ROBUST DIGITAL NUCLEIC ACID MEMORY

by
Golam Md Mortuza

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computing
Boise State University

August 2023

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Golam Md Mortuza

Dissertation Title: ROBUST DIGITAL NUCLEIC ACID MEMORY

Date of Final Oral Examination: 10 July 2023

The following individuals read and discussed the dissertation submitted by student Golam Md Mortuza, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Tim Andersen Ph.D.	Chair, Supervisory Committee
William L. Hughes Ph.D.	Member, Supervisory Committee
Reza Zadegan Ph.D.	Member, Supervisory Committee

The final reading approval of the dissertation was granted by Tim Andersen Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

DEDICATION

To my late parents......

ACKNOWLEDGMENT

First and foremost, I am deeply grateful to my supervisor, Dr. Timothy Andersen, for his invaluable guidance, expertise, and encouragement that shaped every aspect of this research. His mentorship directed me on the right path and provided invaluable emotional support during challenging phases of my life. His profound impact will be cherished forever.

I sincerely thank the members of my dissertation committee, Dr. William Hughes and Dr. Reza Zadegan, for their valuable insights and constructive critiques, which greatly enhanced the quality of this work.

I would also like to acknowledge Dr. Michael Tobiason and Dr. George Dickinson, whose collaboration and input enriched the research process.

Additionally, I am grateful to the faculty and staff at Boise State University, whose dedication to academic excellence provided a nurturing environment for intellectual growth.

Thank you all for being an integral part of this significant milestone in my academic life. Your support and encouragement have been invaluable, and I am truly grateful for each one of you.

ABSTRACT

The rapid growth of data generation from electronic devices has created a critical demand for efficient and sustainable data storage solutions. Traditional storage systems face challenges regarding reliability, energy consumption, and scalability, necessitating the exploration of alternative technologies. This dissertation explores the potential of Deoxyribonucleic Acid (DNA) as an alternative storage medium, along with the associated challenges and potential solutions.

This dissertation focuses on Digital Nucleic Acid Memory (dNAM), which utilizes Single Molecule Localization Microscopy (SMLM) to encode and store data within DNA structures called DNA origami. SMLM surpasses the limitations of light's diffraction limit, enabling the imaging of biological samples at a molecular scale. The robustness and data density of the dNAM algorithm rely heavily on the accuracy and performance of SMLM. Within dNAM, emitter localization and error correction are crucial steps, and this dissertation primarily focuses on these aspects.

To improve emitter localization in dNAM, Deep Learning (DL) techniques are employed. This dissertation investigates the impact of multi-emitter situations, where multiple emitters are attached during data acquisition. A neural network based image up-sampling algorithm is developed to progressively increase the resolution of the image. The developed algorithm preserves the emitter centroid position while up-sampling it to a higher-resolution image, effectively isolating attached emitters. By extracting the emitter centroid positions from multiple resolutions, the dissertation analyzes the impact of attached emitters on localization accuracy.

Additionally, the dissertation addresses the development of an advanced error correction algorithm for dNAM. A preliminary algorithm is initially used to successfully store 20 bytes of digital information in DNA. However, to improve performance and accuracy, the algorithm was enhanced by incorporating the intensity information of each data point. The impact of this addition is thoroughly studied. Furthermore, the error correction algorithm is extended to support arbitrary-shaped 3D/2D DNA origami structures, enabling scalability and versatility.

The findings of this research highlight the potential of DNA as a viable storage medium and shed light on the challenges and solutions specific to dNAM. The incorporation of DL techniques for emitter localization demonstrates improved accuracy and efficiency. Moreover, the advanced error correction algorithm enhances the reliability and capacity of dNAM. These outcomes contribute to the overall robustness and efficiency of dNAM as a data storage method.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A Adenine

AFM Atomic Force Microscopy

AI Artificial Intelligence

ANN Artificial Neural Network

BCH Bose-Chaudhuri-Hocquenghem

C Cytosine

CCTV Closed-Circuit Television

CNN Convolutional Neural Network

CRLB Cramér-Rao Lower Bound

DBPN Deep Back-Projection Networks

DL Deep Learning

DNA Deoxyribonucleic Acid

DNA-PAINT DNA Points Accumulation for Imaging in Nanoscale Topography

dNAM Digital Nucleic Acid Memory

DNN	Deep Neural Network
DRCN	Deeply-Recursive Convolutional Network
EB	Exabyte
EC	Error Correction
ECC	Error Correcting Code
EDSR	Enhanced Deep Super Resolution Network
EReLU	Exponential Rectified Linear Unit
FN	False Negative
FP	False Positive
G	Guanine
GAN	Generative Adversarial Networks
GB	Gigabyte
GF	Galois Field
GPU	Graphics Processing Unit
HR	High Resolution
JI	Jaccard Index
KB	Kilobyte

LapSRN	Laplacian Pyramid Super Resolution Network
LDPC	Low Density Parity Check
LR	Low Resolution
LReLU	Leaky Rectified Linear Unit
MAE	Mean Absolute Error
MB	Megabyte
MDSR	Multi-Scale Deep Super Resolution
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MSA	Multiple Sequence Alignment
MSE	Mean Square Error
NAM	Nucleic Acid Memory
nm	Nanometer
NN	Neural Network
nt	Nucleotide
oligo	Oligonucleotide
PCR	Polymerase Chain Reaction

PReLU Parametric Rectified Linear Unit

PRNG Pseudo Random Number Generator

PSF Point Spread Function

PSNR Peak Signal-to-Noise Ratio

RDN Residual Dense Network

ReLU Rectified Linear Unit

RMSE Root Mean Square Error

RS Reed-Solomon

SMLM Single Molecule Localization Microscopy

SOTA State of the Art

SR Super Resolution

SRGAN Super Resolution Generative Adversarial Networks

SSIM Structural Similarity Index

T Thymine

TB Terabyte

TP True Positive

XOR Exclusive-OR

ZB Zettabyte

NOMENCLATURE

α	Learning rate
$\hat{y}$	Label for neural network
$\mathcal{L}_1$	Mean absolute error loss
$\mathcal{L}_2$	Mean squared error loss
$\mathcal{L}_{BCE}$	Binary cross entropy loss
$\mathcal{N}(\mu, \sigma)$	Normal distribution
μ	Mean of normal distribution
σ	Standard deviation of normal distribution
m	Number of training example
w	Weights of deep learning model
y	Prediction from neural network

CHAPTER 1: INTRODUCTION

1.1 DNA As A Storage Medium

The proliferation of electronic devices and the widespread adoption of technologies like the Internet of Things (IoT) and Artificial Intelligence (AI) have led to an exponential surge in data generation [5]. Based on this escalating trend, it is projected that the global volume of digital data will reach 175 Zettabytes (ZBs) by 2025 [15]. This presents a significant challenge as traditional storage systems struggle to keep pace with such vast quantities of information. Adding to the urgency of the situation is the expected depletion of silicon supply by 2040 [14], a crucial component in current storage technologies.

Certain types of data, such as financial records, historical archives, and Closed-Circuit Television (CCTV) footage, necessitate long-term storage despite infrequent access. However, existing data storage systems are unreliable for extended durations, requiring continuous replication of archival data to ensure its integrity. Consequently, the construction and maintenance of a data center capable of storing 1 Exabyte (EB) of data demand hundreds of megawatts of power and a cost exceeding 1 billion USD over a 10-year period [16]. Extrapolating these expenses to accommodate 175 ZBs would require millions of megawatts solely for information storage, which is clearly unsustainable. These circumstances have prompted an active pursuit for an information storage medium capable of preserving data without replication for hundreds of years while consuming minimal energy. Deoxyribonucleic Acid (DNA) can be a promising alternative for this purpose for the following reasons:

1) Under proper conditions, DNA is known to retain data for hundreds of thousands of years [14]. 2) An individual molecule of single-stranded DNA is theoretically capable of storing information at a density of 455 EBs per gram [1], implying that 0.5 kg of DNA could be sufficient to store all global digital data expected in 2025. 3) DNA's operation energy is many orders of magnitude less than current electronic memories [14].

Table 1.1 provides a comprehensive comparison between established conventional memory technologies and the emerging cellular DNA-based storage medium, highlighting the potential of DNA for future data storage applications. The first column, labeled "Memory ($type$)" identifies the different memory technologies being compared. The second column, "Retention ($years$)" represents the duration for which information can be stored in each medium without corruption. Longer retention times indicate greater stability and longevity of stored data. The third column, "ON Power ($W/Gigabyte(GB)$)" quantifies the power consumption required to store information per GB for each memory technology. Lower values signify more energy-efficient storage mechanisms. The fourth and fifth columns, "Areal Density (bit/cm^2)" and "Volumetric Density (bit/cm^3)" respectively, highlight the data storage capacity in terms of the number of bits that can be stored in a given area and volume. Higher areal and volumetric densities reflect the ability to store more information in a smaller physical space, indicating the potential for increased storage capacities. The sixth column, "Latency (s/bit)," measures the time required to store each bit of information. Lower latency values indicate faster data storage and retrieval processes. Lastly, the seventh column, "Error Rate ($errors/bit$)" denotes the expected number of errors per bit of stored information. Lower error rates indicate greater reliability and accuracy in data storage. Currently, DNA-based storage exhibits higher error rates compared to conventional methods. However, it is anticipated that advancements in synthesis and sequencing technologies will reduce DNA's error rates over time. The table's findings highlight the advantages of DNA-based storage, such as its exceptional retention capabilities, ultra-low power requirements, extraordinarily high information density, and relatively low latency.

Table 1.1: Comparison between established memory technologies and cellular DNA. This table was adapted from the work of Zhirnov et al. [14].

Memory ($type$)	Retention ($years$)	ON Power (W/GB)	Areal Density (bit/cm^2)	Volumetric Density (bit/cm^3)	Latency ($\mu s/bit$)	Error Rate ($errors/bit$)
Flash Memory	10	0.01 – 0.04	10^{10}	10^{16}	100	10^{-15} [1]
Hard Drive	> 10	0.04	10^{11}	10^{13}	$3 * 10^3$– $5 * 10^3$	10^{-15} [2]
Magnetic Tape	30 [2]	0.004 [3]	10^9– 10^{10} [5]	N/A	60 - 200 [5]	10^{-18} – 10^{-21} [2]
Cellular DNA	> 100	$< 10^{-10}$	10^{22}	10^{22} [17]	< 100	10^{-9} – 10^{-8} [6]

1.1.1 Sequence-based DNA Data Storage

Sequence-based DNA data storage has been a primary focus for the last decade [1, 2, 6, 7]. A Nucleotide (nt) is the basic building block of DNA, which can be one of four types of nitrogen bases: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). The digital data is first divided into equal-length segments. Later encoding part of the error correction algorithm adds some additional data to the segments. These additional data are used to verify integrity or fix errors from each segment. Then each segment is converted into DNA sequences using a mapping scheme. The most straightforward mapping scheme can be 00 representing A, 01 representing T, 10 representing G, 11 representing C; here, 0 or 1 represents each data bit. Later, using DNA synthesis technology, the sequences are converted into physical strands of DNA, which can be optimally stored in the appropriate environment and temperature. For data retrieval, DNA sequencing technology is utilized to read the nts from physical DNA strands. Later the same mapping scheme that was used during encoding is employed to convert nts into binary data. It is important to note that errors are likely to occur throughout the process, including synthesis, storage, and sequencing. Therefore, the decoding phase of the error correction algorithm examines the binary data and identifies/corrects any errors encountered. Refer to Figure 1.1 for a more comprehensive understanding of the entire process. Detailed information on recent methodologies published in this field can be found in Section 2.2.

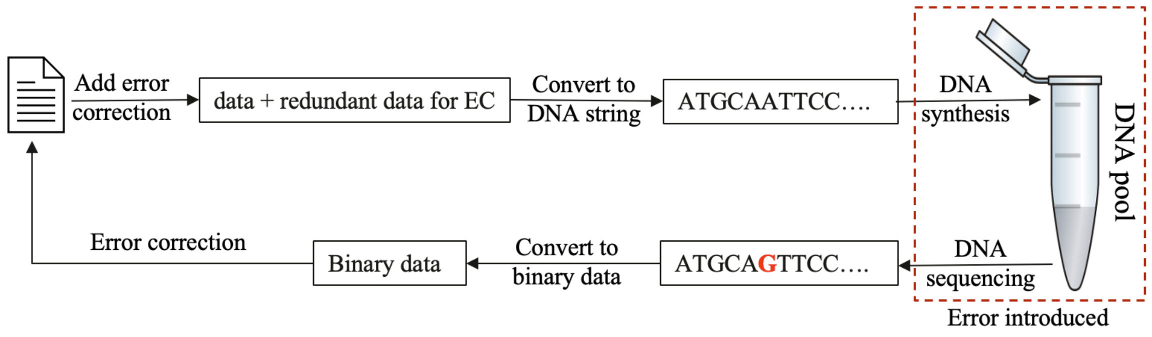


Figure 1.1: The figure illustrates the sequence-based DNA storage process overview. The initial step involves adding redundant information to the original data for error correction purposes. Subsequently, the digital information is converted into DNA nucleotides using a mapping scheme. The synthesized DNA is then stored in a DNA pool. During the reading process, the DNA pool is read using DNA sequencing technologies. The reverse mapping scheme is applied to convert the nucleotides back into binary data. Following this, the error correction decoding algorithm utilizes the redundant information added during the encoding procedure to identify and correct any errors, if present. Finally, the original file is recovered after the error correction process.

1.1.2 Space-based DNA Data Storage

Recently, researchers have explored alternate methods of storing data in DNA that avoids sequencing and synthesizing [18, 19, 20, 21]. These methodologies decouple the DNA storage technology from sequencing/synthesizing advancement. In 2021, we developed a prototype system called dNAM that enabled us to store digital information in DNA using the imaging technique of Single Molecule Localization Microscopy (SMLM), thereby avoiding the need for synthesis and sequencing steps [21]. dNAM is a multi-step process, including emitter localization, drift correction, data extraction, and error correction. Among them, improving the emitter localization, data extraction, and error correction is the primary focus of this dissertation. DNA origami is the building block of dNAM. The following section describes the working principle of

DNA origami.

DNA Origami

Discovered by Paul W. K. Rothemund in 2006 [22], DNA origami is a revolutionary technique that harnesses the unique properties of DNA molecules to create intricate and precisely folded nanostructures. Inspired by the ancient art of paper folding, origami, this method allows scientists to engineer and manipulate DNA strands into complex three-dimensional shapes at the nanoscale level. It provides a programmable building material as a foundation to make nanosctuctures using DNA. Its precision, versatility, and scalability make it a promising tool for advancing various scientific and technological fields, including medicine, electronics, materials science, bioengineering, and data storage [21, 23, 24, 25, 26, 27].

At the heart of DNA origami is the fundamental building block, the DNA double helix. This helical structure, composed of two complementary DNA strands intertwined in a spiral, forms the backbone of DNA origami designs. By carefully designing shorter DNA strands, known as staple strands, scientists can introduce specific sequences that bind to the DNA scaffold and direct its folding into desired shapes.

The process of creating DNA origami begins with a long single-stranded DNA scaffold, typically derived from a viral genome or synthetic sources. This scaffold serves as the foundation upon which the structure will be built. Researchers then strategically design hundreds of staple strands that hybridize with the scaffold at precise locations, facilitating the folding process.

Through a controlled annealing process, the scaffold and staple strands are mixed together under specific temperature and salt conditions, allowing the DNA to selfassemble into the desired shape. The binding interactions between the staple strands and scaffold dictate the final structure, enabling the construction of a wide range of geometric shapes, such as triangles, squares, stars, and even more complex structures like nano-boxes and nano-robots.

DNA origami offers numerous advantages over conventional nano-fabrication techniques. Its ability to produce custom-designed nano-structures with sub-nanometer precision allows for the precise positioning of molecules, nano-particles, and other functional elements on the nano-scale. Additionally, DNA origami exhibits remarkable stability and compatibility with biological systems and can be easily modified by incorporating various chemical or biological moieties, further expanding its potential applications.

Digital Nucleic Acid Memory

In the dNAM approach, digital information is encoded into specific locations within DNA origami structures. During the origami formation, the staple strands are arranged at addressable locations (see Figure 1.2) that define an indexed matrix of digital information. This site-specific localization of digital information is enabled by designing staple strands with nucleotides that extend from the origami. Extended staple strands have two domains: the first domain forms a sequence-specific double helix with the scaffold and determines the address of the data within the origami; the second domain extends above the origami and, if present, provides a docking site for fluorescently labeled single-stranded DNA imager strands. Binary states are defined by the presence (1) or absence (0) of the data domain, which is read with a super-resolution microscopy technique called DNA Points Accumulation for Imaging in Nanoscale Topography (DNA-PAINT) [11]. Unique patterns of binary data are encoded by selecting which staple strands have, or do not have, data domains. As an integrated memory platform, data is entered into dNAM when the staple strands encoding 1 or 0 are selected for each addressable site. The staple strands are then stored directly or self-assembled into DNA origami and stored. Editing data is achieved by replacing specific strands or the entire content of a stored structure. To read the data, the origami is optically imaged below the diffraction limit of light using DNA-PAINT.

Figure 1.2 depicts a single origami structure. We introduced a custom error correction algorithm that enabled us to store and recover 20 bytes of data without any manual intervention. We used a fountain code at the top level of our error correction algorithm. While our error correction algorithm fixed errors at the individual origami , and the fountain code is responsible for recovering data if one or multiple origami are entirely missing. More details on this can be found in Section 4.

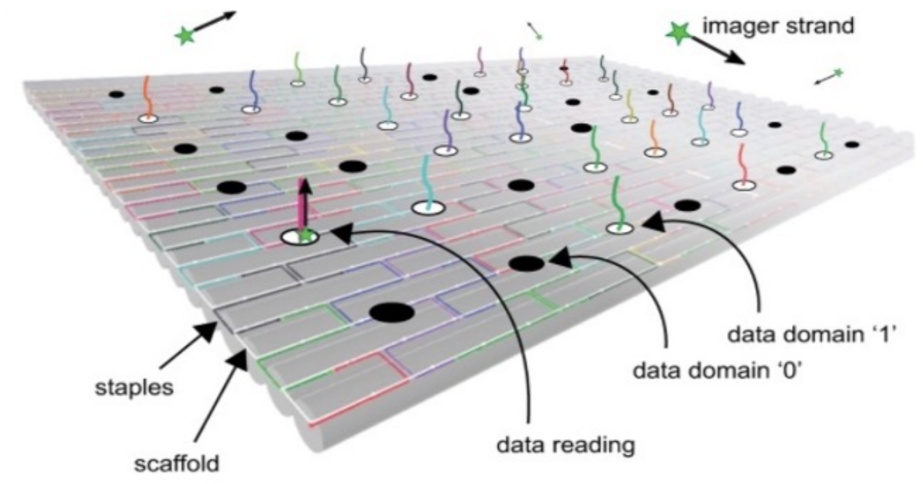


Figure 1.2: Overview of a single origami nanostructure. The black circle and white circle represent binary data '0' and '1', respectively. There is an extending staple strand on data domain '1', providing a docking site for the imager strands.

Our original origami design was in the form of a 6x8 matrix, capable of storing 48 bits of information in a single structure. We used this structure to store 20 bytes (160 bits) of data encoded into 15 different origami (720 bits). 78% of the total capacity is dedicated to error correction, index bits, and rotation bits. While these bits do not contain any information, they ensure error-free recovery of stored information. As the error rate of dNAM is high, it is required to dedicate a significant portion of the total capacity for error correction to ensure error-free data recovery. More precise localization of emitters and better error correction algorithms can significantly reduce the errors from dNAM, which would reduce the number of bits that need to be dedicated to error correction and make dNAM more robust and denser. This will also reduce the computational time for data recovery.

1.2 Machine Learning

Machine Learning (ML) and Deep Learning (DL) enable machines to learn from data and make intelligent decisions without explicit programming. ML involves the development of algorithms that learn patterns and relationships iteratively from a given dataset. It encompasses a wide range of techniques, such as supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, algorithms are trained on labeled data, where input features are associated with corresponding output labels. Through this process, the algorithm learns to make predictions or classifications on new, unseen data. On the other hand, unsupervised learning deals with unlabeled data, where the algorithm discovers inherent patterns and structures within the data. It can be used for tasks such as clustering, dimensionality reduction, and anomaly detection. Reinforcement learning involves an agent learning to interact with an environment and make decisions based on rewards and punishments, aiming to maximize its cumulative reward over time.

DL is a subset of ML that focuses on the development and training of neural networks, which are composed of interconnected layers of artificial neurons. Deep neural networks are capable of learning hierarchical representations of data, extracting intricate and abstract features. The depth of these networks allows them to process increasingly complex patterns, making them particularly effective in domains such as image and speech recognition, natural language processing, and computer vision. DL has been further empowered by the availability of large-scale datasets and advancements in computational resources, enabling the training of deep neural networks with millions of parameters.

The applications of ML and DL are vast and diverse. Over the last couple of years, DL achieved State of the Art (SOTA) performance in various fields of computer vision [28, 29, 30, 31], speech recognition [32], machine translations [33], genomics [34], biomedical imagery [35], and many other domains [36]. In healthcare, these technologies have been used for disease diagnosis and prediction, personalized treatment recommendations, and drug discovery. In finance, ML and DL are employed for fraud detection, algorithmic trading, and credit scoring. Robotics benefits from ML and DL for tasks such as object recognition, motion planning, and autonomous navigation. Image and video analysis, including facial recognition and object detection, have been greatly enhanced by these techniques. Natural language processing allows machines to understand and generate human language, enabling applications such as voice assistants, language translation, and sentiment analysis. Because of recent progress in hardware, software, algorithm parallelization, DL training, and inference time have been reduced to an order of magnitude.

The continuous advancements in ML and DL, coupled with the availability of big data and increased computing power, hold immense potential for further breakthroughs. Researchers and practitioners are constantly exploring new techniques, architectures, and applications in various domains. Emitter localization in the field of SMLM is one of the recent application of DL [37, 38, 39, 40] (more details in Section 2.4.4). As ML and DL continue to evolve, they are expected to drive innovation, transform industries, and shape the future of artificial intelligence.

1.3 Dissertation Contribution and Organization

1.3.1 Dissertation Contribution

The key contributions presented in this dissertation are summarized as follows:

- A comprehensive overview of existing sequence-based storage methodologies (see Section 2.2).
- An emitter localization algorithm was developed to enhance the resolution of an individual frame through DL techniques. This process facilitated the isolation of overlapped emitters, and subsequently, a separate Neural Network (NN) or Maximum Likelihood Estimation (MLE) algorithms were employed to identify and localize the centroid positions of the emitters in the higher resolution space. Additionally, a comprehensive analysis of the effects of isolating overlapped emitters was conducted. (see Chapter 3)
- A bi-level parity, rotation invariant error correction algorithm that enabled us to store and retrieve information into dNAM without any manual intervention (see Section 4.1).

- Improvement of the error correction algorithm by adding photon intensity of each data point that was captured using SMLM. We have also studied the impact of this integration in multiple metrics such as false positive, false negative, and computational time. (see Section 4.2)
- Extension of the error correction algorithm so that it can support arbitrary shaped origami even in 3-dimension. We verified this error correction algorithm by in-silico simulation. (see Section 4.3)
- Our solution on a sequence-based storage method (see Appendix A).

Several findings and components of this dissertation have been previously published or are currently undergoing the publication process in the following scholarly paper:

- An alternative approach to nucleic acid memory [21]
- In-vitro validated methods for encoding digital data in DNA [41]
- A method for storing information in DNA with improved dropout tolerance [In review in Plos one] [42]
- Multi-emitter localization using deep learning [In preparation]

1.3.2 Dissertation Organization

This dissertation presents a comprehensive investigation conducted in five chapters aimed at enhancing the robustness and density of dNAM. Chapter 1 provides the motivation and methodology underlying the utilization of DNA as a storage medium. Additionally, it offers a concise overview of the principles and mechanisms of DNA origami, which serves as a key technique in this research. Furthermore, this chapter provides a succinct summary and application of ML.

Chapter 2 offers essential background information pertaining to error correction, sequence-based DNA data storage, SMLM, and DL. Firstly, it explores previous research on error prevention throughout the processes of DNA synthesis, sequencing, and storage, providing a comprehensive overview of the existing methodologies. Subsequently, it delves into well-established error correction algorithms that have been widely adopted across various industries and subsequently applied to DNA-based storage systems. It also discusses how researchers have integrated these error correction methods and translated binary data into DNA oligomers. Some of this information has been previously published in our recent publication titled "In-vitro validated methods for encoding digital data in deoxyribonucleic acid (DNA)" [41]. Additionally, it provides an in-depth discussion on the principles and commonly used approaches for emitter localization in SMLM. Finally, a concise explanation of DL is provided, along with a literature review on the utilization of DL for natural image super resolution and emitter localization in the context of SMLM.

Chapter 3 presents a thorough examination of our DL based emitter localization algorithm, encompassing a comprehensive overview and detailed methodology. Firstly, the chapter outlines the process of data generation employed for training the DL model. Subsequently, it elucidates the key performance metrics that were prioritized for optimization using the model. Furthermore, the chapter provides an in-depth discussion of the actual algorithms employed for emitter localization. Lastly, the chapter presents and discusses the results obtained from the trained models, offering insights into their performance and efficacy.

Chapter 4 delves into the encoding and decoding processes of our error correction algorithms. In the results section, it provides a detailed account of how the simulations were performed for the error correction algorithm, ensuring alignment with the wet lab experiments. Moreover, it describes the methodology employed for incorporating intensity information and its subsequent impact on performance improvement. Additionally, the chapter outlines the extension of the algorithms for 3dNAM and arbitrary shaped origami, elucidating the methodology and presenting simulation results.

Chapter 5 serves as the concluding chapter of the dissertation, encapsulating the key findings, contributions, and implications of the research. It offers a comprehensive summary of the study's outcomes and addresses the research objectives outlined in the earlier chapters. Furthermore, the chapter identifies potential areas for future research, highlighting avenues that can be explored to further advance the field and build upon the present study.

CHAPTER 2:
BACKGROUND INFORMATION

2.1 Error Correction On Nucleic Acid Memory

DNA is a biochemically active material that undergoes molecular reactions. That makes it more susceptible to errors compared to conventional storage media. Successfully recovering information from DNA is complicated by numerous errors which can arise prior to sequence decoding. Mechanisms leading to such errors include imperfect synthesis, degradation, imperfect amplification via Polymerase Chain Reaction (PCR), or imperfect sequencing. At the start of the decoding algorithm, these errors typically manifest as sequence dropout (an entirely missing sequence), point mutations (substitution of a single nucleotide), indels (insertion or deletion of a single nucleotide), or truncations (removal of several nucleotides from one end of a sequence). Consequently, error rates depend on factors such as the specific synthesis, storage, and sequencing methods employed. For instance, Bornholt et al. [5] reported an error rate of approximately 1% using Illumina sequencing, while Organick et al. [6] reported error rates of up to 10% using nanopore sequencing. Overall, DNA as a storage medium exhibits much higher error rates than conventional storage medium. For example, Church et al. [1] reported error rates of 1 bit per 0.7 Megabytes (MBs), which is much higher than error rates of 1 bit per 10-1000 Terabytes (TBs) exhibited by electronic memories. There are two general mechanisms for handling these errors: 1) by generating DNA sequences that are less likely to exhibit errors (*i.e.* error mitigation, Section 2.1.1) and 2) by incorporating Error Correction (EC) codes that are robust to the expected errors (described in Section 2.1.2). Most of the researchers simultaneously utilize both mitigation and correction.

2.1.1 Error Mitigation

Oligonucleotides (oligos) displaying specific structural patterns are known to be associated with higher synthesis and sequencing errors. For example, Schwartz et al. [43] noted that oligos with more than 60% GC content exhibit higher dropout rates which lead to PCR errors. Tabatabaei Yazdi et al. [44] reported that oligos that have approximately 50% GC content exhibit greater stability, and maintaining 50% GC content ratio reduces synthesis/sequencing errors. Several studies have highlighted the errors related to homopolymer runs, such as repetitive sequences like AAAAA or TTTTTT. In particular, Ross et al. [45] noted that homopolymer runs exceeding four nts are associated with a higher occurrence of indel errors. Additionally, Ananda et al. [46] identified PCR slippage errors that rapidly escalate when homopolymers exceed four nts. Both Poon and Macgregor [47] and Xu et al. [48] noted that homopolymers of six or more nts exhibit high enough thermal stability to make sequencing difficult [47, 48]. Lastly, Church et al. [1] reported that their errors primarily manifested within homopolymer runs near the ends of sequences.

Numerous methods have been documented for mitigating errors in DNA sequencing. Church et al. [1] employed a mapping scheme to circumvent issues associated with extreme GC content and repeated sequences. Additionally, they assert that their method addresses certain secondary structures, although the precise mechanism behind this accomplishment is not explicitly elucidated.

The method reported by Goldman et al. [2] pads indexes with leading zeros to ensure they are a constant length. This has the potential to generate problematic long repeating sequences. This is exacerbated as the amount of data increases since larger files require longer indexing regions (*i.e.* the index size grows as a log factor of the data size). To mitigate this concern, they employed a rotating mapping scheme to resolve the issue. Later, Bornholt et al. [5] and Organick et al. [6] also adopted this mapping scheme in their respective studies.

In the research conducted by Organick et al. [6], they introduced an algorithm for generating random 20-mer primers. This algorithm aimed to avoid higher GC content ratios, circumvent certain secondary structure issues, and maintain a minimum Hamming distance of six between sequences. The Hamming distance refers to the measure of the difference between two sequences of equal length, counting the number of positions at which the corresponding elements are different. To achieve these goals, the algorithm employed a technique of randomizing the input data through XORing with a pseudo-random sequence. This approach effectively mitigated problems associated with secondary structures and longer homopolymer runs.

2.1.2 Error Correction

Despite taking precautions and avoiding error-prone sequences, errors still persist in DNA-based storage mediums. To address these errors and ensure the viability of DNA as a storage medium, EC algorithms are incorporated into the encoding and decoding processes. These algorithms add redundant data to each oligo, ensuring the integrity of the stored information. By employing more robust EC algorithms, the resilience of DNA storage can be enhanced. Fortunately, there are already several established and practical EC algorithms available, widely used in noisy channel communication and conventional data storage systems. Researchers have utilized some of these well-known Error Correcting Codes (ECCs) in DNA-based storage systems. Table 2.1 provides a comparison of popular EC algorithms commonly applied in the field of Nucleic Acid Memory (NAM).

Table 2.1: Comparison of different error correction algorithms. Here, "soft bit decoding" indicates that the algorithm provides a probability of a bit being 0 or 1, while "hard bit decoding" provides a definitive value of 0 or 1. The term "burst errors" refers to consecutive errors that occur within a short span of data, whereas "scatter errors" are errors that occur randomly throughout the data without any specific pattern or clustering.

	Repetition code	Hamming code	Reed-Solomon	Bose-Chaudhuri-Hocquenghem	Low Density Parity Check
Error correction	Multiple bits	Single bit	Multiple bits	Multiple bits	Multiple bits
Error detection	Multiple bits	Two bits	Multiple bits	Multiple bits	Multiple bits
Error types	Scatter + burst	Scatter	Burst	Scatter	Scatter
Soft bit decoding	Yes	No	No	No	Yes

Repetition Code

Repetition codes are the most basic EC scheme. Sometimes referred to as Multiple Sequence Alignment (MSA), a single data block repeats a predetermined number of times to mitigate the loss of data. If a minority of the data blocks differ from the majority, then that particular data block is considered as corrupted, and majority voting is used to select the correct data block. Although this scheme is simple, it suffers from inefficiency as only a fraction of the data blocks contain unique data. Furthermore, if the same error affects all the data blocks, it becomes impossible to detect that error.

Hamming Code

A Hamming code is a member of the linear block code family, which was developed in 1950 by Richard W. Hamming [49]. Hamming codes can detect at most two bits of error in the data block and can fix one bit of error in the data block. However, due to these limitations, Hamming codes have seldom been employed in DNA-based storage systems. Takahashi et al. [50] used a (31, 26) Hamming code for DNA data storage. However, their data size was relatively small (5 bytes). Errors like deletion and read truncation made data retrieval difficult. Out of 25,592 reads, 16 were perfect reads, and eight were corrupted but correctable, and out of these eight reads, only three were corrected perfectly.

Reed-Solomon Code

Reed-Solomon (RS) codes are a special class of Bose-Chaudhuri-Hocquenghem (BCH) codes, which were first introduced in 1960 [51, 52]. These are widely used in EC algorithms and are capable of correcting both burst errors and erasures. RS codes have found extensive application in data storage and digital communication systems, including CD, DVD, QR codes, and mobile/satellite communications [53]. RS codes are more effective for insertion/deletion (indel) errors rather than point mutation errors.

In RS coding, if the message size is n bytes, RS code adds redundant data of size k bytes at the end of the message. RS code can detect up to k bytes of errors at arbitrary locations and can correct up to $\lfloor \frac{k}{2} \rfloor$ bytes of errors at arbitrary locations. One of the advantages of RS code over the other EC scheme is the length of redundant code block k can be adjusted depending on the usage or by analyzing prior error patterns. Several researchers have used the RS code to detect/correct errors in DNA based storage systems [6, 3, 4, 7, 54, 55]. These studies highlight the applicability and effectiveness of RS codes in enhancing the reliability of DNA storage systems.

Low Density Parity Check

Low Density Parity Check (LDPC) [56] codes are a class of linear ECC which was developed by Robert G. Gallagher in 1962. In recent years, LDPC codes have gained popularity due to their ability to decode both hard bits and soft bits. Soft bit decoding provides probabilities for a bit being 0 or 1, while hard bit decoding determines whether a bit is 0 or 1. The coding scheme reported by Yim et al. [57] utilized an LDPC ECC to store a picture of size 438 bytes. Fei and Wang [58] proposed a different version of LDPC codes specifically focused on correcting errors in DNA storage systems. Chandak et al. [59] used a combination of three error correction algorithms. LDPC was used for outer level EC. In order to handle indel errors, they inserted a synchronization marker (sequence “AGT” was used for this purpose) in the middle of the data block sequence. BCH was used for EC in indexes. MSA was used to reduce errors in any individual sequence.

Fountain Codes

A fountain code, introduced by MacKay [60] in 2005, belongs to the class of erasure codes, also known as rateless erasure codes. These codes are designed to enhance robustness to dropouts or erasures in data transmission. Fountain codes generate a large number of smaller data chunks called droplets by XORing segments of the original data. These droplets are typically used for transmitting data over noisy channels. The concept behind fountain codes is that the original data can be reassembled from any subset of the received droplets as long as a sufficient number of them are correctly received. The name “fountain” reflects the analogy of catching water droplets from a fountain to fill a glass with water. Similarly, encoded data can be recovered by collecting enough droplets, regardless of which droplets are received or the order in which they were received.

Although a fountain code is not inherently an EC algorithm, it enables the recovery of the entire file even if a portion of the transmitted data is lost. Fountain codes are often used in combination with other EC algorithms. In such cases, the EC algorithm is responsible for verifying the integrity of the droplets. If the EC algorithm cannot verify a droplet, the fountain code discards that droplet and does not use it in the decoding process.

In the context of DNA-based storage, researchers such as Erlich and Zielinski [7] and Anavy et al. [54] have utilized fountain codes along with RS codes for NAM applications. This combination of fountain codes and RS codes demonstrates the effectiveness of using fountain codes to enhance the robustness and reliability of DNA storage systems.

2.2 Overview of Sequence-based DNA Storage Methodologies

In his 1959 lecture titled "Plenty of Room at the Bottom", Richard Feynman recognized the potential of using DNA to store information [61]. To the best of our knowledge, information was first stored using synthetic DNA around 1988 [62]. In the following decades, advances in DNA sequencing and synthesis technologies –primarily driven by the Human Genome Project– led to an increased interest in storing information in DNA. Numerous methods for writing and reading information from DNA have been reported since. In the following paragraphs, several recently reported representative methods with explicit in-vitro validation are introduced. The methods are presented in chronological order, showing how the field has progressed in terms of volume and complexity of data and sophistication of storage methods. Table 2.2 summarizes the progression of DNA as a storage medium.

The earliest of the methods was reported by Church et al. [1]. This method was used to store 659 Kilobytes (KBs) of data containing a book. The encoding process for this method can be described as the following procedure shown in Figure 2.1. First, the data was split into sequentially addressed data blocks, each containing 96 bits. Each block was then prepended with a 19 bits indexing address. This 115-bit sequence was coverted to a 115-base sequence by mapping 0 to A or C and 1 to G or T. The 115-base sequence was then flanked with a pair of 22 base primer sequences for amplification and sequencing purposes. This resulted in a set of 159-base sequences which collectively encoded the data.

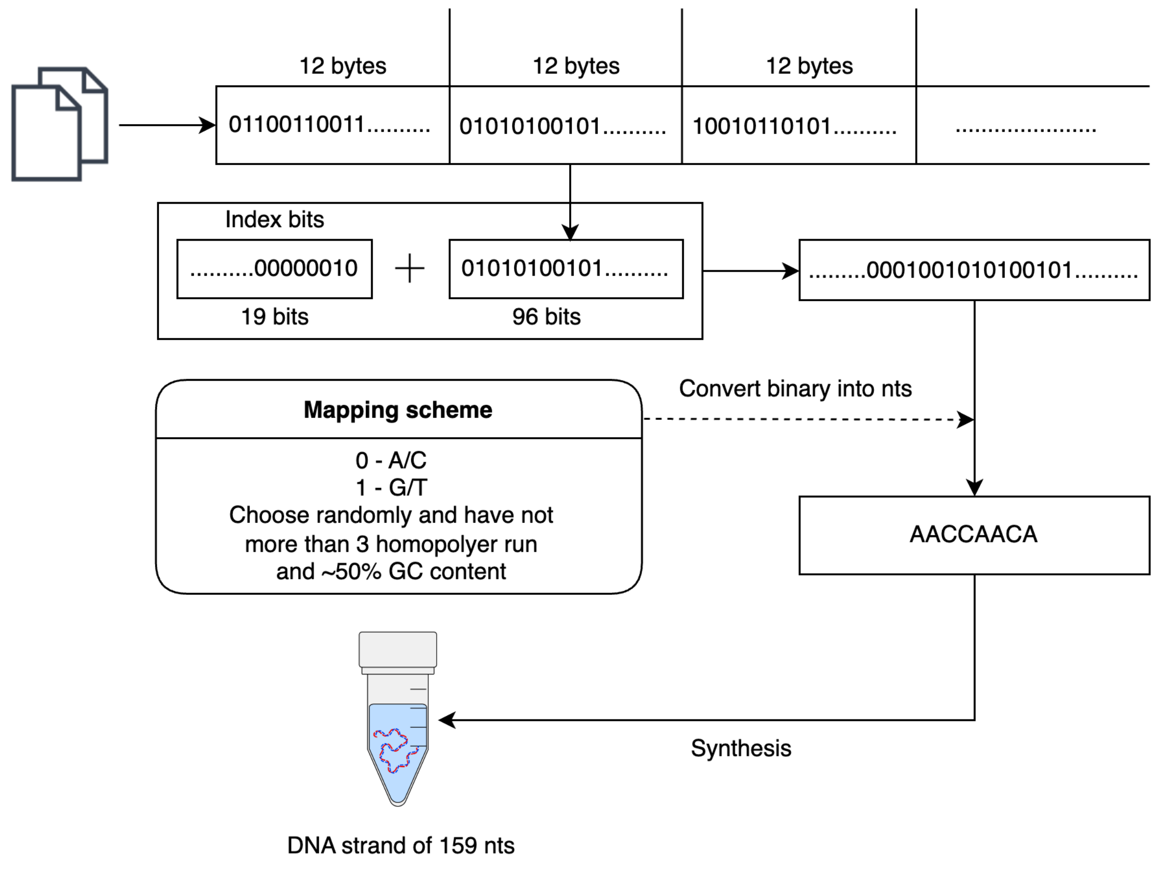


Figure 2.1: Diagram of the encoding algorithm reported by Church et al. [1]. Data is split into data blocks and attached to index bits. The resulting binary sequence is then directly mapped to a base sequence.

The next method was reported by Goldman et al. [2]. This method was used to store five files which totaled 757,051 bytes and included two text files, a pdf, a photograph, and an mp3. The encoding process for this method can be described by the following procedure shown in Figure 2.2. The data was provided as a list of files. An index was assigned to each file and each file was represented as a sequence of bits (base-2 values). The bit-sequence for each file was converted to a sequence of trits (base-3 values) using a Huffman-code. This trit sequence was then converted to a base sequence according to a rotating mapping code (“Mapping Scheme” in Figure 2.2). This yielded a single base sequence encoding the entire file, which was then split into indexed segments containing 100 bases and overlapping by 75 bases. The base sequence of every other segment was replaced with its reverse-complement (*i.e.* A/T swapped, G/C swapped, and then reversed). The following additional bases were then added to each 100-base sequence: 1) Two bases to indicate the file index, 2) twelve bases containing the index of the segment, 3) two bases indicating if the sequence has been reverse-complemented, 4) one parity base for detecting errors. This resulted in a set of 153,335 117-base sequences collectively encoding the data.

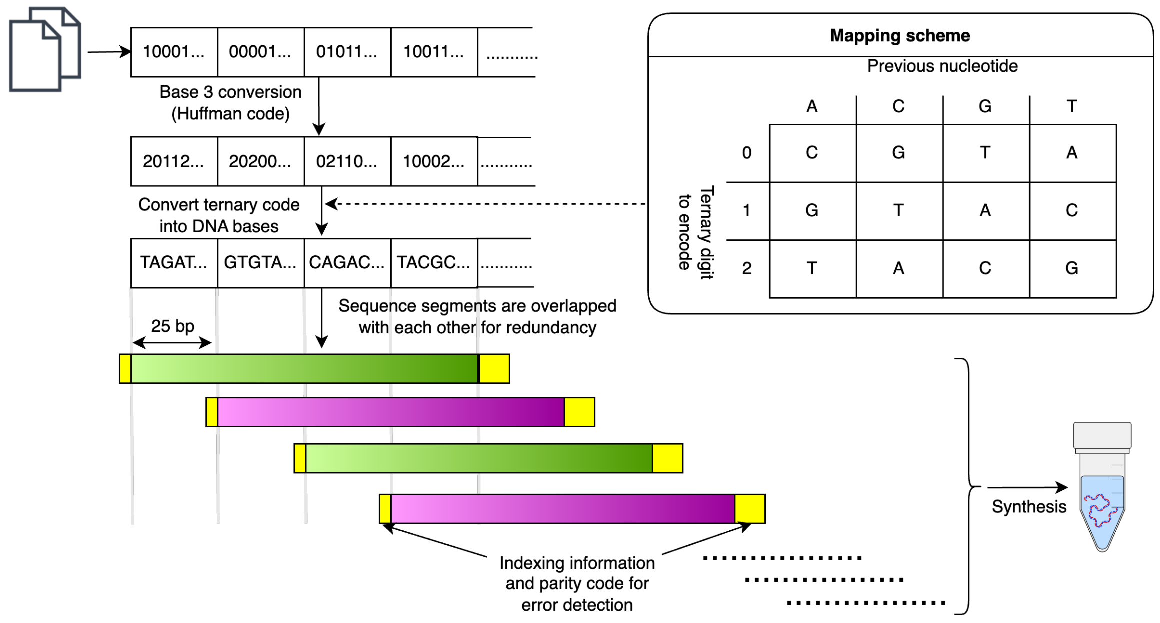


Figure 2.2: Diagram of the encoding algorithm reported by Goldman et al. [2]. A rotating mapping algorithm is used to avoid homopolymers. A parity code ensures the integrity of each segment. Every alternating segment is reverse-complemented for data security (shown in violet color).

The third method was reported by Grass et al. [3]. This method was used to store two text files totaling 83 kilobytes. The encoding process for this method can be described by the following procedure shown in Figure 2.3. First, the digital data was converted into a number within the Galois Field (GF) of size 47 (GF(47)), which is a mathematical structure comprising a finite set of elements and defined operations of addition, subtraction, multiplication, and division that adhere to specific mathematical properties. These numbers were then put into a block of 594×30 values (this can be seen in the encoding block of Figure 2.3). The first RS parity information was added to each row in the form of 119 values from the GF(47). This section was referred to as the outer block (Redundancy A in the diagram). Next, an index section containing three values was added. Next, a second level of RS parity was added. This section containing six values was referred to as the inner block (Redundancy B in the diagram). Each column consisted of 39 base-47 values which were converted to a base sequence according to a word-based mapping code depicted by the wheel in Figure 2.3. Two constant base-sequences used as primers were then attached to each 117-base sequence, yielding a set of 4,991 158-base sequences which collectively encode the data.

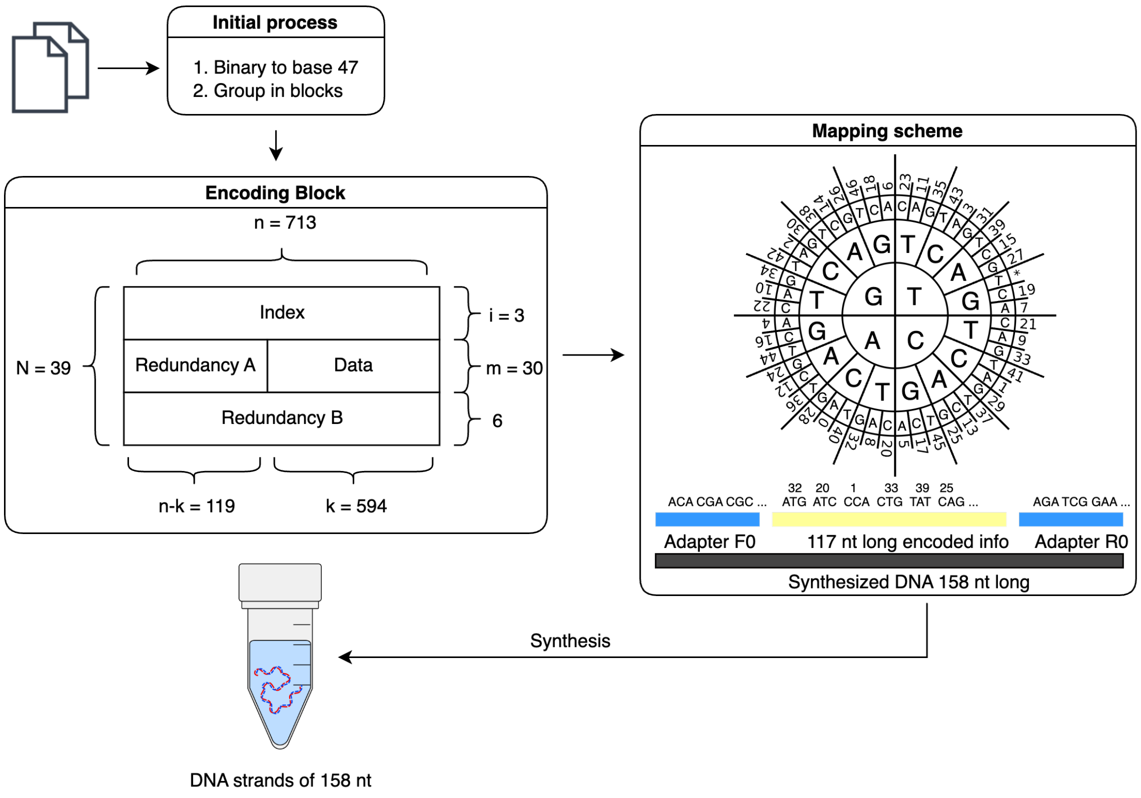


Figure 2.3: Diagram of the encoding algorithm reported by Grass et al. [3]. Binary data is converted to base 47 and packaged into 713 × 39 character matrices. Each column is then encoded as a DNA sequence with each character encoded as a codon according to the mapping scheme.

The fourth method was reported by Blawat et al. [4]. This method was used to store a 22 megabyte video file. First, the digital file was split into segments of non-overlapping bit sequences. A 39 bits sequence used for segment addresses was encoded using a (63,39) BCH code and prepended to the segment's bit sequence. A 16 bit cyclic redundancy check code was then calculated and appended to the bit sequence. The bit sequence was then represented as a byte-sequence and converted to a base-sequence using the following mapping algorithm. The pre-determined mapping code associates each byte value with at least two and at most three 5-base sequences. The first six bits of the byte determine the bases in positions 1, 2, and 4 depending on the table labeled “mapping scheme a” in Figure 2.4. The last two bits of each byte encode bases three and five using the table labeled “mapping scheme b” in Figure 2.4. Valid 5-base sequences satisfied two rules: 1) The first three bases cannot be the same, and 2) the last two bases cannot be the same. This mapping algorithm yielded 190-base sequences, to which a pair of 20 base primer sequences were attached. This resulted in 225,000 230-base sequences, which collectively stored the data. The authors mention the presence of a RS code to protect blocks of consecutive sequences. However, we were unable to determine the exact nature and location of this code from the text of the manuscript.

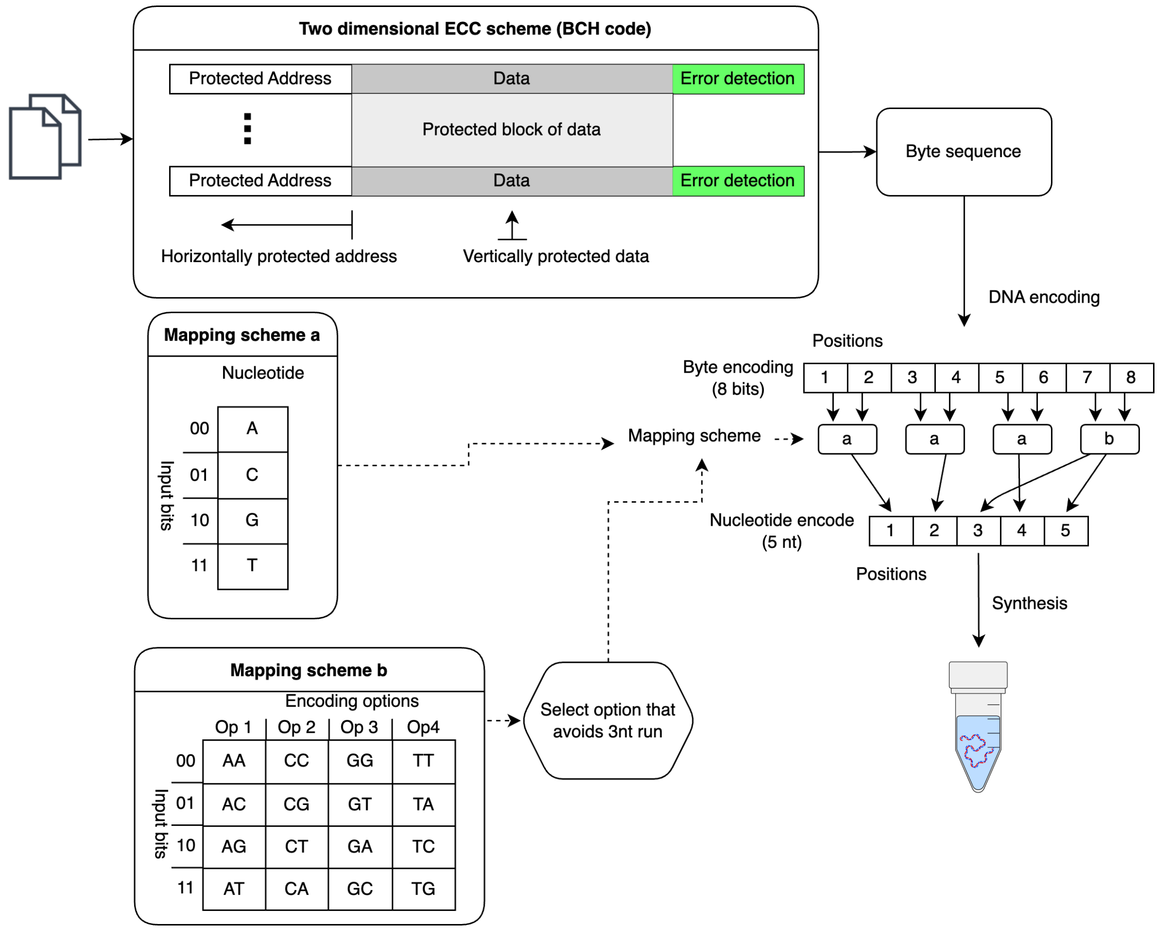
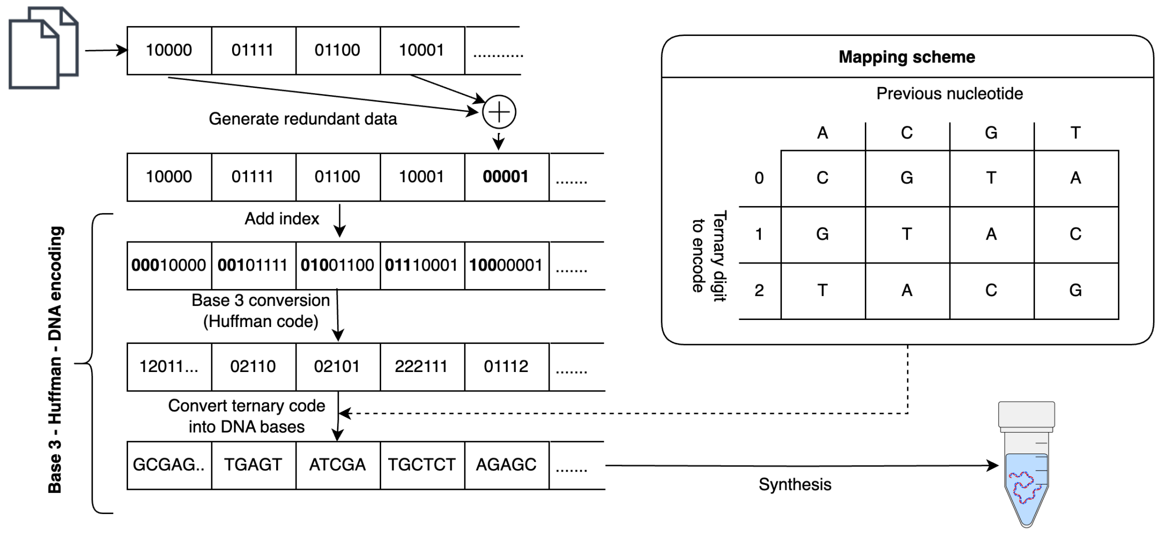


Figure 2.4: Diagram of the encoding algorithm reported by Blawat et al. [4]. The binary sequence is split into bytes, which are encoded to 5-base sequences using a combination of the two mapping schemes (a,b).

The fifth method was reported by Bornholt et al. [5]. This method was used to store four image files totaling 151 KBs of data. The encoding process of this method can be described by the following procedure shown in Figure 2.5. First, a pair of base-sequences for use as primers and a base-sequence for use as a file address were chosen from an existing library. The bit sequence of the file was converted to a trit sequence using a Huffman code. The trit sequence was then converted to a base-sequence according to a rotating mapping code. This base-sequence was then split into non-overlapping segments. The following were then added to each segment: 1) the two 9-base primer sequences, 2) the file address sequence, 3) two bases to indicate if the sequence has been reverse complimented. These sequences were then added to the list of sequences to synthesize. For redundancy, additional sequences were calculated by using an Exclusive-OR (XOR) operation to combine two segments into a single sequence. These additional sequences were included such that all segments were present in one direct sequence and one XOR sequence.



Ternary digit to encode	A	C	G	T
0	C	G	T	A
1	G	T	A	C
2	T	A	C	G

Figure 2.5: Diagram of the encoding algorithm reported by Bornholt et al. [5]. A key component of this method is the inclusion of the XOR operation. A rotating mapping algorithm was used to avoid homopolymers.

The sixth method was reported by Organick et al. [6]. This method was used to store 35 files totaling 200 megabytes of data. The encoding process for this method can be described by the following procedure shown in Figure 2.6. First, the digital data was partitioned into files and each file was assigned a pair of 20-base sequences for use as primers. The bit-sequence of each file was then randomized by performing an XOR operation with bits generated from a pseudo-random number generator. The randomized bit-sequence for each file was then partitioned into indexed rectangular matrices containing 16-bit cells. Each matrix contained 10 rows and up to 55,000 columns. For error correction, a RS code was applied to each row, and these bits were included as additional columns. Next, each column was treated as a sequence of bits and the address information (the matrix index and column index) were appended to this bit-sequence. This bit-sequence was converted to a trit-sequence, which was subsequently converted to a base-sequence using a rotating mapping code. Two 20-base primer sequences indicating the file index were then attached to each sequence.

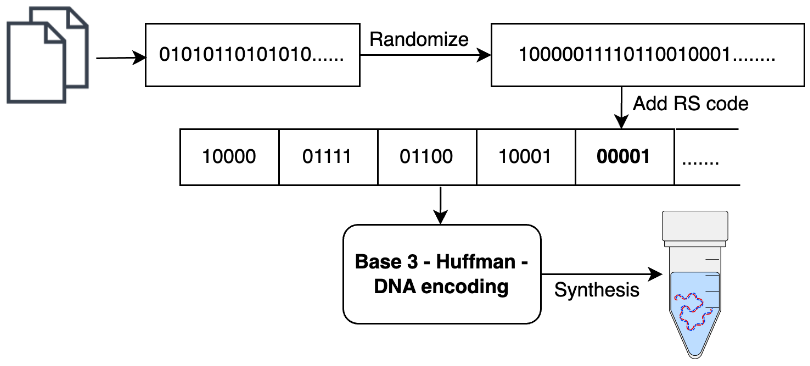


Figure 2.6: Diagram of the encoding algorithm reported by Organick et al. [6]. A key component of this method is the RS code used for generating redundant sequences.

The seventh method was reported by Erlich and Zielinski [7]. This method was used to store a single compressed file representing 2.14 megabytes of data. The encoding process for this method can be described by the following procedure shown in Figure 2.7. First, the file was represented as a bit-sequence and partitioned into equally-sized, non-overlapping segments of 256-bits. A random 32-bit value was generated and used to initialize two Pseudo Random Number Generators (PRNGs). The first PRNG was created over a robust soliton probability distribution and was used to choose the number of data segments to store in the droplet. The second PRNG was created over a uniform distribution and was used to select which segments to include in the droplet. The selected segments were combined into a single 256-bit sequence using an XOR operation. The 32-bit seed was then prepended to the bit-sequence and the resulting 288-bit sequence was encoded using a RS code which appended 16 additional bits for error-correction. This 304-bit sequence was then converted to a 152-base sequence according to a direct mapping code where {00, 01, 10, and 11} map to {A, C, G, and T}, respectively. At this point, the base-sequence was rejected if it had unacceptable GC content or a long stretch of consecutive identical bases. Otherwise, the base-sequence was accepted and added to the list of valid base-sequences. Base-sequences were generated by repeating this process (starting at the generation of a new 32-bit seed) until 7% redundancy had been achieved. At this point, the 72,000 152-base-sequences encoded the 2.14 megabyte file at an information density of 1.57 bits/base. Two 24-base primer sequences were then attached to each base-sequence, bringing the length of each sequence to 200 bases and the information density down to 1.19 bits/base.

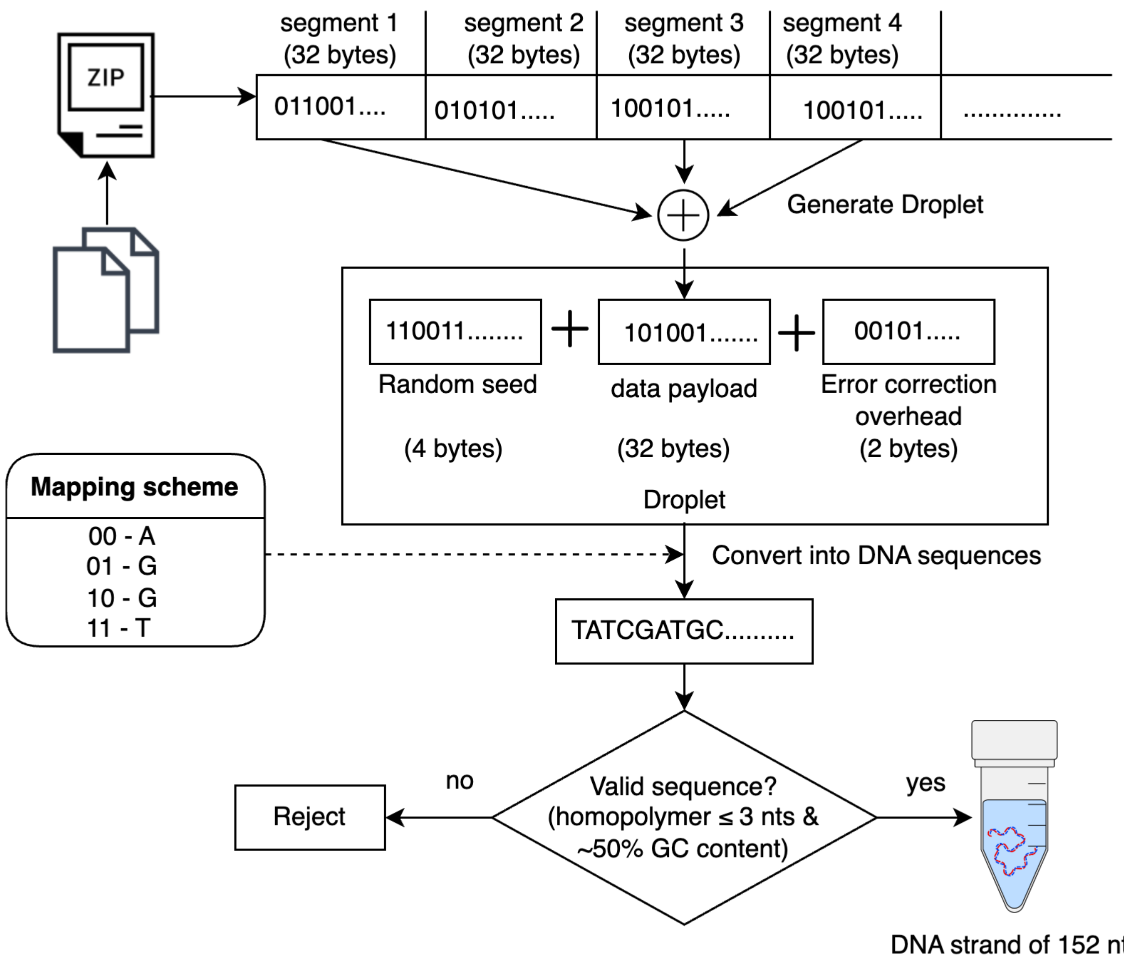


Figure 2.7: Diagram of the encoding algorithm reported by Erlich and Zielinski [7]. A combination of fountain code and RS code was used to provide robustness against dropout.

An eighth method was reported by Anavy et al. [54]. This method was used to store a single zip file totaling 6.4 megabytes of data. The following procedure specifically describes encoding into the 6-letter composite base alphabet. First, the file was represented as a bit-sequence and partitioned into equally-sized, non-overlapping segments of 320-bits. A random 28-bit value was generated and used to initialize two PRNG. The first PRNG was a robust soliton distribution used to choose the number of data segments to store in a given droplet. The second PRNG was a uniform distribution used to select which segments to include in a given droplet. The selected segments were combined using the XOR operation into a given droplet as a single 320-bit sequence. This 320-bit sequence was mapped to a sequence of composite bases according to a word-based mapping code which associated each 5-bit sequence with a 2-composite-base sequence. Composite bases do not represent a specific base (*i.e.* A, T, C, or G), but instead represent the distribution of bases at this position following synthesis. A single base was appended to this 128-composite-base sequence to bring the length to 129 bases. A systematic RS code over GF 7^3 was used to append 6 composite-bases of error correction to this sequence, bringing its total length to 135 bases. A RS code over GF 4^2 was used to append four bits of error correction to the 28-bit random seed, bringing this bit-sequence to 32-bits. This 32-bit sequence was then converted to a (non-composite) 16-base sequence according to a direct mapping code where 00, 01, 10, and 11 map to A, C, G, and T, respectively. The 16-base sequence was appended to the 135-composite-base sequence, yielding a 151-base sequence containing both composite and non-composite bases. At this point, any base-sequences were rejected that contained composite letters not in the original 6-letter composite alphabet. The following base-sequences were then attached to the 151-base sequence: 1) a pair of 20 base sequences used as primers, and 2) a single 3-base sequence used to identify the file or experiment. Multiple droplets (with associated random seed, error correction code, primers, and file ID) were generated by repeating this process (starting at the generation of a new 28-bit seed) until 8% redundancy had been achieved.

Table 2.2: A history and comparison of DNA-based storage advances since 1988.

Authors	File size (MB)	Full recovery	Error handling	Data density $(bits/nt)$	Cost (USD/MB)	Key contribution
J. Davis [62]	4×10^{-6}	No	No	-	-	Introduced DNA as a storage medium
Church et al. [1]	0.65	No	Repetition	0.83	-	Store data in DNA in larger scale
Goldman et al. [2]	0.75	No	Repetition	0.33	$12,400$	Introduced data compression, Handle error by repetition
Grass et al. [3]	0.08	Yes	RS	1.14	$31,250$	Retrieve the data without manual intervention, implement RS code for EC
Yazdi *et. al.* [44]	0.017	Yes	BRDS	-	$236,647$	Increased random access, introduced rewrite ability
Bornholt et al. [5]	0.15	No	Repetition	0.88	-	Reduced redundancy, increased random access
Blawat et al. [4]	22	Yes	RS	0.92	-	Error free retrieval of the data in larger scale
Erlich and Zielinski [7]	2.14	Yes	Fountain + RS	1.57	$3,500$	Incorporated fountain code in DNA, get highest data density
Organick et al. [6]	200.2	Yes	RS	1.1	-	Random access in larger scale
Takahashi et al. [50]	5×10^{-6}	Yes	Hamming	-	$10,000$	Automated device for synthesis and sequencing
Anavy et al. [54]	21.4	Yes	Fountain + RS	$1.93 - 4.29*$	-	Introduced composite DNA letters
Dickinson et al. [21]	2×10^{-5}	Yes	Custom + Fountain	-	-	Introduced an alternate approach to synthesis, sequencing based

More recently, Ping et al. [63] reported a method, the Ying Yang Coding algorithm (YYC), that eliminates long homopolymer runs (allowing for runs of at most three nts). The algorithm achieves this through a flexible codec based on Goldman's rotating code that provides a total of 1536 different transcoding schemes for encoding binary sequences. The algorithm starts by segmenting the binary file into equally sized segments. This is followed by an iterative loop where two segments are randomly selected at the beginning of the loop. For each pair of selected segments, each bit of both segments are processed sequentially, with the first segment's ith bit used to choose one of two possible nucleotide pairings, where 0 maps to 1 of two possiblentcombinations, and 1 maps to the remaining twontcombinations (there are six possiblentpairings for this codec). This is followed by the application of a rotating code that selects antpairing based on the last encodedntand the ith bit of the second segment (giving 256 possible encodings at this step). The intersection of these twontpairs is chosen as the nextntin the encoded output (the construction of the codecs ensures that there is only onentin the intersection of these twontpairs), and the process iterates. The process terminates and rejects the encoded sequence if the GC content of the encoded sequence falls outside of a prescribed range if a homopolymer run of 4 or greater is detected or if the free energy is greater than -30 kJ mol^{-1}. One drawback of their approach is that only 65% of randomly selected segment pairs are able to pass their screening tests in general, and this percentage drops drastically for extremely 0 or 1 biased file segments (although this can be mitigated by compressing the file before processing).

So far in this chapter, our focus has primarily been on the sequence-based DNA storage medium. In the upcoming section, we will shift our attention to dNAM, which is a space-based storage medium. The concept of dNAM has been relatively recent, and only limited work has been conducted in this area, except for the foundational paper [21]. Consequently, the following section will delve into the progression of SMLM, which serves as the basis for dNAM. Furthermore, we will explore the fundamentals of deep learning, as it will be utilized in the emitter localization steps of dNAM.

2.3 Single Molecule Localization Microscopy

Microscopy enables the visualization of objects or samples at a scale that is not perceptible to the naked eye. Conventional light microscopes rely on visible light and lenses for imaging. However, the diffraction phenomenon occurs as light passes through lenses. The image resolution of light microscopy is significantly impacted by this diffraction barrier. In 1873, Ernst Abbe found out a microscope could not isolate two objects located within a distance closer than $\frac{\lambda}{NA}$. Here λ is the wavelength of light and NA is the numerical aperture of the imaging lens [64, 65]. NA represents the light-gathering ability and resolving power of the objective lens of a microscope, determining the maximum angle of light rays that can enter the lens and affecting the image resolution and depth of field. This limits the study of biological samples that are in Nanometer (nm) scale.

To overcome this diffraction barrier and achieve nanoscale resolution, SMLM has emerged. This enables the researcher to observe the cellular structure and study their low-level molecular interactions [66]. In fact, SMLM has revolutionized biological imaging techniques in recent years [67]. SMLM have a wide variety of applications, especially in the sector of biology and medical science. Data storage is a recent application of SMLM [21]. Enhancements in SMLM imaging techniques hold great potential for impact in these sectors.

In SMLM, individual molecules are isolated and imaged over extended periods. In each image, a small fraction of emitters stochastically enters a bright state. The recorded movies capture these bright states, and subsequent analysis using specialized computer programs allows for precise examination of the Point Spread Function (PSF) of each molecule at the nanometer scale. The program extracts the centroid position, photon intensity, and uncertainty of each PSF. Ultimately, the extracted PSF information is combined to construct a super-resolution image. The procedure of SMLM is depicted in Figure 2.8.

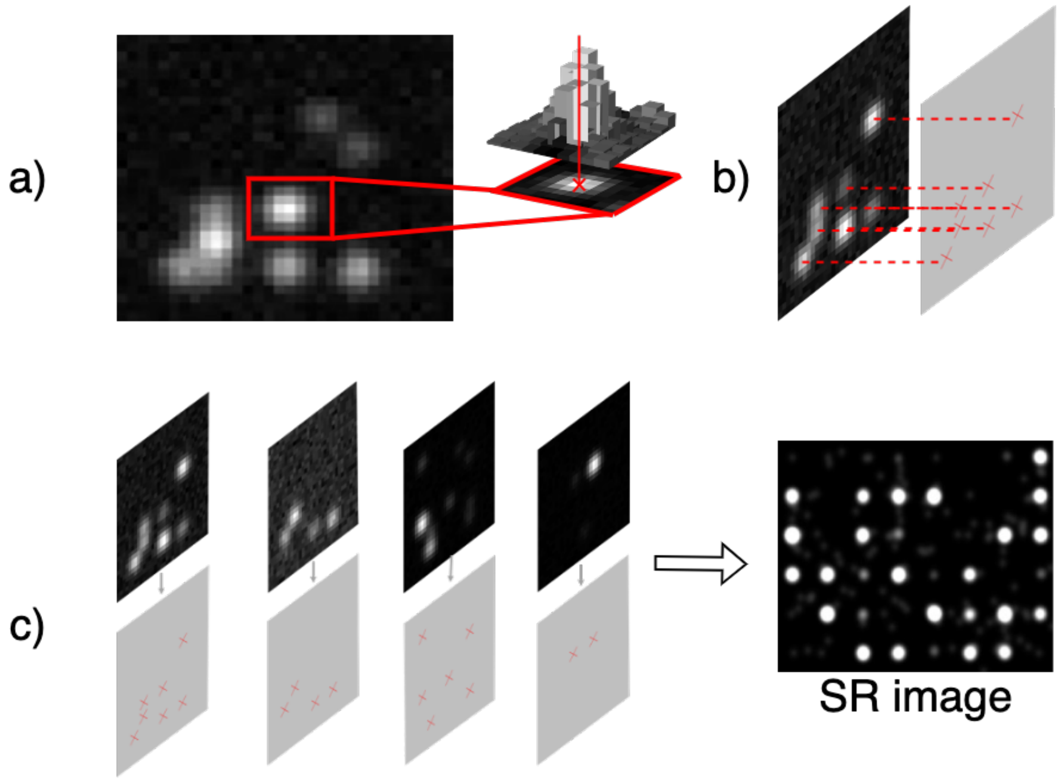


Figure 2.8: Working principle of SMLM. a) Depicts the PSF fitting for an individual emitter from one of many frames. b) All the emitters in an individual frame have been localized. c) Emitters from all the recorded frame have been localized and combined together to construct the SR image.

2.3.1 Sub-pixel Localization

Sub-pixel localization is a crucial step in SMLM. It involves extracting detailed information from the PSF of individual emitters. To assess the performance of different sub-pixel localization methods, the Cramér-Rao Lower Bound (CRLB) is often utilized as a reference. The CRLB represents the minimum achievable localization error that an unbiased algorithm can attain under ideal conditions. These conditions include a Gaussian distribution model for the PSF, homogeneous background models, and non-overlapping PSFs. By comparing the localization results obtained by different methods with the CRLB, researchers can evaluate the effectiveness and accuracy of the sub-pixel localization algorithms [68, 69]. Multiple approaches can be utilized for sub-pixel localization, and a few of them will be discussed in the following sections.

Weighted Mean

The weighted mean method is a basic approach to sub-pixel localization in SMLM. This method involves selecting a local region from the entire image data based on the cumulative photon intensity within that region. It is assumed that each region contains one or more emitters (represented by their PSFs). The centroid position of the emitter is then calculated by taking the mean pixel position, where each pixel's contribution is weighted by its photon intensity. While this method is computationally efficient, it suffers from low accuracy and performs poorly in the presence of noise and dense data. As a result, the weighted mean method is not commonly used in practice. In a study by Henriques et al. [70], the weighted mean method was employed after selecting local regions using the "CLEAN" method developed by Högbom [71] in order to extract the 2D locations of emitters.

Maximum Likelihood Estimation

MLE is the most widely used algorithm for emitter localization [72, 73, 74, 75]. MLE assumes that the emitter's PSF follows a specific probability distribution, often modeled as a Gaussian distribution. MLE computes the centroid position of an emitter through an iterative procedure named gradient descent. At each step, the algorithm updates the previously computed coordinates by a small amount and fits the assumed probability distribution to the data to determine the likelihood of the updated coordinate (see Figure 2.9). Due to its iterative nature, the MLE algorithm can be computationally expensive.

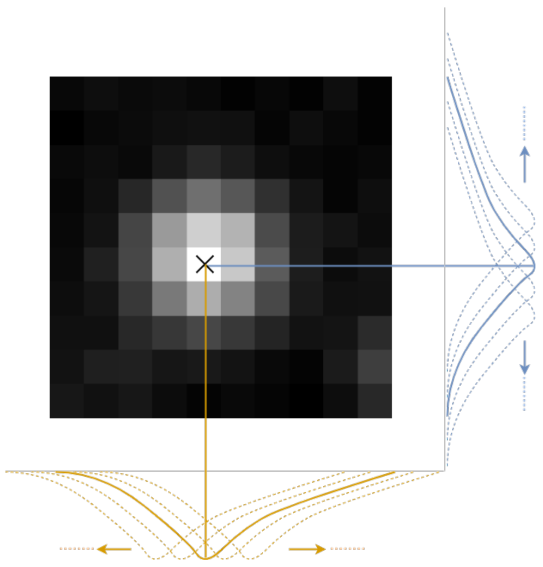


Figure 2.9: The figure illustrates the procedure of finding the centroid position using the MLE algorithm, with the dotted line representing multiple iterations of fitting a Gaussian probability distribution along that line, which may not provide an optimal fit to the data (PSF). On the other hand, the solid line represents the best fit obtained among all the attempted positions.

It is important to note that MLE tends to work better on simulated data where the probability distribution of the emitter's PSF and the noise model are already known. In such cases, MLE attempts to find the optimal coordinate based on the known distribution. However, in real-world scenarios, the emitter's PSF does not always strictly follow a specific probability distribution due to imperfections in the optics or diffraction effects caused by lenses.

In recent years, researchers have increasingly utilized DL techniques for emitter localization. Nehme et al. [37] were the first to employ DL for emitter localization in their work titled "Deep-STORM" [37]. Since then, numerous researchers have applied DL for emitter localization [38, 40, 76, 39]. Like all other field DL also achieved SOTA results. Because of the recent advancements in computer hardware(*e.g.* Graphics Processing Unit (GPU)), DL makes the entire process much faster. The following section provides a fundamental understanding of DL, and detailed information about emitter localization using deep learning can be found in Section 2.4.4.

2.4 Deep Learning

ML is a subset of AI that enables computers to solve complex problems without explicit programming. This is achieved through a process known as training, where the computer learns to solve the problem by analyzing and generalizing from a given dataset. DL, on the other hand, is a sub-field of ML that utilizes multiple layers of abstraction to capture the underlying structure of the training datasets. Each layer of a DL model transforms the input data into increasingly abstract representations, allowing for more sophisticated pattern recognition. The fundamental architecture behind DL is the ANN, which serves as the foundation for modeling complex relationships between inputs and outputs. When an ANN contains numerous layers of abstraction, it is referred to as a Deep Neural Network (DNN). At the core of an ANN is the neuron, which performs calculations on the weighted sum of inputs, followed by the application of an activation function (see Figure 2.10).

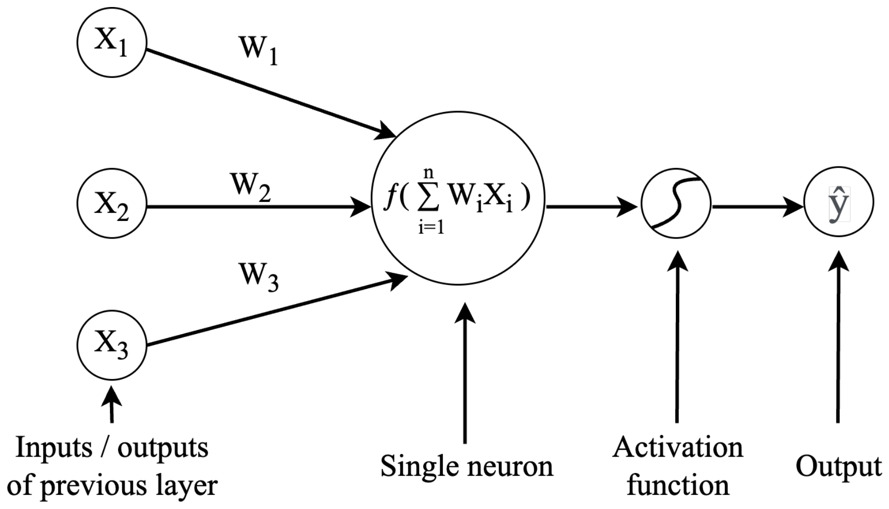


Figure 2.10: Working principle of a single neuron in an ANN. Here X_i represents the input variable. And W_i is the weight of each input. W_i is the learnable parameters that the ANN learns during the training procedure.

Within a layer of an ANN, neurons are independent and do not share connections with each other. However, each neuron is connected to all the neurons in the previous layer. These connections are associated with individual weights, representing the connection's strength. During the training procedure, the ANN learns these weights, which are also referred to as network parameters.

In a DNN model, the number of network parameters can range from a few hundred to several billions [77, 78]. The sheer volume of parameters allows the DNN to capture and model complex relationships in the data, leading to more accurate predictions and higher levels of abstraction.

When the connections between neurons in the ANN do not form cycles, meaning the information flows only in a forward direction, it is termed a feed-forward neural network (see Figure 2.11). In this configuration, data inputs are processed sequentially through the network layers, with each layer transforming the input and passing it to the next layer until the final output is obtained.

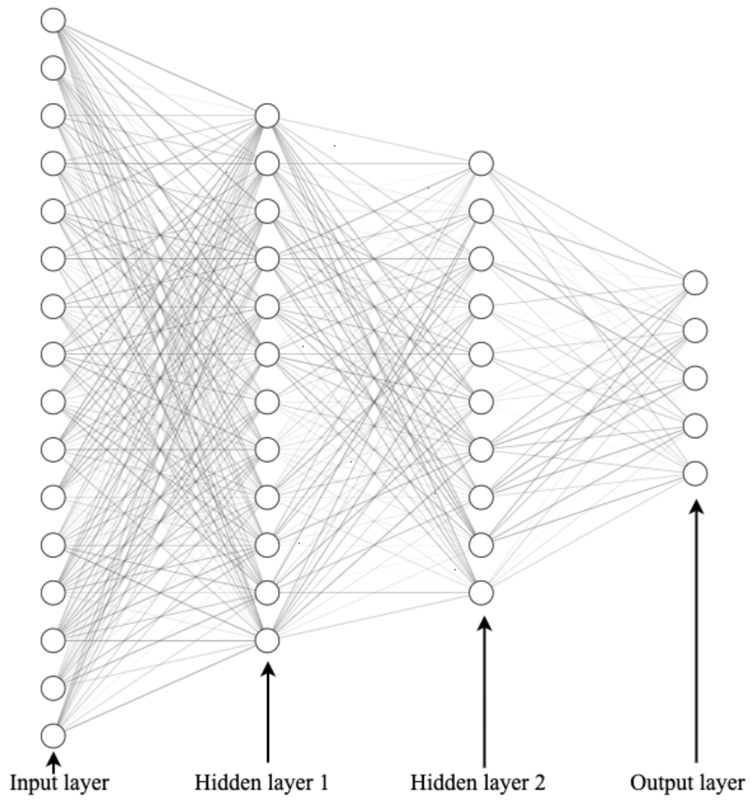


Figure 2.11: Classical example of a feed forward neural network. Here, this network takes 16 feature vectors as input and can predict five different classes. The neurons of one layer are connected with all the neurons in the next layer. Each of these connections has a specific weight. The connection is represented by the gray line. The higher the weight is, the more darker the connection line is.

2.4.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) [79] belong to the broader family of neural network algorithms and are commonly referred to as ConvNets. They are directly inspired by the functioning of the human brain [80, 36]. CNNs are specifically designed to handle matrix-like inputs, such as images. In fact, CNNs have recently achieved SOTA performance in various computer vision tasks [28, 31, 30]. Figure 2.12 provides a visual representation of a classical CNN architecture. In a CNN the architecture typically consists of several layers, each serving a specific purpose in the learning process.

1. Convolutional layer: The convolutional layer is a vital component of a CNN. A CNN can have one or more convolutional layers. These layers extract important features from the input data by convolving the data with a set of filters, also known as kernels. In deeper layers, the extracted features become more exclusive and informative. For instance, in facial recognition applications, earlier layers may detect simple patterns like horizontal or vertical lines, while deeper layers can identify more complex features like eyes or noses in images containing humans. The values of the filters are learnable parameters that are updated during the training process.

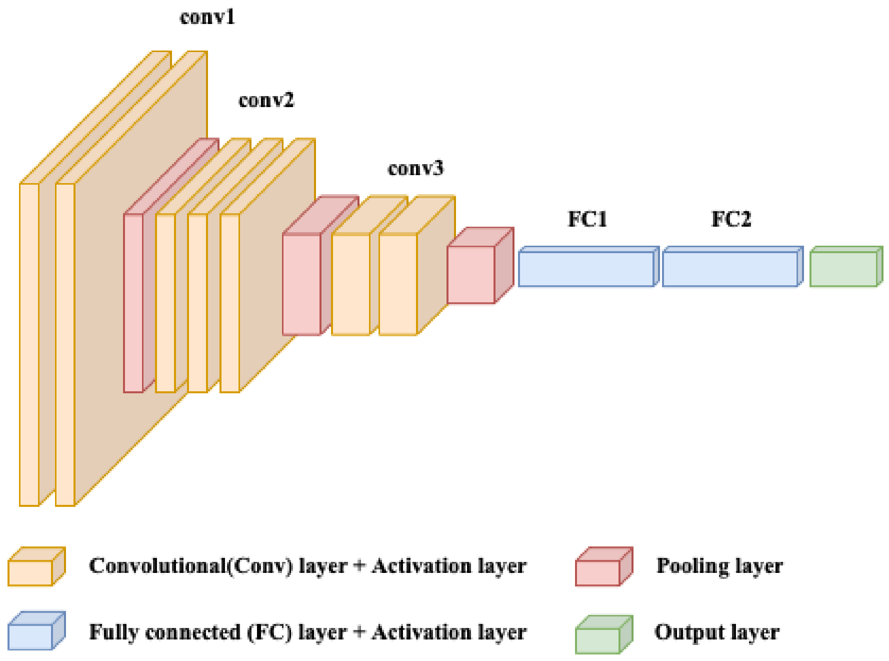

conv1
conv2
conv3
FC1
FC2
Convolutional(Conv) layer + Activation layer
Pooling layer
Fully connected (FC) layer + Activation layer
Output layer

Figure 2.12: Example of a CNN. Each layer have a represented with a separate color. A CNN can have hundreds of these layers [8] .

2. Activation layer: After each convolutional layer, an activation function is applied to the data. This activation function introduces non-linearity in the network. The common activation functions are Sigmoid, Tanh, Rectified Linear Unit (ReLU) [81], Leaky Rectified Linear Unit (LReLU) [82], Parametric Rectified Linear Unit (PReLU) [83], Exponential Rectified Linear Unit (EReLU) [84]. ReLU [81] is the most widely used activation function in DL [85]. In most cases, the activation layer is always applied after the convolutional layer or fully connected layer.

3. Pooling layer: In pooling operations, a certain portion of the data(patch of feature map) is reduced to one single value. The primary purpose of the pooling layers is to progressively reduce the spatial dimension of the data. This op-

eration helps to decrease computational complexity and achieve translational invariance. Pooling also enhances robustness by combining similar features [36]. There are two common pooling operations. i) Max-pooling: maximum value is selected from a patch of feature map ii) Average-pooling: average value of the patch of feature map is used.

4. Fully connected layer: The fully connected layer is a feed-forward neural network layer that is often placed at the end of a CNN. The feature maps obtained from the final convolutional or pooling layers are flattened and used as inputs for the fully connected layers. These layers enable high-level feature extraction and decision-making based on the learned features.

Each of these layers plays a crucial role in the overall architecture of a CNN, allowing it to effectively learn and extract relevant features from input data.

2.4.2 Important Terminologies

In this section, we will discuss several important concepts related to DL.

Training data refers to the data used by ML/DL models to learn and adjust their parameters. It consists of input samples and their corresponding target outputs.

Validation data is a separate dataset used to fine-tune the hyper-parameters of a model during the training process. It helps in selecting the optimal configuration of the model.

Test data is used to evaluate the performance and generalization ability of the trained model. It provides an unbiased assessment of the model's accuracy and effectiveness. The training, validation, and test datasets should be mutually exclusive.

Hyper-parameter are the model settings that are not learned from the data but are set by the user before the training process begins. They include parameters such as learning rate, batch size, and regularization strength.

Learning rate is a critical hyper-parameter that determines the step size taken during the training process. If the learning rate is too small, the learning process will be slow. Conversely, if it is too large, the model's performance may fluctuate or diverge.

Ground truth refers to the actual or correct values of the target data that the ML/DL model aims to predict. It serves as the reference for evaluating the accuracy of the model's predictions.

Loss is a metric that quantifies the difference between the predicted outputs of the model and the ground truth. It represents how well the model is performing on the given task. The goal of training is to minimize the loss function.

Gradient descent is an optimization algorithm used to find the local minimum of a function. It calculates the gradients of the loss function with respect to the model's parameters and updates them iteratively to reduce the loss.

Back propagation is an algorithm used in neural networks to calculate the gradients of the loss function with respect to the model's parameters. It enables efficient gradient computation by propagating the errors from the output layer to the input layer.

Optimizer is a specific algorithm that implements the gradient descent algorithm to optimize the model's parameters during training. Examples of optimizers include stochastic gradient descent (SGD), Adam, and RMSprop.

Overfitting occurs when a model becomes too closely aligned with the training data and fails to generalize well to unseen data. It often results in high training accuracy but poor performance on the validation or test data.

Underfitting happens when a model performs poorly on both the training and test data. It occurs when the model is not complex enough to capture the underlying patterns in the data or when the training dataset is insufficient.

Batch size refers to the number of training examples processed in one iteration during the model training. It affects the trade-off between computational efficiency and model convergence.

Batch normalization(BatchNorm) is a technique used to improve the stability and speed of training deep neural networks. It normalizes the input and output of the activation function in a hidden layer, leading to more stable gradients and faster convergence [86].

Residual connection also known as skip connection, is a technique used to address the vanishing gradient problem in deep neural networks [8]. It provides an alternate path for the data to flow through the network

2.4.3 Super-Resolution Using Deep Learning

The process of generating High Resolution (HR) images from Low Resolution (LR) images is known as Super Resolution (SR) (see Figure 2.13). It has gained significant attention in the field of computer vision [87]. Over the last few years, DL has massively improved the SR technique [88, 89, 90, 10, 91, 92, 93, 94, 9, 95, 96]. A DNN can learn non-linear mapping from LR to HR images, allowing for enhanced image quality.

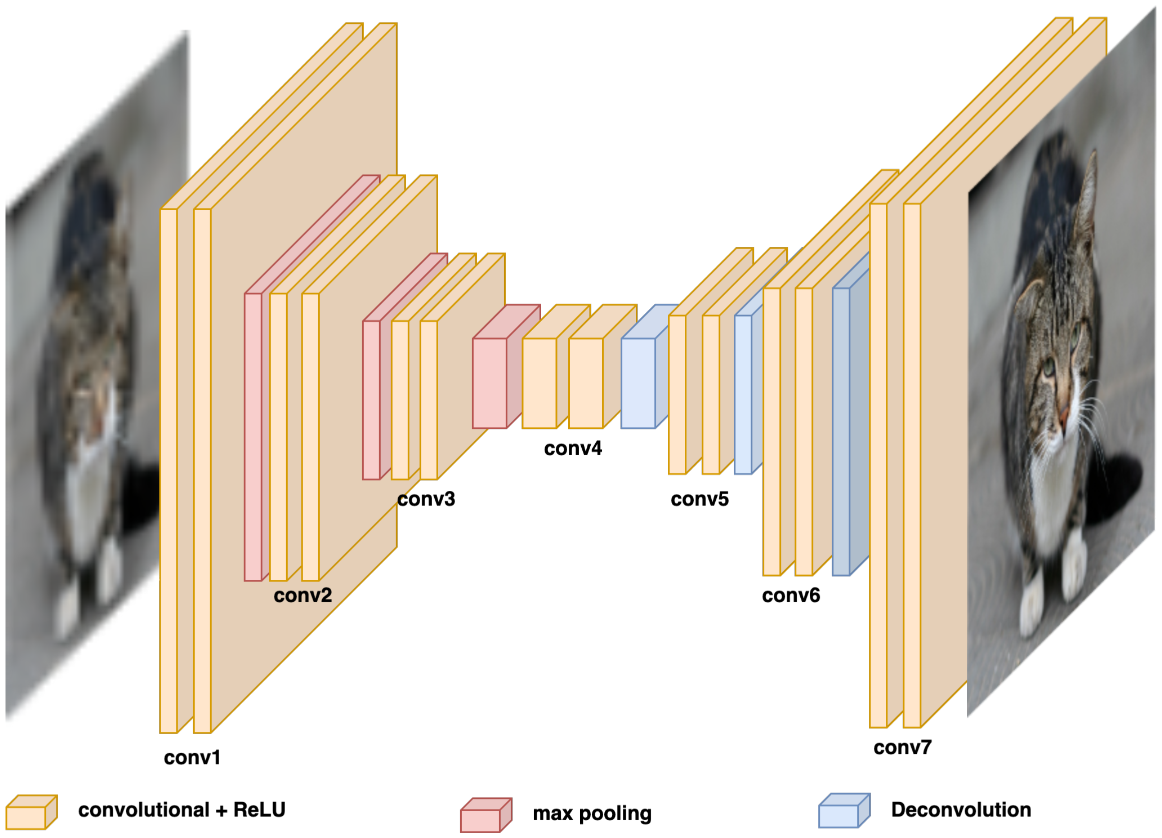


Figure 2.13: Demonstrating single image super-resolution using a deep neural network. The input is a lower-resolution image, and the network generates higher-resolution images by enhancing its quality. During the training process, the network learns a non-linear mapping between the lower resolution image and the corresponding higher resolution image.

Up-sampling Methodologies

One crucial aspect of image super-resolution is increasing the size of the image, also known as up-sampling. Researchers have explored various methodologies to achieve this.

Interpolation

Interpolation is a simple and commonly used technique for image up-sampling. It increases the resolution of the image by filling in the missing pixels based on the neighboring pixel values. The most common image interpolation methods are nearest-neighbor interpolation, bilinear interpolation, bicubic interpolation. Although interpolation algorithms are fast and computationally efficient, they lack the ability to capture complex patterns in the image. Interpolation is often used as a pre-processing step before applying any ML/DL models, effectively enlarging the input image size. This reduces the workload for the subsequent models, as their primary task becomes fine-tuning the image quality. It is important to note that as the input size gets larger, the training and inference times also increase. The computational cost of processing larger images can be a limiting factor, especially for real-time or resource-constrained applications. Therefore, finding a balance between image size and computational efficiency is crucial in practice.

Deconvolution

The idea of deconvolution was first proposed by Zeiler et al. [97] in 2010. Since then, it has been widely used for image up-sampling purposes. This layer is also known as the tranposed convolutional layer [98, 99], inverse/up/backward convolutional layer [100, 101] or fractional convolutional layer [102]. In contrast to interpolation, deconvolution involves learnable parameters that are used to up-sample the image. In regular convolution operation, a convolution kernel/filter is convolved with the image to extract the feature. But in transposed convolution, the kernel/filter is convolved with an extracted feature to get the image. The kernel/filter learning procedure increases the computational time and overall complexity of this algorithm. Because of the sliding nature of the deconvolution operation, the output image contains some uneven overlap patterns. This pattern on spatial dimension creates a problem of checkerboard artifact. This significantly reduces the quality of up-sampled images. More details about these issues can be found in [103].

Sub-pixel layer

Similar to deconvolution, sub-pixel convolution is another learnable method for image up-sampling, initially proposed by Shi et al. [104]. It leverages channel information to upscale the image. For example, we want to upscale an image of shape $H * W * C$ by a factor of r. Here, C is the channel information. Usually, for grayscale images, $C = 1$, and for color image $C = 3$. By using some convolution operation and multiple filters, first, the channel information is increased to $H * W * r^2C$. Later by reshuffling the channel, information is moved in the spatial dimension. And the size of the final output will be $rH * rW * C$. The operation is depicted in Figure 2.14. Compared to the deconvolution approach, sub-pixel convolution offers notable advantages. It is log_2r^2 times faster, significantly reducing the computational burden. Additionally, this methodology encompasses a larger receptive field, enabling the capture of more contextual information during the up-sampling process.

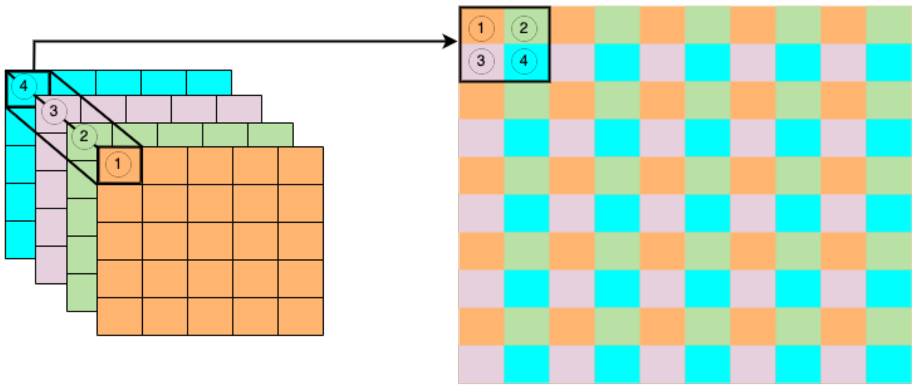


Figure 2.14: Example of the sub-pixel convolution operation. Here we have an input size of $5*5*2^2$ px. The scaling factor r is 2. After reshuffling the desired output shape is $10*10*1$ px.

Loss Functions

An effective choice of the loss function has been observed to facilitate faster convergence of the model. In the context of SR, the selection of an appropriate loss function greatly influences the accuracy of image reconstruction. In recent years, researchers have explored various loss functions for SR tasks. This section provides an overview of some prominent loss functions employed in the SR domain. Throughout this section, h, w, and c represent the height, width, and number of channels of the images, respectively. $\hat{y}$ denotes the ground truth image, while y corresponds to the output generated by the model.

Pixel loss

Pixel loss is a simple yet widely utilized loss function in the field of SR. It involves calculating the pixel-wise difference between the labeled image (ground truth) and the reconstructed image. The choice of measuring this difference can vary, but the two most commonly employed approaches are Mean Absolute Error (MAE) (also known as L1 loss) and Mean Square Error (MSE) (also known as L2 loss).

$$\mathcal{L}_{L1} = \frac{1}{hwc} \sum_{i=1,j=1,k=1}^{i=h,j=w,k=c} \Big| y_{i,j,k} - \hat{y}_{i,j,k} \Big| \tag{2.1}$$

$$\mathcal{L}_{L2} = \frac{1}{hwc} \sum_{i=1,j=1,k=1}^{i=h,j=w,k=c} \Big(y_{i,j,k} - \hat{y}_{i,j,k} \Big)^2 \tag{2.2}$$

The pixel difference of $\mathcal{L}_{L2}$ is penalized by a square term, so this loss is more sensitive to outliers. In SR domain, $\mathcal{L}_{L1}$ loss works better than $\mathcal{L}_{L2}$ loss [105, 10, 106, 87].

Perceptual loss

To address the limitations of pixel loss in capturing contextual information, a perceptual loss function was introduced by Johnson et al. [107] in 2016. Instead of solely considering individual pixels, this loss function incorporates high-level features extracted from both the up-sampled image and the ground truth image. To extract these features, a pre-trained model such as VGG [108] or ResNet [8], which has been trained on the large-scale ImageNet dataset [109], is utilized. The pre-trained model extracts high-level features that capture more complex patterns and structures in the images. The perceptual loss is then computed by measuring the Euclidean distance between these high-level features, providing a measure of perceptual similarity between the up-sampled and ground truth images.

$$\mathcal{L}_{perceptual} = \frac{1}{hwc} \sum_{i=1,j=1,k=1}^{i=h,j=w,k=c} \sqrt{\Big(\phi(y_{i,j,k}) - \phi(\hat{y}_{i,j,k}) \Big)^2} \tag{2.3}$$

Here, ϕ is the output of the pre-trained model. In the case of converting a LR human facial image to HR, it is beneficial to compare specific regions of interest, such as the eyes, nose, and hair, at different resolutions. This targeted approach allows the perceptual loss to focus on capturing the fine details and textures specific to those regions. However, it is important to note that utilizing pre-trained models for extracting high-level features can increase computational time due to the size and complexity of these models.

Performance Metrics

Measuring the quality of an image is a challenging task, as there is no universally agreed-upon metric that can accurately assess image quality. Human perception remains one of the most reliable methods for evaluating image quality, as it takes into account the complexities and nuances that are difficult to quantify. However, human perception-based assessments can be subjective and inconsistent, as individual preferences and biases can influence the perceived quality.

In addition to subjective evaluations, several objective metrics have been developed to quantify the similarity between two images. These metrics aim to capture various aspects of image quality, such as fidelity, distortion, and structural similarity. Some commonly used image quality metrics include:

Peak Signal-to-Noise Ratio

The Peak Signal-to-Noise Ratio (PSNR) is a widely used performance metric for evaluating image similarity. It is defined as the ratio of the maximum pixel value (P_{max}) to the pixel MSE with respect to a reference image. PSNR is often expressed in a logarithmic decibel (dB) scale to accommodate the wide range of pixel values in different images. A higher PSNR value indicates a smaller difference between the images and is generally interpreted as a closer match to the reference.

However, it is important to note that PSNR only considers pixel-level differences and does not necessarily reflect human perception or preference. For example, in the case of a human facial image, if the hair pixels are slightly shifted, PSNR may show a low value even if the overall quality of the hair image is visually acceptable. PSNR primarily focuses on minimizing pixel-level errors rather than capturing higher-level perceptual qualities.

Therefore, while PSNR is a useful metric for certain applications and scenarios, it should be complemented with other metrics and subjective evaluations to obtain a comprehensive understanding of image quality and its alignment with human preference.

$$PSNR = 20log_{10}\Big(\frac{P_{max}}{\sqrt{MSE}}\Big) \tag{2.4}$$

Structural Similarity Index

Published in 2004 by Wang et al. [110], Structural Similarity Index (SSIM) is another widely used metric to measure the similarity between two given images. Unlike PSNR, SSIM takes into account human visual perception when comparing images. SSIM assesses the similarity of two images based on three key aspects: luminance, contrast, and structure. Luminance, represented by μ, is calculated by averaging the pixel values across the entire image. It captures the overall brightness or intensity of the image. Contrast, denoted by σ, measures the standard deviation of the pixel values and represents the variability or differences in intensity within the image. The structure is evaluated based on the covariance between the pixel values of the two images, accounting for spatial relationships and patterns.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.5}$$

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2} \tag{2.6}$$

SSIM combines these three components to generate a similarity index that ranges between -1 and 1, with 1 indicating perfect similarity. Higher SSIM values indicate greater similarity between the images. Luminance(L_c) and contrast(C_c) and structure(S_c) is compared using the following equation. Where C_1, C_2, and C_3 are the constant.

$$C_l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.7}$$

$$C_c(x,y) = \frac{2\sigma_x\sigma_y + C_1}{\sigma_x^2 + \sigma_y^2 + C_1} \tag{2.8}$$

$$S_c(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.9}$$

Finally, SSIM is measured using equation 2.10. Here $\alpha > 0, \beta > 0, \gamma > 0$ indicate the relative importance/weight for each of these terms.

$$SSIM(x,y) = [L_c(x,y)]^{\alpha} * [C_c(x,y)]^{\beta} * [S_c(x,y)]^{\gamma} \tag{2.10}$$

By considering luminance, contrast, and structure, SSIM provides a more comprehensive assessment of image similarity that incorporates important visual factors. However, it is important to note that SSIM may not capture all aspects of human perception and can still exhibit variations based on different test conditions and subjective preferences.

Network architecture

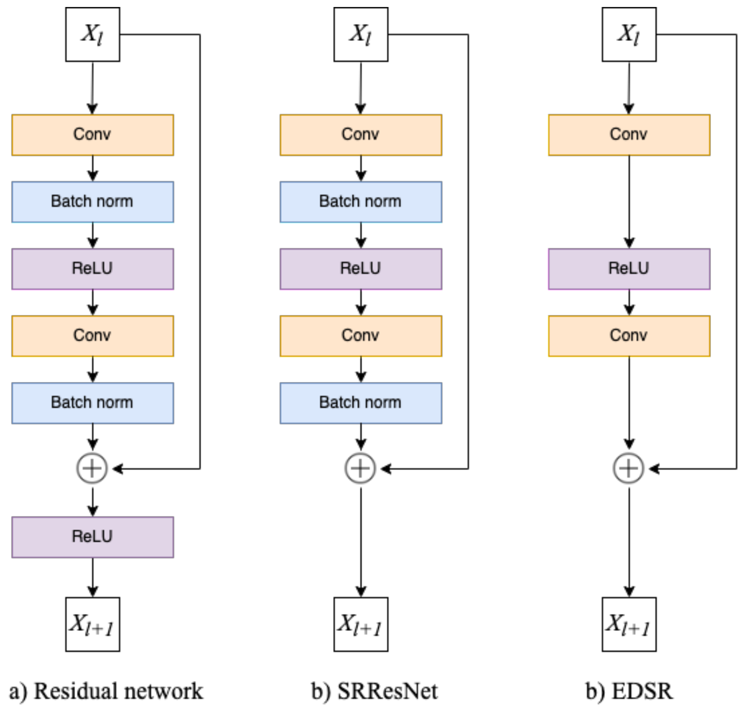


Figure 2.15: Figure shows one of many residual blocks from a CNN architecture. X_l and X_{l+1} is the input and output of layer l, respectively. a) Original ResNet work proposed by He et al. [8]. b) In SRResNet [9] author used a modified version of ResNet. Where they removed ReLU activation after residual connection. c) Lim et al. [10] were able to achieve a better result by removing batch normalization from SRResNet.

In 2015, Dong et al. [91] pioneered the use of DL for SR. They introduced a network architecture consisting of three layers; the purposes of those layers are: i) feature extraction, ii) non-linear mapping, and iii) image reconstruction. The input of this network is the LR image which is up-scaled using bicubic interpolations and the output is the SR image of the input image. However, this approach is computationally intensive due to the use of upscaled inputs, and it may suffer from artifacts such as the checkerboard effect [103].

To address these challenges, Shi et al. [95] proposed the idea of sub-pixel convolution [104] in order to increase resolution while maintaining computational efficiency. They introduced the concept of rearranging the channel information in the spatial dimension, allowing for faster training and avoiding checkerboard artifacts. Kim et al. [88] used similar architecture as proposed by Dong et al. [91]. The author made the network deeper and introduced residual connections [8], which further improved the performance.

Table 2.3: Comparison of super resolution methodology

Methods	Year published	Up-sampling	Loss function	Primary contribution
SRCNN [91]	2015	Bi-cubic interpolation	L2	First use of DL in SR domain
ESPCN [95]	2016	Sub-pixel	L2	proposed sub-pixel layer for first time
DRCN [111]	2016	Bi-cubic interpolation	L2	used recursive block
LAPSRN [90]	2017	Bi-cubic interpolation	L1	increase resolution progressively
SRGAN [9]	2017	Sub-pixel	Adversarial, perceptual, L1	Used GAN for first time
EDSR [10]	2017	Sub-pixel	L1	Remove batch norm from ResNet
RDN [112]	2018	Sub-pixel	L1	Global residual connection
DBPN [89]	2018	Deconvolution	L2	Multiple up- and down-sampling for self correction
ESRGAN [113]	2018	Sub-pixel	L1	Improve SRGAN
SR3 [93]	2021	Bi-cubic interpolation	L1	Repetitive refinement

Deeply-Recursive Convolutional Network (DRCN) [111] uses recursive blocks. The parameters of the recursive block are updated multiple times in a recursive fashion in a single training run. As the same parameter is updated multiple times, the number of parameters does not increase. That makes the network smaller in size for faster training and inference.

In the case of the Laplacian Pyramid Super Resolution Network (LapSRN) [90], two separate neural networks are trained simultaneously. One network extracts features for different scales (*e.g.* 2x, 4x, 8x), while the other network reconstructs the image for each scale. The extracted features from the first neural network are concatenated with the corresponding scaled layers of the second neural network, enabling progressive resolution enhancement.

Super Resolution Generative Adversarial Networks (SRGAN) [9] introduced the use of Generative Adversarial Networks (GAN) [114] for SR. The SRGAN architecture consists of a generator based on a residual network (SRResNet) and a binary classifier as the discriminator. This approach aims to generate photo-realistic high-resolution images.

Enhanced Deep Super Resolution Network (EDSR) and Multi-Scale Deep Super Resolution (MDSR) are two different networks proposed by [10]. EDSR is a single-scale model that can increase the image's resolution at a specific scale, while MDSR is a multi-scale model capable of generating images at multiple scales (*e.g.* , 2x, 4x, 8x). The authors removed the batch normalization layer from SRResNet. The batch normalization layer normalizes the feature during the training process, introducing range flexibility in the network. Normalization in each layer consumes more memory and increases computational time. Removing batch normalization enabled them for faster convergence. High-scale models are trained from pre-trained low-scale models.

Residual Dense Network (RDN) by Zhang et al. [112] adopts a similar network architecture to MDSR. Along with the local residual connection, they also used a global residual connection within the residual block. Deep Back-Projection Networks (DBPN) [89] introduce a projection layer for SR. HR and LR images alternate in the projection layer, which helps them to learn the self-correction mechanism.

Saharia et al. [93] use denoising diffusion probabilistic models [115] to conditional image generation for creating super-resolution images. Here the output image starts with pure Gaussian noise, and a U-Net [35] model is trained to remove Gaussian noise from the output image iteratively. Table 2.3 shows the comparison between recently proposed DL based SR methodologies.

2.4.4 Emitter Localization Using Deep Learning

In 2018, Nehme et al. [37] used deep learning for emitter localization for the first time. The authors employed an encoder-decoder based CNN architecture. The network's input is the entire field of view of the image that may contain overlapping emitters. The network up-sampled the input image by a factor of eight, where emitter positions were projected on a high-resolution grid. To train the model, the author first generated diffraction-limited image and their corresponding super-resolved image. The super-resolved images are used as the target image. They test their data on both simulated data and experimental microtubule datasets Sage et al. [116]. However, it is important to note that this approach does not provide the list of emitter centroid positions, which limits its ability to achieve nanoscale resolution.

In their work, Mannam et al. [76] proposed a U-Net [35] like architecture where the primary goal is to avoid frame-by-frame localization to save computational time.

Here the input is the diffraction-limited images, and the output is the super-resolved images. The input image is created by combining all the frames into one single image. They used SRRF [117] an ImageJ [118] plugin, to generate their target super-resolved image. As their target image is generated from another plugin, they can never surpass the accuracy of that plugin. Their training sample is very small (750 images), which makes the training very fast compared to other methods.

In comparison, DeepLoco [39] model can achieve similar accuracy compared to the conventional MLE algorithm [12, 119] with much faster speed. The input image is generated using a computer simulation. The model is validated using both simulated data and experimental data. The network contains a convolutional layer followed by fully connected layers with skip-connection. The final output have two head. The first head predicts the probability of having an emitter in a position. And the second head outputs a 3D vector that represents the 3D position of the emitters. The author claimed they were able to process 20,000 frames within $\sim$1 second.

Table 2.4: Comparison of deep learning based emitter localization methodology

Methods	Year published	Architecture	Loss function	Comments
DeepSTORM [37]	2018	Encoder-decoder based CNN	L2	Used DL for SMLM for first time
smNET [120]	2018	Similar to ResNet	L2	Take individual images of PSF and output 3-D coordinate of PSFs, PReLU, Batch norm
DeepLOCO [121]	2018	CNN with residual connection	L2	Introduced a novel loss function, predict 3-D coordinate of PSFs
ANNA-PALM [122]	2018	GAN, U-Net	L1, SSIM, Adversarial	Take widefield image as input and return corresponding super-resolved 2-D image
BGnet [123]	2020	U-Net	L2	Take individual PSF as input and return background and intensity of that input.
Deep-STORM3D [38]	2020	CNN with skip connection	L2, dice loss [124, 125]	Take 2-D low-resolution images and return 3-D high resolution volume
DECODE [40]	2021	Two stacked U-Net	Count loss, background loss & localization loss	Take individual images as input and return emitters position along with their intensity and uncertainty.
DenseED [76]	2021	U-Net like architecture	L2	Work on small data set, take diffraction limited image as input and return SR images as output

Deep context dependent (DECODE) by Speiser et al. [40] used two stacked U-Net [35] architectures to predict the emitter position along with their uncertainty directly. As the emitter's activation persists over several frames, DECODE uses adjacent frames to improve emitter detection and localization. So, in the first step, three separate U-Net model takes three adjacent frames as input. These U-Net extract the feature from the image. All the extracted features are concatenated together. These concatenated features work as an input of another U-Net model. This last U-Net model makes the final predictions. For each pixel k, the network predicts: i) p_k that maps the probability that an emitter is detected near that pixel, ii) the coordinate of the detected emitter, relative to the center of the pixel iii) a non-negative value measuring emitter brightness (photons count), iv) uncertainty associated with each of these predictions. Their loss function have three terms: i) Count loss ii) localization loss and iii) background loss. The counting loss is responsible for predicting the total number of emitters present in the image. And the localization loss is responsible for predicting their exact position. Both losses work together where they fit a spatial point process probability distribution Baddeley et al. [126]. The background loss is the MSE between the predicted background and ground truth background.

DeepSTORM3D [38] is an extended version of the previously reported model DeepSTORM [37]. The author trained a neural network that takes 2D image of overlapped Terapod PSFs and outputs a 3D grid with a voxel size $27.5*27.5*33\ nm^3$. This fixed voxel size limit their localization accuracy. The model uses a CNN with residual connections [8]. Their architecture has three main modules: i) multi-scale context, which extracts features from the 2D image, ii) an up-sampling module that increases the resolution of the image by four-fold. iii) the last module refines the depth and lateral position of the predicted emitter.

CHAPTER 3: LOCALIZATION

3.1 Data Generation

DL is an algorithm that heavily relies on a large volume of training data. However, in the context of origami, it is not practical to physically produce the vast quantities of origami required for training a deep neural network. Furthermore, due to the unpredictable nature of how origami can land in various positions and orientations, obtaining precise ground truth labels for the exact position of each origami becomes unattainable. Consequently, both the synthesis of a sufficient number of origami samples and the acquisition of reliable ground truth labels for them are infeasible tasks.

Fortunately, simulations offer a viable alternative by providing the flexibility to generate dense or sparse origami configurations within a single movie. This enables us to create as much data as necessary, accompanied by their corresponding ground truth information. Through simulation, we can overcome the limitations of physical origami production and obtain the required dataset for training deep learning models.

The simulated origami data generation process involves the following steps. Firstly, the position of each origami is determined, either based on a grid position or randomly selected. Subsequently, each origami is assigned to its respective position with a random orientation. The arrangement of the origami in the scene establishes the total number of points for the entire movie. These points are stochastically activated in each frame, and a specific number of photons is assigned to each point for every frame.

The blinking behavior of the emitters is governed by user-defined parameters, such as the mean dark time and mean bright time. The frequency of the blinking events for individual binding events depends on these parameters. To determine the on-time (ONT) and off-time ($OFFT$) for each emitter, an exponential distribution is utilized. Equation 3.1 represents the relationship, where x denotes the random variable and λ represents the mean dark time or mean bright time.

$$ONT/OFFT = f(x;\lambda) = \lambda e^{-\lambda x} \tag{3.1}$$

Subsequently, the number of photons for each "on" event (P) is assigned using a normal distribution, as represented by equation 3.2. This distribution is parameterized by the user-defined values of the photon rate (μ) and photon standard deviation (σ). The assigned photon count is then prorated by the total integration time (t). This allows for realistic variations in the photon count, reflecting the inherent uncertainty and fluctuations in the experimental data.

$$total\ photon\ (P) = f(x;\mu;\sigma) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.2}$$

An "on" event can persist for multiple consecutive frames. For each frame within an "on" event, an exact number of photons (p) is assigned using a Poisson distribution, as described by equation 3.3. This distribution is parameterized by the total number of photons (P), which was previously assigned using equation 3.2. The Poisson distribution captures the randomness in the photon emission process, where the number of photons emitted in each frame is independent of the others. It reflects the probabilistic nature of photon emission and accounts for the variability observed in experimental data.

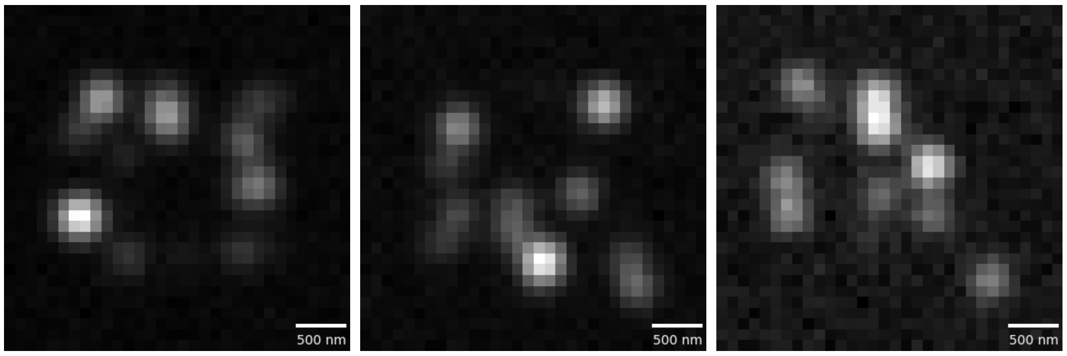
$$frame\ photon\ (p) = f(x; P) = \frac{P^x}{x!} e^{-P} \tag{3.3}$$

For each individual frame, the position of the binding event is determined by sampling from a multivariate normal distribution. This distribution is parameterized by the exact position of the binding event, which represents its spatial coordinates.

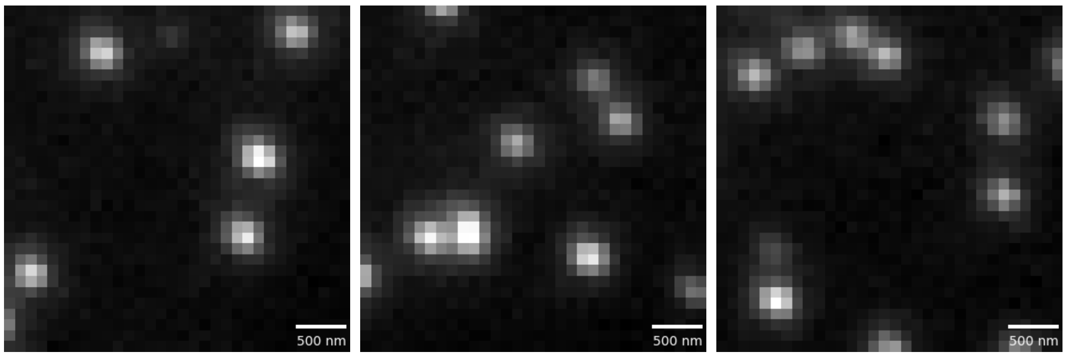
To assign photons within a frame, a 2-dimensional normal distribution is employed. The mean of this distribution corresponds to the photons assigned for that specific binding event in that particular frame. The standard deviation of the distribution is a user-defined parameter known as the (PSF). By sampling from this distribution, the photons are distributed around the center of the binding event, simulating the spread of emitted photons.

By repeating this process for all binding events and frames, a collection of photons is generated, and a 2D histogram is constructed based on their spatial distribution. This histogram represents an image or frame that mimics the observed data in the experiment.

To evaluate the fidelity of the simulated images, a visual comparison is presented in Figure 3.1, showcasing the similarities and differences between the simulated images and experimental images. This visual assessment allows for a qualitative analysis of the simulation's accuracy in capturing relevant features and characteristics of the experimental data.



(a) Simulated image



(b) Experimental image

Figure 3.1: Visual comparison between simulated and experimental images

3.2 Performance Metrics

Performance metrics play a crucial role in evaluating the effectiveness of algorithms, providing insights into their performance. In the context of localization, we employ two types of performance metrics: those based on image similarity and those based on localized points.

3.2.1 Cross Correlation

Cross-correlation (CC) is one of the image similarity metrics we utilize to measure the similarity or dissimilarity between two images. The CC metric, as defined in Equation 3.4, compares two separate images, denoted as X and Y. At each level of up-sampling, we compare the output of our algorithm with the corresponding ground truth data. The CC equation returns a value ranging from 0 to 1, where a higher value indicates a higher degree of similarity between the images.

$$NCC(X,Y) = \frac{\sum_{i=1,j=1}^{i=h,j=w} \left(X_{i,j} - \bar{X}\right)\left(Y_{i,j} - \bar{Y}\right)}{\sqrt{\sum_{i=1,j=1}^{i=h,j=w} \left(X_{i,j} - \bar{X}\right)^2}\sqrt{\sum_{i=1,j=1}^{i=h,j=w} \left(Y_{i,j} - \bar{Y}\right)^2}} \tag{3.4}$$

3.2.2 Jaccard Index

Accurately identifying the total number of emitters is crucial for constructing high-quality images from SMLM data [127, 13]. False identification of emitters can degrade the image quality. Therefore, it is important to maximize True Positive (TP) detections and minimize False Positive (FP) and False Negative (FN) detections. TP is defined as the correct identification of an emitter within a certain threshold radius (r), while FN represents undetected emitters. On the other hand, FP occurs when the algorithm detects an emitter at a position where none exists (Figure 3.2).

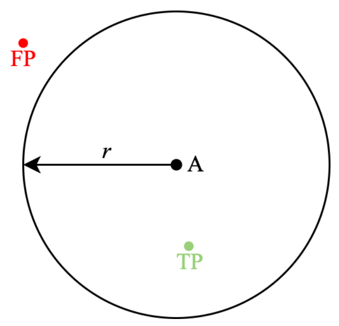


Figure 3.2: ***A*** **is the ground truth position of an emitter. If the prediction is within the r nanometer radius, then the prediction is considered as TP, otherwise prediction is FP.**

To evaluate the accuracy of emitter detection, we need a metric that takes into account these three terms and provides a single value. The Jaccard Index (JI), also known as the Jaccard similarity coefficient, is a similarity measurement metric between two sets. In our case, one set consists of the predicted emitter points, and the other set consists of the ground truth emitter points. The JI is calculated using Equation 3.5, and a higher value indicates a higher similarity between the prediction and the ground truth.

$$JI = 100 * \frac{TP}{TP + FP + FN} \tag{3.5}$$

3.2.3 Root Mean Square Error

Root Mean Square Error (RMSE) measures how close the predicted emitter centroid position is with respect to ground truth. It is calculated using equation 3.6 where here x_p^i represents the prediction of x coordinate of i^{th} emitter. x_{GT}^i is the corresponding ground truth. The lower value indicates better predictions.

$$RMSE(nm) = \sqrt{\frac{1}{TP} \sum_{i \in p} \left(x_i^p - x_i^{GT}\right)^2 + \left(y_i^p - y_i^{GT}\right)^2} \tag{3.6}$$

3.2.4 Efficiency

To compare different algorithms and obtain a single metric, Sage et al. [13] proposed the efficiency metric, which combines both the JI and the RMSE. The efficiency is calculated using Equation 3.7, where α serves as a trade-off parameter between JI and RMSE. In our subsequent results, we adopt the value $\alpha = 1\ nm^{-1}$ as proposed by the authors.

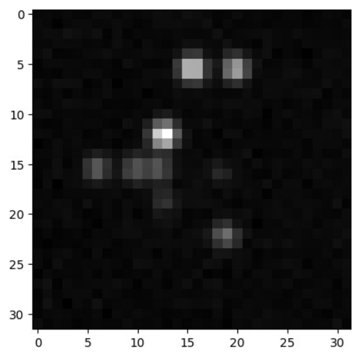
$$efficiency = 100 - \sqrt{(100 - JI)^2 + \alpha^2 RMSE^2} \tag{3.7}$$

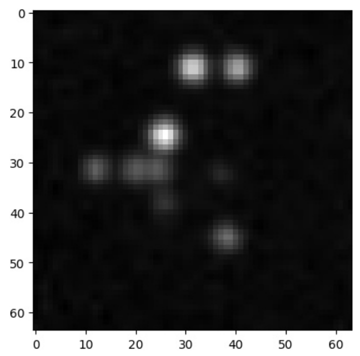
3.3 Image Up-sampling

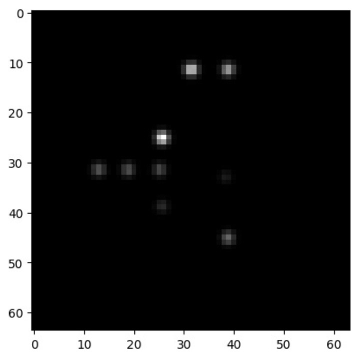
Our DL model takes a different approach to localize the emitter position. Instead of directly predicting the position, the model rescales the centroid of the emitter in a higher-resolution image while preserving its PSFs. This allows for the separation of overlapped PSFs in the higher-resolution image (*i.e.* 16x), enabling accurate localization. These higher-resolution images are then stacked together to form a super-resolved image, where most of the emitters are isolated and noise is reduced by the neural network. As a result, the task of emitter detection becomes easier, and a simpler algorithm, like connected component analysis, can be used to isolate the individual emitters. The connected component algorithm identifies and labels individual connected regions or objects within a binary image. To calculate the centroid positions along the x and y axes, we can employ another neural network (Section

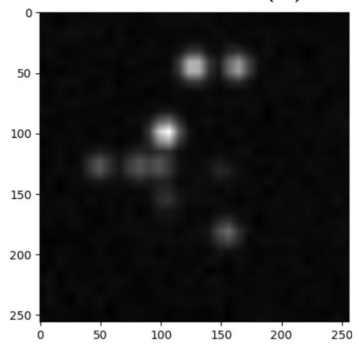
3.4.1) or a simpler method like the weighted mean (Section 3.4.1). This multi-step approach allows us to achieve accurate emitter localization while leveraging the benefits of super-resolution imaging and neural network processing.

When using conventional interpolation methods like bi-linear or bi-cubic interpolation to increase the resolution of a frame, the centroid position of the emitter does not scale up accordingly. Instead, the centroid position remains unchanged while the PSF of the emitter becomes larger. This does not benefit us for emitter isolation. Figure 3.3b demonstrates the effect of bi-linear interpolation at a 2x scale, which makes the separation of emitters more difficult. This difficulty is exacerbated as the resolution increases, as shown in Figure 3.3d. To overcome this challenge, we require a non-linear mapping that not only increases the resolution but also separates the emitters from each other. This mapping should move the centroid position of the emitters as the scale increases while keeping the PSF unchanged. DL has achieved remarkable success in image super-resolution and emitter localization tasks, making it a suitable approach to learn this non-linear mapping. By increasing the image resolution by a factor of 16, we empirically observed that most of the emitters can be successfully isolated. However, it is important to note that higher-resolution images consume more memory and increase training time. Therefore, a trade-off must be considered. In our case, increasing the resolution by 16 times allows for effective emitter isolation, and we subsequently extract the centroid positions of the emitters to evaluate the accuracy of our architecture.


(a) 1x – input image


(b) 2x – Bi-linear


(c) 2x – target


(d) 8x – Bi-linear

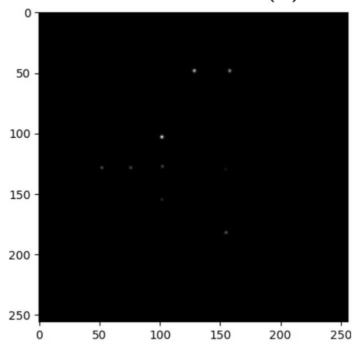

(e) 8x – target

Figure 3.3: The figure illustrates the issues with bi-linear interpolation methods for image up-sampling. In (a), the original image is displayed, containing several emitters. The objective is to isolate the emitter's centroid positions while preserving their PSFs during the up-sampling process, as shown in (c) and (e). However, the bi-linear interpolation method merely increases the image size without isolating the emitters. It retains the emitter's centroid position from the lower resolution image and distorts the emitter's PSFs, as depicted in (b) and (d).

3.3.1 Model

The core architecture of our model is based on the U-Net [35](shown in Figure 3.4), which was proposed in 2015 and has demonstrated state-of-the-art performance in biomedical image segmentation tasks. The U-Net architecture consists of two main sections: a contracting section and a symmetric expanding section. The contracting section helps the model understand the contextual information in the image, while the expanding section enables precise localization based on that contextual understanding. These two sections are connected through residual connections [8].

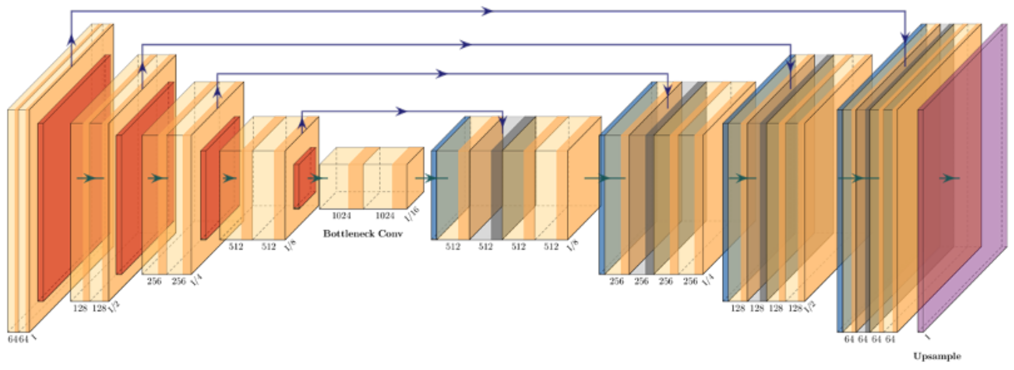


Figure 3.4: The figure depicts a schematic diagram of a U-Net architecture. It comprises a contracting path on the left side, which captures context through convolutional and pooling layers, followed by a symmetric expansive path on the right side that enables precise localization using transposed convolutions. The incorporation of skip connections between corresponding levels on both paths facilitates the flow of information. At the final layer, an additional up-sample layer is utilized to increase the resolution of the image, employing sub-pixel convolution for this purpose.

We utilize the U-Net architecture to progressively increase the resolution of the image. To achieve this, we employ two separate U-Net models, as shown in Figure 3.5. The input image undergoes two forward passes in the first model, resulting in a fourfold increase in resolution. The final output of the first model is then fed into the second model, which performs two additional forward passes to further increase the resolution by a factor of 16. Both models have the same shape and size but have distinct sets of learned parameters.

Our model consists of approximately 63 million trainable parameters. However, we observed that during the early iterations of training, the output from each model is not sufficiently close to the corresponding ground truth. If we directly use these outputs from earlier epochs as inputs to the second model, the model becomes confused as these outputs contain significant noise. To address this issue, we apply bi-linear interpolation to increase the resolution of the input image and concatenate it with the respective output from the neural network. This provides the necessary information to train the neural network effectively during the early epochs.

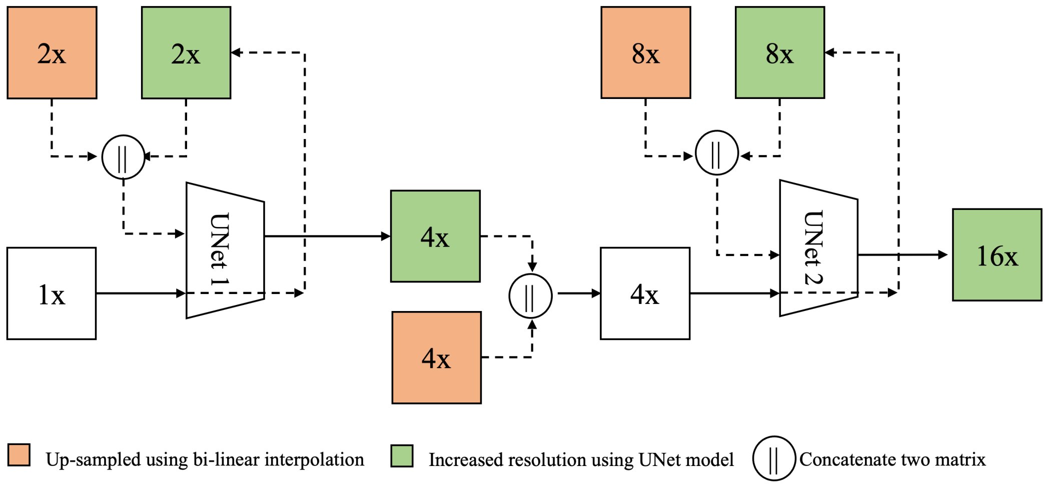
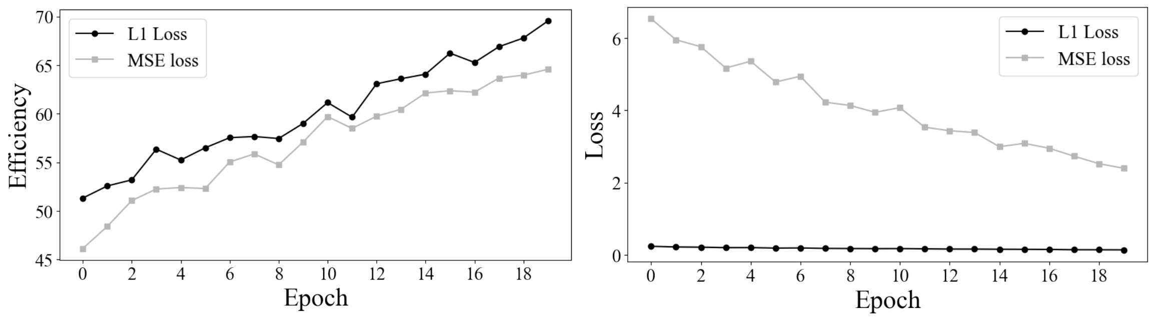

2x
2x
8x
8x
UNet 1
UNet 2
1x
4x
4x
4x
16x
Up-sampled using bi-linear interpolation
Increased resolution using UNet model
Concatenate two matrix

Figure 3.5: The figure illustrates the overview of our deep learning architecture designed for image up-sampling. Our architecture progressively increases the image resolution by factors of 2x, 4x, 8x, and 16x. To achieve this, we utilize two stacked U-Net architectures. Each time the images pass through the U-Net architecture, the resolution is increased by two times, and the weights of the neural network are updated during this process. The first U-Net is responsible for up-sampling from 1x to 4x, while the second U-Net further increases the resolution from 4x to 16x. This multi-step approach allows us to obtain high-resolution images by effectively leveraging the capabilities of the U-Net architecture at different stages of the up-sampling process.

Optimization Objective

Mean absolute errors ($\mathcal{L}_{L1}$) (equation 2.1) and mean squared errors ($\mathcal{L}_{L2}$) (Equation 2.2) are the two most widely used loss functions for image super-resolution. As shown by [128], $\mathcal{L}_{L1}$ loss works better for the image super-resolution domain. We tested both $\mathcal{L}_{L1}$, $\mathcal{L}_{L2}$, and perpetual loss (Equation 2.3) functions and found out $\mathcal{L}_{L1}$ also works better for our applications (shown in Figure 3.6).

The intuition behind the choice of loss functions is as follows: perceptual loss works best for natural images, as it compares images based on their features in the latent space. However, our images contain PSFs that do not possess rich features like natural images. On the other hand, the $\mathcal{L}_{L2}$ loss is more sensitive to outliers. Given these considerations, we opted for the $\mathcal{L}_{L1}$ loss as it provides better performance for our purposes.



(a) Efficiency comparison **(b) Loss comparison**

Figure 3.6: L1 loss provides better efficiency compared to L2 loss. The black line indicates L1 loss and the gray line indicates L2 loss. L2 loss have a square term which is the reason of having a higher value in each epoch compared to L1 loss.

It is worth noting that each of our models has its own loss function. For example, the loss function for model one only updates the parameters of model one, while the loss function for model two updates the parameters for both models one and two. This separation allows for individual optimization of each model while jointly training them to achieve the desired outcome.

Optimizer

During the training process, we employed the Adam optimizer [129] with default parameter values. Specifically, we set $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^9$. These values are commonly used and have been shown to work well in various DL tasks.

To control the learning rate during training, we utilized a variable learning rate strategy. Initially, we set the learning rate to 10^{-4}. If the validation error did not decrease for ten consecutive epochs, we reduced the learning rate by a factor of 0.1. The minimum learning rate allowed during training was set to 10^{-7}. This approach allowed the model to adapt its learning rate based on the progress of the training process, helping to improve convergence and potentially avoid getting stuck in suboptimal solutions.

Training data and Pre-processing

Our training data consist of 180,000 images. Furthermore, validation data have 20,000 images. Both training and validation are from entirely different simulations to isolate image emitter distributions.

During the pre-processing stage of our training data, we performed specific steps to prepare the images for training. The details of the pre-processing steps vary depending on the specific model and scaling factor.

Although DL models generally perform well when the output values are between 0 and 1, in our case, we introduced the penalty mechanism to handle the sparsity in the higher-resolution images and ensure the retention of important emitter information. For the first model, which handles 2x and 4x scaling, we scaled the pixel values of the images to be in the range of 0 to 1. This scaling operation helps to ensure that the input data falls within a reasonable range for better convergence during training.

However, for the 8x and 16x models, simply scaling the pixel values between 0 and 1 was not sufficient. As the resolution increases, the images become sparser, with a significant portion of the pixel values being zero. This can pose a challenge because the neural network could achieve high accuracy and low loss by predicting all zero values, effectively removing all the emitters in the higher-resolution.

To address this issue, we introduced a penalty for predicting non-zero pixel values as zero. Each non-zero pixel value was multiplied by a higher number (specifically, 255) during the training process. This multiplication effectively increased the loss when the network predicted a non-zero pixel as zero. By penalizing these predictions, we encouraged the network to retain the non-zero values and preserve the emitters' information in the higher-resolution images.

It is worth noting that while this approach helps to prevent the network from discarding the emitters, choosing an appropriate multiplier is important. If the multiplier is too large, it can slow down the training process or lead to other issues. Empirically, we determined that a multiplier of 255 worked well for our purposes.

3.4 Point Extraction

In order to obtain the super-resolved image, the up-sampled frames can be merged into a single frame, effectively increasing the image resolution. However, for the purpose of measuring the JI of our predictions, the coordinates of the emitters are required. While MLE algorithms can be utilized for extracting these points, they are computationally intensive. Given that most of the emitters are well separated at a 16x resolution, a simpler algorithm such as a connected component-based approach can be employed to achieve higher accuracy while reducing computational complexity.

To implement this approach, the emitters are initially isolated using a connected component algorithm [130]. Subsequently, a weighted mean is performed within these isolated regions (Section 3.4.1). Additionally, a CNN has been employed for this purpose, yielding improved results in comparison to the weighted mean and MLE approaches (see Figure 3.10).

3.4.1 Weighted Mean

As the emitters are isolated in the up-sampled image, it is much easier to extract them. We used a connected component algorithm to extract patches from the whole image. Each patch contains one or multiple emitters. In most cases, each patch encompasses only one emitter. However, there are instances where emitters remain connected even in the 16x up-sampled version. In such cases, the weighted mean approach fails to accurately locate these emitters. For instance, if two emitters are attached together at the 16x resolution, the weighted mean would select a point at the center of the two emitters, resulting in an incorrect prediction. Consequently, both emitters are not detected (false negative), and an additional false emitter prediction occurs (false positive), resulting in a total of three errors. By disregarding this specific scenario, we can reduce the errors to two false negatives. To identify such cases, we examine the non-zero elements within the extracted patch sizes. Typically, when multiple emitters are connected, the patch sizes increase and contain more non-zero elements. The threshold size for considering a patch containing two emitters depends on the properties of the emitter PSF. In our case, we determined that a threshold of 65 provides the optimal efficiency. Figure 3.7 illustrates the efficiency in relation to different threshold sizes.

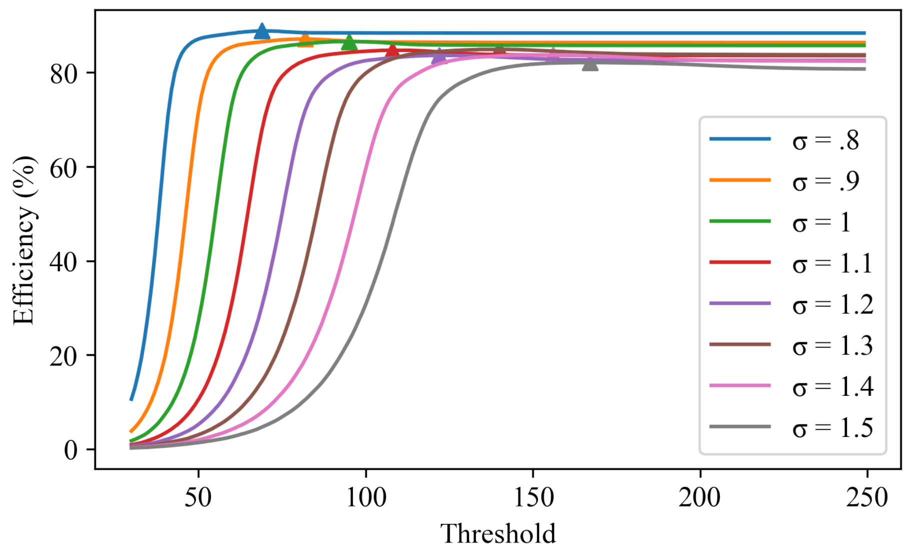


Figure 3.7: The figure shows the efficiency vs. threshold for weighted mean based centroid position extraction of an emitter. Here, threshold means the number of non-zero elements in an extracted patch. If the number of non-zero elements exceeds the threshold value, the corresponding patch is completely disregarded. As this threshold depends on the spread of the PSFs, we tested that multiple PSF having separate standard deviation(σ). The triangle on each line represents the threshold value for which the highest efficiency was achieved. If the threshold is low, then lots of single emitters are also ignored. So, at a lower threshold, we are seeing a very bad efficiency. But after a certain threshold, the efficiency dropped a little bit and remained constant with a higher threshold. Having an attached multi-emitter at 16x resolution is not a common phenomenon. So, even if we ignore this rare event, it does not increase our efficiency a lot. For example, for $\sigma = 1$, we get the highest efficiency 86.6% at threshold 65. And at threshold 99, the accuracy becomes 85.7%. So, by using the threshold, we were able to improve the accuracy by 1%. Here, each test was done on 5000 16x up-sampled images. And the threshold radius(r)(section 3.2.2) for JI was 10 nm.

3.4.2 Neural Network

The weighted mean approach achieved pretty good efficiency at 16x up-sampled resolution. Nevertheless, the presence of attached emitters at this level reduces the overall efficiency of the algorithm. To address this issue, we incorporated a neural network for extracting emitter centroid positions. The utilization of neural networks serves two main purposes: (1) achieving more precise localization and (2) potentially enhancing efficiency by successfully separating two connected emitters that were undetectable by the connected component algorithm.

Data Generation

The training data for our neural network were generated using 16x up-sampled label images. Since we have access to the ground truth information, we performed a breadth-first search from the ground truth to isolate all the connected emitters. Subsequently, we applied random padding to the extracted patches, which resulted in the emitter being placed at a random position within the patch. The size of the padding was set to 30×30 pixels. However, when we experimented with symmetric padding, where the emitter remains at the center position, the neural network exhibited overfitting behavior. In such cases, the model struggled to compute the centroid position if the emitter was not located at the center.

After padding, the extracted region was fed into a neural network for predicting at most two points. For a single emitter, our neural network predicts six properties:

p – Probability of having an emitter in that patch

x – Coordinate of the emitter along the horizontal direction

y – Coordinate of the emitter along the vertical direction

σ_x – Standard deviation of the emitter along the horizontal direction

σ_y – Standard deviation of the emitter along the vertical direction

i – Intensity of the emitter

Since our model predicts two points for each extracted region, the final output of the neural network is a tensor of shape 12. We utilized ResNet [8], specifically the 18-layer variant, as our base model. ResNet, introduced by He et al., is a renowned architecture primarily designed for image classification tasks. We modified the first and last layers of the architecture to enable our model to process grayscale images instead of RGB images and to adapt the number of final outputs to 12 for our specific requirements. The resulting model had approximately 10 million trainable parameters.

Loss Function

We employed a combination of two loss functions for our model. For the probability (p) prediction, we utilized the binary cross entropy loss (Equation 3.8):

$$\mathcal{L}_{BCE} = -\frac{1}{m}\sum_{i=1}^{m}\Big(p_i log\hat{p_i} + (1-p_i)log(1-\hat{p_i})\Big) \tag{3.8}$$

Here, m represents the total number of training examples, p_i denotes the model's predicted probability of an emitter's presence in the i^{th} training example, and $\hat{p}_i$ represents the corresponding ground truth.

For the emitter location and the associated standard deviation, we experimented with three different loss functions. Among them, the $\mathcal{L}_{L1}$ loss (Equation 2.1) yielded the most stable results. A comparison between $\mathcal{L}_{L1}$, $\mathcal{L}_{L2}$, and $\mathcal{L}_{huber}$ losses is depicted in Figure 3.8. The Huber loss (Equation 3.9) combines both the $\mathcal{L}_{L1}$ and $\mathcal{L}_{L2}$ losses.

If the absolute element-wise difference falls below a certain threshold (δ), the Huber loss employs the $\mathcal{L}_{L2}$ loss; otherwise, it utilizes the $\mathcal{L}_{L1}$ loss. This choice is based on the observation that when the difference is small, the likelihood of the value being an outlier is low. Conversely, when the difference is large, the probability of the value being an outlier increases. Consequently, in such cases, the $\mathcal{L}_{L1}$ loss is more suitable, as the $\mathcal{L}_{L2}$ loss is more sensitive to outliers.

$$\mathcal{L}_{huber} = \begin{cases} \frac{1}{2}(y - \hat{y}), & if \quad |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - \frac{1}{2}\delta^2, & otherwise \end{cases} \tag{3.9}$$

One of the primary motivations for utilizing a neural network for point extraction is to identify connected emitters. Therefore, patches containing two emitters are considered more important than those containing only a single emitter. To emphasize this distinction, we multiplied the loss for scenarios with connected emitters by a factor of 10. This adjustment allows the neural network to prioritize learning from these examples more effectively.

$$\text{final loss} = \text{single emitter loss} + 10 * \text{multi emitters loss}$$

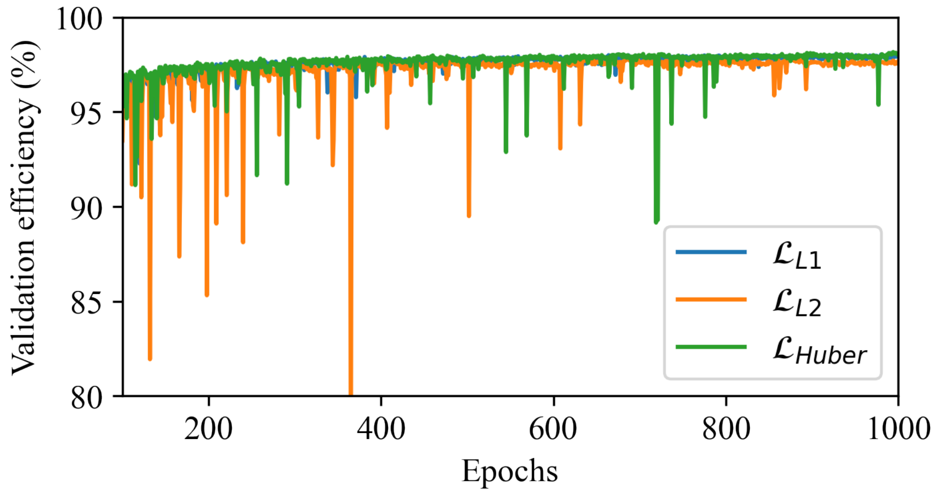
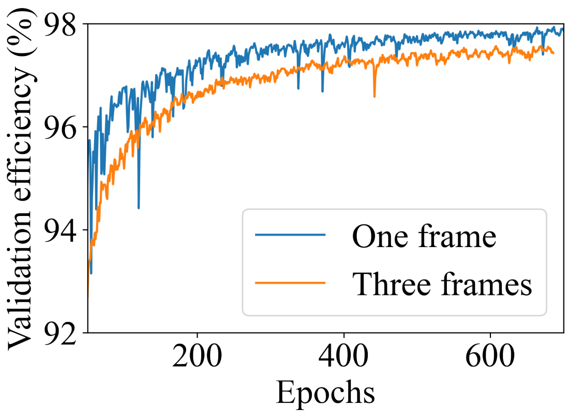


Figure 3.8: Efficiency comparison between different loss functions for point extractor model. Here $\mathcal{L}_{L1}$ returns the best and more stable result. $\mathcal{L}_{L2}$ is more sensitive to outliers, so we are noticing some ups and downs during the training process. Huber loss is less sensitive to data compared to $\mathcal{L}_{L2}$, so it is showing better stability than $\mathcal{L}_{L2}$.

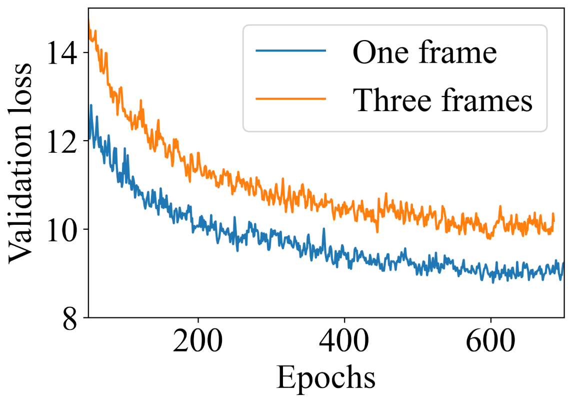
Results

Emitters typically emit signals across multiple subsequent frames, with their intensity potentially varying from frame to frame. A study by Speiser et al. [40] demonstrated that the detection accuracy of emitters can be improved by using three consecutive frames as input. However, in our case, we observed that our accuracy remained nearly the same even when we used adjacent frames as inputs (see Figure 3.9).

Since we extracted patches from a single frame, we also extracted corresponding positional patches from the adjacent frames to maintain spatial consistency across the inputs. Despite not seeing a significant improvement in accuracy, this approach allowed us to maintain the same level of performance as when using three consecutive frames as input.



(a) Efficiency comparison



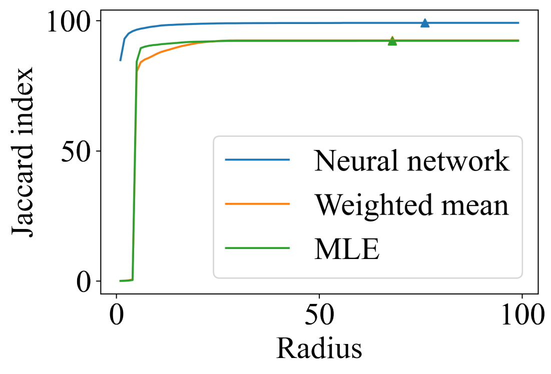
(b) Loss comparison

Figure 3.9: To extract emitter's centroid position from frame t, we used only that frame in a single frame label. And we used frame $t-1$, t, and $t+1$ in three frame labels. Here single frame analysis provides better results compared to multi-frame analysis.

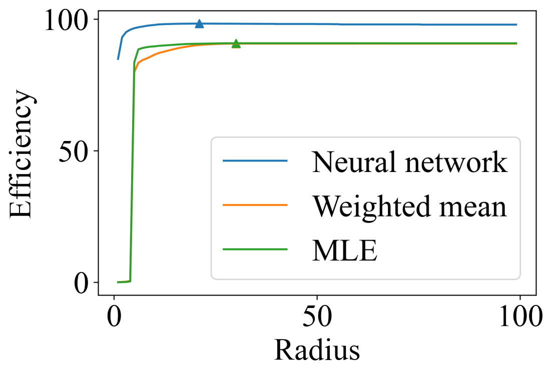
Our neural network-based approach demonstrated superior efficiency and JI compared to the weighted mean and MLE approaches. For a radius (r) of 10 (as described in Section 3.2.2), our approach achieved an efficiency of approximately 98%, whereas the weighted mean and MLE approaches had efficiencies of approximately 86% and 89%, respectively (see Figure 3.10).

When we increase the radius, more points are considered true positives, which leads to an increase in the RMSE because a greater distance between the ground truth and predictions is also counted as true positives. This is why we attained the highest efficiency at an earlier radius compared to the JI.

Furthermore, our neural network-based approach is approximately 36 times faster than the MLE-based approach. We implemented the MLE approach using the Python package SciPy [131].



(a) Jaccard index comparison



(b) Efficiency comparison

Figure 3.10: The figure shows a comparison between three different point extraction methods. Here we use the 16x up-sampled label image to extract the emitter points. The test was done with 1000 frames. The triangular shape indicates the position with minimum radius with the highest efficiency/jaccard index. After that point, there is no effect on increasing the threshold radius(r).

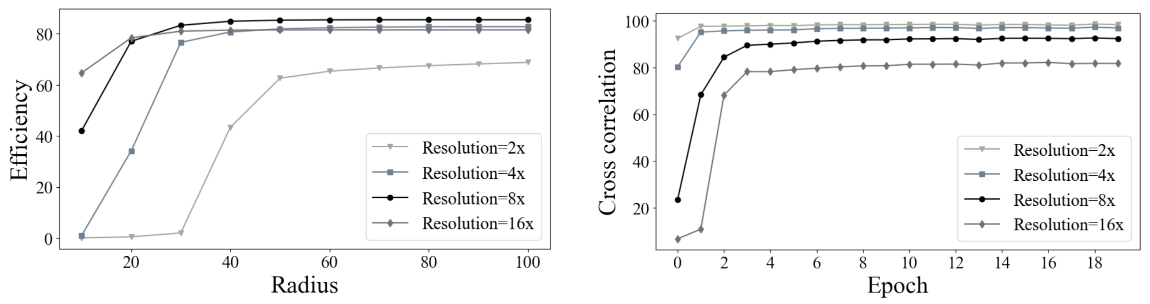
3.5 Results and Discussions

We conducted extensive testing and validation of our method using various metrics and datasets. As previously mentioned, our neural network progressively enhances the image resolution while preserving the PSF of the emitters. Throughout the training, validation, testing, and inference stages, we obtained images at multiple resolutions, such as 2x, 3x, 4x, and 8x. We computed the centroid positions of the emitters from these images to analyze their impact on the target metrics and gain insights into emitter separability.

Figure 3.11a presents a comparison of efficiency across different resolutions. The x-axis represents the threshold radius for classifying an emitter as a true positive, while the y-axis represents the efficiency (see Section 3.2.4). As depicted in the figure, the lowest efficiency was observed at a 2x image resolution, which was expected. At this resolution, most emitters remained merged together, posing a challenge for the connected component algorithm to effectively isolate individual emitters and leading to low efficiency. For instance, in Figure 3.13a, two emitters are merged together in the input image. At 2x resolution (Figure 3.13b), they are only partially isolated. However, the isolation was insufficient for the connected component algorithm to identify them as separate emitters. As a result, the emitter extraction algorithm (e.g., weighted mean, neural network, maximum likelihood estimation) considered these two emitters as a single emitter and assigned a centroid position in the middle of the two emitters. This error represents a false positive, as the algorithm failed to isolate the two real emitters, which are marked as false negatives. Consequently, at 2x resolution, we have a total of three errors. However, this issue is not observed in higher-resolution images (e.g., 4x, 8x, 16x), where these two emitters are properly isolated (see Figure 3.13). That ultimately led to poor efficiency.

Nevertheless, as we increased the resolution, the efficiency improved significantly. At an 8x resolution, a greater number of emitters were successfully isolated compared to the 2x resolution, resulting in higher accuracy. Surprisingly, the best performance was achieved at 8x rather than 16x resolution. This observation suggests that the separability of emitters is relatively similar between 8x and 16x resolutions. Additionally, each increase in resolution caused the neural network to lose some information, leading to a degradation in the overall resolution quality (see Figure 3.11b). This degradation resulted in a deviation in the centroid positions of the emitters, specifically impacting the efficiency at the 16x resolution. For example, Figure 3.12 demonstrates how a low signal emitter can be lost at higher levels (see Figure 3.12e). Moreover, since only a few emitters exhibited greater separability at 16x compared to 8x, their contribution to the efficiency improvement was not significant. More localization example of our localization algorithm can be found in Appendix B.

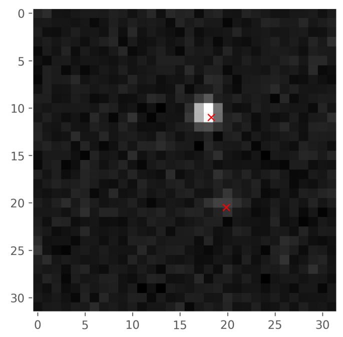
In conclusion, our findings demonstrate the influence of image resolution on emitter separability and efficiency. While higher-resolutions generally enhance efficiency, the loss of information in the neural network and limited separability at 16x resolution resulted in a slight decrease in performance compared to 8x resolution.

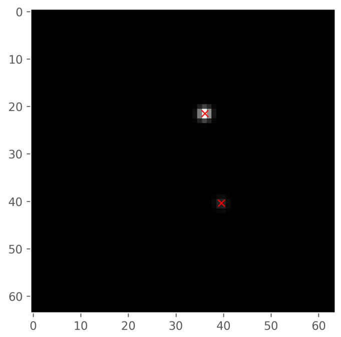


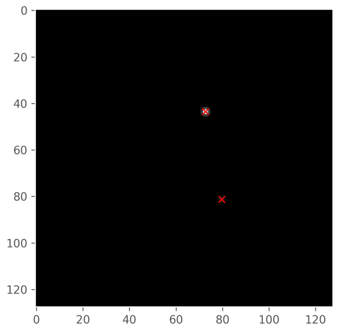
(a) Efficiency comparison between different resolutions

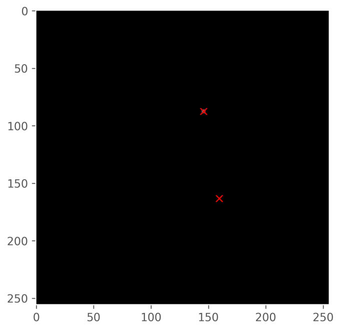
(b) Cross correlation comparison between different resolution

Figure 3.11: As our neural network progressively increases the resolution of the image, we calculate the metrics after each increment in resolution. (a) depicts the efficiency after each increase in resolution, and (b) shows the cross-correlation comparison between different resolution levels.


(a) 1x – input image


(b) Prediction from 2x


(c) Prediction from 4x


(d) Prediction from 8x

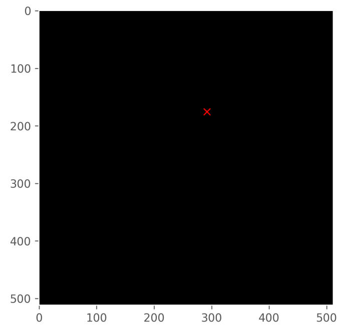
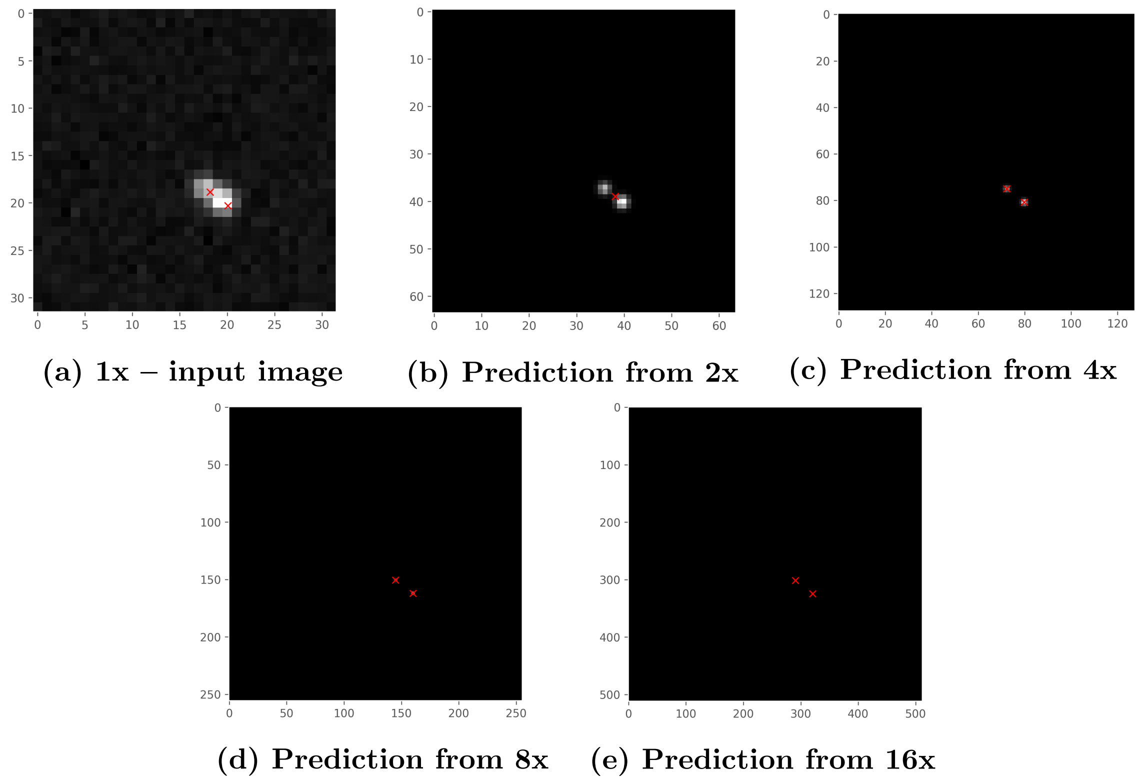

(e) Prediction from 16x

Figure 3.12: The presented images demonstrate the occurrence of information loss during the up-sampling process. The red 'x' symbols denote the true positions (ground truth) or predicted positions (b, c, d, e) of the emitters. Notably, the information pertaining to one of the emitters was rendered indiscernible in the subsequent layers of the neural network (at a 16x up-sampling factor) due to diminished signal strength.


(a) 1x – input image (b) Prediction from 2x (c) Prediction from 4x

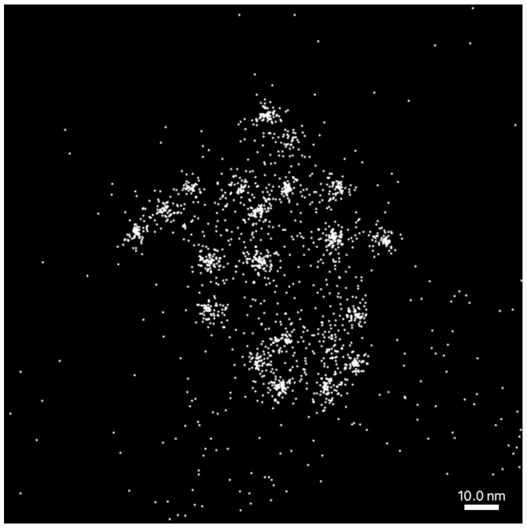
(d) Prediction from 8x (e) Prediction from 16x

Figure 3.13: In the input image (a) we can see there are two emitters that are merged together. However, those emitters are predicted as a single emitter at 2x resolution (b). Because they were not isolated enough at 2x resolution. However, at higher-resolution both the emitters were identified and localized correctly.

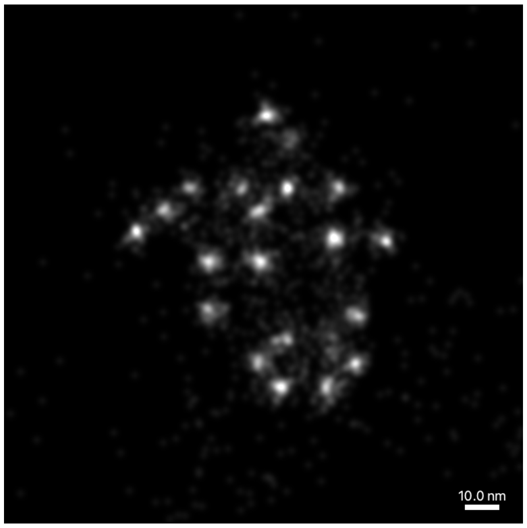
Figure 3.14 presents a visual comparison of the final output generated by our algorithm. In this comparison, we utilized the neural network to up-sample the image and subsequently employed the weighted mean method (see Section 3.4.1) for extracting emitter positions. Our visual results closely correspond to the previously observed efficiency comparison at multiple resolutions. Notably, we observed that the quality of the output improves as the resolution increases.

In the context of the “One-Pixel-Blur” technique, each emitter is subjected to blurring using a Gaussian density function with a standard deviation of one. The

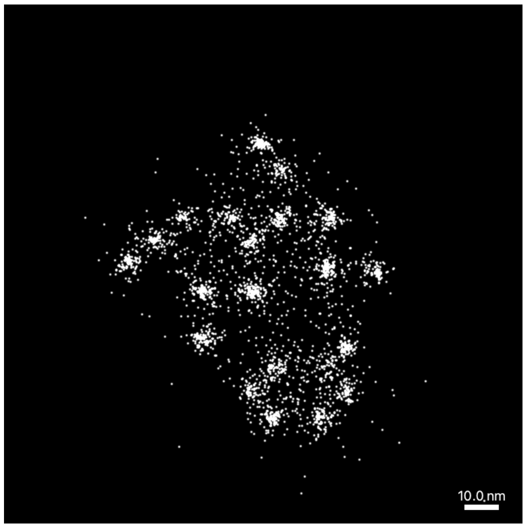
"Global Localization Precision" metric bears resemblance to the "One-Pixel-Blur" technique. Nevertheless, the Gaussian kernel's standard deviation is calibrated to align with the median precision of all localization coordinates. For visualizing the results, we utilized the render module from Picasso's work [11].



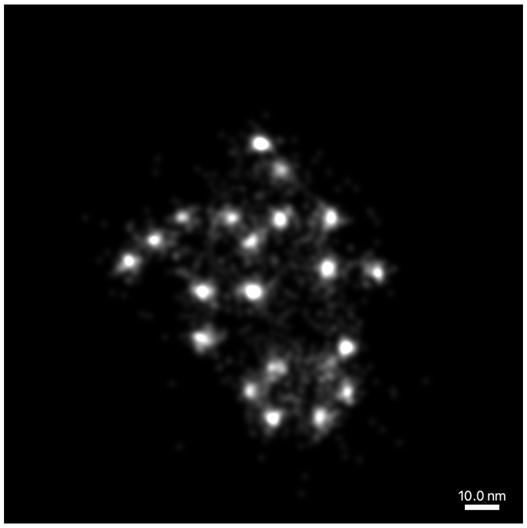
(a) Prediction from 2x – one pixel blur



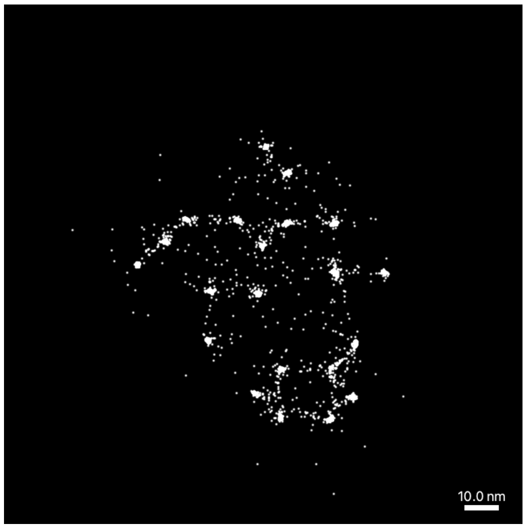
(b) Prediction from 2x – global localization precision



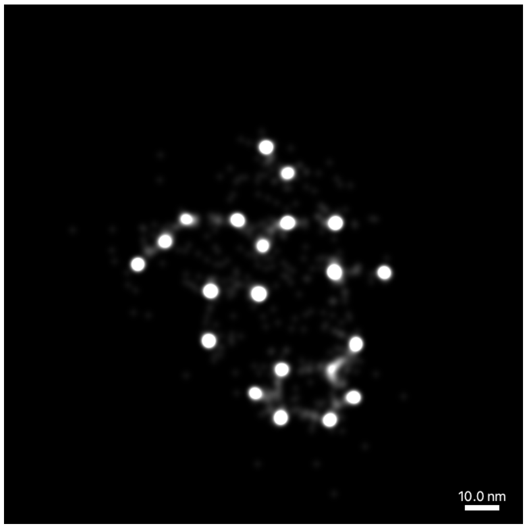
(c) Prediction from 4x – one pixel blur



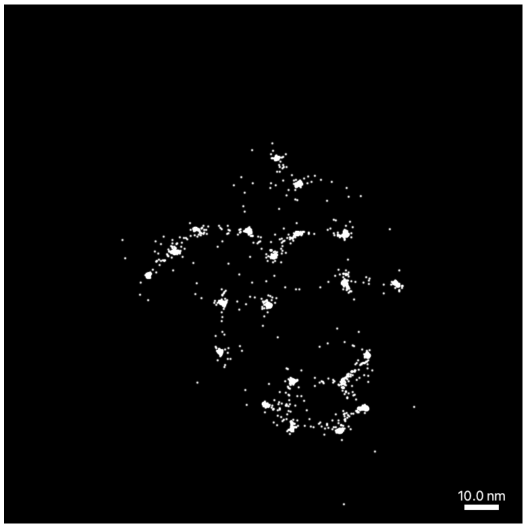
(d) Prediction from 4x – global localization precision



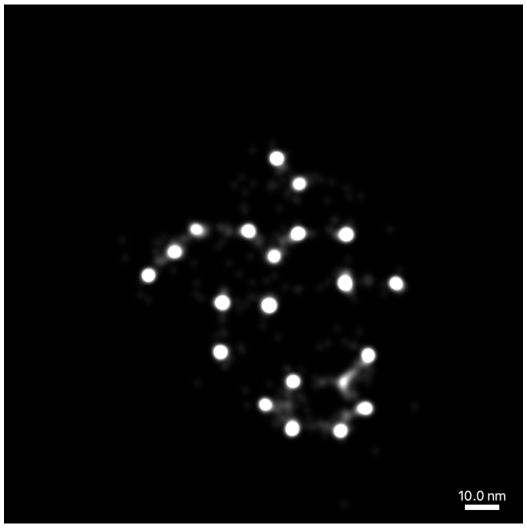
(e) Prediction from 8x – one pixel blur



(f) Prediction from 8x – global localization precision



(g) Prediction from 16x – one pixel blur



(h) Prediction from 16x – global localization precision

Figure 3.14: Visual comparison of our neural network output at different scale

Figures 3.15, 3.16, and 3.17 present a visual comparison of three different approaches: our neural network-based approach, the Picasso [11] MLE based approach, and the ThunderSTORM [12] MLE based approach. From the visual results, it is evident that our neural network-based approach outperforms both Picasso and ThunderSTORM.

For training our neural network, we utilized simulated origami data. To ensure unbiased evaluation, we employed a distinct simulated dataset specifically for testing, thus avoiding any data leakage between training and inference. This same testing dataset was also used for extracting emitters using the Picasso and ThunderSTORM methods.

We further conducted a comparison using the publicly available microtubules dataset (MT0.N1.HD) [13], which was not generated by us. However, the improvement observed in this dataset was not as significant as in the previous case (as shown in Figures 3.15, 3.16, and 3.17). In our neural network training, the emphasis was on capturing the pattern present in the origami data. As neural networks are adept at recognizing patterns, the substantial improvement in visualizing the origami data is understandable. The dataset of microtubules, on the other hand, possesses a distinct shape that differs from the grid-like structure the neural network was trained on. Consequently, the performance on this dataset was comparatively poorer. In the case of Picasso and ThunderSTORM, the structure of the final output is not a significant factor, which explains the consistent performance of these methods in both scenarios.

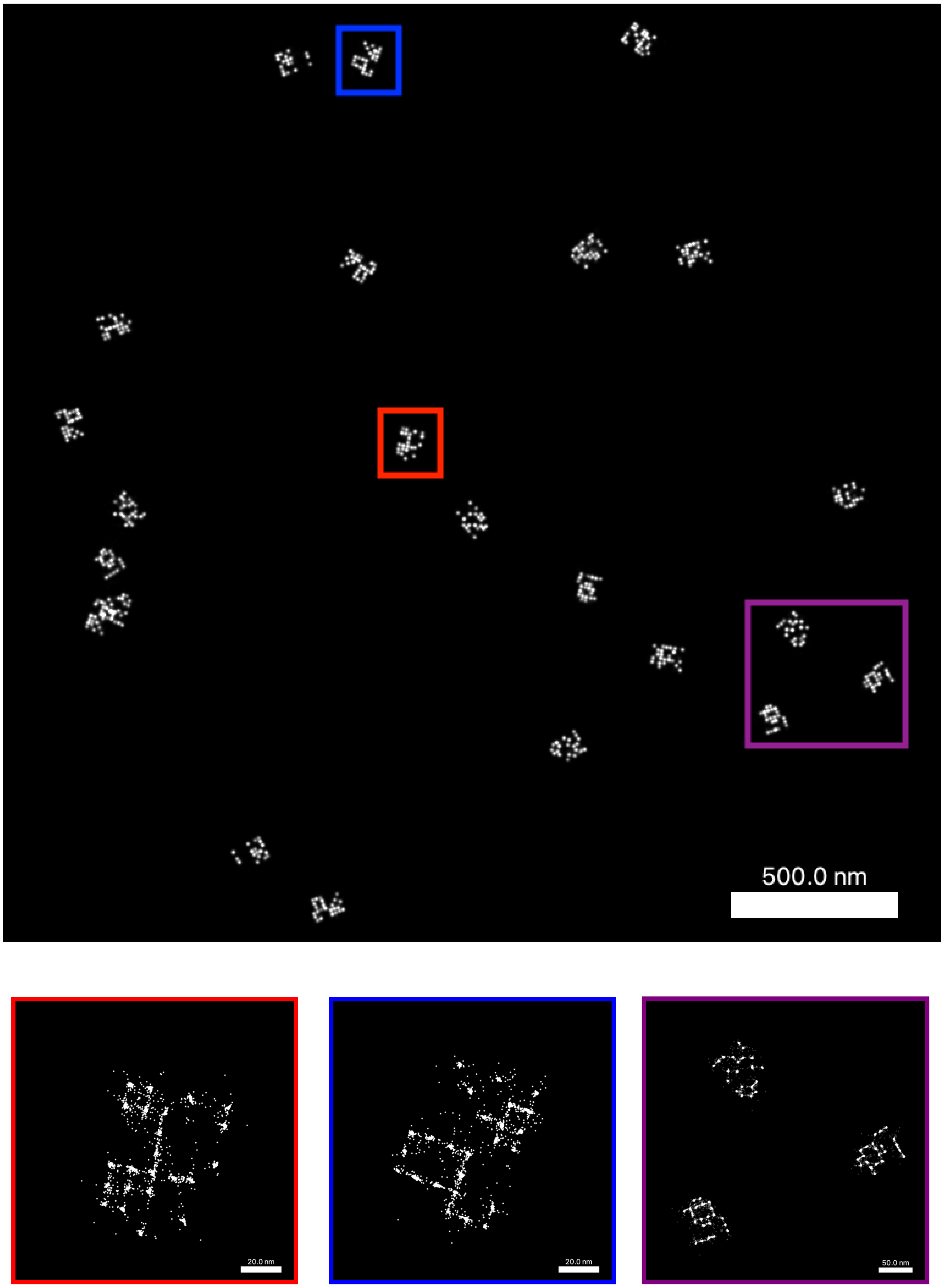
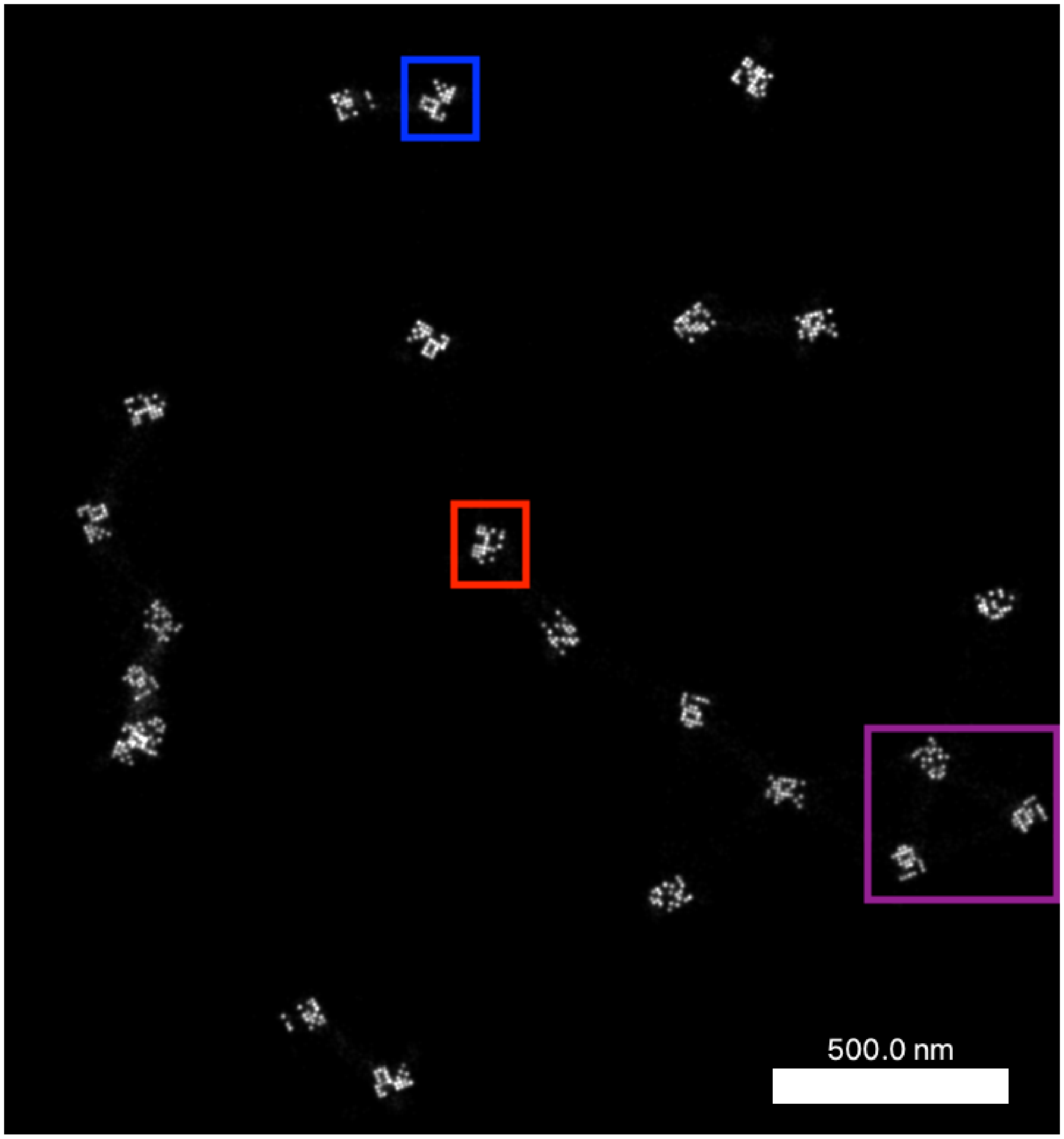


Figure 3.15: Top figure shows all the emitters that were localized using our neural network based method without any blurring effect. Focused regions (bottom row) show emitters using one-pixel blur to better visualize the origami.



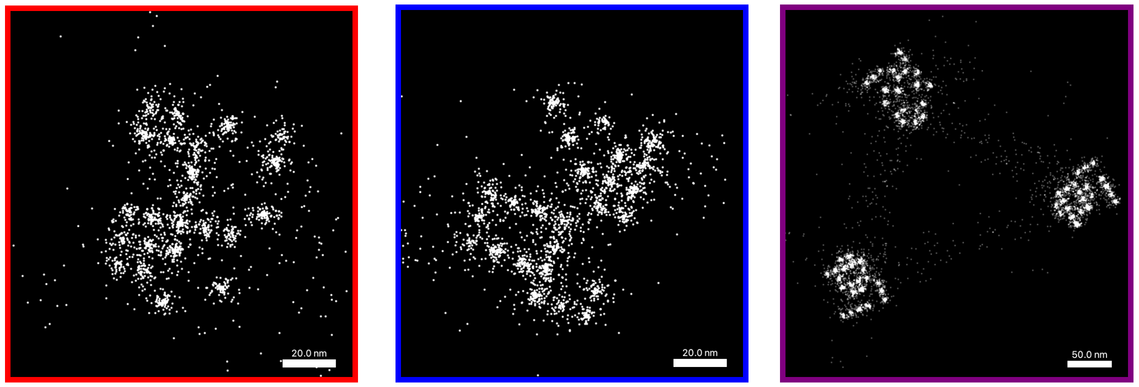
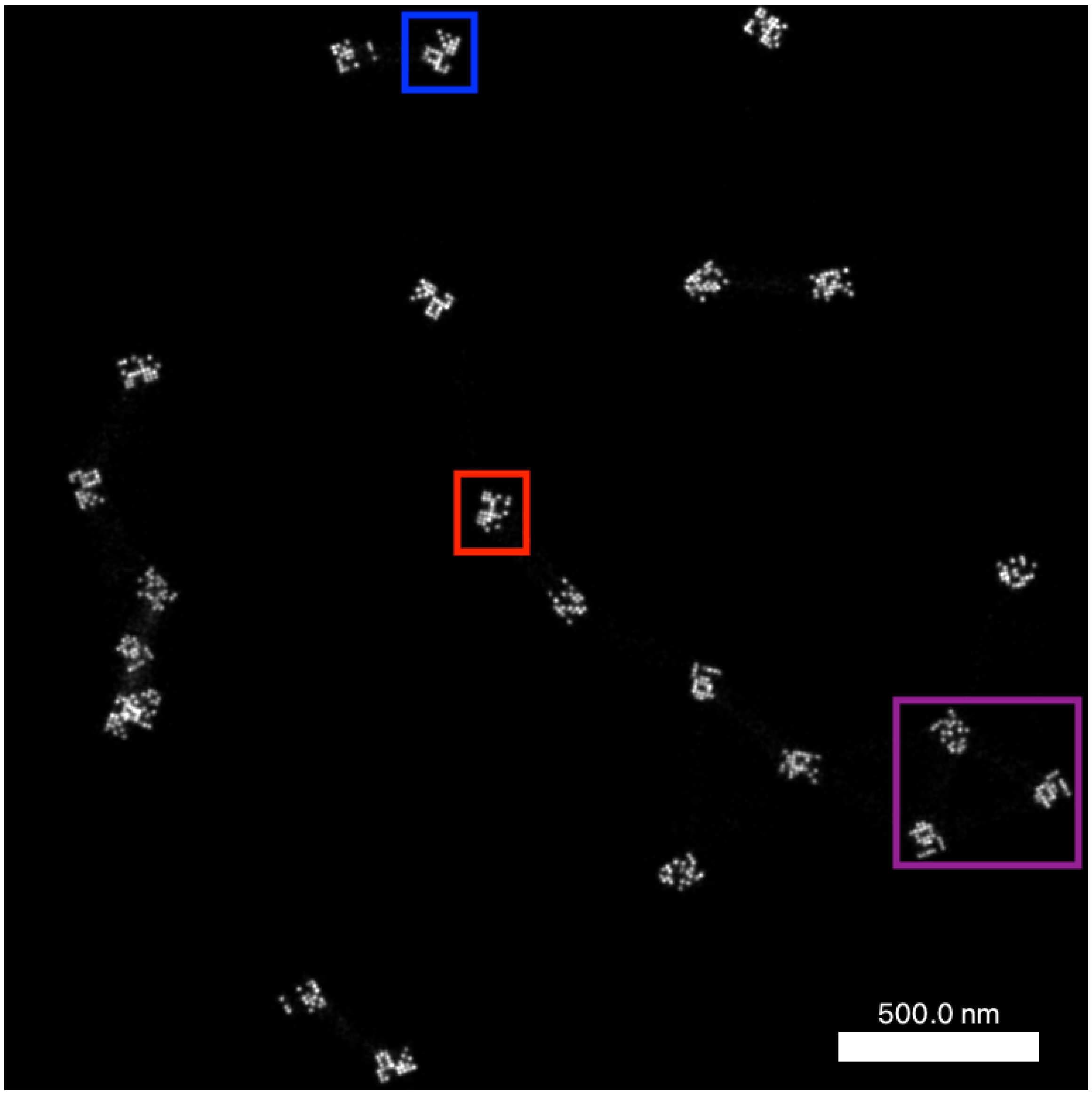


Figure 3.16: Top figure shows all the emitters that were localized using Picasso [11] without any blurring effect. Focused regions show emitters using one-pixel blur to better visualize the origami.



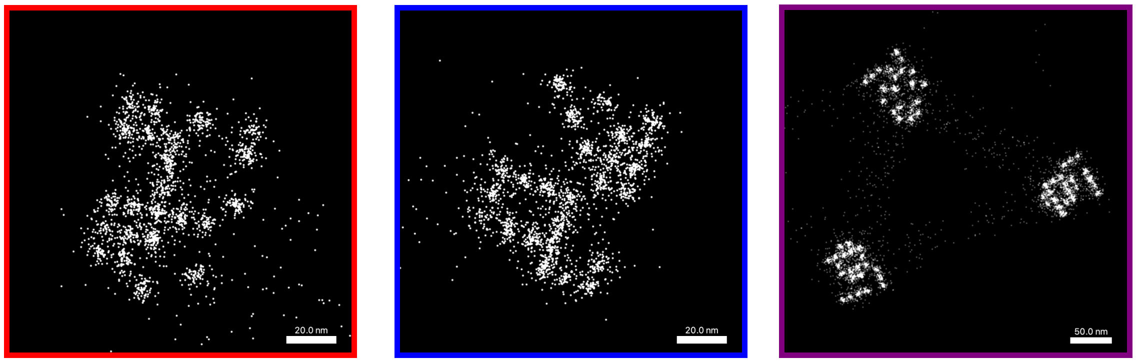


Figure 3.17: Top figure shows all the emitters that were localized using ThunderSTORM [12]. Focused regions (bottom row) show emitters using one-pixel blur to better visualize the origami.

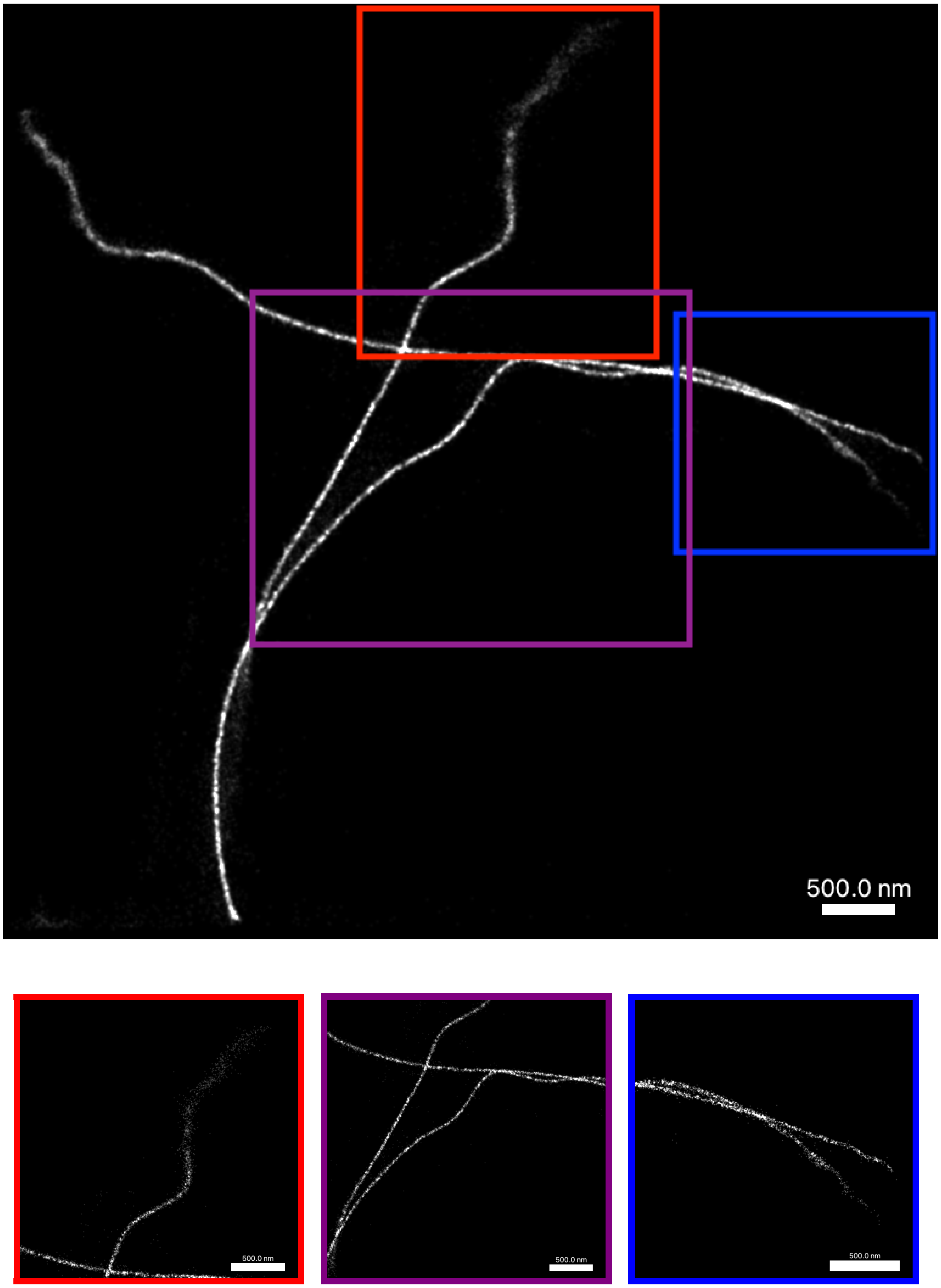
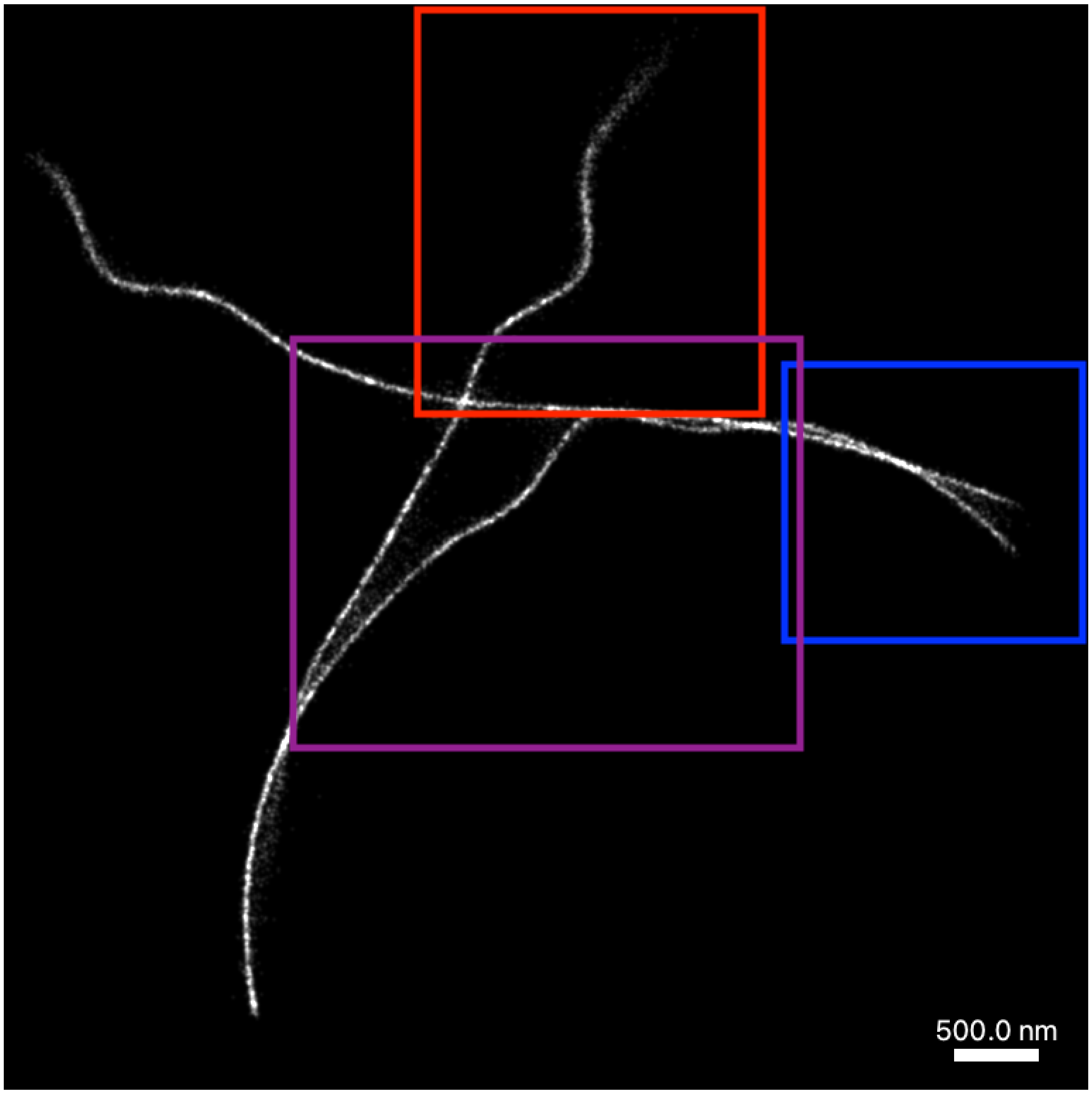


Figure 3.18: Figure shows all emitter's point extracted from dataset MT0.N1.HD [13] using our neural network based method



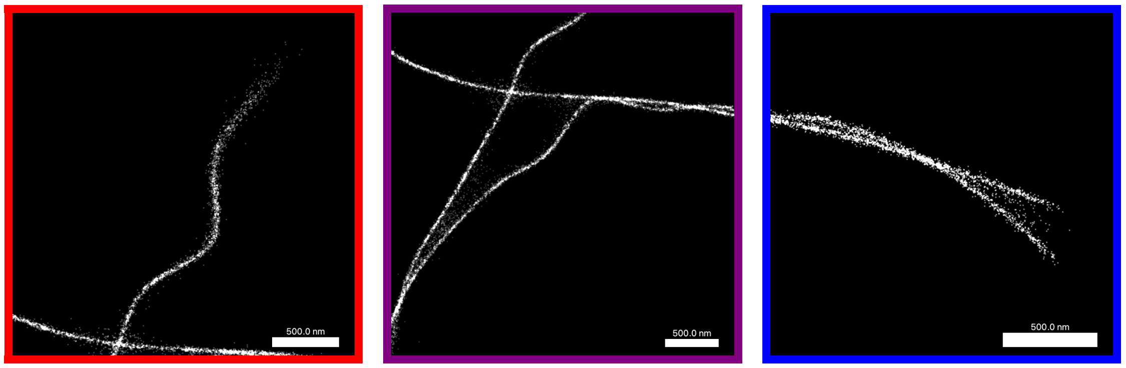


Figure 3.19: Figure shows all emitter point extracted from dataset MT0.N1.HD [13] using picasso [11]

Figure 3.20: Figure shows all emitter point extracted from dataset MT0.N1.HD [13] using thunderSTORM [12]

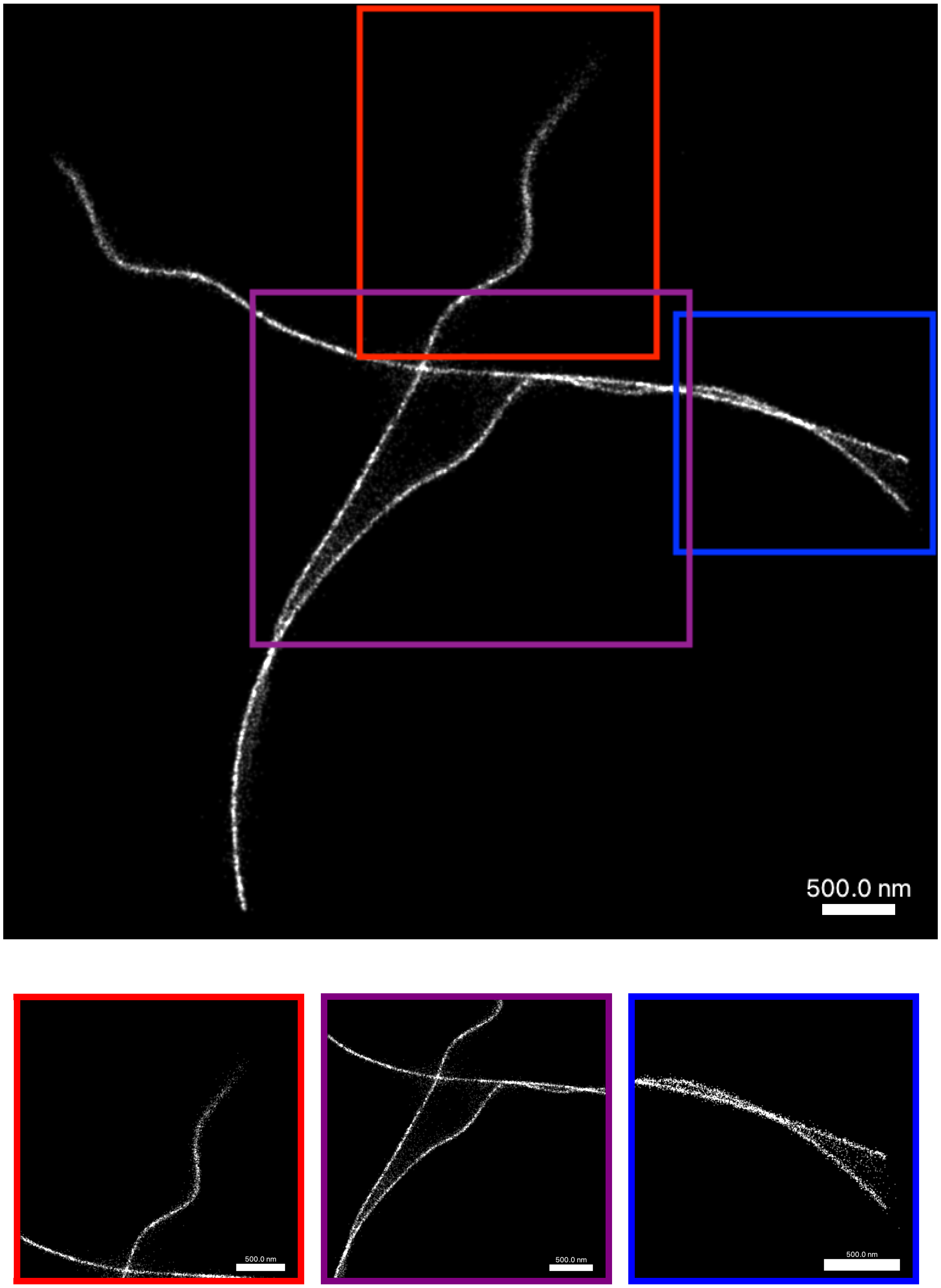


Figure 3.20: Figure shows all emitter point extracted from dataset MT0.N1.HD [13] using thunderSTORM [12]

CHAPTER 4:

ERROR CORRECTION FOR DNAM

4.1 The Initial Version of Error Correction

The key design features of dNAM, crucial for ensuring error-free data recovery, lie in the implementation of our error-correcting algorithms. The detection of individual DNA molecules using DNA-PAINT often encounters limitations such as incomplete staple strand incorporation, defective imager strands, fluorophore bleaching, and background fluorescence [132]. Although averaging multiple images of identical structures can improve the signal-to-noise ratio [132], this approach negatively impacts read speed and information density. To address these challenges, we have developed specific information encoding and decoding algorithms for dNAM, employing fountain codes along with a custom bi-level, parity-based, and orientation-invariant error detection scheme.

Fountain codes, originally introduced by Luby [133], offer an effective means of transmitting data over noisy channels. The encoding process involves dividing a data file into smaller units known as droplets, which are then randomly sent to the receiver. Importantly, droplets can be read in any order and still be decoded to recover the original file [60], as long as a sufficient number of droplets are transmitted to ensure complete reception. In our implementation, we assign each droplet to a single

DNA origami and include additional bits of information for error correction purposes. This ensures the successful recovery of individual droplets from their respective DNA origami, even in the presence of high levels of noise. By integrating EC and fountain codes, we significantly increase the probability of fully recovering the message while minimizing the number of observed origami required.

By employing these error-correcting algorithms and fountain codes, we have successfully addressed the challenges associated with error-free data recovery in dNAM. This achievement not only improves the robustness of the system but also enhances the overall efficiency by reducing the number of observed origami.

4.1.1 Encoding Algorithm

The encoding algorithm utilized a multi-layer error correction scheme to encode message data bits, along with index, orientation, and error correction bits onto multiple origami structures (Figure 4.5).

At the message level, the algorithm employed a fountain code to encode the data. Let m represent a message string consisting of a sequence of n bits. The fountain code algorithm initially divides m into k equally sized and non-overlapping substrings known as a segment, denoted as $s_1, s_2, \ldots, s_k$, where the concatenation $s_1 s_2 \ldots s_k = m$. These substrings were then systematically combined, using the binary XOR operation, to form multiple data blocks known as droplets. The number of segments d used to form each droplet was typically drawn from a distribution based on the Soliton distribution:

$$p(1) = 1/k \tag{4.1}$$

For our experiments, we divided the message “Data is in our DNA!” into ten segments, each consisting of 16 bits. These segments were combined in different combinations using the fountain code algorithm to generate 15 droplets. Although the theoretical minimum number of 16-bit droplets required to decode the message is ten, the redundancy provided by the additional droplets ensured that the message could be recovered even in cases involving the loss of one droplet, and in some cases, up to five droplets (Figure 4.5).

After generating the droplets using fountain codes, the encoding algorithm proceeded to encode each droplet onto fifteen 6x8 matrices. Sequentially, index and orientation marker bits were added, followed by the computation and addition of checksum bits, and finally, the inclusion of parity bits (Figure 4.5). These matrices were then used to construct 15 origami structures, establishing a one-to-one mapping between the matrices and the origami's data domains. The computation of parity and checksum bits was based on a predefined mapping scheme that outlined the relationship between all the bits in the origami. Initially, we employed a hand-picked mapping scheme that maintained two essential properties. However, for the 3dNAM error correction algorithm, this step was automated (see Section 4.3.1).

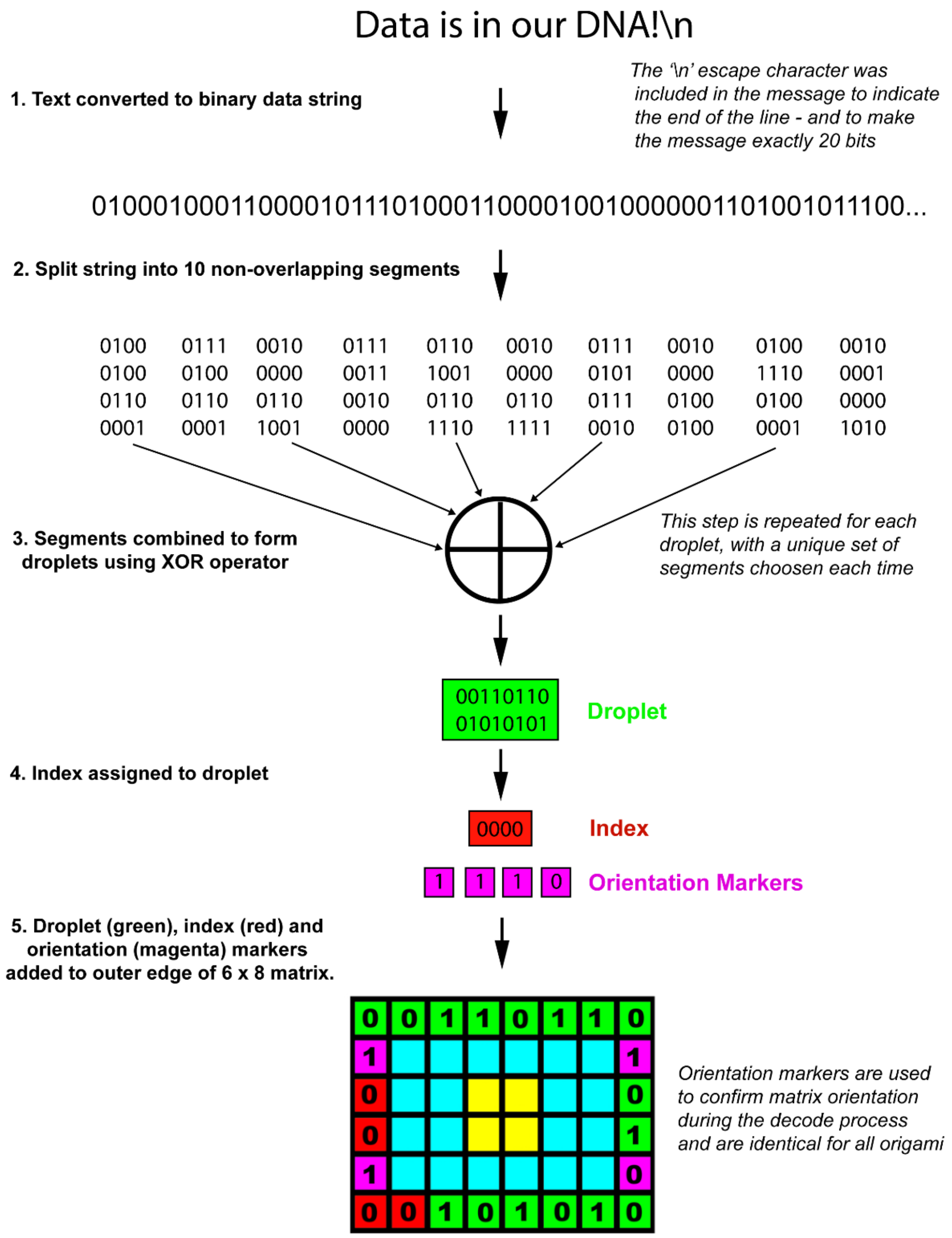
Mapping Requirements

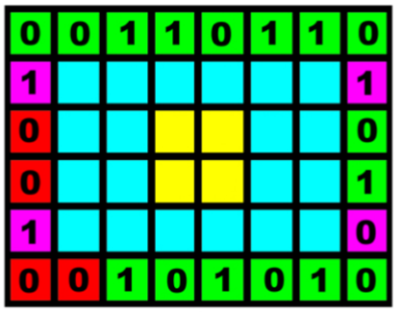
Rotation Invariant: The mapping relationship between each parity and data bit needed to be rotation invariant. This means that the relationship should remain the same regardless of the direction or orientation in which the origami is read. We achieved this property by mirroring the data points based on the center axis in each direction.
Even Parity and Data Coverage: To evenly distribute the mapping, we ensured that each parity contained a predefined number of data bits, and each data bit was present in a predefined number of parity bits. We used two counters to track these relation-

ships and selected available data bits for even distribution.

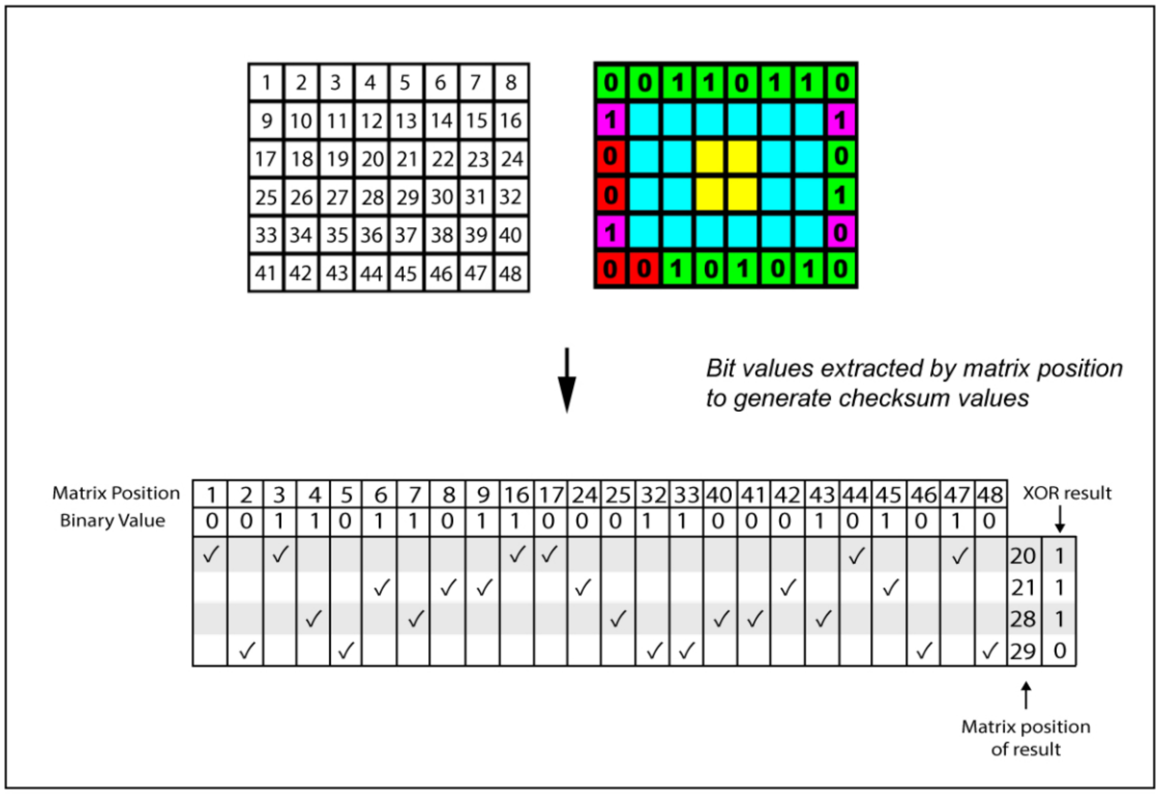
Figure 4.5 illustrates how droplet information was encoded into each origami structure. Each origami consisted of 16 bits of droplet data (colored in green), four indexing bits (colored in red), twenty parity bits (colored in blue), four checksum bits (colored in yellow), and four orientation bits (colored in magenta). The index bit is utilized to determine which segments are present in a specific droplet. The information from the non-parity bits is distributed into the parity bits, and these distributed bits are later used to recover missing information through error correction algorithms. After fixing errors using the parity bits, the checksum bits are employed to verify the integrity of each bit, excluding the parity bits. The orientation bits are then used to correctly orient the origami after the verification from the checksum bits.

Notably, the arrangement of the data, orientation, and index bits relative to the corresponding parity and checksum bits remained invariant to rotation. This property allowed the error correction algorithm to perform error detection and recovery prior to determining the orientation, resulting in more robust data recovery (Figure 4.5).


Data is in our DNA!\n
1. Text converted to binary data string
The '\n' escape character was included in the message to indicate the end of the line - and to make the message exactly 20 bits
010001000110000101110100011000010010000001101001011100...
2. Split string into 10 non-overlapping segments
0100 0100 0110 0001
0111 0100 0110 0001
0010 0000 0110 1001
0111 0011 0010 0000
0110 1001 0110 1110
0010 0000 0110 1111
0111 0101 0111 0010
0010 0000 0100 0100
0100 1110 0100 0001
0010 0001 0000 1010
3. Segments combined to form droplets using XOR operator
This step is repeated for each droplet, with a unique set of segments choosen each time
00110110 01010101
Droplet
4. Index assigned to droplet
0000
Index
1 1 1 0
Orientation Markers
5. Droplet (green), index (red) and orientation (magenta) markers added to outer edge of 6 x 8 matrix.
0 0 1 1 0 1 1 0
1 1
0 0
0 1
1 0
0 0 1 0 1 0 1 0
Orientation markers are used to confirm matrix orientation during the decode process and are identical for all origami

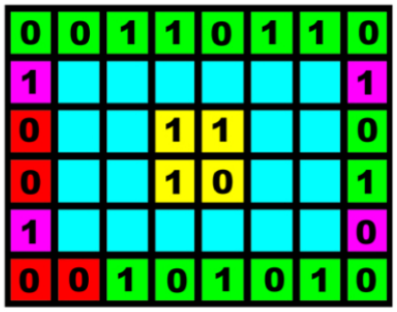


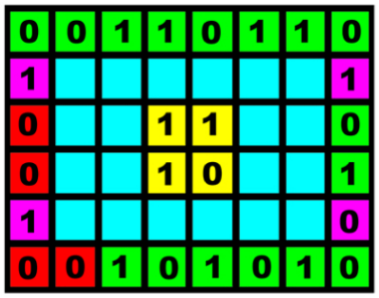
6. Checksum bits calculated from symetrically positioned matrix edge values


Bit values extracted by matrix position to generate checksum values
XOR result
Matrix position of result

Matrix Position	1	2	3	4	5	6	7	8	9	16	17	24	25	32	33	40	41	42	43	44	45	46	47	48		
Binary Value	0	0	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	0	1	0	1	0		
	✓		✓							✓	✓									✓			✓		20	1
						✓		✓	✓			✓						✓			✓				21	1
				✓			✓						✓			✓	✓		✓						28	1
		✓			✓									✓	✓							✓		✓	29	0

7. Checksum bits added to center of matrix (yellow)





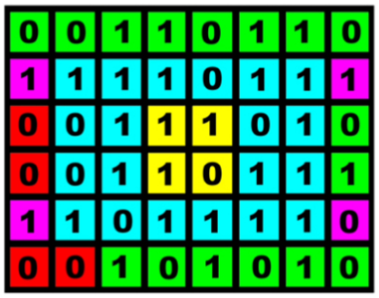
8. Parity bits calculated from symetrically positioned edge values and checksum

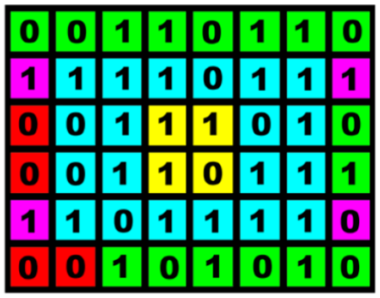
Matrix Position	1	8	41	48	2	7	42	47	43	46	3	6	4	5	44	45	9	16	33	40	17	24	25	32	20	21	28	29		XOR result
Binary Value	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0		
	✓				✓							✓				✓									✓				10	1
		✓				✓					✓				✓											✓			15	1
			✓				✓			✓				✓													✓		34	1
				✓				✓	✓				✓															✓	39	1
					✓								✓				✓					✓				✓			11	1
						✓								✓				✓			✓				✓				14	1
							✓								✓				✓					✓				✓	35	0
								✓								✓				✓			✓				✓		38	1
	✓								✓		✓			✓										✓					12	1
		✓								✓		✓	✓										✓						13	0
			✓						✓		✓					✓						✓							36	1
				✓						✓		✓			✓						✓								37	1
	✓												✓					✓	✓		✓						✓		26	0
		✓												✓			✓			✓		✓						✓	31	1
			✓												✓		✓			✓			✓		✓				18	0
				✓												✓		✓	✓					✓		✓			23	1
	✓					✓			✓										✓				✓						27	1
		✓			✓					✓										✓				✓					30	1
			✓					✓			✓						✓				✓								19	1
				✓			✓					✓						✓				✓							22	0

Matrix position of result

This is similar to step 6, however the same values are combined in multiple differnet XOR operations

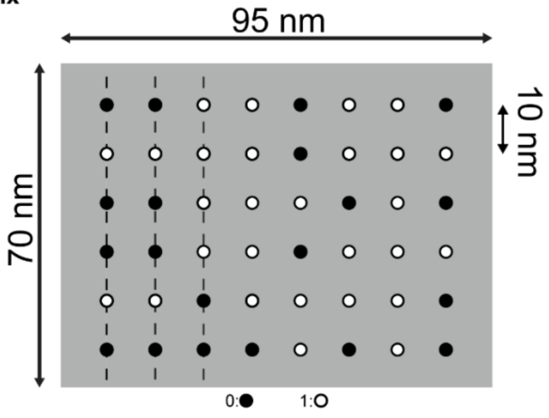
9. Parity bits added to remaining matrix positions (blue)



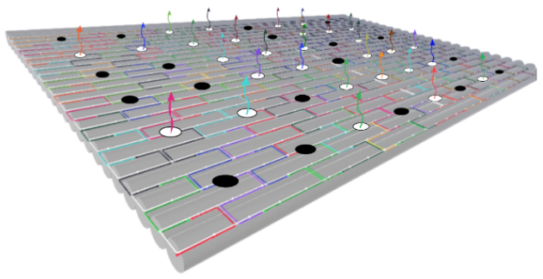


0	0	1	1	0	1	1	0
1	1	1	1	0	1	1	1
0	0	1	1	1	0	1	0
0	0	1	1	0	1	1	1
1	1	0	1	1	1	1	0
0	0	1	0	1	0	1	0

9. DNA-origami design created with the droplet, index, orientations markers, checksum bits, and parity bits encoded into the matrix



10. DNA-origami assembled from staple strands and scaffold in PCR thermocycler and purified by gel filtration



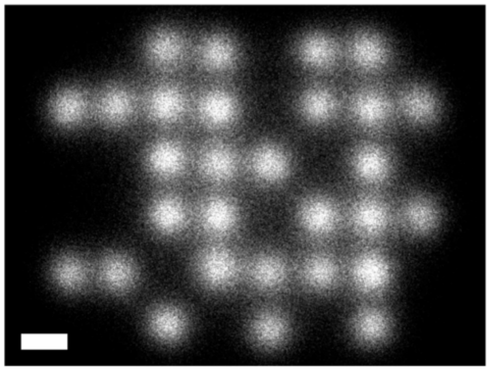
11. DNA-origami combined and stored at 4C

Fifteen differnt origami designs (one for each droplet) were used to encode the message ‘Data is in our DNA!\n’. Each origami was diluted to 0.33 nM for storge.

4°C storage

4°C storage

12. DNA-origami tested using DNA-PAINT



A small aliquot (1-2 ul) of the storage mix was diluted and deposited onto a microscope slide (using glow discharge) for DNA-PAINT imaging

An averaged image of origami-0 (the first of the fifteen designs) is shown here. Even though each of the individual single images from which the average is made is likely to be missing data from one or more data cells (due to data strands not correctly incorporating themselves into the origami structure, or insufficient imager strands binding to the data strand to get a 'read'), all the data cells are recorded correctly in the average. This suggests no systematic failure occured during the origami assembly. Similar averages were created for all fifteen origami designs, and all data cells in all origami were observed as expected.

Figure 4.5: The multi-page figure illustrates the twelve steps involved in encoding a text message using dNAM. The encoding process depicts the proof-of-principle experiment described in the main text, showing the design process for one of the 15 origami, as an example.

4.1.2 Decoding Algorithm

The decoding algorithm (Figure 4.6) utilized a multi-layer error correction/encoding scheme to recover the data in the presence of errors. The algorithm first works at the dNAM origami level (step 1, below), using the parity and checksum bits to identify and correct errors and recover the correct matrix. After recovery, the algorithm uses binary operations to recover the original data segments from the droplets (step 2, below).

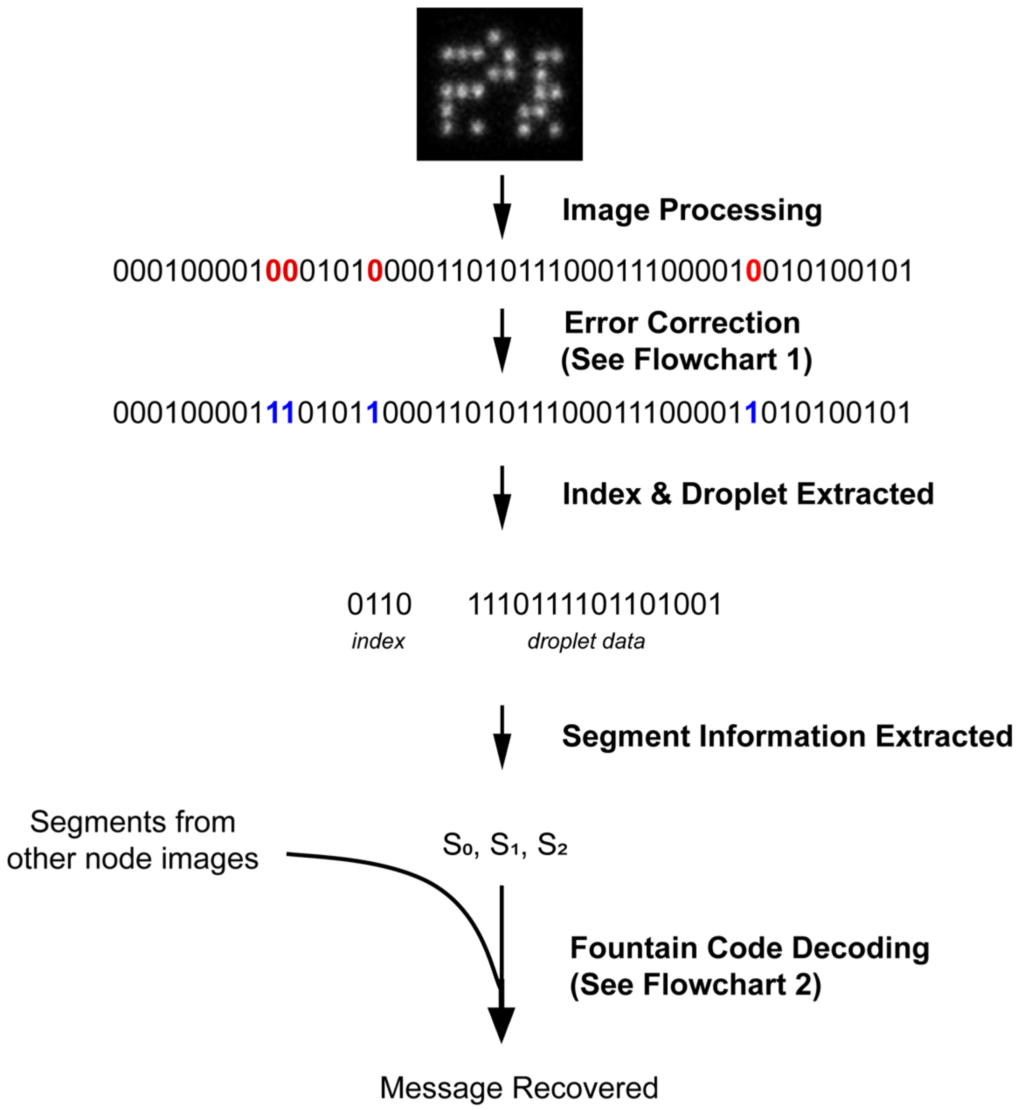


Figure 4.6: The main steps involved in decoding a message from dNAM are depicted. First, each individual origami captured in a DNA-PAINT recording is converted into a binary string (Image Processing). Next, errors in each binary string are detected and corrected if possible (Error Correction) using the algorithm described in figure 4.7 and index and droplet data extracted. Finally, segment information is retrieved from the droplets (segment information extracted) pooled with data from other origami and passed to the fountain code decoding algorithm shown in Figure 4.8, which reassembles the original file (Fountain Code Decoding).

Step 1–Error Correction

Given raw binary matrix data M for a single dNAM origami, the output from the localization data processing step, the matrix decoding algorithm determined which, if any, bits were associated with checksum and parity errors by calculating the bi-level matrix parity and checksum values, as described in Figure 4.5. Any discrepancies between the calculated parity and checksum values and the values recovered from the origami were noted, and a weight for each of the bits associated with the errant parity/checksum calculation was deduced. If no parity/checksum errors were detected for a particular matrix, then the data was assumed to be accurate, and the algorithm proceeded to extract the message data.

To determine the site(s) of likely errors, the decoding algorithm first determined a weight for every cell in M, beginning with data cells (the cells containing droplet, index, or orientation bits) and proceeding to parity and checksum cells. Let $P_{c_{i,j}}$ be the set of parity functions calculated over a given data cell $c_{i,j}$. Then for each data cell $c_{i,j}$:

$$x_{ij} = \sum_{f_{c_{pq}} \in P_{i,j}} \Big| c_{pq} - f_{c_{pq}}(M) \Big| \tag{4.2}$$

Where c_{pq} is the parity cell where the expected binary value of f is stored.

The weight for each parity cell c_{ij} was then calculated based on the number of non-zero weights greater than 1 for the data cells associated with it. More formally, let c_{ij} be a parity cell and $D_{c_{ij}}$ be the set of data cells used in the calculation of c_{ij}. Then the weight x_{ij} for each parity cell c_{ij} is:

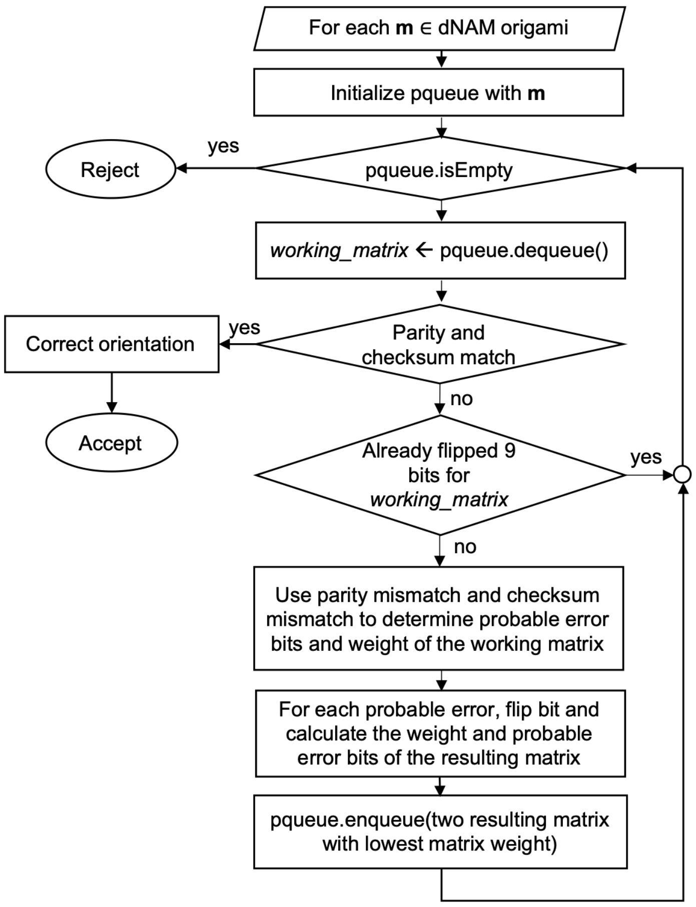
$$x_{ij} = \sum_{c_{pq} \in D_{c_{ij}} \oplus x_{pq} > 1} sgn\Big(x_{pq}\Big) \tag{4.3}$$

The higher the weight value, the higher the probability that the corresponding cell had an error.

An overall score for the matrix was then calculated by summing over all x_{ij} and normalizing by the sum of the correctly matched parity bits. This value was designated as the overall weight of the matrix. Higher values of this weight correspond to matrices with more errors.

$$\text{Overall matrix weight} = \frac{\sum_{i=0}^{6} \sum_{j=0}^{8} x_{ij}}{\#\ \text{number of matched parity}} \tag{4.4}$$

The algorithm then performed a greedy search to correct the errors using a priority queue ordered by the overall matrix weight (Figure 4.7). The algorithm began by iteratively altering each of the probable site errors and computing the overall matrix weight of the modified matrix for each, placing each potential bit flip into a priority queue where the flips that produced the lowest overall weights had the highest priority. At each step, the algorithm selected the bit flip associated with the highest priority in the queue and then repeated this process on the resulting matrix. This process was continued until the algorithm produced a matrix with no mismatches or until it reached the maximum number of allowed bit flips (9 for our simulation/experiment). If it reached the maximum number of flips, it returned to the queue to pursue the next highest priority path. If the algorithm found a matrix with no mismatches, it then checked the orientation bits and oriented the matrix accordingly. The droplet and index data were then extracted and passed to the next step. If the queue was emptied without finding a correct matrix, the algorithm terminated in failure.



For each **m** ∈ dNAM origami
Initialize pqueue with **m**
pqueue.isEmpty
yes
Reject
working_matrix ← pqueue.dequeue()
Parity and checksum match
yes
Correct orientation
Accept
no
Already flipped 9 bits for *working_matrix*
yes
no
Use parity mismatch and checksum mismatch to determine probable error bits and weight of the working matrix
For each probable error, flip bit and calculate the weight and probable error bits of the resulting matrix
pqueue.enqueue(two resulting matrix with lowest matrix weight)

Figure 4.7: A flowchart depicting the operations performed by the error correction algorithm for an individual origami is shown. A priority queue is initialized with an individual origami m (the *working_matrix*). Based on the parity and checksum bits mismatch, the algorithm deduces a set of probable errors and a matrix weight for the working matrix. The matrix weight is proportional to the number of errors, and the main goal of the algorithm is to reduce the matrix weight in a greedy fashion. To that end, each of the probable errors in the *working_matrix* is sequentially flipped, and a matrix weight calculated for every resulting matrix. The two resulting matrices with the lowest weights are enqueued. The algorithm then replaces the *working_matrix* with the recalculated matrix possessing the lowest matrix weight from the queue. If the current working matrix already has 9 bits flipped it is discarded and the next matrix in the queue used. The algorithm repeats these steps until the matrix weight equals zero, at this point the data in the origami is considered to have been error-corrected and is passed to the next stage of the decoding (Accept). If the priority queue is emptied before the matrix weight reaches zero, the origami data is considered unrecoverable and is removed from the analysis (Reject).

Step 2–Fountain Code Decoding

After the extraction of droplet and index data from multiple origami, the algorithm attempts to recover the full message (Figure 4.8). Once decoded, each droplet has one or multiple segments XORed within it. Using the recovered indexes, the algorithm determines the number and specific segments contained in each droplet. To decode the message, the algorithm maintains a priority queue of droplets based on the number of segments they contain (referred to as their degree), with the droplets having the lowest degree assigned the highest priority. The algorithm iterates through the queue, removing the droplet with the lowest degree, attempting to employ it to reduce the degree of the remaining droplets through XOR operations, and subsequently re-queuing the resulting droplets. Upon encountering a droplet with a 'degree one', it stores it as a segment for the final message. If all segments are successfully recovered, the algorithm terminates its execution.

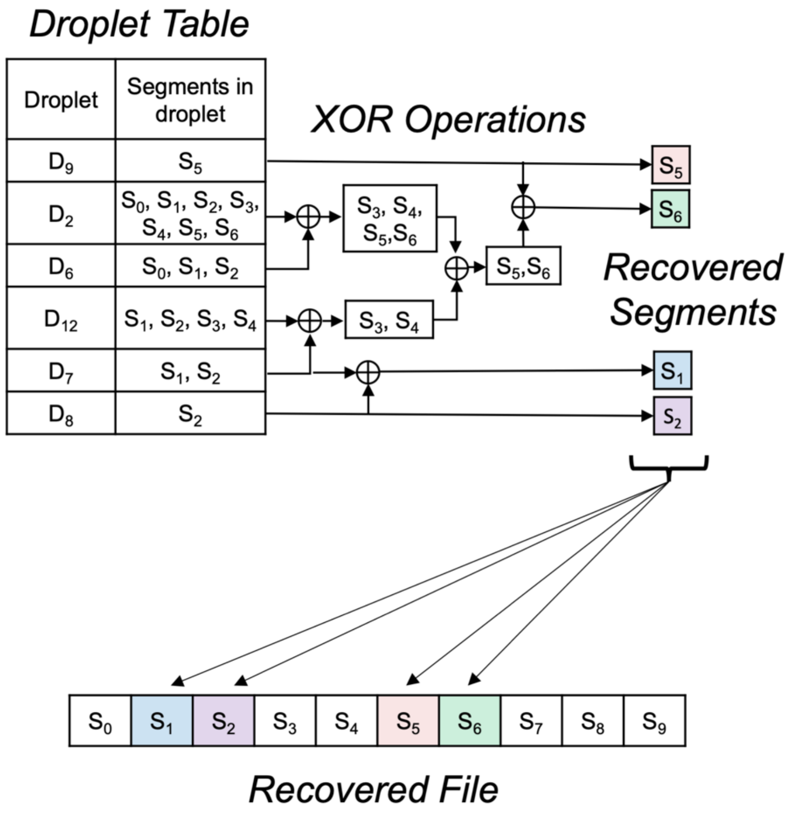


Figure 4.8: The flowchart depicts the operations performed by the fountain decoding algorithm to recover file segment data from droplet data. The recovered origami are stored in the Droplet Table. The data in single degree droplets(*i.e.* D8, D9) are used directly to reconstruct the file (Recovered File). To extract additional individual segment data from multi-segment droplets, the decoding algorithm performs a series of XOR operations. The index information allows the algorithm to determine both the degree of the droplet and which segments of the file that the droplet encodes. Taking the case of D2, a series of XOR operations must be performed in order to retrieve additional segment data from it. The decoding algorithm may XOR a multi-degree droplet with another droplet if the other droplet's segment(s) are a proper subset of the multi-degree droplet. For example, the segments contained in D6 are a proper subset of those in D2. After XORing D2 and D6, a new droplet is generated containing segments S5 and S6, which ultimately leads to the algorithm extracting the data for S6. This process is repeated in a greedy fashion until the algorithm retrieves all of the file's segment data (Recovered File), or it runs out of options for XORing droplets (in which case the entire file cannot be successfully recovered). For simplicity, only six of the 15 possible droplets are shown, with the resulting recovered segments depicted in colored boxes (Recovered Segments).

4.1.3 Results

In-silico Simulation

To test the robustness of our encoding and decoding algorithms, as well as its ability to recover from errors, origami data were simulated with randomly generated messages and errors. First, random binary messages of size m were created (for $m = 160$ to 12,800 bits, at 320-bit intervals). These messages were then divided into m/b equally sized segments, where b is the number of data bits to be encoded onto an individual origami. For fixed-size origami, larger messages necessitated a smaller b, as more bits had to be dedicated to the index. In these cases, b varied between eight (for $m = 12,800$) and twelve (for $m = 160$). After determining message segments, droplets were formed using the fountain code algorithm and encoded onto origami, along with the corresponding index, orientation, and error-correcting bits. Ten in silico copies of each unique origami were created, and 0–9 bits were flipped at random to introduce errors. The origami was decoded as described above.

As shown in (Figure 4.9a), the number of origami required to encode a message of length n increases roughly at a linear rate up to $n = 5000$ bytes of data. Larger message sizes require more bits to be devoted to indexing, decreasing the number of available data bits per origami—creating a practical limit of 64 kB of data for the prototype described in this work. This limit can be increased by increasing the number of bits per origami. To determine the ability of the decoding and error correction algorithm to recover information in the presence of increasing error rates, in-silico origami that encoded randomly generated data were subjected to increasing bit error rates. The decoding algorithm robustly recovers the entire message for all tested message sizes when the average number of errors per origami is less than 7.4 (Figure 4.9b). At 7.4 errors per origami, the message recovery rate drops to 97.5%, and as expected decreases rapidly with higher error rates (55% recovery at 8.2 errors per origami, and 7.5% at 9 errors per origami). An important feature of our algorithm is that the origami recovery rate can be low (as low as 63% in these experiments) and still recover the entire message 100% of the time.

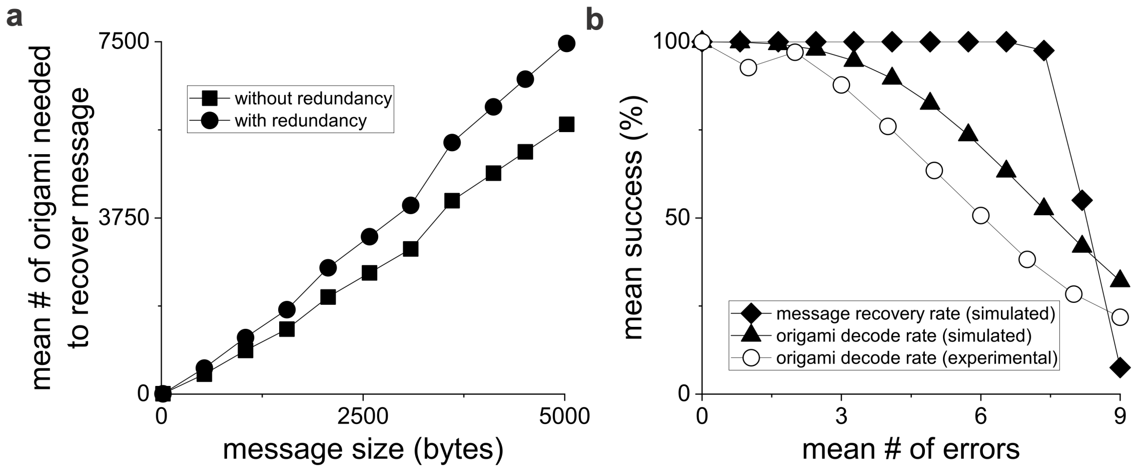


Figure 4.9: Simulations were performed to determine the theoretical success rates for correctly decoding individual dNAM origami and recovering encoded messages. In (a), the mean number of dNAM origami needed to successfully recover messages of increasing length with (circles) or without (squares) redundant bits are plotted. In (b), the mean success for recovering both individual origami (triangles) and the entire message (diamonds) are plotted against the mean number of errors per origami (errors were randomly generated for simulated data). Simulation recovery rates are averages of all message sizes tested (160 to 12,800 bits). For comparison, the mean success rate for experimental data is also plotted (open circles). For experimental data, the mean success was estimated by comparing the decode algorithm's results with that of the template-matching algorithm. All simulations were repeated 40 times. Experimental data were derived from 3 independent DNA-PAINT recordings.

In-vitro Experiment

As a proof of concept, we encoded the message ‘Data is in our DNA!’ into 15 distinct DNA origami nanostructures (Figure 4.10). Each origami was designed with a unique 6×8 data matrix that was generated by our encoding algorithm with data domains positioned 10 nm apart. For encoding purposes, the message was converted to binary code (ASCII) and then segmented into 15 overlapping data droplets that were each 16 bits. Inspired in part by digital encoding formats like QR-codes, the 48 addressable sites on each origami were used to encode one of the 16-bit data droplets, as well as information used to ensure the recovery of each data droplet. Specifically, each origami was designed to contain a 4-bit binary index (0000–1110), twenty bits for parity checks, four bits for checksums, and four bits allocated as orientation markers (Figure 4.10). To fully recover the encoded message, we then synthesized each origami separately and deposited an approximately equal mixture of all 15 designs ($\sim$ 20 fmoles of total origami) onto a glass coverslip. The data domains were accessible for binding via fluorescently labeled imager probes because they faced the bulk solution and not the coverslip. High-resolution Atomic Force Microscopy (AFM) was used in tapping mode to confirm the structural integrity of the origami and the presence of the data domains. 40,000 frames from a single field of view were recorded using DNA-PAINT ($\sim$ 4500 origami identified in 2982 μm^2). The super-resolution images of the hybridized imager strands were then reconstructed from blinking events identified in the recording to map the positions of the data domains on each origami. Using a custom localization processing algorithm, the signals were translated to a 6×8 grid and converted back to a 48-bit binary string—which was passed to the decoding algorithm for error correction, droplet recovery, and message reconstruction. The process enabled successful recovery of the dNAM encoded message from a single super-resolution recording.

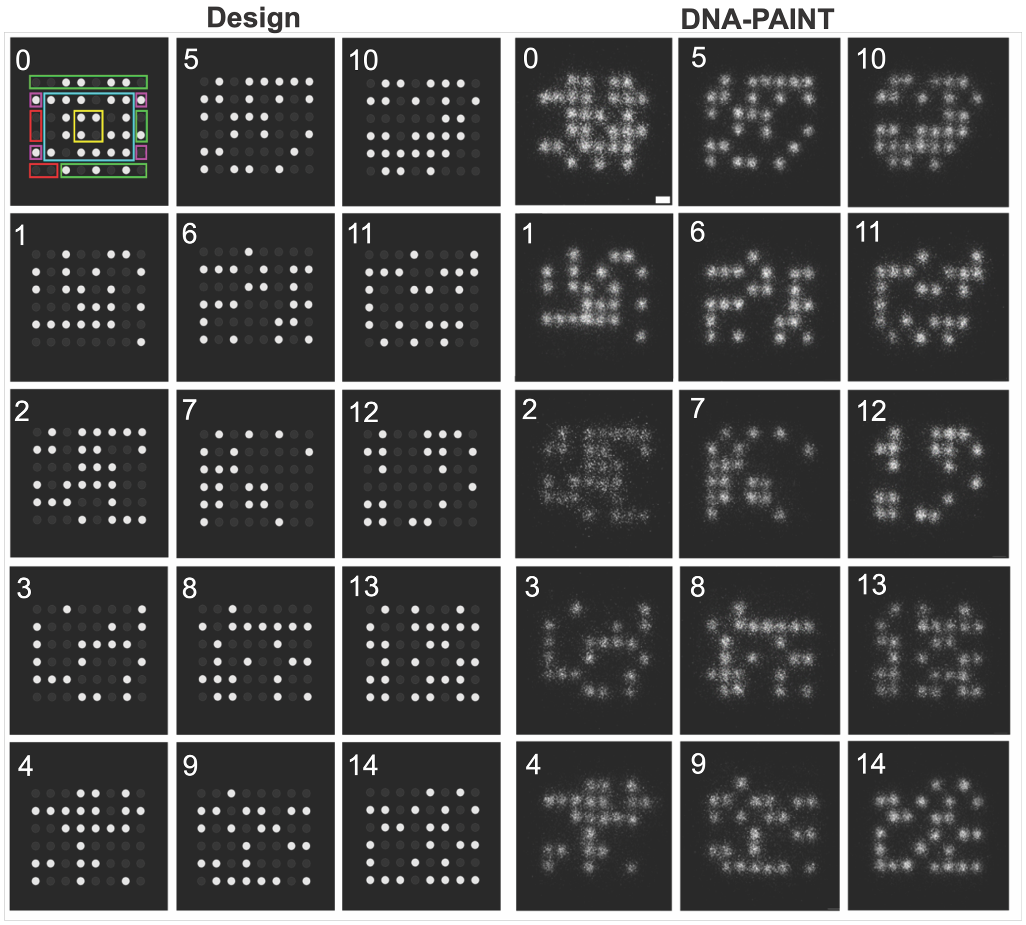


Figure 4.10: dNAM origami from a DNA-PAINT recording were identified and classified by aligning and template matching them with the 15 design co (Design) in which all potential docking sites are shown. Filled circles indicate sites encoded '0' (dark gray) or '1' (white). Colored boxes indicate the regions of the matrixes used for the droplet (green), parity (blue), checksum (yellow), index (red), and orientation (magenta). For clarity, only the first design image includes the colored matrix sites. Averaged images of 4560 randomly selected origami, grouped by index, are depicted (DNA-PAINT). Scale bar, 10 nm.

Given the observed frequency of missing data points, we then used a random sampling approach to determine the number of origami needed to decode the 'Data is in our DNA!' message under our experimental conditions. We started with all the decoded binary output strings that were obtained from the single-field-of-view recordings and took random subsamples of 50–3000 binary strings. We passed each random subsample of strings through the decoding algorithm and determined the number of droplets that were recovered (Figure 4.11). Based on the algorithmic settings used in the experiment, we found that only 750 successfully decoded origami were needed to recover the message with near 100% probability. This number is largely driven by the presence of origami in our sample that were prone to high error rates and thus rarely decoded correctly (*i.e.* origami index 2).

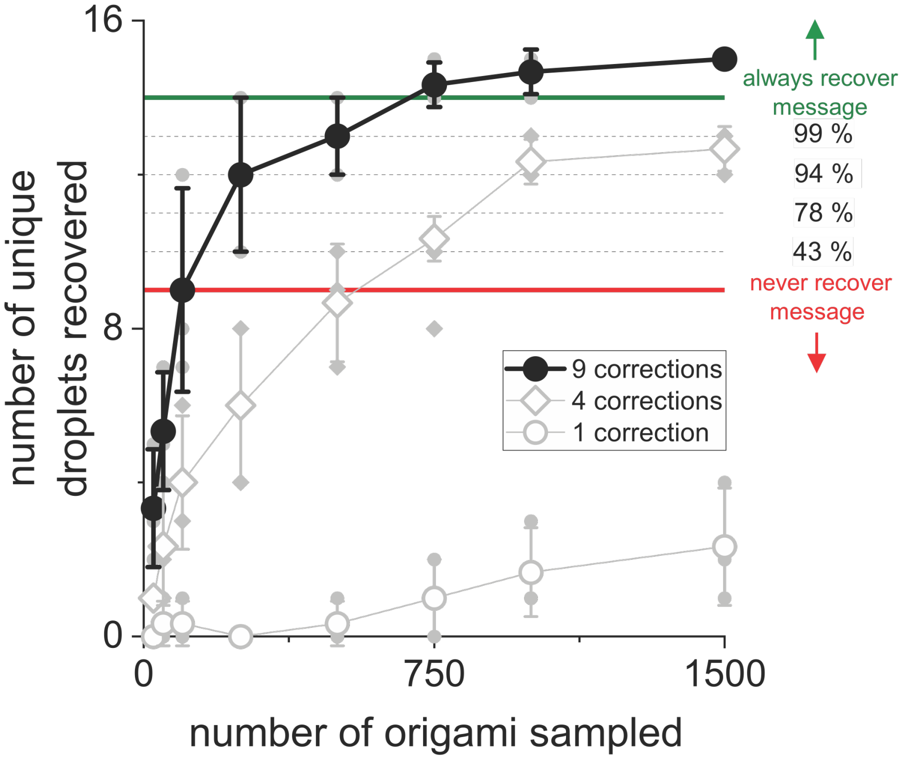
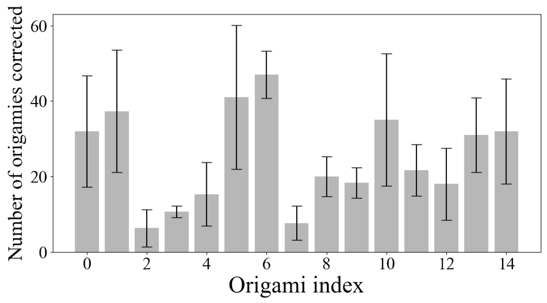


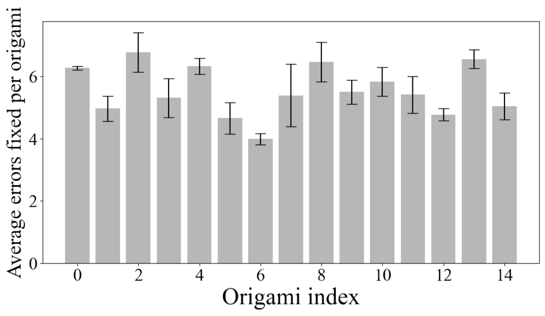
Figure 4.11: The mean number of unique dNAM origami correctly decoded for randomly selected subsamples of decoded binary strings are shown. The analysis was broken out by the number of errors corrected for each origami, three examples are plotted (1, 4, and 9). Black filled circles depict the mean results for nine error corrections, which is the 'maximum allowable number of errors' parameter used in the decoding algorithm for all other analysis reported here. The horizontal lines indicate the probability of recovering the message with different numbers of unique droplets. With fourteen or more droplets, the message should always be recovered (thick green line, and above indicates 100% chance of recovery) and with nine or fewer droplets the message will never be recovered (thick red line and below indicates 0% chance of recovery). Mean values for three experiments are shown. Error bars indicate ±SD. Individual data points are plotted behind as smaller gray symbols.

We have used three independent reads to recover our data. From each read, we randomly sub-sampled 2,000 origami and were able to recover the data independently.

Figure 4.12a shows the average number of origami detected from these reads. These are the origami that our error correction algorithm were able to recover. There were many more origami that were distorted too much that our error correction algorithm were unable to recover/identify them. Out of 6,000 origami from three read, we were able to recover 1120 origami with a recovery rate of 18.7%. Figure 4.12b shows the average number of errors that our algorithm corrects per origami.



(a) Number of origami corrected



(b) Average error fixed per origami

Figure 4.12: The results are summarized from three independent reads. From each read we sub-sampled 2,000 random origami. (a) Shows the number of average origami that were recovered from these reads. Read 3 exhibits better results compared to the other two reads. For that reason, the error bars are large. (b) Shows the number of average errors that our error correction algorithm fixed for an individual origami.

Strauss et al. [134] mentioned the center position(maximum of 95%) have higher strand incorporation rate than the edges(minimum of 48%). So, the error should be position specific. However, we have analyzed the error for each position in the origami(Figure 4.13). The error positions did not follow any specific pattern. The error happened mostly in a random position. Incorporated but inactive data domains play a greater role in producing errors than unincorporated staple strands [135]. Also, we only analyzed the origami that we were able to recover, which is about 18.7% of the total sample. As we were unable to recover/identify the rest of the origami, we could not know the true error position of those origami.

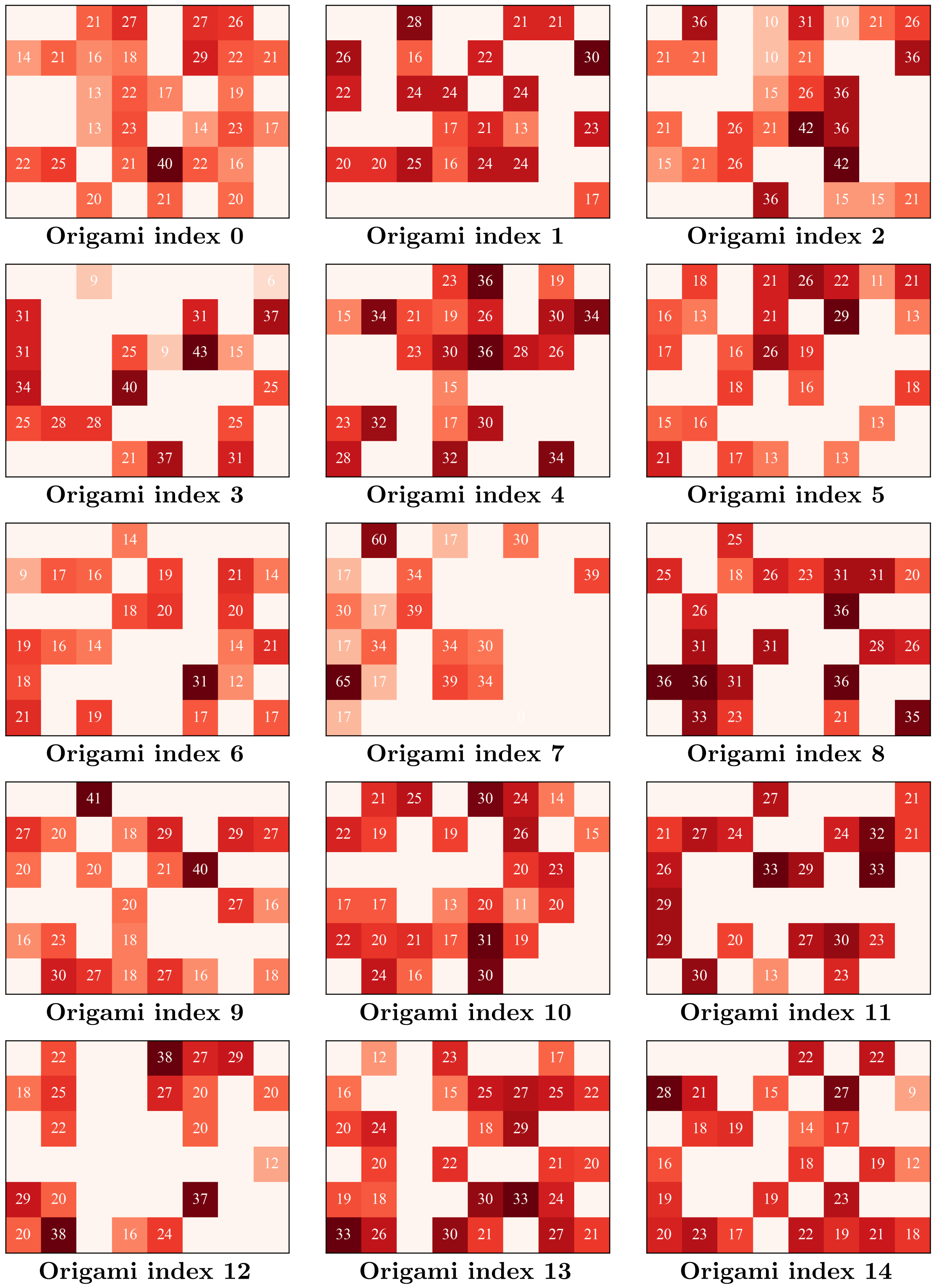


Figure 4.13: The heat map of error for each individual origami. The numerical number in each position indicates the percent of error that particular position have during recovery.

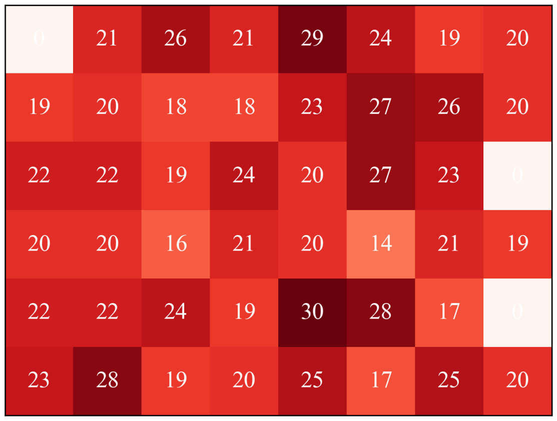


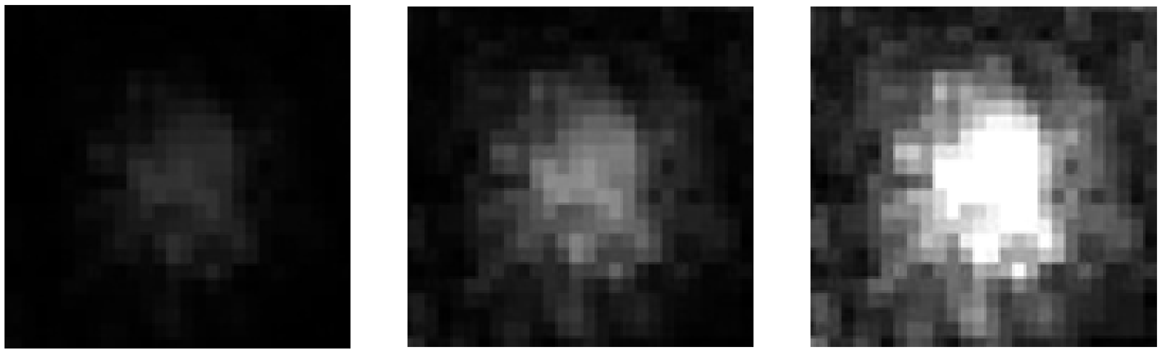
Figure 4.14: The figure presents a heat map depicting the summation of errors in all origami structures. Each numerical value in the heat map represents the percentage of errors observed at a specific position during the recovery process. The heat map indicates that there are likely no error-prone regions in the origami, as every data point is equally likely to have an error. However, it is essential to note that these errors were calculated based on the origami that were successfully recovered. Three data points show no errors because their bit value was always set to zero in each origami. One of these points is attributed to the fixed orientation marker, while the other two are related to the fact that the first character of ASCII is zero for all the letters present in the experimented sentence.

4.2 Adding Photon Intensity Information

dNAM uses a custom error correction algorithm to ensure error-free recovery of data. The algorithm can correct a maximum of nine bits of data out of 48 bits of data. It gets more computationally expensive as the number of errors gets high. The fountain code of the dNAM algorithm increases the computational cost even more. Further, since the fountain code algorithm XORs multiple data segments into each origami, it is almost impossible to provide random access to the data and makes data modification impossible. To modify or add only one bit of data, all the droplets must be recreated and converted into origami which is impractical. A better origami level error correction algorithm can remove the necessity of fountain code, reduce computational complexity and increase data density by reducing the bits dedicated to error correction.

4.2.1 Methodology

In the pre-processing step, we read the photon intensity for each emitter (data point) and convert that intensity value into a binary value of zero or one using an empirically derived threshold value $I_{threshold}$. The error correction algorithm takes the binary value during the decoding process. So, it does not know the level of confidence of each bit's actual value. Initially, the algorithm assumes all the bits have the same probability of errors. If we can provide a level of confidence for each bit, the algorithm can correct errors more intelligently. We can get the level of confidence for each bit from the photon intensity of those bits (Figure 4.15).



(a) $k_{ij} = .08$ **(b)** $k_{ij} = .6$ **(c)** $k_{ij} = .95$

Figure 4.15: Each of the figure represents the photon intensity of each data cell. And k_{ij} shows their normalized intensity. If we get a very low intensity at a particular point, then the probability of that cell being zero is higher and if the intensity is very high, then the probability of that cell being 1 is very high. So, it is less likely that these cell have any errors in them. However, for Fig 4.15b the intensity is closer to the threshold. So, the error chance of that cell is highly likely.

The algorithm uses a combination of parity bits and checksum bits. Both parity and checksum bits contain information about data, index, and rotation bits. During the decoding process, the algorithm calculates parity and checksum mismatches and assigns a cell weight x_{ij} (Equation 4.2 and 4.3) for each data point. This cell weight indicates the error probability for that data point. All the cell weights are summed together to get the overall origami weights (Equations 4.4). The algorithm performs a greedy search to reduce the overall origami weight. In the greedy search, the bit that will be altered next is determined by the origami weight. So we can inject the intensity information of the bit that will be modified during calculating the origami weight. In order to do that, we first have to calculate the error probability of each bit just based on their intensity. If a cell has very high or very low photon intensity, then our algorithm should be confident that cell's bit value is more likely to be accurate. However, if the photon intensity value of a cell is closer to the threshold value ($I_{threshold}$), then it's more likely that the binary value of that cell is most likely to have an error. So, the intensity multiplier I_{ij} needs to be designed in a way that it will have a higher value if the photon intensity is closer to the threshold and have a lower value if the photon intensity value is farther apart from the threshold.

$$\text{Intensity term} I_{ij} = \frac{1}{I_{threshold} + \epsilon} \tag{4.5}$$

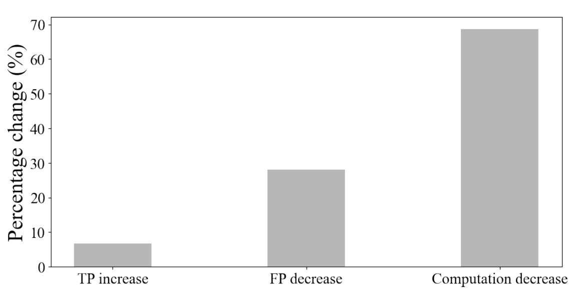
Here $I_{threshold}$ is the threshold intensity value used for converting photon value into binary. K_{ij} is the photon intensity for cell ij. ϵ is a small number to prevent the denominator from being zero. To inject intensity information we modified equation

4.4 as following:

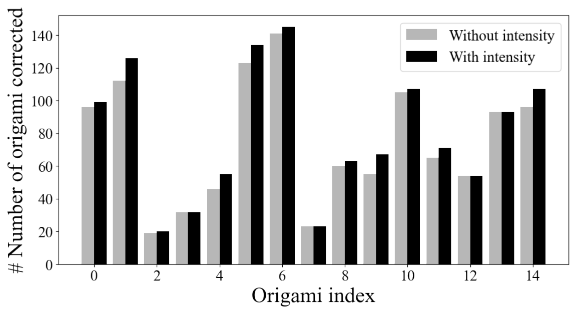
$$\text{Overall matrix weight} = \frac{\sum_{i=0}^{6}\sum_{j=0}^{8} x_{ij}}{\#\text{ number of matched parity} + I_{ij}} \tag{4.6}$$

4.2.2 Results

The inclusion of error intensity information in our error correction algorithm yields improvements in three distinct metrics (see Fig. 4.16a). First, it leads to an increase in the true positive rate, allowing us to correct more errors than before. By leveraging the intensity information obtained from wet lab experiment data, we were able to achieve a 6.76% increase in the origami recovery rate. A comparison of the number of recovered origami with and without incorporating intensity is presented in Figure 4.16b. Our approach involves incorporating the intensity of a cell into our heuristic for error probability, which is generally correlated but not always. As a result, not all origami samples demonstrate an improvement in the recovery rate.



(a) Metrics improvement



(b) Number of recovered origami

Figure 4.16: The figure demonstrates the impact of incorporating photon intensity information into our error correction algorithm. (a) depicts the percentage improvement observed in various metrics following the integration of intensity data. (b) provides a comparison between the number of origami samples that were successfully recovered with and without the use of intensity information.

Secondly, it decreases the false positive rate. We used fountain code at the top level of our custom error correction algorithm. As a result, we did not use the raw data. Instead, data was stored in the form of droplets (see Section 4.1.2. It is crucial to note that, during the decoding process of fountain code, a single false positive droplet has the potential to corrupt the entire decoded file. Consequently, it is imperative that all droplets utilized in decoding fountain code are true positives. Our wet lab experimental data indicate that we were able to decrease the false positive rate by 28.1%. This substantial improvement ultimately led to a higher file recovery rate.

Finally, our approach achieved a substantial reduction in computational steps by 68.7%. In the greedy search, our aim was to minimize the weight of the matrix (as per Equation 4.6). With each path we took, it was necessary to compute the weight of the current state of the matrix. The term "computational steps" refers to the total number of times we computed this matrix weight until the origami was successfully recovered. Computing the weight of the matrix was the most time-consuming aspect of the decoding algorithm. Thus, reducing the total number of computations of this step ultimately reduces the overall computational time required.

Based on our analysis of the experimental data, we have found that a correlation between photon intensity and error probability can exist; however, this relationship is not always straightforward and may be influenced by additional factors. Notwithstanding, the incorporation of photon intensity information into our error correction algorithm resulted in significant improvements in all metrics, thus enhancing the overall robustness of the algorithm. Nonetheless, it is essential to note that not all samples exhibited an improvement in the recovery rate, indicating that the correlation between photon intensity and error probability may not be consistently reliable.

Furthermore, we were unable to identify any discernible pattern in this behavior, highlighting the complexity of this relationship.

4.3 Error Correction for 3dNAM

4.3.1 3dNAM

3dNAM, built upon the framework of dNAM [21], introduces a novel approach for data storage by utilizing three-dimensional instead of two-dimensional structures (see figure 4.17). In 3dNAM, data is encoded based on the presence or absence of hybridization events at n specific binding sites on a data strand. By employing four unique dyes, each binding site can encode 2 bits of digital information, resulting in a total of 2n bits per strand and achieving impressive information densities exceeding 10 $Tbit/cm^2$.

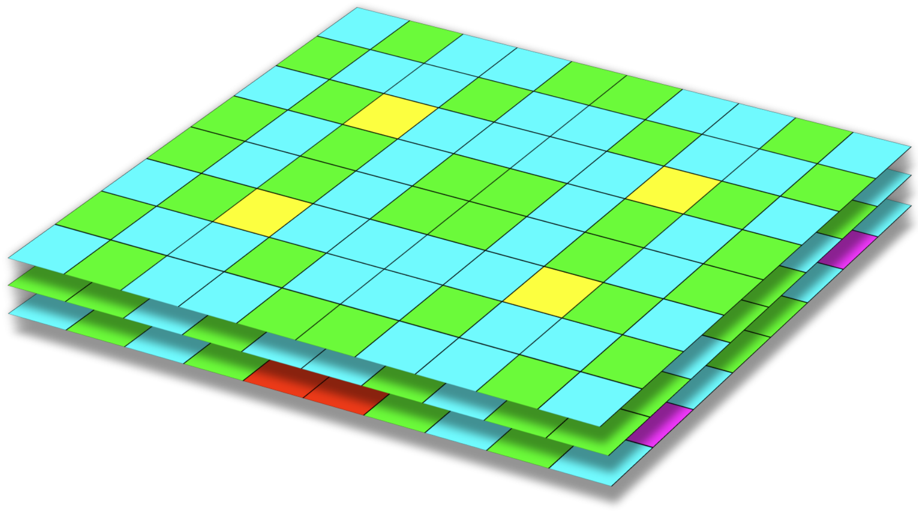


Figure 4.17: The figure illustrates an overview of 3dNAM, showcasing its three-dimensional nature. Unlike 2D structures, 3dNAM can expand in the z-direction as well.

4.3.2 Encoding

To store data in 3dNAM, an extension of our current error correction algorithm is required to accommodate 3-dimensional settings. We made several modifications to our error correction algorithm to incorporate this requirement.

Making fountain code optional

The primary purpose of using a fountain code was to recover the data even if some origami were missing. However, in our previous experiments, we successfully recovered all of the origami, rendering the use of the fountain code unnecessary. Furthermore, the fountain code introduced significant data overhead and made random access impossible. As a result, in our updated algorithm, users have the flexibility to adjust the number of parity cells instead of relying on fountain code. Increasing the number of parity bits enhances the robustness of each individual origami, but at the cost of reduced data capacity.

Data distribution

In the previous version of our error correction algorithm, we stored data and indexing bits around the edges while placing checksum and parity bits in the middle. This distribution was based on the anticipation that the edges would be more error-prone than the middle portion of the origami. However, our observations in the wet lab did not reveal such a pattern (see Figure 4.14). Consequently, in our updated version, we randomly distribute all the parity, data, and checksum bits throughout the origami. Nevertheless, the index and orientation bits always reside in the first layer since they are critical for correctly orienting and localizing the origami. As the first layer contains these vital information bits, we assign more parity bits to this layer to make it more robust.

Optimized mapping scheme

Previously, our error correction algorithm employed a hand-picked mapping scheme that only worked for 6x8 grid-shaped origami, ensuring rotation invariance and adequate coverage of data and parity. However, we have now developed an algorithm capable of generating a mapping scheme for origami of arbitrary shapes in 3-dimensional settings. The new mapping scheme generator operates as follows:

Ranking the Mapping Scheme: The process of ranking the mapping scheme begins by generating a random mapping scheme that fulfills two core requirements (explained in Section 4.1.1), with the number of parity bits and checksum bits as user-defined parameters. After generating the mapping scheme, a score is computed for it using Algorithm 1. A higher score indicates a better mapping scheme. This score is based on the idea that parity bits should contain the information of non-parity bits. If all the parity bits for a particular non-parity bit are located in the same area on the origami and that location becomes corrupted, the information would be lost as all the parity bits would be affected. However, if the parity bits for that specific non-parity bit are evenly distributed across the origami, this problem can be avoided. The algorithm generates 1000 random mapping schemes and selects the one with the highest ranking based on its score.

Algorithm 1 Mapping Score Calculator

```
1: function GetMappingScore(RevereseMapping)            ▷ Non-parity to parity mapping
2:     Score ← 0
3:     for NonParityBit in RevereseMapping.keys() do
4:         distanceX ← []
5:         distanceY ← []
6:         for point1, point2 in Combination(RevereseMapping[NonParityBit], 2) do
7:             distanceX.append(abs($point1[0] - point2[0]$))
8:             distanceY.append(abs($point1[1] - point2[1]$))
9:         end for
10:        localScore ← $\frac{\text{std(distanceX)+std(distanceY)}}{\text{len(RevereseMapping[NonParityBit])}}$            ▷ Get normalized sum of standard deviations along both axes
11:        Score += localScore
12:    end for
13:    Score /= len(RevereseMapping.keys())
14:    return Score
15: end function
```

The rest of our encoding process remains similar to our previous methodology. Each layer of 3dNAM functions as an isolated 2D origami during both encoding and decoding. Parity bits on a specific layer only contain XOR values from that layer, while sharing common information such as indexing and orientation bits, which are typically placed on the first layer. By placing these common bits in a single layer, we can save data space.

4.3.3 Decoding

Since each layer of the origami operates as isolated 2D origami, we can utilize our previous decoding algorithm with minor modifications. During our wet lab experiments, we observed that each origami contained multiple copies (approximately 2000). Based on this information, we have incorporated majority voting into our decoding algorithm. Majority voting considers all the decoded origami and extracts the data for each layer, performing the majority voting on each layer individually instead of the entire origami. This ensures that our algorithm does not need to recover the entire origami to utilize its data. Instead, even if our algorithm can recover only one layer, it can still use the data from that specific layer.

4.3.4 Simulation Results

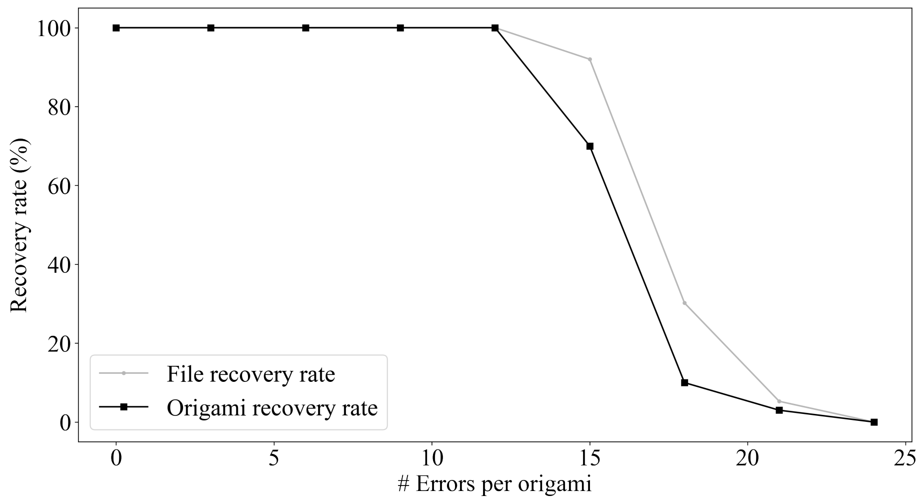


Figure 4.18: The figure illustrates the file and origami recovery rates achieved by our error correction algorithm for 3dNAM. Notably, the implementation of majority voting has proven instrumental in ensuring consistent file recovery even when approximately 60% of the origami is lost. This demonstrates the robustness and effectiveness of our error correction mechanism, which enables reliable data retrieval despite significant origami loss.

We conducted a comprehensive simulation to assess the robustness and efficiency of our algorithm. To begin, we generated files with variable random sizes, ranging from 30 bytes to 2000 bytes. These files were subsequently encoded using our modified encoding algorithm, as outlined in Section 4.3.2. Drawing from insights gained in our previous 2dNAM wet-lab experiment, where we observed the presence of multiple copies of each origami (exceeding 200 copies), we sought to replicate this scenario in our simulation. Thus, we created 20 copies of each origami and randomly oriented them before introducing artificial errors. Various types of errors were introduced, encompassing bit insertions, bit mutations, and bit deletions, with the error rates spanning from 3 to 24 errors per origami.

For our simulation, we utilized origami structures of three layers; each layer contains eight rows and ten columns. Remarkably, our decoding algorithm achieved a flawless file recovery rate of 100% when the average number of errors per origami was less than or equal to 12 (refer to Figure 4.18). However, at 12 errors per origami, the origami recovery rate diminished to approximately 40%. Nevertheless, since we had 20 copies of each origami and employed a majority voting approach, we were still able to successfully recover the file. Consequently, no false positives were encountered in our final results due to this majority voting mechanism. The primary reason for the lower file recovery rate at a higher error rate is missing origami that we were never able to recover.

While we have not yet conducted a wet-lab experiment using our new error correction algorithm for 3dNAM, our simulation results closely resemble those obtained with our previous version of the error correction algorithm (see Figure 4.9). Moreover, we were able to successfully store and retrieve digital information using DNA origami. These findings strongly suggest that our error correction algorithm for 3dNAM is likely to be successful.

CHAPTER 5: CONCLUSION AND FUTURE DIRECTIONS

5.1 Conclusion

The exponential growth of data generation and the limitations of existing storage systems have prompted the search for alternative information storage mediums. DNA has emerged as a promising solution due to its potential for long-term data retention, high storage density, and low energy consumption. This dissertation has explored the application of DNA as a storage medium, focusing on dNAM.

dNAM introduces a novel approach that separates DNA storage technology from sequencing and synthesizing advancements. It utilizes SMLM to encode digital information into specific physical locations within DNA origami structures. Binary data is represented by the presence or absence of fluorescently labeled imager strands at designated docking sites on the DNA origami. As part of this research, we have developed a new error correction algorithm that served as the cornerstone of dNAM. This algorithm enabled the storage of a prototype consisting of 20 bytes of data, with the capability to correct up to 9 bits of error out of 48 bits. The results of this work have been published in Nature Communications.

Emitter localization is one of the most crucial steps of dNAM. One of the primary purposes of our emitter localization algorithm is to isolate attached emitters and see their impact on overall efficiency. To achieve this, our algorithm utilizes a deep learning model that effectively enhances the resolution of the input image. By projecting the centroid position of the emitters from lower-resolution to higher-resolution images while preserving the emitters' PSF, we successfully increase the resolution without distorting the essential characteristics of the emitters. To evaluate the impact of this resolution increase, we conducted an analysis by exporting the emitters from each level. This gave us insight about the multi-emitter's effect on the localization accuracy. As in the later layer multi-emitter gets separated. We were able to observe how the localization accuracy improved with multi-emitter separation. Also, our machine learning algorithm demonstrated remarkable capability in capturing the grid-like structure of the origami.

Building on the success of our initial error correction algorithm, we have made significant advancements by incorporating a simple yet effective intuition. The updated algorithm takes into account the intensity values of data points within an emitter. We observed that data points with intensities closer to the threshold are more likely to contain errors compared to those with higher or lower intensities. In contrast, the previous version treated each data point as having an equal probability of error. By integrating intensity information into our algorithm, we have achieved enhanced efficiency and speed. The algorithm now considers the varying probabilities of error based on the proximity of data points to the threshold value, resulting in improved error correction capabilities.

The previous version of the error correction algorithm relied on a hard-coded mapping scheme, limiting its scalability to specific origami shapes. In contrast, we developed a new mapping scheme that is dynamic and robust, capable of accommodating arbitrary-shaped 3D origami structures. Through the utilization of heuristics and ranking techniques to satisfy origami constraints, we have successfully developed a flexible mapping scheme. Although wet lab experiments for this updated error correction algorithm were not conducted, extensive simulations were performed to validate its effectiveness. These simulations yielded promising results comparable to those obtained during the creation of the dNAM prototype. These results instill confidence in the future success of error correction for 3dNAM.

5.2 Future Directions

DNA, as a storage medium, shows great promise but still requires further research to address its limitations and make it a viable solution for the information storage crisis. The current state of DNA storage is hindered by the high costs and time-consuming processes involved in DNA synthesis, sequencing, and dNAM origami creation. Additionally, the need for sophisticated and expensive equipment restricts extensive research in this field. Despite these challenges, there are several future research topics that can enhance the robustness, efficiency, and viability of dNAM.

To further enhance dNAM, several areas have been identified for future research. These include improving the origami image quality using DL. Make an end-to-end system that will include localization, origami enhancement, origami to binary data conversion and error correction. Cluster the similar origami together to reduce the number of errors.

In this dissertation, we focused on using super resolution primarily for localization purposes. Our super resolution model was designed to re-scale individual emitters rather than the entire origami. However, this same technique can be further employed to enhance the overall quality of the origami and reduce noise, without heavily relying on emitter localization. To achieve this, the super resolution technique will be applied after emitter localization and drift correction. To implement this approach, our existing simulation algorithm to generate origami data (see Section 3.1) can be utilized. Subsequently, the emitters need to be localized, and the corresponding origami images collected. Any available localization algorithm can be used for this purpose. As a result of the localization process, the collected origami images may contain noise. To address this, each origami will also have its corresponding ground truth obtained from the data generation algorithm. These ground truth images can be exported and used as labels in the super-resolution algorithm. The DL model can capture the grid-like structure of the origami during the enhancing process, resulting in further improvement of the input origami image.

Currently, each step of the dNAM process is isolated, leading to information loss at each stage. Developing an end-to-end system that integrates all the steps, such as emitter localization, drift correction, binary data extraction, and error correction, into a single algorithm can mitigate this issue and yield better results. Machine learning and deep learning techniques, such as CNNs for image processing and transformer [136] models for error and drift correction, can be deployed in this unified framework.

In dNAM, multiple copies of each origami are available. Rather than averaging the origami data after error correction, performing the averaging operation before passing them to the error correction algorithm can reduce the need for extensive error correction mechanisms with large amounts of parity and checksum data. Machine learning and deep learning approaches can be employed to identify similarities between origami images at the image level, allowing for averaging of similar origami to filter out errors.

In summary, the exploration of DNA as a storage medium, specifically through sequence-based DNA storage and dNAM, along with the utilization of DNA origami and machine learning techniques, holds significant potential for addressing the growing storage challenges in the digital age. Further research and advancements in these areas will contribute to the development of reliable, high-density, and energy-efficient data storage solutions for the future.

REFERENCES

[1] George M. Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in DNA. *Science*, 337(6102):1628–1628, September 2012. doi: 10.1126/science.1226355. URL `https://doi.org/10.1126/science.1226355`.

[2] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435): 77–80, January 2013. doi: 10.1038/nature11875. URL `https://doi.org/10.1038/nature11875`.

[3] Robert N. Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J. Stark. Robust chemical preservation of digital information on DNA in silica with error-correcting codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, February 2015. doi: 10.1002/anie.201411378. URL `https://doi.org/10.1002/anie.201411378`.

[4] Meinolf Blawat, Klaus Gaedke, Ingo Hütter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W. Pruitt, and George M. Church. Forward error correction for DNA data storage. *Procedia Computer Science*, 80:1011–1022, 2016. doi: 10.1016/j.procs.2016.05.398. URL `https://doi.org/10.1016/j.procs.2016.05.398`.

[5] James Bornholt, Randolph Lopez, Douglas M. Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, March 2016. doi: 10.1145/2872362.2872397. URL `https://doi.org/10.1145/2872362.2872397`.

[6] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher N Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze, and Karin Strauss. Random access in large-scale DNA data storage. 36(3):242–248. ISSN 1087-0156, 1546-1696. doi: 10.1038/nbt.4079. URL `http://www.nature.com/articles/nbt.4079`.

[7] Yaniv Erlich and Dina Zielinski. DNA fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, March 2017. doi: 10.1126/science.aaj2038. URL `https://doi.org/10.1126/science.aaj2038`.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL `http://arxiv.org/abs/1512.03385`. arXiv: 1512.03385.

[9] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution

using a generative adversarial network. 2017. URL http://arxiv.org/abs/1609.04802.

[10] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. 2017. URL http://arxiv.org/abs/1707.02921.

[11] Ralf Jungmann, Christian Steinhauer, Max Scheible, Anton Kuzyk, Philip Tinnefeld, and Friedrich C. Simmel. Single-molecule kinetics and super-resolution microscopy by fluorescence imaging of transient binding on DNA origami. 10 (11):4756–4761. ISSN 1530-6984, 1530-6992. doi: 10.1021/nl103427w. URL https://pubs.acs.org/doi/10.1021/nl103427w.

[12] Martin Ovesný, Pavel Křížek, Josef Borkovec, Zdeněk Švindrych, and Guy M. Hagen. ThunderSTORM: a comprehensive ImageJ plug-in for PALM and STORM data analysis and super-resolution imaging. 30(16):2389–2390. ISSN 1367-4803, 1460-2059. doi: 10.1093/bioinformatics/btu202. URL https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu202.

[13] Daniel Sage, Thanh-An Pham, Hazen Babcock, Tomas Lukes, Thomas Pengo, Jerry Chao, Ramraj Velmurugan, Alex Herbert, Anurag Agrawal, Silvia Colabrese, Ann Wheeler, Anna Archetti, Bernd Rieger, Raimund Ober, Guy M. Hagen, Jean-Baptiste Sibarita, Jonas Ries, Ricardo Henriques, Michael Unser, and Seamus Holden. Super-resolution fight club: assessment of 2d and 3d single-molecule localization microscopy software. 16(5):387–395. ISSN 1548-

7091, 1548-7105. doi: 10.1038/s41592-019-0364-4. URL http://www.nature.com/articles/s41592-019-0364-4.

[14] Victor Zhirnov, Reza M. Zadegan, Gurtej S. Sandhu, George M. Church, and William L. Hughes. Nucleic acid memory. 15(4):366–370. ISSN 1476-1122, 1476-4660. doi: 10.1038/nmat4594. URL http://www.nature.com/articles/nmat4594.

[15] Barry Schechtman, Don Peterson, Will Qualls, Molly Rector, Paul Scheuer, Beth Walker, Rod Wideman, Tom Wultich, and Dave Woito. International Magnetic Tape Storage Roadmap Part I: Applications & Systems. (November), 2011.

[16] Andy Extance. How DNA could store all the world's data. *Nature*, 537(7618): 22–24, August 2016. doi: 10.1038/537022a. URL https://doi.org/10.1038/537022a.

[17] Semiconductor Industry Association. International Technology Roadmap for Semiconductors, 2015 Results. *Itrpv*, 0(March):1–37, 2016. ISSN 0018-9162. URL papers2://publication/uuid/20F56C7C-3684-4039-B043-D3DE7C5293FA.

[18] Yaya Hao, Qian Li, Chunhai Fan, and Fei Wang. Data storage based on DNA. *Small Structures*, 2(2):2000046, November 2020. doi: 10.1002/sstr.202000046. URL https://doi.org/10.1002/sstr.202000046.

[19] Kaikai Chen, Jinglin Kong, Jinbo Zhu, Niklas Ermann, Paul Predki, and Ulrich F. Keyser. Digital data storage using DNA nanostructures and solid-state

nanopores. *Nano Letters*, 19(2):1210–1215, December 2018. doi: 10.1021/acs.nanolett.8b04715. URL `https://doi.org/10.1021/acs.nanolett.8b04715`.

[20] S. Kasra Tabatabaei, Boya Wang, Nagendra Bala Murali Athreya, Behnam Enghiad, Alvaro Gonzalo Hernandez, Christopher J. Fields, Jean-Pierre Leburton, David Soloveichik, Huimin Zhao, and Olgica Milenkovic. DNA punch cards for storing data on native DNA sequences via enzymatic nicking. *Nature Communications*, 11(1), April 2020. doi: 10.1038/s41467-020-15588-z. URL `https://doi.org/10.1038/s41467-020-15588-z`.

[21] George D. Dickinson, Golam Md Mortuza, William Clay, Luca Piantanida, Christopher M. Green, Chad Watson, Eric J. Hayden, Tim Andersen, Wan Kuang, Elton Graugnard, Reza Zadegan, and William L. Hughes. An alternative approach to nucleic acid memory. *Nature Communications*, 12(1), April 2021. doi: 10.1038/s41467-021-22277-y. URL `https://doi.org/10.1038/s41467-021-22277-y`.

[22] Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, March 2006. doi: 10.1038/nature04586. URL `https://doi.org/10.1038/nature04586`.

[23] Chaoyang Guan, Xiaoli Zhu, and Chang Feng. DNA nanodevice-based drug delivery systems. *Biomolecules*, 11(12):1855, December 2021. doi: 10.3390/biom11121855. URL `https://doi.org/10.3390/biom11121855`.

[24] Seaim Aye and Yusuke Sato. Therapeutic applications of programmable DNA nanostructures. *Micromachines*, 13(2):315, February 2022. doi: 10.3390/mi13020315. URL `https://doi.org/10.3390/mi13020315`.

[25] Qian Zhang, Qiao Jiang, Na Li, Luru Dai, Qing Liu, Linlin Song, Jinye Wang, Yaqian Li, Jie Tian, Baoquan Ding, and Yang Du. Dna origami as an in vivo drug delivery vehicle for cancer therapy. *ACS Nano*, 8(7):6633–6643, June 2014. doi: 10.1021/nn502058j. URL `https://doi.org/10.1021/nn502058j`.

[26] Jonathan F. Berengut, Chak Kui Wong, Julian C. Berengut, Jonathan P. K. Doye, Thomas E. Ouldridge, and Lawrence K. Lee. Self-limiting polymerization of DNA origami subunits with strain accumulation. *ACS Nano*, 14(12):17428–17441, November 2020. doi: 10.1021/acsnano.0c07696. URL `https://doi.org/10.1021/acsnano.0c07696`.

[27] Xiaogang Han, Zihao Zhou, Fan Yang, and Zhaoxiang Deng. Catch and release: DNA tweezers that can capture, hold, and release an object under control. *Journal of the American Chemical Society*, 130(44):14414–14415, October 2008. doi: 10.1021/ja805945r. URL `https://doi.org/10.1021/ja805945r`.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[30] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed,

Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46448-0_2. URL https://doi.org/10.1007/978-3-319-46448-0_2.

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015. URL https://arxiv.org/abs/1506.02640.

[32] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020. URL https://arxiv.org/abs/2006.11477.

[33] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL https://arxiv.org/abs/1409.0473.

[34] Lefteris Koumakis. Deep learning models in genomics; are we there yet? *Computational and Structural Biotechnology Journal*, 18:1466–1473, 2020. doi: 10.1016/j.csbj.2020.06.017. URL https://doi.org/10.1016/j.csbj.2020.06.017.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. URL http://arxiv.org/abs/1505.04597. arXiv: 1505.04597.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*,

521(7553):436–444, May 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14539. URL http://www.nature.com/articles/nature14539.

[37] Elias Nehme, Lucien E. Weiss, Tomer Michaeli, and Yoav Shechtman. Deep-STORM: super-resolution single-molecule microscopy by deep learning. 5(4): 458, 2018. ISSN 2334-2536. doi: 10.1364/OPTICA.5.000458. URL https://www.osapublishing.org/abstract.cfm?URI=optica-5-4-458.

[38] Elias Nehme, Daniel Freedman, Racheli Gordon, Boris Ferdman, Lucien E. Weiss, Onit Alalouf, Tal Naor, Reut Orange, Tomer Michaeli, and Yoav Shechtman. DeepSTORM3d: dense 3d localization microscopy and PSF design by deep learning. 17(7):734–740. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-020-0853-5. URL http://www.nature.com/articles/s41592-020-0853-5.

[39] Nicholas Boyd, Eric Jonas, Hazen Babcock, and Benjamin Recht. DeepLoco: Fast 3d localization microscopy using neural networks, February 2018. URL https://doi.org/10.1101/267096.

[40] Artur Speiser, Lucas-Raphael Müller, Philipp Hoess, Ulf Matti, Christopher J. Obara, Wesley R. Legant, Anna Kreshuk, Jakob H. Macke, Jonas Ries, and Srinivas C. Turaga. Deep learning enables fast and dense single-molecule localization with high accuracy. 18(9):1082–1090. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-021-01236-x. URL https://www.nature.com/articles/s41592-021-01236-x.

[41] Golam Md Mortuza, Jorge Guerrero, Shoshanna Llewellyn, Michael D. Tobiason, George D. Dickinson, William L. Hughes, Reza Zadegan, and Tim

Andersen. In-vitro validated methods for encoding digital data in deoxyribonucleic acid (DNA). *BMC Bioinformatics*, 24(1), April 2023. doi: 10.1186/s12859-023-05264-6. URL `https://doi.org/10.1186/s12859-023-05264-6`.

[42] Golam Md Mortuza, Michael D. Tobiason, Kelsey Suyehira, William L. Hughes, Tim Andersen, and Reza Zadegan. A method for storing information in DNA with improved dropout tolerance. June 2023. doi: 10.1101/2023.06.20.545769. URL `https://doi.org/10.1101/2023.06.20.545769`.

[43] Jerrod J Schwartz, Choli Lee, and Jay Shendure. Accurate gene synthesis with tag-directed retrieval of sequence-verified DNA molecules. *Nature Methods*, 9 (9):913–915, August 2012. doi: 10.1038/nmeth.2137. URL `https://doi.org/10.1038/nmeth.2137`.

[44] S. M.Hossein Tabatabaei Yazdi, Yongbo Yuan, Jian Ma, Huimin Zhao, and Olgica Milenkovic. A Rewritable, Random-Access DNA-Based Storage System. *Scientific Reports*, 2015. ISSN 20452322. doi: 10.1038/srep14138.

[45] Michael G Ross, Carsten Russ, Maura Costello, Andrew Hollinger, Niall J Lennon, Ryan Hegarty, Chad Nusbaum, and David B Jaffe. Characterizing and measuring bias in sequence data. *Genome Biology*, 14(5):R51, 2013. doi: 10.1186/gb-2013-14-5-r51. URL `https://doi.org/10.1186/gb-2013-14-5-r51`.

[46] Guruprasad Ananda, Erin Walsh, Kimberly D. Jacob, Maria Krasilnikova, Kristin A. Eckert, Francesca Chiaromonte, and Kateryna D. Makova. Distinct mutational behaviors differentiate short tandem repeats from microsatellites in the human genome. *Genome Biology and Evolution*, 5(3):606–620, December 2012. doi: 10.1093/gbe/evs116. URL `https://doi.org/10.1093/gbe/evs116`.

[47] Karen Poon and Robert B. Macgregor. Unusual behavior exhibited by multistranded guanine-rich DNA complexes. *Biopolymers*, 45(6):427–434, May 1998. doi: 10.1002/(sici)1097-0282(199805)45:6⟨427::aid-bip2⟩3.0.co;2-r. URL `https://doi.org/10.1002/(sici)1097-0282(199805)45:6<427::aid-bip2>3.0.co;2-r`.

[48] Chengtao Xu, Chao Zhao, Biao Ma, and Hong Liu. Uncertainties in synthetic DNA-based data storage. *Nucleic Acids Research*, 49(10):5451–5469, April 2021. doi: 10.1093/nar/gkab230. URL `https://doi.org/10.1093/nar/gkab230`.

[49] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, April 1950. doi: 10.1002/j.1538-7305.1950.tb00463.x. URL `https://doi.org/10.1002/j.1538-7305.1950.tb00463.x`.

[50] Christopher N. Takahashi, Bichlien H. Nguyen, Karin Strauss, and Luis Ceze. Demonstration of End-to-End Automation of DNA Data Storage. *Scientific Reports*, 2019. ISSN 20452322. doi: 10.1038/s41598-019-41228-8.

[51] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960. doi: 10.1137/0108018. URL `https://doi.org/10.1137/0108018`.

[52] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, March 1960. doi: 10.1016/s0019-9958(60)90287-4. URL `https://doi.org/10.1016/s0019-9958(60)90287-4`.

[53] Stephen B Wicker and Vijay K Bhargava. *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[54] Leon Anavy, Inbal Vaknin, Orna Atar, Roee Amit, and Zohar Yakhini. Data storage in DNA with fewer synthesis cycles using composite DNA letters. *Nature Biotechnology*, 2019. ISSN 15461696. doi: 10.1038/s41587-019-0240-x.

[55] Randolph Lopez, Yuan-Jyue Chen, Siena Dumas Ang, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Georg Seelig, Karin Strauss, and Luis Ceze. DNA assembly for nanopore data storage readout. *Nature Communications*, 10(1), July 2019. doi: 10.1038/s41467-019-10978-4. URL `https://doi.org/10.1038/s41467-019-10978-4`.

[56] R. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, 8(1):21–28, January 1962. doi: 10.1109/tit.1962.1057683. URL `https://doi.org/10.1109/tit.1962.1057683`.

[57] Aldrin Kay-Yuen Yim, Allen Chi-Shing Yu, Jing-Woei Li, Ada In-Chun Wong, Jacky F. C. Loo, King Ming Chan, S. K. Kong, Kevin Y. Yip, and Ting-Fung Chan. The essential component in DNA-based information storage system: Robust error-tolerating module. *Frontiers in Bioengineering and Biotechnology*, 2, November 2014. doi: 10.3389/fbioe.2014.00049. URL `https://doi.org/10.3389/fbioe.2014.00049`.

[58] Peng Fei and Zhiying Wang. LDPC codes for portable DNA storage. In *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, July 2019. doi: 10.1109/isit.2019.8849814. URL `https://doi.org/10.1109/isit.2019.8849814`.

[59] Shubham Chandak, Hanlee Ji, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, and Tsachy Weissman. 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)*. IEEE, September 2019. doi: 10.1109/allerton.2019.8919890. URL `https://doi.org/10.1109/allerton.2019.8919890`.

[60] D. J.C. MacKay. Fountain codes. In *IEE Proceedings: Communications*, 2005. doi: 10.1049/ip-com:20050237.

[61] Richard P. Feynman. There's plenty of room at the bottom [data storage]. *Journal of Microelectromechanical Systems*, 1(1):60–66, 1992. doi: 10.1109/84.128057.

[62] Joe Davis. Microvenus. *Art Journal*, 1996. ISSN 00043249. doi: 10.2307/777811.

[63] Zhi Ping, Shihong Chen, Guangyu Zhou, Xiaoluo Huang, Sha Joe Zhu, Haoling Zhang, Henry H. Lee, Zhaojun Lan, Jie Cui, Tai Chen, Wenwei Zhang, Huanming Yang, Xun Xu, George M. Church, and Yue Shen. Towards practical and robust DNA-based data archiving using the yin–yang codec system. 2(4):234–242. ISSN 2662-8457. doi: 10.1038/s43588-022-00231-2. URL `https://www.nature.com/articles/s43588-022-00231-2`. Number: 4 Publisher: Nature Publishing Group.

[64] Ernst Abbe. Beiträge zur theorie des mikroskops und der mikroskopischen wahrnehmung. *Archiv für Mikroskopische Anatomie*, 9(1):413–468, December 1873. doi: 10.1007/bf02956173. URL `https://doi.org/10.1007/bf02956173`.

[65] Beyond the diffraction limit. *Nature Photonics*, 3(7):361–361, July 2009. doi: 10.1038/nphoton.2009.100. URL `https://doi.org/10.1038/nphoton.2009.100`.

[66] Ismail M. Khater, Ivan Robert Nabi, and Ghassan Hamarneh. A review of super-resolution single-molecule localization microscopy cluster analysis and quantification methods. 1(3):100038. ISSN 2666-3899. doi: 10.1016/j.patter.2020.100038. URL `https://www.sciencedirect.com/science/article/pii/S266638992030043X`.

[67] Steffen J Sahl and WE Moerner. Super-resolution fluorescence imaging with single molecules. 23(5):778–787. ISSN 0959-440X. doi: 10.1016/j.sbi.2013.07.010. URL `https://www.sciencedirect.com/science/article/pii/S0959440X13001437`.

[68] Jerry Chao, E. Sally Ward, and Raimund J. Ober. Fisher information theory for parameter estimation in single molecule microscopy: tutorial. *Journal of the Optical Society of America A*, 33(7):B36, June 2016. doi: 10.1364/josaa.33.000b36. URL `https://doi.org/10.1364/josaa.33.000b36`.

[69] Mickaël Lelek, Melina T. Gyparaki, Gerti Beliu, Florian Schueder, Juliette Griffié, Suliana Manley, Ralf Jungmann, Markus Sauer, Melike Lakadamyali, and Christophe Zimmer. Single-molecule localization microscopy. *Nature Reviews Methods Primers*, 1(1), June 2021. doi: 10.1038/s43586-021-00038-x. URL `https://doi.org/10.1038/s43586-021-00038-x`.

[70] Ricardo Henriques, Mickael Lelek, Eugenio F Fornasiero, Flavia Valtorta,

Christophe Zimmer, and Musa M Mhlanga. QuickPALM: 3d real-time photoactivation nanoscopy image processing in ImageJ. *Nature Methods*, 7(5):339–340, May 2010. doi: 10.1038/nmeth0510-339. URL `https://doi.org/10.1038/nmeth0510-339`.

[71] J. A. Högbom. Aperture Synthesis with a Non-Regular Distribution of Interferometer Baselines. *Astronomy and Astrophysics Supplement*, 15:417, June 1974.

[72] Raimund J Ober, Sripad Ram, and E Sally Ward. Localization accuracy in single-molecule microscopy. *Biophysical journal*, 86(2):1185–1200, 2004.

[73] Kim I Mortensen, L Stirling Churchman, James A Spudich, and Henrik Flyvbjerg. Optimized localization analysis for single-molecule tracking and super-resolution microscopy. *Nature methods*, 7(5):377–381, 2010.

[74] Daniel Sage, Thanh-An Pham, Hazen Babcock, Tomas Lukes, Thomas Pengo, Jerry Chao, Ramraj Velmurugan, Alex Herbert, Anurag Agrawal, Silvia Colabrese, et al. Super-resolution fight club: assessment of 2d and 3d single-molecule localization microscopy software. *Nature methods*, 16(5):387–395, 2019.

[75] Alex Small and Shane Stahlheber. Fluorophore localization algorithms for super-resolution microscopy. *Nature methods*, 11(3):267–279, 2014.

[76] Varun Mannam, Yide Zhang, Xiaotong Yuan, and Scott Howard. Deep learning-based super-resolution fluorescence microscopy on small datasets. page 7. doi: 10.1117/12.2578519. URL `http://arxiv.org/abs/2103.04989`.

[77] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[78] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

[79] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10): 1995, 1995.

[80] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[81] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. URL `https://icml.cc/Conferences/2010/papers/432.pdf`.

[82] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[83] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In

Proceedings of the IEEE international conference on computer vision, pages 1026–1034, 2015.

[84] Andrew Senior and Xin Lei. Fine context, low-rank, softplus deep neural networks for mobile speech recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7644–7648. IEEE, 2014.

[85] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[86] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[87] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. Deep learning for image super-resolution: A survey. URL `http://arxiv.org/abs/1902.06068`.

[88] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. URL `http://arxiv.org/abs/1511.04587`.

[89] Muhammad Haris, Greg Shakhnarovich, and Norimichi Ukita. Deep back-projection networks for super-resolution. 2018. doi: 10.48550/ARXIV.1803.02735. URL `https://arxiv.org/abs/1803.02735`.

[90] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep

laplacian pyramid networks for fast and accurate super-resolution. 2017. URL `http://arxiv.org/abs/1704.03915`.

[91] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[92] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2798. IEEE, . ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.298. URL `http://ieeexplore.ieee.org/document/8099781/`.

[93] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. 2021. URL `http://arxiv.org/abs/2104.07636`.

[94] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. MemNet: A persistent memory network for image restoration. . URL `http://arxiv.org/abs/1708.02209`.

[95] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. pages 1874–1883, 2016.

[96] Hao Zhang, Chunyu Fang, Xinlin Xie, Yicong Yang, Wei Mei, Di Jin, and Peng Fei. High-throughput, high-resolution deep learning microscopy based on

registration-free generative adversarial network. 10(3):1044. ISSN 2156-7085, 2156-7085. doi: 10.1364/BOE.10.001044. URL `https://www.osapublishing.org/abstract.cfm?URI=boe-10-3-1044`.

[97] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, June 2010. doi: 10.1109/cvpr.2010.5539957. URL `https://doi.org/10.1109/cvpr.2010.5539957`.

[98] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. URL `https://arxiv.org/abs/1603.07285`.

[99] Andrea Vedaldi and Karel Lenc. Matconvnet - convolutional neural networks for matlab, 2014. URL `https://arxiv.org/abs/1412.4564`.

[100] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[101] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[102] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[103] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and Checkerboard Artifacts. *Distill*, 1(10):10.23915/distill.00003, October 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

[104] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan Wang. Is the deconvolution layer the same as a convolutional layer? URL `http://arxiv.org/abs/1609.07009`.

[105] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European conference on computer vision (ECCV)*, pages 252–268, 2018.

[106] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. 3(1):47–57. ISSN 2333-9403, 2334-0118. doi: 10.1109/TCI.2016.2644865. URL `http://ieeexplore.ieee.org/document/7797130/`.

[107] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[108] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[109] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Confer-*

ence on Computer Vision and Pattern Recognition, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[110] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[111] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. pages 1637–1645, 2016.

[112] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. 2018. URL `http://arxiv.org/abs/1802.08797`.

[113] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018.

[114] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. URL `http://arxiv.org/abs/1406.2661`. arXiv: 1406.2661.

[115] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. *arXiv:2006.11239 [cs, stat]*, December 2020. URL `http://arxiv.org/abs/2006.11239`. arXiv: 2006.11239.

[116] Daniel Sage, Hagai Kirshner, Thomas Pengo, Nico Stuurman, Junhong Min, Suliana Manley, and Michael Unser. Quantitative evaluation of software packages for single-molecule localization microscopy. *Nature Methods*, 12(8):717–724, June 2015. doi: 10.1038/nmeth.3442. URL `https://doi.org/10.1038/nmeth.3442`.

[117] Nils Gustafsson, Siân Culley, George Ashdown, Dylan M. Owen, Pedro Matos Pereira, and Ricardo Henriques. Fast live-cell conventional fluorophore nanoscopy with ImageJ through super-resolution radial fluctuations. *Nature Communications*, 7(1), August 2016. doi: 10.1038/ncomms12471. URL `https://doi.org/10.1038/ncomms12471`.

[118] Caroline A Schneider, Wayne S Rasband, and Kevin W Eliceiri. NIH image to ImageJ: 25 years of image analysis. *Nature Methods*, 9(7):671–675, June 2012. doi: 10.1038/nmeth.2089. URL `https://doi.org/10.1038/nmeth.2089`.

[119] Joerg Schnitzbauer, Maximilian T Strauss, Thomas Schlichthaerle, Florian Schueder, and Ralf Jungmann. Super-resolution microscopy with DNA-PAINT. 12(6):1198–1228. ISSN 1754-2189, 1750-2799. doi: 10.1038/nprot.2017.024. URL `http://www.nature.com/articles/nprot.2017.024`.

[120] Peiyi Zhang, Sheng Liu, Abhishek Chaurasia, Donghan Ma, Michael J Mlodzianoski, Eugenio Culurciello, and Fang Huang. Analyzing complex single-molecule emission patterns with deep learning. *Nature methods*, 15(11):913–916, 2018.

[121] Nicholas Boyd, Eric Jonas, Hazen Babcock, and Benjamin Recht. Deeploco:

fast 3d localization microscopy using neural networks. *BioRxiv*, page 267096, 2018.

[122] Wei Ouyang, Andrey Aristov, Mickaël Lelek, Xian Hao, and Christophe Zimmer. Deep learning massively accelerates super-resolution localization microscopy. *Nature biotechnology*, 36(5):460–468, 2018.

[123] Leonhard Möckl, Anish R Roy, Petar N Petrov, and WE Moerner. Accurate and rapid background estimation in single-molecule localization microscopy using the deep neural network bgnet. *Proceedings of the National Academy of Sciences*, 117(1):60–67, 2020.

[124] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[125] Redouane Lguensat, Miao Sun, Ronan Fablet, Pierre Tandeo, Evan Mason, and Ge Chen. Eddynet: A deep neural network for pixel-wise classification of oceanic eddies. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 1764–1767. IEEE, 2018.

[126] Adrian Baddeley, Imre Bárány, and Rolf Schneider. Spatial point processes and their applications. *Stochastic Geometry: Lectures Given at the CIME Summer School Held in Martina Franca, Italy, September 13–18, 2004*, pages 1–75, 2007.

[127] David Baddeley and Joerg Bewersdorf. Biological insight from super-resolution microscopy: what we can learn from localization-based images. *Annual review of biochemistry*, 87:965–989, 2018.

[128] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, March 2017. doi: 10.1109/tci.2016.2644865. URL `https://doi.org/10.1109/tci.2016.2644865`.

[129] Diederik P Kingma and Jimmy Lei. Adam: A Method for Stochastic Optimization. page 15, 2015.

[130] Luigi Di Stefano and Andrea Bulgarelli. A simple and efficient connected components labeling algorithm. In *Proceedings 10th international conference on image analysis and processing*, pages 322–327. IEEE, 1999.

[131] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[132] Joerg Schnitzbauer, Maximilian T Strauss, Thomas Schlichthaerle, Florian Schueder, and Ralf Jungmann. Super-resolution microscopy with dna-paint. *Nature protocols*, 12(6):1198–1228, 2017.

[133] M. Luby. LT codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–280. IEEE Comput. Soc. ISBN 978-0-7695-1822-0. doi: 10.1109/SFCS.2002.1181950. URL `http://ieeexplore.ieee.org/document/1181950/`.

[134] Maximilian T. Strauss, Florian Schueder, Daniel Haas, Philipp C. Nickels, and Ralf Jungmann. Quantifying absolute addressability in DNA origami with molecular resolution. *Nature Communications*, 9(1), apr 2018. doi: 10.1038/s41467-018-04031-z. URL `https://doi.org/10.1038/s41467-018-04031-z`.

[135] Christopher Michael Green. *Nanoscale optical and correlative microscopies for quantitative characterization of DNA nanostructures.* PhD thesis, 2019.

[136] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL `http://arxiv.org/abs/1706.03762`. arXiv: 1706.03762.

[137] Kelsey Suyehira. Using DNA For Data Storage: Encoding and Decoding Algorithm Development. *Boise State University Theses and Dissertations*, December 2018. doi: 10.18122/td/1500/boisestate. URL `https://scholarworks.boisestate.edu/td/1500`.

[138] Alexey V. Lobanov, Anton A. Turanov, Dolph L. Hatfield, and Vadim N. Gladyshev. Dual functions of codons in the genetic code. *Critical Reviews in Biochemistry and Molecular Biology*, 45(4):257–265, August 2010. ISSN 1040-9238. doi: 10.3109/10409231003786094. URL `https://doi.org/10.3109/10409231003786094`.

APPENDIX A:
A METHOD FOR STORING INFORMATION IN DNA WITH IMPROVED DROPOUT TOLERANCE

This appendix presents an improved version of the existing “DNA Fountain” method (see Figure A.1), reporting a sequence-based DNA storage methodology. In this work we have developed and experimentally tested a robust algorithm to write digital data in pools of DNA strands by applying a rateless erasure code (*i.e.* fountain code), a RS code, and an oligomer mapping code. Our new method includes changes to the fountain code, the oligomer mapping code, and the encoding and decoding processes. We have tested and benchmarked our algorithm vs. similar algorithms and found that our method increases robustness to dropout, decreases encoding time, and decreases decoding time. The new method was validated *in-vitro* by successfully storing and recovering 105,360 bits of information. The advantages of the new method make it more appropriate for applications where information recovery is critical, where substantial sequence loss is expected, and/or where computational resources are limited. Furthermore, the inclusion of a new oligomer mapping code enabled us to mitigate errors by restricting sequences of repeated bases and enhance security by eliminating start/stop codons, thus minimizing the risk of interaction with living cells. A significant portion of the following methodology (Section A.1) has been contributed by Kelsey Suyehira, as documented in their master’s thesis titled “Using DNA For Data Storage: Encoding and Decoding Algorithm Development” [137].

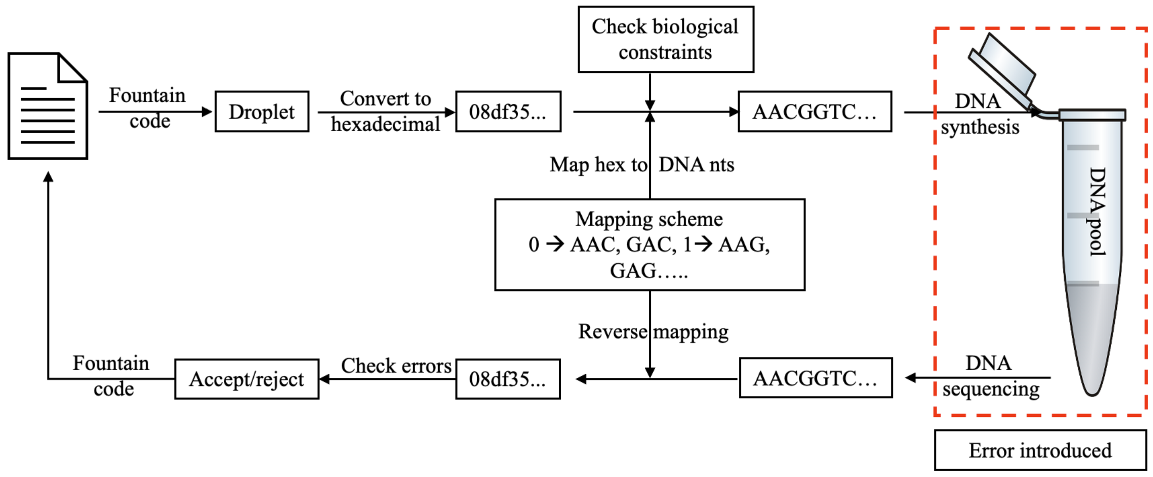


Figure A.1: The figure provides an overview of our proposed solutions for sequence-based DNA storage methodology.

A.1 Methods and Materials

Here, the following information storage method was used. At a high level of abstraction (Figure 1.1), this new method encodes a digital file into DNA oligomers via a write process and recovers the file from DNA oligomers using a read process. In greater detail, this method generates binary sequences according to a rateless erasure code (*i.e.* a fountain code) (see Algorithm 2), extends these binary sequences using a RS error correction code, and then converts these extended binary sequences to base-sequences using an oligomer mapping code. First, a number of oligomers equal to the number of file segments is generated. Next, new oligomers are incrementally generated until the file can be successfully decoded. Once the file has been successfully decoded, additional oligomers are generated until the redundancy is greater than or equal to a requested value. The processes for generating extended binary sequences and converting them to DNA base-sequences are detailed in the following paragraphs.

The following process was used to generate a binary sequence referred to as a

Algorithm 2 Fountain code encoding

```
    Input: Data to encode
    Output: List of droplet
 1: procedure FountainCodeEncode
 2:     segments ← split data into equal non-overlapping segments
 3:     droplets ← []
 4:     while droplets.len() ≤ segments.len() * redundancy do
 5:         seed ← random number
 6:         degree_distributor ← PRNG(seed)
 7:         segment_ids ← degree_distributor.get_droplet_info()
 8:         droplet_data ← map(XOR, segments[segment_ids])
 9:         droplet ← seed + droplet_data
10:         droplet ← ReedSolomonEncode(droplet)
11:         droplets.append(droplet)
12:     end while
13: end procedure
```

binary droplet. A random integer, referred to as the seed value, was generated using a PRNG. This seed value was used to initialize a second PRNG, which was used to select segments of the target file to include in the binary droplet. A binary sequence, referred to as the data value, was created by applying an exclusive-or to the file segments specified by the second PRNG. A binary sequence, referred to as the error-correction value, was calculated by applying a RS error correction code [51] to the concatenated seed and data values. The final binary sequence, referred to as the binary droplet, was created by concatenating the seed, data, and error-correction values.

An oligomer (*i.e.*, sequence of bases) encoding a given binary droplet was generated using the following process. First, the binary sequence is converted to a hexadecimal (hex) sequence. A list of three-base sequences (i.e., codons) corresponding to each hex value is either provided by the user, or the default hex-codon map described

Algorithm 3 Mapping encode

```
    Input: List of droplet
    Output: List of DNA sequences
 1: procedure DropletToDNASequences
 2:     num_backtrack ← 0
 3:     for droplet = list of droplet do
 4:         droplet_hex ← binary_to_hex(droplet)
 5:         codon_list ← []
 6:         while i ≤ droplet_hex.len() AND
                 num_backtrack ≤ maximum_allowed_backtrack do
 7:             codon_options = hex_to_codon_map(droplet_hex[i])
 8:             for codon=codon_options do
 9:                 if codon_list.optimum_gc_count() AND
                       codon_list.no_homopolymer() AND
                       codon_list[i] != codon then
10:                     codon_list[i] = codon
11:                     break
12:                 else
13:                     i -= 1
14:                     num_backtrack += 1
15:                 end if
16:             end for
17:         end while
18:         dna_sequences ← codon_list.toString()
19:     end for
20: end procedure
```

Algorithm 4 Mapping decode

```
    Input: List of DNA sequences
    Output: List of droplet
 1: procedure MappingDecode
 2:     for Sequence=List of DNA sequences do
 3:         codons = seqeunces.toCodon()          ▷ Split into three letter list element
 4:         binary_data ← []
 5:         for codon=codons do
 6:             hex_codon ← codon_to_hex_map(codon)
 7:             binary_data.append(hex_codon.toBinary())
 8:         end for
 9:         binary_string ← binary_data.toString()
10:     end for
11: end procedure
```

below and reported in table A.1 is used. A codon is then assigned to the first hex value by selecting the available codon with with highest GC content. This represents the first partial solution of the backtracking algorithm. If GC content is greater than or equal to 50%, the partial solution is extended by assigning the available codon with lowest GC content. If GC content is less than 50%, the partial solution is extended by assigning the available codon with highest GC content. The partial solution is rejected if it contains enough G's or C's such that the final sequence could not have acceptable GC content. A partial solution is also rejected if it contains sequences of repeated bases larger than a threshold provided by the user. If possible, a rejected partial solution is backtracked by iterating backward through the assigned codons and selecting the next available codon. If no backtrack is possible, a null result is returned. Partial solutions are accepted if a codon was assigned to every hex-value, GC content is acceptable, and no repeat-sequences larger than the threshold are present. The mapping process during encoding is shown in Algorithm 3.

Table A.1: The hexadecimal-codon map for the new software.

Hexadecimal	Codons
0	AAC, GAC
1	AAG, GAG
2	AGG, GTC, TCT
3	TCG, CGA
4	ACT, GCT, TCC
5	ACC, GCC, CGT
6	ACG, GCG
7	AGA, GGA
8	AGT, GGT
9	AGC, GGC, CCG
a	GAA, CGG
b	TAA, CAA
c	TAC, CCT, ATC
d	TAG, CGC
e	TTA, CTA, GTT
f	TTC, CTC, GTA

The default codons corresponding to each hex value are listed in table A.1 and were generated using the following considerations [137]. First, of the sixty-four possible three-base codons, four are homopolymers consisting of a single repeated base. Elimination of these four codons prevent the existence of any homopolymer runs greater than four bases. Of the remaining sixty codons, twelve codons (AAT, ATA, ATT, ATG, CAC, CAT, CAG, CTT, CTG, TAT, TTG, and GTG) are “start codons” used in cellular processes [138] and were removed. Of the remaining forty-eight codons, nine additional codons (ACA, TCA, CCA, GCA, TGA, TGT, TGC, TGG, and GAT) were removed to avoid the creation of “start codons” in the overlap of two adjacent codons. The remaining thirty-nine codons were assigned to hex values based on intuition.

Software implementing the new method was created from the source code provided by Erlich and Zielinski [7]. Relative to this prior method, the following three key updates were made. (1) Oligomers are now generated until the file can be successfully decoded and then additional oligomers are included for redundancy. (2) The binary sequence of each droplet is now converted to a sequence of DNA bases using the new oligomer-mapping code detailed above. (3) Decoding is now done using a breadth-first approach in which a recovered file segment is removed from other binary droplets before any newly recovered segments are processed. The mapping decoding process and fountain decoding process is shown in Algorithm 4 and Algorithm 5 respectively.

A.2 Results

A.2.1 Simulation

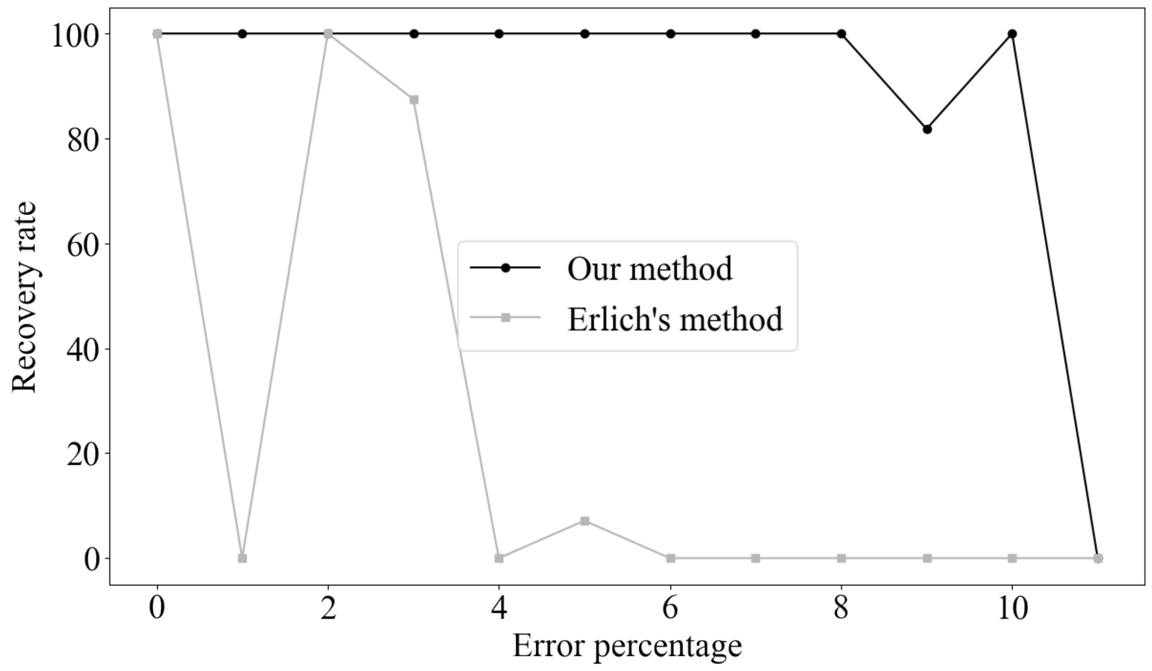
To evaluate the robustness and efficiency of our algorithm, we conducted simulations using randomly generated files ranging in size from 1 MB to 49 MB, with 1 MB intervals. Each file underwent ten simulations with varying levels of simulated insertion, deletion, and mutation errors, including the complete deletion of a randomly selected sequence. Approximately 11% of the total sequences were affected by at least one of these errors. Nevertheless, we were able to successfully recover the file in every

Algorithm 5 Fountain code decode

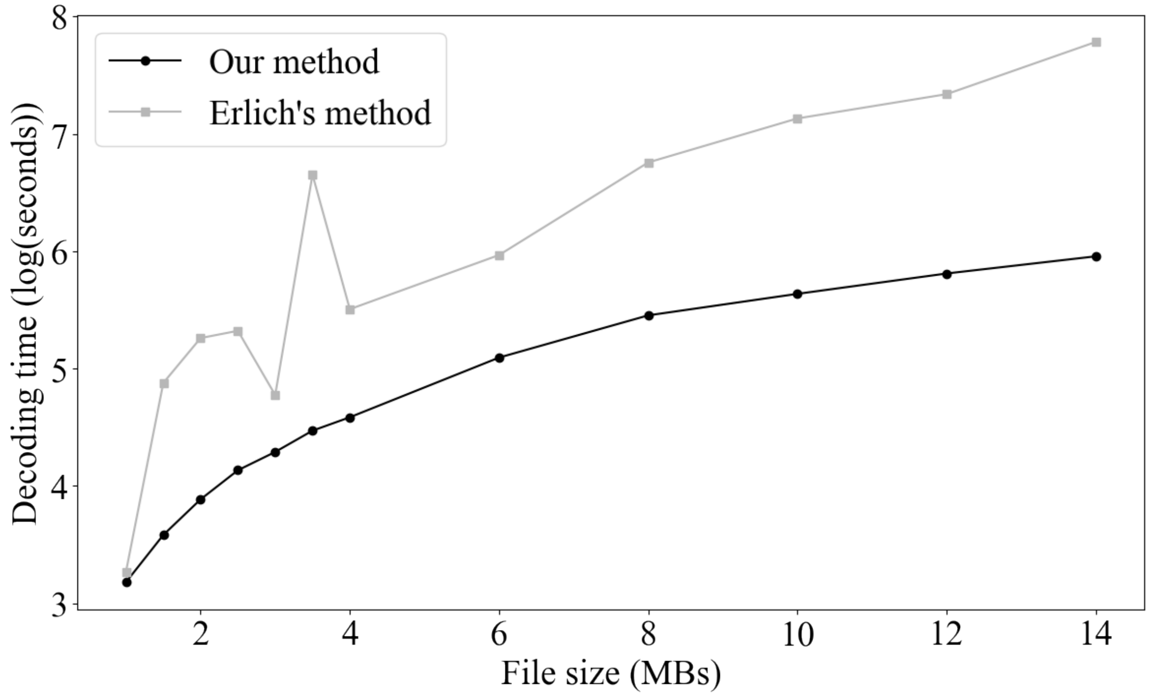
```
    Input: List of droplet, total_segments
    Output: data that was encoded
 1: done_segments_ids ← []
 2: droplet_queue ← queue()
 3: segments ← [None] * total_segments
 4: procedure ProcessSingle(droplet_data, single_segment_id)
 5:     segments[single_segment_id] ← droplet_data
 6:     done_segments_ids.append(single_segment_id)
 7:     for segments_ids = droplet_queue do
 8:         if single_segment_id in segment_ids then
 9:             new_segment_ids ← segments_ids.remove(single_segment_id)
10:             new_droplet_data ← map(XOR, [droplet_queue[segments_ids],
    droplet_data])
11:             if new_segment_ids.len() = 1 then
12:                 ProcessSingle(new_droplet_data, new_segment_list[0])
13:             else
14:                 droplet_queue[segments_ids].remove()
15:                 droplet_queue[new_segment_ids] ← new_droplet_data
16:             end if
17:         end if
18:     end for
19: end procedure
20: procedure FountainCodeDecode
21:     for droplet=list of droplets do
22:         if done_segments_ids.len() = total_segments then
23:             return segments
24:         end if
25:         if droplet.checkReedSolomon() then
26:             seed, droplet_data, rs = droplet.split()
27:             degree_distributor ← PRNG(seed)
28:             segment_ids ← degree_distributor.get_droplet_info()
29:             if segment_ids.len() = 1 then
30:                 ProcessSingle(droplet_data, segment_ids[0])
31:             else
32:                 droplet_queue[segment_ids] ← droplet_data
33:             end if
34:         end if
35:     end for
36: end procedure
```

simulation (see Figure A.2a). Furthermore, to test the upper limit of our algorithm, we attempted to recover a 1 GB file with randomly inserted errors, and we achieved successful retrieval.

We also compared the results of our algorithm with Erlich's algorithm. Our encoding and decoding times were significantly faster. Specifically, for a 16 Megabyte file, our encoding time was 1.5 times faster, and our decoding time was 148 times faster (see Figure A.2b). Moreover, our decoder demonstrated superior robustness in file recovery (see Figure A.2a).



(a) File recovery percentage by only using a sub-sample of reads from overall reads. Each of the sub-sampled sizes is tested 100 times.



(b) Decoding time comparison between Erlich's work and our work. The decoding time is on a logarithmic scale.

Figure A.2: Performance comparison between our work vs Erlich's[7] work

A.2.2 Experimental

To further evaluate the correctness of our algorithm we performed in-vitro experiment. We began by storing a JPEG file with a size of 13,170 bytes (see Figure A.3) into 604 DNA sequences, each consisting of 250 nucleotides. Our algorithm was employed to convert the JPEG file into DNA sequences. To facilitate sequencing, a forward primer (5-ACATCCAACACTCTACGCCC-3) and a backward primer (5-GACTACGAGTAGCGCGACGT-3) were attached to each sequence. The DNA sequences were then ordered as an oligo pool from Integrated DNA Technologies (IDT). Subsequently, we sent the synthesized oligo pool to Genewiz for sequencing. Genewiz only had access to the forward and backward primers required for sequencing and was not provided with any other information regarding the oligo pool. As a result of the sense and antisense strands, we received two distinct read files from Genewiz. The frequency distributions of these two files were almost identical (see Figure A.6).

Figure A.3: JPEG file that we synthesized and sequenced for our experiment

This process yielded approximately 78 million sequence reads, with around 5 million of them being unique (see Figure A.4). Out of the 604 sequences in our library, we observed 602 sequences in the pool. This implies a sequence dropout of approximately 0.33%, which is 4 times lower than the approximately 1.3% dropout reported by Erlich and Zielinski [7]. This lower dropout rate highlights the efficiency of our hex-codon mapping scheme, allowing us to avoid error-prone sequences more effectively. Even with the loss of two sequences during the entire process, we were still able to recover the entire file without requiring any manual intervention. This successful recovery can be attributed to the robustness of the fountain code utilized in our algorithm.

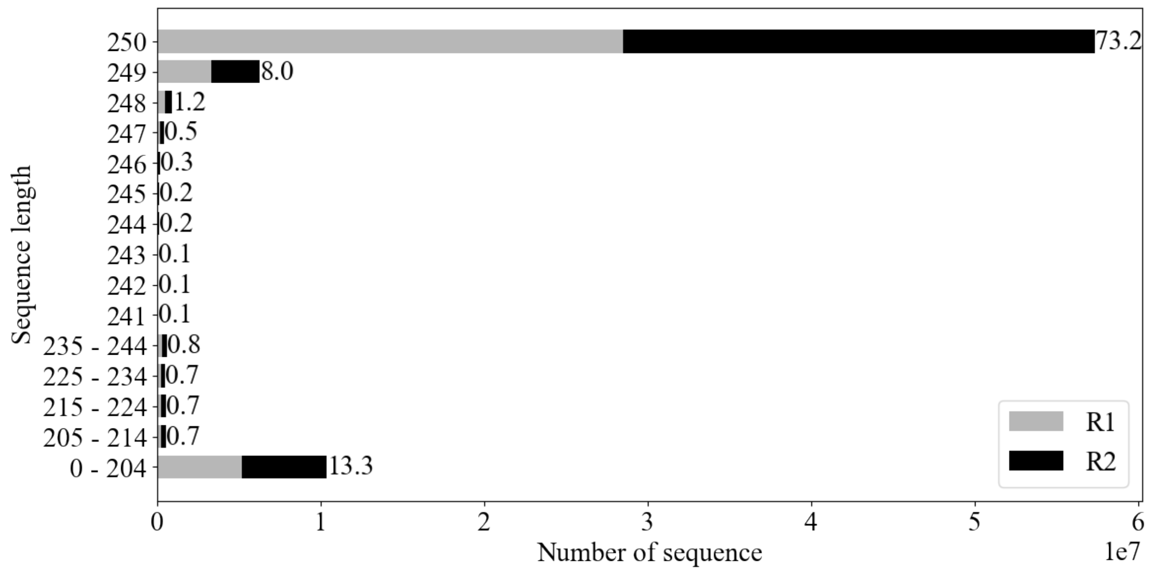


Figure A.4: Sequence frequency distribution among two reads. In total we got 78 million reads where 5 million reads are unique. Out of all the reads 62% of the reads are correct

To further assess the robustness of our algorithm, we conducted tests using different-sized random samples (5,000-20,000 samples) from the 78 million reads. These samples were passed through our decoder to recover the original message. With a random subsampled size of 7,000 reads, our decoder achieved a file recovery rate of 90%. Moreover, for sample sizes exceeding 9,000 reads, our decoder achieved a 100% success rate (see Fig A.5). Each of these tests was repeated 100 times to ensure statistical reliability. We determined that a larger subsampled size was necessary due to the presence of read repetition. For instance, when using 10,000 random sub-sampled sequences, we observed a duplication rate of 50.82±0.42%.

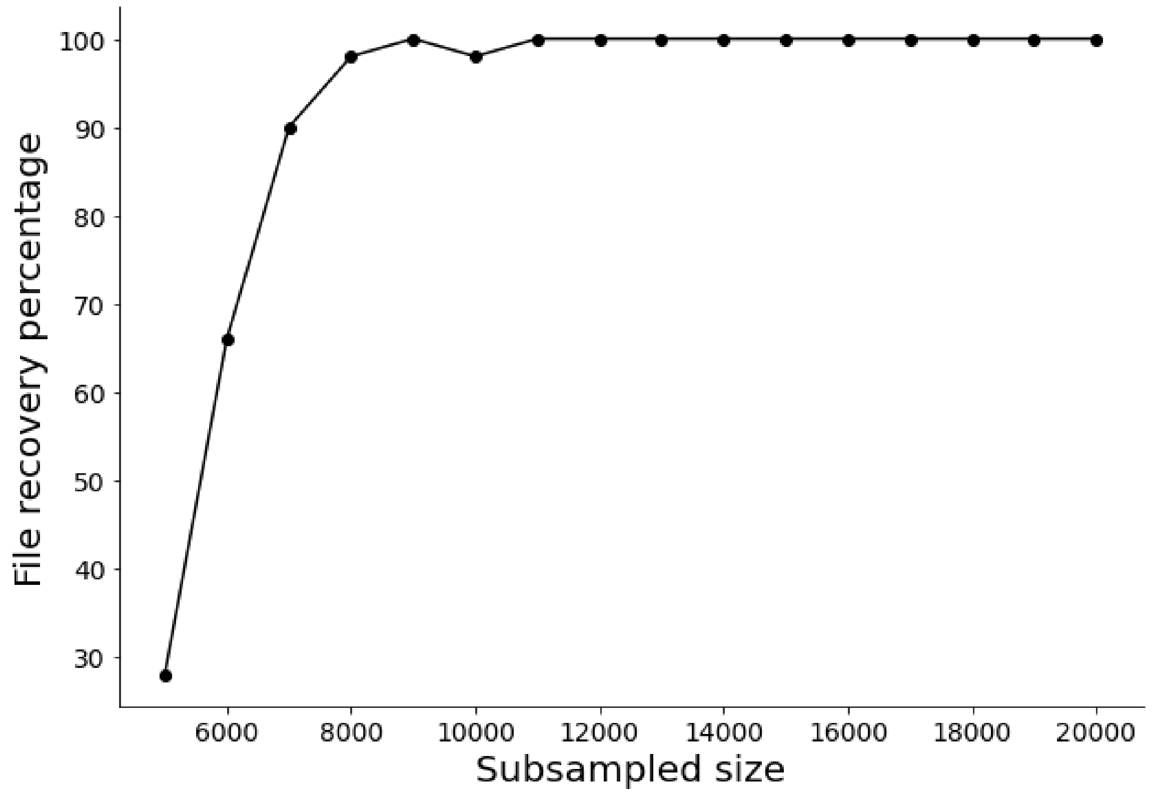


Figure A.5: File recovery percentage by only using a sub-sample of reads from overall reads. Each of the sub-sampled sizes is tested 100 times.

Additionally, individual reads exhibited repetitions of up to approximately 80,000 times (see Figure A.6). Although some sequences exhibited higher frequencies than others, no discernible correlation was found when analyzing the secondary structures of these sequences. This indicates that the discrepancy in read frequency and the loss of the two sequences cannot be explained by structural factors.

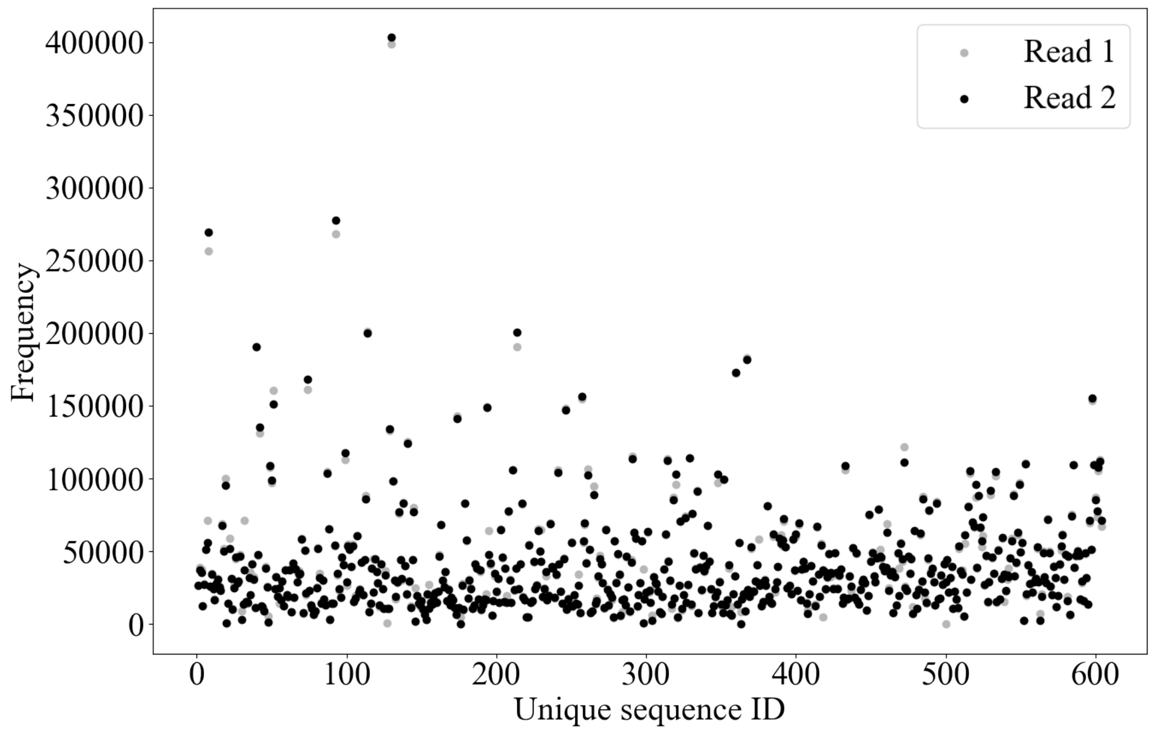


Figure A.6: Frequency of sequences from our library. X-axis shows the 604 sequences that we synthesized. And Y-axis shows how many times that individual sequence appeared in the two different read file.

Upon examining all the sequences, we determined that 48 million reads (corresponding to the repetition of 602 sequences) were error-free, while 7 million sequences contained a single error (see Figure A.7). The number of errors was calculated using the minimum edit distance technique.

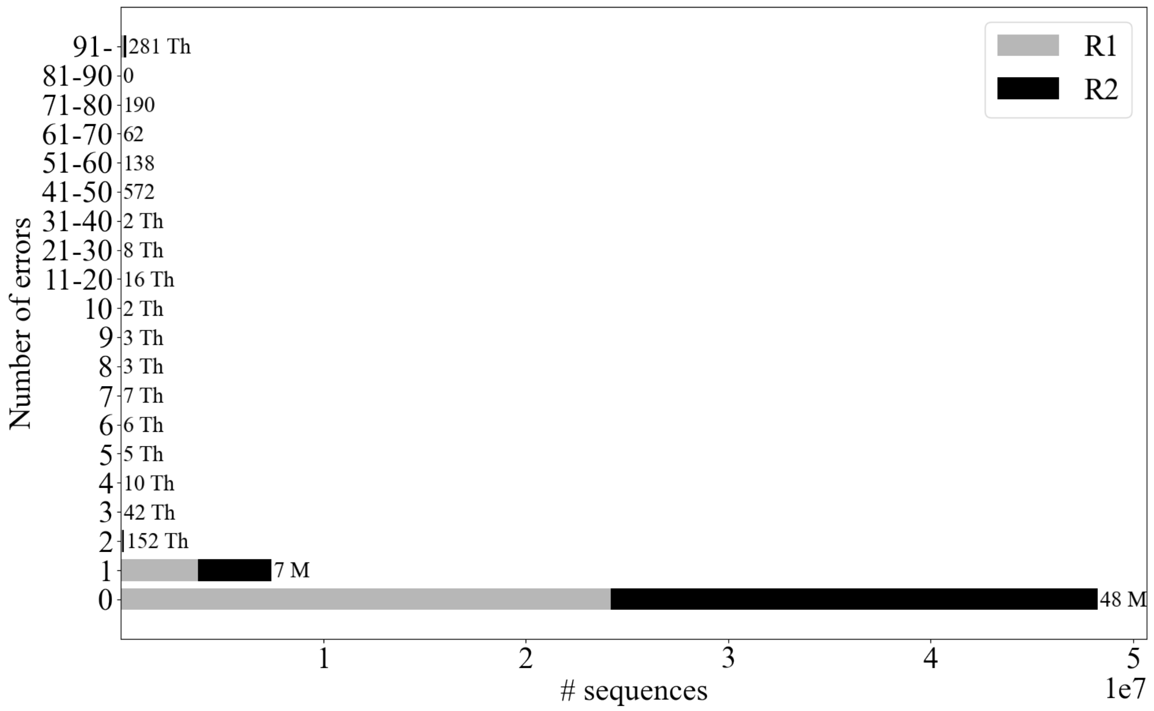


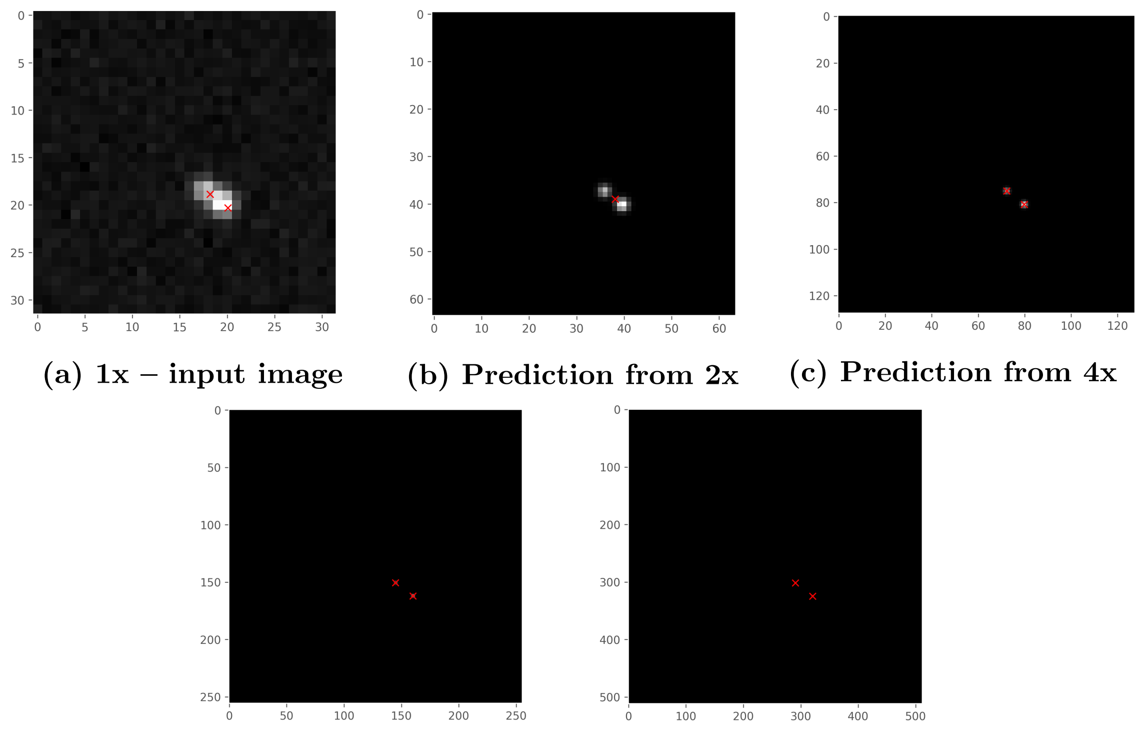
Figure A.7: Error distribution among two reads. 48 million of the sequence had no error. Where 7 million sequences had 1 error. Number of errors are the minimum edit distance (insertion, deletion, mutation) between a single read to any of the target sequences. Where the target sequences are our known library of sequences that we synthesized.

A.3 Discussion

Storing information using the new method was observed to increase dropout tolerance, decrease encoding time, and decrease decoding times. Although our method exhibits a lower information density compared to alternative encoding methods, the advantages of our method make it more appropriate for applications where information recovery is critical, where elevated dropout is expected, and/or where computational resources are limited. The new method additionally contains mechanisms for limiting sequences of repeated bases and eliminating biologically relevant sequences. The former of these is expected to help mitigate synthesis and sequencing errors. The later of these is expected to help mitigate interaction with living cells, preventing an added layer of security against biological interaction.

APPENDIX B:
VISUAL EXAMPLE OF LOCALIZATION ALGORITHM

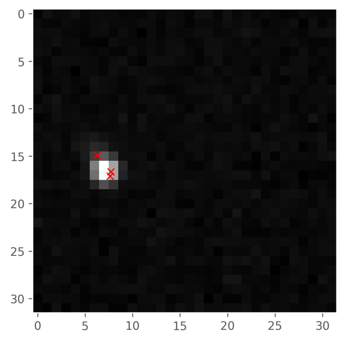
The following figures shows the example of emitter localization at varying resolution(2x, 4x, 8x, 16x) using our algorithm(see Chapter 3) The presented images demonstrate the occurrence of information loss during the upsampling process. The red 'x' symbols denote the true positions (a) or predicted positions (b, c, d, e) of the emitters.

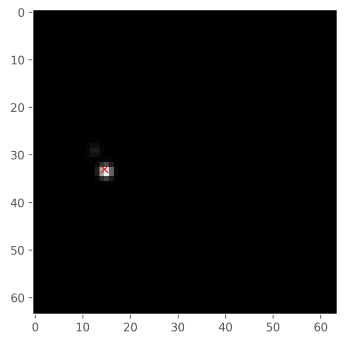


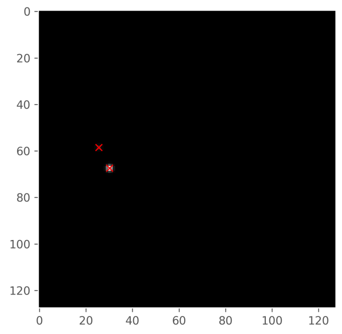
(a) 1x – input image **(b) Prediction from 2x** **(c) Prediction from 4x**

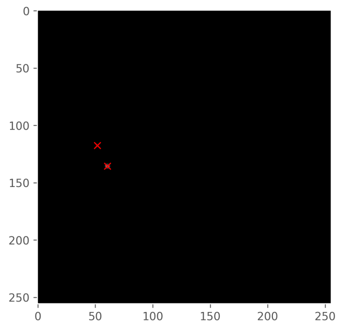
(d) Prediction from 8x **(e) Prediction from 16x**

Figure B.1


(a) 1x – input image


(b) Prediction from 2x


(c) Prediction from 4x


(d) Prediction from 8x

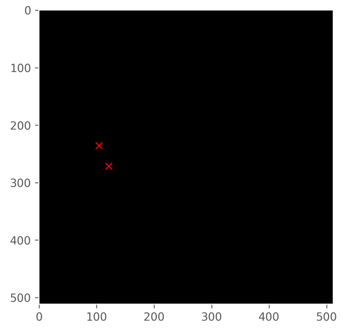
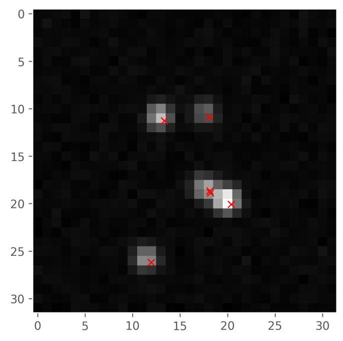
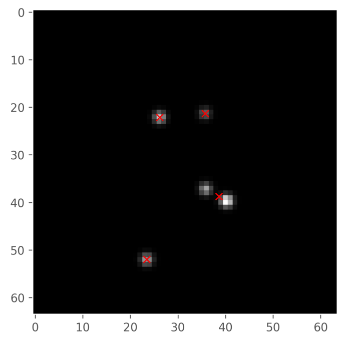

(e) Prediction from 16x

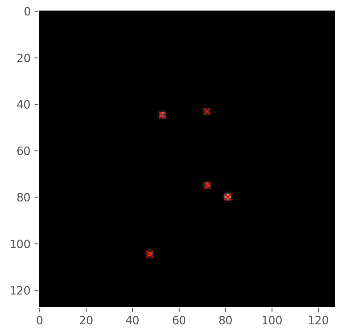
Figure B.2



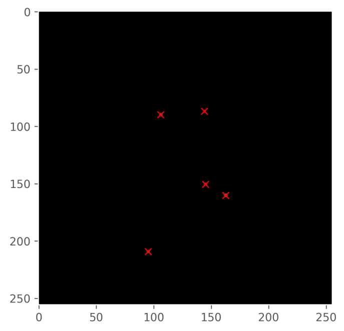
(a) 1x – input image



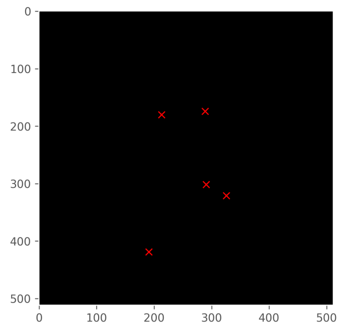
(b) Prediction from 2x



(c) Prediction from 4x

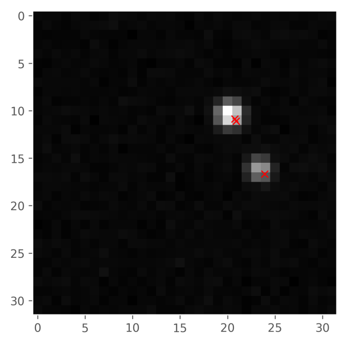


(d) Prediction from 8x

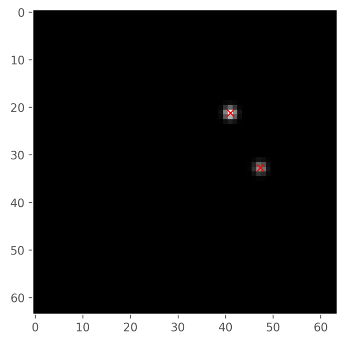


(e) Prediction from 16x

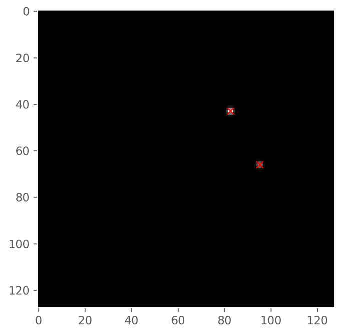
Figure B.3



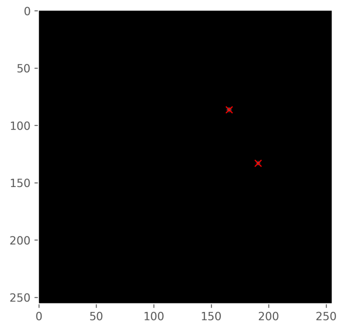
(a) 1x – input image



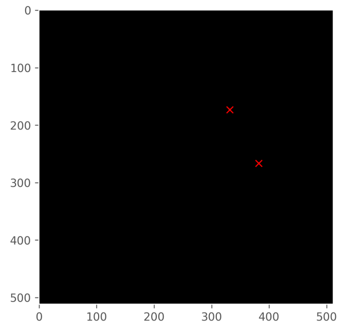
(b) Prediction from 2x



(c) Prediction from 4x

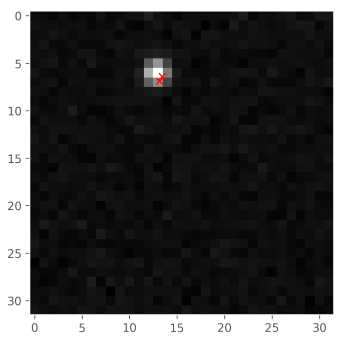


(d) Prediction from 8x

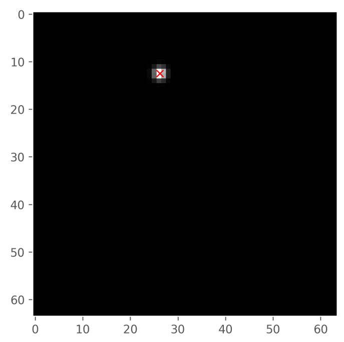


(e) Prediction from 16x

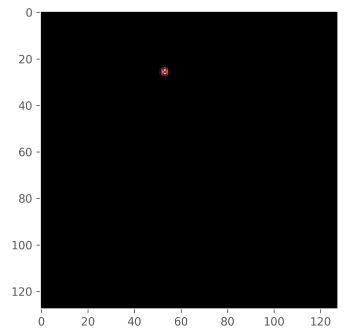
Figure B.4



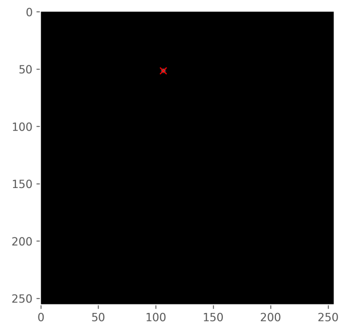
(a) 1x – input image



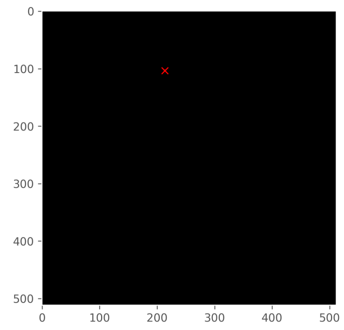
(b) Prediction from 2x



(c) Prediction from 4x



(d) Prediction from 8x



(e) Prediction from 16x

Figure B.5