Evolutionary Algorithms and Neural Network-Based Fitness Functions for Extractive Text Summarization: A Comparative Study with ChatGPT

Anonymous ACL submission

Abstract

Extractive text summarization deals with extracting a limited number of important sentences from a large document to create a summary. One novel approach already proposed in the literature is to model extractive summarization as an optimization problem, where a Genetic algorithm (GA) has been used for optimizing the selection of sentences from a text to generate the best extractive summary, which has been found outperforming state-of-the-art techniques. In this work, we build a similar model where apart from GA we used several different evolutionary algorithms (EA) in order to identify the combination that produces the best result. For this work, we have used different evolutionary algorithms, namely Discrete Differential Evolution (DDE), Cuckoo Search, Particle Swarm Optimization, and Firefly Search along with Genetic Algorithm, and have made comparison of their results with state-of-the art LLM viz. ChatGPT. The results are evaluated on the BBC news dataset using the precision-recall technique metric.

Keywords: Document Summarization, Extractive Text Summarization, Evolutionary Algorithms, Neural Networks, ChatGPT

1 Introduction

The present work models extractive text summarization as an optimization problem since the search space for generating an extractive summary grows exponentially with the length of the input document. To this end, we have experimented with various evolutionary algorithms (EAs) for extractive text summarization. EAs constitute a class of nature-inspired optimization techniques, particularly effective in searching for optimal solutions in extensive solution spaces.

Typically, an evolutionary algorithm works on optimizing a fitness function, say f. Performance of the algorithm depends on the definition of the fitness function. However, obtaining the right fitness function has always been a challenging task for different EAs. Hence for this work instead of using any human-defined function we have used a neural network to define f. Multiple experiments have been conducted to choose an appropriate f by varying several metrics, including the number of training documents from datasets and different distribution techniques for creating candidate summaries for these training documents. The experiments are run over the widely recognized BBC dataset, which comprises 2225 single-document articles from various categories, each accompanied by its ideal extractive summary. In particular, these experiments aim to refine and determine the best fitness function by closely examining the specified factors. We employed the optimized fitness function in five distinct EAs for Text Summarization: Genetic Algorithm (GA), Discrete Differential Evolution (DDE), Firefly Algorithm, Particle Swarm Optimization (PSO), and Cuckoo Search. Subsequently, we conducted a comprehensive comparative study, evaluating their performance against each other and in comparison to modern-day Large Language Model-based technologies, namely ChatGPT.

2 Relevant Past Works

A wide variety of techniques have been experimented with and used for extractive text summarization, but usage of EAs is still less explored. In the traditional schemes, frequency-driven approaches (Nenkova et al., 2011), sentence ranking (Madhuri and Kumar, 2019), k-nearest neighbor (Jo, 2017), fuzzy C-means and aggregate scoring methods (Rahman et al., 2019), centroid-based summarization (Lwin and Nwet, 2019), Latent Semantic Analysis (LSA) (K and N, 2015), Random Indexing (Chatterjee and Sahoo, 2014) and Bayesian Topic Models (Chandra et al., 2011)

have been used. Evolutionary algorithms-based approaches for text summarization include (Karwa and Chatterjee, 2015) wherein they have used Discrete Differential Evolution (DDE) to summarize a single document. In 2018, Chatterjee, Jain, and Bajwa (Chatterjee et al., 2019) worked on single document text summarization with the help of the GA and neural networks (NN). They trained the fitness function with the help of the neural network, and later genetic algorithm was used to search for the best optimal solution.

In the present work we enhance text summarization by employing five different EAs integrating them with a NN-driven fitness function. Our experiments highlight the effectiveness of this combined approach in generating improved extractive summaries.

3 Experimental Modelling and Fitness Function

In the following subsections, we delve into our approach for document and summary representation, paving the way to the definition and computation of the novel neural network-driven fitness function.

3.1 Document Representation

In the proposed model, a document is represented using a weighted Directed Acyclic Graph (DAG). Under this framework, each sentence in the document serves as a set of vertices, denoted as V, within the graph. An edge is drawn from sentence s_1 to sentence s_2 only if s_1 precedes s_2 in the document (Shimpikar and Govilkar, 2017). Additionally, each sentence is encoded as a vector for the purpose of measuring similarity between sentences, employing the TF-IDF vectorization scheme for this representation. The similarity between two sentences is calculated using the cosine similarity measure between the TF-IDF vector representation of the sentences s_i and s_j respectively.

3.2 Summary Representation

For a document comprising N sentences, a summary of S sentences is captured as an N-dimensional binary vector. This vector is uniquely characterized by S occurrences of '1' and N-S occurrences of '0', culminating in what we term a 'summary vector'. Each element in this vector corresponds to a sentence in the document, marked as '1' if included in the summary and '0' otherwise.

The mathematical framework guiding the proposed fitness function calculation relies on eight features (Chatterjee et al., 2019) representing a summary. These features will be used as inputs for our fitness function, helping evolutionary algorithms find the best extractive summaries.

Radial and Angular Theme Similarity: The Radial Theme Similarity and Angular Theme Similarity are defined as follows:

$$RTS_{Summary} = \|\mathbf{T}_N - \mathbf{T}_{\text{Summ}}\|_2$$
$$ATS_{Summary} = \mathbf{T}_N \cdot \mathbf{T}_{\text{Summ}}$$

Here T_N denotes the central theme of the Document, and T_{Summ} denotes the central theme of Summary defined as:

$$\mathbf{T}_{\text{N}}=\frac{\sum_{s_i \in \text{Document}} w_{s_i}}{|\text{Document}|},\ \mathbf{T}_{\text{Summ}}=\frac{\sum_{s_i \in \text{Summary}} w_{s_i}}{|\text{Summary}|}$$

Where, $\|.\|_2$ denotes the Euclidean Distance, and $|.|$ denotes the cardinality of the set.

Sentiment Factor: Online AFINN-111 (Nielsen, 2011) dictionary has been used for this purpose. The sentiment factor is calculated as:

$$\text{sentiment}(\text{s}_i) = \sum_{\text{word}_j \in \text{s}_i} \text{sentiment\_score}(\text{word}_j)$$

$$\text{Senti}_{Summary} = \sum_{s_i \in \text{Summary}} \text{sentiment}(s_i)$$

$$\text{Sentiment Factor: SF}_{Summary} = \frac{\text{Senti}_{Summary}}{\max_{\forall \{summary:|\text{summary}|=S\}}(\text{Senti}_{summary})}$$

Cohesion Factor: This feature considers all the pairwise similarities of sentences in the summary and based on that gives Cohesion Factor score of a summary as follows:

$$C_{Summary} = \frac{\sum s_i, s_j \in \text{ summary subgraph } \text{sim}(s_i, s_j)}{N_s}$$

where $N_s = \frac{S(S-1)}{2}$ is the number of edges in a summary subgraph with S nodes. This formulae is used to normalize the value of CF of a summary:

$$CF_{Summary} = \frac{\log(9 \cdot C_{Summary} + 1)}{\log(9 \cdot M + 1)}$$

where M is $max_{\forall i,j \leq N}\ sim(s_i, s_j)$. Clearly, $C_{Summary} \leq M$ and $CF_{Summary} \leq 1$

Readability Factor: A summary should be readable, i.e. the sentence should be similar to the preceding sentence in the summary(Qazvinian et al., 2008). Readability factor is given by:

$$RF_{\text{Summary}} = \frac{\sum_{1 \leq i < S} \text{sim}(s_i, s_{i+1})}{\max_{\forall i} \text{sim}(s_i, s_{i+1})}$$

Aggregate Similarity: The aggregate score of a sentence ($\text{sent}_{\text{aggregate}}(s_i)$) is the sum of edges incident onto it in the graph. This factor is determined by summing the maximum S aggregate scores among N sentences, where S and N denote the number of sentences in summary and document. Aggregate similarity ($\text{AS}_{\text{Summary}}$) is given as:

$$\text{Aggregate}_{\text{Sent}}(s_i) = \sum_{j=1}^{N} \text{sim}(s_i, s_j)$$

$$\text{AS}_{\text{Summary}} = \frac{\sum_{s_i \in \text{Summary}} \text{Aggregate}_{\text{Sent}}(s_i)}{\sum_{i=1}^{S} \max_j(\text{Aggregate}_{\text{Sent}}(s_j))}$$

Sentence Position: Sentences are assessed based on their position within the text, and the cumulative scores assigned to them result in a sentence position score for the summary(Shimpikar and Govilkar, 2017).

$$SP_i = \frac{2(N-i)}{N(N+1)}$$

where SP_i is the Sentence position score of i^{th} sentence in the document. The overall Sentence Position score of a summary is calculated as:

$$SP_{\text{Summary}} = \sum_{s_i \in \text{Summary}} SP_{s_i}$$

k by N Ratio: It is the ratio of the number of sentences in the summary to the total number of sentences in the document.

3.3 Fitness Function Configuration

Since the goal of the Evolutionary Algorithms is to construct a summary that maximizes the fitness function f learned by the neural network. This function takes the summary represented as set of the eight features as input (X), and its output (Y) is either precision or F1-Score. Ideal extractive summaries, represented as binary vectors, offer a benchmark for precision values, indicating proximity to the ideal. In order to model this a neural network is trained to generate an effective fitness function. Each entry in the dataset comprises a feature vector of size eight representing the summary as the input for the neural network along with associated precision/F1-Score value (Y).

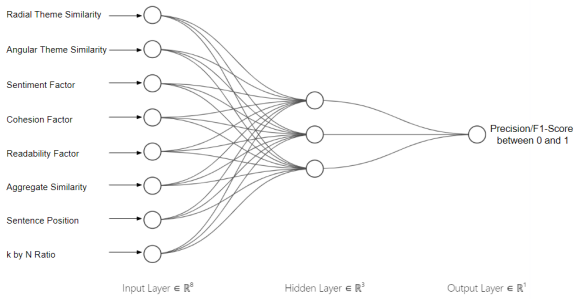

Figure 1: Architecture of Neural Network

The neural network employed for computing the proposed fitness function f comprises three layers, each designed to capture distinct aspects of summarization effectiveness:
(i) Input Layer: The initial layer is composed of eight neurons, each receiving input from the eight features described above representing summary.
(ii) Hidden Layer: Three neurons for learning intricate patterns within the feature space.
(iii) Output Layer: A single neuron estimating precision or F1-Score based on experimental configuration. The activation function is the sigmoid function across all layers.

3.4 Training Fitness Function

To achieve robust extractive text summarization, we created a diverse set of training examples, which helps our neural network understand not only the perfect summary but also different imperfect summaries. This diversity allows us to explore how variations impact precision and F1-Scores.
Training of Fitness over Precision and F1 Score: Recognizing the nuanced evaluation offered by F1-Score, which incorporates both precision and recall, we extended our experiments beyond precision alone and ran experiments for both precision and F1-Score as the output label for the proposed neural network facilitating computation of *f*.
Generating Training Dataset: For a specific document we start with an ideal summary represented as a series of 0s and 1s. Variations are brought in the represented summary using a method we termed as Flip-Distribution. This process systematically switches an equal number of '0's to '1's and vice versa, creating a diverse set of ideal summary varia-

tions. Additionally, we include randomly generated summaries where each sentence in the document has an equal chance (probability of 0.5) of being included in the summary. Different candidate summaries are built this way to ensure that the neural network encounters a wide range of potential summaries during training, improving its ability to generalize and generate effective extractive summaries. Precision and F1-Score are computed for each candidate summary, serving as labels in the two experiments.

Flip-Distribution: In constructing the Flip-Distribution for candidate summary generation, a deliberate focus is placed on producing a higher number of 1-flip summaries, minimizing variation from the ideal summary. This strategic choice ensures the neural network encounters a substantial proportion of examples with minimal deviation during training, and the dataset leans towards preserving the structure and content of the ideal summary, enhancing the model's ability to generate accurate extractive summaries.

Consider a document considering of N = 6 sentences, with the desired summary represented as "001011"(indicating S = 3). To build a dataset of 10 candidate summaries for each document, we employ the Flip-Distribution method, resulting in the following distribution:

$$\text{Flip-Distribution} = \{3 \text{ random, } 3 \text{ 1-flip, } 2 \text{ 2-flips, } 1 \text{ 3-flips, } 1 \text{ ideal}\}$$

(i) 3 *random* : A summary of length 6 is generated where each bit can be 1 with a probability of 0.5. Total 62 ($2^6 - 2$, excluding null summary - 000000 and complete document - 111111) possibilities, such as "111001", "010101", "100001" and "101100". Choose three such summaries.
(ii) 3 *1-flip* : Randomly select a bit in ideal summary and flip its parity. Total $\binom{6}{1}$ i.e. 6 possibilities, such as "101011", "011011", 000011, and "001001". Choose three such summaries.
(iii) 2 *2-flips* : Randomly select two bits in ideal summary and flip their parity. Total $\binom{6}{2}$ i.e. 15 possibilities, such as "111011", "001000", "011001" and "000111". Choose two such summaries.
(iv) 1 *3-flips* : Randomly select three bits in ideal summary and flip their parity. Total $\binom{6}{3}$ i.e. 20 possibilities, such as "000101", "011000", "010111" and "100001". Choose one such summary.
(v) 1 *ideal* : Include the ideal summary itself.
For the given document, consider the following candidate summary *Summ*:

$$\begin{aligned} \text{Summ} &= \text{“10}\mathbf{1}\text{000"} \\ \text{Ideal\_Summary} &= \text{“00}\mathbf{1}\text{011"} \end{aligned}$$

With one common sentence out of two in the candidate summary and three in the ideal summary, the performance metrics are:

$$(\text{Precision, Recall, F1-Score}) = \left(\frac{1}{2}, \frac{1}{3}, \frac{2}{5}\right)$$

3.5 Generating a Sample Dataset from BBC Documents

To train the neural network, a dataset was created using a sample of 100 documents from BBC dataset. Five summaries were generated for each document using Flip-Distribution, comprising four bit-flipped summaries and one ideal summary. This resulted in a total dataset size of 500.

For each candidate summary, eight features and precision (later, F1-Score labels in subsequent experiments) were calculated by comparing it with the original document and its ideal extractive summary, represented as a binary string. The first column ($\#S$), indicates the row entry in the created dataset. The second column ($\#D$), represents the ordered document number. These documents are manually selected from the BBC dataset. Given that five candidate summaries were generated per document, there are five rows corresponding to each document number. A representative subset of 10 values is provided in Table 1.

After training the neural network on this dataset, it serves as the fitness function for the evolutionary algorithms. Multiple experiments were conducted, exploring various dataset sizes and Flip-Distribution configurations to ensure a robust fitness function. Results are presented in Section 6.

4 Evolutionary Algorithms

Evolutionary algorithms represent a heuristic-based approach for addressing optimization problems. Successive iterations tend to yield improved solution sets. Although certain evolutionary algorithms were initially developed for continuous solution spaces, for the present work we have adapted them to accommodate the discrete nature of text summaries. In this section, we discuss the basics of the evolutionary algorithms used in our work. We cover Firefly, PSO, and Cuckoo algorithms. For Genetic (Chatterjee et al., 2019) and DDE (Karwa

#S	#D	CandidateSummary	RTS	ATS	SF	CF	RF	AS	SP	k_by_N	Precision
1	1	10100011001010000000	1.699	2.927	0.593	0.223	0.323	0.33	0.366	0.3	1
2	1	00010000010110110000	1.802	2.85	0.593	0.223	0.086	0.22	0.238	0.3	0.167
3	1	11100111000000000000	1.762	2.472	0.468	0.223	0.148	0.385	0.442	0.3	0.667
4	1	00001111000001001000	1.764	2.883	0.5	0.223	0.02	0.298	0.3	0.3	0.334
5	1	10000101101010000000	1.69	3.018	0.5	0.223	0.094	0.259	0.342	0.3	0.667
...	...	...	...	...	...	...	...	...	...	...	...
496	100	10000001100011010000000	1.8	2.85	0.4	0.122	0.127	0.396	0.278	0.26	1
497	100	10101001010000010000000	1.851	3.698	0.2	0.122	0.087	0.212	0.344	0.26	0.5
498	100	10101000000100101000000	1.853	3.151	0	0.122	0.12	0.287	0.307	0.26	0.167
499	100	10111101000000000000000	1.905	3.708	0.35	0.122	0.1	0.221	0.402	0.26	0.334
500	100	11000111100000000000000	1.918	3.474	0.55	0.122	0.097	0.235	0.38	0.26	0.5

Table 1: Representative Dataset

and Chatterjee, 2014) algorithms, we adopt implementations from existing studies to compare their performance with our adapted evolutionary approaches.

4.1 Firefly Algorithm

Firefly Algorithm (Tomer and Kumar, 2021) draws inspiration from the behavior of fireflies, where each firefly tend to move towards the ones emitting higher light intensity. In the context of text summarization task, each firefly represents a potential solution i.e a summary vector. The light intensity of that firefly corresponds to the fitness function. The algorithm is described below.

Initialization: A population of size equal to $popSize$ fireflies is generated randomly. Each firefly in the population represents a possible solution (candidate summaries), i.e N length binary string with m number of 1. The light intensity is calculated for each firefly using the fitness function. To calculate the distance between two fireflies, Hamming distance function is used. The larger is the unmatched number of bits the more distinct are the summaries. The Hamming Distance is defined as follows:

$$\|\mathbf{x}_i - \mathbf{x}_j\| = \sum_{l=1}^{N} \delta(x_{i,l}, x_{j,l})$$

$$\text{where } \delta(x_{i,l}, x_{j,l}) = \begin{cases} 0, & \text{if } x_{i,l} = x_{j,l} \\ 1, & \text{if } x_{i,l} \neq x_{j,l} \end{cases}$$

$x_{i,l}$ represents the l^{th} bit of the i^{th} firefly, $x_{j,l}$ represents the l^{th} bit of the j^{th} firefly.

Updating of firefly positions: In this step each firefly adjusts its position seeking proximity to brighter fireflies within the population. The update involves calculating the movement of a firefly towards another based on their respective brightness and the distance between them. The movement in a particular iteration is the cumulative effect of all movements towards fireflies brighter than the current one. The update formula for a firefly x_i towards a brighter firefly x_j in one iteration is given by:

$$x_{i,l} = \begin{cases} x_{j,l}, & \text{if } r_{i,l} < \frac{f(x_i)}{1+\gamma \cdot \|\mathbf{x}_i - \mathbf{x}_j\|^2} \\ x_{i,l}, & \text{otherwise} \end{cases}$$

where $r_{i,l}$ is a randomly chosen number from range [0, 1] for each bit, γ a hyper parameter is used to adjust the distance between two fireflies and $f(x_i)$ represent the fitness value for x_i. The fireflies' positions are replaced with the newly generated ones following the update rule.

Bit Re-Balancing: After completing a iteration bit re-balancing is done for each firefly to maintain a constant summary length m. If a firefly's summary contains more 1's than the desired length, specified number of randomly selected 1's are flipped to 0, and vice versa.

Mutation: To avoid local maxima, the top fireflies undergo mutation using a conventional bit-flipping technique. Pairs of bits, a '0' and a '1', are selected and their values are flipped, maintaining a constant count of 1's in candidate summaries. This iterative process continues until the specified stopping condition is reached, such as reaching the maximum number of iterations.The detailed pseudo-code for the algorithm is given in Appendix.

4.2 Particle Swarm Optimization

In this method, several particles are initiated, each aiming to discover the best value for a given fitness function by navigating through the solution space, attempting to converge on a global optimum (Asgari et al., 2014). The particles not only move towards their individual best solutions, but also gravitate towards the best overall position identified till now. This prevents the algorithm from getting stuck in a local optimum.

Initialization: A swarm of particles is generated randomly, akin to GA. Each particle in the population represents a possible solution i.e. a ran-

dom binary string of size N with m number of 1's. The value of a position is calculated using the fitness function. The goal is to optimize the fitness function by identifying the optimal combination of binary values embodied in the particles.

Updating of particle position: Each swarm, in order to find its target makes a move towards the best possible direction in which the target could be found. For this it moves towards the global best location found till now along with the local best found by itself. A simple probabilistic way to update the particle position is chosen. Two movement probabilities are calculated for each bit per particle.

1. Movement towards Local Best: This is calculated by multiplying a randomly calculated probability with an hyper parameter signifying acceleration towards local best solution for that particle found so far.

2. Movement towards Global Best: This is calculated by multiplying a randomly calculated probability with an hyper parameter signifying acceleration towards global best solution.

The combined effect of two probabilities determine whether the bit would be flipped or not. Exact formulae and the corresponding pseudo-code can be found in the Appendix.

Mutation: The traditional technique of bit alteration i.e. selecting a pair of bit, a 0 and a 1, and flipping its value is followed. This keeps the number of 1's constant in the candidate summaries.

4.3 Cuckoo Search

Cuckoo Search (CS) is an optimization algorithm inspired by cuckoos' brood parasitism behavior. Cuckoos deposit eggs strategically, selecting host bird nests' with recent eggs and replacing existing ones with theirs'. Some host birds counteract this parasitic behavior by rejecting foreign eggs. The CS algorithm (Cuevas and Reyna Orta, 2014) simplifies this natural process into three rules for computational modeling:

1. Artificial cuckoos lay eggs one at a time in a nest and each egg represents a solution.
2. Cuckoos seek optimal nests to maximize egg survival, employing an elitist selection strategy favoring high-quality eggs.
3. The fixed population of host nests may discard alien eggs with a certain probability. Mature eggs progress to the next generation, selecting nests via Levy flights around current best solutions.

The original CS algorithm (Cuevas and Reyna Orta, 2014) has been adapted for our discrete task of text summarization and has the following steps:

A. Initialization: Here too the aim is to construct a summary of size m sentences is required for a document containing N sentences. The population E is initialized with *popSize* eggs, each egg representing a candidate summary. The population is evolved over *maxGen* generations. E^k $(e_1^k,\ e_2^k, ..,\ e_{popSize}^k)$ represents the population at k^{th} generation, where e_i^k is a binary vector of length N where each dimension is equal to $'1'$ or $'0'$ depending on if the corresponding sentence is included in the summary or not. Initially randomly m chosen dimensions among N are initialized as $'1'$ and rest are $'0'$.

B. Levy Flight: In Cuckoo search, Levy flights generate new candidate summaries (eggs) through the following process:

$$e_i^{k+1} = e_i^k + c_i \quad \text{for } i = 1, 2, ..., popSize$$

Here, c_i denotes a random step, reflecting a change in position within the original summary/egg. It is computed as:

$$c_i = \alpha \cdot s_i \odot (e_i^k - e^{best})$$

In this equation, α is a hyperparameter (set as 0.01), e^{best} represents the best summary/egg observed thus far, and s_i is generated by a symmetric Levy distribution using Mantegna's algorithm (Cuevas and Reyna Orta, 2014).

C. Replacement of some eggs: Each individual, e_i^k $\forall$ i $\in$ [1,2,..,popSize], can be selected with a probability of p_a and then replaced with a new solution. This operation is analogous to the mutation step in Genetic Algorithm which introduces diversity and explore new regions of the solution space.

$$e_i^{k+1} = \begin{cases} e_i^k + rand \cdot (e_{d_1}^k - e_{d_2}^k) & \text{with prob } p_a \\ e_i^k & \text{with prob } 1 - p_a \end{cases}$$

where rand is a random number normally distributed, whereas $d1$ and $d2$ are distinct random integers from 1 to $popSize$.

D. Elitist selection: After generating the solution e_i^{k+1}, it competes with its predecessor e_i^k based on fitness, and the fitter one advances to the next generation.

$$e_i^{k+1} = \begin{cases} e_i^{k+1} & \text{if } f(e_i^{k+1}) < f(e_i^k) \\ e_i^k & \text{otherwise} \end{cases}$$

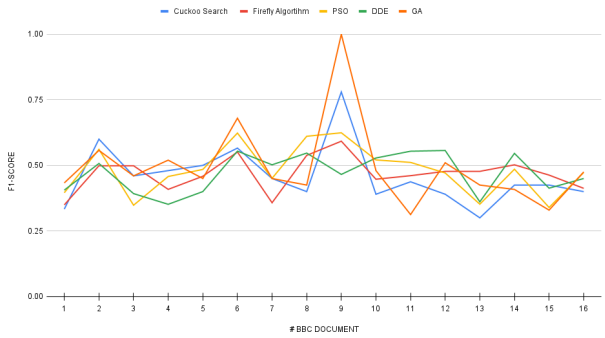

Figure 2: Comparison and Correlation between Evolutionary Algorithms

E. Squishification: After generating the new solution e_i^{k+1} via Levy flights and replacement step we obtain a $N-dimensional$ vector with real values. Since our problem of text summarization of being discrete nature we pass it through a squishification function as follows:

$$e_{i,j}^{k+1} = \begin{cases} 1 & \text{if} \dfrac{1}{1+exp(-e_{i,j}^{k+1})} < threshold \\ 0 & \text{otherwise} \end{cases}$$

$\forall i \in [1,2,..,popSize] \quad and \quad \forall j \in [1,2,..,N]$, where $threshold$ a real value in the range $[0,1]$. Where $e_{i,j}^{k+1}$ represents the j^{th} dimension of i^{th} egg in $(k+1)^{th}$ generation.

Summary length adjustment: After squishification, if the count of '1's in the new solution e_i^{k+1} is not equal to the desired summary length m, random bit flips are introduced at specific indices to maintain consistency. The detailed pseudo-code is given in appendix.

5 Results

We implemented and experimented with five different Evolutionary algorithms namely Genetic, Discrete differential evolution, Firefly, Particle swarm optimization and Cuckoo Search over BBC news articles data-set which consists of around 2250 documents with the available ideal summaries in five different classes namely business, entertainment, sports, politics, and tech. The dataset was created from BBC Documents by generating variations for each document, such as random changes, flip-based alterations, and maintaining an ideal summary for comparison, thereby forming a diverse dataset for experiments.

Since all the algorithms are a part of randomized algorithms, the experiments were run 15 times per algorithm per document and average precision/F1-Score value is considered. The test data-set consisted of 16 documents randomly sampled from the complepte corpus. A total of 10 experiments were conducted, with 4 focusing on precision and 6 on using F1-Score as the label. Table 2 and 3 summarizes the results across various experiments, where *#Epochs* value indicates the number of epochs for which the neural network has been trained, *#Documents* is the number of documents taken to generate the training data-set and [G, D, F, P, C] corresponds to [GA, DDE, Firefly, PSO, Cuckoo].

Inferring from the result tables, the best **Results** are corresponding to **Experiment 4** from **Table 2** where neural network is trained over 750 documents with **precision** as the label. Table 4 illustrates the expanded results of Experiment 4 in comparision to ChatGPT.

6 Conclusions and Discussions

In this study, we explored the application of evolutionary algorithms to the task of text summarization and compared their performance with a state-of-the-art language model, ChatGPT. The evaluation was conducted on a dataset of 16 documents, and F1-Score values were used as the performance metric.

Our results revealed that EAs exhibited promising performance in the context of text summarization. These EAs achieved competitive F1-Scores, with an average F1-Score of **0.4947** across all documents. Notably, on 4 out of the 16 documents (highlighted rows in Table 4), EAs outperformed ChatGPT, indicating their efficacy in generating concise and informative text summaries.

It's worth highlighting that ChatGPT, a language model, has been trained on an extensive corpus of text from various domains, while our model was trained on a comparatively smaller dataset. Despite this disparity, EAs demonstrated their potential as viable alternatives for text summarization tasks.

Furthermore, our analysis unveiled an interesting trend (Fig. 2). The performance of different EAs demonstrated a high degree of correlation. When the F1-Score increased for one EA, it generally increased for others as well, and vice versa. This suggests that these EAs share common strengths and weaknesses when applied to text summarization tasks.

# EXP	# Documents	# Candidate Summaries per document	Flip_Distribution	Fitness Value Taken Precision	# Epochs	Average F1-Score over all documents on test data [G, D, F, P, C]
1	100	20	Randomly Generated	Precision	100	[0.3, 0.4, 0.4, 0.44, 0.47]
2	1900	10	Randomly Generated	Precision	1000	[0.33, 0.3, 0.35, 0.28, 0.33]
3	750	30	15 Randomly Generated 5 1-flip 4 2-flip 3 3-flip 2 4-flip 1 Ideal Summary	Precision	2000	[0.35, 0.36, 0.3, 0.29, 0.3]
4	750	30	15 1-flip 10 2-flip 4 3-flip 1 Ideal Summary	Precision	1000	[0.49, 0.47, 0.46, 0.48, 0.45]

Table 2: Experimentation results across Precision models

# EXP	# Documents	# Candidate Summaries per document	Flip_Distribution	Fitness Value Taken F1-Score	# Epochs	Average F1-Score over all documents on test data [G, D, F, P, C]
1	750	30	15 1-flip 15 2-flip 4 3-flip 1 Ideal Summary	F1Score	1000	[0.36, 0.31, 0.2, 0.3, 0.27]
2	750	36	15 1-flip 12 2-flip 8 3-flip 1 Ideal Summary	Precision	2000	[0.3, 0.36, 0.35, 0.4, 0.35]
3	500	40	15 1-flip 12 2-flip 8 3-flip 4 4-flips 1 Ideal Summary	F1Score	800	[0.37, 0.27, 0.36, 0.47, 0.43]
4	1000	36	15 1-flip 12 2-flip 8 3-flip 1 Ideal Summary	F1Score	1000	< 0.2
5	1500	40	15 1-flip 12 2-flip 8 3-flip 4 4-flips 1 Ideal Summary	F1Score	800	< 0.2
6	150	30	13 1-flip 12 2-flip 4 3-flip 1 Ideal Summary	F1Score	100	[0.35, 0.36, 0.2, 0.1, 0.28]

Table 3: Experimentation results across F1-Score models

Document No.	Cuckoo	Firefly	PSO	DDE	GA	ChatGPT
1	0.3333	0.3496	0.3958	0.4058	**0.4333**	0.5454
2	**0.6000**	0.4987	0.5612	0.5073	0.5571	0.6777
3	0.4600	**0.4987**	0.3481	0.3865	0.4600	0.4600
4	0.4800	0.4086	0.4577	0.3516	**0.5200**	0.5454
5	**0.5000**	0.4585	0.4846	0.4250	0.4500	0.7693
6	0.5660	0.5510	0.6233	0.5544	**0.6800**	0.7693
7	0.4500	0.3576	0.4507	**0.5023**	0.4500	0.5455
8	0.4000	0.5382	**0.6110**	0.5470	0.4250	0.7261
9	0.7800	0.5923	0.6244	0.4656	**1.0000**	0.4444
10	0.3900	0.4473	0.5209	**0.5364**	0.4800	0.5555
11	0.4375	0.4608	0.5114	**0.5541**	0.3125	0.6301
12	0.3900	0.4772	0.4711	**0.5567**	0.5100	0.5130
13	0.3000	**0.4772**	0.3520	0.3611	0.4250	0.5815
14	0.4250	0.5021	0.4854	**0.5458**	0.4083	0.3294
15	0.4250	**0.4630**	0.3395	0.4132	0.3294	0.4962
16	0.4000	0.4125	0.4731	0.4500	**0.4750**	0.7578
Average	0.4586	0.4683	0.4818	0.4727	**0.4947**	0.5635

Table 4: F1-Score values for each of the five algorithms corresponding to the model trained in experiment 4 (Table 2) in comparison with ChatGPT

Limitations and Future Work

The major limitations include:
(1) Limited Exploration: While we have experimented with several evolutionary algorithms, there are various others that we did not explore.
(2) Data Set Constraints: Our experiments were conducted on a restricted set of data. Future plans involve testing with more diverse and relevant datasets.
(3) Extractive Focus: Currently, our work is centered on extractive summarization. We aim to extend our approach to generate abstractive summaries, requiring the development of new fitness functions and features.
Our future work will delve deeper into understanding the factors influencing the performance of evolutionary algorithms. We also plan to explore hybrid approaches that combine the strengths of both evolutionary algorithms and language models for enhanced text summarization.

Ethics Statement

This research adheres to the ACL Ethics Policy, and in compliance with ACL 2024 guidelines, we provide the following ethics statement. We explicitly consider the broader impact of our work and address relevant ethical considerations. We followed the Responsible NLP Research checklist and ACL code of ethics for this work.

Acknowledgements

The authors gratefully acknowledge the dataset provider, BBC Documents and Summary Dataset, for granting access to their valuable data, a critical component in the completion of this research.

References

Hamed Asgari, Behrooz Masoumi, and Omid Sojoodi Sheijani. 2014. Automatic text summarization based on multi-agent particle swarm optimization. In *2014 Iranian Conference on Intelligent Systems (ICIS)*, pages 1–5.

Munehs Chandra, Vikrant Gupta, and Santosh Kr. Paul. 2011. A statistical approach for automatic text summarization by extraction. *2011 International Conference on Communication Systems and Network Technologies*, pages 268–271.

Niladri Chatterjee, Gautam Jain, and Gurkirat Singh Bajwa. 2019. Single document extractive text summarization using neural networks and genetic algorithm. In *Intelligent Computing*, pages 338–358, Cham. Springer International Publishing.

Niladri Chatterjee and Pramod Sahoo. 2014. Random indexing and modified random indexing based approach for extractive text summarization. *Computer Speech Language*, 29.

Erik Cuevas and Adolfo Reyna Orta. 2014. A cuckoo search algorithm for multimodal optimization. *The Scientific World Journal*, 2014:27.

Taeho Jo. 2017. K nearest neighbor for text summarization using feature similarity. In *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pages 1–5.

Geetha J K and Deepamala N. 2015. Kannada text summarization using latent semantic analysis. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1508–1512.

Shweta Karwa and Niladri Chatterjee. 2014. Discrete differential evolution for text summarization. In *2014 International Conference on Information Technology*, pages 129–133.

Shweta Karwa and Niladri Chatterjee. 2015. Discrete differential evolution for text summarization. *Proceedings - 2014 13th International Conference on Information Technology, ICIT 2014*, pages 129–133.

Soe Soe Lwin and Khin Thandar Nwet. 2019. Extractive myanmar news summarization using centroid based word embedding. *2019 International Conference on Advanced Information Technologies (ICAIT)*, pages 200–205.

J. Naga Madhuri and R. Ganesh Kumar. 2019. Extractive text summarization using sentence ranking. *2019 International Conference on Data Science and Communication (IconDSC)*, pages 1–3.

Ani Nenkova, Sameer Maskey, and Yang Liu. 2011. Automatic summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, page 3, Portland, Oregon. Association for Computational Linguistics.

Finn Årup Nielsen. 2011. A new anew: Evaluation of a word list for sentiment analysis in microblogs. In *#MSM*.

Vahed Qazvinian, Leila Sharif Hassanabadi, and Ramin Halavati. 2008. Summarising text with a genetic algorithm-based sentence extraction. *International Journal of Knowledge Management Studies*, 2:426.

Alvee Rahman, Fahim Rafiq, Ramkrishna Saha, Ruhit Rafian, and Hossain Arif. 2019. Bengali text summarization using textrank, fuzzy c-means and aggregate scoring methods. pages 331–336.

Sheetal Shimpikar and Sharvari Govilkar. 2017. A survey of text summarization techniques for indian regional languages. *International Journal of Computer Applications*, 165(11):29–33.

Minakshi Tomer and Manoj Kumar. 2021. Multi-document extractive text summarization based on firefly algorithm. *Journal of King Saud University - Computer and Information Sciences*, 34.

A Algorithms

Algorithm 1 Genetic Algorithm for Text Summarization

Input: Document with N sentences, Summary length m, population size $popSize$, max. number of generations to evolve $maxGen$ Stopping criteria
Output: Optimal summary of the document
Hyperparameters: $popSize = 25$, $maxGen = 20$

//Population Initialization
$parents \leftarrow$ empty list
for $i \leftarrow 1$ **to** $popSize$:
    $string =$ randomly generate binary string of length N with m many $1's$
    add $string$ to $parents$
for $i \leftarrow 1$ **to** $maxGen$:
    $children \leftarrow$ empty set
    $parents\_fitness = fitness\_function(\ parents\ )$
    sort($parents\_fitness$, descending order)
    //Elitist selection strategy
    add best and second best parent to $children$

    while (size of $children$ < size of $parents$) :
        $parent1$, $parent2 \leftarrow$ Choose via Roulette based selection
        $child1$, $child2 = perform\_crossover($ parent1, parent2)
        $child1 = perform\_mutation(child1)$
        $child2 = perform\_mutation(child2)$
        add $child1,\ child2$ **to** $children$

    $combined\_parents\_children = concatenate\ parents\ and\ children$
    new_generation_fitness $= fitness\_function(\ combined\_parents\_children\ )$
    sort($new\_generation\_fitness$, descending order)
    $parents$ = first n individuals from $new\_generation\_fitness$

return $select\_best\_summary(\ parents\ )$

Algorithm 2.1 Discrete Differential Evolution for Text Summarization

Input: Document with n sentences, number of clusters k, population size $popSize$, crossover rate *CR*, maximum number of generations to evolve $maxGen$, sclaing factor λ
Output: Optimal summary of the document
Hyperparameters: $popSize = 25$, $CR = 0.6$, $maxGen = 30$, $\lambda = 1$
//Initialize population
for $i \leftarrow 1$ **to** N **do**
    for $j \leftarrow 1$ **to** k **do**
        $X_{i,j}$ = randInt(1,k)
while $(!StoppingCondition)$ **do**
    for $l \leftarrow 1$ **to** N **do**
        Randomly choose i, j, and m from 1 to N such that $i \neq j \neq l \neq m$
        for $r \leftarrow 1$ **to** n **do**
            if $rand(0, 1) \leq CR$ **then**
                $X'_{l,r}$ = $\lambda(X_{i,r} - X_{j,r}) + X_{m,r}$
                $X'_{l,r}$ = $int(abs(X'_{l,r}))$
                if $X'_{l,r} < 1$ or $X'_{l,r} > k$ **then**
                    $X'_{l,r}$ = randInt(1,k)
        if $fitness(X'_l) > fitness(X_l)$ **then**
            $nextX_l$ = X'_l
        else
            $nextX_l$ = X_l
    X_l = $nextX$
return GetBestSolution(X)

Algorithm 2.2 GetBestSolution(X): Generating Summary using DDE Chromosome

Input: DDE chromosome Y, TF-IDF matrix of N sentences of document W
Output: Summary represented by the DDE chromosome

$summary \leftarrow$ empty list $\quad clusters \leftarrow$ empty dictionary
//Assign sentences to clusters based on chromosome Y
for i, $cluster$ in enumerate(Y):
    if $cluster$ not in $clusters$:
        $clusters[cluster] = [\,]$
    $clusters[cluster].append(W[i])$

//Generate summary from each cluster
for $cluster$ in $clusters$:
    $cluster\_sentences = clusters[cluster]$

    //Calculate the centroid of the cluster
    $centroid$ = average of TF-IDF vectors of all sentences in cluster
    $max\_similarity = -1$
    $representative\_sentence = None$

    //Find the sentence with maximum similarity to the centroid
    for $sentence$ in $cluster\_sentences$:
        $similarity$ = cosine similarity between the sentence and centroid
        if $similarity$ >$similarity$:
            $max\_similarity = similarity$
            $representative\_sentence = sentence$
    $summary.append(representative\_sentence)$
$return\ summary$

Algorithm 3 Firefly Algorithm for Text Summarization

Input: Document with N sentences, population size $popSize$, maximum number of iterations to move $maxIt$
Output: Optimal summary of the document
Hyperparameters: $popSize = 120$, $maxIt = 10$, $\gamma = 0.01$

Initialization:
for $i \leftarrow 1$ **to** $popSize$:
    $X_i \leftarrow$ randomly generated binary string of length N
    $I_i \leftarrow f(X_i)$
end for
$X \leftarrow$ list of $popSize$ binary strings $[X_1, X_2, \ldots, X_{popSize}]$
$I \leftarrow$ list of $popSize$ fitness values $[I_1, I_2, \ldots, I_{popSize}]$
$R \leftarrow$ matrix of hamming distances between fireflies in X
for $i \leftarrow 1$ **to** $popSize$ **do**
    for $j \leftarrow 1$ **to** $popSize$ **do**
        $R[i][j] \leftarrow \|\mathbf{x}_i - \mathbf{x}_j\|$

Updation:
$best\_summary \leftarrow$ randomly generated binary String of length N
for $t \leftarrow 1$ **to** $maxIt$ **do**
    for each firefly i **do**
        for each firefly j **do**
            if $I[i] < I[j]$ **then**
                for $l \leftarrow 1$ **to** N **do**
                    $r_{i,l} \leftarrow$ a random number $\sim$ Uniform$(0, 1)$
                    if $r_{i,l} < \frac{I_i}{1+\gamma \cdot (R[i][j])^2}$ **then**
                        Update position: $X_{i,l} \leftarrow X_{j,l}$ ▷ Move towards a brighter firefly
                    else
                        Update position: $X_{i,l} \leftarrow X_{i,l}$ ▷ Stay in the current position
    Perform $bit\ re\text{-}balancing$ for all fireflies
    Mutate the best fireflies // Traditional random bit-flipping Mutation technique
    Update I with fitness value of new firefly positions
    Update R with hamming distance between new firefly positions
    $best\_summary \leftarrow X_i$ where $i = \arg\max(I)$
return $best\_summary$

Algorithm 4 PSO Algorithm for Text Summarization

Input: Document with N sentences, population size $popSize$, maximum number of iterations to move $maxIt$
Output: Optimal summary of the document
Hyperparameters: $popSize = 30$, $maxIt = 5$, $\omega = 1$, $acc\_towards\_local\_best = 1$, $acc\_towards\_global\_best = 1.5$

Initialize $X \leftarrow$ the population of particles
Initialize $local\_best \leftarrow$ array of personal best position for each particle
Initialize $global\_best \leftarrow$ global best position
Initialization:
for each particle in the population **do**
    Generate a random binary string as the initial position
    Initialize the personal best position as the initial position
    Evaluate the fitness of the initial position
    if the fitness is better than the global best fitness **then**
        Update the global best position
Initialize the current iteration count to 0
Updation:
while the maximum number of iterations is not reached **do**
    for each particle in the population **do**
        for each dimension d in the document size **do**
            $v_{ij} = 0$
            if $X_{ij} =' 1'$ **then**
                $v_{ij} += \omega$
                if local_best$[i][j] =' 0'$ **then**
                    $t_1 =$ random.uniform$(0, 1)$
                    $v_{ij} -= t_1 \times acc\_towards\_local\_best$
                if global_best$[j] =' 0'$ **then**
                    $t_2 =$ random.uniform$(0, 1)$
                    $v_{ij} -= t_2 \times acc\_towards\_global\_best$
                if $v_{ij} < 0$ **then**
                    $X_{ij} =' 0'$
                else
                    $X_{ij} =' 1'$
            else
                if local_best$[i][j] =' 1'$ **then**
                    $t_1 =$ random.uniform$(0, 1)$
                    $v_{ij} += t_1 \times acc\_towards\_local\_best$
                if global_best$[j] =' 1'$ **then**
                    $t_2 =$ random.uniform$(0, 1)$
                    $v_{ij} += t_2 \times acc\_towards\_global\_best$
                if $v_{ij} > 1$ **then**
                    $X_{ij} =' 1'$
                else
                    $X_{ij} =' 0'$
        Mutate using traditional Mutation technique
        Evaluate the fitness of the new position
        if the fitness is better than the personal best fitness **then**
            Update the personal best position
            if the fitness is better than the global best fitness **then**
                Update the global best position
    Increment the current iteration count
return Summary with Global Best Position

Algorithm 5 Cuckoo Search for Text Summarization

Input: Document with N sentences, Summary length m, population size $popSize$, max. number of generations to evolve $maxGen$/ Stopping criteria
Output: Optimal summary of the document
Hyperparameters: $popSize = 25$, $p_a = 0.35$, $maxGen = 15$, $threshold = 0.5$

//Population Initialization
$E^0 \leftarrow popSize$ number of candidate eggs each of dimension N
while (!Stopping Criteria) :
    $E^{k+1} \leftarrow OperatorB(E^k)$ //Operator B: Levy Flight
    $E^{k+1} \leftarrow OperatorE(E^{k+1})$ //Operator E: Squishification
    $E^{k+1} \leftarrow OperatorD(E^k, E^{k+1})$ //Operator D: Elitist Selection
    $E^{k+2} \leftarrow OperatorC(E^{k+1})$ //Operator C: Replacement
    $E^{k+2} \leftarrow OperatorE(E^{k+2})$ //Operator E: Squishification
    $E^{k+1} \leftarrow OperatorD(E^{k+1}, E^{k+2})$ //Operator D: Elitist Selection
return $select\_best\_summary\ (E^{\,final\_generation})$