

# **DNA Sequence Manipulation and Analysis Tool**

A Project Presented to

Dr. Wendy Lee

San Jose State University

In Partial Fulfillment

of the Requirements of the Class

Spring 2025: CS 122 (Advanced Python)

Maria Kuznetsova, Rongjie Mai

May 7th, 2025

## Table of Contents

A Project.....	1
Tools and Libraries.....	3
Program Features.....	3
Installation and Environment.....	3
Project Results.....	4
1. Cropping.....	4
2. Pattern Searching.....	4
3. Sequence Reversal.....	4
4. DNA to Protein Translation.....	4
5. Mutation Detection.....	4
6. Reverse Complement.....	4
Sequence reversal function helps in implementing the reverse complement where A's and T's, G's and C's are swapped and the sequence is then reversed.....	4
7. User Interface.....	4
Translation Logic.....	5
Mutation Detection Logic.....	6
Challenges Faced.....	6
Changes from the Original Plan.....	7
<b>Appendix.....</b>	<b>7</b>
A. Running the Program.....	7

## Introduction

DNA sequence analysis is a critical task in biological research and education. Existing bioinformatics software can be complex, resource-heavy, and often inaccessible to beginner or intermediate users. Our project aims to address this gap by developing a lightweight, easy-to-use tool for DNA sequence manipulation and basic analysis. The tool is designed for students, educators, and researchers who need to perform quick DNA tasks such as cropping sequences, pattern searching, reversing sequences, translating DNA to proteins, and detecting mutations.

The initial phase focuses on building a graphical user interface (GUI) that offers intuitive interaction. Python was selected for its wide support in scientific computing, and key libraries such as `re` were used to streamline DNA manipulation and pattern matching tasks.

## Methods and Materials

## Tools and Libraries

- **Python 3.13** – Main programming language.
- **Regular Expressions (re)** – For pattern matching and search functions.
- **Virtual Environment (venv)** – For dependency management.
- **GitHub Repository** – For version control and collaboration.

## Program Features

The tool currently supports the following functionalities:

1. **Cropping**: Allows users to specify start and end positions to extract subsequences.
2. **Pattern Searching**: Enables users to search for DNA motifs or patterns using regular expressions.
3. **Sequence Reversal**: Reverses the DNA strand for complementary analysis.
4. **Reverse Complement**: Replaces A to T, T to A, C to G, G to C and reverses the sequence.
5. **Translation to Protein**: Converts DNA sequences into amino acid sequences based on the standard codon table.
6. **Mutation Detection**: Compares two sequences to identify mutations such as substitutions, insertions, and deletions.

Each function is modularized to ensure reusability and ease of future extension.

## Installation and Environment

We created a `requirements.txt` file listing necessary libraries and versions to ensure compatibility:

Users are instructed to install dependencies within a virtual environment.

## Results

## Project Results

The DNA Sequence Manipulation and Analysis Tool was successfully implemented as a website app, supporting six core DNA operations.

### 1. Cropping

Users were able to crop sequences by specifying start and end positions. For example, given a 60-base DNA sequence, specifying start=10 and end=30 correctly returned a 21-base subsequence.

### 2. Pattern Searching

The regular expression pattern search successfully detected motifs such as ATG[ATCG]{3}TAA, returning all matches and their positions.

### 3. Sequence Reversal

The tool reversed DNA strands accurately. This is useful for finding reverse complements.

### 4. DNA to Protein Translation

The tool translated sequences into amino acids using the standard codon table. The output stopped at the first stop codon unless manually overridden.

### 5. Mutation Detection

Mutation detection between two sequences successfully identified substitutions and length mismatches.

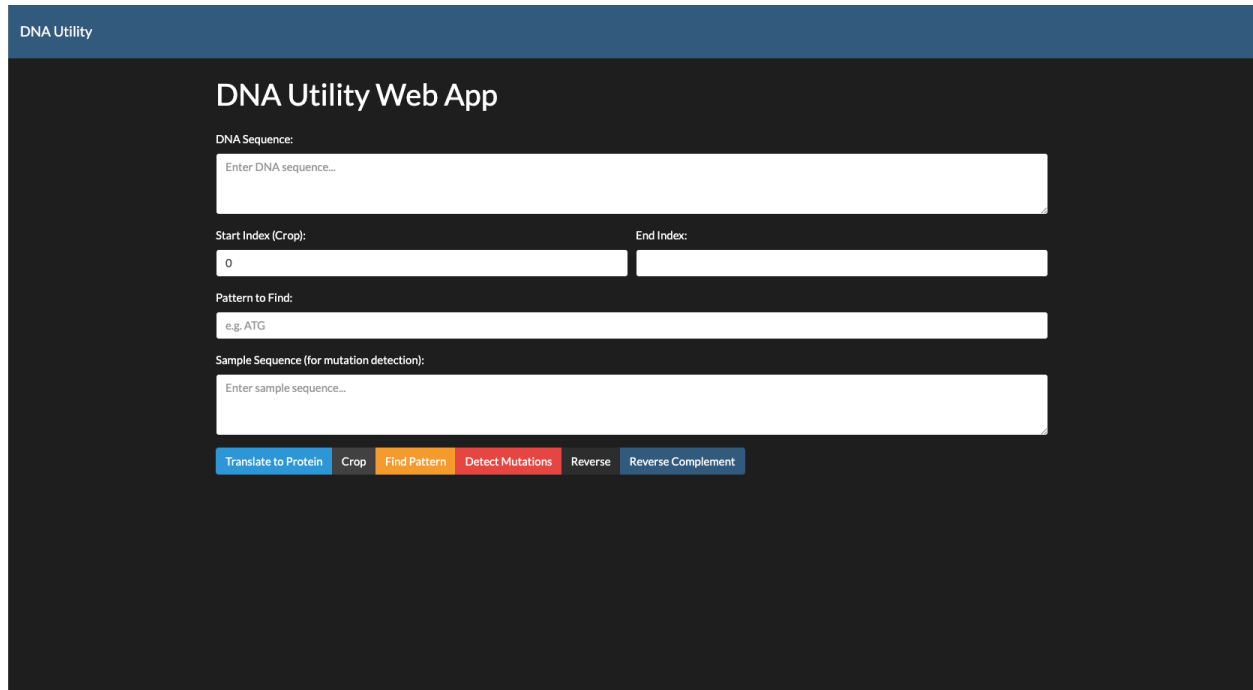
### 6. Reverse Complement

Sequence reversal function helps in implementing the reverse complement where A's and T's, G's and C's are swapped and the sequence is then reversed.

### 7. User Interface

The GUI presented clear menus and user guidance. Each function included helpful error messages for invalid inputs.

## Screenshot of Interface



The screenshot shows the 'DNA Utility Web App' interface. It features a dark blue header with the text 'DNA Utility'. The main content area is dark grey and contains the following elements:

- DNA Sequence:** A large white text input field with the placeholder text 'Enter DNA sequence...'.
- Start Index (Crop):** A white text input field with the value '0'.
- End Index:** A white text input field.
- Pattern to Find:** A white text input field with the placeholder text 'e.g. ATG'.
- Sample Sequence (for mutation detection):** A large white text input field with the placeholder text 'Enter sample sequence...'.
- Buttons:** A row of six buttons: 'Translate to Protein' (blue), 'Crop' (grey), 'Find Pattern' (orange), 'Detect Mutations' (red), 'Reverse' (grey), and 'Reverse Complement' (blue).

## Discussion

One of the core functionalities of our project is **DNA-to-protein translation** and **mutation detection**, both of which are implemented in modular, reusable Python functions. These functions form the analytical backbone of the tool, offering users biologically meaningful insights from raw sequence data.

### Translation Logic

The `translate_dna_to_protein()` function is responsible for converting a raw DNA sequence into a protein sequence using the **standard codon table**, which is embedded in the `CODON_TABLE` dictionary. This dictionary maps every possible three-nucleotide DNA codon (e.g., 'ATG', 'TAA') to its corresponding amino acid represented by a one-letter code (e.g., 'M' for Methionine, '\_' for stop codons).

The function processes the DNA sequence by:

- Removing whitespace and ensuring the string is uppercase for consistency.
- Iterating over the sequence in chunks of three nucleotides.

- Converting each triplet into an amino acid via a lookup in the CODON\_TABLE.
- Appending the result to a growing protein string.

This function allows students and researchers to quickly infer the translated protein sequence from gene data, supporting basic bioinformatics workflows such as gene annotation or protein prediction.

### Mutation Detection Logic

The `detect_mutations()` function compares a **reference DNA sequence** and a **sample sequence** to identify three types of mutations:

- **Substitutions:** when the nucleotides differ at the same position.
- **Insertions:** when the sample sequence has extra nucleotides.
- **Deletions:** when the reference sequence has nucleotides that are missing from the sample.

This function simplifies what could be a computationally complex task into a **character-wise comparison** loop:

- For each position up to the length of the shorter sequence, mismatched bases are reported as substitutions.
- If one sequence is longer, the additional bases are classified as insertions (from the sample) or deletions (from the reference).

This logic avoids the need for advanced alignment algorithms, making the tool efficient and suitable for educational or lightweight research contexts. Initial versions were more complicated and tried to account for alignment gaps and longer indels, but they proved error-prone. The final version favors clarity and accuracy for common use cases.

## Challenges Faced

1. **Preventing internal server error**

The app would crash when the crop button was pushed without entering the DNA sequence first resulting in an internal server error. This was fixed by adding error handling.

## Changes from the Original Plan

- We dropped the idea of using machine learning for mutation prediction to stay within a manageable project scope.
  - Extra focus was put on making modules reusable and well-tested for possible expansion later.
- 

# Appendix

## A. Accessing Program on Website

<https://dna-tool-254912217801.us-west2.run.app/>

## B. Running the Program Locally

1. Clone the GitHub repository  
(<https://github.com/WarnyKuma/dna-python-project?tab=readme-ov-file>).
2. Create a Python virtual environment:  

```
python3 -m venv venv
```

```
source venv/bin/activate
```
3. Install required packages:  

```
pip install -r requirements.txt
```
4. Run the program  

```
export FLASK_APP=app
```

```
flask run
```