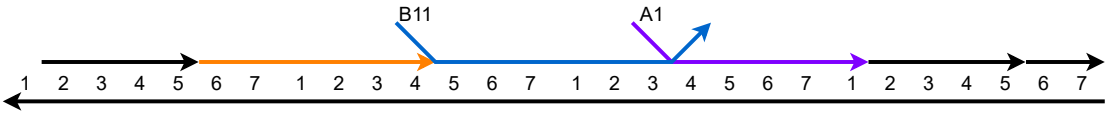
(d) Rewrite the content in the pair so that new identifiers are close to the middle



(e) Two identifier strands replaced by a single identifier if there is a perfect match



(f) Initial sequence is 0011. It will result in an open domain 4 in the cell left to the identifier



(g) Initial sequence is 1010. It will result in an open domain 4 on the identifier itself

Figure 13: Example implementation of search algorithm on target sequence 1011

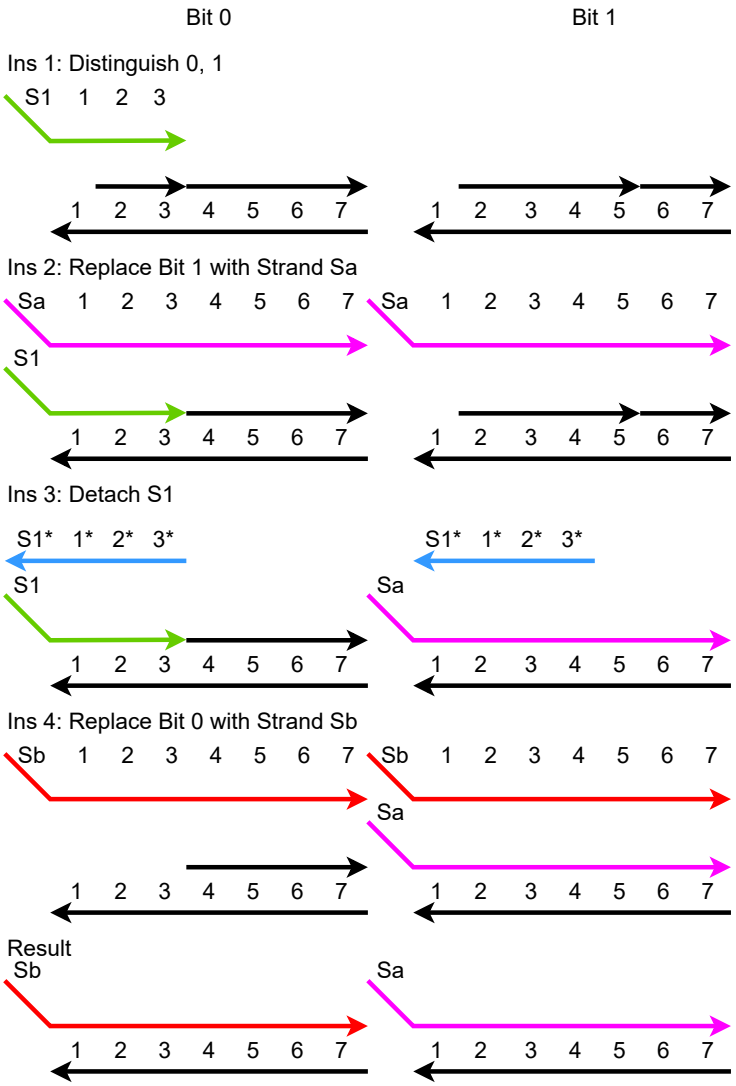


Figure 14: Current coding scheme could be converted to other coding scheme

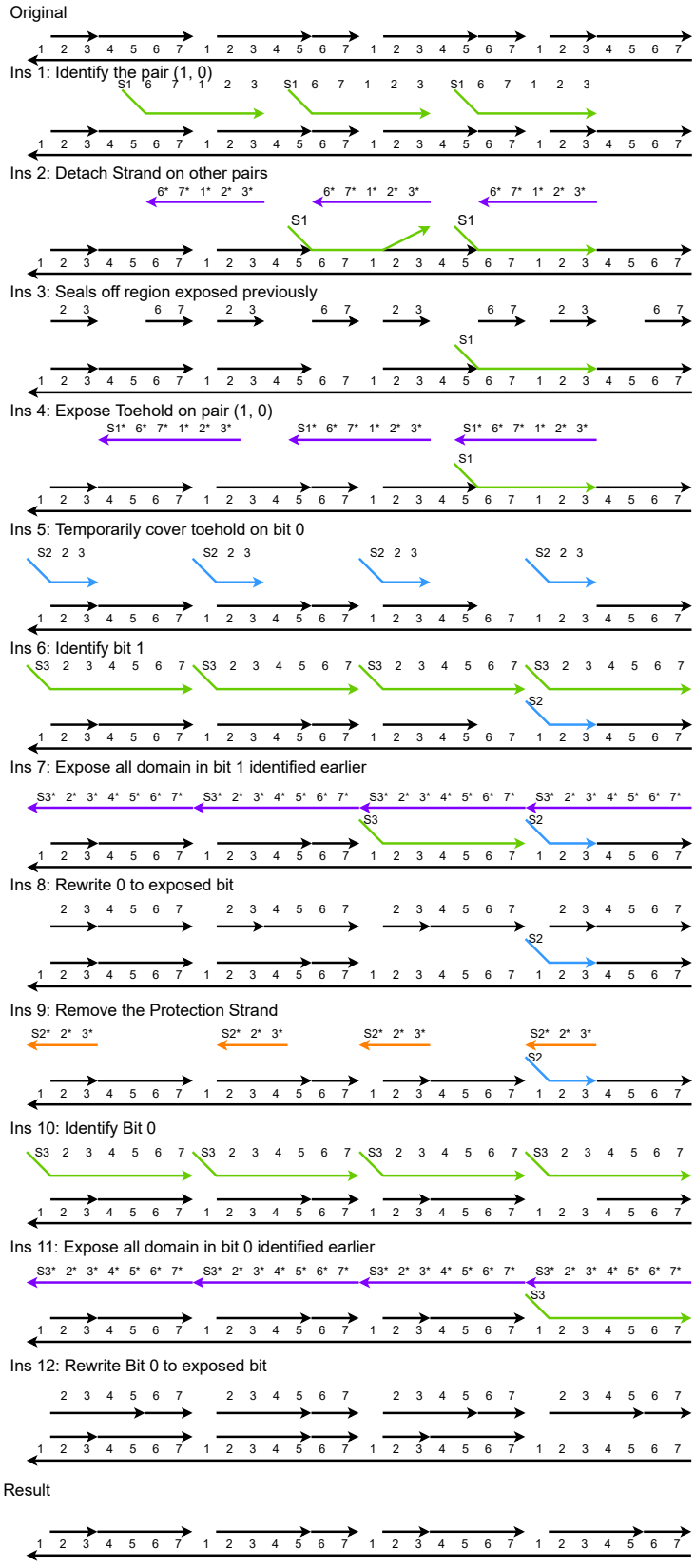


Figure 15: Instructions for Parallel Sorting

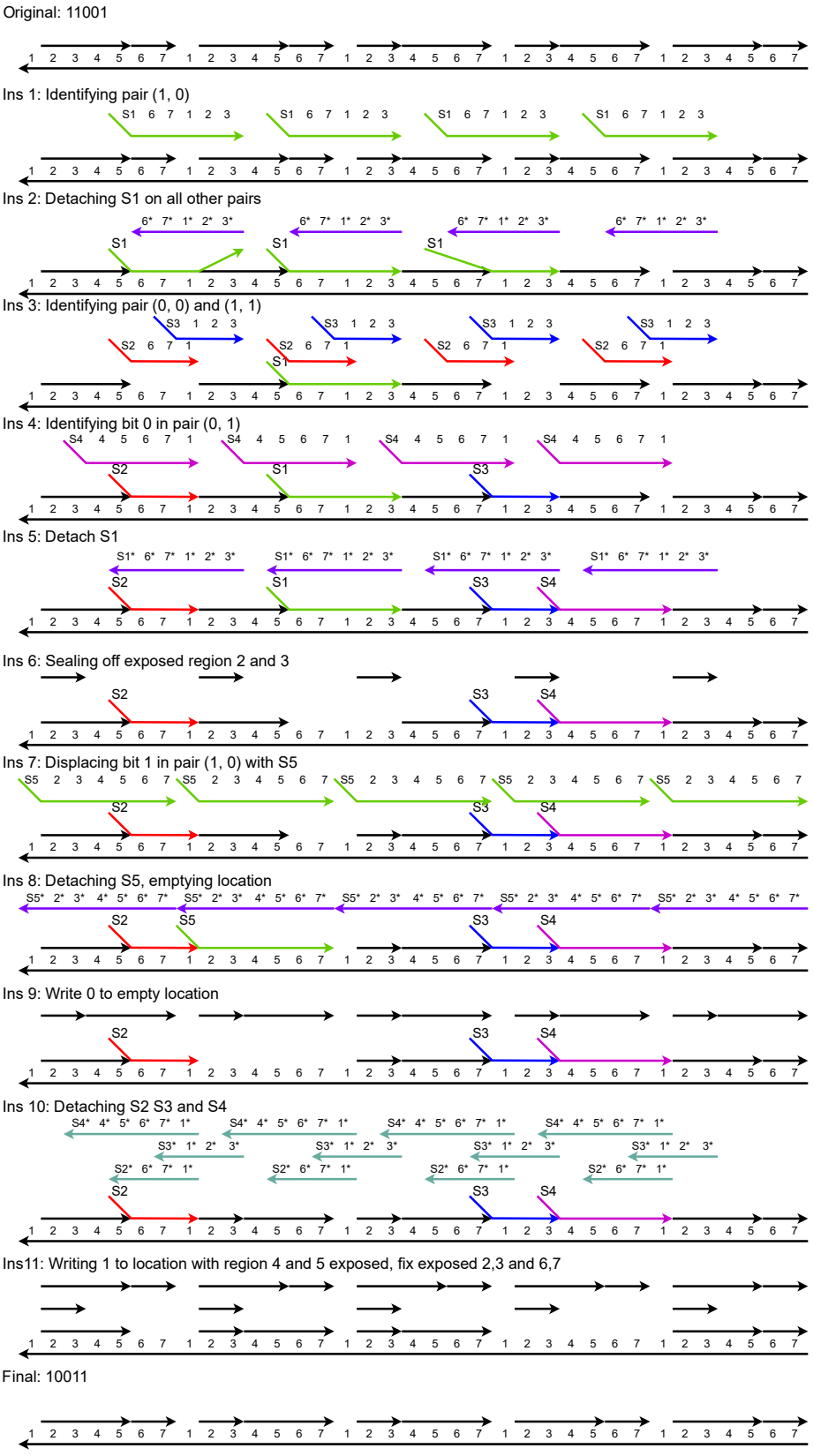


Figure 16: Instructions for the Left Shift cell

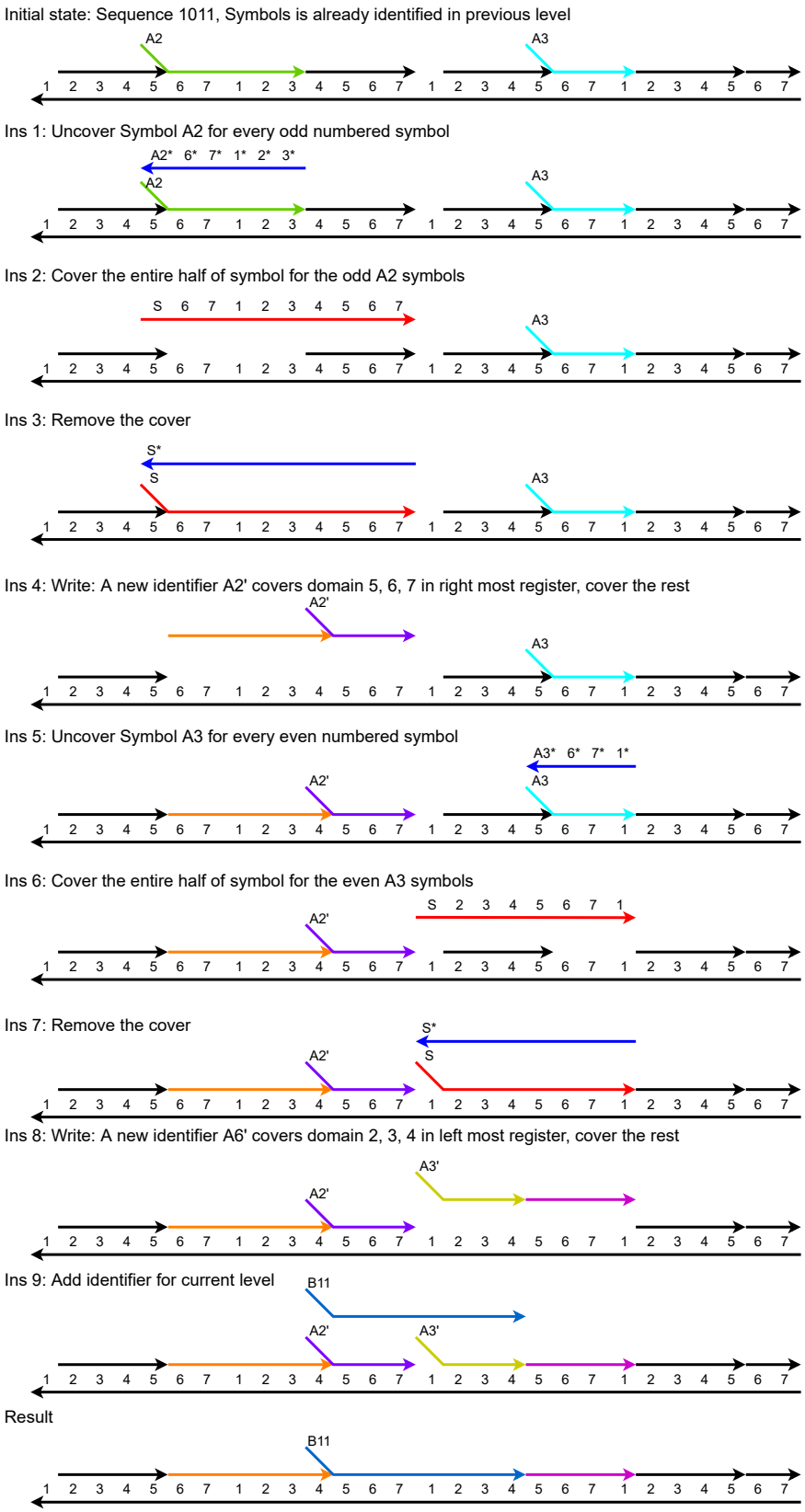


Figure 17: Instructions for a search operation of target sequence 1011

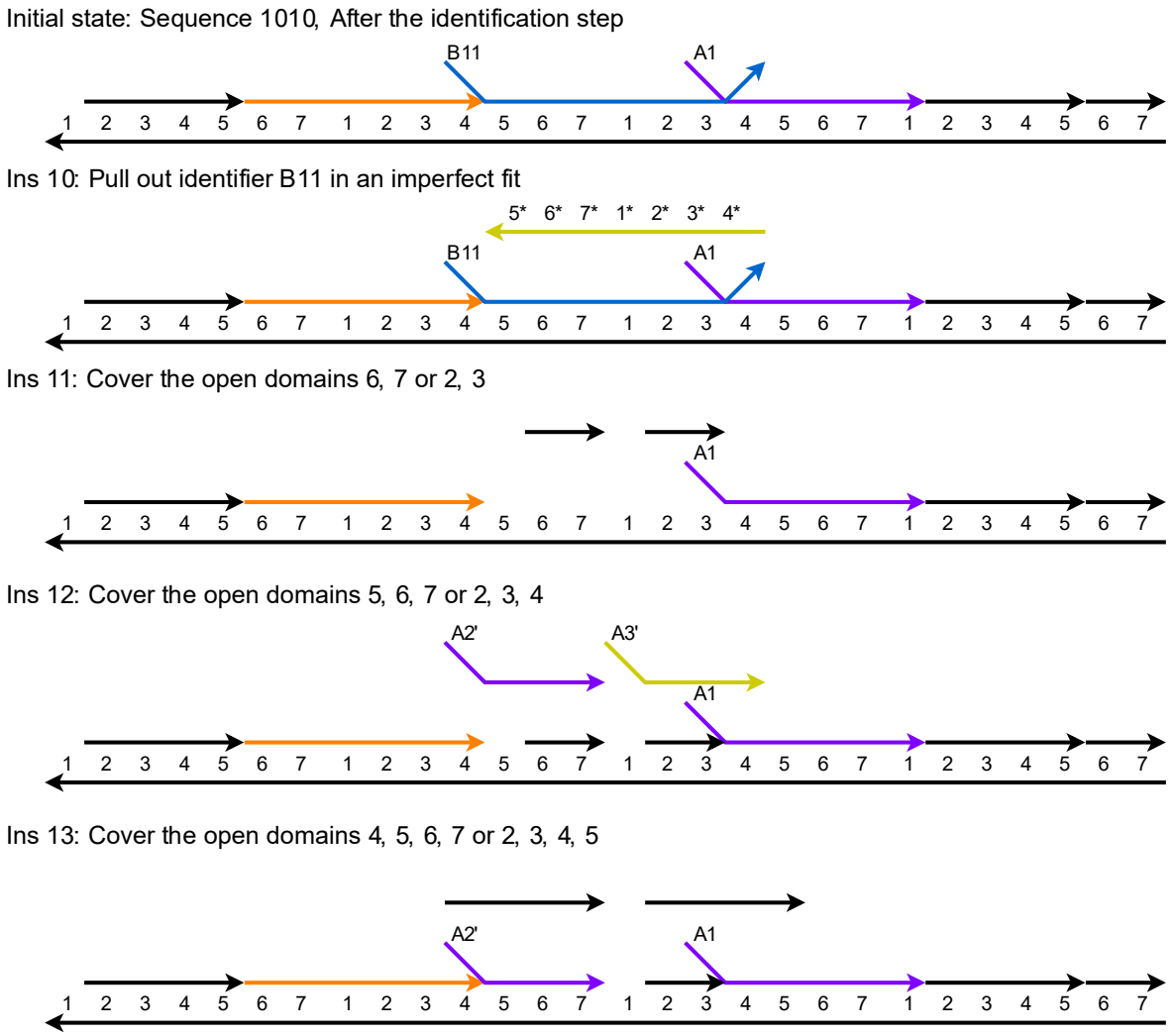


Figure 18: Instructions for the clean up process for a failed searching, these instructions won't affect the result of a successful search.

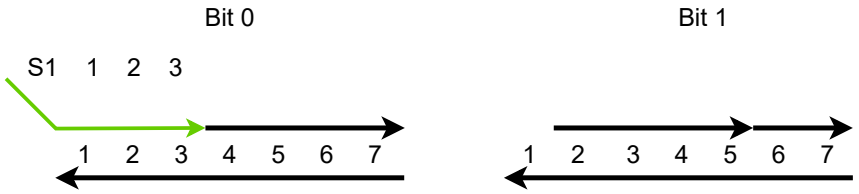


Figure 19: One strand could be used to differentiate two bits

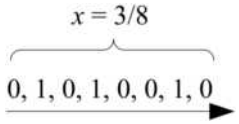


Figure 20: Stochastic representation: A random bitstream. A value $x \in [0, 1]$, in this case 3/8, is represented as a bitstream. The probability that a randomly sampled bit in the stream is one is $x = 3/8$; the probability that it is zero is $1 - x = 5/8$.

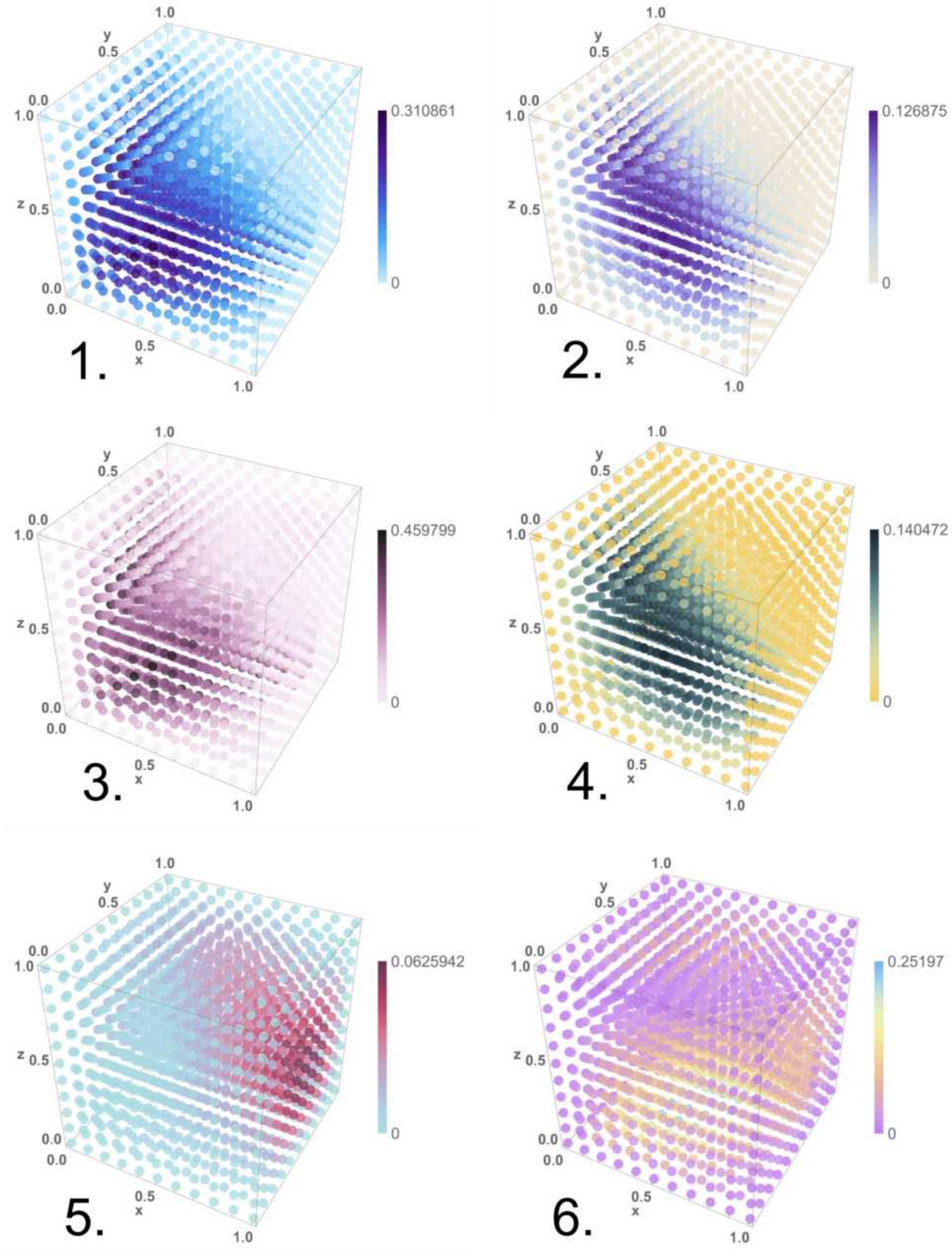


Figure 21: The error cubes for the six trials listed in Section 2.6.1. The three dimensions in the plots span the inputs x, y, and z, each in the interval $[0, 1]$, with a step size 0.1. The color of each point corresponds to the absolute difference between the value computed by the CRN and the expected value of f from Eq. 56. A legend is provided for each cube. The trials were performed with theNDSolveValue function in software tool *Mathematica*.

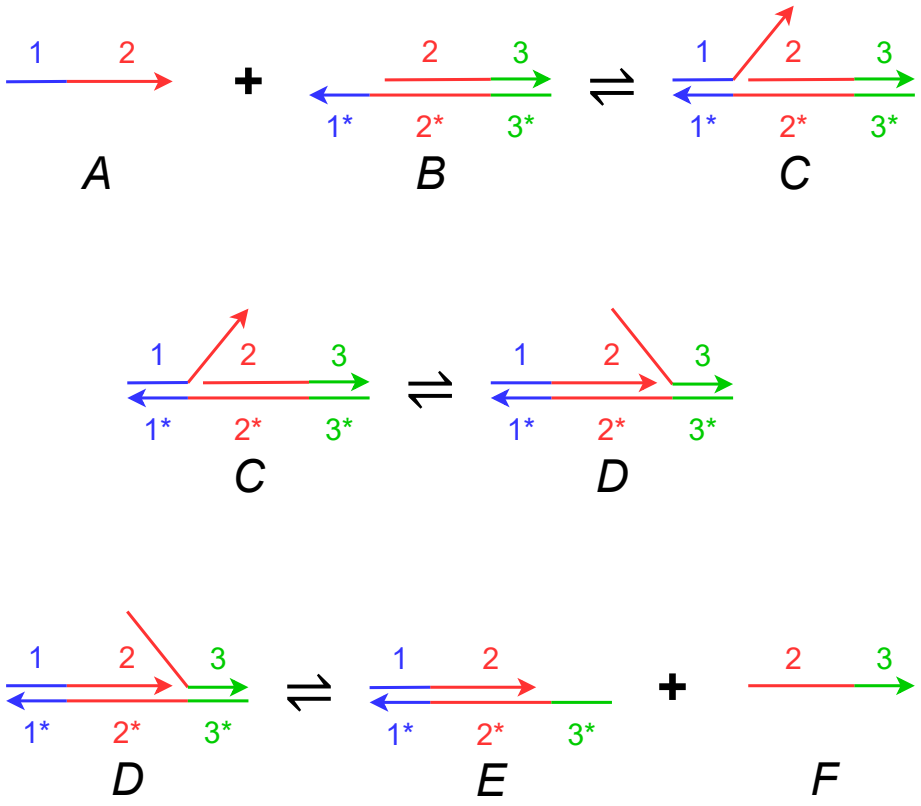


Figure 22: A set of DNA strand displacement reactions. Each DNA single strand is drawn as a continuous arrow, consisting of different colored domains numbered 1 through 3. DNA domains that are complementary to each other due to A-T, C-G binding are paired as 1 and 1^*. The first reaction shows reactant A and B hybridizing together via the toehold at domain 1^* on molecule B. The second reaction depicts branch migration of the overhanging flap of DNA in molecule C, thereby resulting in the nick migrating from after domain 1 to 2. The third reaction shows how an overhanging strand of DNA can be peeled off of molecule D, thereby exposing a toehold at domain 3^* on molecule E and releasing a freely floating strand F. All reactions are reversible. The only domains that are toeholds are 1^* and 3^*.

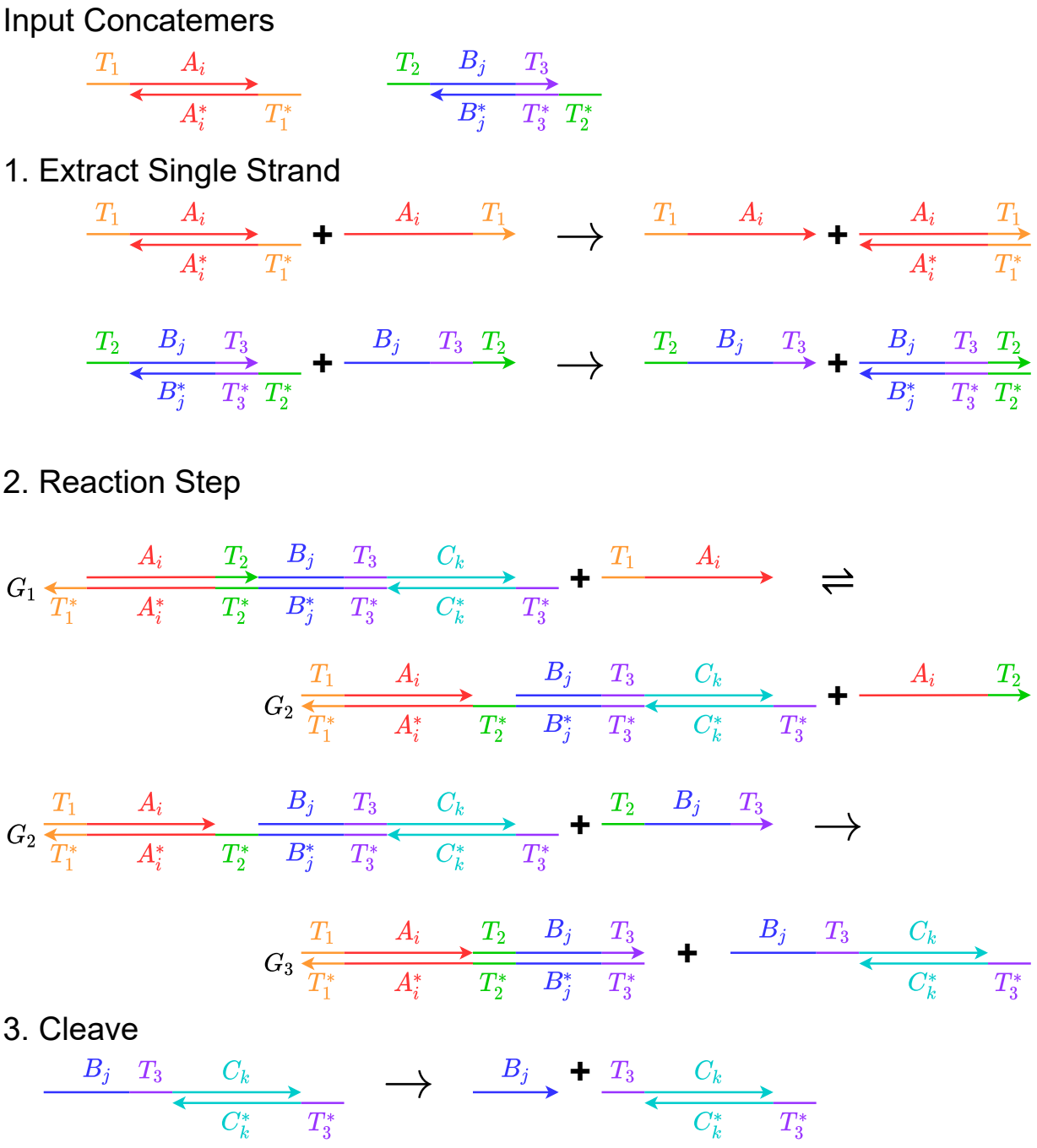


Figure 23: An example illustrating strand displacement reactions, implemented using concatemers. The figure is divided into an example sequence of concatemers, and three reaction steps: 1) extracting a single strand from concatemers; 2) a reaction step that consumes two single strands and outputs a complex; and 3) cleaving.