GenDLN: Evolutionary Algorithm-Based Stacked LLM Framework for Joint Prompt Optimization

Anonymous ACL submission

Abstract

With Large Language Model (LLM)-based applications becoming more common due to strong performance across many tasks, prompt optimization has emerged as a way to extract better solutions from frozen, often commercial LLMs that are not specifically adapted to a task. LLM-assisted prompt optimization methods provide a promising alternative to manual/human prompt engineering, where LLM "reasoning" can be used to make them optimizing agents. However, the cost of using LLMs for prompt optimization via commercial APIs remains high, especially for heuristic methods like evolutionary algorithms (EAs), which need many iterations to converge, and thus, tokens, API calls, and rate-limited network overhead. We propose GenDLN, an efficient genetic algorithm-based prompt pair optimization framework that leverages commercial API free tiers. Our approach allows teams with limited resources (NGOs, non-profits, academics...) to efficiently use commercial LLMs for EA-based prompt optimization. We conduct experiments on CLAUDETTE for legal terms of service classification and MRPC for paraphrase detection, performing in line with selected prompt optimization baselines, at no cost. Our code is available in **<omitted>**.

1 Introduction

LLMs (large language models) are increasingly replacing traditional classification and inference models due to their generality, ability to perform a wide range of tasks, and seemingly advanced "reasoning." As the use of LLMs for domain-specific tasks becomes more ubiquitous, prompt optimization emerges as an important area of research to improve the task-specific performance of LLMs, especially in complex domains like legal text analysis and interpretation (Hakimi Parizi et al., 2023; Lai et al., 2024). In recent years, several prompt design and optimization techniques have been proposed.

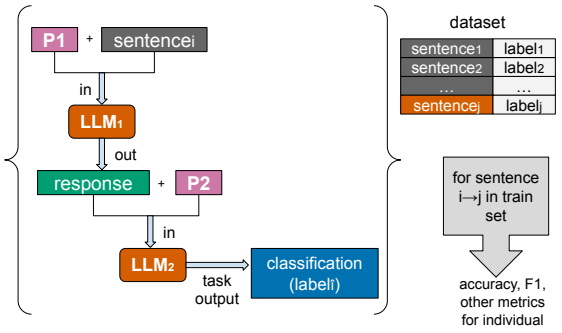

Figure 1: Running an individual through the DLN, where an individual is a prompt pair (p_1, p_2); LLM_1 responds to p_1, and this response, along with p_2, is fed into LLM_2 for classification. E.g.: p_1: "Interpret <ToS sentence i>" - p_2: "Based on the above interpretation, classify <ToS sentence i> as fair or unfair."

Some examples are edit-based instruction search GRIPS (Prasad et al., 2023) and reflection-based frameworks that incorporate LLM self-critique such as ProTeGi (Pryzant et al., 2023) and OPRO (Yang et al., 2024).

Deep Language Networks (DLNs) is a novel approach that stacks LLMs as computational units (Sordoni et al., 2023). Like other prompt optimization methods, the goal is to use frozen-weight LLMs for inference while refining input prompts for better results. However, they stack two LLMs, jointly optimizing two input prompts, where the output of the first LLM, along with the second prompt, are fed into the second LLM, as shown in Fig. 1. The prompts are treated as learnable parameters of the generative distribution, and the prompt pair is jointly optimized using variational inference.

We introduce our framework, GenDLN, where we retain the stacked LLM structure and joint prompt optimization introduced in DLN, but replace the variational inference-based optimization with a Genetic Algorithm (GA) (Fig. 2). The ad-

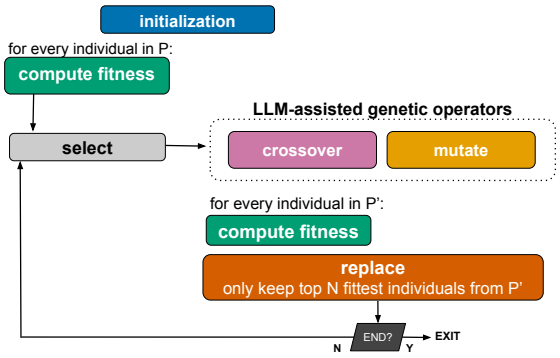

Figure 2: High-level GenDLN Optimization Framework. Initialization starts from a bank of manual prompts, with optional LLM augmentation. Selection, crossover, and mutation follow the chosen strategies. P: starting population. P': population post-genetic operators.

vantage of using a GA is the ability to explore a large search space and end up with a large pool of candidate prompts. We apply our framework to legal document classification, which aims to categorize legal documents into predefined classes, specifically, Terms of Service (ToS) classification. Also known as Terms and Conditions or Terms of Use, ToS are legal agreements between a service provider and its users, sometimes employing deliberately confusing language (Yerby and Vaughn, 2022), or featuring unfair clauses to users (Loos and Luzak, 2021). Due to ToS length and complexity, users often accept them without fully reading them. To that end, automated unfair clause detection allows consumers to better assess ToS in less than the 45 minutes required to completely read an average ToS agreement (Obar and Oeldorf-Hirsch, 2020). We run further tests on the Microsoft Research Paraphrase Corpus (MRPC) for paraphrase detection.

Our contributions include a GA framework that successfully improves a population of prompt pairs for classification across several runs and parameter sets, performing in line with state-of-the-art prompt optimization methods. More importantly, our main contribution is an efficient, parameter-rich, LLM-based genetic algorithm framework for text editing that tackles several problems of applying GAs to prompt optimization, including the bottleneck of using API calls for prompt scoring and the additional overheads and limitations imposed by commercial LLM providers. GenDLN can be used by teams with limited resources to quickly generate a pool of optimized prompts for a given task.

2 Background

2.1 Prompt Optimization

Prompt optimization is the process of systematically refining or designing the textual instructions (prompts) that guide a Large Language Model toward producing higher-quality, task-specific outputs. Various prompt optimization methods have emerged in recent years. Reflection-based frameworks (Pryzant et al., 2023; Ma et al., 2024) collect error feedback or "textual gradients" from LLM output, then edit prompts accordingly, while edit-based approaches (Prasad et al., 2023) iteratively rewrite instructions using operations such as paraphrasing and swapping. Some methods take a meta-prompts approach (Yang et al., 2024), dynamically updating instructions based on historical performance. Additionally, evolutionary algorithm–driven solutions (Guo et al., 2024) simulate natural selection and evolve a population of prompts across generations. All these methods share the same objective: balancing exploration of different prompt variations with exploiting the most promising edits in order to improve the LLM's ability to follow instructions across a range of tasks. In the next sections, we introduce the prompt optimization background used in GenDLN.

2.2 The Stacked LLM

Chaining, stacking, and joining different LLMs has been increasingly explored (Lu et al., 2024; Villarreal-Haro et al., 2024; Ferreira et al., 2024) and shown to perform well across domains for various use cases. The stacked LLM, where outputs from one LLM serve as inputs for another, has proven useful for decomposing complex tasks. One LLM processes raw input, generating intermediate representations or insights; another interprets these representations to complete tasks (classification, reasoning, decision-making). This decomposition boosts accuracy and interpretability (Zhang et al., 2021b) and enhances performance through specialization. Since LLMs excel at when narrowly prompted, this division of labor reduces individual LLM loads and improves result quality (Krishnamurthy et al., 2023). It also allows greater flexibility and modularity in solution design (Khot et al., 2023) while enhancing interpretability, as intermediate outputs clarify reasoning steps (Proca et al., 2024), crucial in fields where black-box decision-making is unsuitable, such as law. Lastly, this stacked paradigm mirrors human inference ("First,

analyze and interpret. Second, draw conclusions and decide" (Correa et al., 2023)). Regardless of the optimization method, stacked LLM architectures offer a clear advantage.

Sordoni et al. (2023) introduced DLNs as a prompt optimization technique leveraging chained LLM calls. Like other prompt optimization methods, the goal is to use frozen-weight LLMs for inference while refining input prompts for better results. They present two models: DLN-1 (single-layer) and DLN-2 (two-layer), treating LLMs as stochastic language layers with learnable natural language prompts as parameters. In DLN-2, the first layer's output is considered a latent variable requiring inference, while prompts are learned as parameters of the generative distribution. It employs variational inference for joint prompt optimization in the stacked LLM structure. The advantage of the stacked LLM in DLN is the ability to perform multi-step reasoning through the chaining of prompts and outputs. Similar to the stacked DLN-2 framework, our approach jointly optimizes a prompt pair (p_1, p_2) for classification, where the scoring function depends on classification metrics. We use the term "DLN" to refer to a two-layer deep neural network (DLN-2). Fig. 1 illustrates GenDLN's prompt pair evaluation.

Unlike DLN, which employs variational inference, we use an LLM-assisted GA for optimization. While LLMs exhibit reasoning-like behavior, research on their stability is mixed, showing high randomness and incoherence (Ma et al., 2024). To mitigate this, we rely on a heuristic optimization strategy (GA) with a reasoning-based evaluation step (DLN) to balance these issues while exploring a larger search space.

2.3 Genetic Algorithms

Genetic Algorithms (GAs) are a class of Evolutionary Algorithms (EAs), global stochastic optimization techniques inspired by Darwin's Theory of Evolution and Natural Selection. They iteratively evolve a "population" of candidate solutions toward the fittest, where the best individual represents the optimal solution (Holland, 1992). Evolutionary approaches excel where traditional methods like gradient descent fail-when the search space is vast, complex, or non-differentiable (Yu and Liu, 2024). Starting with an initial population, candidates are evaluated using a fitness function, with high-fitness individuals more likely to be selected for crossover. Crossover combines features from parents to generate offspring, which serve as new solutions. To maintain diversity, mutations-random occasional changes-are introduced. Repeating this cycle over multiple generations steadily refines solutions, making EAs effective for black-box optimization with minimal system knowledge.

Using GAs for prompt optimization is not new; GAs are proven metaheuristic prompt optimization methods (Pan et al., 2024), with few-shot genetic prompt search surpassing manual tuning (Xu et al., 2022) and evolutionary principles successfully applied to tasks like game comment toxicity classification (Taveekitworachai et al., 2024), pre-prompt optimization for mathematical reasoning (Videau et al., 2024), Japanese prompting (Tanaka et al., 2023), and emotional analysis (Resendiz and Klinger, 2024). EvoPrompt (Guo et al., 2024) employs LLMs for evolutionary operations like crossover and mutation while EAs guide optimization. The framework implements only one type of selection, crossover, and mutation, all executed by LLMs based on generic instructions, using both manual and LLM-generated initial populations. Our approach, GenDLN (Fig. 2), performs joint prompt-pair optimization instead of single prompt optimization, introduces multiple selection, crossover, and mutation strategies, and implements a richer parameter pool for the GA.

3 Methodology

GenDLN is a multi-objective, steady-state, hybrid genetic algorithm. More details on GenDLN's GA characterization can be found in Appendix A. In this section, we outline the 5 steps of the GA prompt optimization lifecyle in GenDLN (Fig. 2).

Initialization (3.1): An initial population of prompt pairs (p_1, p_2) is sampled from a predefined prompt bank, with optional augmentation.
Fitness Computation (3.2): Each individual is scored based on classification metrics by running it through the DLN.
Selection (3.3): Individuals are chosen based on fitness using various implemented strategies.
Genetic Operators (3.4):

Crossover (3.4.1): Combines two parents to generate semantically valid offspring.

Mutation (3.4.2): Introduces controlled variations to explore new solutions.
Replacement (3.5): The next generation is formed by selecting the top individuals, and early stop cri-

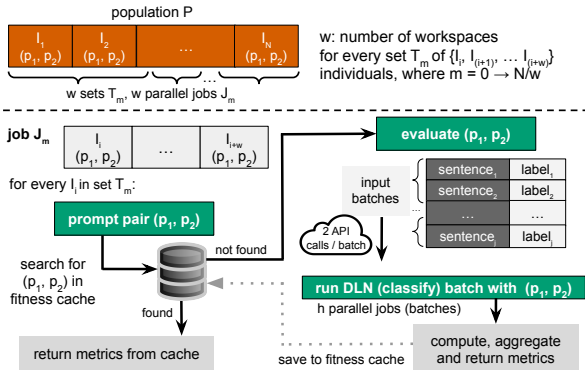

Figure 3: Efficiency strategies implemented as part of GenDLN. *Not shown:* workspace level rate-limiter that keeps the frequency of API calls below the platform-defined limit.

teria are defined.

3.1 Population Initialization

A population P is a set of individuals. Chromosome encoding refers to how an individual is represented. Each individual I is a prompt pair (p_1, p_2), where p_1 is the first-layer prompt for added context and p_2 is the second-layer prompt for classification.

For a population of size N, the initial population consists of N pairs (p_1, p_2) sampled from a predefined prompt bank, where example prompts are manually added. If the selected size exceeds the available prompts, a Population Initialization LLM optionally generates additional diverse prompts using the prompt bank as examples. Details are in Appendix B.

3.2 Fitness Function / Scoring

The fitness of a prompt pair is computed as a weighted sum of classification metrics, including accuracy and F1 scores, using a multi-objective scoring approach. Fitness is evaluated by running the individual through the DLN (Fig. 1) and comparing predicted labels $\hat{y}$ to ground truth y. Metric weights are configurable per GA run to reflect different classification goals. Invalid individuals (e.g., with empty prompts) are assigned a fitness of -1 to avoid propagation. Additional fitness implementation and system prompt details for output specification are in Appendix C and E.

Rate-Limiting Step: DLN Evaluation The bottleneck in GenDLN is the evaluation of individuals through the DLN, which requires two sequential API calls per data point. Since genetic algorithms require exploring large populations over many generations, and given the need to use larger models due to the limitations of using smaller ones for prompt optimization (Zhang et al., 2024b), this becomes both time- and cost-prohibitive. To address this, we implement fitness caching, rate limiting, and concurrency across two levels (Fig. 3). First, at the population level (above the dotted line), individuals are evaluated in parallel across w independent workspaces (each representing a compute node with its API key), creating w jobs J that evaluate subsets of the population. Second, within each job J (below the dotted line), the dataset is split into batches. Each batch is then classified using two API calls (one per DLN layer) instead of two calls per data point. Before evaluating a prompt pair (p_1, p_2), we check for its presence in a persistent fitness cache. If found, stored metrics are reused, avoiding an expensive DLN evaluation. These optimizations, for a dataset of size 100, increase throughput from ≈ 18 to ≈ 300 individuals/hour on an 8-core machine, a 16-fold improvement. More details on efficiency strategies and throughput computation are in Appendix F.

3.3 Selection

Selection can be considered the driving force of the GA; it determines which individuals from the current population will potentially undergo mutation and crossover (and conversely, which members of the current population are discarded), usually based on some function of the individual's fitness. The key is guiding the evolutionary process towards better solutions by preferentially selecting for higher fitness while maintaining population diversity, which is essential to avoid premature convergence. Selection pressure refers to the degree to which individuals with higher fitness are favored during the selection process and directly influences the balance of exploration and exploitation. Higher selection pressure increases the likelihood that fitter individuals will be chosen to pass on their genes, favoring exploitation. This may result in more rapid convergence but also premature convergence if diversity is lost too quickly. Conversely, lower selection pressure allows for a more diverse set of individuals to be selected, favoring exploration but potentially slowing down convergence (Haasdijk and Heinerman, 2018).

The choice of selection strategy is a parameter in GenDLN. We implemented most of the commonly used GA selection strategies, where each has distinct characteristics and influences the

algorithm's selection pressure and, thus, exploration/exploitation. Selection is the only genetic operator in GenDLN that is not fully or partially LLM-assisted. We implement **Random Selection** (lack of selection strategy used for comparison purposes), **Roulette Wheel Selection** (Holland, 1975), **Tournament Selection** (Miller et al., 1995), **Rank-Based Selection** (Baker, 2014), **Stochastic Universal Sampling (SUS)** (Baker et al., 1987), and **Steady-State Selection**. More details on each strategy's exploration-exploitation balance and implementation can be found in Appendix G.

Preprocessing and Elitism Before applying any selection method, an optional parameter elitism (k) is used to directly preserve the top k individuals with the highest fitness scores. This ensures that the best-performing solutions are not lost due to stochastic selection effects. For fitness score ties, indices are shuffled, and ties are broken randomly. When $k \neq 0$, the individuals are ranked by fitness, and the top k elites are selected for direct inclusion in the next generation. The remaining individuals undergo selection according to the chosen strategy.

3.4 Genetic Operators in the Textual Space

Since our chromosome is encoded as a tuple of two strings, applying typical crossover/mutation strategies presents challenges. Crossover and mutation are usually performed on bitstrings, numeric vectors, or structured representations of individuals, often following deterministic rules involving slicing, recombining, or editing genes based on strict positional encoding, which is straightforward for bitstring and numeric chromosomes. In the textual space, this is more complex. We discuss these considerations in Appendix H. Work on grammatically-based genetic programming (Whigham et al., 1995) for creating computer programs has shown the complexity of this task, even in code and query optimization (arguably easier to tokenize than natural language but still sufficiently character- and token-sensitive) (Whigham, 1995).

Research on genetic programming for natural language generation emphasizes the importance of maintaining semantic and syntactic coherence (Araujo, 2020). Thus, we leverage LLMs' ability to dynamically interpret, generate, and refine text as crossover and mutation operators, with prompts passed to an LLM. The response is parsed using regex-based JSON extraction to obtain children in crossover and the mutated prompt in mutation, with a fallback for invalid responses, detailed in Appendix D. Although we have iteratively tested various Mutation and Crossover prompts across different LLMs and included stable ones in GenDLN, these operations remain dependent on LLM responses, with results varying by model and temperature.

3.4.1 Crossover

We define a set of crossover strategies to allow different levels of exploration and exploitation. The LLM is crucial in ensuring that the offspring are grammatically valid, structurally coherent, and meaningful. We implement 5 strategies: **Single-Point**, **Two-Point**, **Semantic Blending**, **Phrase Swapping**, and **Token-Level** crossover. Details about their implementation and behavior can be found in Appendix I. Crossover is applied to individuals with a user-defined "crossover rate" C_r, the probability of an individual getting picked to participate in a crossover, and each crossover operation between 2 parents yields 2 children.

3.4.2 Mutation

Much like crossover, we define a set of different mutation strategies leveraging LLMs. The challenge with mutation is the necessity of "limiting" the edits to only a portion of the prompt, as mutation is typically used to introduce comparatively small changes to the chromosome with a user-defined mutation rate M_r. The goal of mutation is to introduce controlled diversity into the population while maintaining the semantic and syntactic coherence of the prompts. We implement 8 different mutation strategies (**Random**, **Swap**, **Scramble**, **Inversion**, **Deletion**, **Insertion**, **Semantic**, and **Syntactic**) with different editing modalities, whose details and prompts can be found in Appendix J. M_r sets the probability of a "gene" (in our case, a prompt is a gene) to undergo mutation. A "mutate elites" boolean parameter can be used to protect elites from mutation when elitism $k \neq 0$. Our choice of strategies and corresponding prompts for both crossover and mutation were made based on our experience and trial and error during the framework's development. It allows for easy editing/extension to include more crossover/mutation types and different prompts. Invalid responses are dealt with using the same retry-fallback mechanism.

3.5 Replacement and Termination

After mutation and crossover, the fitness of the resulting population (now containing approximately $(N + C_r * N)$ individuals) is calculated, and the top N individuals are the final population of the current generation, with the fittest one being declared the "best in generation."

A GA run is defined for a specific number of generations, but optional stopping criteria can be set, and the GA run will terminate when one of them is met. A "fitness goal" can end the run when the best individual achieves a fitness score equal to or greater than the goal, and a maximum number of stagnant generations S can be set to prematurely terminate the run if the best individual's fitness doesn't improve for S consecutive generations. Otherwise, the GA runs for the predetermined number of generations.

3.6 Logging and Post-Processing

GenDLN features a modular, detailed log structure that allows full retracing of any run. It logs abstractions like best/worst individuals per generation, average metrics, and genetic operator details, alongside full and extracted LLM responses. System details and runtime are also recorded. The output and logging structure is detailed in Appendix K. While implemented in Python, we provide R scripts for post-analysis and extensive GA lifecycle plotting. GenDLN is open source and easily extensible. Additional plots and reproducibility notes are in Appendix L and N.

The following sections describe experiments for binary and multi-label ToS classification on the CLAUDETTE dataset, and binary paraphrase detection on MRPC.

4 Datasets

4.1 CLAUDETTE

The CLAUDETTE dataset (Lippi et al., 2019) focuses on Terms of Service agreements from major online platforms, identifying potentially unfair clauses. It includes 50 contracts from providers like Dropbox, Spotify, Facebook, and Amazon, totaling 12,011 sentences, with 1,032 labeled as potentially unfair. Each document is annotated for two classification tasks: binary classification (fair vs. unfair) and multi-label classification, where unfair sentences receive one or more unfairness categories. These include Arbitration, Unilateral change, Content Removal, Jurisdiction, Choice of Law, Limitation of Liability, Unilateral termination, and Contract binding upon usage. Experts manually labeled sentences based on EU consumer law guidelines and court rulings. The dataset is imbalanced across both tasks. For our experiments, we split the data into train, test, and validation sets. LegalBERT and SVM baselines use the full training set, while prompt optimization baselines (OPRO and GrIPS) and our method use a balanced subset of 100 samples per task. A 1000-sample test set is used for evaluation.

4.2 Microsoft Research Paraphrase Corpus

The Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005) is a standard benchmark for sentence-level semantic equivalence. It contains 5,801 sentence pairs from news sources, labeled for binary paraphrase detection. We chose MRPC to evaluate GenDLN on a more general, smaller dataset that may not suit fine-tuning or traditional, non-prompt optimization methods. Despite its popularity, MRPC includes formatting artifacts that complicate its use in output-constrained LLM pipelines. We therefore created an *LLM-safe* version via two key preprocessing steps:

Quote sterilization: All quote characters (e.g., smart, curly, raw double quotes) were replaced with a Unicode-safe symbol to prevent JSON serialization errors. Mismatched or dangling quotes were manually corrected.

Trigger filtering: We removed examples containing high-risk commercial LLM trigger terms.

Our LLM-safe version (See Appendix P for details) preserves task structure and label distribution while ensuring compatibility with LLM-based classification. For experiments, we used 100 balanced training samples and 1000 stratified test samples.

5 Baselines

We compare our approach to both state-of-the-art and classical prompt optimization methods. Optimization by PROmpting (OPRO) (Yang et al., 2024) iteratively refines prompt instructions using an LLM. It uses a meta-prompt containing a problem description, top-performing instructions, and task examples to guide the LLM in generating and evaluating new prompts. Since Legal-BERT performs well in legal NLP classification (Chalkidis et al., 2020), we fine-tuned it on the full training set for all tasks. Our SVM baseline uses TF-IDF vectorization and is trained separately for each task

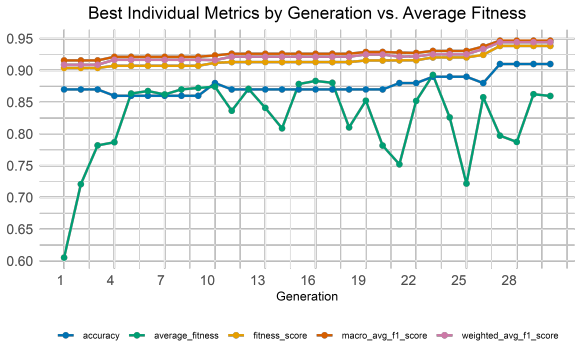

Figure 4: Metrics (best individual) and average fitness (population) for best CLAUDETTE multi-label run in Table 1.

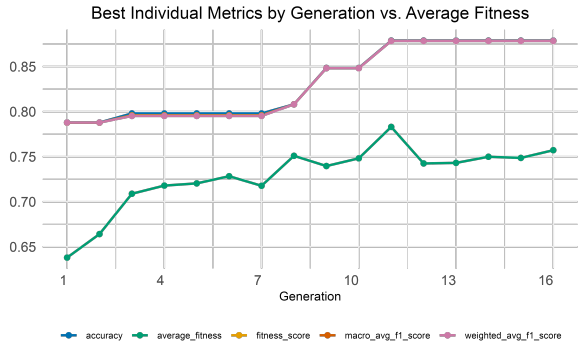

Figure 5: Metrics (best individual) and average fitness (population) for best CLAUDETTE binary run in Table 1. Individual metric lines overlap in the binary case.

on the full training set. The other baselines use the same data splits as our approach. The most comparable method to ours is GrIPS (Gradient-free, Edit-based Instruction Search) (Prasad et al., 2023), which edits prompts via deletion, addition, and word swapping, as well as paraphrasing using another LLM. Unlike our approach, it uses simple edit operations and selects top prompts deterministically, without stochastic operators like mutation or crossover.

6 Results and Discussion

We ran over 110 GenDLN executions on CLAUDETTE with various parameter sets across both tasks (binary and multi-label), and around 35 on MRPC. All runs draw from the same set of 10 binary and 10 multi-label manual prompts for CLAUDETTE, and 25 for MRPC, shown in Appendix B, Tables 3–7.

Table 1 lists the runs yielding the best-performing prompts across the different parameter sets we tried, selected based on Macro F1 performance on the test set. The full prompts for the runs are in Appendix M, Tables 11, 12 and 13. Common parameters for all reported runs: $k = 1$; no elite mutation; fitness function: $0.2 * (\text{accuracy}) + 0.4 * (\text{macro avg. F1}) + 0.4 * (\text{weighted avg. F1})$. We use Mistral Large ("mistral-large-2411", 123B parameters), with LLM temperatures for initialization, crossover, and mutation set to 0.7.

Fig. 4 shows the best non-stagnating multi-label CLAUDETTE run (Table 1). Interestingly, it used an insertion mutation strategy, leading to longer prompts, suggesting insertion is exploratory-supported by the diversity plot 7 in Appendix N, which shows a consistently diverse population after the first few generations. While shorter prompts often yield better results (Brown et al., 2020), this run did not early-stop, and could improve with more generations.

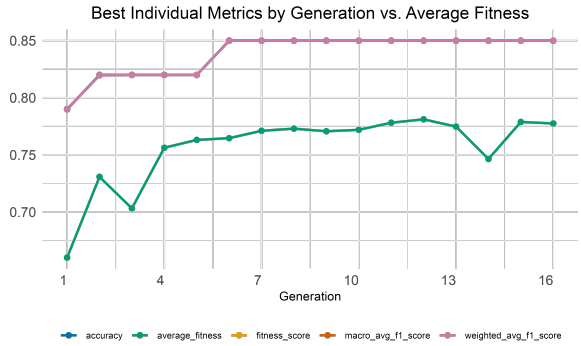

Figure 6: Metrics (best individual) and average fitness (population) for best MRPC binary run in Table 1.

Fig. 5 presents metrics for the best binary CLAUDETTE run. Like the multi-label case, we observe stable convergence and fitness improvements across generations. Table 1 lists the best binary run parameters. Unlike multi-label runs, where high-performing prompts were longer, binary runs maintained a more stable prompt length, suggesting structural modifications were more effective than exploratory insertions.

Fig. 6 shows the best MRPC run. MRPC runs resulted in an improvement in accuracy of 6 percentage points on average, with the range of improvement between 3–8 percentage points. Overall, GenDLN consistently improves initial prompts across reasonable parameter settings and remains stable over diverse configurations, and this consistency holds across both datasets. Appendix M includes additional selected runs, parameters, best prompts, and results. Appendix N contains further plots on metrics, convergence, diversity, and

	Task	Fitness	Performance (Test)			GA Parameters							Early Stop
			Acc.	Macro F1	W. F1	Sel.	Cross.	C_r	Mut.	M_r	Pop.	Gen.	
CLAUDETTE	**Binary**	0.879	0.79	0.652	0.826	Rank	Sem. Blend	0.8	Semantic	0.2	10	16	Yes
	Multi	0.938	0.825	0.862	0.856	Rank	Phrase Swap	0.85	Insertion	0.3	30	30	No
MRPC	**Binary**	0.849	0.813	0.796	0.816	Steady-State	Single Point	0.85	Semantic	0.20	30	16	Yes

Table 1: Best GenDLN runs across tasks and datasets. Dataset label is shown in first column. GA Parameters include Selection, Crossover and Mutation types, Population and Generation size, crossover rate C_r and mutation rate M_r. Early stop indicates that the run stopped early due to stagnation. *W. F1: Weighted F1 score.*

	CLAUDETTE						*MRPC*		
	Binary			Multi			Binary		
	Acc.	**Macro F1**	**W. F1**	**Acc.**	**Macro F1**	**W. F1**	**Acc.**	**Macro F1**	**W. F1**
GenDLN	0.79	0.65	0.83	0.83	0.86	0.86	0.81	0.80	0.82
OPRO	0.80	0.64	0.83	0.71	0.84	0.84	0.80	0.77	0.80
(Legal-)BERT*	0.94	0.85	0.94	0.97	0.91	0.91	0.80	0.78	0.80
SVM TF-IDF	0.93	0.79	0.93	0.77	0.86	0.86	0.70	0.59	0.66
GRIPS	0.82	0.45	0.85	0.94	0.82	0.82	0.79	0.76	0.79

Table 2: Test set performance comparison of baseline optimizers across datasets. *W. F1: Weighted F1 score.* *BERT was used for MRPC, Legal-BERT was used for CLAUDETTE.

similarity for our best runs (Tables 8, 9, and 10 in Appendix M). We also conduct an ablation study on the best runs, re-running them with "random selection" to isolate selection impact. As expected, ablation results show flatlined metrics, confirming that removing selection pressure collapses the GA into random search (details in Appendix O).

Our results align with expected GA behavior. All our runs had a maximum population size and generation size of 30, which is the bare-minimum, exploratory number for GA convergence. Rather than declaring "optimal" parameter sets for specific tasks, we demonstrate that GenDLN converges across diverse settings, tasks, and datasets. Additionally, Table 2 highlights GenDLN's strong performance against state-of-the-art baselines. In CLAUDETTE binary classification, GenDLN outperforms OPRO and GRIPS in macro F1-score (our prioritized metric due to dataset imbalance). Although LegalBERT and SVM reach the highest scores overall, they rely on full dataset fine-tuning and are not viable for prompt-based few-shot settings. In contrast, GenDLN consistently improves across reasonable parameter configurations using only 100 examples. This is especially notable for MRPC, which, unlike CLAUDETTE, does not require domain specificity. GenDLN achieves the overall best performance, and is in line with the highest few-shot F1 benchmark of 78.3 in the literature reported by Zhang et al. (2021a). For multi-label ToS classification, GenDLN also delivers strong macro and weighted F1 scores, outperforming OPRO and GRIPS in both and surpassing SVM in accuracy, demonstrating its ability to optimize prompt pairs effectively without requiring extensive model adaptation.

7 Conclusion

We introduce **GenDLN**, an efficient evolutionary algorithm-based framework for joint prompt optimization using a stacked LLM architecture. Our approach successfully refines populations of prompt pairs, achieving strong performance on ToS classification and paraphrase detection, in line with baselines such as OPRO and GrIPS on the CLAUDETTE dataset while remaining relatively cost and computationally efficient compared to traditional GA implementations. Through its efficiency strategies at several levels, GenDLN leverages commercial API free tiers to optimize prompt pairs at no cost. This enables resource-limited teams to use commercial LLMs for EA-based prompt optimization, as applied to well-defined tasks. Our findings highlight the potential of evolutionary strategies as a scalable alternative to traditional prompt engineering and fine-tuning, paving the way for more accessible and cost-effective LLM-driven classification methods.

8 Limitations

Given its reliance on classification based on extraction from an LLM response, the fitness function is subject to model biases and can be influenced by factors such as dataset quality, prompt structure, and stochastic behavior of LLMs. Consequently, fitness scores in this framework serve as an approximation of the true generalization ability of candidate solutions.

Another limitation is the necessary use of the cache, which may lead to bias, but is essential if using commercial LLMs.

Moreover, our framework is limited to tasks/problems where it is possible to encode a solution as a semi-structured, multi-dimensional individual that lends itself to crossover and mutation, and can be assessed by a fitness function. For reasoning/analysis tasks, especially those of a legal nature, the suitability of a solution may be less straightforward to encode and evaluate. Such tasks would require looking at a solution as a multi-step task (possibly using more DLN layers and a learned-heuristic approach), such as the work done by Chen et al. (2024).

Additionally, due to the modular logging structure, it is possible to run genetic operators individually and post-process their data. As such, it would be interesting to look at the use of LLMs as genetic operators more closely and examine how they compare to the established stochastic methods, and the bias and differences among different LLMs, temperatures, and parameters.

LLMs are known to sometimes suffer from uncontrolled bias (Bender et al., 2021; Li, 2023; Gallegos et al., 2024). In the context of GenDLN, this may lead to search space restriction due to trigger word sensitivity (Zhao et al., 2024), pretraining bias (Mina et al., 2025), and over-optimization bias (since LLMs are trained to minimize loss on text generation rather than maximize diversity). We have observed anecdotal evidence and instances of the above occuring, but this needs formal exploration.

Furthermore, we do not vary the LLMs and temperature parameters across our different runs. Ideally, instead of relying on the same LLM for all GA operations, different models for mutation, crossover, and evaluation can be used. This approach would introduce flexibility and attempt to reduce systemic bias. Since mutation requires diversity, and a model that introduces novelty, an open model would allow unfiltered, exploratory mutations. Crossover, on the other hand, requires consistency and meaning preservation, and an instruction-tuned LLM would be more suitable. For the DLN, a task-specific fine-tuned model would be more reliable for consistent classification.

Moreover, it is important to mention that unlike methods that optimize prompts based on error feedback, GenDLN does not "learn" the dataset in the traditional sense. Due to its reliance on competition and exploration-driven evolution, it shows adaptive improvement, and optimizes prompt pairs for classification with the specific target LLM model used for optimization. This is in line with expected EA behavior. For this reason, specific signals from the dataset will not necessarily make their way to the optimized prompts, and any learning is implicit and general, rather than dataset-specific.

Importantly, we include strong system prompts (based on trial and error) to supplement our optimized prompt pairs. Recent work has explored optimizing system prompts (Zhang et al., 2024a); a development of the idea would be to refine our chromosome encoding to include system prompts. This would make the chromosome carry more than a couple of genes, which is typically the case in GAs.

In addition, due to time constraints, we were not able to run all possible/plausible parameter set combinations. For that same reason, we did not re-run our GA with the same parameter sets, with different random seeds, as is usually done and rely on the high stability (consistent improvement across different parameters sets, tasks, and datasets) of our framework.

Lastly, we only tried one specific fitness score combination, and all our experiments had elitism = 1. We welcome any effort to extend the framework, explore more parameter combinations, and/or formalize parameter exploration for GenDLN through grid search or other techniques.

References

E. Alba and M. Tomassini. 2002. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462.

Lourdes Araujo. 2020. Genetic programming for natural language processing. *Genetic Programming and Evolvable Machines*, 21.

James E Baker et al. 1987. Reducing bias and inefficiency in the selection algorithm.

James Edward Baker. 2014. Adaptive selection methods for genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications*, pages 101–106. Psychology Press.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? .

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020. LEGAL-BERT: the muppets straight out of law school. *CoRR*, arXiv:2010.02559.

Yongchao Chen, Jacob Arkin, Yilun Hao, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Prompt optimization in multi-step tasks (promst): Integrating human feedback and heuristic-based sampling.

Carlos G. Correa, Mark K. Ho, Frederick Callaway, Nathaniel D. Daw, and Thomas L. Griffiths. 2023. Humans decompose tasks by trading off utility and computational cost.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Agoston E. Eiben and James E. Smith. 2015. *Introduction to Evolutionary Computing*. Springer.

Tarek El-Mihoub, Adrian Hopgood, Lars Nolle, and Alan Battersby. 2006. Hybrid genetic algorithms: A review.

Silvan Ferreira, Ivanovitch Silva, and Allan Martins. 2024. Organizing a society of language models: Structures and mechanisms for enhanced collective intelligence. *Preprint*, arXiv:2405.03825.

Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. 2024. Bias and fairness in large language models: A survey. *Computational Linguistics*, 50(3):1097–1179.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *Preprint*, arXiv:2309.08532.

Evert Haasdijk and Jacqueline Heinerman. 2018. Quantifying selection pressure.

Ali Hakimi Parizi, Yuyang Liu, Prudhvi Nokku, Sina Gholamian, and David Emerson. 2023. A comparative study of prompting strategies for legal text classification. In *Proceedings of the Natural Legal Language Processing Workshop 2023*, pages 258–265, Singapore. Association for Computational Linguistics.

Peter J. B. Hancock. 1994. An empirical comparison of selection methods in evolutionary algorithms.

John H Holland. 1975. Adaptation in natural and artificial systems. *University of Michigan Press google schola*, 2:29–41.

John H Holland. 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12).

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. *Preprint*, arXiv:2210.02406.

Yamuna Krishnamurthy, Chris Watkins, and Thomas Gaertner. 2023. Improving expert specialization in mixture of experts. *Preprint*, arXiv:2302.14703.

Jinqi Lai, Wensheng Gan, Jiayang Wu, Zhenlian Qi, and Philip S Yu. 2024. Large language models in law: A survey.

Zihao Li. 2023. The dark side of chatgpt: Legal and ethical challenges from stochastic parrots and hallucination. *arXiv preprint arXiv:2304.14347*.

Marco Lippi, Przemysław Pałka, Giuseppe Contissa, Francesca Lagioia, Hans-Wolfgang Micklitz, Giovanni Sartor, and Paolo Torroni. 2019. Claudette: an automated detector of potentially unfair clauses in online terms of service. *Artificial Intelligence and Law*, 27:117–139.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing.

M. Loos and J. Luzak. 2021. Update the unfair contract terms directive for digital services. PE 676.006.

Jinliang Lu, Ziliang Pang, Min Xiao, Yaochen Zhu, Rui Xia, and Jiajun Zhang. 2024. Merge, ensemble, and cooperate! a survey on collaborative strategies in the era of large language models. *Preprint*, arXiv:2407.06089.

Ruotian Ma, Xiaolei Wang, Xin Zhou, Jian Li, Nan Du, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Are large language models good prompt optimizers? *Preprint*, arXiv:2402.02101.

Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212.

Mario Mina, Valle Ruiz-Fernández, Júlia Falcão, Luis Vasquez-Reina, and Aitor Gonzalez-Agirre. 2025. Cognitive biases, task complexity, and result intepretability in large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 1767–1784, Abu Dhabi, UAE. Association for Computational Linguistics.

Jonathan A. Obar and Anne Oeldorf-Hirsch. 2020. The biggest lie on the internet: ignoring the privacy policies and terms of service policies of social networking services. *Information, Communication & Society*, 23(1):128–147.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Rui Pan, Shuo Xing, Shizhe Diao, Wenhe Sun, Xiang Liu, Kashun Shum, Renjie Pi, Jipeng Zhang, and Tong Zhang. 2024. Plum: Prompt learning using metaheuristic. *Preprint*, arXiv:2311.08364.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. *Preprint*, arXiv:2203.07281.

Alexandra M. Proca, Fernando E. Rosas, Andrea I. Luppi, Daniel Bor, Matthew Crosby, and Pedro A. M. Mediano. 2024. Synergistic information supports modality integration and flexible learning in neural networks solving multiple tasks.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. *Preprint*, arXiv:2305.03495.

Yarik Menchaca Resendiz and Roman Klinger. 2024. Mopo: Multi-objective prompt optimization for affective text generation. *Preprint*, arXiv:2412.12948.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI EA '21, New York, NY, USA. Association for Computing Machinery.

Alessandro Sordoni, Xingdi Yuan, Marc-Alexandre Côté, Matheus Pereira, Adam Trischler, Ziang Xiao, Arian Hosseini, Friederike Niedtner, and Nicolas Le Roux. 2023. Joint prompt optimization of stacked llms using variational inference. *Preprint*, arXiv:2306.12509.

Hiroto Tanaka, Naoki Mori, and Makoto Okada. 2023. Genetic algorithm for prompt engineering with novel genetic operators.

Pittawat Taveekitworachai, Febri Abdullah, Mustafa Can Gursesli, Antonio Lanata, Andrea Guazzini, and Ruck Thawonmas. 2024. Prompt evolution through examples for large language models–a case study in game comment toxicity classification. In *2024 IEEE International Workshop on Metrology for Industry 4.0 IoT (MetroInd4.0 IoT)*, pages 22–27.

Mathurin Videau, Alessandro Leite, Marc Schoenauer, and Olivier Teytaud. 2024. Evolutionary pre-prompt optimization for mathematical reasoning. *Preprint*, arXiv:2412.04291.

Kapioma Villarreal-Haro, Fernando Sánchez-Vega, Alejandro Rosales-Pérez, and Adrián Pastor López-Monroy. 2024. Stacked reflective reasoning in large neural language models.

P.A. Whigham. 1995. A schema theorem for context-free grammars. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 1, pages 178–.

Peter A Whigham et al. 1995. Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, volume 16, pages 33–41. Citeseer.

Aled Williams and Yilun Cai. 2024. Insights into weighted sum sampling approaches for multi-criteria decision making problems. *Preprint*, arXiv:2410.03931.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. 2022. Gps: Genetic prompt search for efficient few-shot learning. *Preprint*, arXiv:2210.17041.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Johnathan Yerby and Ian Vaughn. 2022. Deliberately confusing language in terms of service and privacy policy agreements.

He Yu and Jing Liu. 2024. Deep insights into automated optimization with large language models and evolutionary algorithms. *Preprint*, arXiv:2410.20848.

Lechen Zhang, Tolga Ergen, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. 2024a. Sprig: Improving large language model performance by system prompt optimization. *arXiv preprint arXiv:2410.14826*.

Ningyu Zhang, Luoqiu Li, Xiang Chen, Shumin Deng, Zhen Bi, Chuanqi Tan, Fei Huang, and Huajun Chen. 2021a. Differentiable prompt makes pre-trained language models better few-shot learners. *arXiv preprint arXiv:2108.13161*.

Tuo Zhang, Jinyue Yuan, and Salman Avestimehr. 2024b. Revisiting opro: The limitations of small-scale llms as optimizers. *arXiv preprint arXiv:2405.10276*.

Yi Zhang, Sujay Kumar Jauhar, Julia Kiseleva, Ryen White, and Dan Roth. 2021b. Learning to decompose and organize complex tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2726–2735, Online. Association for Computational Linguistics.

Shuai Zhao, Meihuizi Jia, Zhongliang Guo, Leilei Gan, XIAOYU XU, Xiaobao Wu, Jie Fu, Feng Yichao, Fengjun Pan, and Anh Tuan Luu. 2024. A survey of recent backdoor attacks and defenses in large language models. *Transactions on Machine Learning Research*.

A GenDLN: GA Characteristics

GenDLN is a multi-objective, steady-state genetic algorithm (SSGA), whereby only a subset of the population is replaced in each generation, and parents evolve alongside their children (through rolling selection, crossover, and mutation) rather than generating an entirely new population. Also, elitism (keeping the best k solutions unchanged) is implemented as an optional parameter, ensuring that the best individual(s) survive to the next generation. Due to employing LLMs in the population initialization, and the mutation and crossover genetic operators, the framework can also be described as a hybrid genetic algorithm (HGA), where domain-specific methods are integrated into the evolutionary process (El-Mihoub et al., 2006). In our domain, textual prompt optimization, GenDLN uses LLM inference to indirectly optimize the initial population, or yield a "good" mutation or crossover product, as opposed to deterministic bit-wise or function-aided manipulations used in classical GAs. Furthermore, in the fitness evaluation, employing the deep-language network (DLN) to determine the suitability of the solution (prompt pair) also makes use of LLM inference and classification-based fitness to guide the optimization process instead of using a deterministic, mathematical function. Our framework is also a multi-objective GA since we use weighted summing of multiple objectives into a single scalar fitness score (Williams and Cai, 2024).

B Population Initialization

This section provides an overview of the population initialization process for the GA, incorporating structured prompt generation and augmentation techniques.

Overview The population is initialized using predefined sets of prompts, which serve as the basis for generating diverse individuals. These prompts are loaded and paired to create an initial pool of candidates. These "prompt banks" as used in our experiments are shown in Tables 3 and 4.

Handling Population Size If the predefined set of individuals is smaller than the required population size, additional individuals are generated through augmentation. This ensures a sufficient and varied population.

Augmentation Process When augmentation is enabled, additional prompts are created by an LLM based on the existing prompt bank. The process ensures that newly created prompts maintain coherence and contribute to the diversity of the population.

Prompt Generation Details The augmentation process is guided by a structured system role and user input specification. The following details outline the LLM prompt construction.

System Role

```
You are an expert prompt generator. Based on
a given task description and examples, your
goal is to generate a specified number of new
prompt pairs.
Each prompt pair consists of two prompts:

Prompt 1: An initial instruction to an LLM,
to which the LLM would provide a response.
Prompt 2: A follow-up instruction, which
will be fed to another LLM along with the
prior response. This prompt should relate to
the expected response from the first LLM.

The new prompt pairs must be unique but
adhere to the task description.
Each prompt pair must follow this JSON
structure:
{
  "prompt_1": "The first instruction for an
   LLM, which it will respond to.",
  "prompt_2": "The second instruction for a
   different LLM, which will be given the
   response to prompt_1 as input and should
   be related to the expected response."
}
```

User Input

Prompt 1	Prompt 2
Summarize the following text.	Based on the description above, is the following fair or unfair?
Analyze the following sentence for any potential ambiguities or unclear language.	Based on the analysis above, would a reasonable person find this sentence to be fair or unfair?
Explain the meaning and implications of the following sentence in a legal context.	Considering the explanation above, is this sentence likely to be considered fair or unfair by a court of law?
Identify any potential biases or assumptions present in the following sentence.	Based on the identified biases and assumptions, do you think this sentence is fair or unfair?
Provide a paraphrased version of the following sentence to ensure clarity and understanding.	Considering the paraphrased version above, is this sentence fair or unfair in its original form?
Discuss the potential consequences of the following sentence in a legal or regulatory context.	Based on the discussion above, do you think this sentence is fair or unfair in its current formulation?
Evaluate the following sentence for compliance with relevant laws or regulations.	Considering the evaluation above, is this sentence fair or unfair in terms of its compliance with applicable laws?
Interpret the following sentence in the context of a specific industry or sector.	Based on the interpretation above, is this sentence fair or unfair in its application to this industry or sector?
Highlight any potential areas of concern or controversy surrounding the following sentence.	Considering the highlighted areas of concern above, do you think this sentence is fair or unfair in its current form?
Consider the following sentence in light of relevant case law or precedents.	Based on the consideration of case law above, is this sentence fair or unfair in terms of its alignment with established legal principles?

Table 3: CLAUDETTE - Manual binary prompt bank used to initialize every GenDLN binary run.

```
The task involves assessing the fairness or
unfairness of a given sentence using prompt
pairs.
The sentence will be taken from the Terms &
Conditions or similar documents of
businesses. In this context, fairness or
unfairness is meant in the strictly legal
sense.
This means determining whether the sentence,
in isolation, could be deemed unfair by a
lawyer or court of law.
We do not know in advance the nature of the
sentence or the area of law it relates to.

For this reason, prompt_1 must be general
and not tied to any specific scenario,
law, or  jurisdiction.
The sentence will be provided alongside
prompt_1 to an LLM. Prompt_1 will ask for
some insight, analysis, or interpretation
of the given sentence.
The response from the first LLM, along with
prompt_2, will then be passed to a second
LLM.

Prompt_2 acts as the final "classifier" and
must induce a fair/unfair classification
by the second LLM based on the response
to prompt_1.

Here are some examples of prompt pairs for
the mentioned task:

user_input += "- Prompt 1: " +
individual["prompt_1"] + "\n  Prompt 2: "
+ individual["prompt_2"] + "\n"

user_input += "\nGenerate " +
```

```
str(total_needed)
+ " additional pairs of prompts."

user_input += "Ensure all new pairs
are distinct from the examples."
```

Finalization Once the population reaches the desired size, unique identifiers are assigned to each individual. Logging mechanisms help track the composition of the population, distinguishing between original and augmented individuals.

This implementation supports prompt-based population initialization while maintaining flexibility through structured augmentation and validation mechanisms.

C Fitness Function

The fitness of a prompt pair is a weighted sum of classification metrics using a multi-objective weighted sum approach.

To compute fitness, the individual is evaluated through the DLN (Fig. 1). The classification results $\hat{y}$ are compared to real labels y, and raw metrics (accuracy, class precision, recall, F1-score, and aggregate metrics like macro- and weighted-average precision, recall, and F1-score) are output by the DLN. Metric weights in the fitness function are configurable per GA run, allowing adaptation to different classification goals, such as prioritizing class-balanced performance by emphasizing macro and weighted metrics or optimizing for specific classes. The sum of metric weights must equal 1, and the resulting fitness score lies in the $[0, 1]$ range. Invalid individuals (where at least one prompt is empty) are assigned a fitness score of -1 to prevent their propagation, as per the fallback mechanism outlined in the next section.

D Fallback Mechanism for Invalid LLM responses

In GenDLN, LLMs are employed for mutation, crossover and population initialization. The LLM is instructed to generate responses in a valid JSON format, which is necessary for the extraction of prompts and subsequent processing and evaluation of the individuals. However, there are several reasons why the LLM might fail to produce a valid JSON response, beyond ambiguity in prompt instructions (Liu et al., 2023; Reynolds and McDonell, 2021), which is not the case in GenDLN:

1. **Model Limitations and Hallucinations**: LLMs are known to potentially "hallucinate" or generate outputs that deviate from the expected format, especially when the task involves complex constraints or novel combinations of concepts (Ji et al., 2023). JSON generation requires strict adherence to syntax rules, and any deviation (e.g., missing brackets, incorrect key-value pairs) results in an invalid response.

2. **Token Limitations and Truncation**: LLMs have a finite context window, and if the generated response exceeds this limit, it may be truncated. Truncation can lead to incomplete JSON structures, rendering the output invalid. This issue is exacerbated when the response includes nested or lengthy JSON objects (OpenAI, 2024).

3. **Stochastic Nature of LLMs**: LLMs are probabilistic models, and their outputs can vary significantly even with identical inputs due to temperature settings and sampling strategies. This stochastic behavior increases the likelihood of generating invalid JSON, especially if the temperature parameter is set too high, encouraging creativity at the expense of consistency (Brown et al., 2020). Although our LLM temperature is 0.7 for all experiments, this does not discount the stochastic effects.

4. **Crossing Over Identical Prompts**: Some selection strategies naturally lead to the presence of the same individual more than once in the population. Moreover, it is possible to have individuals with one identical prompt through the natural trajectory of evolution. Since individuals are paired up for crossover randomly, the Crossover LLM might be prompted to crossover two "identical" sentences. In most of these cases, the LLM outputs an invalid response. This was a problem for all LLMs we tried, including Llama-3.1-8B, Llama-70B, Ministral 8B, and even Mistral Large. We chose not to mitigate this by explicitly including this case and how to deal with it in the prompt to the operator and instead detect this with the fallback mechanism.

D.1 Fallback Mechanism

To mitigate these issues, we implemented a fallback mechanism that retries the operation up to a specified limit (3 in our experiments). If all retries fail, an empty string is returned, which is detected during fitness calculation. The assignment of a fitness score of -1 to such individuals ensures that they are not propagated further in the evolutionary process, maintaining the integrity of the population. This approach aligns with established practices in evolutionary computation, where invalid or malformed individuals are penalized to prevent their influence on future generations (Eiben and Smith, 2015) and limit their downstream propagation. We observe that invalid responses occur quite frequently, and can be visualizes as "X" on the y-axis in the convergence plots 31, 32, 33, 35, 36, 37.

E System Prompts

E.0.1 System Prompts

GenDLN's DLN implementation includes system prompts in scoring. These specify the input/output format (e.g., JSON), define the task, and may include few-shot examples.

Our approach utilizes four distinct system prompts, corresponding to the two-layer binary and multi-label classification approaches. Each prompt defines the input format, specifies the expected output structure, and ensures consistency in model responses.

All prompts follow a common structure:

- The embedded prompt generated by our GA.
- A description of the input format, including identifiers and sentence text.
- A specification of the expected output format, ensuring valid JSON at the second layer.
- Example inputs and outputs to showcase the expected input and output format.

Few-Shot Examples Each system prompt includes six few-shot examples to guide the model's responses. For binary classification, we randomly select three fair and three unfair sentences from the training set, ensuring they are distinct from those used in the optimization task. For multi-label classification, we select six sentences, each representing a unique class. Additionally, for Layer 2 prompts, the examples include the feature-enriched output from Layer 1 to provide a more contextualized input.

This approach ensures a balanced representation of labels while maintaining consistency across both classification tasks.

We present the full system prompts in the following sections.

E.1 Binary Classification

E.1.1 System Prompt Layer 1

```
<Prompt_01_Placeholder>

Input Data
The input data is a dictionary containing
 sentences from the CLAUDETTE dataset,
 where each entry has:
Key: An identifier
(e.g., "sentence_1", "sentence_2")
Value: The sentence text

Example Input
{
    "sentence_1": "This is the text
             representing sentence 1.",
    "sentence_2": "This is the text
              representing sentence 2."
}
```

E.1.2 System Prompt Layer 2

```
<Prompt_02_Placeholder>

Input Data
The input data is composed of two parts.
 The first part ("previous_outputs:")
 contains a feature-enriched version
 of the user input that has already been
 processed by a different LLM and
 system prompt. The second part
 ("sentences_to_classify:") is
 a dictionary containing sentences
 to classify, where each entry has:

Key: An identifier
(e.g., "sentence_1", "sentence_2")
Value: The sentence text

Example Input
"previous_outputs": "Feature enriched
                     version of the
                sentences to classify"
"sentences_to_classify":
{
```

```
   "sentence_1": "This is sentence 1.",
    "sentence_2": "This is sentence 2."
}

Output Requirements
For each sentence, add:
"classification": "fair" or "unfair".
"rationale": Explanation highlighting
             influential words.

Example Output
{
    "sentence_1": {
        "text": "This is sentence 1.",
        "classification": "fair",
     "rationale": "Explain the decision."
    },
    "sentence_2": {
        "text": "This is sentence 2.",
        "classification": "unfair",
     "rationale": "Explain the decision."
    }
}

Ensure JSON format is valid!
```

E.2 Multi-Label Classification

E.2.1 System Prompt Layer 1

```
<Prompt_01_Placeholder>

CLAUDETTE Classes:
- PINC (Pins and Cookies)
- USE (Usage Restrictions)
- CR (Content Removal)
- TER (Termination)
- LTD (Liability Limitation)
- A (Arbitration)
- LAW (Applicable Law)
- J (Jurisdiction)
- CH (Changes)

Input Data:
A dictionary of “unfair” sentences:
- Key: Sentence ID (e.g., "sentence_1").
- Value: The sentence text.

Example Input:
{
   "sentence_1": "We may terminate your
                  account at any time.",
    "sentence_2": "By using Pinterest,
                   you agree to our
                   policies."
}
```

E.2.2 System Prompt Layer 2

```
<Prompt_02_Placeholder>

CLAUDETTE Classes:
- PINC, USE, CR, TER, LTD, A, LAW, J, CH

Input Data:
First Part: "previous_outputs"
     - Feature-enriched sentences.
Second Part: "sentences_to_classify"
      - Dictionary of sentences.

Example Input:
"previous_outputs": "Feature enriched
                     version"
"sentences_to_classify":
{
    "sentence_1": "We may terminate
          your Account at any time.",
    "sentence_2": "By using Pinterest,
          you agree to our policies."
}

Example Output:
{
    "sentence_1": {
        "text": "We may terminate
              your account.",
        "classification": ["TER"]
    },
    "sentence_2": {
        "text": "By using Pinterest,
              you agree.",
        "classification":
              ["PINC", "USE"]
    }
}

Each sentence is classified
into one or more labels.
Ensure JSON validity.
```

F Efficiency Strategies

F.1 Motivation and Setup

Since we use commercial LLM APIs and GAs require exploring a vast search space to converge, running our framework is both cost- and time-

intensive, especially for fitness evaluation. Evaluating a prompt pair through the DLN requires two API calls per data point. For large datasets and populations (essential for exploration), running the framework for enough generations becomes too expensive, not to mention the need to test various parameter sets and the significant trial-and-error phase inherent to evolutionary optimization. To mitigate this, we implemented efficiency strategies at different framework stages. We apply metric caching, request rate limiters, and concurrency at two DLN levels (Fig. 3).

F.1.1 Metric Caching

As mentioned, running an individual through the DLN yields a set of classification metrics. In GenDLN, these raw metrics are cached for every prompt pair to avoid rerunning the evaluation of the same prompt pair within the same run; we also extend it to avoid rerunning the evaluation of the same prompt pair for the same LLM-dataset-task combination. The cost savings and speed-up provided by caching comes at the risk of introducing some bias (LLM-classification is inherently unstable, and the same prompt can lead to different responses from the same LLM). However, this is primarily used to explore parameter sets, and for suitable, stable parameter definitions, the GA should eventually be rerun three times to discount noise.

F.1.2 Parallelization

Significant work has been done on parallelizing the execution of GAs (Alba and Tomassini, 2002). For GAs in general, evaluation of an individual is independent, and for GenDLN (DLN classification using prompts (p_1, p_2)), this allows population evaluation to be parallelized. To accelerate the prompt optimization process, our framework employs a two-layer parallelization approach, addressing both the evaluation of individual prompt pairs and the internal processing of data batches for each individual.

Inter-Individual Parallelization In Fig. 3, the top section (above the dashed line) shows population-level parallelization, our first concurrency layer.

A workspace W is a compute node with an independent API token handling requests. For a w-core machine, w sets of individuals from population P run in parallel across w workspaces, creating w jobs J, each evaluating up to N/w individuals. Rather than processing individuals sequentially, our framework concurrently evaluates several prompt pairs. This strategy exploits multicore architectures to significantly reduce the overall optimization time. By partitioning the population across multiple execution threads or processes, each prompt pair can be evaluated independently. Importantly, each individual maintains its own isolated "workspace," meaning that the computational resources and rate-limiting mechanisms are managed on a per-individual basis.

Intra-Individual Concurrency The bottom section (Fig. 3) details job J. Within the evaluation of a single prompt pair (job J), further efficiency is gained by concurrently processing the training dataset. We first partition the dataset into multiple batches, then evaluate the prompt pair on these batches concurrently, using 2 API calls (one per DLN layer/prompt) per batch rather than 2 per sentence.

This fine-grained parallelism allows us to aggregate evaluation metrics faster, as each batch is processed in parallel rather than sequentially. The results across individuals and batches are aggregated to determine (p_1, p_2)'s overall performance, with metrics stored in the cache for future use.

A notable constraint in our setup is the use of an external API that enforces a strict rate limit of one request per second (RPS). To adhere to this limit while still maintaining high throughput, we integrate a rate limiter into our concurrency model. For each prompt pair, the batch-level evaluations are regulated such that API calls are spaced appropriately. Since each individual has its own "workspace," the rate limiting is applied independently per prompt pair. This design ensures that the API is not overwhelmed by simultaneous requests across the entire population while still exploiting concurrency within each evaluation task.

Overall, the combination of inter-individual parallelization and intra-individual concurrency leads to a significant speedup in our prompt optimization process, allowing us to efficiently explore the search space while managing the operational constraints imposed by the external API.

F.1.3 Individual Evaluation Throughput

To quantify the efficiency of our genetic algorithm runs, we define the *individual evaluation throughput* as the number of individuals evaluated per unit of time. Given a genetic algorithm run with G generations, a population size of N, a crossover rate of

C_r, and a total runtime of T hours, the number of individuals evaluated per generation is computed as:

$$N(1 + C_r) \tag{1}$$

Thus, the total number of individual evaluations across all generations is:

$$G \cdot N(1 + C_r) \tag{2}$$

To determine the throughput in terms of individuals evaluated per hour, we divide the total evaluations by the runtime:

$$\text{Throughput} = \frac{G \cdot N(1 + C_r)}{T} \tag{3}$$

This metric allows us to compare different genetic algorithm configurations by normalizing their efficiency in terms of evaluations processed per hour, thereby accounting for variations in runtime across different experimental settings.

G Selection Strategies

G.1 Random Selection

Random selection is the absence of a selection strategy. It refers to selecting individuals uniformly at random, irrespective of fitness values. We implement it for use as a baseline for comparison purposes.

G.2 Roulette Wheel Selection

Also known as fitness proportionate selection, roulette wheel selection is one of the very first explored GA selection strategies (Holland, 1975). It simulates spinning a wheel where each individual occupies space proportional to its fitness, and selections are made probabilistically (by "spinning" a wheel and selecting the individual the "pointer" lands on). It ensures that individuals with higher fitness have a higher chance of selection, but any individual could potentially be selected. However, if relatively high-fitness individuals dominate early, this may lead to premature convergence. Also, when fitness values are very similar, low selection pressure may lead to stagnation (Hancock, 1994).

Tournament Selection First introduced by Miller et al. (1995), tournament selection is a simple and widely-used selection strategy. For a tournament size t, it randomly picks t individuals from the population, and selects the individual with highest fitness (the "tournament winner") for the next generation. For a population size N, N tournaments are held, with t participants each (if elitism $k \neq 0$, $N - k$ tournaments are held). Tournament selection aims to establish a balance between exploration and selection pressure, which can be tuned with tournament size t. Larger tournaments lead to stronger selection pressure and lower diversity (exploitation), while smaller tournament sizes favor exploration.

Rank-Based Selection Conceptually similar to roulette wheel, rank-based selection assigns individuals space on the wheel according to their rank rather than their fitness, where the total space on the wheel is equal to the sum of the ranks. Introduced by Baker (2014), to mitigate scaling issues where individuals in the population have fitness values that are either too extreme (high-fitness outliers would be selected too often in classical roulette), or too similar (if fitness values are too close together, each individual would have roughly the same chance of being selected in classical roulette). Rank selection ensures a linear selection probability distribution which prevents bias towards disproportionately high fitness individuals, while maintaining selection pressure.

Stochastic Universal Sampling (SUS) SUS was introduced by Baker et al. (1987) as an improvement over roulette wheel selection. In this variant, N evenly spaced pointers are assigned to the wheel, on which the individuals occupy space proportional to their fitness values, and N individuals are selected in one go when the wheel is "spun." It ensures a more diverse selection and reduces stochastic noise, but will still suffer from premature convergence in the presence of a high-fitness outlier (if an individual occupies a disproportionately large space on the wheel, several pointers will land on it).

Steady-State Selection Our framework is inherently an SSGA due to the way our replacement step (discussed in a futher section) operates, however, we also implement an explicit steady-state selection strategy for greater flexibility. Steady state selection requires elitism $k \neq 0$ or else it will behave like random selection. In this strategy, the top k fittest individuals are selected for the next generation, and $N - k$ are randomly selected from the remaining individuals to complete the population. Steady-state selection ensures that only a few individuals are replaced at a time in each genera-

tion. Always keeping many elites in the population may accelerate convergence at the risk of reducing diversity.

H Adapting Chromosomes to the Textual Space - Considerations

Although we have encoded the chromosome as a tuple, that does not mean the individual only has 2 genes (p_1 and p_2). The "suitability" of the solution depends on unstructured, hard-to-define components or "tokens" within the two text prompts, as well as hidden "genetic material" in the textual features of each prompt string. In natural language, different words, phrases, and clauses hold different weights in conveying meaning, unlike in structured encoding, where every component's contribution to the solution's suitability is defined. If classical strategies were to be applied (slicing the strings at arbitrary points, editing the characters at arbitrary indices), this would risk yielding too many syntactically invalid or semantically nonsensical prompts. Additionally, words and phrases are interdependent (much like real genes), and simple positional swapping and randomized editing may distort the meaning. In fact, textual meaning can completely collapse if crossover/mutation is badly applied, yielding individuals far inferior to their progenitors, which defeats the purpose. Determining where and how to split/edit text dynamically while ensuring coherence of results is an inherently non-deterministic process, contrary to the established concept of crossover and mutation in GAs.

I Crossover Strategies

We implemented the following strategies:

Single-Point Selects a single random point in each sentence and swaps the latter halves to form new sentences.

Two-Point Selects two random points in each sentence, swapping alternating segments to form new sentences.

Semantic Blending Blends the core meaning of both parents into two complementary sentences. Offspring are not simple recombinations but rather semantically fused versions of the inputs.

Phrase Swapping Identifies key phrases in each parent and swaps them while maintaining grammatical integrity.

Token-Level Swaps individual words or tokens between sentences.

I.1 Crossover System Prompt

"You are an expert linguist and copywriter, acting similar to how genetic crossover works, but in a textual context. Generate two complementary sentences as children of the provided parent sentences. Here complementary means that the two child sentences must have complementary parts of the parents, as in genetic crossover. Make sure the children sentences are wrapped in a JSON-object as follows:

```
{"child_1": "child sentence 1",
"child_2": "child sentence 2"}
```

The rest of your response can be plain text, but the new sentences must be in a JSON. Both sentences must be grammatically correct and reasonably meaningful."

I.2 Crossover Strategy Prompts

Single-Point "Combine the following two sentences by splitting each at a single random point. The first child should take the first half of the first sentence and the second half of the second sentence. The second child should take the first half of the second sentence and the second half of the first sentence. Ensure both sentences remain coherent and meaningful."

Two-Point "Combine the following two sentences by selecting two random points in each sentence. The first child should integrate the segments alternately, starting with the first part of the first sentence. The second child should integrate the remaining segments alternately. Ensure both sentences are coherent and meaningful."

Semantic Blending "Blend the following two sentences to create two complementary sentences. Each child should focus on combining the core meaning of both sentences in a unique way. Ensure that both sentences are coherent, meaningful, and distinct from one another."

Phrase Swapping "Swap one or more phrases between the following two sentences to create two new sentences. Each child should incorporate phrases from the other parent in a way that creates a coherent and meaningful result."

Token-Level "Swap individual words or tokens between the following two sentences to create two

new sentences. Each child should incorporate words from the other parent in a way that creates a coherent and meaningful result."

I.3 Crossover Examples

Below are some selected illustrative crossover examples.

Single-Point

```
Parent 1: "Summarize the following text."
Parent 2: "Explain the meaning
and implications of the
following sentence in a legal context."
Child 1: "Summarize the following text
in a legal context."
Child 2: "Explain the meaning and
implications of the following text."
```

Two-Point

```
Parent 1: "Summarize the following text."
Parent 2: "Explain the meaning and
implications of the following sentence
in a legal context."
Child 1: "Summarize the meaning and
implications of the following sentence in
a legal context"
Child 2: "Explain the following text in
a concise manner and its potential impact
on the law"
```

Semantic Blending

```
Parent 1: "Based on the description above,
is the following fair or unfair?"
Parent 2: : "Considering the explanation
above, is this sentence likely to be
considered fair or unfair by a court
of law?"
Child 1: "Considering the description
above, is the treatment likely to be
considered fair or unfair by a court
of law?"
Child 2: "Based on the explanation
above, is the sentence likely to be
considered fair or unfair in a court
of law?"
```

Phrase Swapping

```
Parent 1: "Summarize the following text."
Parent 2: : "Explain the meaning and
implications of the following sentence
in a legal context."
Child 1: "Explain the meaning and
implications of the following summary in
a legal context."
Child 2: "Summarize the following sentence
to understand its core message and
implications."
```

Token-Level

```
Parent 1: "Based on the description above,
is the following fair or unfair?"
Parent 2: "Considering the explanation
above, is this sentence likely to be
considered fair or unfair by a court
of law?"
Child 1: "Considering the description
above, is the following sentence likely
to be considered fair or unfair by a
court of law?"
Child 2: "Based on the explanation above,
is the following sentence likely to be
considered fair or unfair by a court of
law?"
```

J Mutation Strategies

The following is a summary of the introduced strategies and their intended result.

Random Changes a single word or phrase in the sentence to a synonym or a similar concept.

Swap Swaps existing words or phrases in the sentence to introduce minor structural variation.

Scramble Rearranges the order of words/phrases while maintaining the original meaning.

Inversion Reverses the order of words or phrases in part or all of the sentence.

Deletion Removes a word or phrase from the sentence to create a more concise variation.

Insertion Adds new words or phrases to provide additional context while preserving meaning.

Semantic Rephrases the sentence slightly while keeping the core meaning intact.

Syntactic Alters the sentence structure while preserving the meaning.

J.1 Mutation System Prompt

"You are an expert linguist and copywriter. Make sure the sentence you return is wrapped in a JSON-object as follows:

```
{"mutated_sentence": "new sentence
you generate based on the instruction"}.
```

The rest of your response can be plain text, but the new sentence must be in a JSON. The new sentence you suggest must be grammatically correct and reasonably semantically similar to the original."

J.2 Mutation Strategy Prompts

Random "Change only one single word or phrase in the sentence to a synonym or similar concept."

Swap "Swap two existing words or phrases in the sentence."

Scramble "Rearrange the existing words and/or phrases in the sentence with a minimal addition of new words."

Inversion "Invert the order of the existing words or phrases in all or part of the sentence."

Deletion "Delete a word or phrase in the sentence."

Insertion "Insert words or phrases in the sentence that could provide more context/clarity while keeping the same base meaning."

Semantic "Slightly rephrase the sentence."

Syntactic "Modify the sentence structure of the sentence while keeping the same base meaning."

J.3 Mutation Examples

Below are some selected illustrative mutation examples.

Semantic

```
Initial Prompt: "Produce a detailed output for each sentence, outlining the reasoning for its classification into the most likely category."
Mutated Prompt: "Generate a comprehensive output for each sentence, explaining the rationale for its categorization into the most probable group."
```

Insertion

```
Initial Prompt: "Interpret each sentence and provide a comprehensive rationale for its legal classification."
Mutated Prompt: "Carefully interpret each individual sentence within the context of the document and provide a comprehensive rationale for its specific legal classification."
```

Random

```
Initial Prompt: "Summarize the following text."
Mutated Prompt: "Condense the following text."
```

Swap

```
Initial Prompt: "Based on the description above, is the following fair or unfair?"
Mutated Prompt: : "Based on the description above, is the following unfair or fair?"
```

Deletion

```
Initial Prompt: "Based on the description above, is the following fair or unfair?"
Mutated Prompt: "Based on the description, is the following fair or unfair?"
```

Scramble

```
Initial Prompt: "Based on the description above, is the following fair or unfair?"
Mutated Prompt: "Is the following fair or unfair, based on the description above?"
```

K GenDLN Logging

Every sub-component of GenDLN (fitness calculation, selection, crossover, mutation, replacement, caching) has a dedicated logger and defined structure, and a GA Log (which is the output of the framework), is a structured log of these components. Below we provide the expected output and logger functionality and examples.

The logging system in the Genetic Algorithm (GA) serves as a comprehensive tracking and debugging framework, capturing detailed records of key evolutionary events at multiple levels. It ensures traceability of the entirety of the GA run. The logging structure is hierarchical, with nested loggers handling distinct operations, and a centralized GA logger aggregating all logs.

Hierarchical Structure of Logging The logging framework consists of specialized loggers:

- **GA Logger** – The central log for the entire evolutionary process, containing per-generation records of all key operations.
- **Population Initialization Logger** – Tracks how the initial population is created, including augmentation details.
- **Selection Logger** – Records selected individuals, strategy parameters, and elitism effects.

- **Crossover Logger** – Captures the details of crossover operations, including parent-offspring relationships.
- **Mutation Logger** – Stores information on how individuals are mutated, along with mutation types.
- **Fitness Logger** – Logs individual fitness scores and overall generation-level fitness statistics.
- **Fitness Cache Logger** – Tracks cache hits and misses.
- **Replacement Logger** – Logs how individuals are retained or replaced in the next generation.
- **Run-Specific Details** – Runtime, system specs, configs, and hyperparameters of the GA run are appended to the end of the log.

GA Logger: Centralized Evolution Tracking Each generation's log entry contains the following:

```
{
    "generations": [

        {"generation_id" : i,
        "initial_population": [...],
        "selection_data": [...],
      "population_after_selection": [...],
        "crossover_data": [...],
      "population_after_crossover": [...],
        "mutation_data": [...],
      "population_after_mutation": [...],
        "fitness_data": {...},
        "replacement_data": [...]
        },
         {...}, ...],
    "early_stopping":
        {"status": false, "reason": ""},
    "runtime": "3.25 minutes",
    "system_info": {...},
    "config": {...},
    "hyperparameters": {...},
    "ga_log_filename":
        {"ga_log_date-timestamp.log"}
}
```

This hierarchical logging system ensures that all operations are transparently recorded, aiding both debugging and performance analysis of the genetic algorithm.

L Reproducibility

We provide a set of R Scripts that allow the reproduction of our results, plots, and analyses. The scripts are structured to ensure transparency and ease of replication, and enforce a file path structure for inputs and outputs.

L.1 Environment Setup

All necessary dependencies are installed and loaded at the start of the execution. The required R libraries include `tidyverse`, `jsonlite`, `here`, `purrr`, `data.table`, `dplyr`, `ggplot2`, `tidyr`, `readr`, and `stringdist`. The script automatically installs missing dependencies.

L.2 Data and Directory Structure

The project assumes a structured directory for data storage and result output:

- **Root Directory**: Automatically set to the location of the script.
- **Log Directory**: Stores raw Genetic Algorithm (GA) log files (output of GenDLN).
- **Summary Directory**: Contains extracted metadata and performance summaries.
- **Test Directory**: Stores test results.
- **Output Directory**: Stores processed results and plots.
- **Plot Directory**: Contains visualization outputs.

All necessary directories are created if they do not exist.

L.3 Processing and Normalization

Log File Normalization GA log files are processed into structured formats. Key extracted elements include:

- Initial generation data (fitness scores, raw metrics, attributes).
- Subsequent generation data with performance metrics.
- Total number of completed generations.
- Metadata including runtime, system configuration, hyperparameters, and early stopping conditions.

Metadata Extraction Log files are further processed to extract structured information on:

- GA parameters (population size, mutation rate, selection strategy, fitness function).
- Run performance (best fitness scores, accuracy, raw evaluation metrics).
- Execution environment (system specifications, runtime details).

L.4 Batch Processing and Summary Generation

Aggregating Run Summaries A batch processing script collects metadata from all runs and produces a consolidated summary. The summary includes:

- Number of runs per batch.
- Associated test results.
- Log files used in the batch.

This process ensures that interrupted runs are accounted for and test data is linked correctly.

Appending Notes to Summaries Notes can be appended to individual summaries to document special conditions or anomalies in the runs.

L.5 Analysis and Visualization

GA Performance Report Each log file is processed to produce a detailed report that includes:

- Performance metrics across generations (fitness scores, accuracy, F1 scores...).
- Statistical summaries (mean, variance, min, max values of key metrics).
- Evolutionary trends of best and worst individuals.

Metric Extraction and Visualization Metrics such as fitness score, accuracy, and F1 scores are extracted for each generation and visualized to track GA progression.

GA Convergence Analysis The convergence of the GA is visualized by plotting best and worst fitness scores across generations.

Diversity and Similarity of Best Individuals The script computes diversity across generations, tracking:

- Unique individuals per generation.
- Similarity of best individuals across generations.
- Levenshtein and Jaccard similarity scores for best individuals.

Comprehensive Run Summary A final combined summary consolidates all extracted information, test results, and log metadata into a structured CSV file.

M Detailed Results

M.1 CLAUDETTE

The best prompts from the top 4 selected binary runs in Table 8 are shown in Table 11

As for multi-label, results are in Table 9, and prompts are in Table 12.

M.2 MRPC

The best prompts from the top 4 selected runs in Table 10 are shown in Table 13

N Detailed Plots

N.1 Metrics Over Generations

The metrics over generations plot tracks key performance metrics across generations, such as accuracy, fitness score, average fitness, and F1 scores. It is a multi-line plot where each line represents a metric and its trend over generations. The x-axis represents the generation number, while the y-axis represents the value of the metric. Different colors indicate different metrics.

Higher values generally indicate better performance. Fluctuations in fitness and accuracy reflect instability or exploration by the genetic algorithm (GA), while a converging trend suggests stabilization around optimal solutions. A steadily increasing or stable fitness score implies progress and convergence, whereas a volatile or fluctuating fitness score suggests ongoing evolution.

CLAUDETTE Plots for the top multi-label runs are on the left side of Fig. 7, 9 and 11, 13. For the binary runs, they are on the left of Fig. 15, 17 and 19, 21.

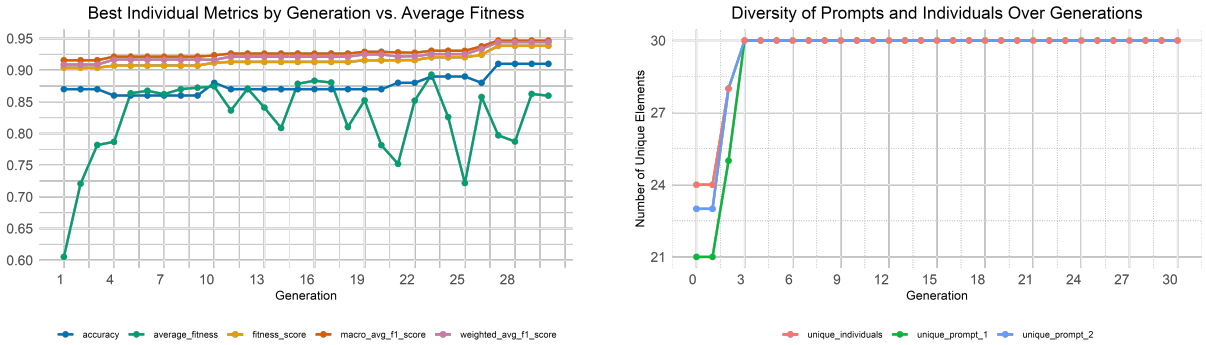

Figure 7: CLAUDETTE - **Left:** plot of metrics and average fitness for best run A in Table 9. **Right:** Diversity plotting for best multi-label run A in Table 9

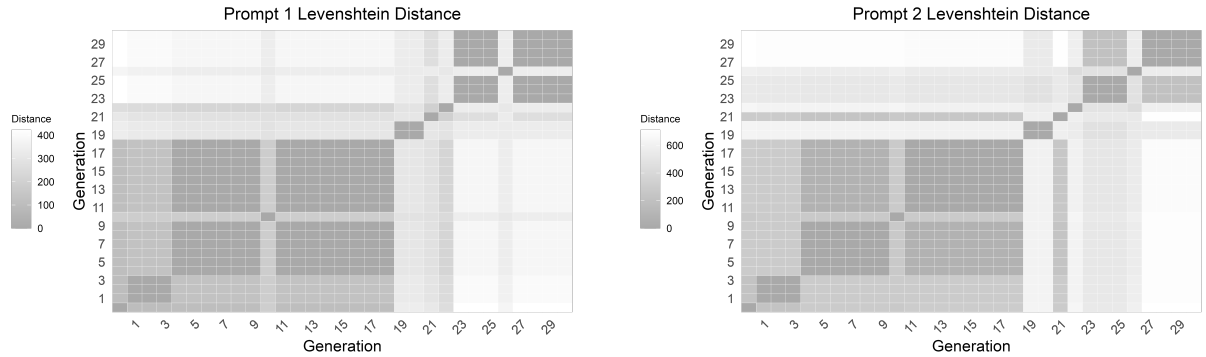

Figure 8: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run A in Table 9.

MRPC Plots for the top runs are on the left side of Fig. 23, 25 and 27, 29.

N.2 Convergence Plot

The convergence plot visualizes how the best and worst individuals change across generations, providing insight into GA optimization progress. This line plot features a dashed blue line representing the best fitness and a dotted red line representing the worst fitness. A shaded region between these lines indicates population fitness spread. The x-axis represents the generation number, and the y-axis represents the fitness score. The best fitness line tracks the top-performing individual in each generation, while the worst fitness line tracks the least-performing individual. A narrowing gap between the two lines indicates that the population is converging toward similar solutions. If the best fitness stagnates early, the algorithm may have prematurely converged to a suboptimal solution. Convergence occurs when the best and worst scores stabilize and remain close together. A wide gap between best and worst scores suggests high diversity in the population. If the worst score is constantly low, it may indicate poor-quality individuals or unfit solutions. The X on the Y-axis represents a worst individual with an empty prompt, which was detected by the fallback mechanism described in D and assigned a fitness score of -1, not represented in the y-axis scale in order not to skew the graph.

CLAUDETTE The convergence plot for the top multi-label runs are in Fig. 31, 32, 33, and 34. For binary, they can be found in in Fig. 35, 36, 37, and 38.

MRPC The convergence plot for the top runs are in Fig. 39, 40, 41, and 42.

N.3 Diversity Plot

The diversity plot tracks the number of unique individuals and prompts across generations to assess genetic diversity. This multi-line plot shows the unique count of prompt 1, prompt 2, and unique individuals. The x-axis represents the generation number, while the y-axis represents the count of unique individuals. A high count indicates high diversity, suggesting that the GA is still exploring solutions, whereas a sharp drop in diversity sug-

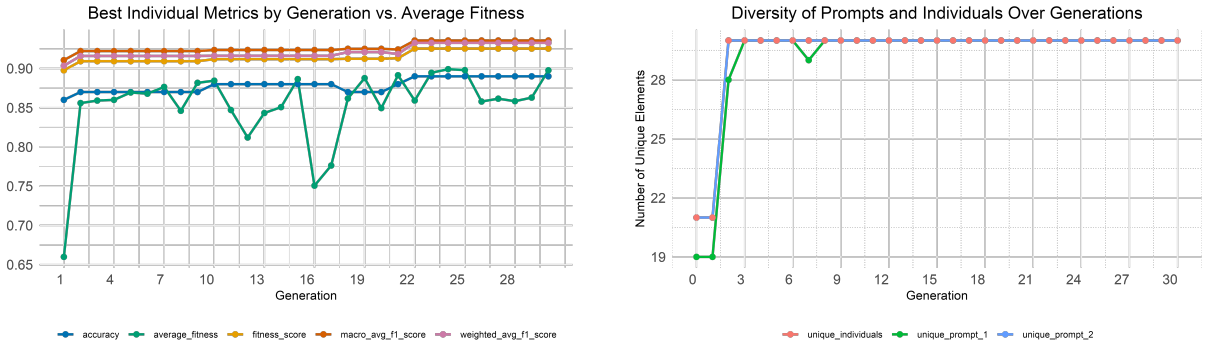

Figure 9: CLAUDETTE - **Left:** plot of metrics and average fitness for best multi-label run B in 9. **Right:** Diversity plotting for best multi-label run B in Table 9

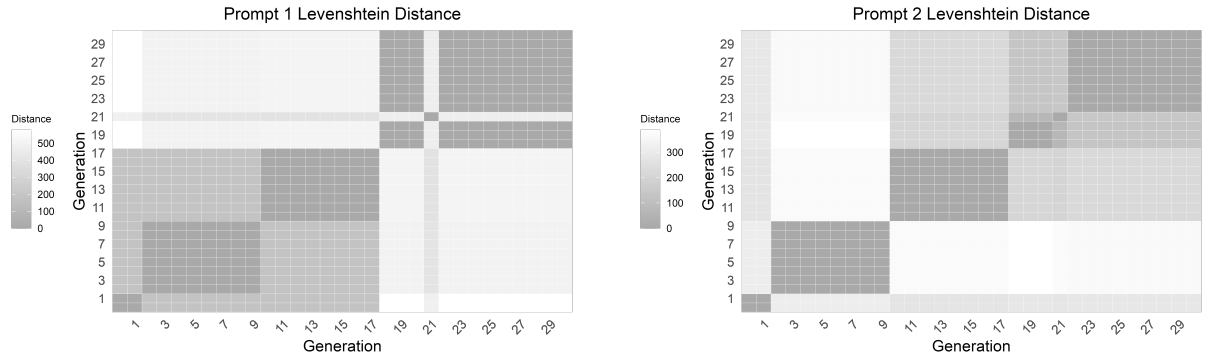

Figure 10: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run B in 9.

gests exploitation, whereby the same individual is being selected for the next generation several times due to high selection pressure. Diversity is crucial for exploration in early generations. The GA may get stuck in a local optimum if diversity drops too early. If diversity remains high for too long, the GA may struggle to converge.

CLAUDETTE Diversity plots for the top multi-label runs are on the right side of Fig. 7, 9 and 11, 13. For the binary runs, diversity plots are on the right of Fig. 15, 17 and 19, 21.

MRPC Diversity plots for the top runs are on the right side of Fig. 23, 25 and 27, 29.

N.4 Similarity Heatmaps

The similarity heatmap compares the similarity of best individuals across generations using Levenshtein distance. These plots take the form of heatmaps where the x-axis and y-axis represent generations, and the color intensity represents the distance. The darker the color, the more similar (smaller distance) the prompts are. The Levenshtein distance measures character-level differences between best individuals. If distances are high between adjacent generations, it suggests significant mutation and exploration. If distances are low, it suggests convergence and exploitation. Each cell compares the similarity of the best individuals from one generation to another. Diagonal cells should always be darkest since they compare identical generations. Clusters of dark squares suggest stable solution phases in the GA. Although we also plotted the tokenized version of this (where token distance rather than character distance is compared), the plots differ very slightly and globally communicate the same information.

CLAUDETTE Prompt similarity plots for the top 4 multi-label runs are in Fig, 8, 10, 12, and 14. For the binary they are in Fig. 16, 18, 20, and 22.

MRPC Prompt similarity plots for the top 4 runs are in Fig, 24, 26, 28, and 30.

N.5 Summary of Plot Interpretations

The combination of these plots provides a comprehensive view of how the genetic algorithm progresses over time. The metrics over generations

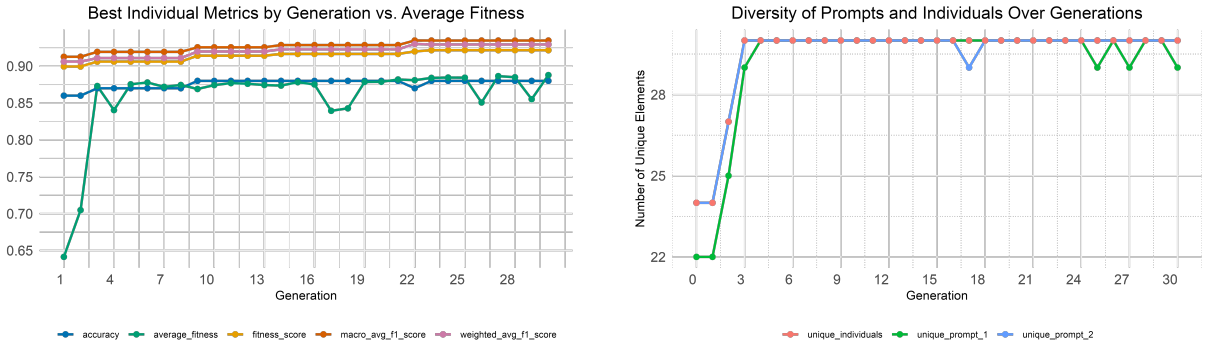

Figure 11: CLAUDETTE - **Left:** plot of metrics and average fitness for best multi-label run C in 9. **Right:** Diversity plotting for best multi-label run C in 9

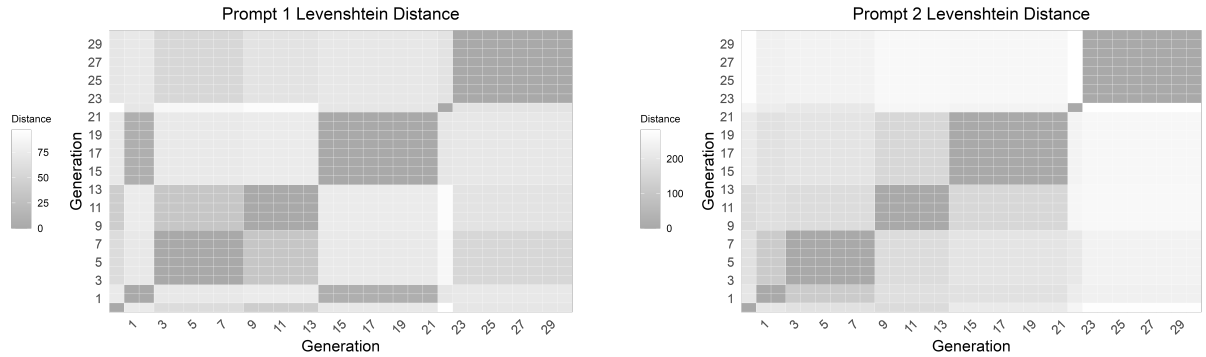

Figure 12: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run C in 9.

plot tracks performance trends, the convergence plot highlights stability and volatility, the diversity plot indicates exploration versus exploitation, and the similarity heatmaps reveal how best individuals evolve.

O Ablation Study

Comparing the pre and post-ablation metric plots (Fig. 43), we observe that the post-ablation plot flatlines for all metrics, including average fitness (and looks similarly flat for the binary case). In contrast, the pre-ablation plot shows a clear trend of exploration and improvement, demonstrating the role of selection in guiding the search toward optimal solutions. By removing it, the evolutionary process collapses into a random stagnating search.

P LLM-Safe MRPC

We performed a thorough preprocessing of the Microsoft Research Paraphrase Corpus (MRPC) (Dolan and Brockett, 2005) to ensure its suitability for modern large language model (LLM) pipelines. MRPC consists of sentence pairs extracted from news sources, labeled as semantically equivalent or not. Our preprocessing was carried out with the intent to sanitize potentially problematic content and eliminate parsing issues during downstream processing, which we faced in practice, when we attempted to run our framework on the unprocessed dataset.

P.1 Trigger Keyword Removal

We defined a list of content-sensitive trigger keywords that might introduce bias or lead to malformed LLM output due to content flagging. This list included terms such as: `["murder", "terrorist", "rape", "suicide", "nazi", "porn", "overdose", "deep state", ...]`

Using a compiled regex, we flagged and removed any sentence pair where either sentence contained one of these keywords. This was applied separately to the training and test sets. We flagged and removed 124 rows from the training set and 53 rows from the test set.

P.2 Quote Normalization

Many sentences contained unbalanced or malformed quote characters (e.g., unmatched ", improper smart quotes like “ and ”, or terminal es-

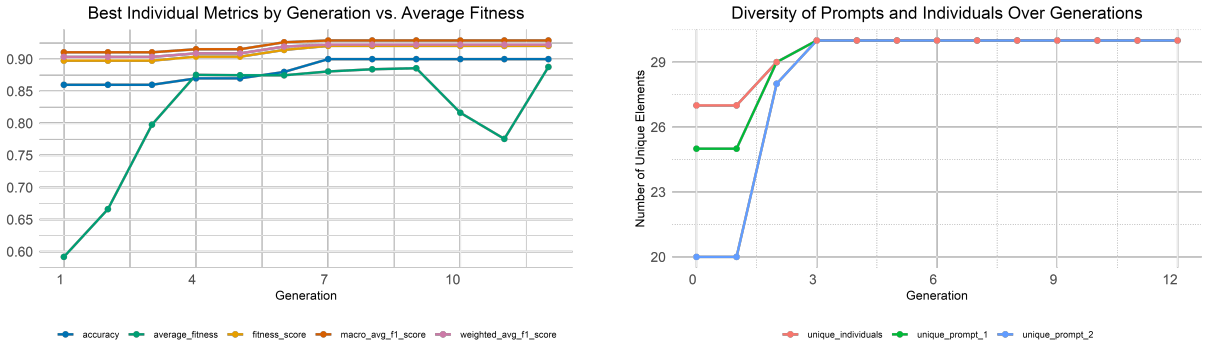

Figure 13: CLAUDETTE - **Left:** plot of metrics and average fitness for best multi-label run D in 9. **Right:** Diversity plotting for best multi-label run D in 9

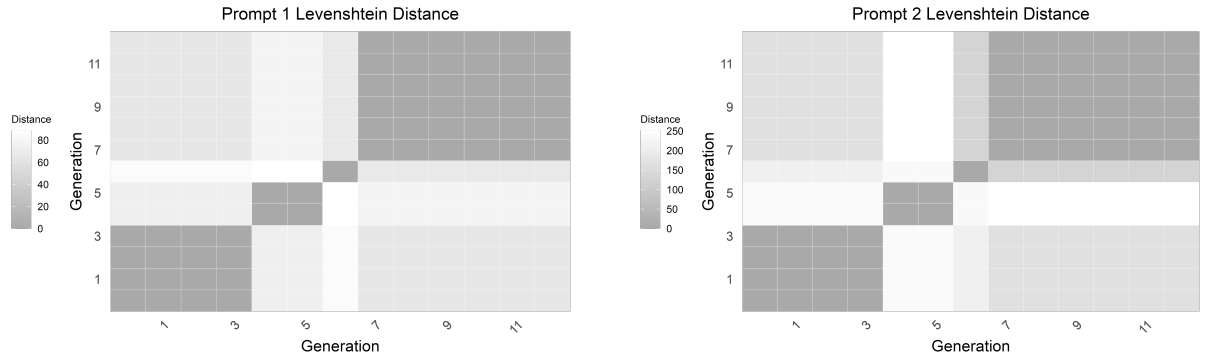

Figure 14: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run D in 9.

caped quotes like ¨). These were identified using a custom detection function that counted quote occurrences per sentence and flagged anomalies where the quote count was odd. We manually corrected 374 such cases across both sentence columns. All forms of quotation marks were then normalized to a single safe, non-standard Unicode character (U+2033 Double Prime), visually identical to a double quote, and interpreted the same by an LLM, but would not interfere with JSON parsing.

P.3 Final Output

The final version of the dataset:

- Contains only rows free of trigger words.
- Has quote balance issues corrected across all sentence pairs.
- Is JSON-safe and fully parsable by LLMs and downstream systems.

We refer to this cleaned version as the **LLM-Safe MRPC Dataset** and use it consistently throughout our experiments.

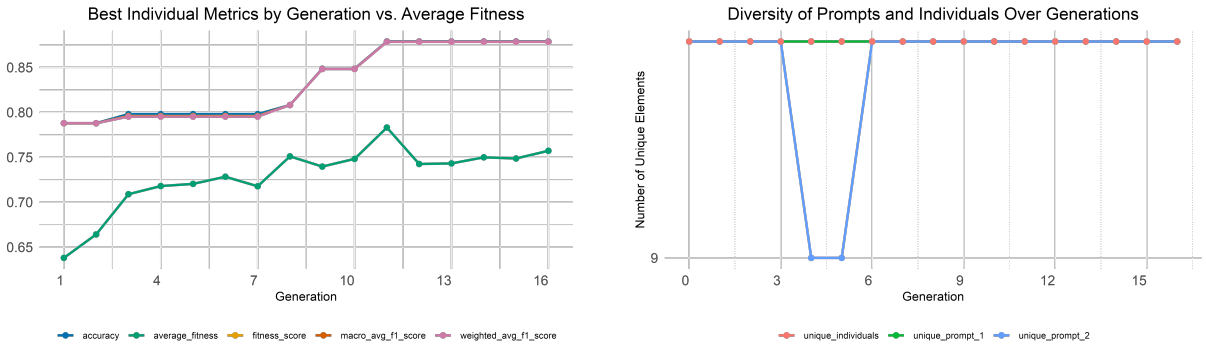

Figure 15: CLAUDETTE - **Left:** plot of metrics and average fitness for best binary run A in 8. **Right:** Diversity plotting for best binary run A in Table 8.

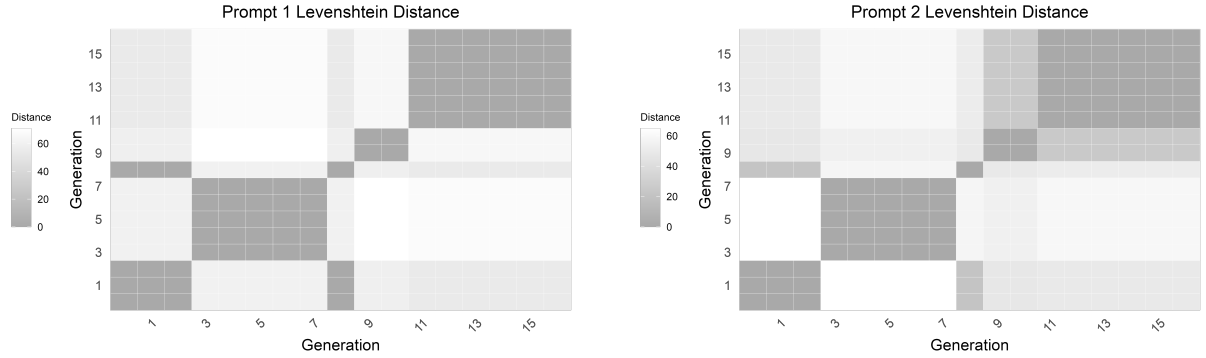

Figure 16: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best binary run A in Table 8.

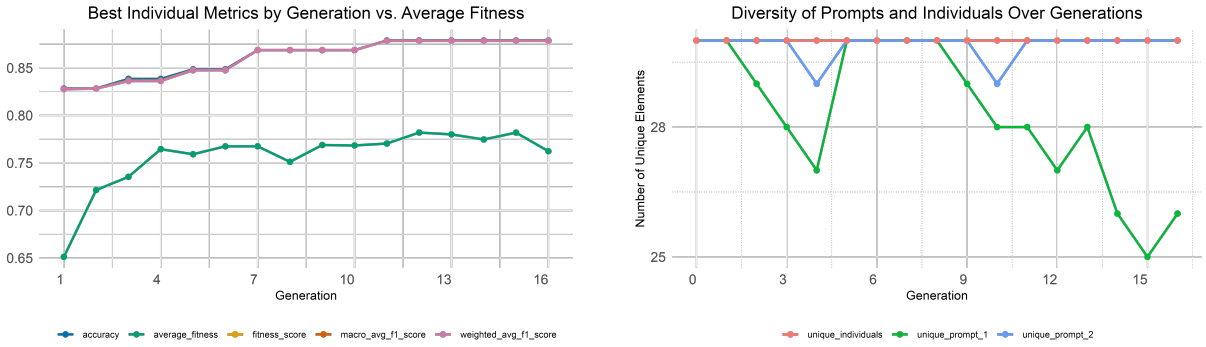

Figure 17: CLAUDETTE - **Left:** plot of metrics and average fitness for best run B in 8. **Right:** Diversity plotting for best binary run B in Table 8.

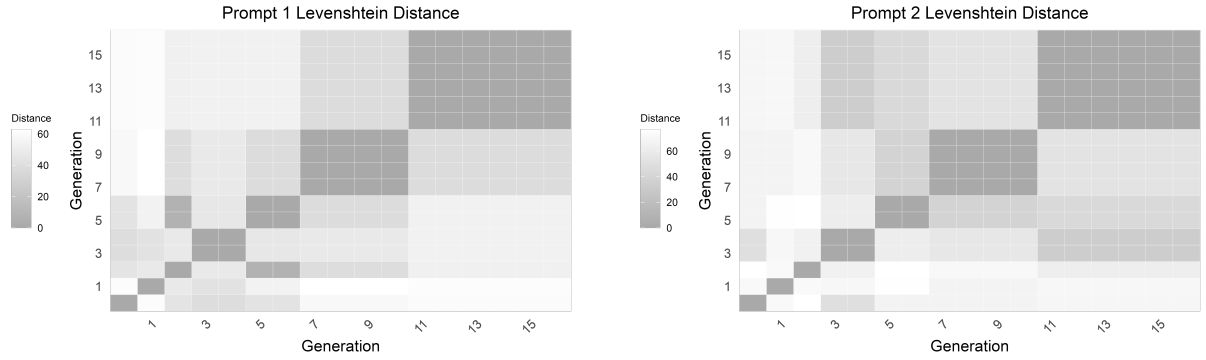

Figure 18: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best binary run B in Table 8.

Prompt 1	Prompt 2
Create a feature-enriched output that provides a reasoning for each sentence's most likely classification.	For each sentence contained within the input data, evaluate and accurately classify it into one or more of the following categories: ((category listing ...)) Carefully analyze the content and implications within each sentence to determine the comprehensive set of categories it belongs to.
Generate an explanation-rich classification for each sentence, including the reasoning behind the classification decision.	Analyze each sentence in the input data and classify it into one or more relevant categories based on their content and implications, ensuring precision in multi-label classification.
Provide a detailed analysis for each sentence, outlining the reasoning for its classification into the most likely category.	Perform a comprehensive classification of each input sentence into appropriate categories, ensuring all applicable labels are captured.
Construct a comprehensive output that explains the rationale for each sentence's classification.	Evaluate each sentence thoroughly, assigning it to relevant categories and providing precise multi-label classifications.
Develop an enriched response that details the reasoning for each sentence's assigned classification.	Classify the input sentences, ensuring a rigorous multi-label classification for relevant aspects such as: ((category listing ...))
Offer a feature-oriented output that justifies the classification of each sentence with clear reasoning.	For every sentence in the dataset, determine the applicable categories and provide an accurate multi-label classification for these: ((category listing ...))
Generate a detailed report justifying each sentence's classification with specific reasoning.	Thoroughly analyze each sentence to classify it into one or more relevant categories, capturing all dimensions of the classification.
Create a classification output enriched with reasoning for every sentence in the input.	Assign appropriate classifications to each input sentence, reflecting its content and intent while addressing these categories: ((category listing ...))
Produce an output that pairs each sentence with an explanation for its classification.	Evaluate and classify each sentence in the dataset into all relevant categories, focusing on ((category listing ...)).
Develop a thorough output that provides reasoning for the classification of each input sentence.	Analyze the input data sentence by sentence to identify the most applicable categories for each, ensuring completeness in multi-label classification.
Deliver a reasoning-augmented classification output for each provided sentence.	Classify the content of each sentence with a focus on accurate multi-label categorization, rigorously addressing ((category listing ...)).

Table 4: CLAUDETTE - Manual multi-label prompt bank used to initialize every GenDLN multi-label run.

Prompt 1	Prompt 2
You are a linguistic analysis model specialized in paraphrase tasks. For each input pair, extract key semantic and syntactic features relevant for paraphrase classification.	You are an expert in paraphrase detection. In the following your task is to analyze if sentence 2 is a paraphrased version of sentence 1. Thus, you shall classify each sentence pair into 0 ('not equivalent') or 1 ('equivalent') depending on whether sentence 1 and 2 are semantically equivalent.
Analyze each sentence pair to identify meaningful features that help determine if the two sentences are paraphrases.	Given each sentence pair, determine if the second sentence is a paraphrase of the first. Output 1 if they are semantically equivalent, 0 if they are not.
Given a list of sentence pairs, extract discriminative features for each pair that can support downstream paraphrase detection.	Your job is to judge whether the meaning of sentence 1 is preserved in sentence 2. Classify the pair as 1 for paraphrase or 0 for non-paraphrase.
You are tasked with analyzing sentence pairs. For each pair, return a compact description of important features that would help in classifying paraphrase relationships.	Classify each sentence pair by checking if sentence 2 can be considered a paraphrase of sentence 1. Use 1 for equivalent, 0 for not equivalent.
Analyze the input sentence pairs and extract useful features that would support a classifier in detecting semantic equivalence.	You are a paraphrase classification assistant. For each sentence pair, assign a binary label: 1 if sentence 2 is a paraphrase of sentence 1, else 0.
You are a feature extraction system for paraphrase detection. For each sentence pair, output key comparison features in the specified format.	You are to detect paraphrases. For each sentence pair, determine if both express the same meaning. Label with 1 if equivalent, otherwise 0.
Given sentence pairs, identify and summarize linguistic or semantic cues that are relevant for determining paraphrasing.	For each given pair of sentences, assess whether sentence 2 paraphrases sentence 1. Output 1 for equivalent meaning, 0 for different meaning.
For each pair of sentences, write a brief set of features that capture their semantic, lexical, and structural alignment.	You are evaluating sentence-level semantic similarity. Classify each pair with 1 if both sentences are paraphrases, and 0 if they are not.

Table 5: MRPC - Manual binary prompt bank (Part 1/3) used to initialize GenDLN binary runs.

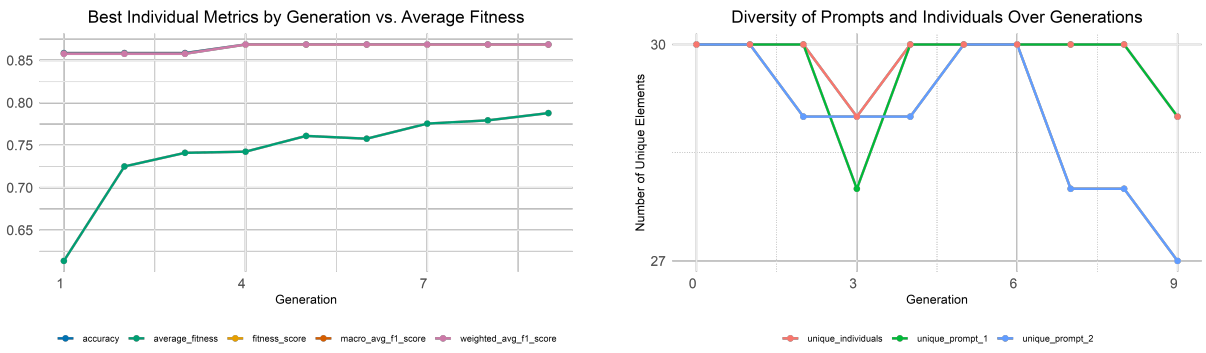

Figure 19: CLAUDETTE - **Left:** plot of metrics and average fitness for best run C in 8. **Right:** Diversity plotting for best binary run C in Table 8.

Prompt 1	Prompt 2
Inspect each input sentence pair and generate a meaningful feature description that reflects their similarity or difference in meaning.	You are an NLP expert assessing paraphrase relationships. Label each sentence pair as 1 if semantically equivalent, else 0.
You are a natural language understanding model. For each sentence pair, extract features that reveal differences or overlaps in meaning and expression.	You are a binary classifier for sentence equivalence. Judge whether sentence 2 retains the meaning of sentence 1. Output 1 or 0 accordingly.
Identify semantic relationships and stylistic variations in each sentence pair. Output concise features that explain their alignment or divergence.	Your goal is to assess if sentence 2 can be considered a reasonable paraphrase of sentence 1. Output 1 if so, otherwise 0.
For every input pair, generate a feature-based comparison that highlights differences in structure, meaning, or terminology.	Examine the semantic content of each sentence pair and decide if they convey the same core meaning. Return 1 for paraphrase, 0 for otherwise.
You are helping a classifier understand sentence similarity. Extract key features that could guide a model in deciding paraphrase equivalence.	Determine whether sentence 2 is interchangeable with sentence 1, i.e. a suitable paraphrase. Output 1 if they are interchangeable, else 0.
Assess each sentence pair for shared meanings, nuanced differences, or structural shifts. Provide these insights as short, structured features.	You are assessing paraphrase validity. Classify each pair as 1 if the second sentence accurately reflects the meaning of the first, or 0 if not.
Your goal is to support a paraphrase detection system by extracting features that capture lexical, syntactic, and semantic properties of sentence pairs.	For every pair, identify whether sentence 2 expresses the same meaning as sentence 1 using a binary label: 1 (yes), 0 (no).
Review each sentence pair and write a concise summary of alignment cues and linguistic differences that may affect paraphrase detection.	Your task is to judge if sentence 2 carries the same intent and meaning as sentence 1. Output 1 for equivalence, 0 otherwise.

Table 6: MRPC - Manual binary prompt bank (Part 2/3) used to initialize GenDLN binary runs.

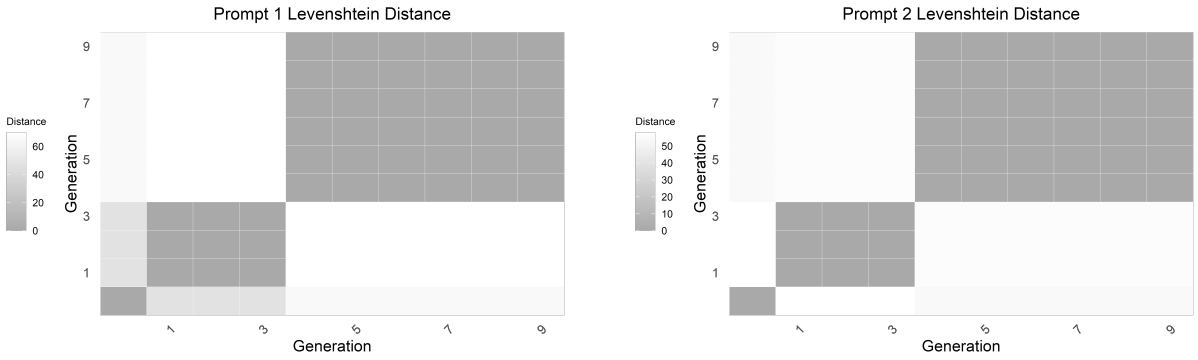

Figure 20: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best binary run C in Table 8.

Prompt 1	Prompt 2
As a sentence-level feature extractor, outline the textual signals that could be used to determine if two statements express the same idea.	Determine semantic equivalence at the sentence level. For each pair, output 1 if meaning is preserved between the two sentences, 0 if it is lost or altered.
Examine each sentence pair and extract distinguishing features that would help a downstream model judge paraphrase likelihood.	Review each sentence pair and determine whether sentence 2 retains the essential meaning of sentence 1. Respond with 1 for equivalence, 0 otherwise.
Your job is to find patterns in sentence pairs that indicate whether they express similar or different meanings. Output a compact list of relevant features.	Your job is to classify whether sentence 2 can logically be interpreted as expressing the same idea as sentence 1. Output 1 for yes, 0 for no.
You are a linguistic alignment engine. Identify whether key predicates, named entities, and relationships are preserved across the sentence pair.	Assess whether sentence 2 paraphrases sentence 1 without introducing or omitting critical information. Output 1 for paraphrase, 0 if meaning changes.
Highlight phrasing shifts, information asymmetry, or reordering patterns that could influence whether the sentence pair is semantically aligned.	For each pair of statements, decide whether sentence 2 communicates the same content as sentence 1. Respond with 1 for equivalent, 0 for not equivalent.
For each input pair, extract lexical and structural markers - including synonym usage, clause structure, and entity alignment - that contribute to paraphrase detection.	Analyze the sentence pair and determine if their meanings align well enough to be considered paraphrases. Output 1 if they do, 0 if not.
Extract the central premise of each of the two sentences, what information does each convey?	Are they paraphrases of each other? Output 1 for yes, 0 for no.
As an expert writer, would you say the two sentences convey the same main idea? What would you say is the point of each sentence?	Would it be reasonable to replace one sentence with the other in a text without changing the overall meaning? In other words, are the sentences paraphrases of each other? Output 1 if yes and 0 if no.
Could the two sentence reasonably be exchanged within a text without changing the general meaning of the text? Why or why not?	Given that assessment, can the sentences be classified as paraphrases of each other? Answer with 1 if they are paraphrases, and 0 if not.

Table 7: MRPC - Manual binary prompt bank (Part 3/3) used to initialize GenDLN binary runs.

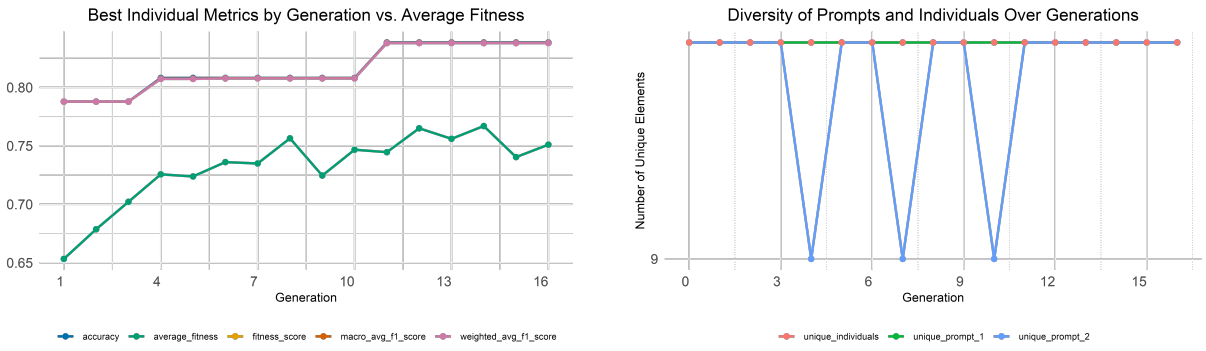

Figure 21: CLAUDETTE - **Left:** plot of metrics and average fitness for best run D in 8. **Right:** Diversity plotting for best binary run D in Table 8.

Metric	Run A	Run B	Run C	Run D
Runtime (mins)	58.565	160.9097	100.8069	53.262
Best Fitness	0.8785	0.8785	0.8687	0.8380
Best Accuracy	0.8788	0.8788	0.8687	0.8384
Test. Accuracy	0.7897	0.7706	0.7646	0.7404
Best Macro F1	0.8785	0.8785	0.8686	0.8380
Test. Macro F1	0.6523	0.6364	0.6338	0.6172
Best Weighted F1	0.8784	0.8784	0.8687	0.8379
Test. Weighted F1	0.8256	0.8115	0.8073	0.7894
Selection Strategy	Rank	SUS	SUS	Rank
Crossover Type	Semantic Blending	Token Level	Semantic Blending	Semantic Blending
Crossover Rate	0.800	0.800	0.800	0.800
Mutation Type	Semantic	Syntactic	Semantic	Semantic
Mutation Rate	0.200	0.200	0.200	0.200
Population Size	10	30	30	10
Completed Generations	16	16	9	16
Stopped Early	Yes	Yes	Yes	Yes
Stopped Early Reason	5 stag. gens.	5 stag. gens.	5 stag. gens.	5 stag. gens.

Table 8: CLAUDETTE - Selected runs for binary (fair/unfair) classification.

Metric	Run A	Run B	Run C	Run D
Runtime (mins)	469.689	439.694	373.876	155.367
Best Fitness	0.938	0.925	0.922	0.921
Best Accuracy	0.910	0.890	0.880	0.900
Test. Accuracy	0.825	0.769	0.809	0.802
Best Macro F1	0.947	0.936	0.935	0.929
Test. Macro F1	0.862	0.799	0.844	0.855
Best Weighted F1	0.944	0.933	0.929	0.923
Test. Weighted F1	0.856	0.808	0.842	0.851
Selection Strategy	Rank	Steady-State	SUS	Steady-State
Crossover Type	Phrase Swap	Phrase Swap	Token Level	Semantic Blending
Crossover Rate	0.850	0.850	0.850	0.800
Mutation Type	Insertion	Insertion	Syntactic	Semantic
Mutation Rate	0.300	0.300	0.300	0.200
Population Size	30	30	30	30
Completed Generations	30	30	30	12
Stopped Early	No	No	No	Yes
Stopped Early Reason	-	-	-	5 stag. gens.

Table 9: CLAUDETTE - Selected best runs for multi-label classification.

Metric	Run A	Run B	Run C	Run D
Runtime (mins)	137.681	167.228	67.070	127.262
Best Fitness	0.850	0.840	0.850	0.840
Best Accuracy	0.850	0.840	0.850	0.840
Test. Accuracy	0.813	0.807	0.798	0.799
Best Macro F1	0.850	0.840	0.850	0.840
Test. Macro F1	0.796	0.787	0.782	0.781
Best Weighted F1	0.850	0.840	0.850	0.840
Test. Weighted F1	0.816	0.809	0.802	0.802
Selection Strategy	Steady-State	Roulette	Tournament	SUS
Crossover Type	Single Point	Semantic Blending	Token Level	Two Point
Crossover Rate	0.85	0.85	0.85	0.80
Mutation Type	Semantic	Insertion	Insertion	Deletion
Mutation Rate	0.20	0.20	0.20	0.20
Population Size	30	30	30	30
Completed Generations	16	23	12	15
Stopped Early	Yes	Yes	Yes	Yes
Stopped Early Reason	10 stag. gens.	10 stag. gens.	10 stag. gens.	10 stag. gens.

Table 10: MRPC - Selected best runs for binary paraphrase classification.

Run	Prompt Text
A	**Prompt 1**: Assess the potential legal consequences and issues of the following sentence. **Prompt 2**: Based on the previous discussion, would you consider this sentence to be fair or unfair as it stands?
B	**Prompt 1**: Interpret the following sentence in any hidden clauses or implications. **Prompt 2**: Will the described potential impact be considered fair or unfair?
C	**Prompt 1**: Assess the possible legal ramifications and effect on consumer rights of the following sentence. **Prompt 2**: Considering the impact of the ethical implications discussed, is this sentence fair or unfair in its current phrasing?
D	**Prompt 1**: Identify any potential legal issues when analyzing the meaning of the following sentence in a legal context. **Prompt 2**: Given the emphasized issues, is this sentence fair or unfair in its current state?

Table 11: CLAUDETTE - Prompt 1 and 2 of the best individuals for the runs as reported in Table 8 for the binary classification task.

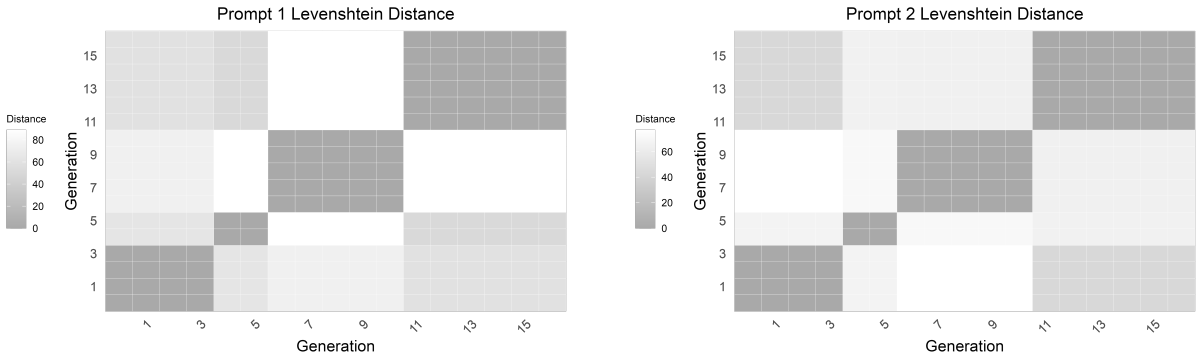

Figure 22: CLAUDETTE - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best binary run D in Table 8.

Run	Prompt Text
A	**Prompt 1**: To enhance transparency for the end user, who may not be familiar with the internal mechanics of our system, we should annotate each individual sentence contained within the given customer review that is specifically about our recently introduced product, including a clear, concise, and straightforward explanation that meticulously details the reasoning, justification, and rationale behind its specific classification, ensuring that the user comprehends why we classified the sentence as such. **Prompt 2**: To thoroughly organize and accurately assign a precise data monitoring technique or pertinent cookie policies that are explicitly outlined in a legal privacy policy document, a team of legal experts should meticulously review the entire policy document, starting from the introduction to the conclusion, and systematically classify each individual clause from the contract with high precision during the detailed multi-label classification process, ensuring that the resulting labels are not only relevant to the contractual obligations clearly outlined in the legal documents but also precise in their legal definition.
B	**Prompt 1**: To ensure thorough documentation and transparency in our contractual legal analysis efforts within the jurisdiction of the relevant state legal system, produce a comprehensive legal classification of the content within each individual clause that is clearly outlined in the case files pertaining to the ongoing corporate lawsuit. **Prompt 2**: When examining corporate legal documents, such as those related to IT service agreements, systematically classify each individual sentence from various types of contractual clauses, including confidentiality, liability, and termination clauses, into relevant and predefined labels for better organization and analysis.
C	**Prompt 1**: Present a detailed report on the categorization of every sentence, accompanied by relevant evidence. **Prompt 2**: Every sentence, in the multi-label classification process, will be assigned to its fitting categories to maintain it thoroughly, emphasizing suitable labels that range from PINC for cookie and tracking to LAW for legal frameworks.
D	**Prompt 1**: Generate a feature-focused output that matches each sentence with a reason for its categorization. **Prompt 2**: Sort and classify each sentence in the dataset, taking into account these categories: PINC (Cookies or data collection), USE (Rules on user activities), CR (Removal rights), TER (Service terminations), LTD (Limitation of liability), A (Arbitration resolutions), LAW (Governing legal codes), J (Jurisdiction clauses), CH (Agreement changes).

Table 12: CLAUDETTE - Prompt 1 and 2 of the best individuals for the runs as reported in Table 9 for the multi-label task.

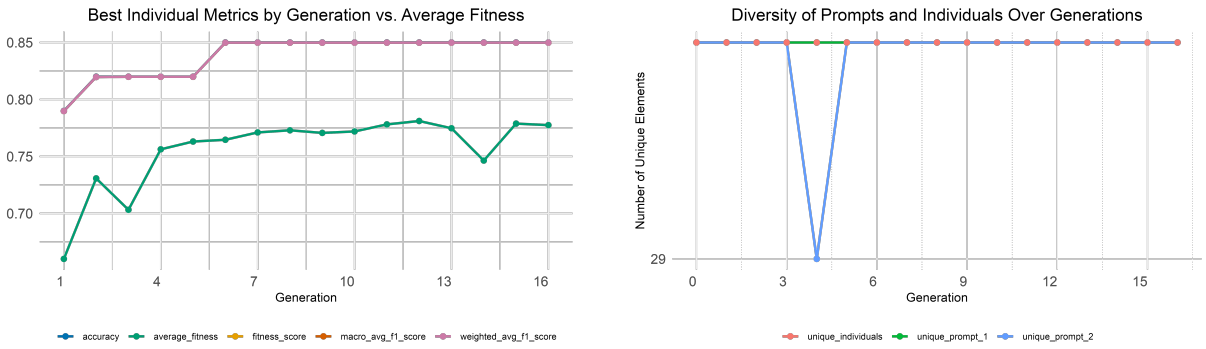

Figure 23: MRPC - **Left:** plot of metrics and average fitness for best run A in Table 10. **Right:** Diversity plotting for best multi-label run A in Table 10

Run	Prompt Text
A	**Prompt 1**: Assess each pair of sentences and generate a feature-based comparison that highlights differences in structure, meaning, or terminology. **Prompt 2**: You are evaluating each pair of sentences to determine if they express the same central meaning; return 1 if they are paraphrases, and 0 otherwise.
B	**Prompt 1**: For each individual pair of sentences that you evaluate within a comparative text analysis study, output a meaningful feature description that accurately captures their shared meanings, specific word choices, sentence structure, and stylistic differences. **Prompt 2**: After carefully examining each individual pair of sentences for their meaning and content, determine if they are paraphrases and convey the same meaning; label with a 1 if they are semantically equivalent, otherwise label them with a 0.
C	**Prompt 1**: For each sentence pair, extract semantic relationships and output concise features that reveal differences or overlaps in meaning and expression. **Prompt 2**: Your goal is to assess whether or not sentence 2 retains the meaning of sentence 1, taking into account all aspects of semantics and context. Judge whether sentence 2 can be considered a reasonable paraphrase of sentence 1, with an equivalent core interpretation. Output 1 for yes or 0 for no accordingly.
D	**Prompt 1**: Compare each sentence pairs that reveal distinguishing features in meaning. **Prompt 2**: Judge whether they are expressing the same intent of each other in a text.

Table 13: MRPC - Prompt 1 and 2 of the best individuals for the runs as reported in Table 10 for the paraphrase classification task.

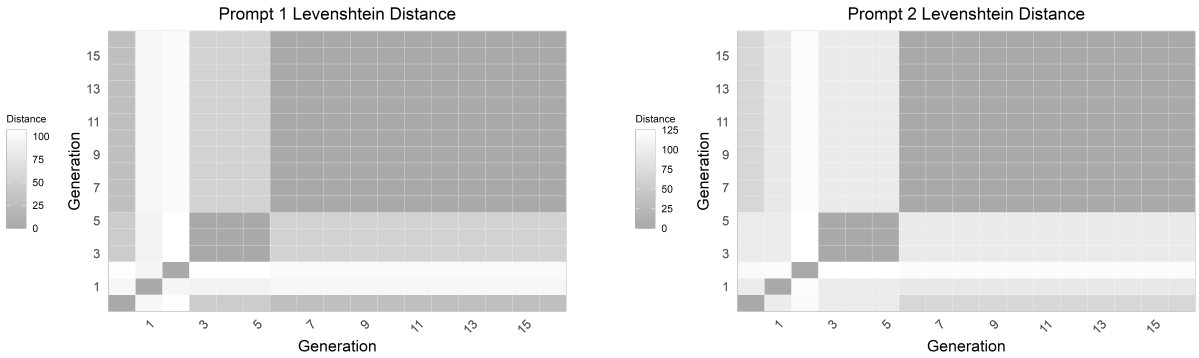

Figure 24: MRPC - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run A in Table 10.

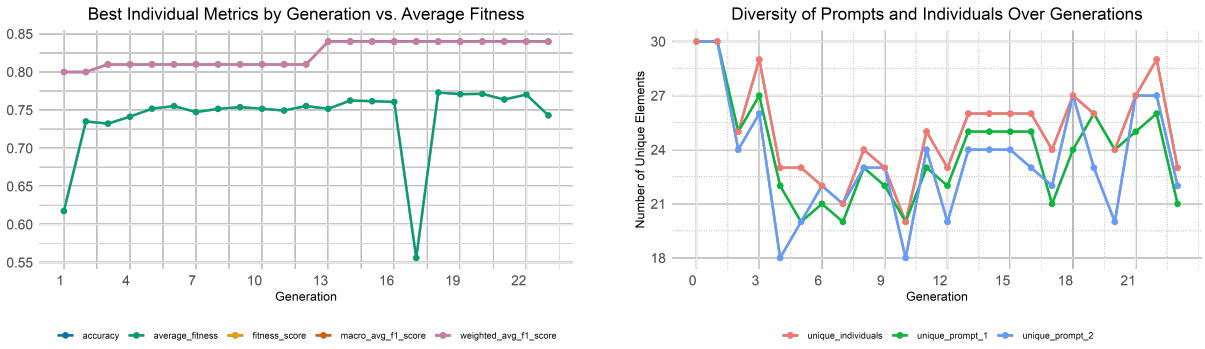

Figure 25: MRPC - **Left:** plot of metrics and average fitness for best run A in Table 10. **Right:** Diversity plotting for best multi-label run B in Table 10

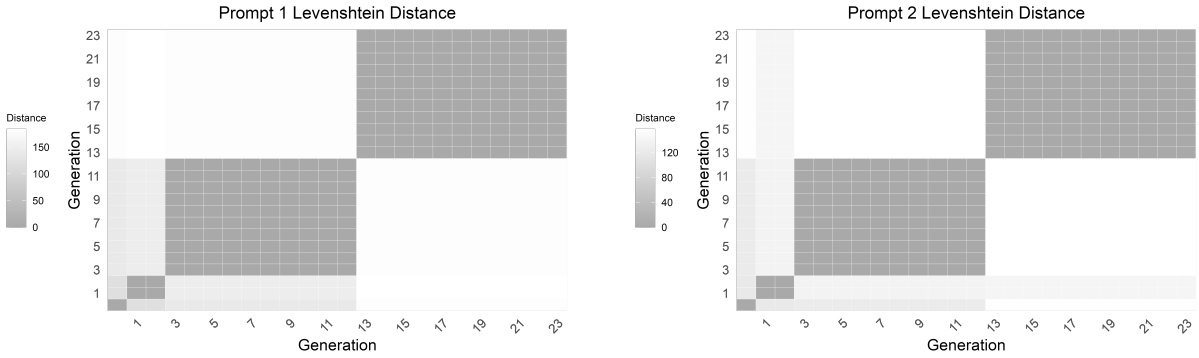

Figure 26: MRPC - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run B in Table 10.

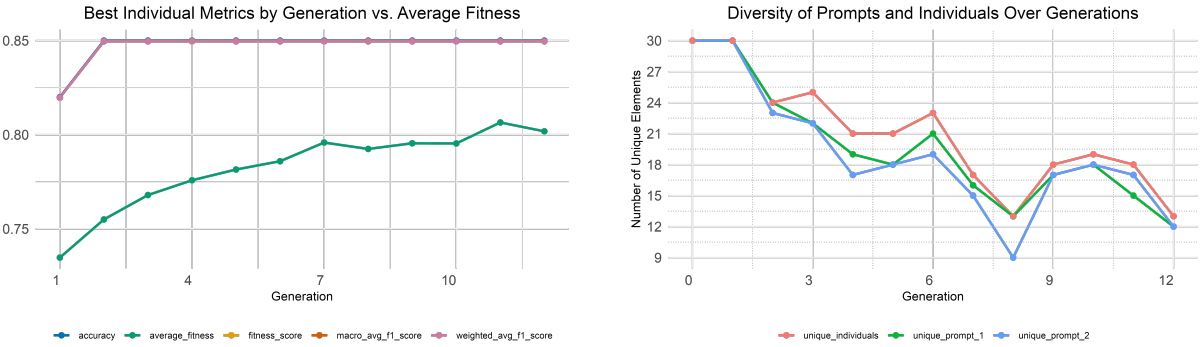

Figure 27: MRPC - **Left:** plot of metrics and average fitness for best run C in Table 10. **Right:** Diversity plotting for best multi-label run C in Table 10

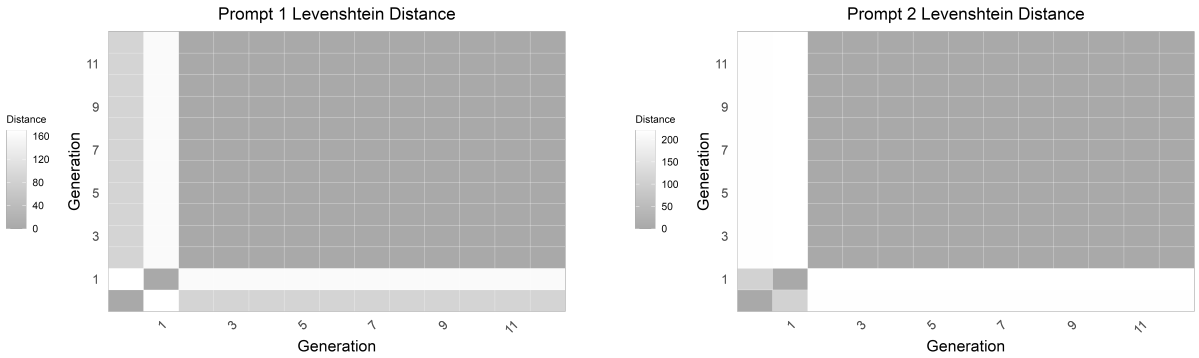

Figure 28: MRPC - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run C in Table 10.

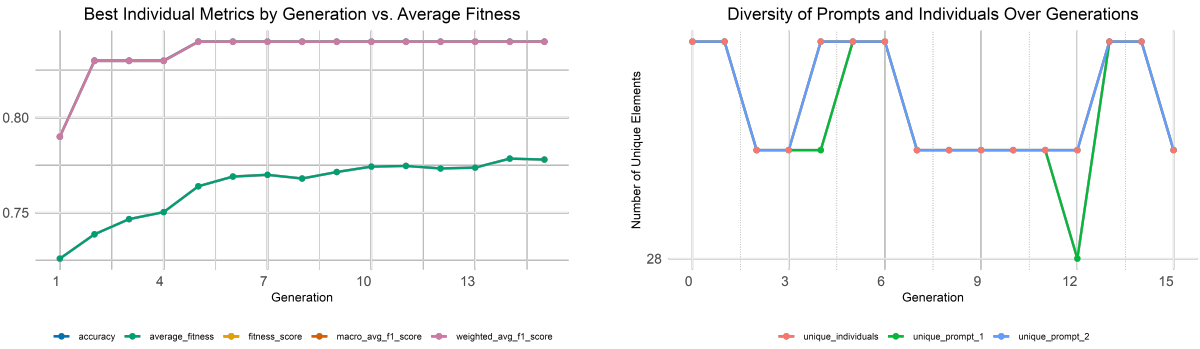

Figure 29: MRPC - **Left:** plot of metrics and average fitness for best run D in Table 10. **Right:** Diversity plotting for best multi-label run D in Table 10

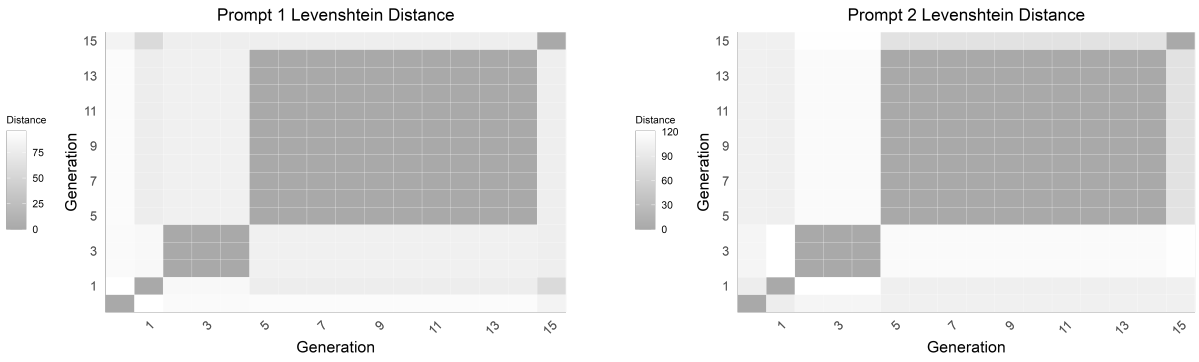

Figure 30: MRPC - Best Prompt 1 and Prompt 2 Levenshtein distance matrix across generations for best multi-label run D in Table 10.

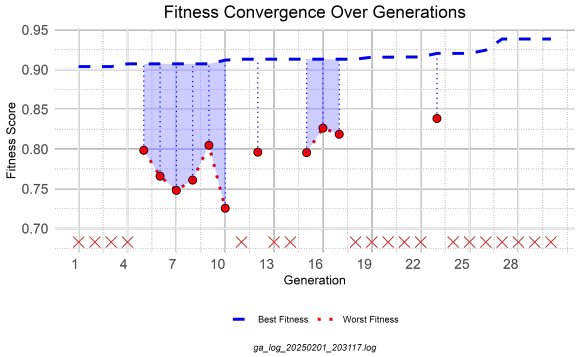

Figure 31: CLAUDETTE - Convergence plot for best multi-label run A in Table 9. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

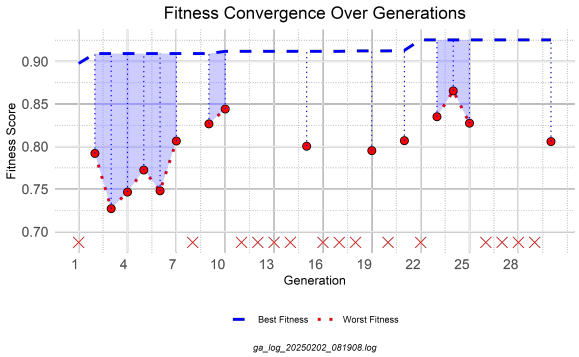

Figure 32: CLAUDETTE - Convergence plot for best multi-label run B in Table 9. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

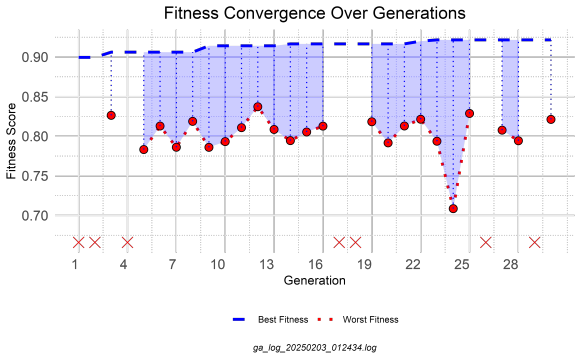

Figure 33: CLAUDETTE - Convergence plot for best multi-label run C in Table 9. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

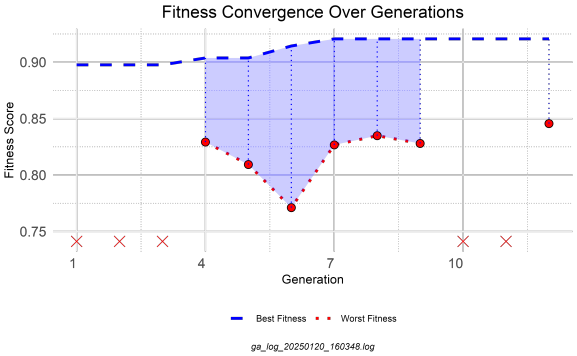

Figure 34: CLAUDETTE - Convergence plot for best multi-label run D in Table 9. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

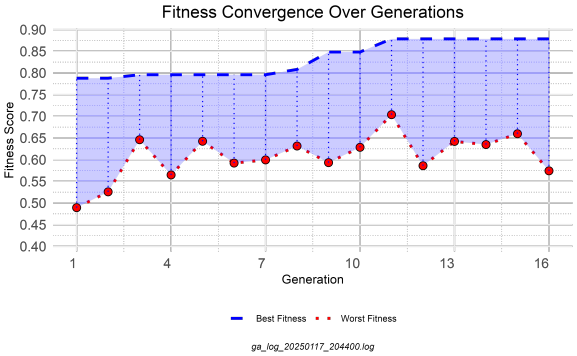

Figure 35: CLAUDETTE - Convergence plot for best binary run A in Table 8.

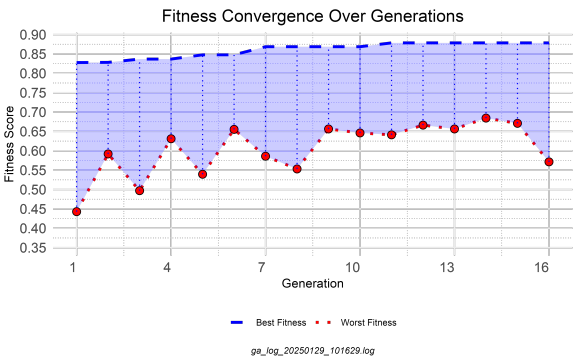

Figure 36: CLAUDETTE - Convergence plot for best binary run B in Table 8.

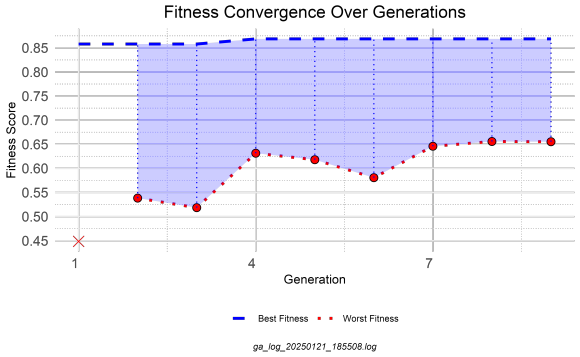

Figure 37: CLAUDETTE - Convergence plot for best binary run C in Table 8. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

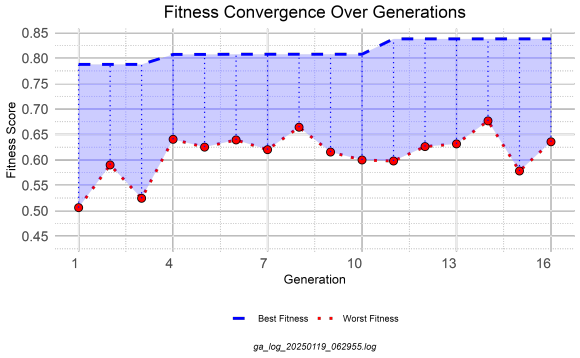

Figure 38: CLAUDETTE - Convergence plot for best binary run D in Table 8.

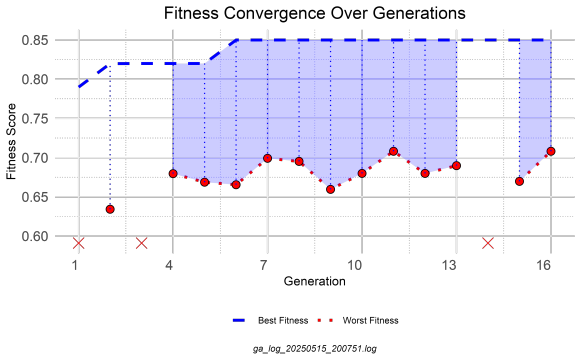

Figure 39: MRPC - Convergence plot for best binary run A in Table 10. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

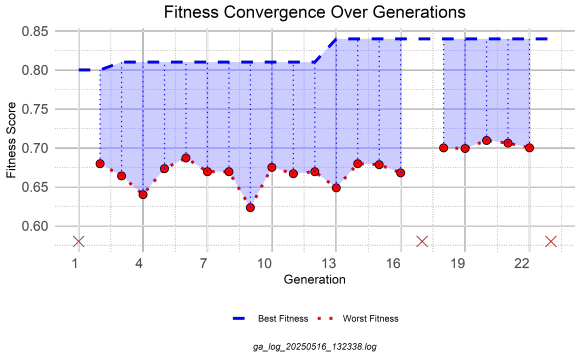

Figure 40: MRPC - Convergence plot for best binary run B in Table 10. X on the x-axis indicates an illegal individual as per the fallback mechanism in Appendix D.

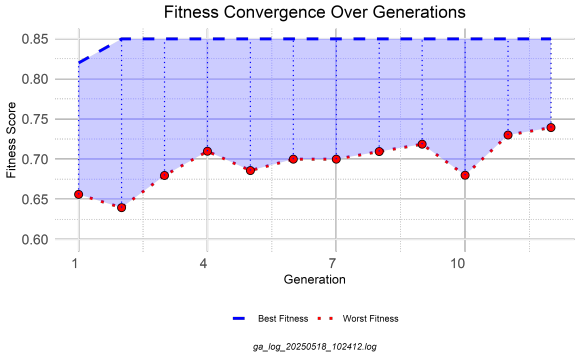

Figure 41: MRPC - Convergence plot for best binary run C in Table 10.

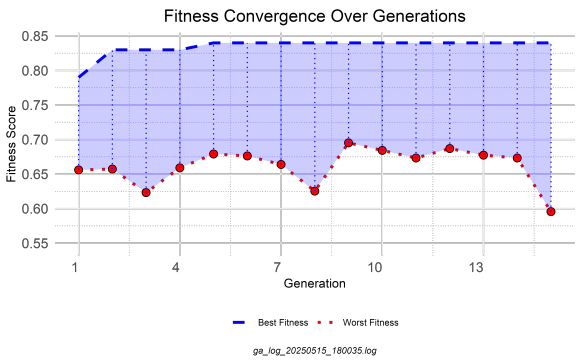

Figure 42: MRPC - Convergence plot for best binary run D in Table 10.

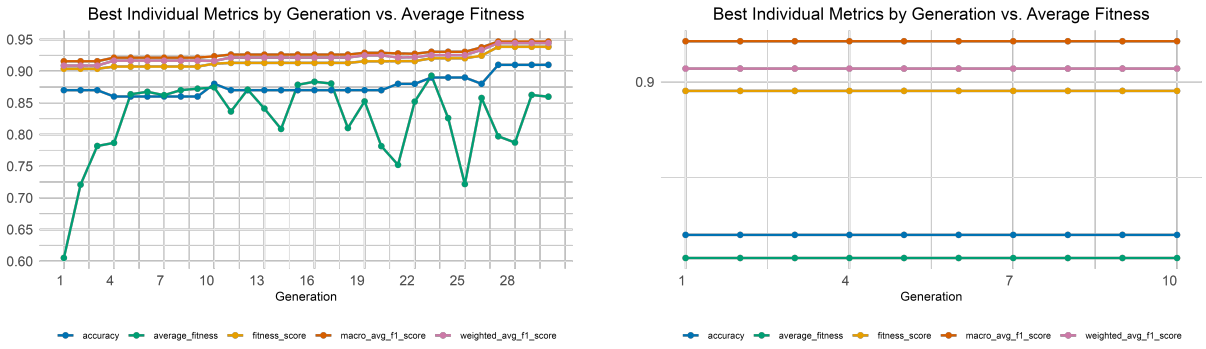

Figure 43: CLAUDETTE - **Left:** plot of metrics and average fitness for best multi-label run in Table 1. **Right:** Ablation of selection pressure for the same run.