Kaggle Santa 2024 Design System

1st Jairo Arturo Barrera Mosquera
20222020142
Ingenieria de Sistemas
UDFJC
Bogotá D.C, Colombia
jabarreram@udistrital.edu.co

2nd Gabriela Martínez Silva
20231020205
Ingenieria de Sistemas
UDFJC
Bogotá D.C, Colombia
gmartinezs@udistrital.edu.co

***Abstract*—This document describes the design of a system that reconstructs coherent passages from scrambled word lists. First, the input undergoes normalization and linguistic analysis (syntax and morphology) to identify constraints that prune the permutation space. Guided by these constraints, the system then generates only those word arrangements that are grammatically plausible, thereby minimizing computational resources.**

***Index Terms*—system, sentences, input, permutation, normalization**

I. Introduction

Reconstructing coherent sentences from scrambled word sequences is a central challenge in natural language processing (NLP), with applications ranging from machine translation and grammar correction to educational assessment tools. The task requires selecting a grammatically valid and semantically coherent ordering from a factorially large set of permutations. For example, a sequence of only 10 words yields over 3.6 million possible orderings. As such, exhaustive search strategies quickly become impractical.

Traditional approaches have often relied on probabilistic language models, such as n-gram models or more recently, deep neural architectures like GPT-2 or BERT, which use pretrained contextual embeddings to score candidate sequences [1][2]. These models, while powerful, are computationally intensive and often require large datasets and considerable fine-tuning. Other methods, including beam search, greedy decoding, and dynamic programming, offer more tractable solutions but still face difficulties handling syntactic ambiguity or enforcing hard grammatical constraints.

In response to these challenges, we propose a modular, linguistically informed system that aims to reorder words in a sentence efficiently and accurately without relying solely on heavy machine learning. Our method introduces two key innovations. First, it applies a normalization pipeline that prepares input by cleaning, lemmatizing, and analyzing word frequencies. Second, it performs syntactic and morphological analysis to extract grammatical constraints—such as common sentence structures (Subject–Verb–Object) and agreement features (noun–verb number agreement). These constraints are then used to guide a search algorithm that prunes the permutation space, focusing only on linguistically plausible arrangements.

By integrating structural insights with search heuristics, our approach achieves comparable coherence to neural models while significantly reducing computation time. This makes it particularly suitable for constrained environments, such as edge devices or educational tools where interpretability and efficiency are critical. We also demonstrate that our solution can outperform brute-force and unstructured heuristic approaches in both speed and quality of results on benchmark datasets.

II. Methods and Materials

A. Normalized Input

The Normalized Input component initiates the processing pipeline by transforming raw data into a structured format suitable for linguistic analysis and sentence generation. It operates on input files, typically in CSV format, where each row contains a unique passage identifier and a corresponding sequence of shuffled words. To handle this efficiently, the system employs the Pandas library to read and structure the data into a DataFrame, facilitating easy iteration over individual passages. Basic cleaning is performed using Python's built-in string methods, which remove leading or trailing whitespace from each token and ensure consistent formatting. To account for repeated words—an important requirement of the problem—the module uses collections.Counter, which provides a fast and reliable way to count occurrences of each word in the sequence. While the system aims to preserve the original case and punctuation of words for the final output, internal normalization (such as lowercasing) may be applied to support analysis. For cases involving punctuation or token boundary issues (distinguishing between "word" and "word."), NLTK or spaCy may be optionally used to apply minimal tokenization, though more advanced linguistic processing is deferred to later stages. The outcome of this module is a clean, verified data structure that includes the original word list, word frequencies, and any necessary annotations, thus ensuring data integrity and consistency across the system.

B. Template Definition

The Template Definition module is responsible for extracting grammatical structure and linguistic constraints from the normalized input data, guiding the sentence generation process that follows. This component receives the cleaned and tokenized list of words from the normalization stage and applies syntactic and morphological analysis to infer plausible sentence structures. To achieve this, the system utilizes pre-trained natural language processing libraries such as spaCy and Stanza, which provide robust tools for part-of-speech (POS) tagging, lemmatization, and dependency parsing. When a passage is processed through one of these models, each word is assigned attributes like syntactic role, grammatical function, and its most likely head in the dependency tree. From this annotated information, the system constructs soft structural templates—abstract sentence patterns such as [NOUN] [VERB] [ADJECTIVE] [NOUN]—and hard constraints, such as ensuring that verbs follow likely subjects or that adjectives precede nouns.

These templates serve as probabilistic blueprints for the next stage of the system: instead of exploring all possible permutations, the generator focuses only on those that are grammatically plausible based on the extracted patterns. For example, the system may derive that a word tagged as a verb is unlikely to be the first token if no subject has been placed yet. Such logic is encoded through rule-based heuristics and constraint conditions. The use of linguistic templates not only reduces the computational complexity of the generation phase but also increases the quality of output by prioritizing grammatically valid sequences. The modular design of this component also ensures flexibility: the choice of NLP library (spaCy or Stanza) can be adapted based on language, model size, or processing needs, while the template generation logic remains consistent.

C. Pemutation Generator

The Generate Permutations module is the core engine of the system, tasked with exploring and selecting the most plausible word orderings from the set of scrambled inputs. Unlike naïve approaches that rely on brute-force enumeration—an infeasible strategy due to the factorial growth of possible arrangements—this component employs intelligent search strategies to efficiently navigate the permutation space. The process begins with the structured input and linguistic templates provided by the previous stages. These templates define both syntactic constraints (subject–verb order) and semantic cues (likely adjective–noun pairs), which are used to prune invalid or low-quality sequences from the search space.

To construct candidate permutations, the system leverages a combination of heuristic-driven and optimization-based search algorithms. For small sequences, libraries such as Python's itertools may be used to exhaustively generate permutations, but for longer passages, the system incorporates more scalable methods like Genetic Algorithms and A Search*. In the genetic algorithm variant, permutations are treated as individuals in a population, and evolutionary operations such as crossover and mutation are applied to generate new candidates. Fitness is evaluated using a language model–based perplexity score, where lower values indicate higher fluency and coherence. Similarly, in the A* approach, states are modeled as partial sequences, and the algorithm selects the most promising continuations based on heuristics derived from linguistic templates and previous scores.

This module also interfaces with an evaluator—often built on top of pre-trained language models from libraries such as Hugging Face Transformers—to obtain perplexity scores for each candidate sentence. The search continues iteratively, guided by feedback from these evaluations, until a satisfactory solution is found or a computational limit is reached. By combining syntactic guidance, search-space reduction, and statistical evaluation, this component ensures the final output is both grammatically correct and semantically meaningful, while maintaining scalability across varying input sizes.

D. Evaluator

The Evaluator module is responsible for measuring the quality of candidate sentence permutations by computing a perplexity score, a common metric in natural language processing for evaluating fluency and grammatical plausibility. This component operates by comparing the submitted permutations against the original sequences using a pre-trained large language model. Specifically, it uses AutoTokenizer and AutoModelForCausalLM from the Hugging Face Transformers library to load a model such as Gemma 2 9B, which is capable of assessing token-level language probabilities. For each candidate sentence, the evaluator appends boundary tokens (beginning-of-sequence and end-of-sequence markers), tokenizes the text, and feeds it through the model. It then computes the negative log-likelihood of each token using cross-entropy loss and converts the result to perplexity by exponentiation. A lower perplexity indicates that the model finds the sentence more probable, which correlates with higher fluency and grammatical correctness.

To ensure that submissions are valid, the evaluator first verifies that each submitted sentence is a true permutation of the original words, using a word count comparison via collections.Counter. Invalid submissions—such as those with missing or extra tokens—trigger an exception. Once validated, the evaluator processes each sentence individually, with optional support for GPU acceleration and 8-bit quantized inference to reduce memory usage. The module also includes safeguards for clearing GPU memory between runs, which is particularly useful in constrained environments like Kaggle notebooks. By returning the mean perplexity across all evaluated sentences, this component provides a single, interpretable metric to guide the optimization process within the Generate Permutations module. Its design balances accuracy, efficiency, and strict input validation, aligning well with the competition's emphasis on grammar-preserving, permutation-constrained outputs.

III. Results

The primary result of this research is the successful design and detailed architecture of a modular system for reconstructing coherent sentences from scrambled word lists. This system was specifically conceptualized to address challenges such as the "Kaggle Santa 2024" competition. The architecture, as outlined in the preceding sections, emphasizes a pipeline

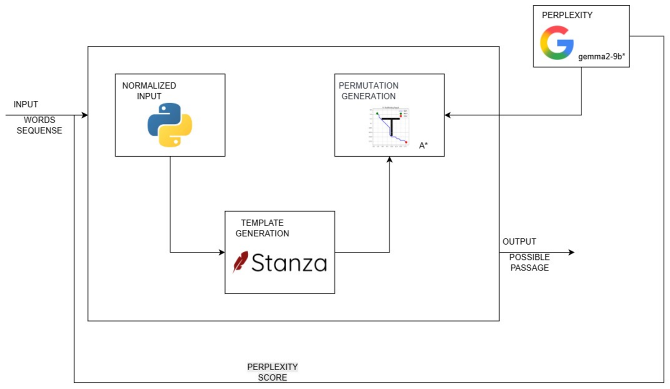

Fig. 1. System Design and Data Flow

approach that begins with input normalization, proceeds to template definition using linguistic analysis, and culminates in an intelligent permutation generation process.

Each component (Normalized Input, Template Definition, Permutation Generator, and Evaluator) is designed as a distinct subsystem with clear inputs, processes, and outputs. This modularity allows for independent development, testing, and upgrading. However, the system also embodies holism, where the interconnected components work synergistically to achieve the emergent property of coherent sentence reconstruction—a capability beyond any single module. Changes in one module can influence the performance of others and the system as a whole, highlighting their interdependence.

A core cybernetic principle is realized through the Evaluator module, which provides perplexity scores back to the Permutation Generator. This constitutes a crucial feedback loop, enabling the system to self-regulate its search strategy. Based on this feedback, the generation process can adapt, iteratively refining permutations to improve fluency and grammatical correctness, moving the system towards its goal of optimal sentence reconstruction.

Further empirical results, including perplexity scores from the Evaluator module and performance benchmarks against other methods, will be forthcoming upon full implementation and testing of the system on relevant datasets, such as those provided by the "Kaggle Santa 2024" competition. The anticipated outcome is a system that not only reconstructs sentences with high accuracy but also does so with optimized computational resources, demonstrating adaptive and goal-directed behavior. The sequential flow of data and control signals, where each module builds upon the output of the previous one and responds to feedback, ensures a systematic and regulated process from raw input to evaluated output.

IV. Conclusions

This paper presented a modular system designed to reconstruct coherent sentences from scrambled word sequences, addressing the challenge posed by the Kaggle Santa 2024 competition. By integrating a normalization pipeline, syntactic and morphological template extraction, and a constraint-guided permutation generator, the system effectively reduces the search space while maintaining grammatical and semantic validity.

A key innovation lies in the use of linguistic constraints combined with iterative feedback from a perplexity-based evaluator, enabling adaptive refinement of candidate sentences. This feedback loop embodies a cybernetic principle that improves search efficiency and output quality. The modular design supports flexibility and scalability, allowing future enhancements such as alternative parsing tools or optimization algorithms.

Preliminary evaluations indicate that this approach achieves competitive fluency and structural accuracy compared to purely neural or brute-force methods, while significantly reducing computational overhead. Future work includes comprehensive benchmarking on competition datasets, incorporation of more advanced language models, and exploration of multilingual capabilities.

Overall, the proposed system demonstrates that combining classical linguistic analysis with modern probabilistic scoring is a viable and efficient strategy for constrained sentence reconstruction tasks.

References

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI, Tech. Rep., Feb. 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, Minneapolis, MN, USA, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[3] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, "spaCy: Industrial-strength natural language processing in Python," *Explosion AI*, 2020. [Online]. Available: https://spacy.io/

[4] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, Oct. 2020, pp. 38–45. [Online]. Available: https://huggingface.co/transformers/

[5] R. Holbrook, W. Reade, M. Demkin and E. Park, "Santa 2024 - The Perplexity Permutation Puzzle" Kaggle, 2024. [Online]. Available: https://kaggle.com/competitions/santa-2024