A Randomized, Probabilistic Algorithm for Decrypting Monoalphabetic Substitution Ciphertext

Anjali Sreenivas
Stanford University

December 2, 2024

1 Motivation and Overview

When brainstorming for my project, I was excited by the idea of building a new probabilistic algorithm of my own to tackle some challenge. Possibilities using *inference* in particular captivated me, especially having seen applications of inference extending from poker game predictions (as Chris showed us in class) to real-time updates of projected NFL game champions.

I eventually arrived at the idea of building a probabilistic, inference-driven algorithm to **decrypt ciphertext encrypted via a monoalphabetic substitution cipher**—that is, a cipher where every alphabetic letter (A-Z) in the original plaintext is mapped to some other alphabetic letter (A-Z) in the ciphertext. The **encryption key**, which we define as the mapping from the encrypted text characters to the corresponding plaintext characters, is unknown. For example, an encryption key of $A \mapsto Q,\ B \mapsto A,\ C \mapsto Z,\ D \mapsto W,\ E \mapsto S, \ldots$ denotes that the letter 'A' in the ciphertext maps to 'Q' in the original (plain/decrypted) text. The **goal** I set for my project was to **leverage probability theory to build an algorithm** that **takes in encrypted text** (via a monoalphabetic substitution cipher) as input, and efficiently identifies the encryption key to then **return the decrypted ciphertext**.

My findings illustrate the computational efficiency, performance capabilities, and overall potential of applying probabilistic techniques to search for optima within vast sample spaces. In terms of educational value, my work can be used to help learners understand how probabilistic models may leverage weaknesses in ciphers, therein motivating the need for more advanced cryptography / data security techniques.

2 The Algorithm

I develop a **probabilistic Monte Carlo Markov Chain (MCMC) method** [5, 9] that exploits **inference** to decrypt an encrypted ciphertext input. Put simply, this means that my algorithm has the following key characteristics:

- Monte Carlo: My algorithm uses **randomization** and probabilistic inclinations to explore the encryption key search space, meaning that it *does not always return the correct output.* **So why use randomization?** Leveraging this approach yields **high computational efficiency** across a wide range of passage lengths, as shown in Figure 9. This is **tremendous improvement compared to the efficiency of an exhaustive brute force search** through the 26! **possible encryption keys** to identify the most likely one. As an additional note for practical use, my algorithm—like many randomized algorithms—can be **run multiple times on the same input to decrease error rate,** and the output that yields the highest decryption accuracy can be selected.
- Markov Chain: At each iteration, the decision I make for what the next step will be is solely based on the current state. In the context of my algorithm, at each step, the **next encryption key that I choose to explore is based entirely on my current belief** of the true encryption key.

Below is a step-by-step description of how my algorithm finds the likely encryption key, given an encrypted passage:

- **Step 1:** First, I **generate an initial prior key belief** by sorting the letters of the ciphertext in descending order of their frequencies. I map the ith character in the sorted ciphertext character list to the ith most frequent letter of the English language. For example, suppose that the most frequently occurring letter in the ciphertext is 'Q'. Then, I map 'Q' to 'E' in my initial prior, since 'E' is the most frequent letter in the English language.

- **Step 2:** Using this initial prior, I **compute the likelihood of the current key belief being our true encryption key**, given the encrypted passage. Using **Bayes' Theorem**, we see the following:

$$P(\text{current key} \mid \text{encrypted passage}) = \frac{P(\text{encrypted passage} \mid \text{current key}) \cdot P(\text{current key})}{P(\text{encrypted passage})} \tag{1}$$

$$P(\text{current key} \mid \text{encrypted passage}) \propto P(\text{encrypted passage} \mid \text{current key}) \cdot P(\text{current key}) \tag{2}$$

$$P(\text{current key} \mid \text{encrypted passage}) \propto P(\text{encrypted passage} \mid \text{current key}) \tag{3}$$

In Equation (1), by the **Law of Total Probability**, our denominator can be expanded as follows:

$$P(\text{encrypted passage}) = \sum_{\text{key}} P(\text{encrypted passage} \mid \text{key}) \cdot P(\text{key}). \tag{4}$$

This is a constant irrespective of our current key input, which is why we are able to go from Equation (1) to Equation (2). Furthermore, since the probability of any key guess being the true unknown encryption key is $\frac{1}{26!}$, another constant, we can go from Equation (2) to (3).

Now, to **compute** $P($**encrypted passage** $\mid$ **current key**$)$, I do the following:

  - First, I decrypt the encrypted passage according to my current key belief, and then tokenize this decrypted passage guess. My goal is now to compute P(decrypted passage guess).
  - Since I **assume that tokens are independent of each other**, I compute $P($encrypted passage $\mid$ current key$)$ as $P($decrypted passage guess$) = \prod_{\text{token}} P($decrypted token$)$. The probability of seeing each decrypted token is computed using the matching approach below:
    * Recognized Tokens: I reference a dataset containing the **top 333,333 English words** and their frequencies within Google's one trillion word corpus. After converting this frequency data into probabilities, I used the probabilities to answer the question of how likely it is to observe a decrypted token for any tokens that are in this word bank.
    * Unrecognized Tokens: For the tokens that are not found within the word bank, I compute each token's probability as the product of the likelihoods of seeing each constituent character. This follows from my **assumption that each character in the input passage is also independent** of all others. To do this, I refer to known probabilities of the letters A-Z in the English language. When tuning my algorithm, I ended up **setting a lower bound for the probability I assign to each token** as 10^{-7}–without this correction, my algorithm appeared to be **over-penalizing** encryption keys that could, in fact, lead us towards the optimal key.

  I store my final probability as a **log likelihood** to avoid underflow. That is, I store $\log(P($decrypted passage guess$))$ as my estimate for the original probability I sought to compute: $P($current key $\mid$ encrypted passage$)$.

- **Step 3:** I use an MCMC method to **update my current belief of the encryption key**. To determine the next encryption key I will explore, I **randomly select any two keys** from my current encryption key guess and **swap the characters they map to**. I then **repeat Step 2 with this new key guess** to get the log likelihood of the new encryption key guess being the true encryption key. Let us denote the log likelihood of our current key belief as p, and the log likelihood of our new key guess as q. Now, I consider two cases:

  - Case 1: $q > p$. Since we found an encryption key with a higher likelihood of being the true mapping, I update my current key belief to be this new encryption key and proceed.
  - Case 2: $q \leq p$. I compute the **log of the likelihood ratio**, r, of the new key guess to the current key belief. Mathematically, this value is $r = q - p$ where r is always ≤ 0. I then define an **acceptance probability** p_a as $p_a = e^r$. I perform a binary **Bernoulli "coin flip" with probability** p_a**;** if the outcome is 1, I accept the new key guess and update my current key belief. If not, I keep my current key belief as is and proceed.

My strategy here for exploration and acceptance of encryption key guesses closely aligns with the **Metropolis-Hastings sampling method**. The fundamental idea is that we **randomly, yet gradually walk through our sample key space**, taking a "step" in some direction from our current key belief. When we are *certain* we have found a better key guess, we update our belief. When we arrive at a key guess with a lower likelihood than the our current key belief, the chance that we accept the new key guess decreases exponentially as the

difference in the observed likelihoods increases. The purpose of this logic is to **avoid getting stuck at local likelihood optima**—our probabilistic acceptance strategy allows the algorithm to explore new key hypotheses, particularly when the likelihood difference is small, suggesting that the optimal encryption key might still be undiscovered. Furthermore, **as we approach the true encryption key, the likelihood of our current key belief should be much higher than the likelihoods of other keys farther from the optimal solution**, resulting in a lower acceptance probability for these suboptimal keys. As I iteratively built up my algorithm, I found that token matching within the word bank was crucial to capitalize on this characteristic and therein improve my algorithm's performance.

- **Step 4:** I repeat Steps 2 and 3 many times within a loop. In my implementation, I run my loop 8,000 times. Figure 1 shows my **algorithm converging in its encryption key belief over the iterations** for a variety of input passages, which follows from the likelihood of the key belief becoming increasingly greater than other keys being sampled. As the iterations progress, the log likelihood of the algorithm's encryption key belief gradually increases until we reach convergence. As an additional note, when the algorithm starts converging at a lower likelihood compared to the true encryption key (the global optima) that it should discover, the acceptance probability concept sometimes allows us to recover (e.g. the long "Pride and Prejudice" passage illustrated in Figure 10). Other times, the algorithm struggles to recover (e.g. the medium-length "Federalist Papers" passage in Figure 1), in which case we end up predicting an incorrect encryption mapping.
- **Step 5:** Our key belief at the end of our 8,000 iterations (Step 4) is our **predicted encryption key mapping**. We decrypt the encrypted passage input according to this mapping and return the decrypted text.

3 Performance Analysis and Findings

I selected arbitrary short (100-300 words), medium (600-800 words), and long (1,000+ words) passages from each of the eight following texts: recent journal articles from *New York Times* Politics [6, 10], the 2024 UN Adaptation Gap Report about climate adaptation [3], the Federalist Papers [4], "Pride and Prejudice" by Jane Austen [1], "The Lion, The Witch, and The Wardrobe" by C.S. Lewis [2], the CS109 Course Reader by Chris Piech [8], "Harry Potter and the Philosopher's Stone" by J.K. Rowling [11], and "To Kill a Mockingbird" by Harper Lee [7].

For each of the passages (for each length grouping), I repeated the following $N = 10$ times:

1. I generated a random encryption key and encrypted the passage.
2. I inputted the encrypted passage into my algorithm and obtained the decrypted passage prediction.
3. I computed the decryption accuracy as follows: $\frac{\text{Number of alphabetic characters decrypted correctly}}{\text{Total number of alphabetic characters in original passage}}$

I then used my 10 decryption accuracy score samples to compute the following for each passage:

- I estimated the passage's mean accuracy score as the sample mean accuracy score, $\bar{X}$, since the Mean Passage Accuracy is $\mathbb{E}[\bar{X}]$ according to the **Central Limit Theorem**.
- I computed the **sample variance** as follows: $S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X})^2$ for all X_i in my samples list X.
- In accordance with the Central Limit Theorem, I estimated the **passage's mean accuracy variance** as $\frac{S^2}{N}$.
- I square rooted my passage's mean accuracy variance estimate from above to compute **mean accuracy standard deviation**.

Using the passage mean accuracy scores and mean accuracy standard deviations, I constructed Figures 2 through 4, which are bar plots **illustrating the effectiveness of my algorithm on the wide assortment of test passages stratified by their length grouping**. I also made a scatterplot (Figure 5), to model the **results as a continuous function of passage length**. Figures 6 through 8 are bar plots showing median sample accuracy scores for each passage organized by their length grouping.

In analyzing the figures, we can see that the **algorithm tends to perform better when passage lengths are longer**. For long passages, the mean Mean Accuracy Score (MMAS) is 0.808. For medium-length passages, the MMAS is 0.726, and for short passages, it is 0.612. Moreover, as shown in Figure 5, there is a **weak positive correlation** (with correlation coefficient 0.40) between passage length and Mean Accuracy Score. It is also noteworthy that the median sample accuracy score for 6 out of the 8 long passages was 1.0 (Figure 8), and this was the case only for 4 of the 8 short passages (Figure 6). This trend highlights the algorithm's strength in leveraging the larger

linguistic context contained in longer passages, which allows it to better capture the distribution of letters and words in the English language.

But perhaps the more interesting pattern is the **types of passages that the algorithm decrypts well versus poorly**. Across all passage lengths, we see the algorithm struggled to accurately decrypt excerpts from *The New York Times* politics and the 2024 UN Adaptation Gap Report. In the medium and longer excerpts, the Federalist Papers also had a significantly lower decryption accuracy. All three of these sources have a high saturation of proper nouns, which are not found within the reference word corpus. In contrast, the algorithm performed exceptionally well on passages from "To Kill a Mockingbird", "Harry Potter", and the CS109 Course Reader, where the prevalence of common words found in our reference word bank increased the disparity in likelihood scores between correct (or near-correct) keys and incorrect ones.

4 Closing Thoughts

4.1 Conclusions / Future Work

Our findings demonstrate the significant **potential of the algorithm to accurately decrypt monoalphabetic substitution ciphertext, especially for texts of sufficient length and a higher concentration of commonly used English words.** However, the algorithm does appear to be significantly **biased towards words that are frequent within the English language.** As a next step, we can continue to refine / tune the algorithm to specifically target this issue and try to achieve consistency even in texts saturated with proper nouns. We can also explore what happens when the number of keys we change / swap in any given iteration of our algorithm when formulating our new key guess is greater than two.

4.2 Code Availability

All code and data generated for the project is available here, as well as the reference test passages used: Probabilistic MCMC Decryption

4.3 Acknowledgements

A huge thank you to my professor, Chris Piech, and the entire CS109 teaching team for their warmth, support, and encouragement the whole way. Also a big thank you to my family and friends with whom I could soundboard my ideas and share my excitement!

4.4 LLM Usage

After generating all my decryption trial data, I asked ChatGPT to give me code samples for constructing various plots with matplotlib. I used this as a baseline for writing code to build my figures, and then further stylized / customized my visuals. When initially learning about the Metropolis-Hastings algorithm and thinking about whether / how to integrate something similar into my project, I found that brainstorming with ChatGPT was thought-provoking.

5 Figures

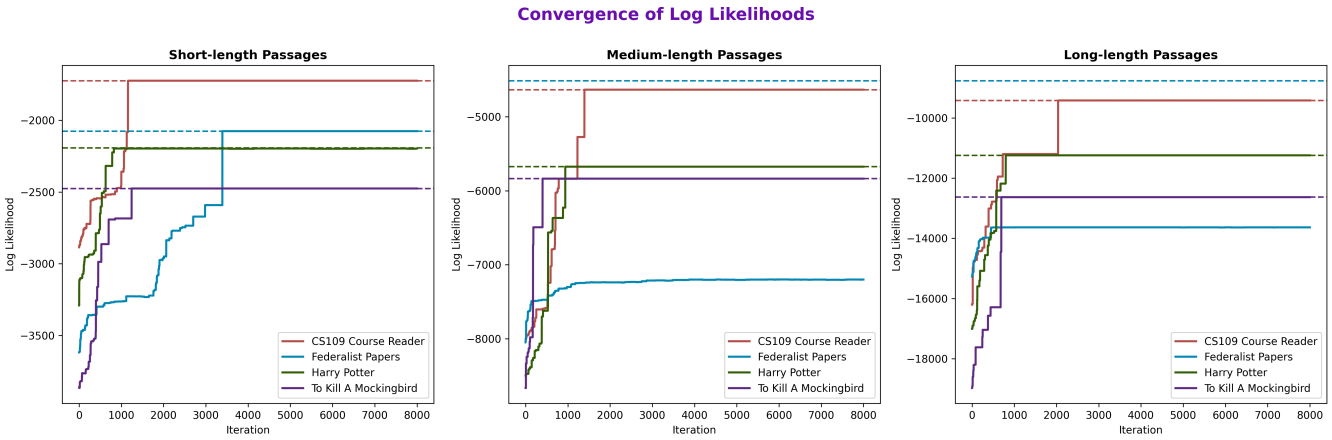

Figure 1: The plots illustrate the algorithm's convergence on its encryption key belief across 8,000 iterations given a variety of inputs of different lengths (short passages: 100-300 words; medium passages: 600-800 words; long passages: 1,000+ words). For each example, a random encryption key was generated and used to encrypt the passage text. The y-axis is the log likelihood of the algorithm's current key belief. The dashed lines show where the algorithm should converge if the correct encryption key is found—that is, the log likelihood of the passage when decrypted with the correct encryption key.

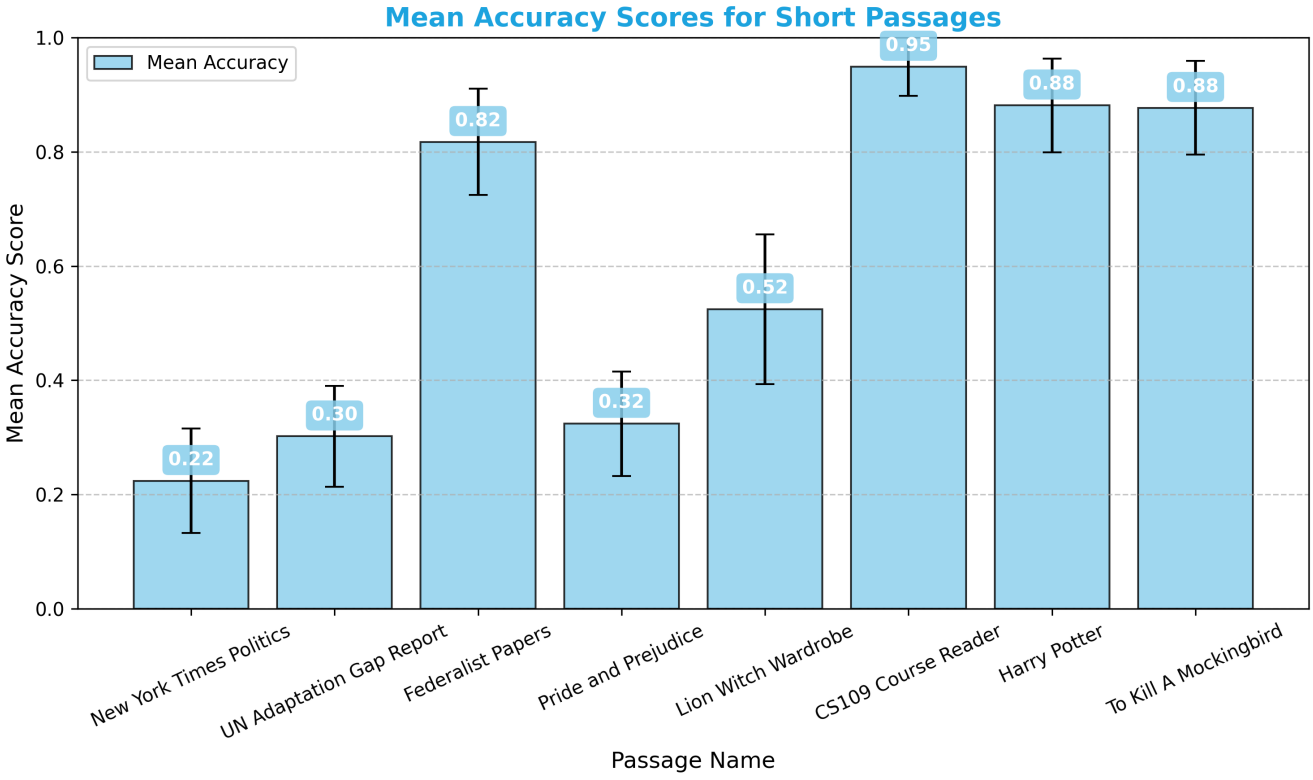

Figure 2: Mean Decryption Accuracy Scores for Short Passages (100-300 words)

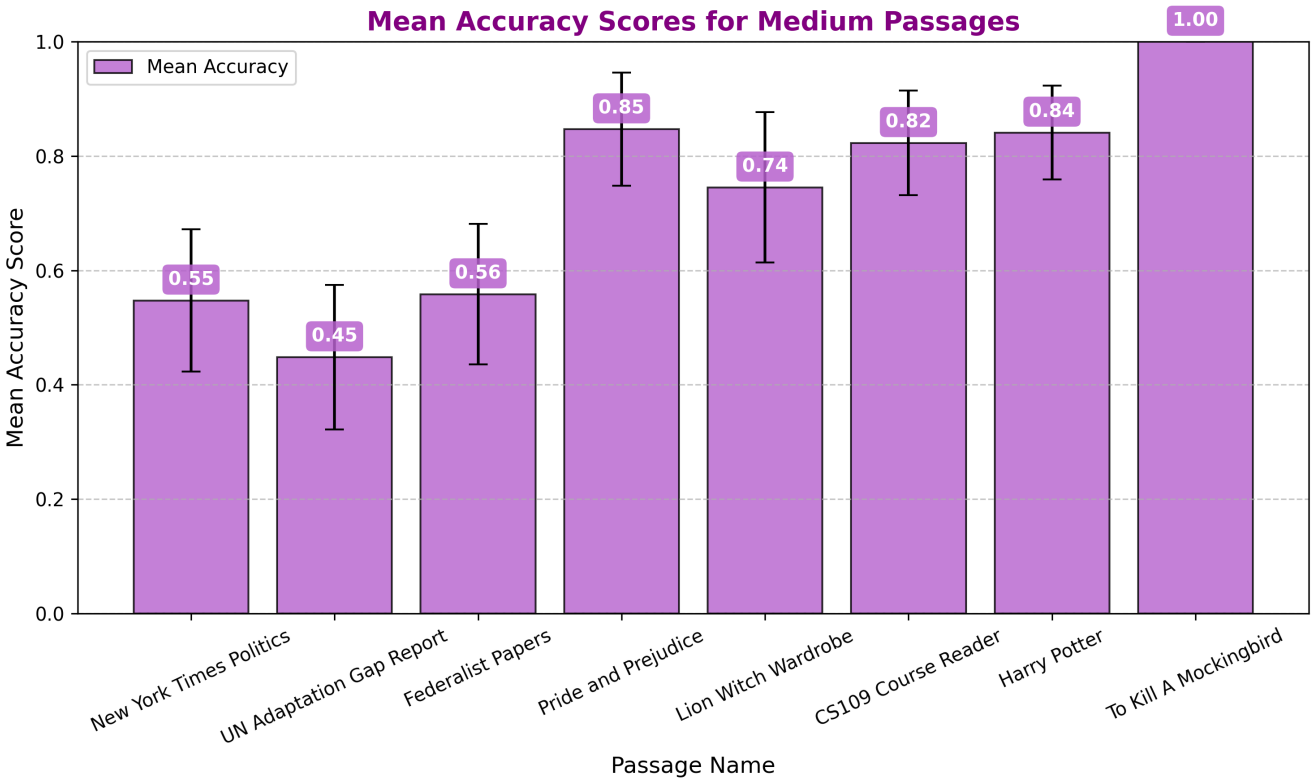

Figure 3: Mean Decryption Accuracy Scores for Medium Passages (600-800 words)

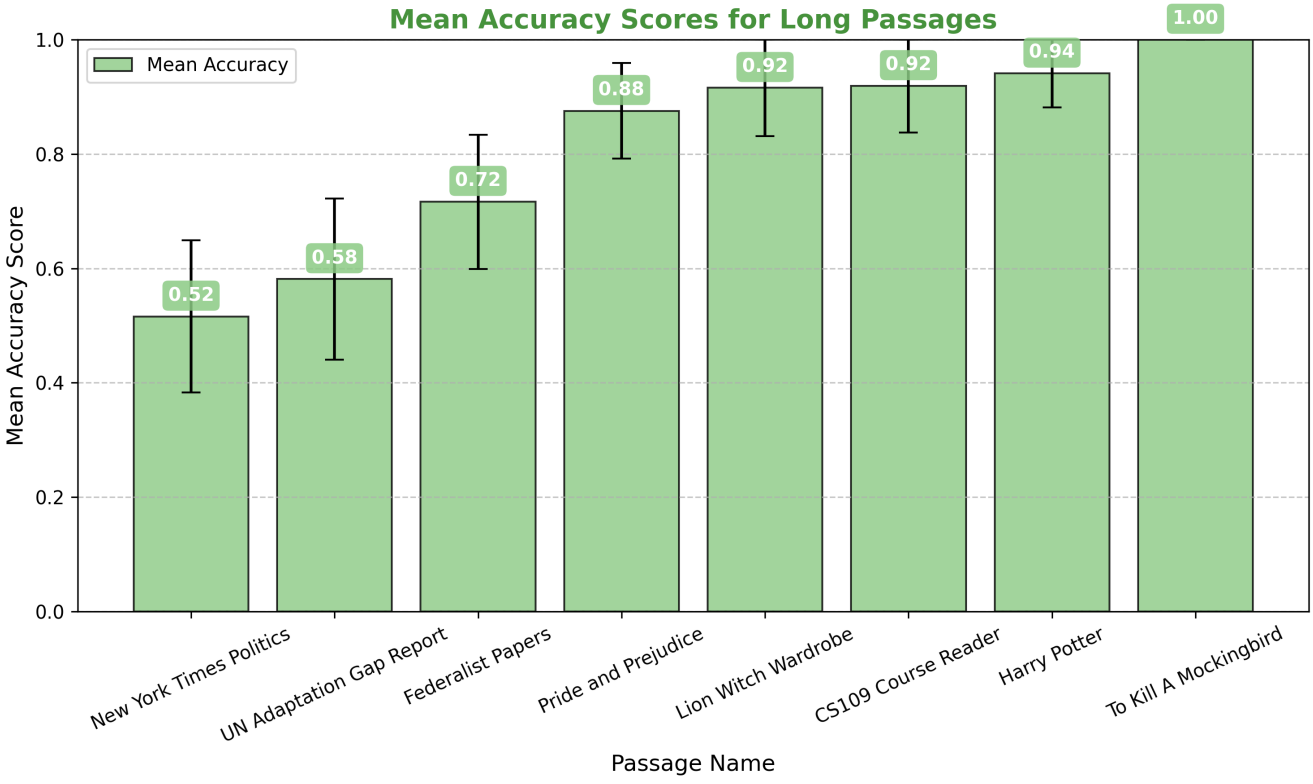

Figure 4: Mean Decryption Accuracy Scores for Long Passages (1000+ words)

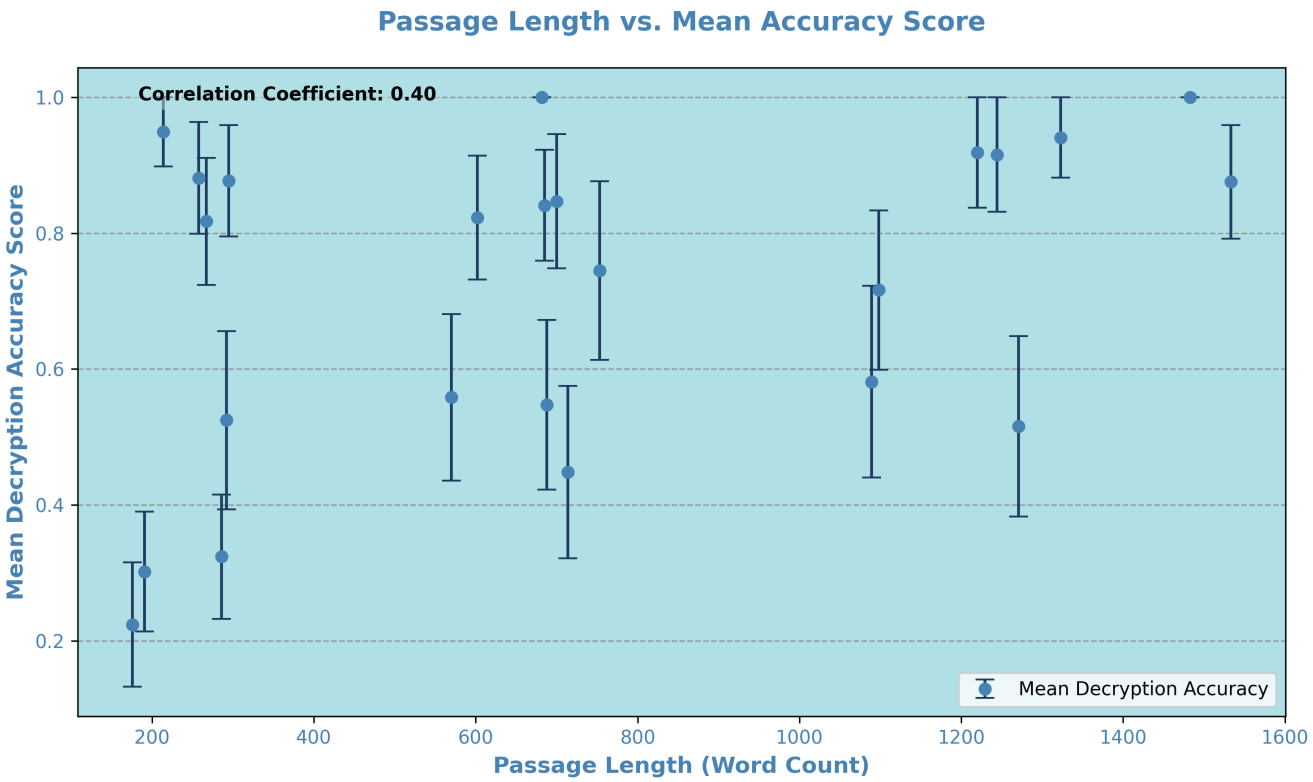

Figure 5: Scatterplot illustrating a weak positive correlation between input passage length and algorithm decryption accuracy.

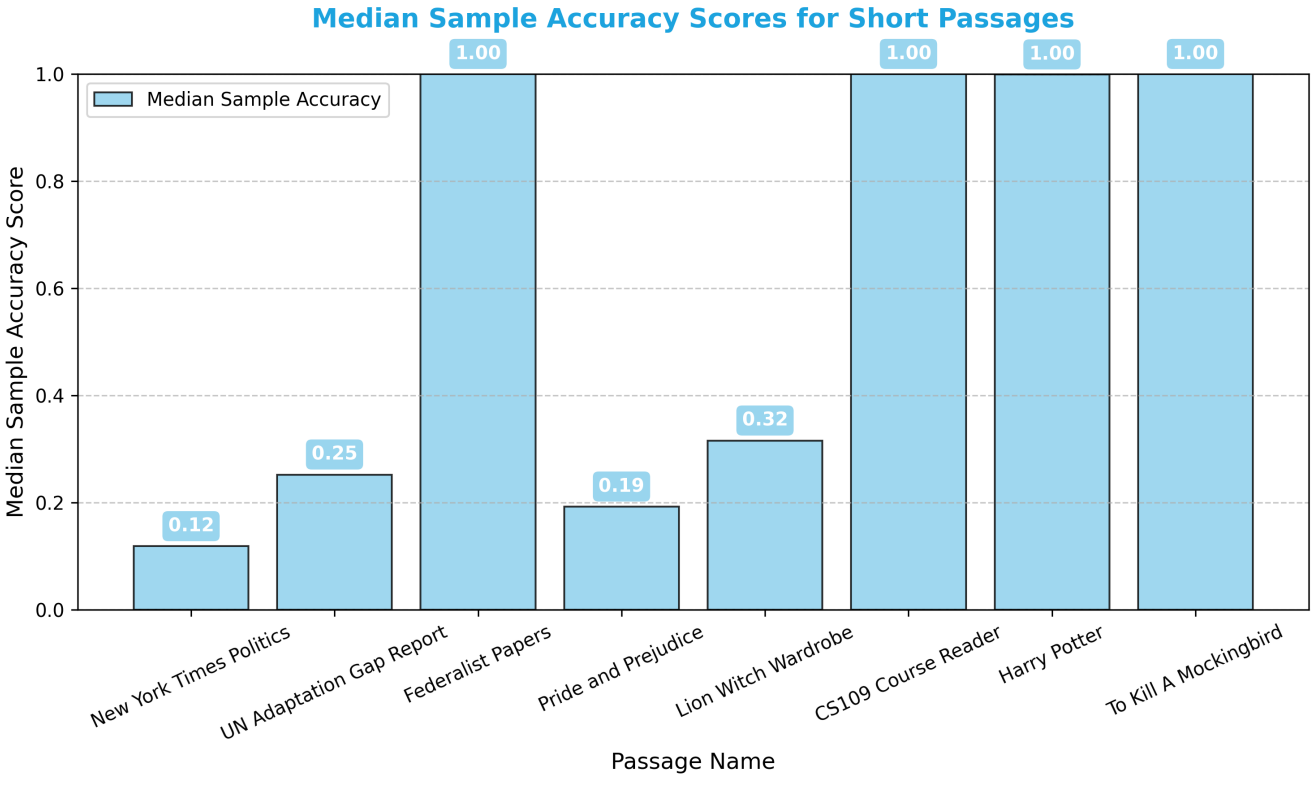

Figure 6: Median Decryption Accuracy Scores for Short Passages (100-300 words)

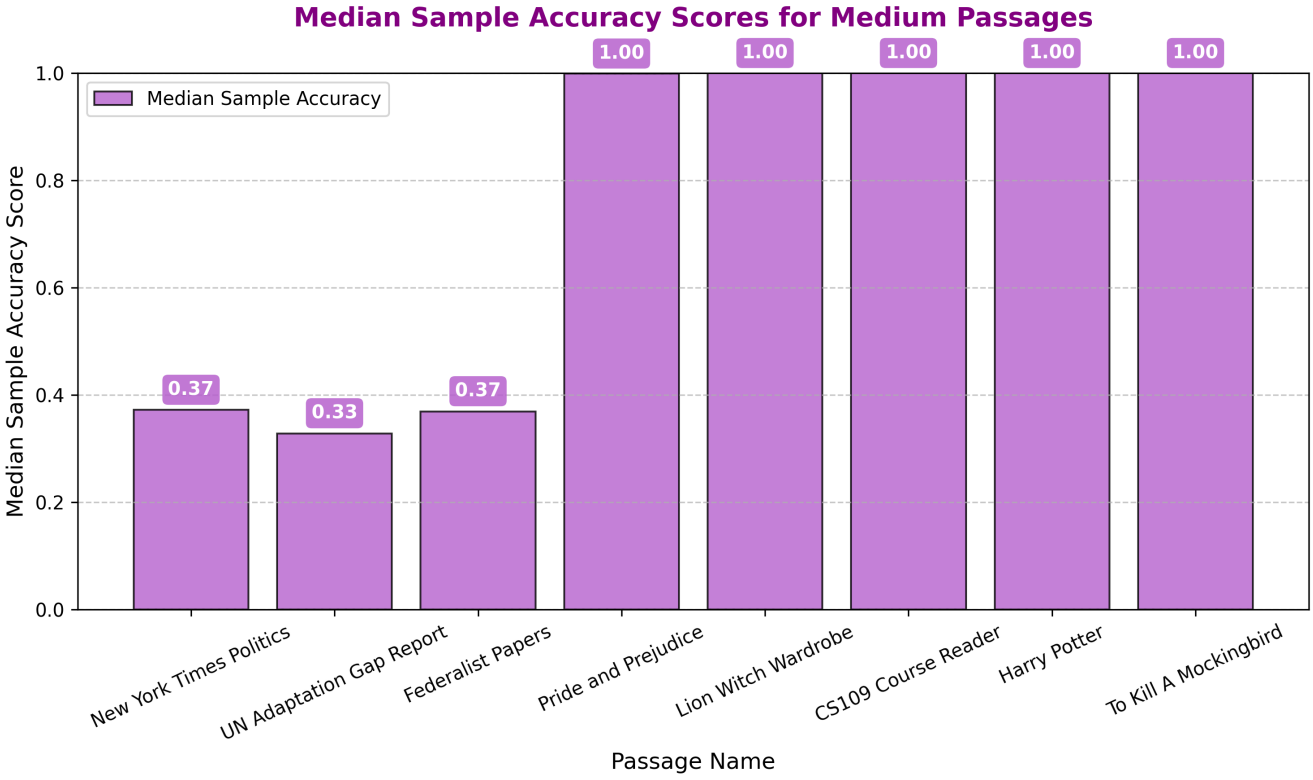

Figure 7: Median Decryption Accuracy Scores for Medium Passages (600-800 words)

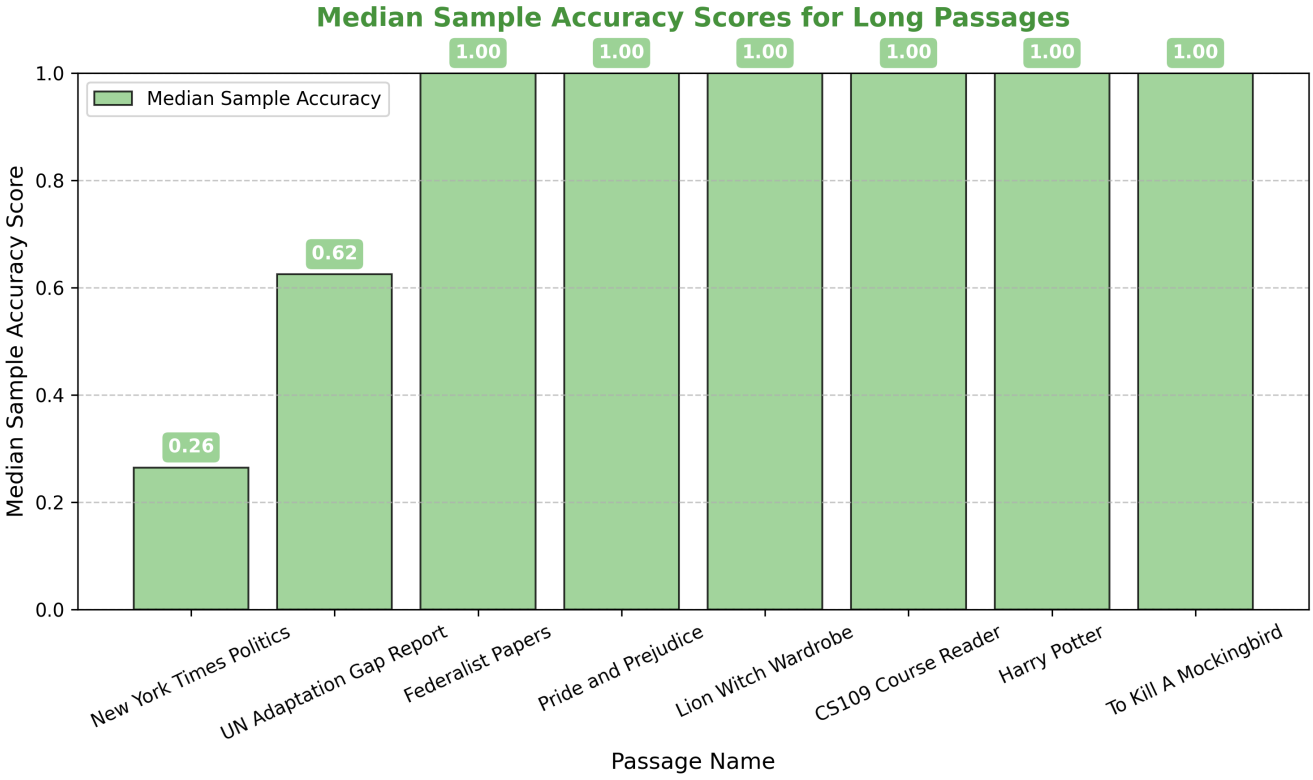

Figure 8: Median Decryption Accuracy Scores for Long Passages (1000+ words)

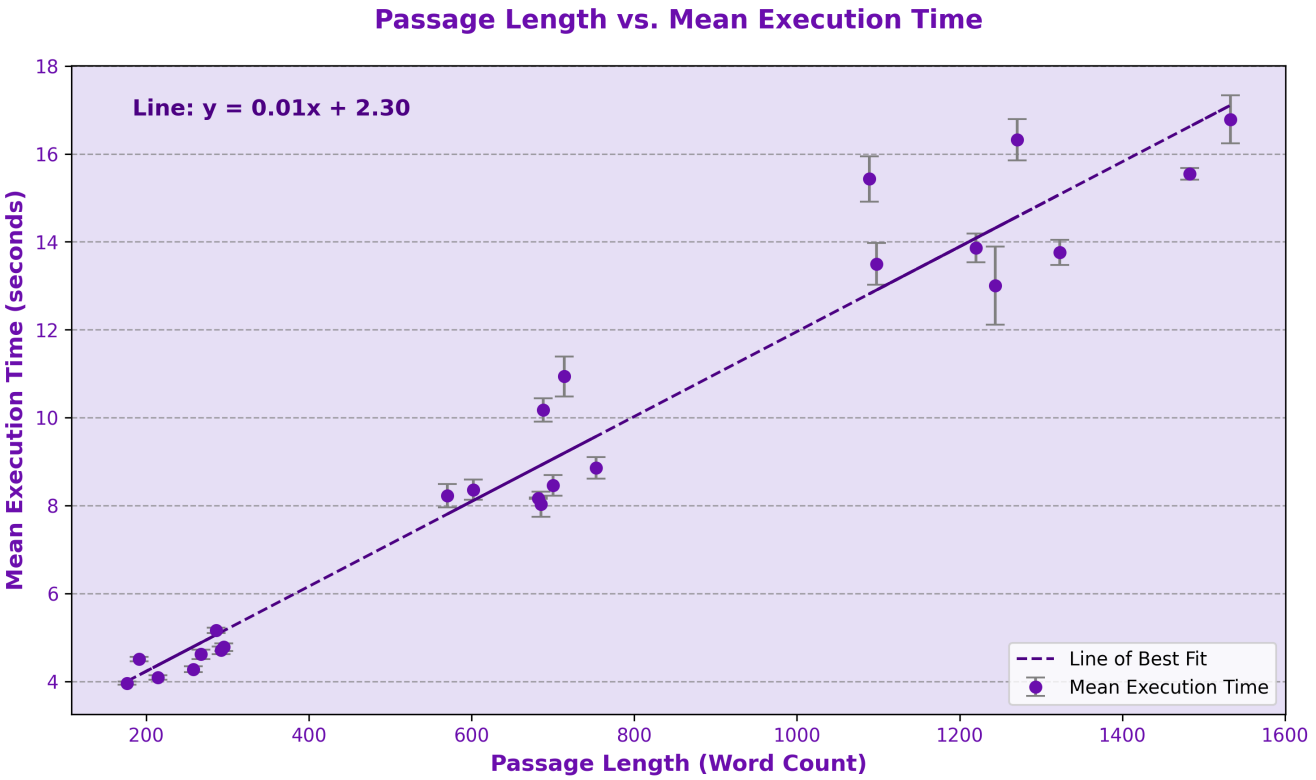

Figure 9: Scatterplot illustrating a linear trend in execution time between input passage length and algorithm execution time.

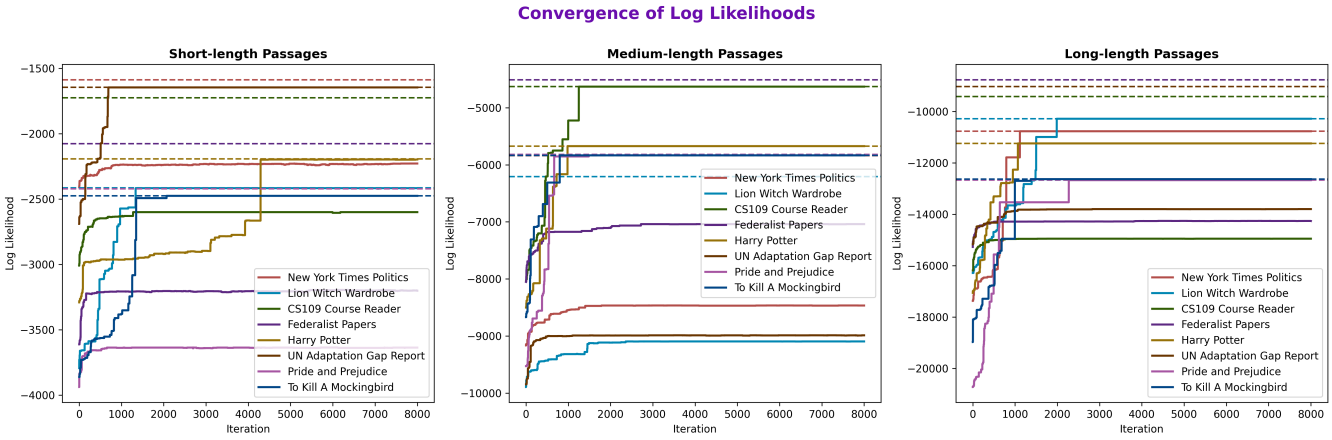

Figure 10: Algorithm Convergence on a Random Run with Full Test Passage Assortment

References

[1] Jane Austen. *Pride and Prejudice*.

[2] C.S.Lewis. *Chronicles of Narnia Books*. 1950.

[3] U. N. Environment. Adaptation Gap Report 2024 | UNEP - UN Environment Programme, October 2024. Section: publications.

[4] Alexander Hamilton, James Madison, and John Jay. The Federalist Papers (1787-88).

[5] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[6] Mike Isaac, Jonathan Swan, Maggie Haberman, and Theodore Schleifer. Mark Zuckerberg Meets With Trump at Mar-a-Lago. *The New York Times*, November 2024.

[7] Harper Lee. *To Kill a Mockingbird*.

[8] Chris Piech. Probability for Computer Scientists.

[9] Christian P. Robert. The Metropolis-Hastings algorithm, January 2016. arXiv:1504.01896.

[10] Katie Rogers. Donald Trump Jr. Emerges as a Loyal Enforcer. *The New York Times*, November 2024.

[11] J. K. Rowling. *The Sorcerer's Stone*.