Deep Science Publishing
https://doi.org/10.70593/978-93-49910-92-8_5

Chapter 5: Turing machine in DNA computation

Nandini Rao, Tarun Kumar Gupta

Abstract: This chapter examines how DNA computing can simulate Turing Machines (TMs) by employing DNA strands to represent states, symbols, and transitions. Turing Machines serve as a model for understanding the limits of computation. DNA computing, demonstrated by Adleman in 1994 through the Hamiltonian Path Problem, takes advantage of biochemical reactions such as hybridization and ligation to mimic TM operations.

Keywords: Adelman's experiment, DNA, Hamiltonian Path problem, Molecular Algorithms, Turing

1 Introduction

Named after Alan Turing in 1936, the Turing machine was first described as a simple abstract computing device that helps you explore the scope and limitations of what can be calculated. Turing's automatic machinery was specially developed to estimate actual numbers, as explained in 1936. These are some of the fundamentals of theoretical computer science.

Nandini Rao
Department of Mathematics, Chandigarh University, Punjab, India.

Tarun Kumar Gupta
Department of Mathematics, Chaman Lal Mahavidyalaya, Landhaura, Uttarakhand, India.

Alan Turing developed this machine to pave the way for computer science with knowledge of cryptography and mathematics and to explain devices that can perform calculations. (L. De Mol , 2024) The formal definition is given by:

A Turing machine can be specified with 7 tuples: (Q, ∑, T, q0, F, B, δ) where:

Q: The finite set of state

∑: the finite set of input symbols

T: the tape symbol

q0: the initial state

F: a set of final states

B: a blank symbol used as an end marker for input

δ: a transition or mapping function.

The mapping function shows the mapping from states of finite automata and input symbols on the tape to the next states, external symbols, and the direction for moving the tape head. This is known as a triple or a program for Turing machines.

(q0, a) → (q1, A, R). This means if we read the symbol 'a' in the q0 state, we will go to state q1, replace a with A, and the head will point to the right (R stands for right).

Basic model of Turing Machine:

i. The input tape has infinite cells, each containing one input symbol; thus, the input string can be placed on the tape. The empty tape is filled with blank characters.
ii. The finite control and the tape head are responsible for reading the current input symbol. The tape head can move from left to right.
iii. A machine must undergo a finite set of states.
iv. A finite set of symbols called external symbols is used to build the logic of the Turing machine.

Various features of the Turing Machine:

i. It has external memory that remembers arbitrarily long sequences of i/p.
ii. It has unlimited memory capacity
iii. The i/p at left or right on the tape can be read easily
iv. The machine can produce a certain o/p based on i/p.

Like finite automata, Turing Machines work in non-deterministic settings as well. A non-deterministic Turing machine is defined by 9 tuples (Q, Σ, T, δ, ⊢ , ∪, s, t, r) where:

Q: The finite set of states

∑: The finite set of input symbols

T: The tape symbol

s or q0: The initial/start state

F or t: A set of final/accepting states

r: A rejection state or set of rejecting states

B: A blank symbol used as an end marker for input

δ: A transition or mapping function.

However, we can group these into the original seven.

These types of Turing machines are known as 'choice machines' a term coined by Turing himself. The execution of this machine does not depend completely on the input and the transition function, but the machine has the flexibility to choose from a set of transition functions. While the nondeterministic Turing machine shares the same components as that of a deterministic one, the difference comes in its transition function - δ: $Q \times T \rightarrow 2^{Q \times T \times \{-1,+1\}}$. Further, to understand the difference between deterministic and non-deterministic Turing machines, we can consider the existence of non-deterministic polynomial time (NP) problems. Polynomial time refers to the amount of time (i.e., number of computational steps) an algorithm or Turing Machine takes to solve a problem, where the time grows at most like a polynomial function of the size of the input. (Kumar, T., and Namasudra, S, 2023)

To highlight the difference between Deterministic Turing Machines (DTMs) and Nondeterministic Turing Machines (NTMs) in the context of DNA computing, consider the problem of finding a path in a graph. Given a graph and two vertices, the task is to determine whether a path exists between them. (Tiwari, K. K., Singh, A., & Kumar, S., 2025)

A DTM solves this problem systematically by sequentially exploring all possible paths one by one until it either finds a valid path or exhausts all options. This approach operates in polynomial time, but its sequential nature can make it computationally expensive.

An NTM, however, can solve this problem more efficiently by non-deterministically guessing a potential path and verifying its correctness in polynomial time. Instead of checking paths one after another, an NTM can explore all possible paths simultaneously, increasing the likelihood of quickly finding a valid solution. (Sharma, S., Rana, S., & Dubey, S. S., 2024)

This concept directly relates to DNA computing, which inherently exhibits massive parallelism—a property like how an NTM operates. In other words, the DNA-based

Turing Machine simulates the behaviour of the NTM. In Adleman's 1994 DNA computing experiment, DNA strands were used to represent all possible paths in a graph, and biochemical reactions (such as hybridization and ligation) were employed to find the correct path. Just as an NTM explores multiple possibilities at once, DNA molecules process numerous solutions in parallel, making DNA computing a natural physical implementation of nondeterministic computation. This experiment (based on the HPP) is an example of an NP since finding all permutations of paths is exponential in time, which is non-polynomial time. (Kumar, M., Dubey, S. S., Gupta, P., & Khandelwal, Y., 2020)

Thus, while traditional DTMs struggle with problems requiring extensive exploration, DNA computing leverages the parallel nature of molecular reactions to solve problems more efficiently—mirroring the behaviour of NTMs in theoretical computation.

1.1. Construction of TM:

To construct a Turing machine, we require a language, markers, reject and accept states, and directions for reading the tape (left or right).

Ex: Construct a TM that accepts the language = {a, b}
We will assume the string 'abaΔ' is placed on the input tape. The tape will read the characters up to Δ characters. If the tape has read the desired string, the Turing machine will halt after reading and will reach a halt or an acceptance stage.

At the start, we have q0 as the initial state, and the head points to 'a'. We will apply the transition function to move around- (q0, a) = (q1, A, R). This means we will read 'a' in q0 and move right, denoted by R, to q1 while replacing 'a' with 'A'.

Now, the head points are on 'b.' Similar to the previous cell, we will apply (q1, b) = (q2, B, R). Then we move on to q3. Next, we apply (q2, a) = (q3, A, R). The machine has reached the character through (q3, Δ) = (q4, Δ ,S)

Table 1.1 Transition Table of Turing Machine

States	A	B	Δ
q0	(q1, A, R)	-	-
q1	-	(q2, B, R)	-
q2	(q3, A, R)	-	-
q3	-	-	(q4, Δ, S)
q4	-	-	-

This transition table can be represented in a transition diagram too.

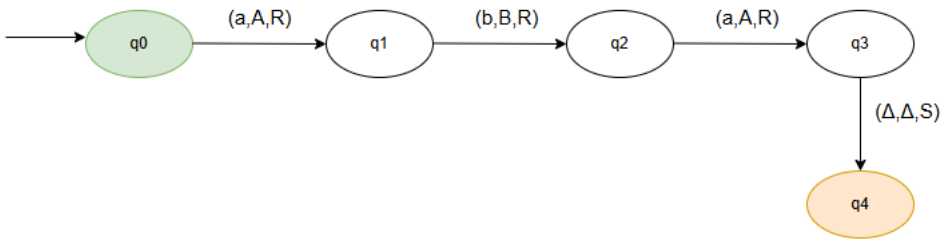

Fig 1.1 Transition diagram of Turing Machine

2 DNA Computing (Kari, L., Seki, S., and Sosík, P., 2012)

The information in the DNA is represented using the four-character genetic alphabet- A [adenine], G [guanine], C [cytosine], and T [thymine]. These are known as bases and are linked through *deoxyribose.* This connection between the bases follows a direction such that the lock-and-key principle is achieved -

(A)(T) and (C)(G)

This means Adenine (A) pairs with Thymine (T), and Cytosine (C) pairs with Guanine (G). These bases make DNA a computational medium.

Drawing the analogy from traditional computers to DNA computing, we see that while the former process formation sequentially, DNA computing allows for parallelism, the ability to perform many computations simultaneously by leveraging the vast number of DNA molecules. This significantly speeds up the process.

DNA computing uses biochemical reactions and DNA molecules instead of silicon-based computing like conventional computers. The analogy between the two is outlined below:

Table 1. 2 Difference between Traditional Computing and DNA Computing

Traditional Computing	**DNA Computing**
Bits (0s and 1s)	DNA bases (A, T, G, C)
Circuits	DNA Strands
Logic gates (AND, OR, NOT)	Biochemical operations (Enzymes and hybridization reactions)
Memory (RAM)	DNA storage

DNA computing follows stochastic processing. When DNA molecules are added to a test tube to obtain a desirable set(s) of output(s), they naturally combine in various ways without further intervention in the tube and present potential solutions.

2.1 Properties of DNA computing

(i) Molecular Algorithm

DNA computing relies on the use of nucleic acids (DNA and RNA) to perform molecular computing, i.e., carrying out operations at a molecular scale, which in turn allows for massive parallelism.

Molecular algorithms occur from molecular operations. Some of these are listed below, along with the algorithm they induce. These operations do not follow a step-by-step occurrence; they all happen at once.

Table 1.3 Molecular Algorithms

Molecular Operation	Algorithmic equivalent	Description
Hybridization	Pattern matching	Base pairing between complementary strands (A–T, C–G)
Ligation	Concatenation	Enzyme ligase merges DNA strands, creating new sequences
Strand Displacement	Memory overwrite	Replace one strand with another
Enzyme cutting	Conditional branching	Cuts the DNA at specific sequences
Gel electrophoresis	Sorting or selection	Separates DNA by size

(ii) Parallel Processing

Parallelism is a term that refers to utilising multiple processing units simultaneously executing tasks to speed up the process. DNA computing naturally follows this property, making it a suitable choice to solve problems involving numerous potential results. (Singh, M. K., & Kumar, S. 2024)

Parallel processing is done when DNA strands start to interact through natural chemical processes (molecular operations). These interactions provide all possible combinations to our problem. We add a biological filter (like adding enzymes) that omits the incorrect combinations and keeps the ones that meet the criteria of the problem. The last DNA strand remaining (survived all the filters) is the final solution to our problem.

Example:

Say we want to find out whether a person is infected with a certain virus or not. The virus is known to leave **3 specific DNA markers** in the patient's cells:

Marker A, Marker B, and Marker C.

We know the virus is present if all three markers are present. An 'AND' logic gate would help in this scenario.

- We create sensors (synthetic DNA strands) that bind to all three markers.
- We then add a sample of the person's DNA to these strands. If the markers are present, the probes will hybridize (bind) to them.
- The DNA-based AND gate will produce a visual indication for a positive result (like a visible color change) if all three strands bind successfully.
- If one or more markers are missing, nothing happens, i.e., the result is negative.

(iii) Self-Replication

Self-replication means that DNA can make exact copies of itself using enzymes and basic building blocks (nucleotide bases). This process occurs naturally in all living beings. This process begins when the DNA strand starts to unwind with the help of an enzyme called *helicase.* Consequently, the double helix is broken into two single strands. The free-floating nucleotides match up with the now exposed bases on the strands to synthesize new sequences in the presence of DNA polymerase (an enzyme that helps attach new bases, forming two identical DNA molecules from the original one).

2.2 Adleman's Experiment (Rozen, D. E., McGrew, S., & Ellington, A. D., 1996)

This experiment, conducted by Dr. Leonard M. Adleman, was the first DNA computing experiment. Its goal was to find a route through the network of cities (1-7) associated with disposable roads. The problem shows that routes must start and end in a particular city and only visit all cities once. (This is also known as the Hamiltonian Path issue.)

A directed graph G with designated vertices vstart and vend is said to have a Hamiltonian path if and only if there exists a sequence of compatible directed edges e1; e2; ...; ez (i.e., a directed path) that begins at vstart, ends at vend, and enters every other vertex exactly once. The following (nondeterministic) algorithm solves the problem:

Input. A directed graph G with n vertices and designated vertices vstart and vend.

Step 1. Generate random paths through the graph.

Step 2. Keep only those paths that begin with vstart and end with vend.

Step 3. Keep only those paths that enter precisely n vertices.

Step 4. Keep only those paths that enter all the vertices of the graph at least once.

Output. If any paths remain, output ‘‘YES’’; otherwise, output ‘‘NO.’’

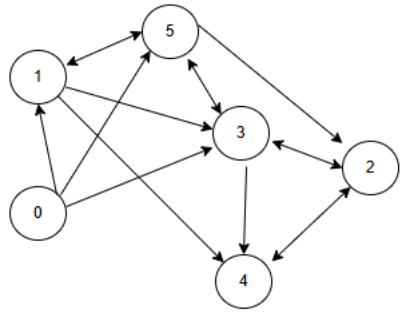

Fig. 1.2 Adleman's Node graph

2.3 Example of Adleman’s experiment

Consider the graph below:

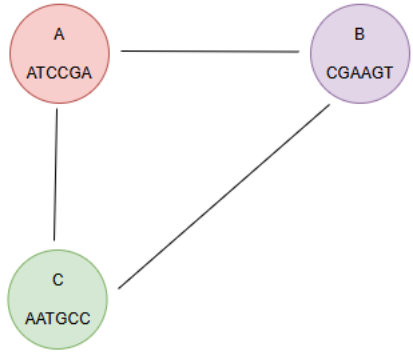

Fig. 1.3 Directed graph with three nodes

Here, we can see that each node has its unique DNA sequence. Right now, we have not specified the path we want to find, so all possibilities among the three nodes are possible.

Say our desired path is $A \rightarrow B \rightarrow C$. This forms a directed graph.

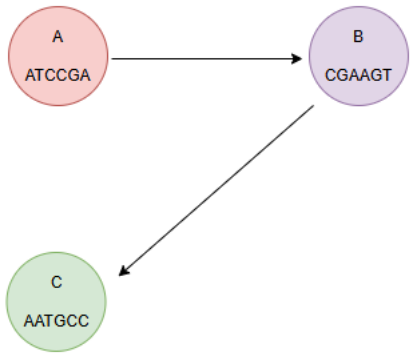

Fig. 1.4 Directed graph with $A \rightarrow B \rightarrow C$ path

To come up with a sequence that abides with our required path, we need the sequence for each of the two edges that combine the nodes. To do so, we understand the directionality of DNA strands (each DNA strand has a chemical orientation). (Kumar, S., 2024)

We follow these steps:

1. Write the sequence at hand that reads from 5' (five prime) to 3' (three prime) according to the DNA orientation. This means it can only synthesize when used in 5' to 3' form. Consider the sequence 5'—ATGCCT—3'.

2. Find the complement of the sequence, which is read in 3' to 5' form. The complement of the strand is 3'—TACGGA—5'.

3. The reverse complement, i.e., write the complement in 5' to 3' form. The reverse complement is AGGCAT.

4. Finally, we have a sequence that binds with our original sequence.

Coming back to our example, let us find the sequence of the edge that joins A and B. We take the last part of A's sequence and the first part of B's sequence and find and combine their complements.

Last part of A	First part of B	Reverse complement of A	Reverse complement of B	Combined strand
CGA	CGA	TCG	TCG	TCGTCG

So, $A \rightarrow B = TCGTCG$.

Similarly, we find for $B \rightarrow C$.

Last part of B	First part of C	Reverse complement of B	Reverse complement of C	Combined strand
AGT	AAT	ACT	ATT	ACTATT

So, $B \rightarrow C = ACTATT$.

To get the collective strand representing $A \rightarrow B \rightarrow C$, we combine the nodes with edges and eliminate the nucleotides that bind.

We keep **the entire node A**, the last part of $A \rightarrow B$ since the first part overlaps with the end of A**, the last part of node B** since the first part overlaps with $TCGTCG$, **the last part of $B \rightarrow C$** since the first part overlaps with the end of B, and **the last part of C** since the first part overlaps with the end of $ACTATT$.

So, our final sequence is – ATCCGA TCG AGT ATT GCC

While working in practical conditions, this path would have appeared after DNA molecules would have ruled out other possibilities that did not include strands that followed $A \rightarrow B \rightarrow C$. For instance, if we want paths from A to C, the DNA molecules would operate such that they pull out strands starting with A's sequence and keep the ones that end with C's sequence.

We can claim that Adleman's experiment used the property of DNA parallelism, i.e., processing multiple possibilities simultaneously.

3. Turing Machine Implementation via DNA

A DNA computation mimics the operations of a TM. DNA computing can simulate a TM by performing the same read, write, and move operations in a biological system. (Van Nies, P., Westerlaken, I., Blanken, D., 2018)

Table 1.4 Similarities between Turing Machines and DNA Computing

Components of Turing Machine	**Equivalent Components of DNA Computing**
Tape (memory storage)	DNA sequence (A, T, C, G strands)
Symbols (0, 1, blank)	Specific DNA sequences
Read/Write head	Enzymes (cut, copy, and modify DNA)
State Transitions	Chemical reactions
Computation process	DNA hybridization, ligation, and cleavage

Table 1.5 Transition table of Turing Machine converting 0 to 1 and 1 to 0.

Current state	**Reads symbol**	**Writes Symbol**	**Direction**	**Next state**
q0 (start)	0	1	R	q1
q0	1	0	R	q1
q1	0	1	R	q1
q1	1	0	R	q1
q1	_	_	stops	qf (halt state)

Table 1.6 Equivalent transition table of DNA Computing

Current state	**Reads Sequence**	**Writes Sequence**	**Direction**	**Next state**
q0 (start)	AAC	GTT	R	q1
q0	GTT	ACC	R	q1
q1	AAC	GTT	R	q1
q1	GTT	AAC	R	q1
q1	_	_	stops	qf (halt state)

3.1 DNA-based Turing Machine

The motivation behind a DNA-based Turing machine is to show proof of principle that molecules can carry out symbolic computations. The Turing machine is the fundamental model of computation and gives DNA computing a formal theoretical foundation. A DNA-based TM forms the blueprint of how molecules follow logic. It allows us to run general logic without redesigning a new circuit for every task. (Jonoska, N., & Winfree, E., 2023)

Example:

An application of a DNA-based Turing Machine is that of biological simulations (like protein synthesis in cells). Say we want to model gene repression via a

repression protein on a Turing machine. Before we begin, let us define a basic logic we will follow:

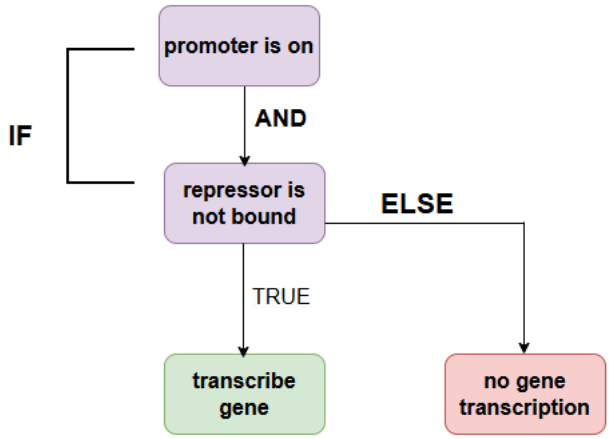

Fig. 1.5 Flowchart of logic used in DNA-based Turing Machine

Consider the gene that controls eye colour: OCA2. This gene depends on HERC2 mutation. When HERC2 is on, normal gene regulation occurs; otherwise, it does not.

Table 1.7 Transition table for biological simulation in Turing Machine

Current state	Read symbol	Write symbol	Move	Next state	Description
q0	HERC2_ON	HERC2_ON	R	q_checkOCA2	Check OCA2 regulation
q0	HERC2_OFF	HERC2_OFF	R	q_blockOCA2	Mutation blocks expression
q_checkOCA2	OCA2	OCA2	R	q_brown	Gene active
q_blockOCA2	OCA2	OCA2	R	q_blue	Gene repressed
q_brown	Eye_colour	BROWN	-	q_accept	O/p eye colour
q_blue	Eye_colour	BLUE	-	q_accept	o/p eye colour

3.2 Instance of DNA-based Turing Machine

There are multiple ways we can find correspondence between Turing Machines and DNA computing. One such way is through restriction enzymes, proteins that cut DNA at specific sequences. The core idea is to represent the Turing machine's tape as a single-stranded DNA molecule composed of specific sequences (domains), where each domain encodes a symbol or a state. These DNA sequences act as the data and control units of the Turing machine.

Each combination of a current state and symbol (i.e., the current position of the machine) corresponds to a unique recognition site for a specific enzyme. When the enzyme is introduced, it recognizes the matching site on the DNA strand and cleaves it at a specific location. This cut effectively simulates a transition: it modifies the structure of the DNA in a way that corresponds to changing the symbol, moving the tape head, and updating the state.

By designing these DNA sequences carefully and choosing appropriate enzymes, the system can follow a series of programmed transitions just like a Turing machine. After a series of enzymatic reactions, the resulting DNA strand represents the final configuration of the tape and the state of the machine. This approach effectively implements computational logic using biochemical reactions, demonstrating how molecular systems can be used to carry out formal computations.

4 Challenges & Future Directions

Implementation. Scalability issues arise because the quantity of DNA strands needed for computing grows exponentially as a function of problem size, making large calculations resource-intensive and time-consuming. In addition, excessive error caused by hybridization mismatch, degradation of DNA, and molecular noise may lead to incorrect results, giving rise to error correction technologies like redundancy, proofreading enzymes, and precision editing through CRISPR-based techniques. (Kumar, S., & Kour, G., 2025) Cost restrictions remain a hindrance mainly because DNA synthesis, sequencing, and lab automation are still expensive and not yet accessible to ordinary use.

To solve these problems, researchers are working on hybrid DNA-silicon computing, which will utilize the tremendous parallelism of DNA combined with the high speed and reliability of traditional silicon-based processors. Error correction techniques such as self-healing strands of DNA and molecular proofreading enzymes are under investigation to increase reliability. Parallel processing of DNA is being optimized to speed up molecular reactions and make them more efficient. (Kumar, M., Gupta, M. K., Mishra, R. K., Dubey, S. S., Kumar, A, 2021)

Upcoming breakthroughs in DNA computing hold significant promise in various areas. Data storage based on DNA is evolving as a revolutionary method for ultra-dense and long-term storage of data, with major tech companies making research investments to allow DNA to serve as a viable alternative to traditional storage. AI-driven DNA neural networks have the potential to be responsible for creating molecular computing devices that can learn and evolve. In biomedicine, programmable DNA nanorobots could be used for targeted drug delivery, early

diagnosis of diseases, and even intelligent therapeutics that adapt based on biological signals inside the human body. As synthetic biology and nanotechnology continue to advance, DNA computing has the potential to revolutionize cryptography, AI, big data processing, and personalized medicine, potentially changing the future of computing and biotechnology.

Conclusions

Turing Machines have, to date, served to be foundational to the theory of computation. It has applications beyond the study of computation; it is profoundly used in bioinformatics. DNA computing simulates Turing Machines (TM) by using DNA strands to represent the tape, symbols, and states. Enzymes act as the read/write head, and biochemical reactions (like hybridization and ligation) simulate state transitions. Just as a Turing Machine processes symbols based on rules, DNA computing processes DNA sequences using base pairing rules (A-T, C-G). This allows DNA computing to solve complex problems through massive parallelism and high data storage capacity. By initiating this relationship with Turing machines, DNA computation allows us to approach various problems, including the Hamiltonian path problem. Overall, DNA computing follows the same logic as a Turing Machine, with biochemical reactions replacing the TM's symbolic processing.

References

Kari, L., Seki, S., and Sosík, P., "DNA Computing — Foundations and Implications." Handbook of Natural Computing, vol. 33, no. -, pp. 1073-1127, 2012, doi: 10.1007/978-3-540-92910-9_33.

Kumar, S. (2024, May). Advancements in meta-learning paradigms: a comprehensive exploration of techniques for few-shot learning in computer vision. In *2024 International conference on intelligent systems for cybersecurity (ISCS)* (pp. 1-8). IEEE.

L. De Mol, "Turing Machines," in Stanford Encyclopedia of Philosophy, Metaphysics Research Lab, Stanford, 2024.

Kumar, T., and Namasudra, S., Advances in Computers. vol. 129. Elsevier, 2023.

Rozen, D. E., McGrew, S., & Ellington, A. D. (1996). Molecular computing: Does DNA compute? Current Biology, 6(3), 254-257.

Kumar, S., Rampal, S., Gaur, M., & Gaur, M. (2024, March). Advanced ensemble learning approach for asthma prediction: Optimization and evaluation. In *2024 International Conference on Automation and Computation (AUTOCOM)* (pp. 283-288). IEEE.

Van Nies, P., Westerlaken, I., Blanken, D. et al. Self-replication of DNA by its encoded proteins in liposome-based synthetic cells. Nat Commun 9, 1583 (2018).

Tiwari, K. K., Singh, A., & Kumar, S. (2025, February). A Comprehensive Analysis of CNN-Based Deep Learning Models: Evaluating the Impact of Transfer Learning on Model Accuracy. In *2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)* (pp. 62-67). IEEE.

Sharma, S., Rana, S., & Dubey, S. S. (2024). ESAF: An Enhanced and Secure Authenticated Framework for Wireless Sensor Networks. *Wireless Personal Communications*, *136*(3), 1651-1673.

Kumar, M., Dubey, S. S., Gupta, P., & Khandelwal, Y. (2020). ImproveSsecurity of Quantum Proxy Signature Scheme using Quantum One-, Way Function and Bell States, Vol. 9, *Advances in Mathematics: Scientific Journal*.

Kumar, M., Gupta, M. K., Mishra, R. K., Dubey, S. S., Kumar, A., & Hardeep. (2021). Security Analysis of a Threshold Quantum State Sharing Scheme of an Arbitrary Single-Qutrit Based on Lagrange Interpolation Method. In *Evolving Technologies for Computing, Communication and Smart World: Proceedings of ETCCS 2020* (pp. 373-389). Springer Singapore.

Jonoska, N., & Winfree, E. (2023). Visions of DNA Nanotechnology at 40 for the Next 40. Springer Nature.

Eghdami, H., & Darehmiraki, M. (2011). Application of DNA computing in graph theory. Artificial Intelligence Review, 38(3), 223-235. https://doi.org/10.1007/s10462-011-9247-5

Kumar, S., & Kour, G. (2025, March). Advanced Machine Learning Approaches for Fastag Fraud Detection. In *2025 International Conference on Automation and Computation (AUTOCOM)* (pp. 149-154). IEEE.

Singh, M. K., & Kumar, S. (2024, April). Stress Detection During Social Interactions with Natural Language Processing and Machine Learning. In *2024 International Conference on Expert Clouds and Applications (ICOECA)* (pp. 297-301). IEEE.