

Deep Generative Model for Mechanical System Configuration Design

Yasaman Etesam^{1,2}, Hyunmin Cheong¹, Mohammadmehdi Ataei¹, Pradeep Kumar Jayaraman¹

¹Autodesk Research, Toronto, Ontario, Canada

²School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
yetesam@sfu.ca, {hyunmin.cheong, mehdi.ataei, pradeep.kumar.jayaraman}@autodesk.com

Abstract

Generative AI has made remarkable progress in addressing various design challenges. One prominent area where generative AI could bring significant value is in engineering design. In particular, selecting an optimal set of components and their interfaces to create a mechanical system that meets design requirements is one of the most challenging and time-consuming tasks for engineers. This configuration design task is inherently challenging due to its categorical nature, multiple design requirements a solution must satisfy, and the reliance on physics simulations for evaluating potential solutions. These characteristics entail solving a combinatorial optimization problem with multiple constraints involving black-box functions. To address this challenge, we propose a deep generative model to predict the optimal combination of components and interfaces for a given design problem. To demonstrate our approach, we solve a gear train synthesis problem by first creating a synthetic dataset using a domain-specific language, a parts catalogue, and a physics simulator. We then train a Transformer-based model using this dataset, named *GearFormer*, which can not only generate quality solutions on its own, but also augment traditional search methods such as an evolutionary algorithm and Monte Carlo tree search. We show that *GearFormer* outperforms such search methods on their own in terms of satisfying the specified design requirements with orders of magnitude faster generation time. Additionally, we showcase the benefit of hybrid methods that leverage both *GearFormer* and search methods, which further improve the quality of the solutions.

Introduction

Configuration design of mechanical systems in engineering (Mittal and Frayman 1989; Wielinga and Schreiber 1997) is a time-consuming task that relies on the domain expertise of engineers. Typically, engineers manually select the optimal combination of components to meet multiple design requirements based on their experience and knowledge, often leading to sub-optimal solutions that negatively impacts product development (Suh 1990; Dym 1994).

Computationally, configuration design can be defined as a combinatorial optimization problem with categorical design variables (e.g., component choices) and design requirements expressed as objectives and constraints (Levin 2009).

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Evaluating these objectives or constraints frequently necessitates the use of black-box physics simulation tools. Due to these aspects, configuration design problems have traditionally been approached using gradient-free, meta-heuristic optimization methods like evolutionary algorithms (Deb and Jain 2003; Angelov et al. 2003; Grignon and Fadel 2004; Cheong et al. 2019) or simulated annealing (Schmidt and Cagan 1998; Shea, Cagan, and Fenves 1997). In certain problems, search methods such as Monte Carlo tree search (Zhao et al. 2020; Luo et al. 2022) or heuristic search (Campbell, Cagan, and Kotovsky 1998; Piacentini et al. 2020; Zhao et al. 2020) have been employed. While these methods have shown some success, they are computationally expensive and time-consuming, limiting their practical adoption by engineers who ideally seek real-time recommendation of solutions to perform multiple design iterations within constrained time-frames (Cagan and Vogel 2002).

The need for a fast, interactive tool motivated our investigation into deep generative models for mechanical system configuration design. Recent progress in generative models, particularly Transformers (Vaswani et al. 2017), offers a promising approach to the problem. Originally designed for natural language processing, Transformers excel at generating complex sequences and have proven effective in solving combinatorial problems, such as AlphaFold’s protein structure predictions (Varadi et al. 2022; Jumper et al. 2021). In contrast to traditional methods, these generative models can instantly predict solutions given the user input.

Mechanical systems have *domain-specific languages* similar to the natural language, consisting of grammar and lexicon. Just as sentences must follow syntactic rules with comprehensible words, valid mechanical configurations must consist of specific parts that adhere to compatibility constraints. For example, constructing a basic mechanical linkage involves a sequence such as [pivot point] - [connecting rod] - [pin joint]. Each component must be compatible with the next for the linkage to be physically realizable and function correctly. While this is a simple example, more complex systems follow similarly structured rules. This parallel further motivated us to investigate applying Transformer models for generating valid and functional mechanical designs.

To explore this premise, a novel Transformer-based generative model is developed for gear train synthesis. Gear trains are complex and ubiquitous mechanical systems, consisting

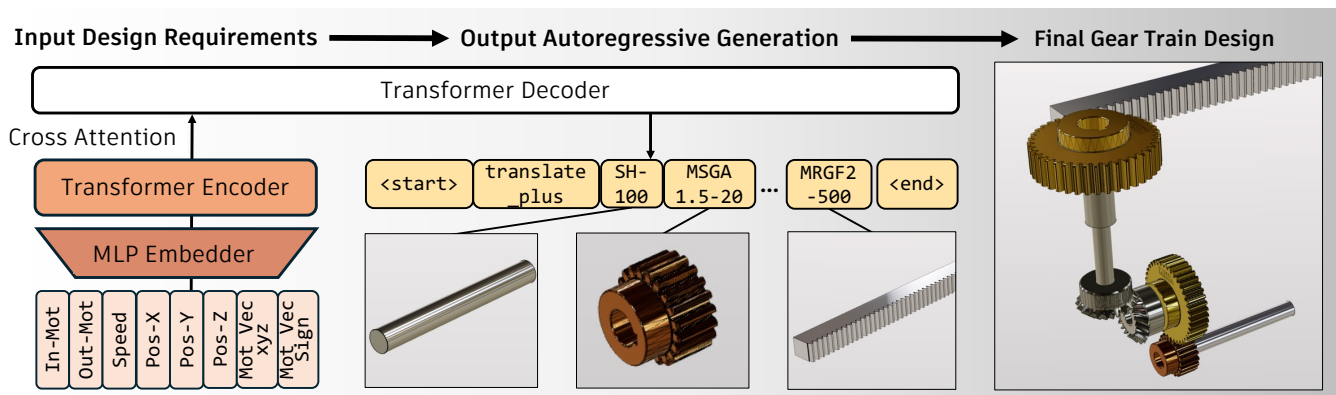


Figure 1: GearFormer is based on the Transformer architecture. It includes an encoder module that processes the input design requirements that have been embedded by a multi-layer perceptron (MLP). These input embeddings are consumed by the Transformer decoder module via cross-attention to generate the gear train sequence that satisfies the requirements.

of various types of components and interfaces that transmit motion while adjusting speed or torque. Designing gear trains that satisfy multiple requirements demands significant domain expertise and time, thus serves as an ideal problem for tackling the challenges of configuration design. To train such a Transformer-based model, we developed a method to synthetically generate datasets for mechanical configuration design problems because such datasets are not publicly available. Our main contributions are as follows:

- The introduction of a Transformer-based model named *GearFormer* (Figure 1), to solve an archetypal mechanical configuration design problem in gear train synthesis.
- Augmenting traditional search methods such as Estimation of Distribution Algorithm (EDA) and Monte Carlo tree search (MCTS) with GearFormer, e.g., Figure 2, to obtain higher quality solutions than their own.
- The creation of the GearFormer dataset, the first dataset for gear train synthesis, created with a domain-specific language and augmented with physics simulations.
- The development of a physics simulator capable of evaluating multiple requirements of a gear train design.

We show that GearFormer efficiently generates high-quality designs (e.g., Figure 3) that meet design requirements compared to traditional search methods. Moreover, we demonstrate that it can augment search methods to produce better solutions in a shorter amount of time than their own. A link to our code, data, and supplementary material referenced in this paper can be found at <https://gearformer.github.io/>. The website also features video demos that illustrate how GearFormer can be used in design workflows.

Related Work

Configuration design has been a subject of extensive research in engineering design automation. Various computational approaches have been developed to find optimal component combinations that satisfy design requirements, including evolutionary methods. Deb and Jain (2003) proposed a multi-objective evolutionary algorithm for solving complex engineering design problems. Angelov et al.

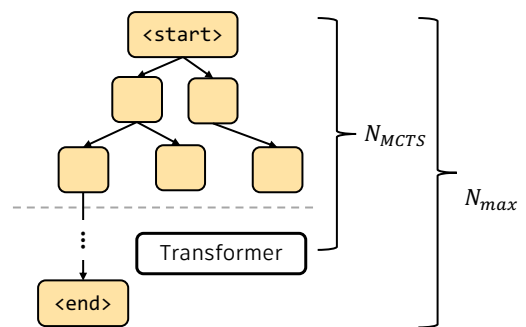


Figure 2: A hybrid method that combines MCTS to explore the first few critical tokens in a sequence and a Transformer-based model to complete the sequence for evaluation.

(2003) developed a fuzzy rule-based evolutionary approach for robot configuration design. Grignon and Fadel (2004) applied genetic algorithms to optimize product family designs. Cheong et al. (2019) used EDAs for configuration design of suspension systems.

Tree search methods have also shown promise in solving configuration design problems. Campbell, Cagan, and Kotovsky (1998) introduced an agent-based approach using heuristics for conceptual design. Piacentini et al. (2020) applied heuristic search to a multi-speed gearbox design problem. Zhao et al. (2020) developed a graph grammar-based approach with MCTS for robot design. Luo et al. (2022) employed MCTS for optimizing truss structures.

Incorporating design knowledge in the form of grammars or heuristics has been a key strategy to improve the efficiency and effectiveness of configuration design methods. Schmidt and Cagan (1998) used simulated annealing with a grammar-based representation for mechanical configuration design. Shea, Cagan, and Fenves (1997) applied shape grammars to generate structural designs. Piacentini et al. (2020) and Zhao et al. (2020) also incorporated design knowledge through heuristics and grammars in their approaches.

Recent advances in generative models have primarily fo-

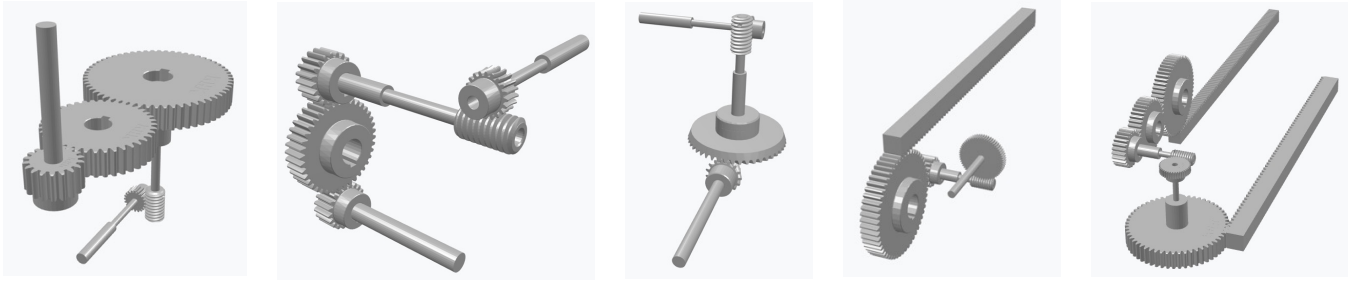


Figure 3: Gear train designs generated with GearFormer

cused on design ideation, which typically precedes configuration design. Regenwetter, Nobari, and Ahmed (2022) reviewed deep generative models in engineering, while Chen et al. (2024b) and Ataei et al. (2024) explored large language models and ontologies for conceptual generation and requirements elicitation. Chen et al. (2024a) integrated generative models for early-stage design. Research into AI-augmented design (Thoring, Huettemann, and Mueller 2023) and sketch-based inputs (Zhang et al. 2023) with tools like DesignAID (Cai et al. 2023) advance the design ideation process but are still limited in practical configuration design.

Problem

Configuration Design of Mechanical Systems

We address a class of configuration design problems where the design solution can be represented as a sequence. For example, designing an open-loop kinematic mechanism with single input and single output motions/forces would fall under this category. A design solution can be expressed as:

$$X = (\mathcal{S}, C_1, I_{1,2}, C_2, \dots, C_i, I_{i,i+1}, C_{i+1}, \dots, C_N, \Sigma) \quad (1)$$

where \mathcal{S} is a start symbol, C_i are components, $I_{i,i+1}$ are interfaces between the preceding and the following components, Σ is a terminal symbol, and N is the number of components in the sequence. The possible values of C_i are typically based on the list of off-the-shelf components available to the engineer and the possible values of $I_{i,i+1}$ are based on the allowed interfacing between C_i and C_{i+1} . X must be chosen such that it conforms to a set of rules, \mathcal{R} . For example, if C_1 was a rack component, C_2 must be a spur gear component that can mesh with C_1 and $I_{1,2}$ must convey the corresponding meshing. The possible values and set of rules for a particular application domain constitute the lexicon and grammar of a *domain-specific language* (DSL).

Solving a configuration design problem involves finding an optimal sequence X that minimizes a particular objective function while satisfying multiple design requirements posed as constraints. In general, evaluating the objective function or constraints involves a black-box solver such as a physics simulator. Due to the combinatorial nature of the problem and non-differential functions, solving the problem with traditional computational methods is challenging.

Gear Train Synthesis Problem

To investigate solving a configuration design problem with a generative model approach, we focus on the gear train syn-

thesis problem. A gear train with single input/output motions can be represented as a sequence defined in Eq. (1), an enumerated collection of gear components and their interfaces.

Design requirements considered for gear trains include the cost, weight, output speed/torque ratio, output motion position/vector, and input/output motion type conversion. We used the total weight of a gear train as our objective function, which also serves as a good proxy for the cost. Also, since a torque ratio is simply an inverse of a speed ratio for a gear train ignoring its efficiency, it was omitted for this problem. The rest of the requirements were posed as constraints. We also ensure that a gear train is physically realizable by conforming to the domain-specific grammar and having no interference among components. The optimization problem for gear train synthesis can be formulated as follows.

$$\begin{aligned} \min_X \quad & f_w = \sum_{i=1}^N (w_{C_i}), \quad C_i \in X \\ \text{s.t.} \quad & \tilde{s} - s(X) = 0; \quad |\tilde{p} - \vec{p}(X)| = 0 \\ & \tilde{m} \cdot \vec{m}(X) - 1 = 0 \\ & \tilde{\tau}_{in} - \tau_{in}(X) = 0; \quad \tilde{\tau}_{out} - \tau_{out}(X) = 0 \\ & g_{\text{grammar}}(X) = 0; \quad g_{\text{interfere}}(X) = 0 \end{aligned} \quad (2)$$

f_w is the total weight of the gear train, w_{C_i} is the weight of each component C_i that is a member of the gear train sequence X , s is the speed ratio, \vec{p} is the output position in x, y, z coordinates, \vec{m} is the output motion vector where the nonzero index indicates the coordinate and its sign indicates the direction, τ_{in} is the input motion type, and τ_{out} is the output motion type. Those denoted with *tilde* are targets provided by the engineer. Operators g_{grammar} and $g_{\text{interfere}}$ return 0 if X conforms to the grammar and does not have interference among parts, respectively. The rest of operators that are functions of X are evaluated using a physics-solver.

Proposed Approach

Transformer-based Model

To solve the configuration design problem, we propose a generative model that takes the representation of the design requirement constraints as an input encoding vector \mathcal{E} and outputs a sequence X conditioned based on that input while also minimizing the objective function $f(X)$. In other words, we aim to find optimal parameters θ for a generative

sequential model such as a Transformer

$$X_i = \text{Transformer}(\mathcal{E}_i, \theta) \quad (3)$$

that minimizes the loss function

$$\mathcal{L}(X, X^t) = \sum_{i=1}^{|D|} -\mathcal{L}_{\text{pred}}(X_i, X_i^t) - \alpha f(X_i), \quad (4)$$

where the first term is a measure of the model’s ability to generate the expected ground-truth sequence X^t and the second term corresponds to the design objective, weighted with α . The model is trained using a dataset $D = \{X_i, X_i^t\}$.

Note that the generative model could not only be used to predict a solution to the problem on its own but also used in conjunction with traditional search methods to efficiently solve the problem. To this end, we propose hybrid methods by combining our Transformer-based model with two distinct search methods: Estimation of distribution algorithm (EDA) and Monte Carlo tree search (MCTS).

Hybrid Methods

EDA+Transformer EDA is an evolutionary algorithm that uses a probabilistic model to iteratively sample and improve a population of solutions (Larrañaga and Lozano 2012). We introduce a hybrid approach that combines EDA with a Transformer-based model to leverage the strengths of the two methods. In this method, EDA is used to explore the first few tokens of the sequence, which have the strongest influence on the whole sequence generation. Given the maximum sequence length of N , the method limits the sequence length considered by EDA to $N_{\text{EDA}} < N$ and then uses a Transformer-based model to complete the subsequent tokens before evaluating each candidate solution.

We use a bi-gram probabilistic model for EDA:

$$P(x_{1:N}) = \prod_{i=1}^N P(x_i | x_{i-1}) \quad (5)$$

where x_1 is the start token and subsequent tokens x_i are generated using the conditional probability based on the previous token x_{i-1} .

MCTS+Transformer MCTS is a heuristic search algorithm that balances the exploration of under-explored search paths with the exploitation of promising paths ((Coulom 2006; Kocsis and Szepesvári 2006)). Similar to the first method, we combine MCTS with a Transformer-based model to create a hybrid method. We limit the depth of the search tree explored by MCTS to $N_{\text{MCTS}} < N$, as MCTS is used to explore the critical initial tokens, while a Transformer-based model is used to complete the subsequent tokens before evaluating each candidate solution.

MCTS uses a heuristic function to decide which child node (token) to expand during the search, typically:

$$h = \frac{R_i}{v_i} + c \sqrt{\frac{\ln V_i}{v_i}} \quad (6)$$

where R_i is the accumulated reward for the i -th child, v_i is the number of visits made at the child, V_i is the total number

of visits made at the parent, and c is an exploration parameter (set to 1.4 in our case).

Both hybrid methods combine the strengths of search algorithms (EDA and MCTS), namely their exploration capabilities, with a Transformer-based model to generate high-quality solutions, potentially leading to more efficient exploration of the search space and better overall results.

Domain-Specific Language

To represent valid gear train designs and help constrain the search space, a domain-specific language (DSL) is developed. A valid gear train sequence must conform to the grammar and lexicon of the DSL as follows.

Grammar

We define $\mathcal{S} = \langle \text{start} \rangle$ and $\Sigma = \langle \text{end} \rangle$. The set of variables enumerate the types of gear components and interfaces (the last two in the following list) considered, $\mathcal{V} = \{ \text{Shaft, Rack, Spur gear, Bevel gear, Miter gear, Worm gear, Hypoid gear, Translate, Mesh} \}$. The simplified set of rules \mathcal{R} is shown in Table 1, while the full grammar can be found in the supplementary material.

Lexicon

The lexicon defines the possible tokens for each variable. For each component type, we assume that two to twelve different parts are available for an engineer (Table 2), based on a gear component manufacturer’s catalogue¹.

For the interface variable “Translate”, we define a pair of translation tokens $tra+$ and $tra-$. They indicate toward which direction a shaft should be placed. The $+$ or $-$ sign indicates the direction relative to the current orientation. For example, if the current component is a spur gear oriented toward $[1, 0, 0]$, $tra+$ would represent the following shaft being placed toward $[1, 0, 0]$ while $tra-$ toward $[-1, 0, 0]$.

Finally, we define a set of meshing tokens $\{(\perp^1, +), (\perp^1, -), (\perp^2, +), (\perp^2, -)\}$ for “Mesh”, which encode how the next gear component is placed relative to the current component (Figure 4). The first element in the token tuple indicates along which axis the next component should be placed relative to the motion axis of the current component. Then, the second element indicates whether to place the component in the positive or negative direction. So for example, given a gear component with the motion axis of $[0, 0, 1]$, applying the token $(\perp^1, -)$ would result in the next component being placed along the direction of $[-1, 0, 0]$.

In total, the lexicon size is 52, consisting of 44 gear components, 6 interface tokens, and start and end tokens.

Dataset Generation

An important contribution of our work is a method to create the synthetic dataset required to train a Transformer model, named *GearFormer* dataset, because there lacks any existing dataset of gear train designs. The dataset includes sequences that can be mapped to real-life gear trains paired with multiple requirement metrics evaluated with a physics simulator.

¹https://khkgears.net/new/gear_catalog.html

LHS	RHS
start	Translate-Shaft Rack-Mesh-Spur gear
Shaft	$Gear^\dagger$ -Mesh- $Gear$ Spur gear-Mesh-Rack end
Spur gear	Mesh-Spur gear Translate-Shaft end
Rack	end
$Gear$	Translate-Shaft end

Table 1: Simplified gear train grammar. LHS and RHS stand for left-hand side and right-hand side, respectively. $^\dagger Gear$ represents multiple component types listed in Table 2 excluding racks. Each variable type can be expanded with either part tokens for the component types or translation/meshing tokens for the interface types.

Components	Part numbers / tokens
Shafts †	SH-(*, 100, 200, 300, 400, 500)
Racks	MRGF(1.5, 2, 2.5, 3)-500
Spur gears	MSG1.5-(20, 40, 60, 80), MSG2-(18, 25, 40, 60), MSG2.5-(15, 40, 55, 70), MSGA3-(15, 30, 45, 60)
Bevel gears	SBSG2-(3020R, 2030L, 4020R, 2040L, 4515R, 1545L)
Miter gears	MMSG2-20(R,L)
Worm gears	<i>Worm</i> : SWG1-R1 <i>Wheels</i> : AG1-(20R1, 40R1, 60R1)
Hypoid gears	<i>Pinions</i> : MHP1-(3045L, 2060L, 1045L) <i>Rings</i> : MHP1-(0453R, 0602R, 0451R)

Table 2: Part numbers (used as tokens) considered for each component type obtained from the commercial catalogue. Abbreviations are used to combine similar part numbers with a consistent prefix; the numbers in parentheses indicate variations of the base part number. † Shafts are assumed to be pre-cut into different lengths at 100mm, 200mm, etc., denoted by the part number, or to place two gear components immediately side-by-side in the case of SH-*

Generate variable sequences Using the grammar in Table 1 as production rules, we generated variable sequences having maximum 10 components to bound the problem but still produce realistic gear trains. A variable sequence with a maximum of 10 components can have up to 21 tokens, including the interface variables and the start/terminal tokens. An example variable sequence with five components is: (<start>, Rack (1), Mesh, Spur Gear (2), Translate, Shaft (3), Bevel Gear (4), Mesh, Bevel Gear (5), <end>). In total, 37,606 unique variable sequences are generated.

Generate token sequences For each variable in a variable sequence, one of the possible tokens is randomly chosen from our lexicon list. This generates a token sequence that resembles a real gear train design. For the variable sequence example above, a possible token sequence is (<start>, MRGF2-500, $(\perp^2, -)$, MSGA2-40, $tra-$, SH-200, SBSG2-3020R, $(\perp^1, +)$, SBSG2-2030L, <end>).

We discard sequences that are not physically feasible (i.e., at least two components interfere with each other) as explained in the following subsection. We generated a total of

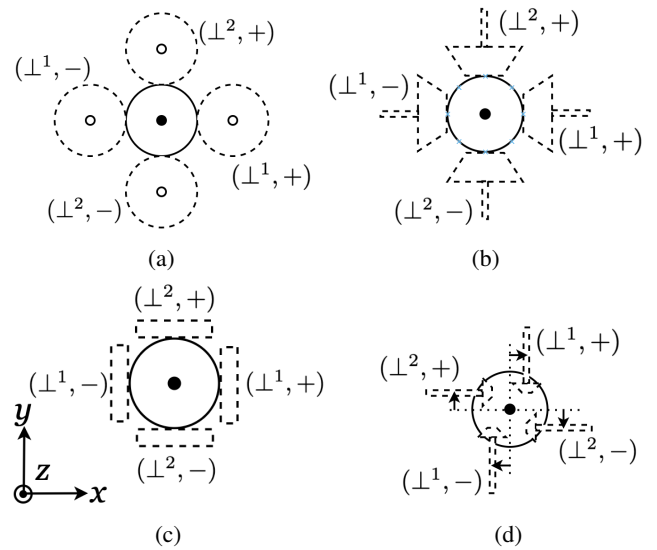


Figure 4: Examples of gear pair placements for each meshing token. (a) Spur gears. (b) Miter/Bevel gears. (c) Rack-and-pinion and worm gears. (d) Hypoid gears.

7,363,640 grammatically valid and feasible sequences. Out of these, 0.05% (3,681) were randomly selected for validation, another 0.05% for testing, and the remaining sequences for training. Relatively small validation and test sets are used because each predicted sequence during validation/testing must be evaluated with a simulator to compute the actual requirement metrics, which can be time-consuming.

Physical feasibility check To verify the physical feasibility, we check for interference between all possible pairs of components in the sequence. We use the dimensions from the commercial catalogue to define a bounding box for each component, while keeping track of its current position and orientation. We check for intersections between the bounding box of the current component and those of the previous components, except for the one immediately preceding it, as they are supposed to be connected.

Input encoding vector We construct a vector of the desired requirements with the following elements, referring to Eq. (2): 1) $\tilde{\tau}_{in} = 1$ for translation or 0 for rotation, 2) $\tilde{\tau}_{out} = 1$ for translation or 0 for rotation, 3) $\tilde{s} \in \mathbb{R}$, 4-6) $\tilde{p} \in \mathbb{R}^3$, 7) the nonzero element index of \tilde{m} as 0, 1, or 2, and 8) the sign of the nonzero element of \tilde{m} as 1 or -1.

Gear train simulator We implemented a simulator using Dymos² to compute all the requirement metrics given a gear train sequence. Dymos is an open-source Python library that enables simulation of time-dependent systems. The kinematics and spatial computations for each component type required were implemented as a Dymos *component*. Given a sequence, a gear train system (defined as a *group* in Dymos) is composed on-the-fly by instantiating the components cor-

²<https://github.com/OpenMDAO/dymos>

responding to the parts in the sequence and connecting them based on the interface information.

GearFormer for Gear Train Synthesis

Model architecture GearFormer (Figure 1) is based on the standard Transformer architecture (Vaswani et al. 2017). It includes a bidirectional encoder module that takes an input vector that encodes the design requirements, which are embedded with linear layers, ReLU activations, and batch normalization. The output of this encoder serves as context for the autoregressive decoder and is consumed via cross-attention. The decoder then predicts a sequence of output tokens, which represents a valid and feasible gear train design meeting the specified requirements while also minimizing the weight objective. Example designs generated with GearFormer are shown in Figure 3.

Loss function Predicting each token of the sequence is a classification task that attempts to select the best class from a vocabulary of 53 tokens (52 tokens from the lexicon plus one for the end-of-sentence token), given the previous tokens and the context (design requirements). Hence, we use a standard cross-entropy loss for $\mathcal{L}_{\text{pred}}$ in Eq. 4. We also include the weight of the gear train as the design objective loss and experiment with different ratios of α .

$$\mathcal{L}(p(X), p(X^t)) = \sum_{i=1}^{|D|} -p(X_i^t) \log p(X_i) + \alpha f_w \quad (7)$$

Computing the weight objective with Gumbel-Softmax

Evaluating the weight objective in the training loop requires the ability to differentially sample the output sequence from the model, and fetch the corresponding component weights from the catalogue. To achieve this, we first extract the component weights from the gear catalogue, and store them in a coefficient vector W_{coef} . Then, we use the straight-through Gumbel-Softmax (Jang, Gu, and Poole 2016) trick in the autoregressive sampling loop to generate each token in the sequence as a one-hot vector $s_i \in \{0, 1\}^{53}$. By computing the dot product $s_i \cdot W_{\text{coef}}$, we can retrieve the weight of the component represented by the token. Repeating this for the entire output sequence and summing the weights gives us the value of the weight objective function. This technique is quite general and can be used for any set of attributes associated to the tokens in the vocabulary. Note that we calculate the weight of the shafts based on their length, assuming the material is carbon steel and the diameter of all the shafts is $0.01m$. We also assume that the weight of SH-* is $0kg$.

Adaptive loss weighting We set $\alpha = w_1 \cos(w_\epsilon)$, where w_1 is a model parameter and w_ϵ is determined based on the epoch number ϵ as $w_\epsilon = \max(0, \frac{\pi}{2} - (\epsilon - 1) \times \frac{\pi}{6})$ to bias the model on the cross entropy loss during the initial epochs.

Experiments

First we present the metrics used to evaluate GearFormer plus its training and selection details. We then compare GearFormer and our hybrid methods against search methods on their own with a benchmark problem set.

Evaluation Metrics

We evaluated the model using several metrics based on Eq. (2). Validity and feasibility were assessed for the entire test set, while the remaining metrics applied only to valid sequences, as the simulator produces outputs solely for those.

- **Validity (Valid):** The percentage of sequences conforming to the grammar.
- **Feasibility (Feas):** The percentage of sequences that are valid and non-interfering.
- **Output position match (Pos):** Average Euclidean distance between target and actual positions $|\tilde{p} - \bar{p}(X)|$.
- **Speed ratio match (Speed):** Average RMSLE($\tilde{s}, s(X)$), i.e., root mean squared logarithmic error to account for the wide range of speed ratios (10^{-5} – 10^5).
- **Output motion vector match (Mot Vec):** The percentage of $\tilde{m} \cdot \bar{m}(X) = 1$.
- **Input/output motion type match (In-Mot/Out-Mot):** The percentages of $\tilde{\tau}_{\text{in}} = \tau_{\text{in}}(X)$ and $\tilde{\tau}_{\text{out}} = \tau_{\text{out}}(X)$.
- **Weight:** The average of f_w .

GearFormer Model Training and Selection

We implemented our model using the x-transformers library³ and PyTorch (Ansel et al. 2024). A Tesla V100-SXM2-16GB GPU was used for training and evaluation. Training was conducted for a maximum of 20 epochs, but stopped if the validation loss increased at any point. Each epoch requires over 200 seconds for evaluation with the simulator, resulting in a maximum of one hour per experiment. More details are provided in the supplementary materials.

Loss function We experimented with various values of (w_1) for our objective function weighting. As shown in Figure 5, as w_1 increases, the average weight of the generated sequences drops; however, the model generates fewer valid and feasible sequences. Note that the average weight for the ground truth sequences in the validation set is $6.72kg$. We chose $w_1 = 1.0$ for our model to prioritize generating valid and feasible sequences for our experiments.

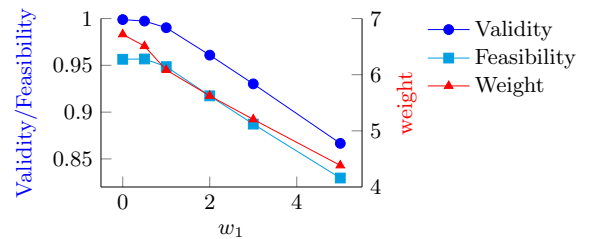


Figure 5: As we increase w_1 to prioritize the weight loss term, the average weight of the generated sequences drops at the expense of producing fewer valid and feasible sequences.

Model size We experimented with different values for dimension, depth, and attention heads on the validation set (see the supplementary materials). To balance having fewer parameters with achieving good results, we decided on the dimension of 512, depth of 6, and 8 attention heads.

³<https://github.com/lucidrains/x-transformers>

	Model	Valid \uparrow %	Feas \uparrow %	Pos \downarrow m	Speed \downarrow log(\cdot)	Mot Vec \uparrow %	In-Mot \uparrow %	Out-Mot \uparrow %	Weight \downarrow kg	Cand # \downarrow -
test	GearFormer	98.70	94.73	0.041	0.0229	99.04	100.0	99.92	6.25	-
	Rand	-	0.4766	0.860	1.9393	16.96	54.13	71.79	6.69	-
30 random set	GearFormer	96.67	93.33	0.045 \pm 0.047	0.0266 \pm 0.042	100.0	100.0	100.0	5.98 \pm 4.35	1
	EDA+GF	-	100.0	0.134 \pm 0.163	0.0098 \pm 0.016	83.33	100.0	100.0	3.26 \pm 3.59	10e3
	MCTS+GF	-	100.0	0.209 \pm 0.362	0.0275 \pm 0.047	86.67	100.0	100.0	2.62 \pm 2.19	10e3
	EDA	-	100.0	0.652 \pm 0.362	0.0930 \pm 0.132	80.00	100.0	100.0	1.98 \pm 1.55	10e4
	MCTS	-	100.0	0.760 \pm 0.400	0.2175 \pm 0.308	66.67	100.0	100.0	3.20 \pm 3.09	10e4

Table 3: We compared GearFormer against multiple baselines. The upper section of the table compares GearFormer with the random baseline (Rand) on the test set, where the latter generates random sequences that conform to the grammar. The lower section of the table compares the metrics of GearFormer, hybrid methods, and search methods on their own, evaluated with 30 randomly sampled benchmark problems. ‘Cand #’ stands for the number of candidate solutions considered to find the best solution used to compute the metrics.

GearFormer Against and With Search Methods

We evaluate GearFormer and the two hybrid methods developed, named EDA+GF and MCTS+GF, versus the search methods on their own. Because it is impractical to run search methods on the entire test dataset, we randomly sampled 30 benchmark problems from the test set. We set both $N_{\text{EDA}} = 6$ and $N_{\text{MCTS}} = 6$.

Results

The upper section of Table 3 compares the performance of GearFormer with the random baseline (Rand) method on the test set. The baseline method randomly generates sequences that conform to the grammar. GearFormer significantly outperforms the baseline across all metrics. The results for the baseline also highlight the difficulty of finding feasible solutions that address multiple design requirements.

The lower section of Table 3 shows that GearFormer outperforms or is at par with EDA and MCTS in four out of the five design requirements. Because the search methods are specifically designed to only generate valid sequences and can pick only the feasible solutions out of multiple candidates, they achieve 100% accuracy on those metrics. They can also find more lightweight solutions, at the expense of not addressing the design requirements. It should be emphasized that GearFormer can instantly generate a high-quality solution, compared to the search methods that selected the best solution out of 10,000 candidate solutions evaluated.

Table 3 also highlights the benefit of our hybrid methods. Compared to the search methods on their own, all design requirement metrics improved with the hybrid methods (in fact, the speed metric is better than GearFormer’s). This came at the expense of the weight objective for EDA+GF but for MCTS+GF, the weight objective even improved slightly. Lastly, the hybrid methods only used 1/10th of the candidate solutions compared to the search methods on their own to achieve such results, demonstrating the improved efficiency.

Note that the average inference time for GearFormer to generate a single candidate solution were 0.328s. For the search methods, the average evaluation time for a single candidate solution was 0.039s (which includes simulation), or about 6.5 minutes to perform a single run of EDA or MCTS.

Discussion and Conclusions

Our work has showcased the possibility of solving a challenging mechanical configuration design problem, particularly gear train synthesis, with a Transformer-based model named GearFormer. GearFormer not only generates solutions instantly but also outperforms traditional search methods in finding quality solutions that meet design requirements. This is tremendously valuable in assisting engineers to quickly explore multiple solutions within a shorter amount of time. Additionally, GearFormer could be used to auto-complete a partial gear train design provided by the engineer, either by suggesting or ranking potential lists of subsequent components in real-time. This enables an interactive workflow between the engineer and the tool that is unattainable with traditional search methods.

We also introduce hybrid search methods by combining EDA and MCTS with GearFormer. These methods leverage the explorative capability of search methods for early decisions during sequence generation and the generative model’s ability to complete the remaining decisions toward high-quality solutions. The hybrid methods found solutions that better address design requirements than the search methods on their own within a shorter amount of time.

Another important contribution of our work is the introduction of the first dataset on gear train synthesis. We developed a domain-specific language (DSL) and a physics simulator that can be used to generate more datasets in the future. Our language can be extended to incorporate additional grammar and lexicon to produce richer datasets that can be used to train a model for more complex gear train designs.

Finally, this work elucidates how a difficult engineering design problem can be formulated and solved using a deep generative model approach and provides a strong evidence that the approach can be effectively utilized in engineering design. The modular nature of our methodology allows for extension to a wide range of configuration design problems, such as hydraulic systems, modular robots, frame structures, etc., by leveraging appropriate DSLs and physics simulators for the respective application. We believe that many research opportunities exist in this space with notable implications for societal innovation.

References

- Angelov, P. P.; Zhang, Y.; Wright, J. A.; Hanby, V. I.; and Buswell, R. A. 2003. Automatic design synthesis and optimization of component-based systems by evolutionary algorithms. In *Genetic and Evolutionary Computation—GECCO 2003: Genetic and Evolutionary Computation Conference Chicago, IL, USA, July 12–16, 2003 Proceedings, Part II*, 1938–1950. Springer.
- Ansel, J.; Yang, E.; He, H.; Gimelshein, N.; Jain, A.; Voznesensky, M.; Bao, B.; Bell, P.; Berard, D.; Burovski, E.; Chauhan, G.; Chourdia, A.; Constable, W.; Desmaison, A.; DeVito, Z.; Ellison, E.; Feng, W.; Gong, J.; Gschwind, M.; Hirsh, B.; Huang, S.; Kalambarkar, K.; Kirsch, L.; Lazos, M.; Lezcano, M.; Liang, Y.; Liang, J.; Lu, Y.; Luk, C.; Maher, B.; Pan, Y.; Puhersch, C.; Reso, M.; Saroufim, M.; Siraiichi, M. Y.; Suk, H.; Suo, M.; Tillet, P.; Wang, E.; Wang, X.; Wen, W.; Zhang, S.; Zhao, X.; Zhou, K.; Zou, R.; Mathews, A.; Chanan, G.; Wu, P.; and Chintala, S. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM.
- Ataei, M.; Cheong, H.; Grandi, D.; Wang, Y.; Morris, N.; and Tessier, A. 2024. Elicitor: An LLM Agent-Based Simulation Framework for Design Requirements Elicitation. *arXiv preprint arXiv:2404.16045*.
- Cagan, J.; and Vogel, C. M. 2002. *Creating Breakthrough Products: Innovation from Product Planning to Program Approval*. Ft Press.
- Cai, A.; Rick, S. R.; Heyman, J. L.; Zhang, Y.; Filipowicz, A.; Hong, M.; Klenk, M.; and Malone, T. 2023. DesignAID: Using Generative AI and Semantic Diversity for Design Inspiration. In *Proceedings of The ACM Collective Intelligence Conference*, 1–11.
- Campbell, M.; Cagan, J.; and Kotovsky, K. 1998. Agent-based Synthesis of Electro-mechanical Design Configurations. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 80333, V003T03A003. American Society of Mechanical Engineers.
- Chen, L.; Jing, Q.; Tsang, Y.; Wang, Q.; Sun, L.; and Luo, J. 2024a. DesignFusion: Integrating Generative Models for Conceptual Design Enrichment. *Journal of Mechanical Design*, 146(11).
- Chen, L.; Zuo, H.; Cai, Z.; Yin, Y.; Zhang, Y.; Sun, L.; Childs, P.; and Wang, B. 2024b. Towards Controllable Generative Design: A Conceptual Design Generation Approach Leveraging the FBS Ontology and Large Language Models. *Journal of Mechanical Design*, 1–40.
- Cheong, H.; Ebrahimi, M.; Butscher, A.; and Iorio, F. 2019. Configuration Design of Mechanical Assemblies Using an Estimation of Distribution Algorithm and Constraint Programming. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2339–2346. IEEE.
- Coulom, R. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *International conference on computers and games*, 72–83. Springer.
- Deb, K.; and Jain, S. 2003. Multi-speed Gearbox Design Using Multi-objective Evolutionary Algorithms. *J. Mech. Des.*, 125(3): 609–619.
- Dym, C. L. 1994. *Engineering Design: A Synthesis of Views*. Cambridge University Press.
- Grignon, P. M.; and Fadel, G. M. 2004. A GA Based Configuration Design Optimization Method. *J. Mech. Des.*, 126(1): 6–15.
- Jang, E.; Gu, S.; and Poole, B. 2016. Categorical Reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*.
- Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Židek, A.; Potapenko, A.; et al. 2021. Highly Accurate Protein Structure Prediction with AlphaFold. *nature*, 596(7873): 583–589.
- Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-carlo Planning. In *European conference on machine learning*, 282–293. Springer.
- Larrañaga, P.; and Lozano, J. A. 2012. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, volume 2. Springer Science & Business Media.
- Levin, M. S. 2009. Combinatorial optimization in system configuration design. *Automation and Remote Control*, 70: 519–561.
- Luo, R.; Wang, Y.; Xiao, W.; and Zhao, X. 2022. Alpha-Truss: Monte Carlo Tree Search for Optimal Truss Layout Design. *Buildings*, 12(5): 641.
- Mittal, S.; and Frayman, F. 1989. Towards a Generic Model of Configuration Tasks. In *IJCAI*, volume 89, 1395–1401. Citeseer.
- Piacentini, C.; Cheong, H.; Ebrahimi, M.; and Butscher, A. 2020. Multi-speed Gearbox Synthesis Using Global Search and Non-convex Optimization. In *International conference on integration of constraint programming, artificial intelligence, and operations research*, 381–398. Springer.
- Regenwetter, L.; Nobari, A. H.; and Ahmed, F. 2022. Deep Generative Models in Engineering Design: A Review. *Journal of Mechanical Design*, 144(7): 071704.
- Schmidt, L. C.; and Cagan, J. 1998. Optimal Configuration Design: An Integrated Approach Using Grammars. *J. Mech. Des.*, 120(1): 2–9.
- Shea, K.; Cagan, J.; and Fenves, S. J. 1997. A shape Annealing Approach to Optimal Truss Design with Dynamic Grouping of Members. *J. Mech. Des.*, 119(3): 388–394.
- Suh, N. P. 1990. *The Principles of Design*. Oxford University Press.
- Thoring, K.; Huettemann, S.; and Mueller, R. M. 2023. The Augmented Designer: A Research Agenda for Generative AI-enabled Design. *Proceedings of the Design Society*, 3: 3345–3354.

Varadi, M.; Anyango, S.; Deshpande, M.; Nair, S.; Natassia, C.; Yordanova, G.; Yuan, D.; Stroe, O.; Wood, G.; Laydon, A.; et al. 2022. AlphaFold Protein Structure Database: Massively Expanding the Structural Coverage of Protein-sequence Space with High-accuracy Models. *Nucleic acids research*, 50(D1): D439–D444.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. *Advances in neural information processing systems*, 30.

Wielinga, B.; and Schreiber, G. 1997. Configuration-design Problem Solving. *IEEE expert*, 12(2): 49–56.

Zhang, C.; Wang, W.; Pangaro, P.; Martelaro, N.; and Byrne, D. 2023. Generative Image AI Using Design Sketches as Input: Opportunities and Challenges. In *Proceedings of the 15th Conference on Creativity and Cognition*, 254–261.

Zhao, A.; Xu, J.; Konaković-Luković, M.; Hughes, J.; Spielberg, A.; Rus, D.; and Matusik, W. 2020. Robogrammar: Graph Grammar for Terrain-optimized Robot Design. *ACM Transactions on Graphics (TOG)*, 39(6): 1–16.