POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA CIVILE,
AMBIENTALE E TERRITORIALE

Truss Structure Optimization with Large Language Models

Tesi di Laurea Magistrale in
Civil Engineering - Ingegneria Civile

Author: **Luigi Altieri**

Student ID: 252022
Advisor: Prof. Stefano Mariani
Co-advisors: Dr. Matteo Torzoni
Academic Year: 2024-25

Abstract

Artificial Intelligence, and in particular Large Language Models (LLMs), have demonstrated remarkable capabilities in modeling complex sequential data through the learning of long-range dependencies. While originally developed for Natural Language Processing, their underlying mechanism is inherently domain-agnostic. This observation opens the possibility of extending LLMs to engineering tasks.

This thesis investigates the application of a GPT-based framework to the topology optimization of planar truss structures. The optimal design problem is reformulated as a sequential construction task, in which admissible actions are regulated by predefined grammar rules. Sequences of generative actions are encoded symbolically and mapped into text strings. Each truss configuration is thus represented as a token sequence, enabling the use of a pretrained GPT-2 model fine-tuned on a dataset of structurally valid designs. To incorporate mechanical performance into the learning process, a mechanically weighted loss function is introduced, biasing the model toward configurations with optimal structural response.

The proposed approach is validated across six distinct configurations, achieving performance levels between 82% and 100% of the global optimum, also generating novel topologies not observed during training. These results suggest that, when properly adapted, LLMs can serve as complementary tools for structural optimization.

Keywords: Large Language Models, Structural Optimization, Truss Synthesis, Grammar-Based Design, GPT, Fine-Tuning, Mechanically Weighted Loss.

Abstract in lingua italiana

I Large Language Models (LLM) hanno dimostrato notevoli capacità nella modellazione di dati sequenziali attraverso l'apprendimento di dipendenze a lungo raggio. Sebbene siano stati sviluppati per l'elaborazione del linguaggio naturale, il loro meccanismo di funzionamento è indipendente dal dominio applicativo e può essere adattato a problemi ingegneristici.

Questa tesi analizza l'applicazione di un GPT per l'ottimizzazione topologica di strutture reticolari. Il problema di progettazione viene riformulato come una procedura costruttiva sequenziale in cui le azioni ammissibili sono regolate da un insieme di regole grammaticali predefinite. Tali azioni vengono codificate simbolicamente e rappresentate come stringhe di testo. Ogni configurazione reticolare è quindi descritta come una sequenza di token, consentendo l'utilizzo di un modello GPT-2 preaddestrato e successivamente sottoposto a fine-tuning su un dataset di strutture meccanicamente ammissibili. Per integrare le prestazioni meccaniche nel processo di apprendimento, viene introdotta una funzione di perdita pesata meccanicamente, che orienta il modello verso la generazione di strutture caratterizzate da una risposta strutturale ottimale.

L'approccio proposto viene validato su sei differenti configurazioni, raggiungendo livelli prestazionali compresi tra l'82% e il 100% dell'ottimo globale e generando configurazioni nuove rispetto a quelle presenti nel dataset di addestramento. I risultati suggeriscono che, se opportunamente adattati, i LLM possano costituire strumenti complementari per l'ottimizzazione strutturale.

Parole chiave: Large Language Models, Ottimizzazione Strutturale, Sintesi di Strutture Reticolari, Progettazione Basata su Regole Grammaticali, GPT, Fine-Tuning, Funzione di Loss Ponderata Meccanicamente.

Ringraziamenti

Ringrazio i miei genitori per i sacrifici fatti in questi anni e per i valori che mi hanno trasmesso con il loro esempio. I loro sforzi hanno reso possibile il raggiungimento di questo obiettivo. Ringrazio Marisa per la stima e il bene che mi ha sempre dimostrato. Ringrazio la mia famiglia per non avermi mai fatto sentire solo. Ringrazio Chiara per i momenti belli passati insieme e per il sostegno in quelli difficili.

Desidero esprimere la mia gratitudine al Prof. Stefano Mariani per avermi messo a disposizione tutti gli strumenti necessari allo svolgimento di questa tesi e per l'estrema umanità e professionalità dimostrate nei miei confronti in molteplici occasioni. Ringrazio il Dr. Matteo Torzoni per avermi messo a disposizione il suo talento e la sua esperienza. Senza la loro guida questa tesi non sarebbe stata realizzabile.

Infine, ringrazio le persone che mi hanno permesso di godermi gli anni universitari, i miei amici, il GSO Calcio e i miei compagni di corso. Con tutti voi ho condiviso sfide e traguardi che non dimenticherò.

Contents

1 | Introduction

Artificial Intelligence (AI) has rapidly emerged as one of the most promising paradigms across scientific and engineering disciplines. Among the various models developed in recent years, Large Language Models (LLMs) have attracted particular attention due to their ability to learn complex patterns and capture long-range dependencies within sequential data, enabling them to generate coherent sequences by predicting, step by step, the most likely subsequent token.

In the context of Natural Language Processing (NLP), tokens typically correspond to words, subwords, or characters. By adopting a more creative perspective, this notion can be extended beyond natural language, enabling physical or engineering problems to be represented as sequences of discrete building blocks. In this light, LLMs can be adapted beyond linguistic tasks by leveraging learned statistical relationships that are independent of the semantic meaning of individual tokens. This intuition motivates the use of Generative Pretrained Transformers (GPTs), as their ability to capture long-range dependencies, i.e., the capacity of distant tokens to influence the prediction of the next one, is learned during pretraining and remains effective when applied to semantically different tasks.

This thesis explores the application of LLM-based techniques to the structural optimization of planar truss systems. The central idea is to address the optimization problem through a sequential process governed by a set of grammar rules. Starting from a seed configuration, the structure is progressively generated through a sequence of actions. Each action is characterized by a set of information, identifying a structural element, a candidate node to be added within a predefined two-dimensional grid, and a particular operator defining the action effect, as discussed later in this thesis. These grammar rules are designed to ensure that, at every construction step, the resulting structure remains kinematically and statically admissible.

By encoding each action with text symbols, the entire construction sequence of a truss structure can be represented as a string of characters. As a result, every structure is mapped to a text sequence that an LLM can process. The LLM is trained on a dataset

of structurally valid trusses, generated in textual form through random sampling of admissible actions.

In this study, a GPT-2 architecture is adopted, as its size proved sufficient for the problems considered, making larger models with a greater number of parameters unnecessary. The rationale behind the choice of a pretrained model lies in its ability to capture long-range dependencies learned during large-scale pretraining. This capability allows the model to be effectively adapted through fine-tuning to tasks beyond NLP. As a result, fine-tuning can be performed on a relatively small task-specific dataset while still leveraging the sequential modeling abilities acquired during pretraining.

Since standard LLMs are trained using a purely grammatical loss function and do not possess any inherent understanding of physical laws, a training strategy that differs from the standard GPT-2 procedure is introduced. Specifically, the loss function is adapted to bias the learning process toward mechanically efficient solutions by assigning higher weights to structures that exhibit lower maximum nodal displacement under prescribed loading conditions while penalizing less performant configurations. This approach enables the use of a conventional fine-tuning strategy, commonly used to adapt pretrained models to specific downstream tasks, while implicitly embedding structural performance information into the training process.

The proposed framework is evaluated on six distinct benchmark tasks with varying loading conditions and structural constraints. Once fine-tuned, the model demonstrates the ability to generate truss configurations with efficient mechanical performance, achieving design objectives between 82% and 100% of the globally optimal solution, while discovering novel configurations not present in the training dataset.

Beyond the specific application to truss optimization, this work contributes to a broader reflection on the use of LLMs in scientific and engineering problems. The results demonstrate that, when appropriately adapted, LLMs can be effectively employed to address tasks outside the domain of natural language. At the same time, a key factor behind the success of LLMs lies in their large-scale self-supervised pretraining on extensive text corpora. In engineering applications, however, no naturally occurring corpora are available, and domain-specific datasets must be explicitly constructed. This requirement entails a significant effort to generate a dataset.

Nevertheless, despite this limitation, the ability of LLMs to capture long-range dependencies, combined with the accessibility and maturity of their architectures, suggests that they remain promising tools for engineering applications. This work therefore positions LLMs not as a replacement for traditional methods but as a complementary paradigm

whose potential and limitations must be critically evaluated when extending their use beyond natural language.

The original contributions of this thesis include the formulation of a grammar-based symbolic representation for planar truss construction, the design of a GPT-driven autoregressive framework for topology synthesis, the introduction of a mechanically weighted training objective that embeds structural performance into the learning process, and a systematic validation across six benchmark tasks with distinct seed configurations. These contributions are developed and analyzed throughout the manuscript as follows. Chapter 2 presents the literature review, outlining the state of the art in optimal truss design and discussing recent developments in the application of LLMs to mechanical and engineering problems. Chapter 3 introduces the methodological framework adopted in this work, detailing the dataset construction process, the training strategy, and the inference procedure used to generate candidate structures. Chapter 4 reports and discusses the results, highlighting both the strengths and the limitations of the proposed approach. Finally, Chapter 5 provides concluding remarks and outlines possible directions for future developments.

2 | Literature Review

This chapter reviews the state of the art in AI approaches to truss structure optimization. The objective of this literature review is to establish the theoretical and methodological background necessary to motivate the proposed research framework by progressively narrowing the scope from general machine learning (ML) approaches to LLM.

The chapter first examines AI applications in materials science and mechanical systems, where complex configurations are modeled as sequences of elementary building blocks and processed with LLMs. It then moves on to truss structure optimization, where grammar-based structural synthesis similarly represents design as a sequence of admissible actions. This analogy motivates the application of LLMs to truss structure optimization.

Section 2.1 addresses the motivation for employing ML in mechanical and materials engineering problems. It introduces the fundamental concepts of AI and ML, with particular emphasis on methodological aspects, model selection strategies, and data processing techniques suitable for the application of ML approaches to physical systems.

Section 2.2 presents the fundamental architecture of LLMs. The learning strategies that enable these models to capture sequential dependencies and hierarchical structures are briefly discussed. Finally, it motivates the use of LLMs for structural problems by discussing how mechanical systems can be represented as sequences of elementary building blocks, making them well-suited to LLMs.

Finally, Section 2.3 provides a brief overview of both classical and emerging approaches to structural optimization. Particular attention is devoted to recent methodologies that formulate structural optimization as a sequence of actions aimed at progressively constructing optimal truss configurations. In this work, such action sequences are reinterpreted as sequences of elementary building blocks, forming the basis of the proposed framework.

2.1. Artificial Intelligence and Machine Learning in Mechanical and Materials Design

AI, particularly in the form of ML and deep learning (DL), has become an increasingly important tool in materials and structural engineering, allowing predictive capabilities to support the design stage [1]. As the structural complexity increases, the optimization of mechanical behavior often involves vast design spaces that are difficult to explore using conventional methods. In this context, ML models trained on large datasets to map from structure to mechanical properties offer efficient strategies for exploring large design spaces. The success of ML methods in design relies on high-quality data, appropriate preprocessing informed by domain knowledge, and the selection of suitable learning models [2].

2.1.1. Challenges in Mechanical and Materials Design

Materials play a central role in engineering applications, as their microstructure organization governs mechanical, thermal, electrical, and other functional properties [3]. Therefore, the design of advanced materials focuses on tailoring the composition and architecture to achieve targeted performance. In recent years, data-driven and AI-based approaches have increasingly been explored to accelerate this design process [4].

For example, synthetic composites can be systematically engineered by controlling both the properties of individual constituents and the overall composite structure [5]. Bio-inspired materials aim to replicate the multifunctionality of natural biomaterials, although their design is challenging due to the hierarchical and heterogeneous nature of the underlying structures [6]. In addition, metamaterials have attracted significant interest because of their unconventional properties enabled by advances in experimental fabrication and computer-aided optimization techniques. Architected materials, as a subclass of metamaterials, have demonstrated exceptional mechanical performances [7, 8]. However, the increasing compositional and topological complexity of these systems often results in extremely large design spaces, exceeding the capabilities of conventional design approaches.

In recent decades, AI has emerged as a promising framework to address these challenges [9]. In particular, ML enables the discovery of complex relationships between high-dimensional input data and relevant outputs. While early ML approaches relied on manually designed features, the emergence of deep learning has enabled the automatic extraction of hierarchical representations of underlying structural patterns from raw data using complex neural network (NN) architectures [10]. As a result, ML has achieved ma-

jor successes in fields such as computer vision, NLP, and autonomous systems [11]. These advances have motivated growing interest within the materials communities [12, 13]. More recently, numerous studies have explored ML applications across a wide range of materials and mechanical domains to predict properties, accelerate materials discovery, and enable inverse design strategies, including energy materials [14, 15], glasses [16], composites [17], polymers [18], bio-inspired materials [19], and continuum materials mechanics [20].

In this context, ML-based mechanical research typically relies on three key components: a well-structured materials dataset, an appropriate ML model capable of learning meaningful representations, and a clearly defined mechanical problem that benefits from data-driven approaches. Effective integration requires careful data preprocessing to construct appropriate numerical representations compatible with the selected ML model [2].

2.1.2. Summary of Machine Learning Models

ML methodologies can be broadly categorized into three main paradigms: supervised learning, unsupervised learning, and reinforcement learning (RL). Supervised learning focuses on learning a mapping between input variables and target outputs using labeled data, commonly referred to as ground truth. In contrast, unsupervised learning relies on unlabeled data and aims to uncover hidden patterns or intrinsic structures within a dataset. RL differs fundamentally from these approaches, as it is based on the interaction between an agent and its environment, rather than on labeled datasets. While supervised and unsupervised learning typically assess model performance by minimizing a loss or objective function, RL seeks to maximize a cumulative reward signal. A graphical overview of ML approaches is shown in Fig. 2.1.

An intermediate category, known as semi-supervised learning, combines aspects of both supervised and unsupervised learning by exploiting a limited amount of labeled data together with a larger pool of unlabeled data during training. Within the domain of mechanical design, supervised learning approaches remain the most widely adopted, as they are generally more mature and provide higher predictive accuracy compared to other learning paradigms. Owing to the rapid evolution of ML techniques, the methods discussed here do not constitute an exhaustive list, but rather represent a selection of approaches that are currently feasible and relevant for mechanical design applications [2].

A brief overview is now presented, progressing from the simplest machine learning techniques to the most advanced and accurate approaches. Linear regression is among the simplest techniques, aiming to establish linear relationships between input features and continuous outputs [21]. Variants such as the Least Absolute Shrinkage and Selection

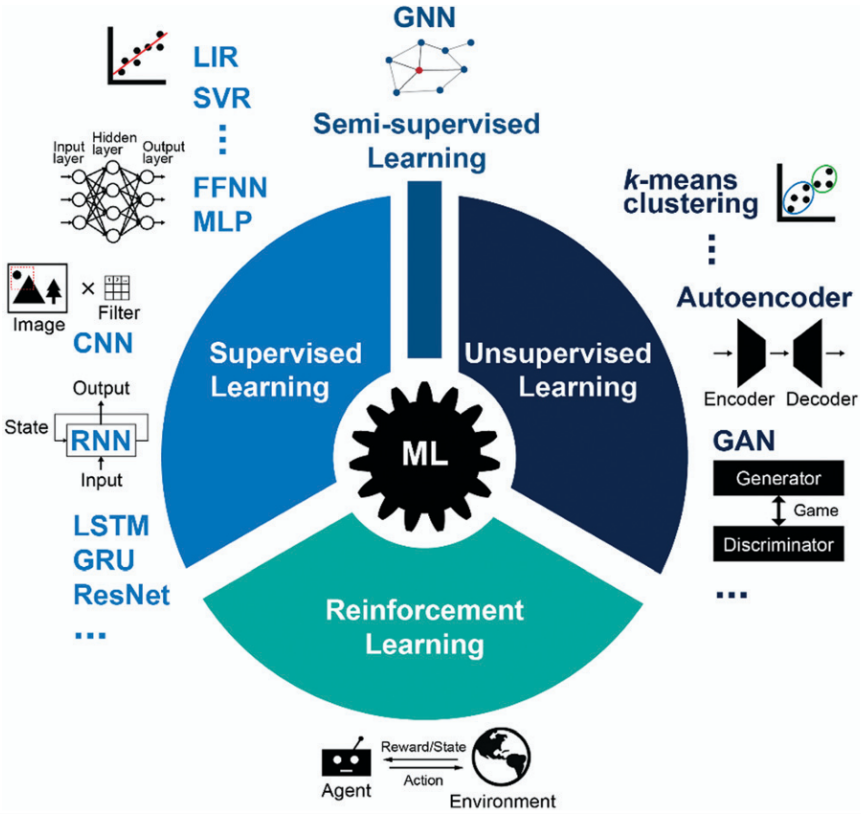

Figure 2.1: overview of ML approaches (adapted from [2]).

Operator introduce regularization through additional penalization terms in the loss function [22], while polynomial regression extends linear models by incorporating polynomial features [21]. To better capture nonlinear relationships, methods such as support vector regression and random forest have been developed, which generally offer improved robustness to outliers and higher predictive accuracy [23, 24].

In addition to regression tasks, classification represents another major category of ML applications. Unlike regression, classification algorithms assign inputs to predefined classes rather than predicting continuous values. Logistic regression [25] is a representative example, despite its name, as it employs a logistic loss function for classification purposes. Other widely used classical ML methods capable of handling both regression and classification problems include decision trees [26] and gradient boosting techniques [27].

Beyond classical approaches, artificial neural networks, inspired by the interconnected structure of biological neurons, have been developed to enable more advanced data-driven modeling. The foundational concept originates from the perceptron model proposed in 1958 [28]. By stacking multiple layers of neurons, NNs are able to learn complex non-linear relationships and detailed data distributions. As network depth increases, DL models have

demonstrated remarkable success across computer science and interdisciplinary research fields.

Feedforward neural networks, also referred to as multilayer perceptrons, represent the most basic class of DL architectures. In these models, information propagates in a unidirectional manner from the input layer to the output layer through successive hidden layers [29, 30]. Each neuron computes weighted combinations of its inputs, and the associated trainable parameters are optimized by minimizing a loss function. Training is typically performed using backpropagation combined with gradient descent, where gradients guide parameter updates until convergence is achieved [31].

Among DL architectures, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have received particular attention due to their success in computer vision and NLP. CNNs progressively extract hierarchical features, ranging from simple patterns to complex features [32]. In materials science, CNNs are especially well suited to describing materials with inherent hierarchical structures, such as biomaterials [4].

RNNs, on the other hand, are tailored to handle sequential data by accounting for dependencies between elements in a data stream. Unlike CNNs, which do not explicitly model sequential dependencies due to the absence of an internal memory state, RNNs incorporate memory mechanisms that allow outputs to depend on previous inputs. However, training deep RNNs can suffer from vanishing or exploding gradients [33, 34]. To mitigate these issues, advanced architectures such as Long Short-Term Memory [35], Gated Recurrent Units [36], residual networks [37], and attention mechanisms [38] have been introduced, significantly enhancing RNN performance in applications including language translation and speech processing. In parallel, graph neural networks (GNNs) have emerged as powerful tools for handling non-Euclidean data represented as graphs [39]. Recent developments, including graph convolutional networks [40], have demonstrated strong performance in learning graph embeddings, making GNNs particularly promising for mechanical applications that naturally exhibit graph representations.

Generative models constitute another important class of ML techniques, aiming to generate new data samples that follow the distribution of existing datasets. Among these, generative adversarial networks (GANs) [41] have gained particular prominence. GANs consist of two competing NNs: a generator that produces synthetic data and a discriminator that distinguishes between real and generated samples. Training proceeds through a competitive process until a Nash equilibrium is reached. In this context, a Nash equilibrium is a state where neither NN can benefit from changing strategy. Conditional GANs extend this framework by incorporating labels or constraints, enabling controlled data

generation and applications such as image-to-image translation [42, 43]. An alternative generative framework is provided by variational autoencoders [44], which encode input data into a latent representation and subsequently decode it to reconstruct the original data.

Finally, RL focuses on learning optimal decision-making strategies through interaction with an environment, aiming to maximize long-term rewards that quantify the overall performance of a sequence of actions [45]. A key aspect of RL training is the balance between exploration and exploitation, which regulates the trade-off between discovering new strategies and refining previously identified high-performing policies. The success of RL-based systems has highlighted the potential of these methods for applications in materials and structural design.

2.1.3. Data Preparation for Machine Learning Design

Data plays a fundamental role in machine learning. A sufficient amount of data is a basic requirement for training effective models, whereas high-quality data are essential to ensure accurate performance. This naturally raises several critical questions, including how much data is sufficient, how such data can be obtained, how its quality can be evaluated, and how it can be improved. Data can be collected from existing literature and databases or generated through experiments and numerical simulations. However, directly feeding raw data into ML models often presents challenges. When data are either trivially accessible or extremely difficult to obtain, the use of ML methods may be unnecessary or impractical. For instance, when the underlying problem can be efficiently addressed through analytical approaches, data-driven techniques may provide limited additional benefit. More commonly, available datasets cover only a small fraction of the design space or consist of representations, such as images or text, that are intuitive for human interpretation but unsuitable for machine processing. In these cases, data preprocessing is generally required before training ML models. For these reasons, data collection, generation, and preprocessing strategies need to be further explored [2]. The data preprocessing pipeline is summarized in Fig. 2.2.

First, data can be obtained from existing studies or generated through physics-based simulations. To enable the training of deep learning models, datasets have been created. Additionally, experimental datasets can combine information from existing literature [46]. The size of such datasets strongly depends on the amount of available literature in a given research field, although relatively small datasets comprising tens to hundreds of data points may still be effective for optimization tasks when coupled with active learning

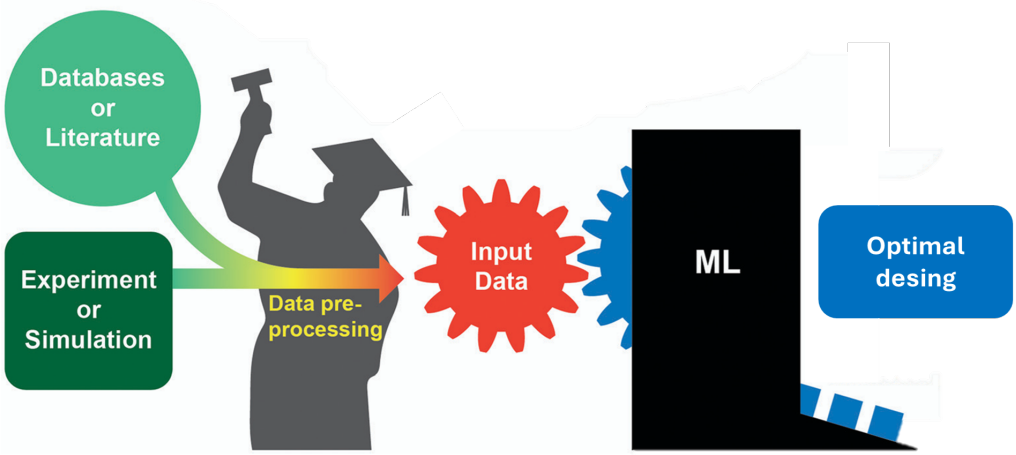

Figure 2.2: Schematic pipeline for data using ML (adapted from [2]).

strategies [47].

Beyond manual data collection, text processing techniques have been increasingly adopted to automate feature extraction from scientific publications. Using NLP methods, automated workflows for article retrieval, text extraction, and database construction have been developed, enabling the compilation of large-scale datasets. For example, [48] reports the construction of large-scale materials datasets obtained by training text-mining algorithms on hundreds of thousands of scientific articles.

Data generation through experiments or simulations offers greater flexibility in controlling dataset size, feature selection, and data distribution. However, a key challenge lies in balancing the cost of data generation with the resulting improvement in ML model performance.

Before training machine learning models, datasets must be transformed to comply with the required input representations, such as numerical vectors, images, graphs, or sequential data. During this preprocessing stage, data augmentation strategies may be employed to expand limited datasets, while irrelevant, redundant, or noisy information that could negatively affect model performance should be filtered out [2]. For instance, in the prediction of fracture patterns in brittle materials, atomic configurations generated through molecular dynamics simulations have been converted into structured pixel-based images, retaining the spatial characteristics of cracks while discarding other not relevant details [49]. Image processing techniques have likewise been applied to enhance datasets that are otherwise insufficient in size for effective deep learning training [50]. In an analogous manner, data can be generated using a FEM framework [51]. In every ML application, preprocessing remains a crucial point, especially when handling sparse or noisy datasets.

2.2. Large Language Model Architecture

The development of LLMs is the result of a sequence of technological turning points, each marking a shift in how language is represented, learned, and generated by machines.

A first decisive transition occurred in the early 2010s with the introduction of word embeddings, which demonstrated how semantic and syntactic regularities could be captured in continuous vector spaces. This enabled learning directly from raw text to become the dominant paradigm [52].

A fundamental rupture with previous architectures occurred in 2017 with the publication of the paper *"Attention Is All You Need"* by the Google research team [38]. This paper represents a major milestone in the field of LLM and introduces the Transformer architecture, which relies on attention mechanisms to model dependencies across entire sequences. This innovation dramatically improved training efficiency and scalability, making it feasible to train models on unprecedented amounts of data. In retrospect, this paper represents the architectural foundation upon which modern LLMs are built [38].

A further milestone was reached in 2020 with the release of GPT-3 by the OpenAI research team and the publication of the paper *"Language Models are Few-Shot Learners"* [53]. This work showed that, beyond a certain scale, LLMs exhibit emergent abilities, such as few-shot and zero-shot learning, namely the ability to generalize to previously unseen tasks from only a few examples provided in the input or even from task instructions alone, without explicit task-specific training. This finding shifted the research focus from architectural novelty to the systematic scaling of model size, training data, and computational resources. Thanks to these capabilities, LLMs can be adapted to new tasks that were not explicitly encountered during pretraining by further optimizing the model on a small task-specific dataset, a process known as fine-tuning.

This section therefore, aims to further investigate the architectures and processes introduced above, providing a deeper insight into the internal functioning of LLMs.

2.2.1. Token Embeddings

LLMs operate on discrete linguistic units, referred to as tokens, and are trained to predict the next token given a sequence of preceding ones. This formulation is particularly well suited to natural language, which can be decomposed into words, subwords, or character-level fragments forming a finite vocabulary of tokens [54]. Since NNs operate in continuous spaces, these discrete symbols must be mapped to numerical representations before being processed.

To this end, each token in the vocabulary is associated with a dense vector representation, referred to as a "token embedding", which represents the first transformation applied to the input sequence. Formally, given a tokenized sequence $\{x_t\}_{t=1}^{S}$ of length S, with $x_t \in \mathcal{V}$, where $\mathcal{V}$ denotes a finite vocabulary, the embedding layer maps each token x_t to a vector $\mathbf{e}_t \in \mathbb{R}^{d_{\text{model}}}$, where d_{model} defines the dimensionality of the embedding space.

Empirical studies have shown that embedding spaces encode meaningful semantic and syntactic information: tokens with similar meanings tend to be located close to each other in a multi-dimensional space, while more distant points correspond to semantically unrelated concepts. Moreover, certain linguistic relationships can be captured through linear transformations in the embedding space, as illustrated by the well-known analogy

$$king - man + woman \approx queen.$$

Therefore, in the embedding vector space, the operation $king - man$ can be interpreted as encoding the concept of a ruler, which, when linearly combined with the vector *woman*, results in a vector that lies very close to the embedding of the word *queen* [52].

Subword tokenization techniques, such as Byte Pair Encoding (BPE), allow text to be decomposed into smaller units that can be recombined to form words, while maintaining a fixed vocabulary size [54]. Consequently, the resulting tokens may represent either complete words or subword units, each of which is mapped to a distinct embedding vector.

2.2.2. Transformer Architecture

The Transformer architecture, originally introduced in *"Attention Is All You Need"* [38], is illustrated in Figure 2.3. It represents a sequence-to-sequence model based entirely on attention mechanisms.

Sequence Modeling Formulation Let $\mathbf{x} = \{x_t\}_{t=1}^{S_x}$ be an input sequence of length S_x, and $\mathbf{y} = \{y_t\}_{t=1}^{S_y}$ an output sequence of length S_y, with $x_t, y_t \in \mathcal{V}$, where $\mathcal{V}$ denotes a finite vocabulary.

The Transformer models the conditional probability

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{S_y} p(y_t \mid y_{<t}, \mathbf{x}), \tag{2.1}$$

where $y_{<t} = (y_1, \ldots, y_{t-1})$. Equation (2.1) highlights the autoregressive nature of the

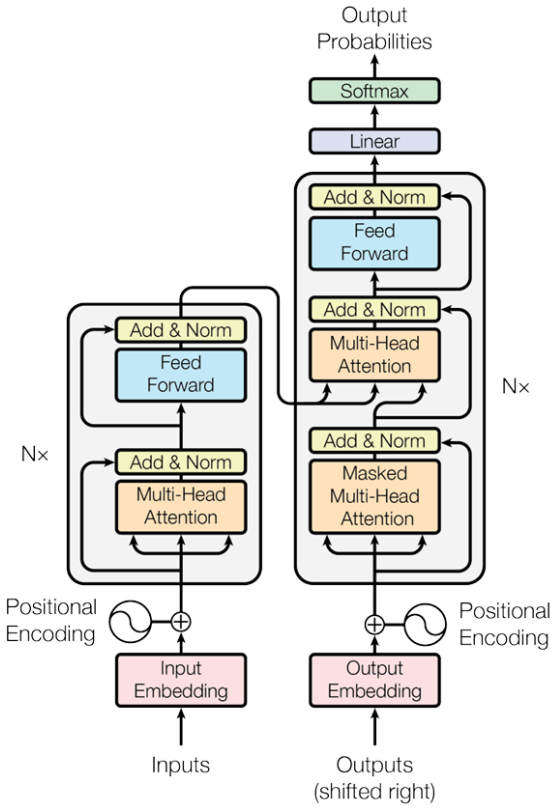

Figure 2.3: Transformer architecture (adapted from [38]).

decoder, which predicts each token conditioned on the previously generated outputs and on the encoded input sequence.

Embedding and Encoder Stack Each token is mapped to a continuous vector through a learned embedding:

$$x_t \mapsto \mathbf{e}_t \in \mathbb{R}^{d_{\text{model}}}, \tag{2.2}$$

where d_{model} denotes the embedding dimension. Collecting all embeddings yields

$$\mathbf{E} \in \mathbb{R}^{S_x \times d_{\text{model}}}. \tag{2.3}$$

Positional encodings of the same dimension are added to $\mathbf{E}$ to inject information about token order. The encoder stack consists of N identical layers, each composed of:

- Multi-head self-attention,
- Position-wise feed-forward network (FFN).

The encoder produces a sequence of latent representations

$$\mathbf{Z} = \text{Encoder}(\mathbf{E}), \qquad \mathbf{Z} \in \mathbb{R}^{S_x \times d_{\text{model}}}. \tag{2.4}$$

Each sub-layer is wrapped with residual connections and layer normalization, as shown in Figure 2.3.

Decoder Stack The decoder receives as input a shifted-right version of the target sequence, embedded into

$$\mathbf{Y}_{\text{in}} \in \mathbb{R}^{S_y \times d_{\text{model}}}. \tag{2.5}$$

Each decoder layer contains three sub-layers:

- Masked multi-head self-attention,
- Encoder-decoder (cross) attention,
- Position-wise feed-forward network.

The masking mechanism ensures that, at decoding step t, the representation depends only on $y_{<t}$, thereby enforcing causality.

The decoder produces hidden representations

$$\mathbf{H} = \text{Decoder}(\mathbf{Y}_{\text{in}}, \mathbf{Z}), \qquad \mathbf{H} \in \mathbb{R}^{S_y \times d_{\text{model}}}.$$

Let $\mathbf{h}_t \in \mathbb{R}^{d_{\text{model}}}$ denote the t-th row of $\mathbf{H}$, and let $\mathbf{W}^U \in \mathbb{R}^{d_{\text{model}} \times |\mathcal{V}|}$ be the learned output projection matrix (unembedding matrix) that maps latent representations to the vocabulary space. A linear projection maps $\mathbf{h}_t$ to the vocabulary space:

$$\mathbf{o}_t = \mathbf{h}_t \mathbf{W}^U, \tag{2.6}$$

where $\mathbf{o}_t \in \mathbb{R}^{|\mathcal{V}|}$ contains the unnormalized logits for all vocabulary tokens.

The conditional probability distribution over the next token is then obtained via

$$p(y_t \mid y_{<t}, \mathbf{x}) = \text{softmax}(\mathbf{o}_t), \tag{2.7}$$

where the softmax function maps the logits to a probability vector, ensuring non-negativity and unit sum.

Attention Mechanism The attention mechanism enables each position in a sequence to interact with all other positions through learned projections.

Let

$$\mathbf{X} \in \mathbb{R}^{S \times d_{\text{model}}}$$

denote a sequence of S input representations (e.g., encoder outputs or decoder hidden states).

Three learned linear projections generate queries, keys, and values:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{2.8}$$

where

$$\mathbf{W}^Q, \mathbf{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \qquad \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

are learned parameter matrices. Consequently,

$$\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{S \times d_k}, \qquad \mathbf{V} \in \mathbb{R}^{S \times d_v}.$$

Scaled dot-product attention is defined as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \tag{2.9}$$

where the matrix $\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{S \times S}$ contains pairwise similarity scores between sequence positions. The softmax operation is applied row-wise, producing attention weights that sum to one for each query.

In self-attention, $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are computed from the same input $\mathbf{X}$. In cross-attention, queries are derived from the decoder, while keys and values originate from encoder outputs.

Multi-Head Attention To enhance representational capacity, attention is computed in parallel across h heads. For $i = 1, \ldots, h$:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \tag{2.10}$$

where

$$\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}.$$

The outputs are concatenated and projected:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O, \tag{2.11}$$

with

$$\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}.$$

Figure 2.4 illustrates both scaled dot-product and multi-head attention mechanisms.

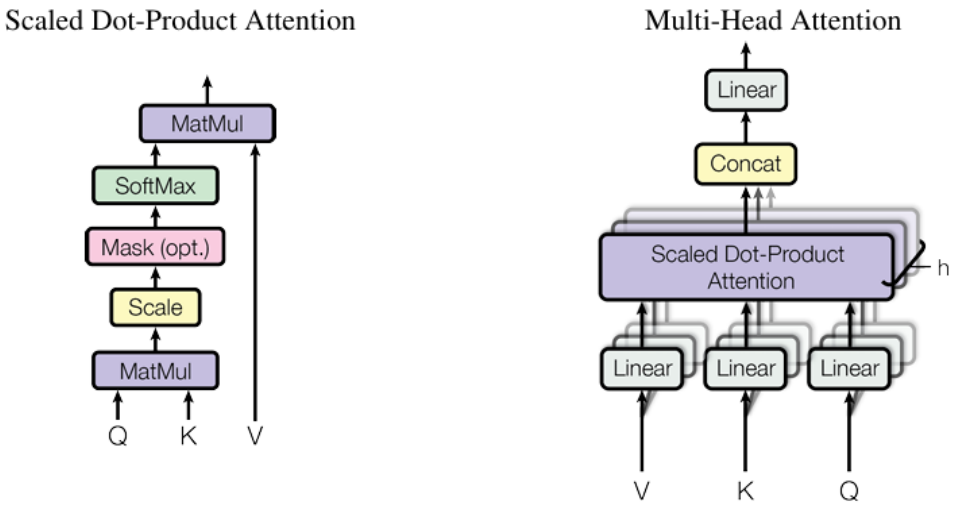

Figure 2.4: Scaled Dot-Product Attention and Multi-Head Attention (adapted from [38]).

Adopted Architecture: GPT-2 In this thesis, a GPT architecture is adopted, specifically the GPT-2 *base* model introduced in [55]. GPT models implement a decoder-only Transformer architecture, consisting of masked multi-head self-attention layers followed by position-wise feed-forward networks, with residual connections and layer normalization.

The adopted GPT-2 *base* configuration is characterized by:

- $N = 12$ decoder layers,
- $d_{\text{model}} = 768$,
- $h = 12$ attention heads,
- $d_k = d_v = 64$,
- approximately 124 million trainable parameters.

GPT-2 *base* was selected due to its moderate size among available GPT variants. It represents one of the smallest pretrained models suitable for fine-tuning, while remaining adequate for the structural generation task addressed in this thesis.

2.2.3. Pre-training and Fine-tuning

LLMs such as GPT (Generative Pretrained Transformer) are designed according to a two-stage learning paradigm consisting of pretraining followed by fine-tuning [53]. During the pretraining phase, the model is trained on extremely large text corpora, typically collected from publicly available sources. This process enables the model to learn rich statistical representations of sequential data, including syntactic patterns, semantic structures, and long-range dependencies between tokens [38]. As a result, the pretrained model already possesses the capability to model complex relationships within sequences, independently of any specific downstream task. In practice, this corresponds to the publicly available version of the model, which can be directly used without requiring training from scratch.

For a given application, the model is then adapted through fine-tuning. Fine-tuning consists of adapting a pretrained model by updating its parameters using a labeled dataset tailored to a specific task [53]. This approach generally leads to strong performance across a variety of evaluation benchmarks; however, it requires the availability of a task-specific dataset for each new application. Nevertheless, fine-tuning requires significantly fewer labeled samples than training a model from scratch, since the core representational capacity has already been learned during pretraining.

This approach has already been explored in physics applications. For instance, the pre-trained LLM *"Galactica"* [56] has been used to connect natural language with scientific tasks such as molecule classification and protein function prediction. This work demonstrates the feasibility of transferring GPT abilities to physical domains.

2.2.4. Mechanical Systems as Sequences of Elementary Building Blocks

A crucial conceptual step in extending LLMs to structural mechanics lies in looking at mechanical systems as structured symbolic sequences. This paradigm builds upon recent developments in materials science and computational mechanics, where physical systems are represented through tokenizable sequences [51]. Such a representation enables the direct use of transformer architectures and LLMs.

A clear and illustrative example of this approach is provided in the *"MeLM"* framework [51]. In that work, the authors consider honeycomb structures and employ a language modeling strategy to solve both forward and inverse mechanical problems. In the forward setting, a given structure is encoded by means of a so-called *"gene vector"*, consisting of ten integer values that describe the thickness of the honeycomb structure, as shown in Figure 2.5. This vector provides a compact representation of the structural configuration.

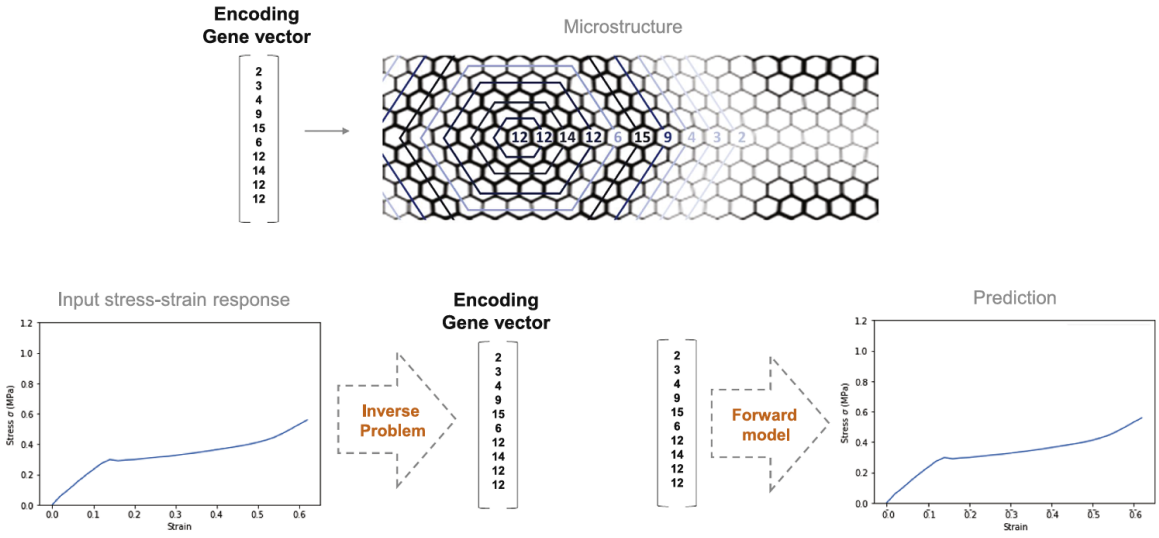

Figure 2.5: Top: Gene vector interpretation with respect to the microstructure. Bottom: representation of the Forward and Inverse problem (adapted from [51]).

The corresponding output is the mechanical response, expressed in terms of the stress–strain behavior. The stress–strain curve is discretized and represented as a vector. Each training example is therefore formatted as a textual sequence in which the input, enclosed within angle brackets $\langle\ \rangle$, corresponds to the gene vector, and the output, enclosed within square brackets $[\ ]$, corresponds to the associated stress–strain response, as illustrated in Figure 2.6. In this way, each data sample is treated as a string, fully compatible with the training procedure of an LLM.

GeneGetProperties< 12, 4, 7, 3, 0, 0, 0, 0, 0, 0> [−0.988,−0.913,−0.835,−0.760,−0.695,−0.629,−0.567,
−0.520,−0.517,−0.511,−0.503,−0.497,−0.489,−0.481,−0.475,−0.467,−0.461,−0.452,−0.445,−0.435,−0.430,−0.420,−0.410,-
−0.402,−0.393,−0.384,−0.374,−0.364,−0.352,−0.339,−0.320,−0.300]

Figure 2.6: Text string in the form <input>[output] (adapted from [51]).

During inference in the forward problem, the model receives as input a structural configuration expressed through the gene vector and is expected to generate the corresponding stress–strain response. The model was able to predict the required mechanical behavior with high accuracy. In particular, the predicted stress–strain curves showed strong agreement with reference solutions computed via Finite Element Method (FEM) simulations, demonstrating that the transformer architecture successfully learned the mapping between geometric descriptors and mechanical response.

In the inverse problem setting, the formulation is reversed. Given a target stress–strain behavior, the model is conditioned to generate a corresponding gene vector representing a structural configuration capable of reproducing the desired mechanical performance. Remarkably, the generated structures were not limited to those explicitly present in the training dataset. The model demonstrated the ability to propose novel configurations that replicated the prescribed mechanical behavior with high fidelity.

These results highlight the potential of using LLMs and motivate the use of the same approach in this work. The reformulation of a mechanical problem as a sequence of tokens is not automatic; rather, it requires the human intuition to identify the appropriate interpretative key through which physical problems can be represented as a succession of tokens.

The aforementioned contribution provides a relevant methodological inspiration to adapt LLMs to scientific domains. However, the problem addressed therein primarily concerns predictive regression tasks, whereas the present thesis aims to explore a combinatorial design setting. While their framework learns mappings between structured inputs and continuous outputs, the objective here is to investigate the generation of valid structural configurations through sequences of discrete construction actions. This conceptual analogy introduces the discussion of truss structure optimization in Section 2.3.

2.3. Truss Structure Optimization

Structural optimization – the process of identifying the most efficient structural configuration under specified loading conditions – has long been a fundamental challenge in structural engineering. Over the years, this problem has been addressed through increasingly sophisticated mathematical formulations and computational strategies, many of which are characterized by numerical complexity and high computational cost. With the advent of AI, research efforts have progressively shifted toward integrating ML methodologies with traditional optimization techniques, seeking more adaptive and scalable solutions [57].

A successful conceptual development has been the reformulation of structural optimization as a sequential decision-making process [58, 59]. Rather than determining the optimal configuration in a single global step, the problem is addressed through a sequence of discrete actions. While this reformulation may simplify the problem by reducing the effective search domain, it may also increase the risk of converging to sub-optimal solutions, particularly in inherently combinatorial settings where local decisions constrain the global search space [60].

Such a methodology naturally suggests a logical bridge toward token-based representations. If the construction of a reticular structure can be expressed as an ordered sequence of discrete operations, it becomes possible to interpret the corresponding generative process as a sequence of tokens encoding nodes, elements, and operators. This abstraction opens the possibility of leveraging LLM frameworks, whose core capability lies precisely in modeling and generating structured sequences, thereby extending their applicability beyond natural language to structured engineering design processes.

2.3.1. Evolution of Structural Optimization

Structural optimization has undergone a significant transformation over the past decades. Early approaches were primarily rooted in deterministic formulations, where the structure was described through continuous variables and optimized using mathematical programming techniques. Classical methods such as the ground structure approach [61] established a rigorous framework for topology optimization, allowing the identification of optimal layouts within a predefined set of candidate members. Despite their theoretical robustness, these methods often face practical limitations when the size of the design domain increases, due to computational complexity and numerical instability issues [58].

To overcome these challenges, alternative strategies were introduced. Genetic algorithms [62–64], particle swarm optimization [65, 66], differential evolution [67], and simulated

annealing [68] enabled the treatment of discrete design variables and reduced reliance on gradient-based information. However, as highlighted in both recent reinforcement learning studies and earlier synthesis frameworks, these methods can become inefficient when the combinatorial explosion of possible structural configurations dominates the search process [57].

A conceptual shift occurred when structural design began to be interpreted not simply as the optimization of a fixed topology, but as a design synthesis problem. In this perspective, the structure is not optimized all at once; instead, it is progressively constructed. Ororbia and Warn [59] formalized this idea by modeling design synthesis as a Markov Decision Process (MDP), where each structural configuration corresponds to a state and each admissible modification represents an action. This formulation reframes structural optimization as a sequential decision-making problem, in which the final topology emerges from a series of discrete, locally meaningful transformations.

Building upon this interpretation, RL techniques were introduced to learn optimal sequences of design actions [57, 59, 69]. More recently, Monte Carlo tree search has demonstrated the ability to efficiently explore extremely large state spaces by selectively expanding promising design trajectories and propagating performance information backward through the decision tree [57]. This evolution marks a transition toward sequential structural synthesis.

2.3.2. Progressive Generation of Truss Structures

For sequential learning frameworks to operate effectively, the structural problem must be expressed in a discrete and structured way. This requirement naturally leads to the adoption of a grid-based representation, where a finite set of predefined nodes defines the admissible design domain. In such a setting, each structural configuration corresponds to a specific subset of activated nodes and connecting elements, and can therefore be interpreted as a state within a finite MDP.

The process begins from a seed configuration, typically a sparse and mechanically admissible structure that satisfies the prescribed boundary conditions [57, 59]. From this initial state, the structure evolves step by step through discrete actions modeled after generative grammar rules. These grammar rules, originally introduced in computational design synthesis [70], define how a configuration can be modified. In the truss setting, actions generally involve selecting an inactive node of the grid, choosing an existing element, and applying a legal operator that connects the new node, replacing or not an existing element, while maintaining kinematic admissibility.

A crucial aspect of this grammar-based construction is that mechanical admissibility is embedded directly into the action space. Only transformations that preserve structural consistency, such as the generation of triangulated forms ensuring static and kinematic compatibility, are allowed. This prevents the generation of inadmissible intermediate configurations and drastically reduces the effective search space [59].

The progressive generation strategy is essential for integrating machine learning into structural optimization. First, it transforms topology optimization into a sequential decision-making process. Second, it ensures that intermediate configurations remain physically analyzable through finite element evaluation, allowing reward signals to be computed. Third, it introduces a strong inductive bias: instead of exploring arbitrary combinations of members, the algorithm navigates a constrained, grammar-driven design space that reflects engineering knowledge [57]. Under this formulation, the optimal truss is no longer selected from a predefined list of complete layouts. Rather, it is generated in a mechanically admissible space of actions over a grid.

This two-dimensional and grid-based formulation, structured as a sequence of steps and actions, motivates the adoption of LLMs. Even if the methods employed, which will be discussed in the next chapter, differ from those previously discussed.

2.4. Research Gaps and Thesis Outline

The reviewed literature highlights two parallel developments. On the one hand, the ability of LLM to model sequential dependencies has been extended to physical domains. On the other hand, structural optimization has progressively evolved toward sequential and grammar-based formulations. These trajectories suggest conceptual compatibility between the two research directions. Despite this alignment, a direct integration between LLMs and grammar truss synthesis remains unexplored. As a result, the potential of LLMs as sequence predictors for structured truss growth has not been investigated. The present work positions itself within this gap. Rather than proposing an alternative to established optimization methods, it explores whether a GPT can be adapted to generate truss configurations. The objective is not to replace established optimization methods, but to evaluate the feasibility and inherent limitations of the proposed approach.

The remainder of the thesis is structured as follows. Chapter 3 presents the methodological framework, including the formulation of the physical problem, the construction of the dataset, and the fine-tuning procedure based on a customized loss function designed to bias the model toward mechanically efficient structures. The chapter concludes with the inference strategy adopted for the practical use of the model.

Chapter 4 reports the results, analyzing both the generation strategy and the mechanical performance of the produced truss configurations across the six proposed tasks. Particular attention is devoted to cases in which the model fails to follow the globally optimal sequence of actions, instead favoring trajectories that lead to suboptimal solutions.

Finally, Chapter 5 discusses the main limitations of the approach, emphasizing its supervised and data-dependent nature, and examining whether the model has effectively captured a notion of optimization, rather than merely generating structures by sampling from a learned probability distribution.

3 | Methods

This chapter presents the methodological framework developed in this work, describing the full pipeline from the formulation of the physical optimization problem to the generation of new truss configurations through a fine-tuned LLM. The overall computational workflow of the proposed framework is illustrated in Fig. 3.1.

Section 3.1 introduces the optimal design problem for planar truss structures, formally defining the mechanical setting and the performance indicator adopted throughout the study. Section 3.2 then describes the generative grammar rules used to synthesize admissible truss configurations, interpreting structural design as a sequential process starting from a seed configuration.

Building upon this formulation, Section 3.3 details the dataset construction procedure. Random truss configurations are generated by sampling admissible construction steps, evaluated through finite element analysis, and encoded as text strings. Each configuration is associated with its structural performance. A performance-based weighting strategy is then introduced, so that each textual example in the dataset is paired with a corresponding weight reflecting its structural quality.

Section 3.4 describes how the GPT-2 architecture converts text strings into token sequences. Section 3.5 presents the fine-tuning procedure, including the treatment of token sequences during training, the definition of a customized loss function incorporating performance-based weights, and the subsequent parameter update process. Finally, Section 3.6 discusses the inference stage, where the trained model is used to generate new truss configurations.

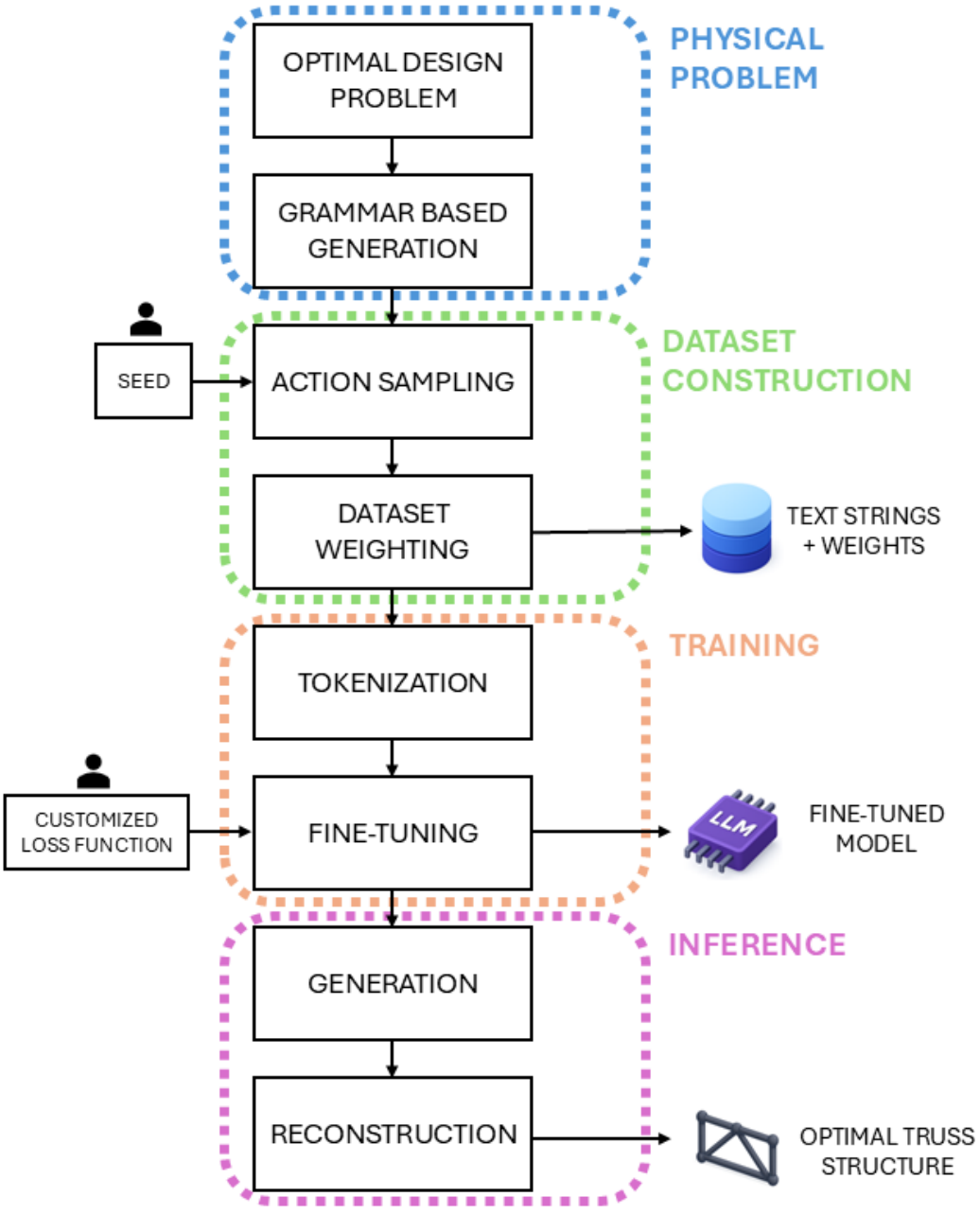

Figure 3.1: Computational pipeline of the proposed framework. The central blocks mirror the main sections of the Methods chapter and are organized into four macro-blocks: physical problem, dataset construction, training, and inference. On the left side, user-defined inputs are indicated, including the initial seed configuration and the customization of the loss function. On the right side, the outputs of each macro-block are highlighted: the dataset construction block produces a dataset composed of text strings with their associated weights; the training block outputs the fully fine-tuned model; and the inference block generates optimized truss configurations.

3.1. Optimal Design Problem for Truss Structures

The structural design task consists in determining the geometry of a truss structure that minimizes a prescribed performance measure under a given loading configuration. In this thesis, the objective is the minimization of the maximum absolute displacement attained within the structure. This choice enables direct comparison with previous contributions in the literature [57]. The mathematical formulation of the optimal truss design problem is also inspired by the same work.

To maintain generality, the design problem is formulated within the framework of continuum elasticity, with truss design being treated as a specific case of this broader context. In particular, a set of I subdomains $\{\Omega_1^s, \dots, \Omega_I^s\}$ is sought, each covering a portion of the design region, such that their union defines a candidate structural domain

$$\Omega = \bigcup_{i=1}^{I} \Omega_i^s. \tag{3.1}$$

The objective is to determine the configuration of these subdomains that minimizes the infinity norm of the displacement field:

$$\bar{\Omega} = \bigcup_{i=1}^{I} \bar{\Omega}_i^s = \arg \min_{\Omega = \bigcup_{i=1}^{I} \Omega_i^s} \|\mathbf{u}(\mathbf{x})\|_\infty, \qquad x \in \Omega. \tag{3.2}$$

where $\mathbf{u}$ denotes the displacement field and $\mathbf{x}$ represents the spatial coordinates. For a generic vector $\mathbf{a} \in \mathbb{R}^M$, the infinity norm is defined as

$$\|\mathbf{a}\|_\infty = \max_{m=1,\dots,M} |a_m|. \tag{3.3}$$

The optimization problem is constrained by the governing equations of static linear elasticity, which must hold over the structural domain:

$$\nabla \cdot \boldsymbol{\sigma} + \mathbf{b} = \mathbf{0}, \tag{3.4}$$

$$\boldsymbol{\sigma} = \mathbf{E}\,\boldsymbol{\varepsilon}, \tag{3.5}$$

$$\boldsymbol{\varepsilon} = \frac{\nabla \mathbf{u} + (\nabla \mathbf{u})^\top}{2}. \tag{3.6}$$

These equations enforce equilibrium, linear elastic constitutive behavior, and kinematic

compatibility, respectively. In this formulation, $\boldsymbol{\sigma}$ denotes the stress tensor, $\boldsymbol{\varepsilon}$ the strain tensor, $\mathbf{b}$ the body force vector, and $\mathbf{E}$ the elasticity tensor. The symbols $\nabla\cdot$ and ∇ indicate the divergence and gradient operators.

The mechanical problem must be completed with appropriate boundary conditions. Let the boundary of the domain be decomposed into a Dirichlet portion $\partial\Omega_g$ and a Neumann portion $\partial\Omega_h$, defined as

$$\partial\Omega_g = \bigcup_{i=1}^{I} \partial\Omega_{gi}^s, \tag{3.7}$$

$$\partial\Omega_h = \bigcup_{i=1}^{I} \partial\Omega_{hi}^s. \tag{3.8}$$

The boundary conditions read

$$\mathbf{u} = \mathbf{u}_g \quad \text{on } \partial\Omega_g, \tag{3.9}$$

$$\boldsymbol{\sigma}\cdot\mathbf{n} = \mathbf{f} \quad \text{on } \partial\Omega_h, \tag{3.10}$$

where $\mathbf{u}_g$ is the prescribed displacement field, $\mathbf{f}$ represents the applied surface tractions, and $\mathbf{n}$ is the outward unit normal vector. Although the present work adopts a linear constitutive law, the same framework can be extended to nonlinear material behavior, which remains a natural direction for future investigation.

The transition to planar truss systems is achieved by introducing a finite element discretization of the elasticity problem. In this discretized setting, the optimization problem becomes

$$\min_{\Omega=\bigcup_{i=1}^{I}\Omega_i^s} \|\mathbf{U}(\Omega)\|_\infty, \tag{3.11}$$

where $\mathbf{U}(\Omega)$ denotes the vector of nodal displacements obtained by solving the discretized equilibrium problem associated with configuration Ω.

The equilibrium equations reduce to the classical algebraic system

$$\mathbf{K}\,\mathbf{U} = \mathbf{F} \quad \text{in } \Omega = \bigcup_{i=1}^{I} \Omega_i^s, \tag{3.12}$$

supplemented by the essential boundary conditions

$$\mathbf{U} = \mathbf{U}_0 \quad \text{on } \partial\Omega_g = \bigcup_{i=1}^{I} \partial\Omega_{gi}^s. \tag{3.13}$$

A global volume constraint is also imposed:

$$V \leq V_{\max}, \qquad V = \sum_{i=1}^{I} A_i L_i. \tag{3.14}$$

In these expressions, $\mathbf{U}$ denotes the vector of nodal displacements, $\mathbf{K}$ the global stiffness matrix, and $\mathbf{F}$ the vector of external nodal forces. The vector $\mathbf{U}_0$ collects the prescribed nodal displacements. For each truss element $i = 1, \ldots, I$, A_i and L_i indicate the cross-sectional area and length, respectively. The scalar $V_{\max}$ defines the maximum admissible structural volume.

Non-dimensional formulation To ensure consistency with the benchmark settings adopted in [57, 59], the optimal truss design problem is formulated in dimensionless form. The member properties are prescribed in dimensionless terms, with Young's modulus set to $E = 10^3$ and cross-sectional area $A = 1$. The external loads are likewise defined with dimensionless components $f_x = f_y = 10$. Under this formulation, the structural response quantities, including the maximum displacement adopted as performance indicator, are dimensionless.

3.2. Generative Grammar Rules for Truss Structure Synthesis

The generation of admissible truss configurations is guided by a set of predefined grammar rules that regulate the synthesis process on a planar grid. The procedure starts from an initial seed configuration s_0, constructed by placing a limited number of bars to form a statically determinate truss.

From this starting point, the structure evolves through a sequence of discrete modifications. At each step t, an admissible action is applied to the current configuration s_t, producing a new structural state s_{t+1}. This iterative process continues until a terminal condition is met. In the present setting, the process terminates after k actions, while enforcing the global volume constraint $V \leq V_{\max}$ at each step.

The set of admissible actions is defined according to the same grammar formalism adopted in [57, 59]. Beginning from an isostatic seed structure, the grammar restricts the exploration of the design space by allowing only operations that preserve static determinacy. More precisely, new elements may be introduced exclusively in such a way that triangular substructures are formed. This restriction guarantees that each intermediate configuration remains statically determinate throughout the construction process. Therefore, the term admissible action refers to those construction steps that satisfy the aforementioned requirements.

Given a current structural configuration s_t, any admissible action is defined through a sequence of three ordered operations. First, a node is selected among those not yet connected to the existing truss members. These nodes are referred to as inactive nodes, in contrast to active nodes that already belong to the evolving structure. Second, one of the truss elements already present in the configuration is chosen. Third, a valid operator is applied, depending on the geometric relationship between the selected inactive node and the chosen element.

Two operators are permitted within this framework, denoted as $\mathcal{D}$ and $\mathcal{T}$. The $\mathcal{D}$ operator introduces the new node and connects it to the current structure without removing any existing element. In contrast, the $\mathcal{T}$ operator first removes the selected element and then connects the new node to the structure. In both cases, the new connections are generated in such a way that no geometric intersection with existing truss members occurs. This condition ensures geometric consistency of the resulting configuration. A visual representation of operators $\mathcal{D}$ and $\mathcal{T}$ is shown in Fig. 3.2.

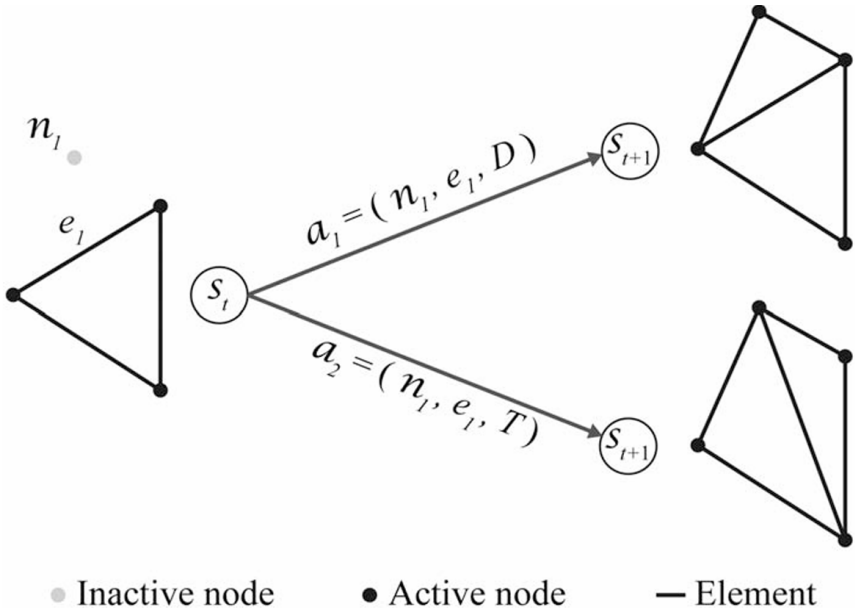

Figure 3.2: Visual representation of operators $\mathcal{D}$ and $\mathcal{T}$. The current structural state s_t (left) evolves either through action a_1 (top right), obtained by applying operator $\mathcal{D}$, or through action a_2 (bottom right), obtained by applying operator $\mathcal{T}$, leading in both cases to a new configuration s_{t+1}. In the two illustrated transitions, the selected truss member is e_1, and the inactive node introduced into the structure is n_1 (adapted from [57]).

The adoption of grammar rules represents a fundamental difference with respect to classical topology optimization approaches. Instead of exploring a continuous material distribution, the design space is explicitly constrained through mechanically admissible construction steps. This structured exploration introduces strong inductive biases that embed engineering knowledge directly into the synthesis process, significantly reducing the size of the feasible search space.

3.3. Dataset construction

This section describes in detail the construction of the dataset used to train the LLM.

The dataset is generated in two consecutive stages. First, starting from a predefined seed configuration, admissible truss structures are produced by randomly sampling the actions allowed by the grammar rules described in Section 3.2. Each generated configuration is subsequently evaluated through finite element analysis, and its structural efficiency is quantified in terms of the maximum displacement $U_{\max}$. At this stage, every structure is stored as a text string together with the corresponding performance indicator.

In a second stage, a weighting procedure is applied to each example. Specifically, a scalar weight in the range $[0, 1]$ is assigned based on the structural performance exhibited by the configuration, so that better-performing structures receive higher values. The final dataset, therefore, consists of pairs of textual strings and their associated weights, forming the basis for the subsequent performance-informed fine-tuning procedure that will be discussed in Section 3.5.

3.3.1. Generation of Training Samples

A dedicated code was developed to automatically generate truss structures starting from predefined seed configurations. The generation process systematically applies the previously introduced grammar rules by randomly sampling admissible actions, thus producing sequences of admissible structural configurations. Through iterative application of these actions, increasingly complex structural configurations are obtained. Each generated configuration is evaluated through the finite element formulation introduced in Eq. 3.12, from which the maximum displacement is computed as

$$U_{\max} = \|\mathbf{U}(\Omega)\|_{\infty}. \tag{3.15}$$

This scalar quantity is stored as the performance indicator associated with the generated configuration.

In parallel, the sequence of randomly generated actions is recorded in textual format. For each generated structure, the corresponding sequence of applied actions is encoded as a text string of the form:

$$T\langle id_{\text{task}}\rangle\langle k\rangle[element_i, node_i, operator_i] \tag{3.16}$$

where:

- id_{task} identifies the seed configuration;
- k is the number of applied actions (equivalently, the number of activated nodes);
- $element_i$ represents the i-th selected element, identified by its end nodes;
- $node_i$ denotes the selected node at step i;
- $operator_i$ indicates the operator applied at step i, $\mathcal{D}$ or $\mathcal{T}$,

for $i = 1, \ldots, k$.

Representing each truss configuration as a text sequence establishes a direct correspondence between structural design and sequential symbolic encoding, enabling its direct use for training the LLM.

The overall procedure adopted for the automatic generation of the dataset is summarized in Algorithm 3.1. The algorithm describes the sequential construction of truss configurations starting from a given seed structure, the random action generation, the application of geometrical admissibility checks, and the final evaluation of the structural performance indicator $U_{\max}$ through FEM analysis, before storing the resulting data in textual format.

Having introduced the overall objective of the algorithm 3.1, we now analyze it in detail by describing its main components in three paragraphs: the input, the internal code, and the final output.

Input The input of Algorithm 3.1 is defined by three main parameters: the number of examples to generate N_{examples}, the maximum number of node additions k, and the seed configuration (or task definition).

N_{examples} determines the number of iterations executed by the outer loop of the algorithm. It represents the number of candidate structures that are attempted during the generation process. However, it does not necessarily coincide with the final number of valid samples stored in the dataset. Since the action sequences are generated randomly, identical configurations may occasionally be produced. These duplicates are identified and discarded during the final uniqueness check of the algorithm, see Lines 18–22. As a consequence, the effective dataset size may be smaller than N_{examples}.

k defines the number of actions to perform and so the number of nodes that can be sequentially added to the seed configuration.

The seed configuration plays a central role. It specifies the grid definition through the

Algorithm 3.1 Dataset Generation Procedure

Input:

Seed configuration (or task definition)
Number of examples to generate N_{examples}
Number of node additions k

Initialize structure with seed configuration
for $n = 1$ to N_{examples} **do**
    for $i = 1$ to k **do**
        Sample candidate node $node_i$
        Sample candidate element $element_i$
        Sample operator $operator_i \in \{\mathcal{D}, \mathcal{T}\}$
        Perform geometrical checks:
          - No bar intersection
          - No collinearity
          - $V < V_{\max}$
          - $\text{cond}(\mathbf{K}) < 5 \times 10^4$
        Update structure if all checks are satisfied
    end for
    Solve $\mathbf{KU} = \mathbf{F}$ (FEM analysis)
    Extract $U_{\max}$
    Convert action sequence into textual string representation
    if structure already generated **then**
        Discard structure
    else
        Store textual sequence and $U_{\max}$ in dataset
    end if
end for

Output:

JSON file containing textual structural sequences and associated $U_{\max}$

parameters N, M, dx, and dy, which determine the number of nodes per row, the number of nodes per column, and the horizontal and vertical spacing between adjacent nodes, respectively. Nodes are indexed from 1 to $N \times M$ following a row-wise ordering, proceeding from left to right and starting from the lowest row before moving upward. In addition, the seed configuration defines the initial set of structural members through their start and end nodes, the prescribed supports, the applied loads, and the maximum admissible total volume $V_{\max}$. The seed configuration may also include an additional design constraint representing a pinned support that must be reached by the structure during the growth process. In that case, at least one of the nodes introduced during generation must coincide with this prescribed support location.

For the sake of clarity, a graphical example is now provided to illustrate the construction of a seed configuration. In particular, we consider the seed configuration used for the first task that will be discussed in the next chapter.

The considered seed configuration is characterized by the parameters reported in Table 3.1. A graphical representation of the corresponding seed configuration is shown in Figure 3.3.

Parameter	**Value**
N	3
M	4
dx	10
dy	10
$V_{\max}$	160
Initial elements	$(1-12), (12-3)$
Supports	1, 3
Load	Node 12: $F_x = 10$, $F_y = 0$
Additional constraint	None

Table 3.1: Parameters defining the illustrative seed configuration.

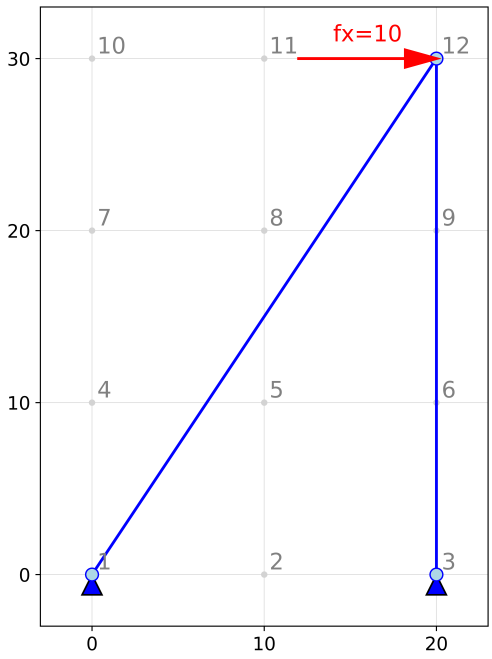

Figure 3.3: Graphical representation of the considered seed configuration.

Action sampling After initializing the seed configuration, the growth process is carried out through an inner loop of k steps. At each step, a candidate node is selected from the predefined grid, ranging from 1 to $N \times M$, excluding the nodes that are already active, i.e., those that already belong to the current structural configuration. An existing structural element is then selected from the current structure, and one of the admissible operators ($\mathcal{D}$ or $\mathcal{T}$) is sampled. These three components uniquely define the action applied at that iteration.

Before permanently updating the structure, the algorithm performs a series of admissibility checks to ensure geometric consistency, mechanical stability, and compliance with global constraints. First, the absence of bar intersections is verified, meaning that a newly generated element is rejected if it intersects any pre-existing structural member, except at shared end nodes. Second, a collinearity condition is checked. The sampled node cannot lie on an existing element unless it coincides with one of its end nodes. In other words, if an element connects node a to node b and geometrically passes through an intermediate node c, the node c cannot be selected for the insertion of a new element along that same segment. This condition avoids the creation of degenerate structural members. In addition, a global volume constraint is imposed by verifying that the total structural volume does not exceed the prescribed limit $V_{\max}$. This ensures compliance with the restriction defined in the task configuration. Finally, a mechanical consistency check is carried out by evaluating the conditioning of the global stiffness matrix $\mathbf{K}$ and enforcing

the requirement $\text{cond}(\mathbf{K}) < 5 \times 10^4$. Although structural instability formally occurs when $\mathbf{K}$ becomes singular, very large condition numbers correspond to near-mechanism states. In such cases, the displacement field becomes highly sensitive to numerical perturbations, potentially leading to unreliable results. The adopted threshold is intentionally selected with a relatively high value in order to ensure compatibility with any seed configuration. Its purpose is to discard configurations that are extremely close to a mechanism state, while still allowing moderately ill-conditioned configurations to pass this preliminary screening. These configurations are subsequently filtered out during the dataset weighting phase. This choice enables the use of a single condition number threshold that remains valid across all possible starting configurations.

Only if all these conditions are satisfied, the structure is updated, and the candidate action is accepted; otherwise, the proposed modification is discarded. Once the sequential growth process is completed, the resulting structure is analyzed through a finite element procedure, and $U_{\max}$ is extracted. Finally, the corresponding action sequence is converted into its text representation and stored together with $U_{\max}$, provided that the configuration is not already present in the dataset.

Output The output of the algorithm consists of a textual representation of the generated structural configuration together with its associated structural performance indicator. An illustrative example of a randomly generated configuration is reported below, in Figure B.6. The corresponding action sequence is:

$$T1\langle 2\rangle[\text{1-12}, 11, T;\ \text{1-11}, 7, D]$$

In this notation, `T1` indicates that the structure refers to Seed 1. $\langle 2\rangle$ indicates that $k = 2$ was sampled.

The first action, `1-12,11,T`, selects the element connecting nodes 1 and 12. Since the operator $\mathcal{T}$ is applied, the selected element is removed and node 11 is connected to the structure according to the grammar rules.

The second action, `1-11,7,D`, selects the element connecting nodes 1 and 11. In this case, operator $\mathcal{D}$ does not remove the selected element, and node 7 is connected to the structure while preserving the existing bar.

The associated performance indicator is $U_{\max} = 0.1681$.

It is important to emphasize that, at this stage of the dataset generation procedure, structures are produced through purely random sampling of admissible actions. Consequently,

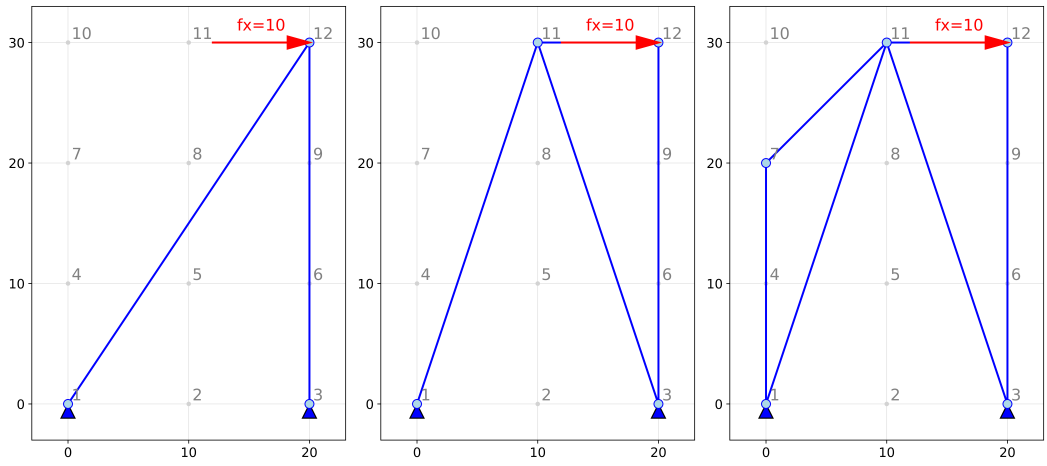

Figure 3.4: Example of a randomly generated structure corresponding to the action sequence $T1\langle 2\rangle[1\text{-}12, 11, T;\ 1\text{-}11, 7, D]$. From left to right: initial seed configuration, intermediate configuration after the first action, and final structure after the second action.

the resulting configurations are not optimized and may exhibit non-optimal geometrical layouts. Their purpose is solely to populate the training dataset with structurally admissible and diverse examples.

3.3.2. Dataset Weighting Strategy

Before applying the weighting procedure, a preprocessing step is performed. The worst 10% of the structures — i.e., the 10% with the largest values of $U_{\max}$ — are removed. This choice is motivated by numerical considerations. The most ill-conditioned configurations generated exhibit extremely large displacement values, which would excessively stretch the normalization range and significantly reduce the effectiveness of the subsequent weighting procedure. By removing this upper tail of the distribution, the normalization becomes more representative of the physically meaningful range of solutions and the weighting mechanism becomes more sensitive.

To formalize the procedure, let us consider a filtered dataset composed of n_{valid} strings and define the vector

$$\mathbf{u} = [u_1, u_2, \ldots, u_{n_{\text{valid}}}] \in \mathbb{R}^{n_{\text{valid}}}, \tag{3.17}$$

where u_i denotes the maximum displacement $U_{\max}$ associated with the i-th structure in the dataset with $i = 1, \ldots, n_{\text{valid}}$.

For the filtered dataset, we then define

$$u_{\min} = \min_i u_i, \qquad u_{\max} = \max_i u_i, \tag{3.18}$$

which represent the minimum and maximum displacement values within the dataset. These quantities are used to compute the normalized performance indicator

$$u_i^{\text{norm}} = \frac{u_i - u_{\min}}{u_{\max} - u_{\min}}, \qquad u_i^{\text{norm}} \in [0, 1], \tag{3.19}$$

for $i = 1, ..., n_{\text{valid}}$.

In order to assign greater importance to structurally optimal configurations while reducing the influence of suboptimal ones, the weight associated with the i-th example is defined through the exponential mapping

$$w_i = \exp(-\alpha\, u_i^{\text{norm}})\,, \tag{3.20}$$

where $\alpha > 0$ controls the sharpness of the weighting distribution.

This function was specifically selected to amplify performance differences in a smooth and continuous manner. The exponential form guarantees strictly positive weights and introduces a nonlinear emphasis on high-quality designs: since lower values of $U_{\max}$ correspond to better structural performance, the negative exponent ensures that smaller u_{norm} values produce larger weights, thereby biasing the gradient updates toward structurally superior configurations.

The choice of a normalized indicator $u_{\text{norm}} \in [0, 1]$ is crucial. Constraining the argument of the exponential within a bounded interval prevents excessive growth of the weights and avoids destabilizing the learning process. Preliminary experiments were conducted using more aggressive weighting strategies, in which optimal configurations were assigned weights significantly larger than 1. However, this approach led to unstable optimization dynamics: the model progressively lost its ability to reproduce valid grammatical structures, including intrinsic syntactic capabilities inherited from pretraining.

Two weighted dataset configurations, corresponding to different values of the hyperparameter α, were defined to support a staged training strategy, in which the model is first fine-tuned for two epochs using the dataset with $\alpha = 6$, and subsequently for four additional epochs using the dataset with $\alpha = 10$:

- $\alpha = 6$, corresponding to a moderately selective weighting;
- $\alpha = 10$, corresponding to a stronger emphasis on high-performing configurations.

The configuration with $\alpha = 6$ is designed to promote stable grammar acquisition while gradually introducing performance awareness into the learning process. In contrast, the configuration with $\alpha = 10$ increases the selection pressure, more strongly biasing the training toward structurally efficient designs and encouraging the model to prioritize high-quality configurations. The qualitative behavior of the exponential weighting function for the two values of α is illustrated in Fig. 3.5.

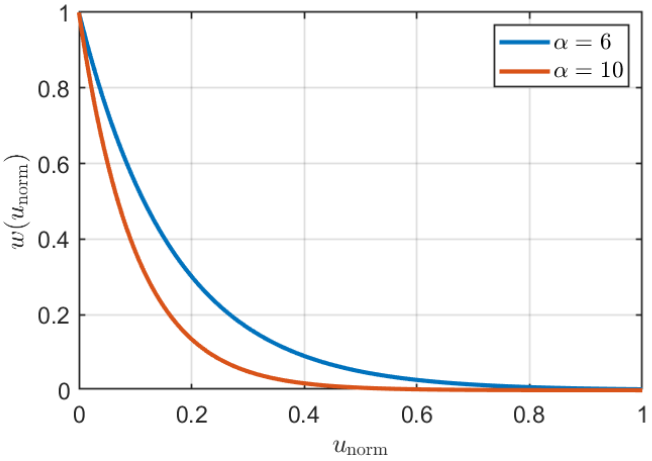

Figure 3.5: Exponential weighting function $w(u) = \exp(-\alpha u_{\text{norm}})$ for $\alpha = 6$ and $\alpha = 10$. Increasing α amplifies the relative contribution of high-performance configurations (low displacement values).

Formally, the two datasets are defined as

$$\mathcal{D}_{\alpha=6} = \left\{ \left(s_i, w_i^{(6)} \right) \right\}_{i=1}^{n_{\text{valid}}},$$

$$\mathcal{D}_{\alpha=10} = \left\{ \left(s_i, w_i^{(10)} \right) \right\}_{i=1}^{n_{\text{valid}}},$$

where s_i denotes the text string representation of the i-th structure, and $w_i^{(\alpha)} \in [0, 1]$ is the weight assigned according to the chosen value of α.

3.4. GPT-2 Model and Tokenization

In this work, the GPT-2 base model (124M parameters), initialized with the pretrained weights released by OpenAI and implemented in the HuggingFace `Transformers` library, is adopted as the backbone language model. The model is not trained from scratch; instead, a full fine-tuning procedure is performed on the generated dataset of structural sequences described in the previous section. The choice of fine-tuning rather than training a model from random initialization is motivated by the limited size of the available domain-specific dataset. Training a Transformer architecture from scratch typically requires very large corpora to achieve stable convergence and meaningful generalization. By starting from pretrained weights, the model needs less data to learn new features.

The selection of GPT-2 base represents a deliberate trade-off between model capacity and computational efficiency. With approximately 124 million parameters, the model is sufficiently expressive to capture the combinatorial patterns underlying the structural generation process. At the same time, larger-scale models with billions of parameters were not necessary for the present task.

The structural sequences are processed using the standard `GPT2Tokenizer`, which relies on Byte-Pair Encoding (BPE) and the original GPT-2 vocabulary of 50257 tokens. BPE operates through a frequency-based merging procedure: starting from byte-level units, the most frequent adjacent symbol pairs observed during pretraining are iteratively merged to form subword tokens. The resulting merge rules are fixed by the pretrained tokenizer and are reused without modification in the present work.

As a consequence, the strings are segmented according to statistical patterns learned from natural language corpora, rather than structural semantics. For example, the sequence `T1<2>[1-12,11,T;1-11,7,D]` is decomposed into subword tokens such as `T`, `1`, `<`, `2`, `>`, `[`, `1`, `-`, `12`, `,`, `11`, `,`, `T`, `;`, ..., depending on the predefined merge rules.

Each token is mapped to a 768-dimensional vector through the pretrained embedding matrix W_e of size 50,257 $\times$ 768, initialized with the official GPT-2 weights released by OpenAI. Although W_e is loaded with pretrained parameters, it is not kept fixed during training. Instead, it is updated through full fine-tuning, allowing the token representations to progressively adapt to the structural grammar and combinatorial patterns characteristic of the truss generation process. This design choice preserves full compatibility with the pretrained architecture and benefits from the regularization induced by large-scale pretraining. At the same time, it requires the embedding space to reorganize so as to encode the structural and symbolic relationships specific to this application.

3.5. Fine-tuning

This section describes the customized fine-tuning strategy adopted in this thesis. The training procedure is modified to explicitly incorporate structural performance information through a weighted loss formulation. Particular attention in this section is devoted to:

i batching and padding of token sequences;

ii modified loss formulation introduced in this framework.

3.5.1. Batching and Padding of Token Sequences

To formalize the representation of the training data, let s_i denote the textual string encoding the i-th truss configuration in the dataset. The GPT-2 tokenizer maps s_i into a sequence of tokens,

$$\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,T_i}), \tag{3.21}$$

where T_i is the sequence length. The token sequence $\mathbf{x}_i$ constitutes the actual input provided to the language model during training.

Let $\mathcal{D} = \{(\mathbf{x}_i, w_i)\}_{i=1}^{n_{\text{valid}}}$ denote the dataset, where $\mathbf{x}_i$ is the token sequence associated with the i-th truss configuration and w_i is the corresponding performance-based weight.

During training, the dataset is not processed one sequence at a time, but rather in groups of examples called mini-batches. A batch enables multiple sequences to be processed simultaneously, allowing efficient parallel computation and a more stable training process. Formally, a batch of size B is defined as

$$\mathcal{B} = \{\mathbf{x}^{(b)}\}_{b=1}^{B}. \tag{3.22}$$

where each tokenized sequence $\mathbf{x}^{(b)}$ represents an independent training example.

Since truss configurations are generated through a variable number of grammar operations, the resulting sequences generally have different lengths T_b within a batch. To enable parallel computation, all sequences must share a common length. For batch B, the maximum sequence length is defined as

$$T_{\max}^{B} = \max_{b \in \{1,\ldots,B\}} T_b, . \tag{3.23}$$

Each sequence $\mathbf{x}^{(b)}$ in the batch is then extended by appending <pad> tokens after its last valid token until length $T^B_{\max}$ is reached. An illustrative example of this dynamic padding procedure is shown in Fig. 3.6.

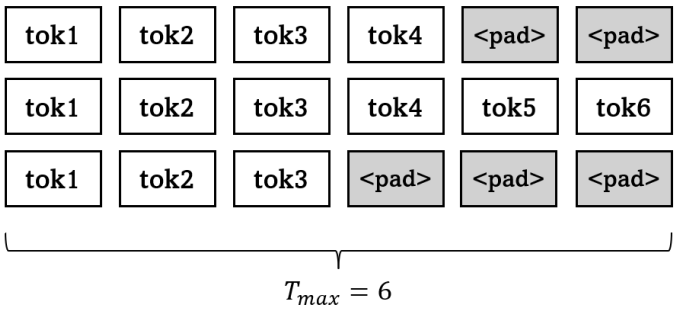

Figure 3.6: Illustrative example with three sequences padded to a common length $T^B_{\max} = 6$.

We denote by t the position index within a sequence, ranging from the first token to the last one. Autoregressive training requires that the prediction at position t depends only on tokens at positions strictly preceding it. This constraint is enforced through the causal attention mask, defined as the lower-triangular matrix

$$\mathbf{C}^B = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \in \{0,1\}^{T^B_{\max} \times T^B_{\max}}, \tag{3.24}$$

which ensures that each position t can attend only to positions $\leq t$ within the sequence.

Although padding tokens are introduced for dimensional consistency, they do not correspond to meaningful structural actions. For this reason, a padding mask

$$\mathbf{P}_B \in \{0,1\}^{B \times T^B_{\max}} \tag{3.25}$$

is defined, with entries

$$(\mathbf{P}_B)_{b,t} = \begin{cases} 1 & \text{if } t \leq T_{i_b}, \\ 0 & \text{if } t > T_{i_b}, \end{cases} \tag{3.26}$$

where T_{i_b} is the length of the sequence b before the padding procedure. This padding mask ensures that padded positions are excluded from contributing to the loss computation described in the following paragraph.

3.5.2. Customized Loss Formulation

For the sake of clarity, the loss formulation is presented here for a single sequence, omitting the batching procedure introduced in Section 3.5.1. The masking mechanism previously defined is implicitly assumed to exclude padded positions from the loss computation.

Consider a tokenized sequence $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,T_i})$ from the dataset. The customized objective is constructed through three distinct stages.

Stage 1 — Token-level loss. Under the standard autoregressive training paradigm, the model processes the sequence sequentially. At each position t, it produces a probability distribution over the vocabulary conditioned on the prefix $x_{i,<t} = (x_{i,1}, \ldots, x_{i,t-1})$. This probability distribution thus measures the likelihood of the correct next token $x_{i,t}$ under the model.

The token-level negative log-likelihood is defined as

$$\ell_{i,t} = -\log p_\theta(x_{i,t} \mid x_{i,<t}), \tag{3.27}$$

which penalizes low probability assigned to the true token.

Stage 2 — Sequence-level aggregation. Due to the autoregressive shift, the first token does not generate a prediction target. The standard sequence-level loss is therefore obtained by averaging the token-level losses over the valid prediction positions $t = 2, \ldots, T_i$:

$$\mathcal{L}_{\text{GPT2}}(\mathbf{x}_i) = \frac{1}{T_i - 1} \sum_{t=2}^{T_i} \ell_{i,t}. \tag{3.28}$$

This expression corresponds to the conventional fine-tuning objective for autoregressive LLMs.

Stage 3 — Performance-based reweighting. In the proposed framework, the standard objective is modified at the sequence level by introducing structural performance

information through a weight $w_i \in [0, 1]$, computed according to the exponential weighting function defined in Eq. 3.20.

The customized loss is therefore defined as

$$\mathcal{L}_{\text{final}}(\mathbf{x}_i) = w_i \, \mathcal{L}_{\text{GPT2}}(\mathbf{x}_i). \tag{3.29}$$

The modification occurs explicitly at this stage: instead of contributing uniformly to the optimization process, each sequence is scaled according to its structural quality. Sequences associated with lower displacement values (i.e., better structural performance) receive larger weights and therefore produce proportionally larger gradient contributions, while structurally inferior configurations exert a reduced influence on the parameter updates.

3.6. Inference

After fine-tuning, the trained LLM is used to generate new truss configurations through an autoregressive process. Starting from a given prompt, the model produces a continuation by predicting one token at a time. At each step, the prediction is conditioned on all previously generated tokens, so that the structure is built sequentially. For every new position, the model outputs a probability distribution over the vocabulary of possible grammar actions. One token is then sampled from this distribution and appended to the sequence. The process is repeated until a stopping criterion is reached, such as the generation of a special end-of-sequence token or the attainment of a predefined maximum length.

In our application, the input prompt consists of the task identifier and the number of nodes to be added, formatted as

$$T\langle id_{\text{task}}\rangle \ \langle k\rangle \ [$$

During inference, generation is forced to terminate as soon as the first closing bracket `]` is produced. To regulate the level of randomness during generation, temperature scaling is applied at sampling time. The temperature parameter T controls how strongly the model favors high-probability tokens over less likely ones [55].

For very low values of T (e.g., $T \approx 0.01$), the distribution becomes highly peaked, and the model behaves almost deterministically, repeatedly selecting the most probable token at each step. This leads to highly consistent but potentially repetitive outputs.

As T increases, the distribution becomes progressively flatter. Less probable tokens acquire a non-negligible chance of being selected, which increases variability and promotes exploration of alternative structural configurations.

For sufficiently large values of T, the generation process becomes highly stochastic, often at the expense of structural coherence and grammatical validity.

Temperature therefore provides a continuous mechanism to balance exploitation of the most likely structural patterns against exploration of new design possibilities.

Within the proposed framework, inference is initially performed with a low temperature in order to favor the generation of structurally consistent configurations. This strategy prioritizes solutions that the model considers most reliable, effectively targeting designs associated with lower structural displacement. If a generated structure coincides with one already encountered during training, the temperature is progressively increased to encourage exploration of alternative configurations. This adaptive adjustment promotes

the discovery of novel configurations beyond those contained in the training dataset.

In general, the fine-tuned model proves capable of generating grammatically admissible sequences of actions that satisfy the same consistency checks adopted during dataset construction.

Temperature parameter The generation process is controlled through a temperature parameter T applied to the softmax distribution of the next-token probabilities. Let p_i denote the original probability associated with token i. Temperature scaling modifies the distribution as

$$p_i^{(T)} = \frac{\exp\left(\frac{\log p_i}{T}\right)}{\sum_j \exp\left(\frac{\log p_j}{T}\right)}, \tag{3.30}$$

where j ranges over the vocabulary indices. This mechanism implies that for $T \to 0$, the distribution becomes increasingly peaked, approaching deterministic greedy decoding. In this regime, the model consistently selects the most probable token, resulting in highly reproducible and exploitation-driven generation. Conversely, increasing T flattens the probability distribution, allowing lower-probability tokens to be sampled more frequently. This promotes structural diversity and exploration of less likely configurations.

In the context of structural optimization, low temperatures favor solutions closely aligned with the dominant patterns learned during training, while higher temperatures enable the exploration of alternative topologies that may not have been explicitly observed in the dataset, at the cost of increasing the risk of violating the prescribed grammar rules.

Reconstruction module In addition to the generation procedure, a reconstruction module is employed to convert textual outputs into explicit truss configurations. Each generated string follows the structured format introduced in Eq. 3.16.

The reconstruction procedure parses the string sequentially. First, the task identifier and the parameter k are extracted in order to initialize the corresponding seed configuration. Subsequently, the ordered list of triplets is read from left to right. At each step i, the operator $operator_i$ is applied to the specified $element_i$ and $node_i$, updating the structural state deterministically.

The module operates in a manner analogous to Algorithm 3.1, with the key difference that no action sampling is performed. Instead, the actions are sequentially read from the input text string, after which the admissibility checks are carried out, and finally, the finite element analysis is executed.

4 | Results

In this chapter, the proposed framework is evaluated on a set of benchmark truss optimization problems. Specifically, six case studies, hereafter referred to as tasks, are characterized by different seed configurations, loading conditions, grid resolutions, and boundary constraints. A detailed description of each task is provided in the following section. The use of established benchmark problems enables a direct comparison with previously reported results in the literature [57] and provides a consistent reference for assessing the effectiveness of the generative strategy.

The computational workflow was divided into two distinct stages: dataset generation and model training. For each task, a dedicated dataset was generated in a `customized Python` environment on a machine equipped with an `11th Gen Intel® Core™ i7-11800H CPU, 16 GB RAM`. The training was performed in a `PyTorch` environment on a separate machine featuring an `AMD Ryzen™ 9 5950X, 128 GB RAM` and `NVIDIA GeForce RTX 3080 GPU, 10 GB VRAM`.

The remainder of this chapter is organized as follows. Section 4.1 introduces the investigated benchmark tasks, and Section 4.2 analyzes the relative datasets. Section 4.3 describes the inference strategy adopted to generate structures by varying the temperature parameter, which controls the token selection probability during generation. Section 4.4 presents the results obtained with the best structure generated for each task. Section 4.5 discusses those results and the main strengths and limitations of the proposed approach.

4.1. Investigated Tasks

This section introduces the six benchmark truss optimization problems adopted to evaluate the proposed framework. The case studies are adapted from [57, 59], allowing a direct comparison with previously reported results and with the corresponding optimal configurations identified in the literature. Consistent with the reference works, all problems concern planar trusses characterized by dimensionless mechanical properties.

Each case study is defined by a set of key parameters, summarized in Table 4.1: the size of the design domain, the number of actions N_a, the maximum admissible structural volume $V_{\max}$, the nodal spacing parameters d_x and d_y, and possible additional boundary constraints.

The parameters d_x and d_y denote the horizontal and vertical spacing between two adjacent nodes in the design grid, respectively. The column labeled "Additional constraint" specifies the presence of an extra pinned hinge introduced in the problem formulation. When present, the structure is required to reach it during the sequential construction process. In other words, the generative procedure must produce a configuration that connects to this constrained node, ensuring structural consistency with the imposed boundary conditions. The parameter N_a bounds the length of the generated action sequence and, consequently, the maximum structural complexity. Its value is selected consistently with Refs. [57, 59].

For each case study, the corresponding design domain, initial seed configuration s_0, externally applied loads, and boundary conditions are illustrated in Fig. 4.1 and Fig. 4.2. The globally optimal configuration s_T, obtained through brute-force search in the reference works, is shown alongside for comparison. It is worth noting that Case Study 4 involves the introduction of an additional constraint at node 1, located at coordinates $(0, 0)$. Moreover, Case Study 5 is characterized by a grid spacing equal to $dx = 20$ and $dy = 10$.

Table 4.1: Task configuration.

	Domain size	N_a	$V^{\max}$	d_x	d_y	Additional constraint
Task 1	4×3	2	160	10	10	–
Task 2	5×3	3	240	10	10	–
Task 3	5×5	3	225	10	10	–
Task 4	5×9	3	305	10	10	Node 1
Task 5	5×5	4	480	20	10	–
Task 6	7×7	4	350	10	10	–

The objective function considered in all case studies corresponds to the infinity norm of the displacement vector of the global optimal configuration, namely

$$\|\mathbf{U}(\bar{\Omega})\|_{\infty}.$$

Table 4.2 shows the objective value for the six tasks.

Table 4.2: Task objective.

	$\|\mathbf{U}(\bar{\Omega})\|_{\infty}$
Task 1	0.0895
Task 2	0.1859
Task 3	0.0361
Task 4	0.5916
Task 5	0.0390
Task 6	0.0420

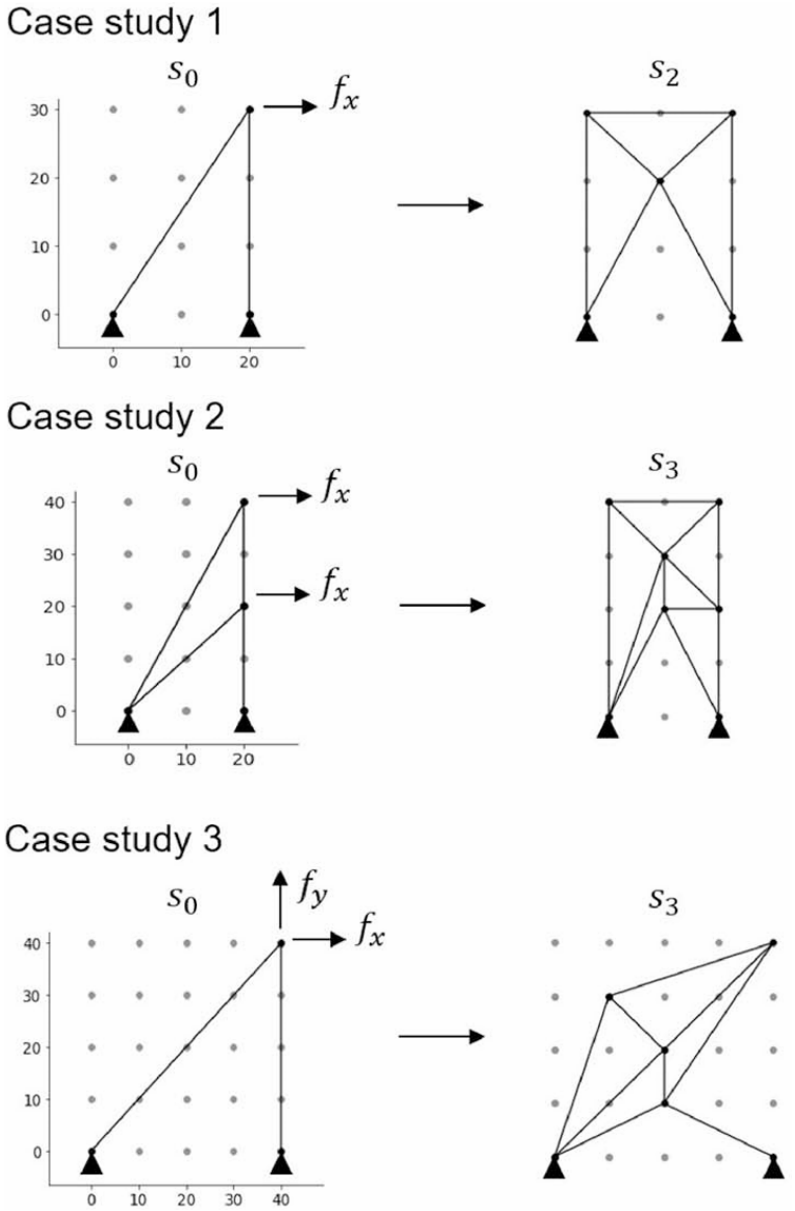

Figure 4.1: Task 1, Task 2, Task 3: initial seed configuration s_0 and corresponding globally optimal design s_T (adapted from [57]).

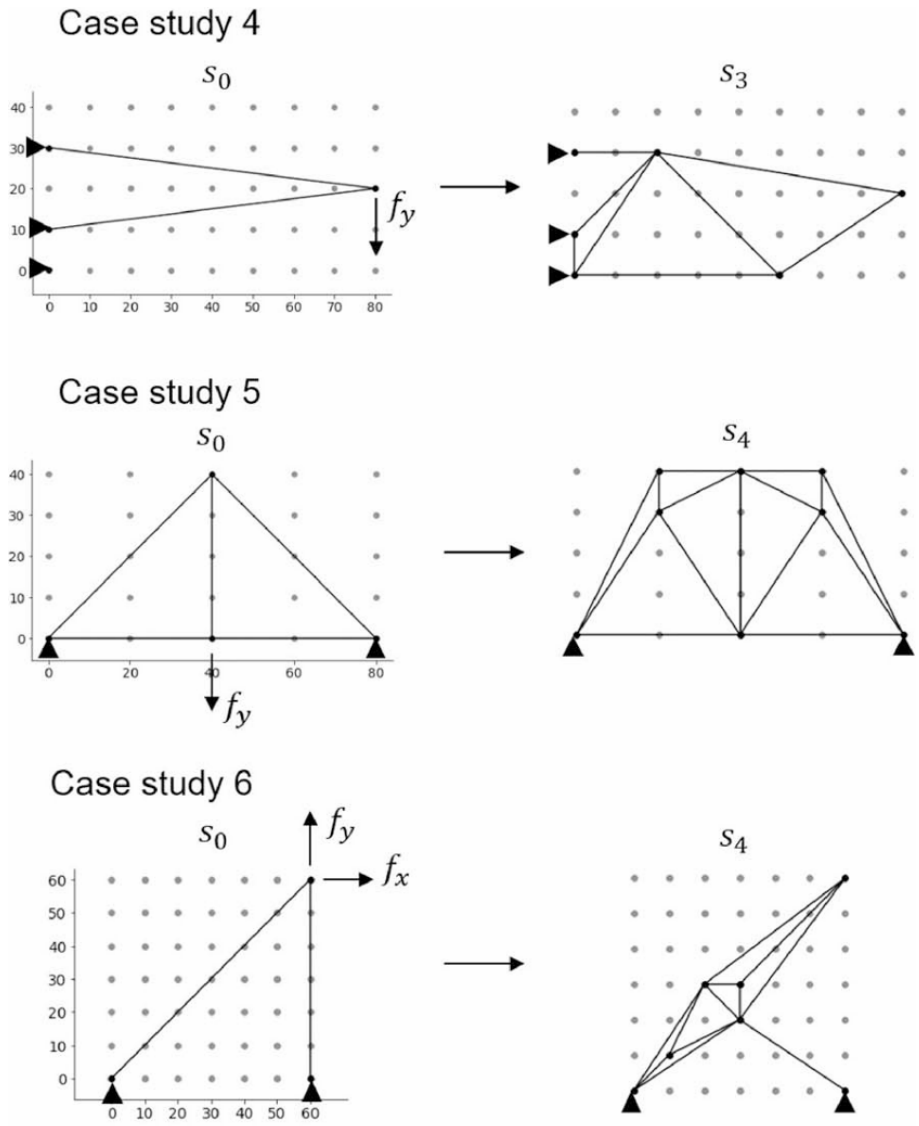

Figure 4.2: Task 4, Task 5, Task 6: initial seed configuration s_0 and corresponding globally optimal design s_T (adapted from [57]).

4.2. Dataset Analysis

Table 4.3 summarizes the dataset for each task, reporting the number of configurations n_{valid} and the corresponding range of displacement values. Moreover, the empirical distributions of the $U_{\max}$ values for all samples included in the datasets of the six tasks are reported in Appendix C.

Table 4.3: Summary table – Dataset

Task	n_{valid}	$u_{\min}$	$u_{\max}$
1	339	0.0895	0.2725
2	8375	0.1859	0.6363
3	6358	0.0361	0.2516
4	5653	0.5916	1.8047
5	83597	0.0390	0.1524
6	40497	0.0420	0.6761

From a theoretical standpoint, the maximum number of possible configurations for each task is not derived in the present work. In principle, such a quantity would require a combinatorial enumeration involving the number of selectable elements, activatable nodes, and admissible operators at each construction step. However, due to the grammar rules dynamically constraining the action space, these quantities are state-dependent and therefore not fixed a priori, preventing a straightforward computation. The total number of configurations can nevertheless be obtained through an exhaustive brute-force search. For instance, for Task 5, an exhaustive enumeration reported in [71] yields a total of 1 436 800 admissible configurations. This means that for Task 5, the dataset provided in this work included 5.82% of all possible configurations.

Taking Task 5 as a reference, Figure 4.3 shows the empirical distribution of the $U_{\max}$ values for all samples included in the dataset.

It can be observed that the majority of configurations are far from the minimum value. However, thanks to the weighting strategy described in Section 3.3, configurations associated with larger displacement values are assigned significantly lower weights. The distributions of the weighting function w for $\alpha = 6$ and $\alpha = 10$ are shown in Figures 4.4 and 4.5, respectively. As can be seen, most structures receive a negligible weight; consequently, only a limited number of high-performing configurations contribute significantly to the loss function.

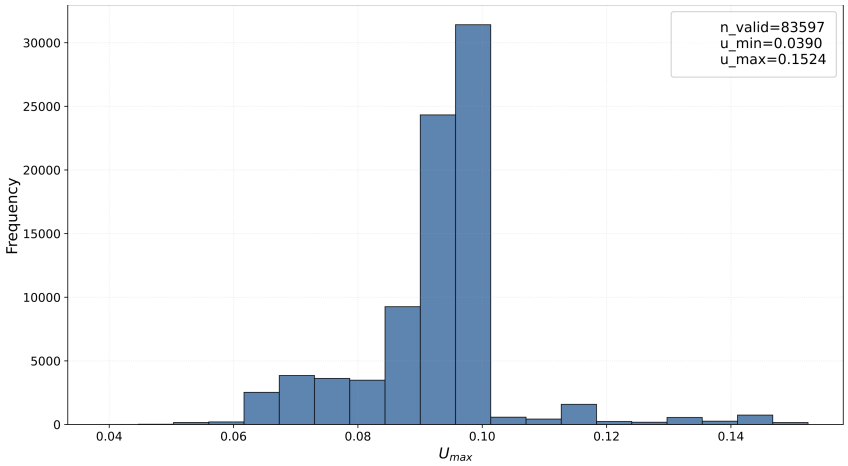

Figure 4.3: Distribution of $U_{\max}$ values for Task 5.

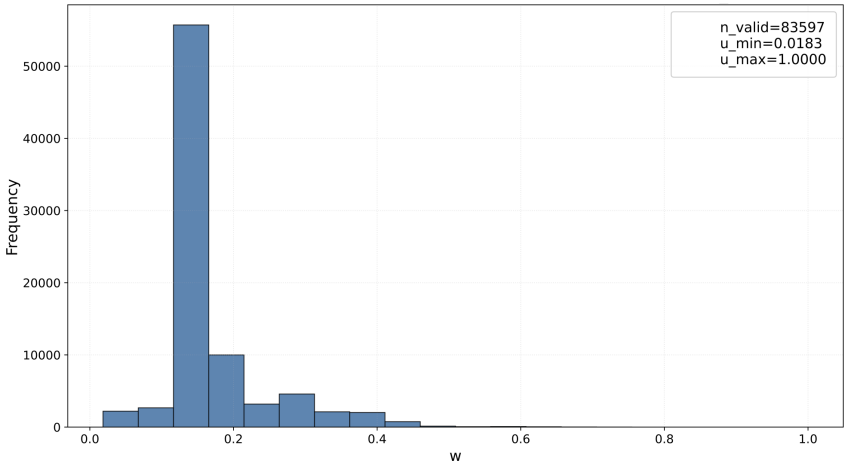

Figure 4.4: Distribution of w for $\alpha = 6$.

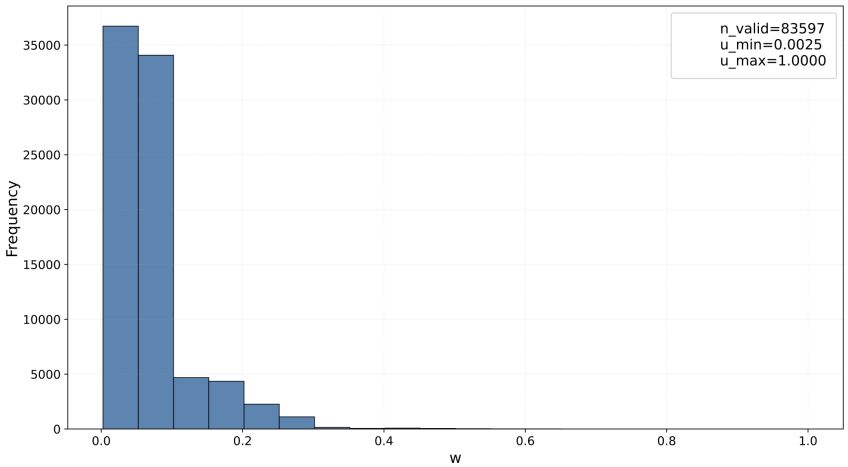

Figure 4.5: Distribution of w for $\alpha = 10$.

4.3. Generation Strategy

This section describes the inference procedure adopted to generate structural configurations from fine-tuned LLMs and the adaptive sampling strategy in which the temperature is progressively increased to balance exploitation and exploration of the solution space.

For each case study, a dedicated fine-tuned LLM is loaded together with the corresponding prompt encoding the task identifier and the required number of actions to perform. During inference, the model operates autonomously and does not require access to the training dataset. The dataset is not used to guide generation; instead, it is employed solely as a reference to verify whether a generated structural sequence has already been observed during training. The generative process is exclusively based on the probability distribution learned by the model during training and encoded in the model parameters.

Given a prompt like `"T1<2>["` the model generates a sequence of tokens until a termination condition is reached; in this application, the token `"]"`. The resulting string can be later reconstructed into the corresponding truss configuration and evaluated through FEM.

The generation process is controlled through the temperature parameter T. The adopted generation strategy follows a progressive temperature adjustment mechanism. The procedure starts from a low temperature value $T_0 = 0.01$, selected to provide a reference solution corresponding to the most probable structural sequence predicted by the model. After that, other configurations are explored by increasing the temperature until 10 different configurations are generated.

The adaptive generation procedure reported in Algorithm 4.1 controls the sampling temperature to balance exploitation and exploration during structural generation. The algorithm takes as input a task-specific prompt, the corresponding fine-tuned LLM, and the training dataset. The procedure starts from a baseline temperature $T_0 = 0.01$. If no novel candidate is produced, the algorithm proceeds to the next temperature level. Once a novel structure is identified, the temperature is fixed at the corresponding level and additional configurations are generated, up to a maximum of 50 candidates. This strategy is motivated by the observation that, at low temperature values, the model tends to repeatedly generate the same configuration. Therefore, when a novel structure is first obtained, the corresponding temperature is considered sufficient to initiate the exploration of the design space.

Each generated sequence undergoes two verification steps. The novelty check verifies whether a generated sequence is already present in the training dataset and is used to assess the model's ability to go beyond mere memorization of previously observed struc-

tural configurations. If present in the dataset, the structure is labeled as seen; otherwise, it is classified as novel. The Duplicate check, instead, ensures that the newly generated structural string has not already been produced during the current inference session. If the string has already been generated, it is discarded to prevent repetition.

The output of the procedure consists of a set of 10 distinct structural configurations, each labeled with the associated sampling temperature and classified as either seen or novel with respect to the training dataset. The sets of configurations for each case study are reported in Appendix A.

Algorithm 4.1 Adaptive Temperature-Based Structure Generation

Input:

Prompt
Fine-tuned LLM
Training Dataset (for novelty check)

Initialize $T_0 = 0.01$, $T_{\max} = 1.0$, $\Delta T = 0.10$, $T_{\text{selected}} \leftarrow$ None
Generate structure with T_0
Novelty check
for $T = 0.10$ to $T_{\max}$ step ΔT **do**
    for $t = 1$ to 10 **do**
        Generate structure with T
        Duplicate check
        Novelty check
        $T_{\text{selected}} = T$
        for $t = 1$ to 50 **do**
            Generate structure with T_{selected}
            Duplicate check
            Novelty check
            break
        end for
    end for
end for

Output:

Set of 10 different structures labeled with associated T and Novel/Seen

4.4. Tasks Results

In this section, the results obtained for each structural optimization task are presented. For every task, ten candidate configurations were generated, and only the best-performing one is reported. We denote its maximum displacement value as $\overline{U}_{\max}$. The complete set of generated configurations is provided in Appendix A. Table 4.4 summarizes the objective ratios obtained for each task.

The "objective ratio" is a dimensionless parameter that quantifies how close the performance of the generated solution is to that of the global optimum. It is defined as

$$\left(\frac{\|\mathbf{U}(\bar{\Omega})\|_\infty}{U_{\max}} \right) \cdot 100$$

Table 4.4: Summary table - Objective ratio by task

Task	Objective ratio (%)
1	100
2	99.04
3	99.72
4	96.44
5	90.49
6	82.68

For each task, the best-performing structure is presented through a standardized layout consisting of a summary table followed by a graphical representation of the seed and final configurations. The complete sequence of actions for each task is reported separately in Appendix B. The table reports the following information:

- # (identifying which of the 10 generated candidates has been selected)
- prompt
- generated text
- $\overline{U}_{\max}$
- Objective value $\|\mathbf{U}(\bar{\Omega})\|_\infty$
- Objective ratio (%)
- temperature T
- novel/seen classification
- admissibility (compliance with the grammar rules)

Task 1

Table 4.5: Task 1-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T1<2>[1-12,10,D;1-12,8,T]`			0.0895	0.0895
#	Objective ratio (%)	T	Novel/Seen	Admissible
1	100	0.01	seen	yes

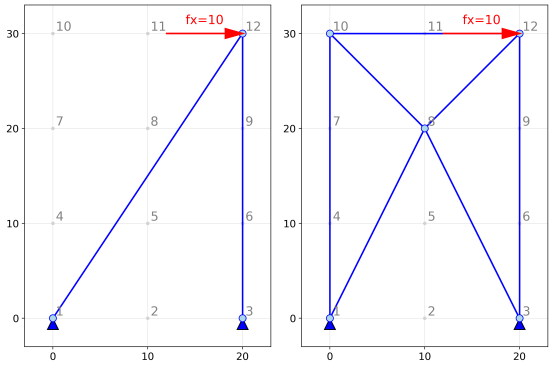

Figure 4.6: Task 1-seed and final configurations

Task 2

Table 4.6: Task 2-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T2<3>[1-15,11,T;1-9,8,T;8-11,14,D]`			0.1877	0.1859
#	Objective ratio (%)	T	Novel/Seen	Admissible
1	99.04	0.01	seen	yes

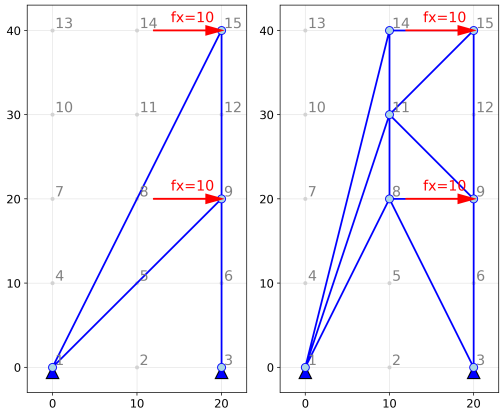

Figure 4.7: Task 2-seed and final configurations

Task 3

Table 4.7: Task 3-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T3<3>[5-25,8,D;1-25,12,T;5-25,19,T]`			0.0362	0.0361
#	Objective ratio (%)	T	Novel/Seen	Admissible
3	99.72	0.10	novel	yes

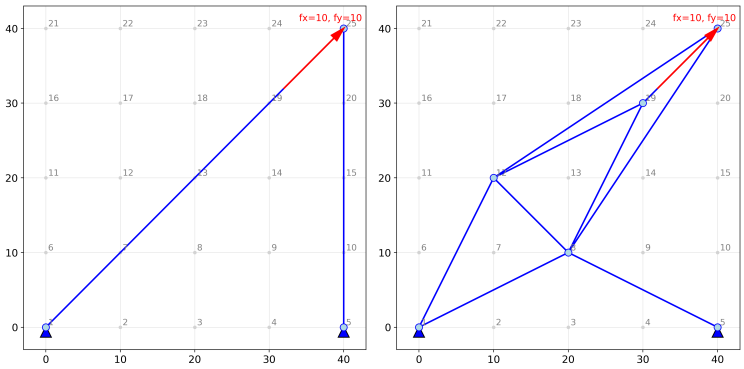

Figure 4.8: Task 3-seed and final configurations

Task 4

Table 4.8: Task 4-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T4<3>[10-27,1,T;1-27,3,T;27-28,41,T]`			0.6180	0.5916
#	Objective ratio (%)	T	Novel/Seen	Admissible
6	96.44	0.10	seen	yes

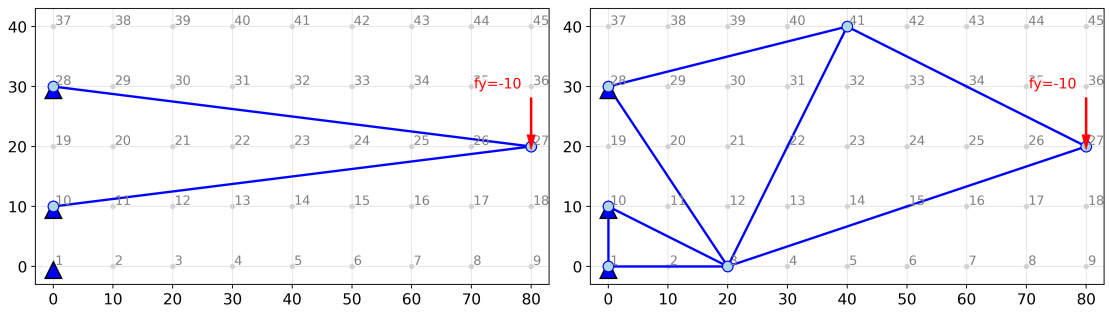

Figure 4.9: Task 4-seed and final configurations

Task 5

Table 4.9: Task 5-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T5<4>[3-5,20,D;1-23,17,T;5-23,19,T;3-17,22,D]`			0.0431	0.0390
#	Objective ratio (%)	T	Novel/Seen	Admissible
7	90.49	0.10	novel	yes

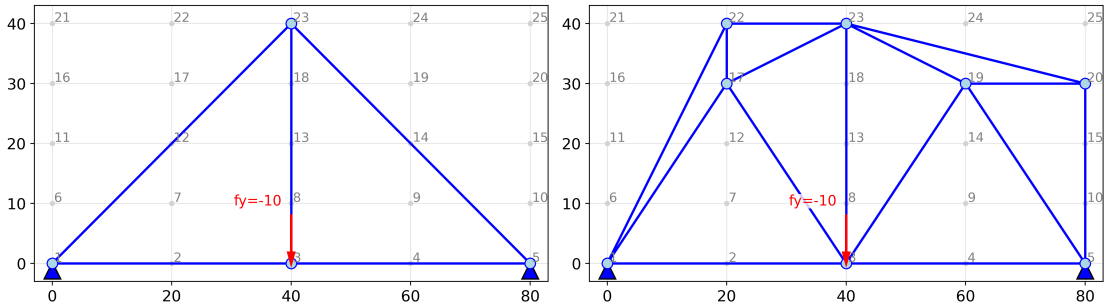

Figure 4.10: Task 5-seed and final configurations

Task 6

Table 4.10: Task 6-summary table

Generated text			$\overline{U}_{\max}$	Objective value
`T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-26,8,D]`			0.0508	0.0420
#	Objective ratio (%)	T	Novel/Seen	Admissible
8	82.68	0.10	novel	yes

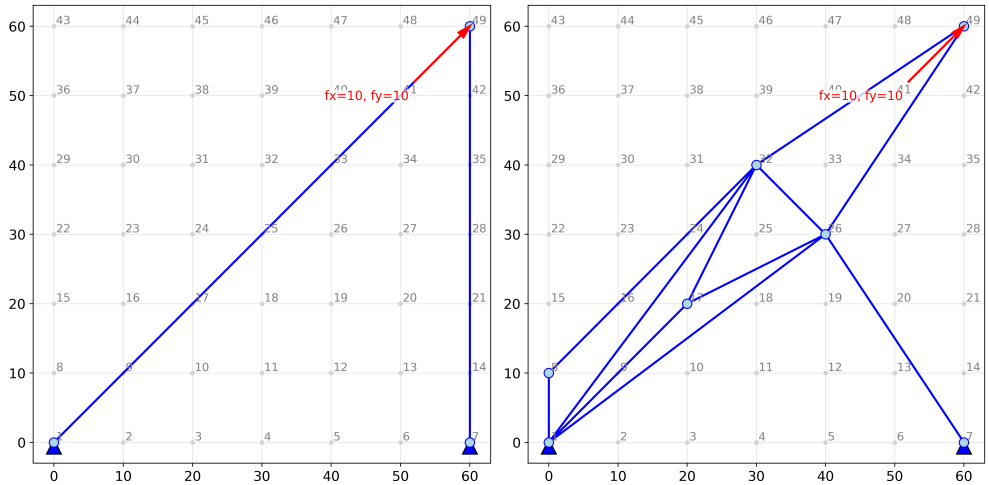

Figure 4.11: Task 6-seed and final configurations

4.5. Analysis of Results

The discussion is divided into two complementary parts: the first provides a general overview highlighting the capabilities of the model, while the second offers a more detailed examination, emphasizing the limitations of such models in reaching the global optimal solution.

In subsection 4.5.1, general observations are provided to identify common trends in the generative behavior of the models. Particular attention is devoted to the admissibility of the generated action sequences, the ability of the model to reach the global optimum in domains of different sizes, the capacity to generate novel yet admissible structural configurations, and the relationship between sampling temperature and structural performance.

In subsection 4.5.2, a more detailed probabilistic analysis is conducted to examine specific cases in which it was observed that the generated solution differs from the global optimum by a single action. In this context, the token probability distributions at critical decision steps are analyzed under different temperature settings in order to assess the influence of the temperature parameter on the sampling process. This investigation provides further insight into the underlying generation dynamics.

4.5.1. General Considerations

This subsection presents general observations derived from the 60 generated structures produced across the six case studies.

First, the models consistently learned the syntactic structure of the token sequences. In particular, the ordering and placement of separator tokens such as "`,`", "`;`", and "`-`" were correctly reproduced in all cases. Similarly, the closing token "`]`" was always generated in the appropriate position. These syntactic constraints were satisfied in 60 out of 60 generations, indicating that the LLM fully captured the formal structure of the encoding scheme.

Beyond syntax, the logical consistency imposed by the grammar rules was also largely respected. Specifically, 52 out of 60 generated configurations were admissible, meaning that they satisfied the geometric and topological constraints associated with the action selection process. This aspect suggests that the model has internalized implicit structural relationships embedded in the training data, rather than merely reproducing token patterns. The remaining 8 configurations were rejected during the reconstruction phase, as they violated the checks enforcing the grammar rules, typically through operations such as activating already active nodes or disrupting the continuity of existing members.

The model also demonstrated the ability to generate novel and admissible configurations, confirming that its behavior extends beyond simple memorization of the dataset. This capability becomes especially evident in tasks characterized by larger design spaces. In particular, Case Studies 5 and 6 produced exclusively novel configurations. These tasks involve larger grids and require a greater number of sequential actions, resulting in a combinatorially richer design space that is less exhaustively covered by the training dataset.

From a performance standpoint, the generated structures generally exhibited competitive objective ratios. However, these favorable results are partly attributable to the fact that only the best configuration out of the ten generated was considered. Nevertheless, this outcome still demonstrates the effectiveness of the generative approach.

A final general observation concerns deterministic decoding, corresponding to $T = 0.01$. In four out of six tasks, the deterministic solution was outperformed by configurations generated at higher temperature values. Moreover, in Case Study 5, the deterministic generation produced the only non-admissible structure within the corresponding set. These results suggest that moderate temperature increases can enhance exploration without significantly compromising structural validity. Notably, the adopted temperature range remained relatively conservative, with values not exceeding $T = 0.30$ for the most relevant generations.

4.5.2. Probabilistic Analysis at Critical Decision Steps

In Case Study 2, the model did not generate the global optimum, but it produced a configuration really close to it. A comparison with the reference optimal structure shows that the two designs differ only in the node selected during the final action. As illustrated in Fig. 4.12, selecting node `13` instead of node `14` in the last step would have resulted in the global optimum configuration.

This observation naturally raises the following question: once the first two actions had been correctly generated, was the optimal continuation represented within the model's learned probability distribution? More specifically, was node 13 a candidate node in the last solution? To answer this question, an inspection of the token probability distribution was performed at the critical generation step corresponding to the partial sequence

```
T2<3>[1-15,11,T;1-9,8,T;8-11,
```

and the next-token probabilities were analyzed under different temperature settings. The five most probable tokens with their associated probabilities are reported in Table 4.11.

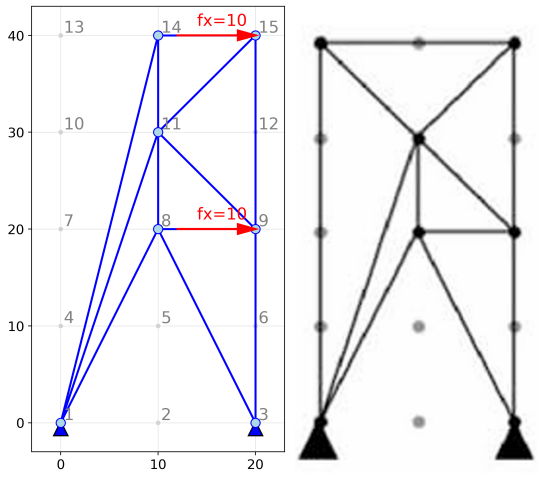

Figure 4.12: Task 2 - Generated solution on the left and optimal solution on the right (adapted from [57]).

Table 4.11: Probability distribution of the most probable 5 next tokens for different values of T given `T2<3>[1-15,11,T;1-9,8,T;8-11,.`

Token	$T = 1$	$T = 0.30$	$T = 0.01$
14	0.523	0.938	1.000
10	0.228	0.058	0.000
13	0.081	0.002	0.000
7	0.065	0.001	0.000
2	0.038	0.000	0.000
...	...	...	...

The token associated with node 13 appears as the third most probable candidate, with an assigned probability of approximately 8% for the original probability distribution. This confirms that the optimal continuation was present within the model's learned distribution, albeit not as the most likely choice. From Table 4.11, the effect of temperature scaling can be clearly understood. Reducing T significantly sharpens the distribution towards the most likely token. This behavior is fully amplified at $T = 0.01$.

This analysis highlights a critical aspect. It is true that the path toward the global optimum is present within the learned probability distribution, but it is also true that the model does not prioritize the path toward the optimal configuration, even if it was included in the training dataset and associated with the highest weight. This behavior suggests that the model is not driven by an intelligent optimization strategy as we would traditionally define it, but by a probabilistic mechanism that favors highly likely structural

patterns.

A similar analysis can be performed for Case Study 5. One might argue that, had the model selected node 24 instead of node 20 during the first action, the globally optimal structure would have been obtained. As in the previous case, we therefore examine the probability distribution associated with the generation of the next token given the partial sequence `T5<4>[3-5,`. The analysis reveals that token `20` is ranked second, with a probability comparable to that of token `24` (see Table 4.12).

Table 4.12: Probability distribution of the most probable 5 next tokens for different values of T given `T5<4>[3-5,`.

Token	$T = 1$
20	0.146
24	0.126
25	0.121
22	0.113
21	0.112
...	...

Furthermore, by explicitly forcing the first action through the prompt `T5<4>[3-5,24,D;`, the model generates the globally optimal configuration among the 10 candidates. At first glance, this could suggest that the suboptimal outcome was merely due to sampling variability. However, a deeper inspection requires analyzing the probability distribution associated with the first generated token itself (see Table 4.13).

Table 4.13: Probability distribution of the most probable 5 next tokens for different values of T given `T5<4>[`.

Token	$T = 1$	$T = 0.10$
1	0.461	0.891
3	0.309	0.107
5	0.207	0.001
23	0.015	0.000
4	0.002	0.000
...	...	...

The distribution associated with $T = 0.10$ is the one that appeared during the structure generation and shows that token `3` has a low probability of being generated, approximately 11%. The problem is that, following the considerations made above, it can be inferred that the path toward the global optimum passes through the generation of the first node `3`. However, because of the low temperature values, that path is very often avoided, since the model tends to select token `1` instead.

This behavior is reflected in the generated set reported in Appendix A: among the 10 sampled structures, only one begins with token `3`, and this configuration achieves the best objective value within the set. The remaining 9 structures start with token `1`.

These findings expose a limitation of the generation strategy described in Section 4.3. In order to maintain structural coherence and satisfy the grammar rules, low temperature values are required; however, such values may hinder the discovery of the global optimum.

Finally, an analysis is performed on the last node generated in Case Study 6. This action results in the addition of an isostatic appendage, as shown in Fig. 4.13. Human intuition suggests that, had a node located within the existing structure been selected instead of node 8, the structure would have been stiffened. The question is whether the model was capable of making such a consideration.

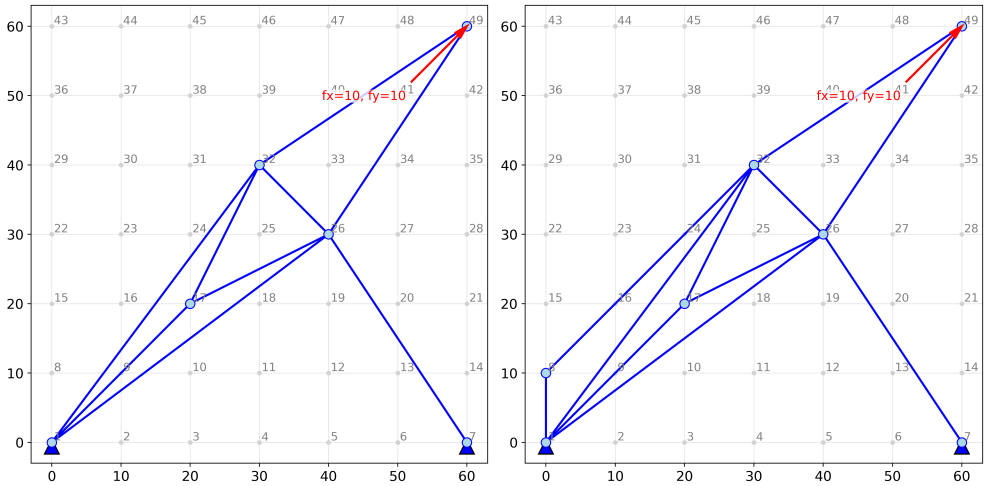

Figure 4.13: Last construction action `[17-26,8,D]` for the structure adopted in Task 6. Node `8` is added to the structure with operator $\mathcal{D}$.

In a manner analogous to the previous analysis, we examine the probability distribution associated with the partial sequence `T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-26,`. The resulting probability distribution, see Table 4.14, shows that the five most probable tokens would all have generated isostatic appendages. The sixth most probable token

would have produced a non-admissible structure, while the seventh, eighth, and ninth most probable tokens would have led to a structurally beneficial action. Therefore, the answer to the previous question is that the model does not possess the capability to capture that consideration. Furthermore, increasing the temperature is not a viable solution, since the displayed probability distribution is conditioned on all previously generated tokens. Therefore, even at higher temperature levels, the model would not have been able to generate a sufficient number of tokens leading to admissible actions.

Table 4.14: Probability distribution of the most probable 10 next tokens for different values of T given `T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-26,`.

Token	$T = 1$	$T = 0.10$
48	0.112	0.727
8	0.101	0.247
15	0.078	0.020
40	0.066	0.003
47	0.056	0.001
32	0.055	0.001
33	0.050	0.000
41	0.046	0.000
9	0.044	0.000
24	0.043	0.000
...	...	...

5 | Conclusions and Future Developments

Although LLMs were originally developed for NLP, this work has shown that they can be adapted to engineering problems like truss structure optimization. The structural design task was reformulated as a sequential token-generation process, and the custom loss function effectively biases the generation toward mechanically efficient configurations.

The obtained results demonstrate that the model is capable of generating mechanically admissible and novel structures with competitive performance. A relevant advantage of the proposed approach emerges during inference: once trained, the model can generate multiple candidate structures, enabling the selection of the best-performing solution among diverse alternatives. Notably, even when trained on datasets exploring only a limited portion of the design domain, as shown for Task 5, the model was able to produce previously unseen structures, suggesting that LLM-based frameworks can serve as exploratory tools in settings where only partial datasets are available.

Despite these positive results, several limitations must be acknowledged. The training phase remains computationally demanding. In fact, full fine-tuning of the adopted model required updating approximately 124 million parameters. While feasible, this represents a considerable computational investment. Furthermore, the dataset generation process constituted the most labor-intensive component of this thesis. The dataset was constructed from scratch and labeled with performance indicators such as the maximum displacement. While this ensured high-quality supervision, it also highlights a potential limitation of the approach. In fact, the key advantage of LLMs is that their training relies on large-scale, readily available textual data. However, their objective is fundamentally to replicate patterns observed in the training phase. Although this approach is well-suited for language generation, it is not directly aligned with optimization objectives. Consequently, when applied to optimization problems, the adoption of a customized loss function becomes necessary, unless the training data consist exclusively of high-performing solutions. Nevertheless, data generation and treatment are necessary steps for enabling the application

of an LLM-based framework to this class of problems.

These limitations motivate several directions for future research. To address the computational cost of full fine-tuning, parameter-efficient strategies such as Low-Rank Adaptation (LoRA) could be investigated, allowing only a subset of parameters to be updated while keeping the majority of the model frozen. Such an approach would significantly reduce the training time while preserving performance. Further developments of this thesis could involve the adoption of adaptive sampling strategies during inference. Dynamically varying the temperature parameter may enable the model to balance exploration and exploitation at each generation step, potentially mitigating the issues encountered in reaching the global optimal solution. For instance, higher temperature values could be employed in the early stages of generation to encourage the exploration of diverse structural patterns, whereas lower temperatures could be adopted in later stages to promote refinement and convergence toward high-performing configurations. Another possible direction is to adapt token generation in a greedy manner, selecting the locally optimal token at each step. In this setting, the finite element analysis would be executed after each action to evaluate intermediate configurations. This contrasts with the approach adopted in this work, where the entire structure is generated first and a single finite element analysis is performed only once the sequence is completed.

In conclusion, the primary objective of this thesis was to investigate whether an LLM-based framework could be applied to a problem of a physical nature. The results indicate that this is feasible when the underlying task is reformulated as a structured sequential representation. However, such a reformulation into token-based sequences remains dependent on human interpretative effort.

Bibliography

[1] Miguel A. Bessa, Pawel Glowacki, and Martin Houlder. Machine learning in materials science: Recent progress and emerging applications. *Advanced Materials*, 31:1904845, 2019.

[2] Kai Guo, Zhenze Yang, Chi-Hua Yu, and Markus J. Buehler. Artificial intelligence and machine learning in design of mechanical materials. *Materials Horizons*, 8:1153–1172, 2021.

[3] Marc A. Meyers and Krishan K. Chawla. *Mechanical Behavior of Materials*. Cambridge University Press, 2 edition, 2008.

[4] Markus J. Buehler. Machine learning for materials science. *Nano Futures*, 4:035004, 2020.

[5] Ever J. Barbero. *Introduction to Composite Materials Design*. CRC Press, 3 edition, 2017.

[6] Marc A. Meyers, Po-Yu Chen, Albert Y. M. Lin, and Yasuhide Seki. Biological materials: Structure and mechanical properties. *Progress in Materials Science*, 53: 1–206, 2008.

[7] L. R. Meza, S. Das, and J. R. Greer. Strong, lightweight, and recoverable three-dimensional ceramic nanolattices. *Science*, 345(6202):1322–1326, 2014.

[8] X. Zhang, Y. Wang, B. Ding, and X. Li. Mechanical metamaterials with tunable properties. *Small*, 16:1902842, 2020.

[9] Patrick Henry Winston. *Artificial Intelligence*. Pearson, 3 edition, 1992.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[12] Y. Liu, T. Zhao, W. Ju, and S. Shi. Materials discovery and design using machine learning. *Journal of Materiomics*, 3:159–177, 2017.

[13] Rohit Batra, Linyou Song, and Rampi Ramprasad. Emerging materials intelligence ecosystems propelled by machine learning. *Nature Reviews Materials*, 6(8):655–678, 2020. doi: 10.1038/s41578-020-00255-y.

[14] Phil De Luna, Jennifer Wei, Yoshua Bengio, Alán Aspuru-Guzik, and Edward H. Sargent. Use machine learning to find energy materials. *Nature*, 552(7683):23–27, 2017. doi: 10.1038/d41586-017-07820-6.

[15] G. H. Gu, J. Noh, I. Kim, and Y. Jung. Machine learning for renewable energy materials. *Journal of Materials Chemistry A*, 7:17096–17117, 2019.

[16] H. Liu, Z. Fu, K. Yang, X. Xu, and M. Bauchy. Machine learning for glass science and engineering: A review. *Journal of Non-Crystalline Solids: X*, 4:100036, 2019. doi: 10.1016/j.nocx.2019.100036.

[17] C. T. Chen and G. X. Gu. Machine learning for materials design. *MRS Communications*, 9:556–566, 2019.

[18] G. Chen, Z. Shen, A. Iyer, U. F. Ghumman, S. Tang, J. Bi, W. Chen, and Y. Li. Machine learning assisted polymer design. *Polymers*, 12:163, 2020.

[19] C. Zhai, T. Li, H. Shi, and J. Yeo. Machine learning for biomaterials. *Journal of Materials Chemistry B*, 8:6562–6587, 2020.

[20] F. E. Bock, R. C. Aydin, C. J. Cyron, N. Huber, S. R. Kalidindi, and B. Klusemann. A review of machine learning in materials science. *Frontiers in Materials*, 6:110, 2019.

[21] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, 5 edition, 2012.

[22] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58:267–288, 1996.

[23] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[24] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[25] Peter McCullagh and John A. Nelder. *Generalized Linear Models*. Taylor & Francis, 2 edition, 1989.

[26] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[27] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38:367–378, 2002.

[28] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[29] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 2003.

[30] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2 edition, 2004.

[31] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL `https://doi.org/10.1109/5.726791`.

[33] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318, Atlanta, GA, USA, 2013. June 16–21, 2013.

[34] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 1998.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[36] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, 2014. October 25–29, 2014.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, 2016. June 27–30, 2016.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, 2017. December 4–9, 2017.

[39] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. arXiv:1812.08434, 2018. URL `https://arxiv.org/abs/1812.08434`.

[40] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, Toulon, France, 2017. April 24–26, 2017.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2672–2680, Montréal, Canada, 2014. December 8–13, 2014.

[42] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, Venice, Italy, 2017. October 22–29, 2017.

[43] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, Honolulu, HI, USA, 2017. July 21–26, 2017.

[44] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. arXiv:1312.6114, 2013. URL `https://arxiv.org/abs/1312.6114`.

[45] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[46] R. Ravinder, K. H. Sridhara, S. Bishnoi, H. S. Grover, M. Bauchy, Jayadeva, H. Kodamana, and N. M. A. Krishnan. Machine learning framework for accelerated materials design. *Materials Horizons*, 7:1819–1827, 2020.

[47] Prasanna V. Balachandran, Brian Kowalski, Alp Sehirlioglu, and Turab Lookman. Adaptive strategies for materials design using machine learning. *Nature Communications*, 9:1668, 2018.

[48] E. Kim, K. Huang, A. Tomala, S. Matthews, E. Strubell, A. Saunders, A. McCallum,

and E. Olivetti. Materials synthesis insights from scientific literature via text mining. *Scientific Data*, 4:170127, 2017.

[49] Yu-Chen Hsu, Chi-Hua Yu, and Markus J. Buehler. Data-driven prediction of fracture patterns in brittle materials. *Matter*, 3:197–211, 2020.

[50] Z. Yang, S. Papanikolaou, A. C. E. Reid, W. Liao, A. N. Choudhary, C. Campbell, and A. Agrawal. Image-based data augmentation for materials microstructure learning. *Scientific Reports*, 10:8262, 2020.

[51] Markus J. Buehler. Melm, a generative pretrained language modeling framework that solves forward and inverse mechanics problems. *Journal of the Mechanics and Physics of Solids*, 181:105454, 2023. doi: 10.1016/j.jmps.2023.105454.

[52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR) Workshop*, Scottsdale, AZ, USA, 2013. May 2–4, 2013.

[53] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, Virtual Conference, 2020. December 6–12, 2020.

[54] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, Berlin, Germany, 2016. August 7–12, 2016.

[55] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. Technical report, OpenAI, 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

[56] Ross Taylor et al. Galactica: A large language model for science. arXiv:2211.09085, 2022. URL `https://arxiv.org/abs/2211.09085`.

[57] Gabriel Garayalde, Luca Rosafalco, Matteo Torzoni, and Alberto Corigliano. Mastering truss structure optimization with tree search. arXiv:2406.06145, 2024. URL `https://arxiv.org/abs/2406.06145`.

[58] G. Garayalde, M. Torzoni, M. Bruggi, and A. Corigliano. Real-time topology opti-

mization via learnable mappings. *International Journal for Numerical Methods in Engineering*, 125(15):e7502, 2024. doi: 10.1002/nme.7502.

[59] Maximilian E. Ororbia and Gordon P. Warn. Design synthesis through a markov decision process and reinforcement learning framework. *Journal of Computing and Information Science in Engineering*, 22(2):021002, 2022. doi: 10.1115/1.4051598.

[60] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, NY, 1998.

[61] W. S. Dorn, R. E. Gomory, and H. J. Greenberg. Automatic design of optimal structures. *Journal of Mechanics*, 3(6):25–52, 1964.

[62] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, MA, 1992.

[63] V. Permyakov, V. Yurchenko, and I. Peleshko. An optimum structural computer-aided design using hybrid genetic algorithm. In *Proceedings of the 2006 International Conference on Progress in Steel, Composite and Aluminium Structures*, pages 819–826, Rzeszow, Poland, 2006. Taylor & Francis. May 31–June 2, 2006.

[64] A. Hooshmand and Matthew I. Campbell. Truss layout design and optimization using a generative synthesis approach. *Computers & Structures*, 163:1–28, 2016.

[65] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, WA, Australia, 1995. IEEE. November 27–December 1, 1995.

[66] G.-C. Luh and C.-Y. Lin. Optimal design of truss-structures using particle swarm optimization. *Computers & Structures*, 89(23):2221–2232, 2011.

[67] V. Ho-Huu, T. Nguyen-Thoi, T. Vo-Duy, and T. Nguyen-Trang. An adaptive elitist differential evolution for optimization of truss structures with discrete design variables. *Computers & Structures*, 165:59–75, 2016.

[68] Luciano Lamberti. An efficient simulated annealing algorithm for design optimization of truss structures. *Computers & Structures*, 86(19-20):1936–1953, 2008.

[69] Maximilian E. Ororbia and Gordon P. Warn. Deep reinforcement learning for design synthesis of truss structures. *Journal of Mechanical Design*, 145(10):101701, 2023. doi: 10.1115/1.4063155.

[70] R. Alber and S. Rudolph. On a grammar-based design language that supports auto-

mated design generation and creativity. In J. C. Borg, P. J. Farrugia, and K. P. C. Camilleri, editors, *Knowledge Intensive Design Technology*, pages 19–35. Springer, St. Julians, Malta, 2004.

[71] Arvan Sedighzadeh. Designing with intelligence: Optimizing truss structures via hierarchical tree search. Master's thesis in civil engineering for risk mitigation, Politecnico di Milano, 2025.

A | Appendix A

In the following pages, the generated sequences for each task are reported. For each case study, 10 structural sequences were generated and listed together with their associated sampling temperature, novelty label (seen/novel), admissibility with respect to the grammar rules, and objective ratio.

Table A.1: Task 1 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T1<2>[1-12,10,D;1-12,8,T]	0.01	Seen	yes	100
2	T1<2>[3-12,7,D;1-12,8,T]	0.10	Seen	yes	96.44
3	T1<2>[3-12,10,D;1-12,8,T]	0.10	Seen	yes	100
4	T1<2>[3-12,11,D;1-12,8,T]	0.10	Seen	yes	99.67
5	T1<2>[1-12,7,D;1-12,8,T]	0.10	Seen	yes	96.44
6	T1<2>[1-12,11,D;1-12,8,T]	0.20	Seen	yes	99.67
7	T1<2>[3-12,5,D;1-12,8,T]	0.60	Seen	yes	67.14
8	T1<2>[1-12,8,D;1-12,8,T]	0.70	Novel	no	–
9	T1<2>[1-12,5,D;1-12,8,T]	0.70	Seen	yes	67.14
10	T1<2>[3-12,5,D;1-12,5,T]	0.70	Novel	no	–

Table A.2: Task 2 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T2<3> [1-15,11,T;1-9,8,T;8-11,14,D]	0.01	Seen	yes	99.04
2	T2<3> [1-15,11,T;1-9,8,T;9-15,14,D]	0.30	Novel	yes	99.04
3	T2<3> [1-15,11,T;1-9,8,T;8-9,14,D]	0.30	Seen	yes	99.04
4	T2<3> [1-15,11,T;1-9,8,T;8-11,10,D]	0.30	Novel	yes	97.95
5	T2<3> [1-15,11,T;1-9,8,T;3-9,14,D]	0.30	Seen	yes	99.04
6	T2<3> [1-15,11,T;9-15,13,D;1-9,8,T]	0.30	Seen	yes	100
7	T2<3> [1-15,11,T;1-9,8,T;9-11,14,D]	0.30	Novel	yes	99.04
8	T2<3> [1-15,11,T;1-9,8,T;11-15,14,D]	0.30	Seen	yes	99.04
9	T2<3> [1-15,11,T;1-9,8,T;3-9,10,D]	0.30	Seen	yes	97.95
10	T2<3> [1-15,11,T;9-15,14,D;1-9,8,T]	0.30	Novel	yes	99.04

Table A.3: Task 3 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T3<3> [5-25,8,D;1-25,12,T;5-25,6,T]	0.01	Seen	yes	77.30
2	T3<3> [5-25,8,D;1-25,12,T;5-25,18,T]	0.10	Novel	yes	47.13
3	T3<3> [5-25,8,D;1-25,12,T;5-25,19,T]	0.10	Novel	yes	99.72
4	T3<3> [5-25,18,D;1-25,14,T;5-25,7,T]	0.10	Seen	yes	97.57
5	T3<3> [5-25,18,D;1-25,13,T;5-25,24,D]	0.10	Seen	yes	70.92
6	T3<3> [1-25,14,D;1-25,18,T;5-25,7,T]	0.20	Novel	yes	97.57
7	T3<3> [1-25,8,D;1-25,12,T;5-25,19,T]	0.20	Seen	yes	99.72
8	T3<3> [5-25,12,D;1-25,8,T;5-25,19,T]	0.20	Seen	yes	99.72
9	T3<3> [5-25,18,D;1-25,13,T;5-25,14,T]	0.20	Seen	yes	86.36
10	T3<3> [5-25,8,D;1-25,12,T;5-25,7,T]	0.20	Seen	yes	88.05

Table A.4: Task 4 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T4<3>[10-27,1,T;1-27,4,T;4-10,11,D]	0.01	Seen	yes	77.57
2	T4<3>[10-27,1,T;1-27,4,T;4-10,19,D]	0.10	Seen	yes	77.51
3	T4<3>[10-27,4,T;4-10,1,D;4-27,11,D]	0.10	Seen	yes	77.57
4	T4<3>[10-27,4,T;4-10,1,D;4-10,19,D]	0.10	Seen	yes	77.51
5	T4<3>[10-27,4,T;4-10,1,D;4-27,6,D]	0.10	Seen	yes	77.51
6	T4<3>[10-27,1,T;1-27,3,T;27-28,41,T]	0.10	Seen	yes	96,44
7	T4<3>[10-27,1,T;1-27,3,T;3-10,11,T]	0.20	Novel	yes	66.75
8	T4<3>[10-27,1,T;1-27,4,T;4-10,11,T]	0.20	Novel	yes	75.67
9	T4<3>[10-27,1,T;27-28,40,T;10-40,38,T]	0.20	Seen	yes	76.22
10	T4<3>[10-27,1,T;1-27,4,T;1-10,19,D]	0.20	Seen	yes	77.51

Table A.5: Task 5 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T5<4> [1-23,17,T;3-5,25,D;5-23,19,T;3-5,19,T]	0.01	Novel	no	–
2	T5<4> [1-23,17,T;3-5,20,D;3-23,18,T;3-5,25,D]	0.10	Novel	yes	71.69
3	T5<4> [1-23,12,T;3-23,18,T;3-5,20,D;3-5,24,T]	0.10	Novel	yes	60.37
4	T5<4> [1-23,12,T;3-23,18,T;3-5,20,D;3-18,16,D]	0.10	Novel	yes	74.86
5	T5<4> [3-5,20,D;1-23,17,T;3-23,18,T;3-5,25,D]	0.10	Novel	yes	71.69
6	T5<4> [1-23,12,T;3-23,18,T;3-5,20,D;3-18,22,D]	0.10	Novel	yes	75.29
7	T5<4> [3-5,20,D;1-23,17,T;5-23,19,T;3-17,22,D]	0.10	Novel	yes	90.49
8	T5<4> [1-23,12,T;3-23,18,T;3-5,25,D;3-5,19,T]	0.10	Novel	yes	61.90
9	T5<4> [1-23,12,T;3-23,18,T;3-5,20,D;3-18,24,D]	0.10	Novel	yes	60.56
10	T5<4> [1-23,17,T;3-5,20,D;5-23,19,T;3-5,25,D]	0.10	Novel	yes	75.44

Table A.6: Task 6 – Generated structural sequences.

#	Text string	T	Novel/Seen	Admissible	Objective ratio (%)
1	T6<4>[1-49,26,D;1-49,32,T;7-49,33,T;33-49,48,T]	0.01	Novel	yes	64.71
2	T6<4>[1-49,18,D;1-49,24,T;7-49,8,T;7-18,8,D]	0.10	Novel	no	–
3	T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-49,33,T]	0.10	Novel	no	–
4	T6<4>[7-49,26,D;1-49,32,T;7-49,33,T;33-49,48,T]	0.10	Novel	yes	64.71
5	T6<4>[1-49,18,D;1-49,24,T;7-49,17,T;17-49,8,T]	0.10	Novel	no	–
6	T6<4>[7-49,26,D;1-49,32,T;7-49,25,T;25-49,33,T]	0.10	Novel	no	–
7	T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-49,32,T]	0.10	Novel	no	–
8	T6<4>[1-49,26,D;1-49,32,T;7-49,17,T;17-26,8,D]	0.10	Novel	yes	82.68
9	T6<4>[1-49,26,D;1-49,32,T;7-49,8,T;1-32,15,D]	0.10	Novel	yes	64.71
10	T6<4>[1-49,26,D;1-49,32,T;7-49,25,T;25-49,33,T]	0.10	Novel	no	–

B | Appendix B

This appendix contains the complete action sequences for the structures generated across the considered tasks.

Task 1

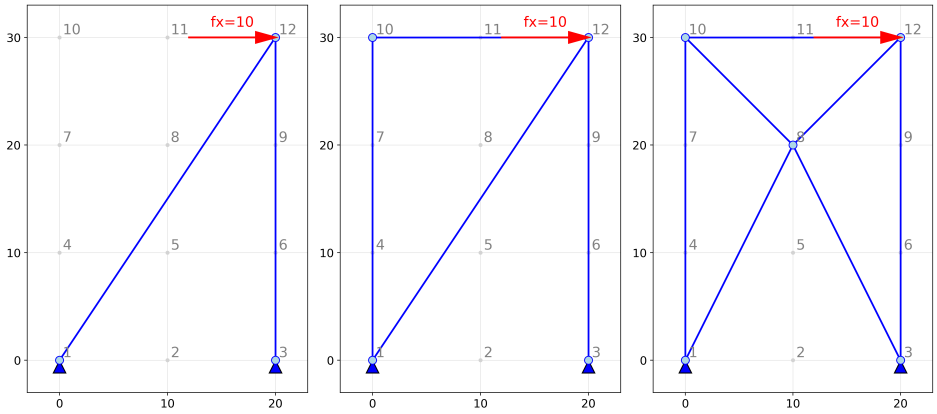

Figure B.1: Task 1-sequence of actions

Task 2

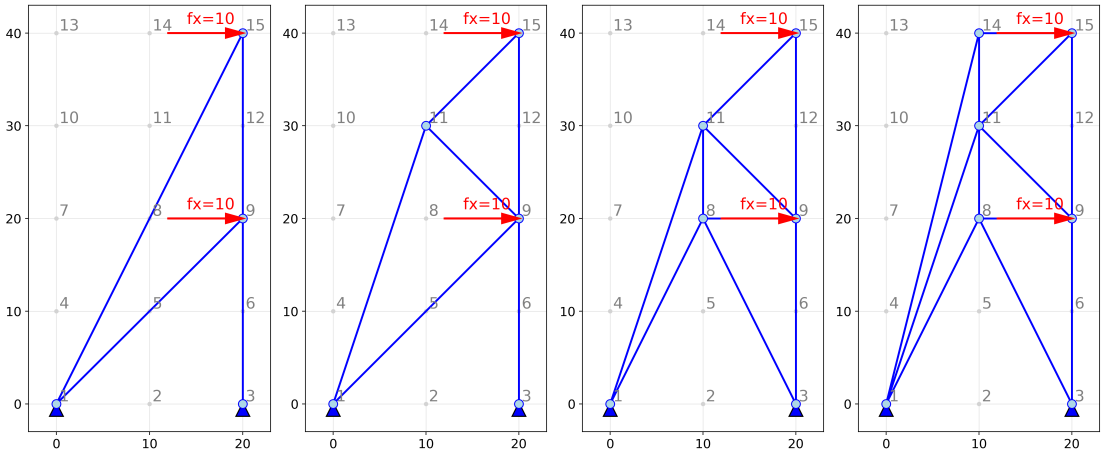

Figure B.2: Task 2-sequence of actions

Task 3

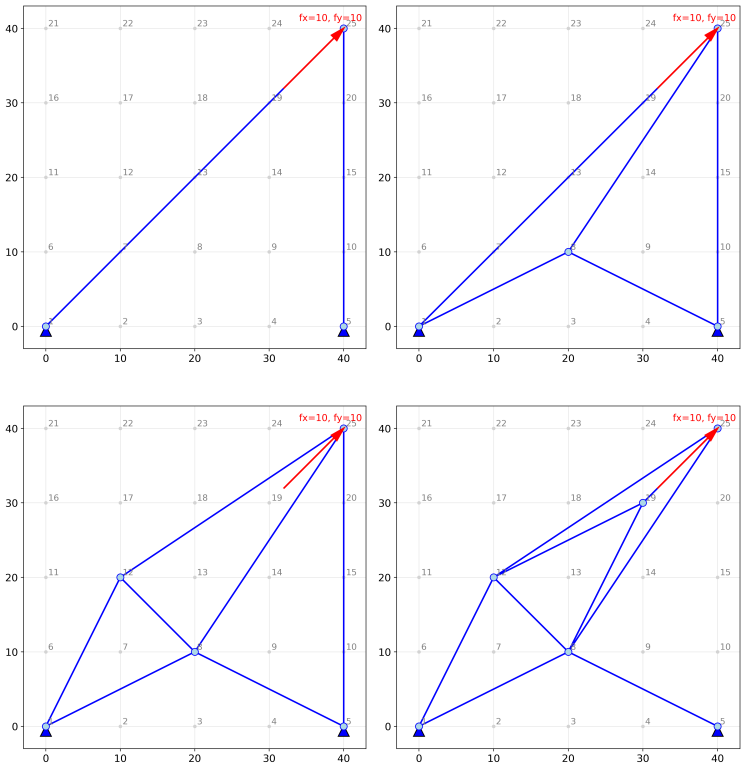

Figure B.3: Task 3-sequence of actions

Task 4

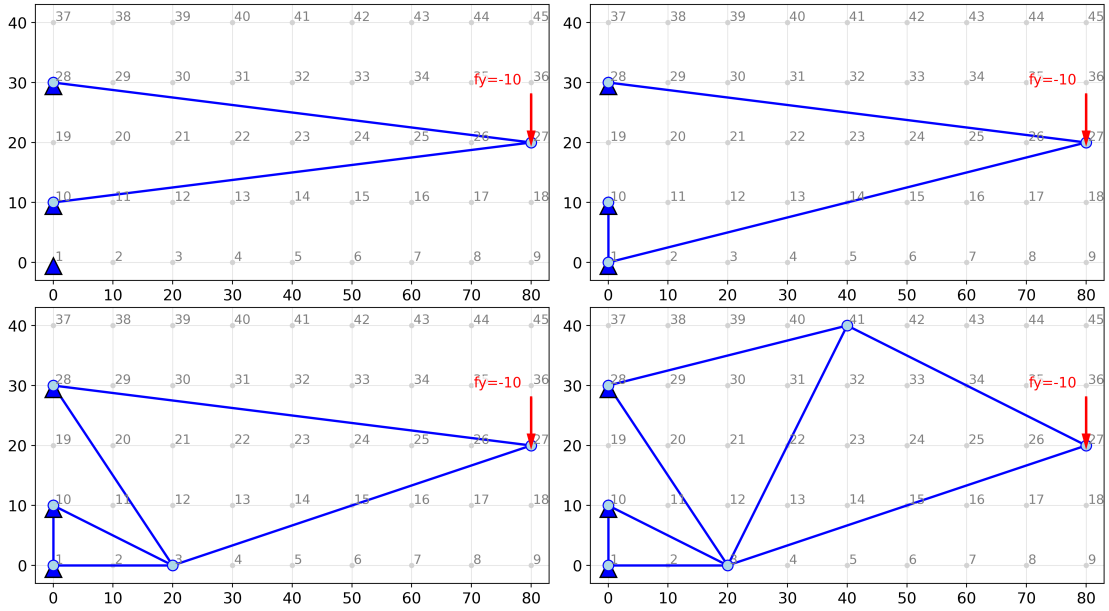

Figure B.4: Task 4-sequence of actions

Task 5

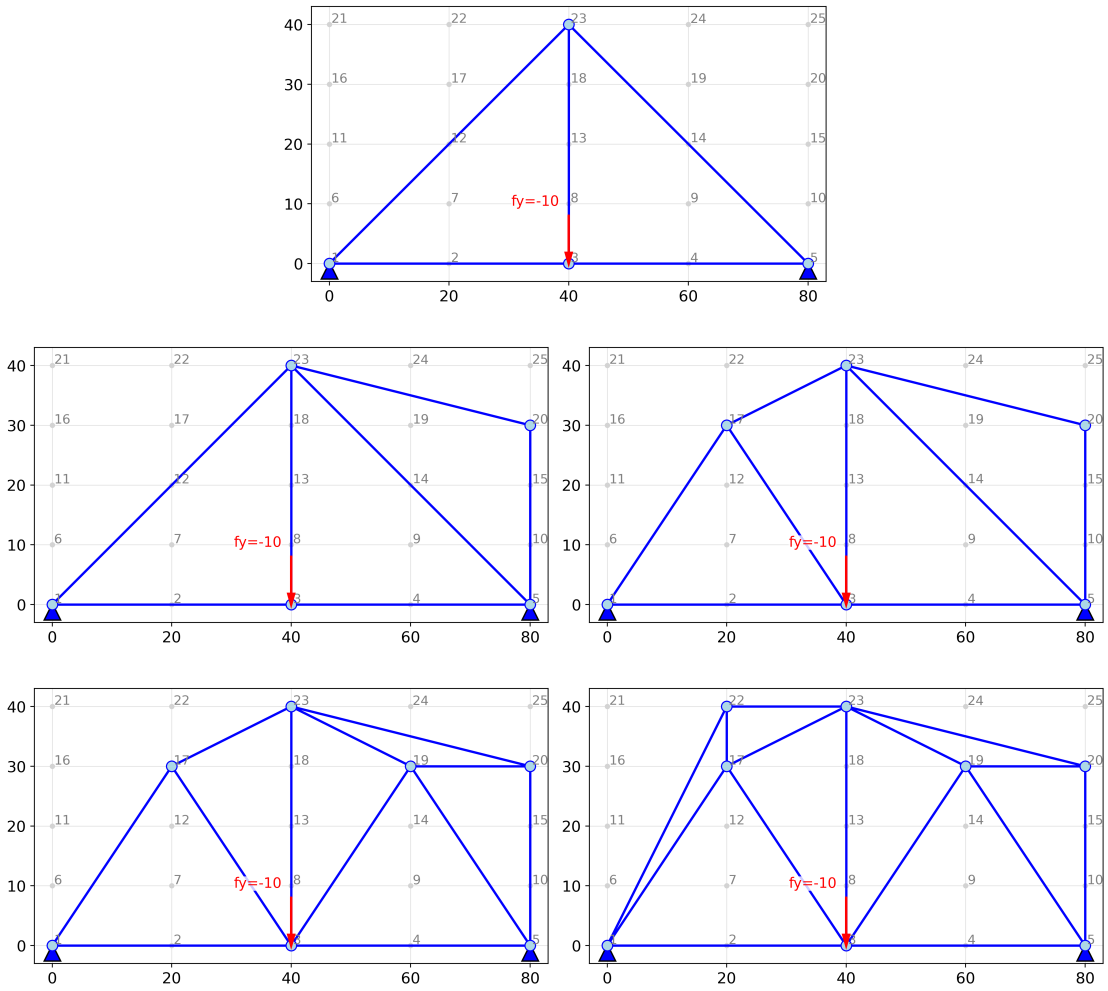

Figure B.5: Task 5-sequence of actions

Task 6

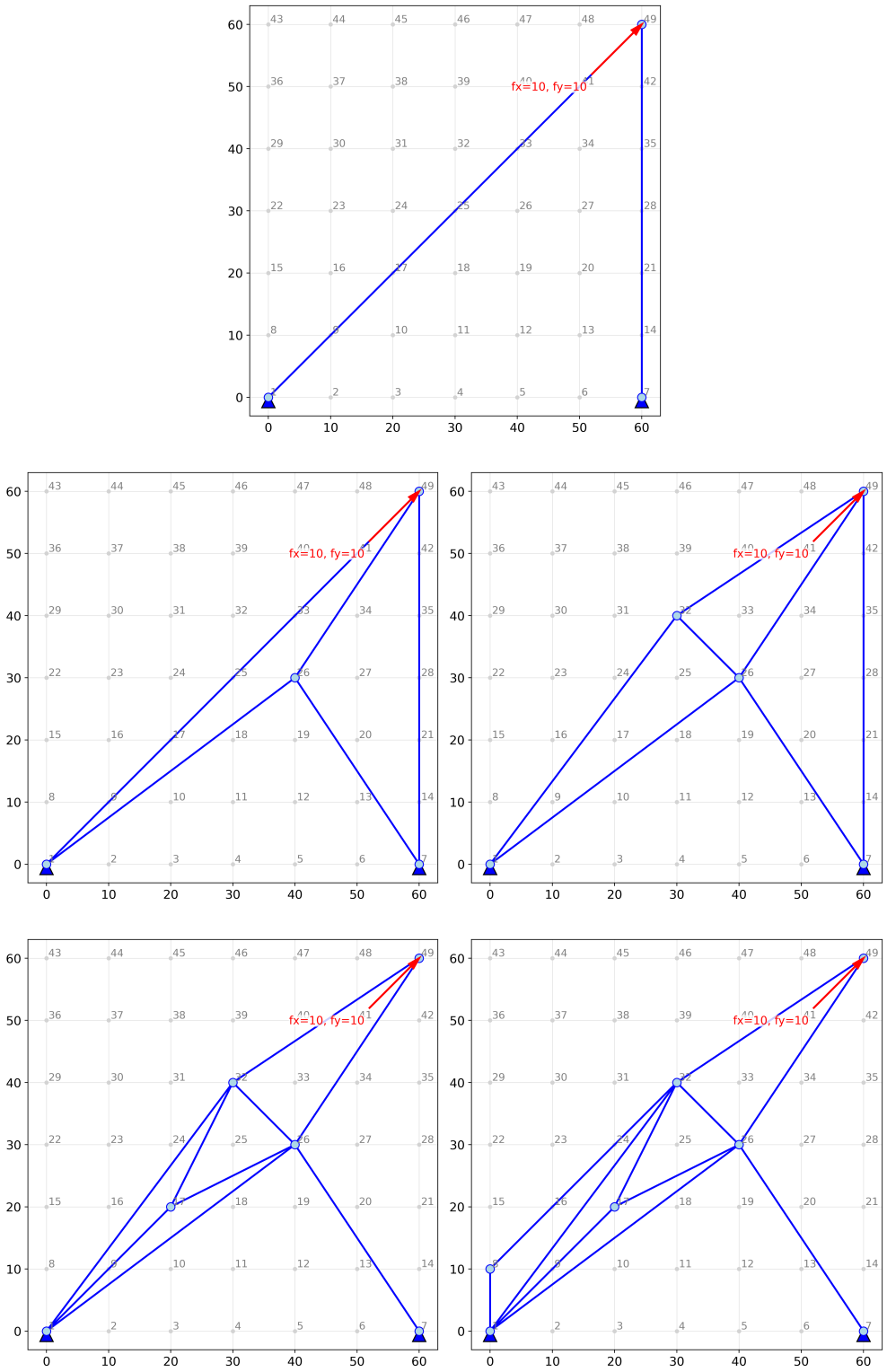

Figure B.6: Task 6-sequence of actions

C | Appendix C

Figures C.1–C.6 report the empirical distributions of the $U_{\max}$ values for all samples included in the datasets of the six tasks.

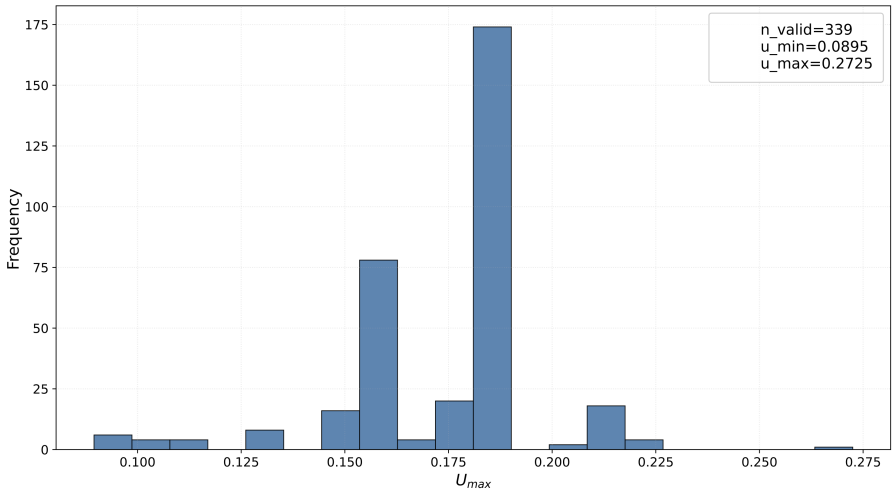

Figure C.1: Distribution of $U_{\max}$ values for Task 1.

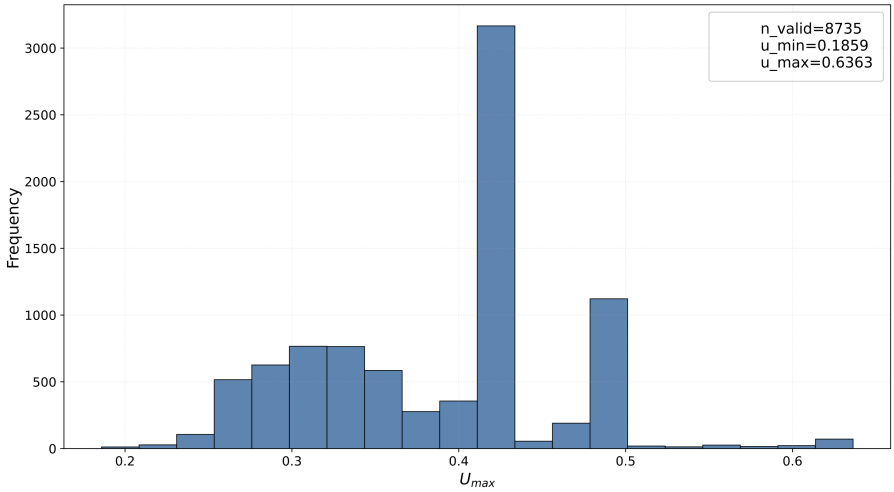

Figure C.2: Distribution of $U_{\max}$ values for Task 2.

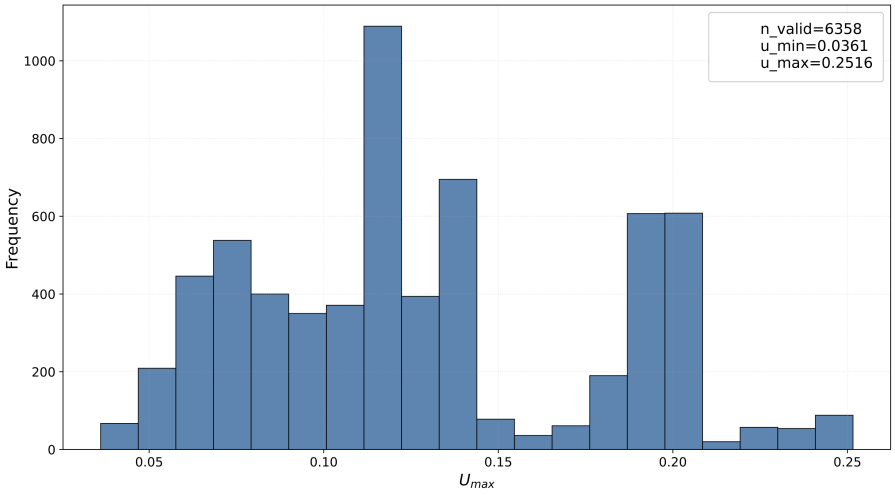

Figure C.3: Distribution of $U_{\max}$ values for Task 3.

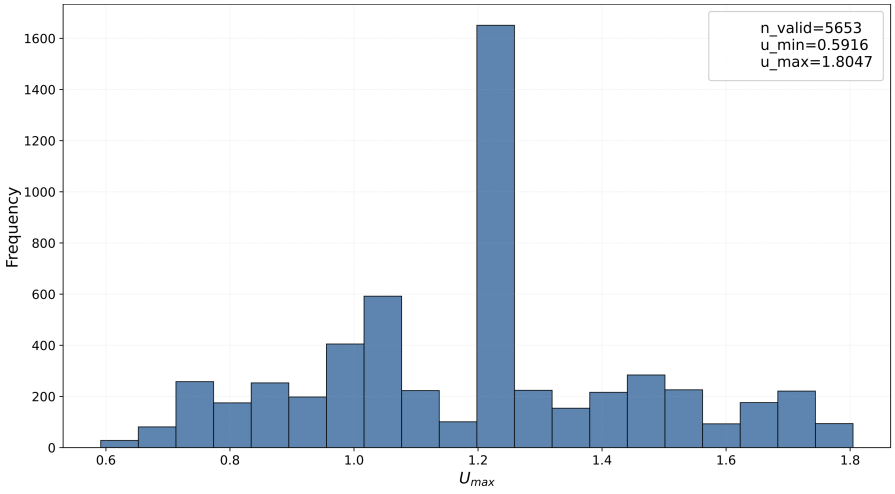

Figure C.4: Distribution of $U_{\max}$ values for Task 4.

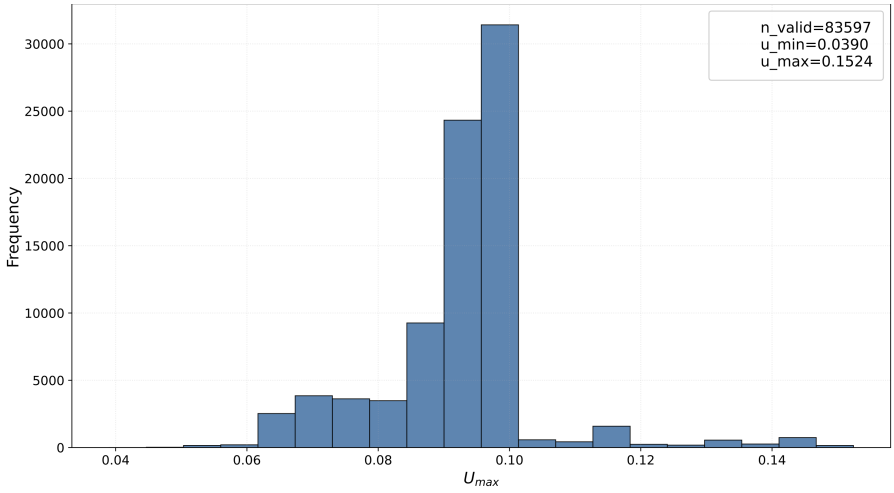

Figure C.5: Distribution of $U_{\max}$ values for Task 5.

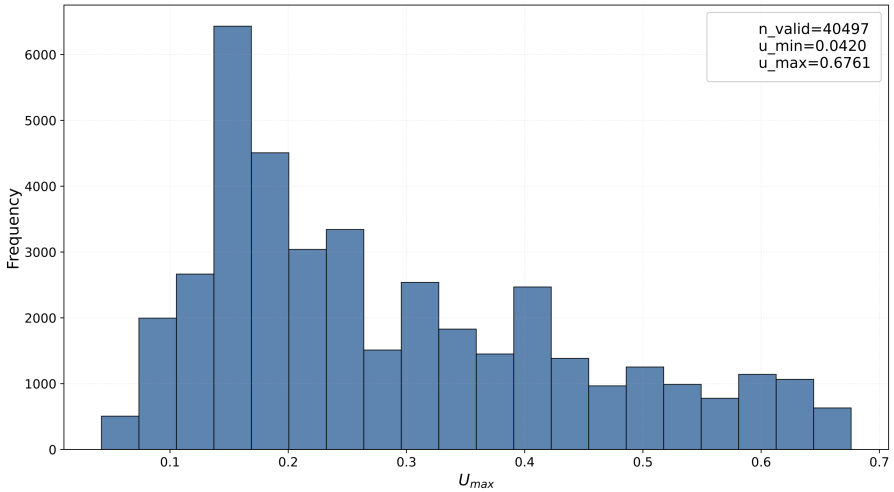

Figure C.6: Distribution of $U_{\max}$ values for Task 6.

List of Figures

List of Tables

List of Acronyms

Acronym	Full Term
AI	Artificial Intelligence
LLM	Large Language Model
NLP	Natural Language Processing
GPT	Generative Pretrained Transformer
ML	Machine Learning
DL	Deep Learning
NN	Neural Networks
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
RL	Reinforcement Learning
MDP	Markov Decision Process
GNN	Graph Neural Network
FEM	Finite Element Method
BPE	Byte Pair Encoding
FFN	Feed-Forward Network