Computer-Aided Design 196 (2026) 104065

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

From dialogue to design: GenAI-Based automation of parametric modeling and sizing tasks in CAD workflows

Tobias Rosnitschek [a,*], Jan Walschewski [a], Peter Grohmann [a], Sascha Eckardt [b], Malte Stonis [b], Bettina Alber-Laukant [a], Stephan Tremmel [a]

[a] Engineering Design and CAD, University of Bayreuth, Universitätsstr. 30, 95447 Bayreuth, Germany
[b] IPH – Institut für Integrierte Produktion Hannover gGmbH, Hollerithallee 6, 30419 Hannover, Germany

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.18269442

Keywords:
Artificial intelligence applications
Computer-aided design
Design automation
Natural Language Processing
Engineering computation

ABSTRACT

This study explores the automation of engineering design tasks using Generative Artificial Intelligence, focusing on the dimensioning machine elements and generation of their respective CAD models, specifically bolted connections, in the open-source software FreeCAD. Three system architectures were developed and evaluated: All-in-One, Chatbot-Designbot, and Chatbot-Calculator-Code Generator. These frameworks integrate Large Language Models, such as GPT-2 and CodeGen, which were fine-tuned using Parameter-Efficient Fine-Tuning and Low-Rank Adaptation. While the All-in-One architecture consolidates all tasks into a single model, the Chatbot-Designbot and Chatbot-Calculator-Code Generator architectures decompose the process into specialized modules for dialogue interaction, parameter extraction, part dimensioning, and CAD code generation. The evaluation results show that the Chatbot-Calculator-Code Generator configuration achieves the lowest overall error rate for the four steps combined (2%) with a total training time of 63.4 min. This configuration outperforms the Chatbot-Designbot (3.96%, 117.7 min) and All-in-One (53%, 24.2 min) architectures. These findings demonstrate that compact, fine-tuned Large Language Models can enable accurate and efficient design automation, even with limited data. This work establishes a methodological foundation for scalable, Generative Artificial Intelligence -driven CAD systems and interactive engineering design workflows.

List of Abbreviations

A_S	Loaded cross-section area
M_{global}	Total error rate
M_n	Error rate
E_{-1}	*eval_loss* of the previous epoch
E_0	*eval_loss* of the current epoch
F_A	Longitudinal operating force
T_{global}	Total training time
T_n	Training time
R_e	Yield strength
α	Factor for calculating the permissible operating stress
ε	Relative *eval_loss* change
σ_{lim}	permissible operating stress
AIO	All-in-One
CCC	Chatbot-Calculator-Code Generator
CD	Chatbot-Designbot
c_attn	convolutional attention
c_fc	convolutional fully connected
c_proj	convolutional projection
fc_in	fully connected input
fc_out	fully connected output
FMC	FreeCAD-Macro-Code
GenAI	Generative Artificial Intelligence
AI	Artificial Intelligence
LLM	Large Language Model
lm_head	language modeling head
LoRA	Low-Rank Adaptation
out_proj	output projection
PEFT	Parameter-Efficient Fine-Tuning
PDP	Parameter-Diameter-Pair
qkv_proj	query-key-value projection
QLoRA	Quantized Low-Rank Adaptation
wte	word token embedding

* Corresponding author.
E-mail address: tobias.rosnitschek@uni-bayreuth.de (T. Rosnitschek).

https://doi.org/10.1016/j.cad.2026.104065
Received 30 September 2025; Received in revised form 16 March 2026; Accepted 20 March 2026
Available online 20 March 2026
0010-4485/© 2026 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

wpe word positional embedding

1. Introduction

The drive to implement Industry 4.0 technologies spans every economic sector and has reached a global scale [1]. Central to this fourth industrial revolution is the seamless communication and networking between humans and machines. Computer-Aided Design (CAD) plays a pivotal role in this, having become integral to modern production workflows [2]. Consequently, the demand for CAD expertise is high, and industries are seeking cost-effective methods to train large numbers of users [3] and enabling novice users to obtain desired solutions efficiently [4], respectively.

Artificial Intelligence (AI) promises to further disrupt this landscape. Just as CAD replaced hand-drafted technical drawings, AI may soon make extensive CAD training unnecessary. AI already accelerates processes and reduces costs across every day and industrial domains. For instance, AI-powered chatbots now routinely handle customer service inquiries [5,6]. Similar AI-driven automation is being actively explored in finance, healthcare, agriculture, and beyond [7].

In engineering and manufacturing, AI is destined to become an essential design tool [8,9]. Akay and Kim illustrated that in particular at early stages of the design process, designers can benefit by collaborating with machines [10,11], as functional requirements and design parameters, expressed in words and sentences can be represented in a vector space [12].Tadokoro et al. used generative AI for human-AI co-creation for solving engineering problems [13]. The ultimate vision is an AI system that not only follows engineers' instructions but autonomously solves design problems and creates components, or variants thereof. Research toward this goal is vigorous, and early successes in CAD automation, such as AI-assisted quality control and model generation, have been reported [14–16]. For instance, Lee et al. present an automated design inspection system for automobile trimming dies by integrating CAD software with data-efficient AI modules capable of detecting and evaluating critical geometry parameters. Through bidirectional "zigzag" interaction between CAD and AI, the system replaces manual inspection tasks, achieving up to 80 % time savings while maintaining high accuracy even with limited training data [17]. Li et al. propose a generative AI framework that integrates stable diffusion models with CAD automation to generate wheel rim designs under severely limited data conditions, achieving feasible designs with high design accuracy using only a few hundred training samples. The approach also incorporates an automated validation mechanism to filter out infeasible designs, significantly reducing data requirements and overall design time compared to traditional workflows [18]. Furthermore, Wdowik and Bełzo explored the use of generative AI, specifically GPT models, to automate Python-scripted part and assembly generation in FreeCAD [19]. Their work demonstrates the feasibility of using LLMs for text-based CAD scripting and highlights both the potential and limitations of such approaches in mechanical design. However, their implementation remains limited to isolated geometry generation tasks and does not address the structured acquisition of design parameters or functional constraints through dialogue. Khan et al. introduce an AI framework that can generate parametric CAD models directly from natural language descriptions by learning design sequences such as sketches and extrusions [20]. While CAD drawings and 3D models offer a compact and information-dense representation of geometry, they are not inherently intuitive for specifying or modifying designs, especially for non-expert users. Textual descriptions, in contrast, provide a natural and low-barrier interface for capturing intent, constraints, and parametric relationships in a way that can be automatically translated into CAD operations. This development of a large-scale annotated dataset and a transformer-based autoregressive model was therefore necessary to bridge this gap, enabling designer-friendly, text-driven CAD generation for users of all skill levels. Additionally, Badagabettu et al. present a framework that uses large language models to generate and refine CAD models iteratively via self-refinement loops and human-in-the-loop feedback [21]. This approach achieves significant improvements in design success rates without the need for supervised training.

One promising approach is text-based generation of CAD models, that leverages the programming capabilities of Large Language Models (LLMs) [22–24], which are potentially capable of performing complex tasks related to CAD including data generation, coding, parametric generation, image synthesis, model evaluation and text generation [25]. FreeCAD, with its Python scripting interface [26], provides a flexible platform for such AI-driven macro generation and execution [27,28]. However, LLMs must be fine-tuned to understand domain-specific norms to meet specialized engineering standards and procedures. Techniques such as Low-Rank Adaptation (LoRA) [29] and its quantized variant QLoRA [30] have emerged to reduce the number of trainable parameters and thereby lowering hardware requirements and training costs without sacrificing model performance [31,32]. Combining the advanced capabilities of LLMs, resource-efficient fine-tuning strategies, and FreeCAD's scripting functionality, establishes a promising foundation for AI-augmented CAD workflows.

Unlike prior approaches, which focus on either downstream CAD code generation or isolated parameter extraction, our work uniquely investigates full-pipeline architectures, from interactive parameter collection through to automated FreeCAD modeling. For the first time, we systematically compare integrated versus modular LLM configurations. The defined task in this work is to gather all necessary parameters to correctly design a bolt connection based on a dialogue with the user. To accomplish this, we present and compare three architectures for AI-augmented bolt-connection design and subsequent CAD modeling in FreeCAD: (i) the All-in-One (AIO) architecture, in which a single LLM jointly handles user dialogue, parameter extraction, dimensioning, and macro generation; (ii) the Chatbot-Designbot (CD) architecture, which decouples natural language understanding from model synthesis; and (iii) the Chatbot-Calculator-Code Generator (CCC), which further isolates numerical computation into a dedicated module. Using fine-tuned GPT-2 [33,34] and CodeGen [23] models, we evaluate each pipeline based on its accuracy in extracting parameter, computational sizing fidelity, dialogue coherence, script correctness, and training efficiency. Our results elucidate the trade-offs between monolithic and modular approaches. They demonstrate that task-specific decomposition substantially improves design accuracy without requiring excessive computing power.

2. Materials and methods

Building on the motivation and objectives outlined in the previous section, this section presents the technical methodology underpinning our investigation into AI-augmented CAD workflows. First, we describe three alternative system architectures, each based on LLMs, for automating the design and modeling of bolt connections (bolt and nut) in FreeCAD. These architectures represent different approaches to decomposing the generative design pipeline into dialogue management, parameter extraction, dimensioning, and script generation. Next, we detail the construction of tailored training datasets and introduce the Low-Rank Adaptation (LoRA) fine-tuning approach employed across the models. Finally, we report on the hyperparameter sensitivity analyses conducted. Lastly, we define the quantitative metrics used to evaluate model accuracy, interaction quality, script validity, and computational efficiency. For benchmarking purposes, analytically derived reference values for the required bolt diameter are used based on established engineering standards.

2.1. System architectures

We explored three distinct architectures for integrating LLMs into the FreeCAD modeling pipeline. Each architecture varies in the degree of task decomposition for the information flow depicted in Fig. 1(a).

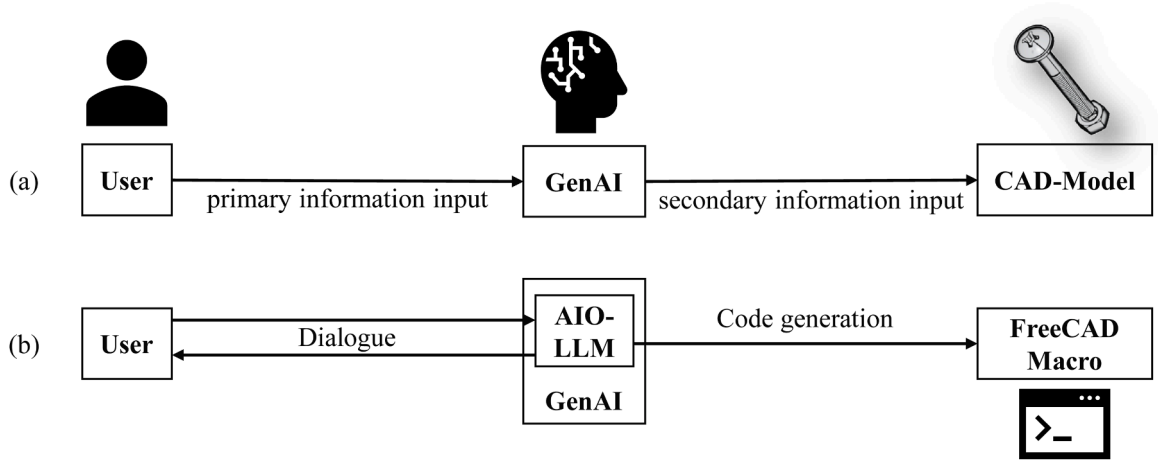

Fig. 1. (a): General information flow from user to final CAD Model. (b): Schematic representation of the AIO architecture.

In the All-in-One (AIO) design, a single fine-tuned GPT-2 model assumes end-to-end responsibility: engaging in a multi-turn dialogue with the user, extracting necessary design parameters, computing the required thread diameter via common design formulas, and synthesizing the complete Python macro for FreeCAD. This monolithic approach tests the model's capacity to balance heterogeneous objectives within one network, see Fig. 1(b).

The AIO dataset format, illustrated in Fig. 2, comprises two distinct parts: (i) a user-bot dialogue simulating parameter elicitation, and the associated Python FreeCAD macro code (FMC) output, for brevity, the FMC in Fig. 2 was truncated but the full code is available on Zenodo in [35]. Generally, a training sample is divided into two distinct parts. The first part (highlighted in green) contains the complete user-Chatbot dialogue. Following the final user input, the second part (highlighted in blue) begins. This section lists the extracted bolt parameters in a predefined format. A sentinel token (###) marks the boundary between natural language and code. the transition point, signaling that the parameters are now presented in a structured way suitable for downstream processing via script.

The Chatbot-Designbot (CD) pipeline (Fig. 3) distributes these responsibilities among two specialized LLMs. First, a fine-tuned GPT-2 model referred to as "Chatbot", parses the user input and outputs a structured parameter list. Then, a separate CodeGen-350 M model referred to as "Designbot" consumes those parameters to perform dimensional calculations and generate the corresponding macro code. By decoupling language understanding from code synthesis, the CD pipeline aims to reduce conflicting learning signals.

The first task is to maintain a structured dialogue with the user and extract the relevant bolt parameters. This responsibility is assigned to the Chatbot component. The second task is handled by the Designbot, which uses the extracted parameters to compute the required dimensions and generate the corresponding FreeCAD macro (FMC). Both components are trained using user-Chatbot dialogues generated with Grok 3 [36] and parameter-diameter pairs (PDPs). An example training sample for the Chatbot model is shown in Fig. 4 below.

A Designbot training sample, see Fig. 5, is free of any dialogue and thus consists of the parameter list extracted by the Chatbot (blue) and the corresponding FreeCAD macro (FMC) (red) to be generated from it.

Since the FMC remains identical apart from the inserted bolt parameters, its generation is expected to be relatively easy for the model to learn. Unlike the All-in-One (AIO) architecture, this design leaves most of the model's capacity available for mastering the engineering computation rather than being consumed by dialogue processing.

Finally, as shown in Fig. 6, the Chatbot-Calculator-Code Generator (CCC) architecture further isolates numerical reasoning from code generation.

```
[User]: Model a DIN967 bolt with a static load of 500N.
[Bot]: Please provide the grade and length of the bolt.
[User]: The grade is 6.8 with a length of 60mm.
[Bot]: Perfect! Modelling now.
###
import FreeCAD
import Part
import screw_maker2_2


doc = FreeCAD.ActiveDocument
if doc is None:
    doc = FreeCAD.newDocument("Screw")

... remaining FMC
```

Fig. 2. Structure of data point for the AIO models. The dialogue between user and bot is displayed in green the resulting FreeCAD-script is written in blue, while being separated by three hashtags from the input.

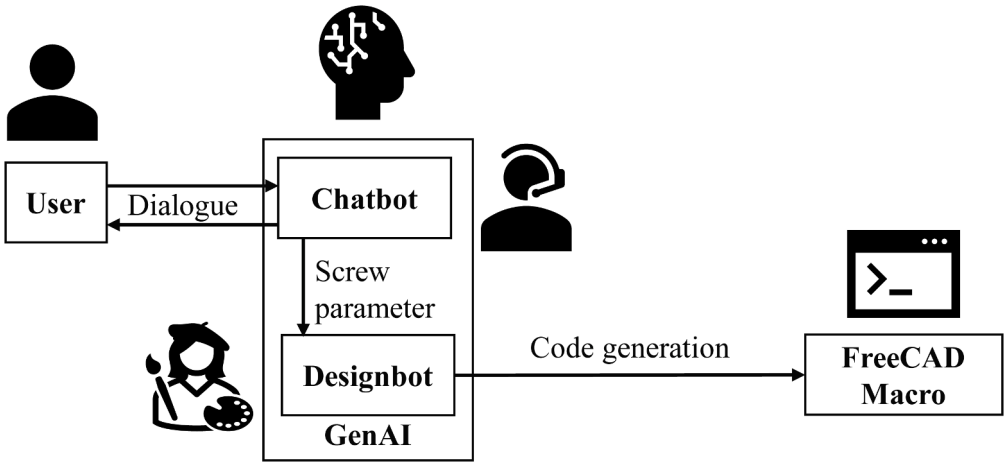

Fig. 3. Schematic representation of the CD architecture.

```
[User]: Model a DIN967 bolt with a static load of 500N.
[Bot]: Please provide the grade and length of the bolt.
[User]: The grade is 6.8 with a length of 60mm.
[Bot]: Perfect! Modelling now.
###static,500N,6.8,DIN967,60mm
```

Fig. 4. Data point structure for the CD architecture's Chatbot. The output (blue) differs, due to the specialisation of the models used for specific tasks rather than solving all tasks at once.

```
###static,500N,6.8,DIN967,60mm
import FreeCAD
import Part
import screw_maker2_2


doc = FreeCAD.ActiveDocument
if doc is None:
    doc = FreeCAD.newDocument("Screw")

screw_obj = screw_maker2_2.Screw()

screw_type = "DIN967"
nut_type = "ISO4032"
diameter = "M4"
length = "60"
thread_type = "real"

screw =
screw_obj.createScrew(screw_type,diameter,length,thread_type )
nut = screw_obj.createScrew(nut_type,diameter,length,thread_type )
nut.Placement.Base = FreeCAD.Vector(0, 0, -10-int(length))
doc.recompute()
```

Fig. 5. Data point structure for the CD architecture's Designbot.

The Chatbot component is the same as the CD's and is followed by a dedicated "Calculator" model (either GPT-2 or CodeGen), which is trained exclusively on the parameter-to-diameter mapping. Finally, there is a "Code Generator" that transforms the complete parameter, including the computed diameter, into the FreeCAD macro. This high degree of modularization enables each model to focus on a narrowly defined subtask, potentially improving overall accuracy and training efficiency.

An example training sample for the Calculator model is shown in Fig. 7.

Each training sample consists of the parameter list generated by the Chatbot. This model's sole task is to compute the appropriate thread

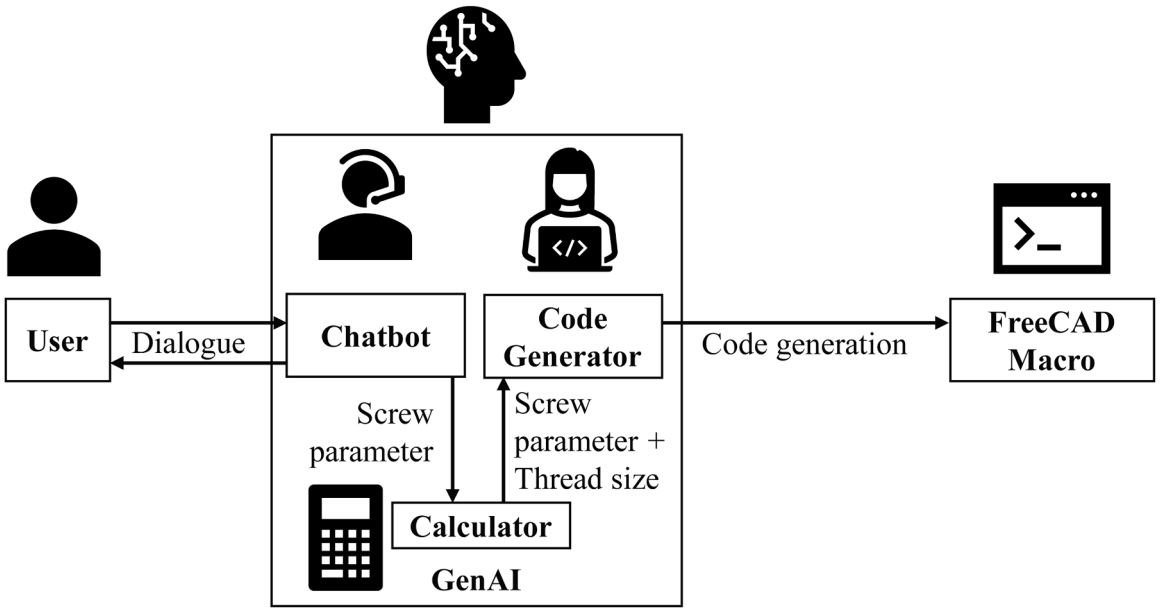

Fig. 6. Schematic representation of the CCC architecture.

```
###static,500N,6.8,DIN967,60mm ,M4
```

Fig. 7. Data point structure for the CCC architecture's Calculator.

diameter, which is then appended to the end of the parameter list. Each training sample consists of the parameter list extracted by the Chatbot, followed by the computed thread diameter. The Calculator's sole task is to derive this diameter based on the given parameters and append it to the list. While simplified empirical formulas for bolt sizing are well established, their practical application requires careful interpretation of context-specific parameters, design constraints, and standards. The integration of such reasoning into an automated, dialogue-based CAD workflow represents a significantly higher level of abstraction than applying fixed equations. The key challenge here is to teach the model reliable mathematical reasoning in context, allowing it to flexibly infer appropriate diameters across a variety of use cases. Although LLMs often exhibit weaknesses in numerical computation, recent research has shown that such capabilities can indeed be improved through targeted training [37,38].

```
###static,500N,6.8,DIN967,60mm ,M4
import FreeCAD
import Part
import screw_maker2_2


doc = FreeCAD.ActiveDocument
if doc is None:
    doc = FreeCAD.newDocument("Screw")

screw_obj = screw_maker2_2.Screw()

screw_type = "DIN967"
nut_type = "ISO4032"
diameter = "M4"
length = "60"
thread_type = "real"

screw =
screw_obj.createScrew(screw_type,diameter,length,thread_type )
nut = screw_obj.createScrew(nut_type,diameter,length,thread_type )
nut.Placement.Base = FreeCAD.Vector(0, 0, -10-int(length))
doc.recompute()
```

Fig. 8. Data point structure for the CCC architecture's Code Generator. Extracted parameters (blue) by the Chatbot; Calculated Thread size (red) by the Calculator; FMC (green) that must be generated by the Code Generator.

Since the inputs are consistently formatted and limited to specific parameters, the model can focus exclusively on calculating the correct thread diameter. This targeted formulation is expected to improve the model's performance on the sizing task. The same input-output structure, parameter list followed by the computed diameter, is also used for the Code Generator's training data (see Fig. 8).

The sub-division into three architectures leads to a total of five separate LLM components used in this work and comparatively displayed in Tab 1.

2.2. Dataset generation

A bespoke dataset was synthesized to support each architecture, comprising approximately 2000 examples per pipeline. Since no public corpus exists for the needed text-to-calculation-to-CAD translation, we constructed our own. User-Chatbot exchanges were generated with Grok 3. Prompts directed Grok 3 to produce varied multiturn dialogues that elicited five key parameters: load type, force magnitude, material grade, standard designation, and length. Diverse linguistic formulations were used to enhance model robustness, an exemplary user-bot conversation is given in Fig. 9, where the system got all needed parameters two user inputs.

In the synthesized dialogues, the user provides or requests bolt parameters, while the language model responds by extracting known values, requesting missing ones, and confirming the start of the modeling process. The dialogues were designed with varied phrasing and structure to expose the model to a broad range of user behaviors and reduce the risk of overfitting – memorizing patterns specific to the training set rather than generalizable interactions.

The dialogues were guided by a carefully engineered prompt that was iteratively refined until consistent results were achieved. Each prompt request produces 40 dialogues based on 50 template variations. Rather than hardcoding numeric values into the dialogues, Grok 3 inserted placeholders, that were populated later using a Python script. This approach significantly reduces the preprocessing effort because extracting numerical parameters from free-text dialogue is more labor-intensive than generating and inserting predefined parameter–diameter pairs (PDPs), as depicted in Fig. 10.

The PDPs were computed analytically using common design formulas from [39] and then inserted programmatically into the generated templates. A Python script applied these equations to calculate the minimum required loaded stress cross-section area of the bolt thread, A_S, based on the design load:

$$\sigma_{lim} = \alpha \cdot R_e \tag{1}$$

$$A_S = \frac{F_A}{\sigma_{lim}} \tag{2}$$

σ_{lim} being the permissible operating stress. α describes the safety factor used to calculate σ_{lim}. R_e is the yield strength of the bolt, and F_A the axial working load.

The chosen formula reflects a widely used approximation from classical bolt dimensioning standards and is applicable under static loading conditions with predefined material properties and safety factors. While this approach is sufficient for a wide range of standard applications, it does not cover cases such as fatigue loading, high-temperature environments, or non-standard geometries. It is therefore important to emphasize that this implementation serves as a proof of concept for the automated use of engineering equations within a generative AI workflow and not as a general-purpose design substitute. Future iterations could integrate rule-based filtering or incorporate expert-defined boundary conditions to ensure that the applicability domain of each equation is respected. The current system is built around a fixed predefined parameter schema, as such, it does not reason about which parameters are necessary for a given task but assumes that a fully populated set represents all needed parameters. This distinction is critical when assessing the autonomy of the system. The models do not infer which inputs are needed, they are trained to recognize and extract parameters from dialogues that conform to a known pattern. While this controlled setup facilitates technical benchmarking, it limits applicability in open-ended design scenarios. Incorporating context-aware boundary checks or dynamic reasoning remain key directions for future work.

Table 1
Overview of the components needed and their use in the three system architectures.

	AIO	CD	CCC
AIO-LLM	X		
Chatbot		X	X
Designbot		X	
Calculator			X
Code Generator			X

For designing the dialogues within the scope of this work, the parameters were randomized sampled uniformly from the prescribed ranges, as listed in Table 2.

To generate training data for each architecture, we defined a canonical FreeCAD macro template containing the necessary imports (FreeCAD, Part, screw_maker2_2), document initialization, and *createScrew* calls. Sampled parameters and computed diameters were inserted into template placeholders to produce executable Python scripts. Depending on the architecture, these templates were formatted differently:

- AIO: The user dialogue and the complete macro code were concatenated into a single training sample, separated by a sentinel token (###).
- CD: The user dialogue was followed by a comma-delimited parameter list after the sentinel token (###); the Designbot then generated the macro code from these parameters.
- CCC: The Calculator received only the parameter list, while the Code Generator examples combined this list with the macro code to produce the final script.

This synthesis ensures end-to-end coverage from natural-language specification to CAD macro while enabling focused specialization in the CD and CCC pipelines.

2.3. Model fine tuning

All fine-tuning experiments were conducted on an NVIDIA A40 GPU with 48 GB of VRAM. The experiments used the PyTorch framework [40] in combination with HuggingFace Transformers [41] and the PEFT library [42] for parameter-efficient adaptation. Different model families were selected for distinct subcomponents to support the modular architecture variants described in Section 2.1:

- **Dialogue and calculator tasks** were handled using GPT-2 models (small: 124 million parameters; medium: 355 million parameters).
- **Macro code generation** was performed by CodeGen-350M-mono, a transformer model optimized for Python syntax.
- **Dialogue quality assessment** was automated using a fine-tuned DistilBERT [43] classifier trained to detect conversational inconsistencies and failure patterns.

The selection of GPT-2 models was informed by prior successes in comparable tasks, particularly those involving goal-oriented dialogue and the tracking of structured parameters [34]. GPT-2 also exhibits strong zero-shot capabilities; that is, it can follow task-specific instructions without prior exposure to the target task [44]. Although the models in this study were fine-tuned, the relatively small training sets

```
[User]: Model a DIN967 bolt with a static load of 500N.
[Bot]: Please provide the grade and length of the bolt.
[User]: The grade is 6.8 with a length of 60mm.
[Bot]: Perfect! Modelling now.
```

Fig. 9. Exemplary multiturn dialogue used for model training.

```
static,500,6.8,DIN967,60,M4
```

Fig. 10. Structure of the PDPs with the user-defined parameters (green) and the calculated diameter (blue).

Table 2
Extracted possible random parameter values used for the PDPs.

Parameter	Possible Values
Load type	Static, dynamic
Longitudinal force	10 N to 1 MN
Grade	3.6, 4.6, 4.8, 5.6, 5.8, 6.8
Standard	ISO 4017, ISO 4762, ISO 10,642, EN 1662, DIN 967
Length	10 mm to 300 mm

(approximately 2000 samples) mean that these zero-shot properties remain relevant and likely beneficial.

For code generation, the CodeGen model family was used, specifically the smallest variant, CodeGen-350M-mono, to remain within hardware constraints. The "mono" versions are pre-trained exclusively on Python source code, making them well suited to the FreeCAD macro generation task [23]. Like GPT-2, CodeGen models demonstrate robust zero- and few-shot performance characteristics. An additional model, DistilBERT-base-uncased (66 M parameters) [43], is used for dialogue evaluation. However, since this model is not involved in the generative AI pipeline itself, it is not discussed in detail.

All selected models, GPT-2, CodeGen, and DistilBERT, are publicly available via Hugging Face and are compatible with the HuggingFace Transformers library. Furthermore, they support parameter-efficient fine-tuning using PEFT techniques such as LoRA, which facilitates rapid adaptation with minimal resource demands. To enable efficient specialization with reduced computational overhead, all models were fine-tuned using Low-Rank Adaptation (LoRA). Adapters were injected into selected transformer layers as follows:

- For **GPT-2**, LoRA modules were applied to the multi-head attention (c_attn), projection (c_proj), and feedforward (c_fc) components.
- For **CodeGen**, the adaptation targeted qkv_proj, out_proj, fc_in, and fc_out.
- In both cases, the adapter rank was set to $r = 8$, with a scaling factor $\alpha = 8$.

The general fine-tuning setup was consistent across all model roles unless otherwise noted:

- **Epochs:** Training was run for up to 20 epochs, with evaluation after each epoch.
- **Learning rate:** A default of 2×10^{-4} was used, with additional experiments at 1×10^{-4} and 5×10^{-4} in sensitivity analyses.
- **Batch size:** Dynamically selected to maximize GPU utilization per model type.
- **Validation split:** A random 10 % of training samples were reserved for validation.

To explore how key training variables affect model performance and efficiency, a structured sensitivity analysis was conducted for the All-in-One (AIO) architecture. One hyperparameter varied at a time:

- **Dataset size:** 1000, 2000, and 4000 examples
- **Adapter rank:** $r = 4$, 8, 16
- **Learning rate:** 1×10^{-4}, 2×10^{-4}, 5×10^{-4}
- **Adapter placement:** single layer vs. all linear layers in the model

Although real-time early stopping was not implemented, we conducted a retrospective estimation to assess potential training time savings. The relative change in validation loss after each epoch was calculated as follows:

$$\varepsilon = \frac{E_{n-1} - E_n}{E_{n-1}} \tag{3}$$

Where E_n denotes the validation loss at epoch n. Thresholds of 1 %, 2 %, and 3 % were analyzed to identify candidate stopping points. This fine-tuning strategy provided a scalable and hardware-efficient way to adapt general-purpose LLMs to the domain-specific tasks of natural language interpretation, numerical design calculation, and FreeCAD macro generation. The general fine-tuning code, along with all used parameters, is provided in Appendices A and B, respectively. The modular approach also enabled targeted optimization of individual model components to improve overall system performance.

2.4. Evaluation metrics

We evaluated all models using a consistent set of six quantitative metrics to assess the effectiveness and reliability of each architectural variant. These metrics capture performance across the three core tasks: user dialogue processing, engineering dimensioning, and FreeCAD macro generation, as well as training efficiency.

1. Parameter extraction error (%):
   This metric quantifies the rate at which the model incorrectly extracts user-specified parameters (e.g., load type, force magnitude, grade, or length) from the input dialogue. Each mismatch between the ground truth and the extracted value is counted as an error.
2. Sizing error (%):
   Sizing accuracy is evaluated by comparing the model's predicted thread diameter to the correct value computed. Any deviation from the valid solution range is counted as a sizing error. While the analytical sizing formula is fixed and deterministic, the large language model is not explicitly programmed to apply it. Instead, it must learn to approximate the formulaic logic from training examples. This introduces the potential for errors in numerical reasoning, especially in edge cases or when input parameters fall outside the typical training distribution.
3. Code generation error (%):
   This measures the proportion of generated Python macros that fail due to syntax errors, missing dependencies, or incorrect usage of FreeCAD's scripting API. Validity was assessed through automated execution in the FreeCAD environment.
4. Dialogue failure rate (%):
   A fine-tuned DistilBERT classifier was used to detect dialogue breakdowns, including incomplete parameter queries, redundant prompts, and premature termination. Evaluation was conducted on a held-out set of 100 user-Chatbot interactions.
5. Validation loss:The average cross-entropy loss on the validation split (10 %) was recorded after each epoch to monitor model

convergence. This serves as an internal measure of the model's ability to generalize to unseen prompts.

6. Training time (minutes):
   The wall-clock time required to train each model to completion was tracked to quantify resource efficiency. For CD and CCC, the total training time is the sum of the times across all submodels.

For composite pipelines (CD and CCC), we also computed the overall error rate M_{global} assuming independent error propagation across com-

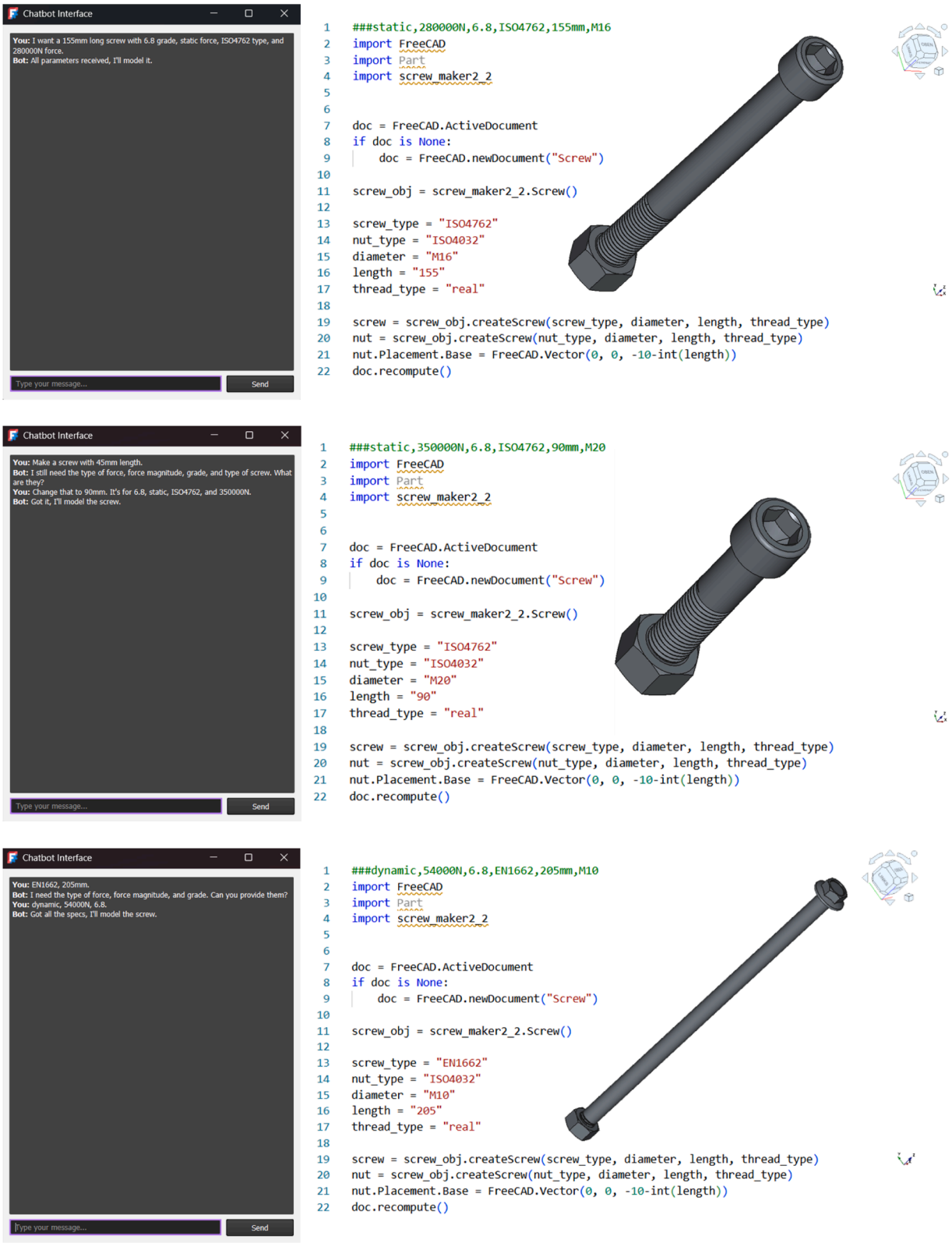

Fig. 11. Examples of Resulting FreeCAD Macros and CAD-generated bolt connections using different Chatbot interactions within the CCC architecture.

ponents:

$$M_{global} = 1 - \prod_n (1 - M_n) \tag{4}$$

This formulation, where M_n denotes the error rate of component n, provides a more realistic estimate of system-level performance when downstream components depend on upstream outputs.

The same equation was also used to calculate the overall error rates for each system architecture, incorporating all individual error sources, including parameter extraction, code generation, sizing accuracy, and dialogue consistency.

Total training time (T_{global}) was computed as the sum of the training durations of all model components within a given pipeline.

$$T_{global} = \sum_n T_n \tag{5}$$

Together, these metrics allow for a comprehensive comparison between monolithic and modular architectures, balancing accuracy, robustness, and computational cost.

3. Results and discussion

This section presents a comparative analysis of the three AI-driven CAD-Chatbot system architectures described above: AIO, CD, and CCC. These architectures were evaluated based on accuracy, model robustness, and computational efficiency. Each architecture reflects a distinct strategy for integrating LLMs into parametric design automation, ranging from monolithic to modular configurations. The emphasis is placed on key performance indicators relevant to CAD automation: parameter extraction fidelity, compliance with engineering design rules, and the generation of syntactically and semantically correct feature modeling code. The results provide insight into the feasibility of using fine-tuned foundation models to support end-to-end CAD workflows and highlight both the capabilities and limitations of current generative AI methods, when applied to engineering design tasks. As demonstration of the developed GenAI-assisted design process, Fig. 11displays three examples of bolt connections generated by different Chatbot Interfaces.

3.1. Comparison of the system architectures

To establish a baseline, we implemented a reference bolt connection using the analytical sizing procedure described in [39] and scripted a FreeCAD macro representing the expected deterministic output. This serves as the engineering ground truth against which the AI-generated models are compared. To contextualize the model outputs, we clarify that the objective was to generate an FMC that designs the bolt connection. As such, the visual appearance of the bolt connections remains identical across baseline and generated samples. The only variations emerge from parameter-specific differences, particularly in thread diameter sizing, which is quantified below. This uniform macro structure enables a fair comparison across architectures and supports reproducibility. Further, the Calculator component does not perform open-ended engineering reasoning but learns a fixed analytical mapping from a fully specified parameter set to a corresponding thread diameter. This design choice was intentional, as the objective of this study is not to replace engineering judgment, but to evaluate how generative AI systems behave when embedded into deterministic CAD workflows under controlled conditions.

Supplementary C provides a detailed overview of all models used in the three system architectures. This section compares their performance in terms of error rates and total training time. In the All-in-One (AIO) architecture, all models achieved identical results: a 0 % error rate for parameter extraction and varying from 0 % to 39 % in dialogue handling, but a high 53 % error rate in design calculation, which dominates the total error. Among these, the GPT2-small model trained with a learning rate of 5×10^{-4} had the shortest training time, at 24.2 min, and was therefore selected for comparison, despite similar training durations among other models.

For the Chatbot–Designbot (CD) pipeline, the GPT2-small Chatbot achieved an error rate of 2 % for parameter extraction and 0 % for dialogue consistency. This matched the performance of the larger GPT2-medium model, but the GPT2-small Chatbot had a shorter training time of 22.2 min. The Designbot used the codegen-350M-mono model, which produced a 2 % error in bolt dimensioning, had no errors in parameter extraction, and took 95.5 min to train.

In the Chatbot-Calculator-Code Generator (CCC) pipeline, the same Chatbot as in the CD pipeline was used. The Calculator employed codegen-350M-mono, achieving 0 % error in bolt dimensioning with a training time of 17.6 min. The Code Generator was based on GPT2-small, with a 0 % error rate in parameter handling and a training time of 23.6 min.

The overall error rate (M_{global}) and training time (T_{global}) are calculated as follows:

$$M_{global} = 1 - (1 - M_{Chatbot})\left(1 - M_{Designbot}\right) = 1 - (1 - 0.02)(1 - 0.02) = 0.0396$$

$$T_{global} = T_{Chatbot} + T_{Designbot} = 22.2\ min + 95.5\ min = 117.7\ min$$

Table 3 summarizes the comparative results for all system architectures.

The reproducibility of outputs depends on the architecture, thus variation can arise from learned model behaviour, particularly in parameter extraction or sizing. The AIO architecture, where dialogue, parameter extraction and sizing are performed jointly with one LLM, showed 0 % in dialogue, but 53 % sizing error, indicating high sensitivity to training imbalance, task complexity and input prompts. CD and CCC decouple tasks and show significantly improved stability and robustness: both yielded only 2 % parameter extraction error, while sizing error was reduced to 2 % (CD) and 0 % (CCC).

Consequently, for a fixed prompt within the dataset range, CCC achieves near-zero variance in final output, while AIO exhibits high potential for divergent results due to entangled task handling. These findings emphasize the importance of modular task separation to achieve robust and repeatable AI-CAD integration.

3.2. Fine-tuning of the system parameters

The fine-tuning of all system architectures, AIO, CD, and CCC, was evaluated in terms of error rate, training time, and output quality. The analysis focused on model scalability, parameter optimization, and the benefits of task decomposition across multiple language models.

Even small GPT-2 models (e.g., GPT2-small) were shown to effectively learn to extract and assign engineering parameters from small datasets (approximately 2000 entries) using the AIO architecture. These models also successfully handled multitasking, such as managing dialogue and generating engineering code (FMC) simultaneously. However, all GPT-2 variants failed to correctly learn the internal logic for bolt dimensioning. This failure is attributed to dataset imbalance, visualized in Fig. 12. M30 threads were overrepresented, causing the models to

Table 3
Performance comparison across system architectures.

Metric	AIO	CD	CCC
Dialogue handling (M %)	0	0	0
Parameter extraction (M %)	0	2	2
Design sizing (M %)	53	2	0
Total error (M %)	53	3,96	2
eval_loss (final epoch)	0.040	0.038	0.035
Training time (min)	24.2	117.7	63.4
Inference time per request (s)	1.21	1.34	1.48
Peak GPU memory usage (GB)	5.4	9.2	7.6
Number of LLMs used	1	2	3

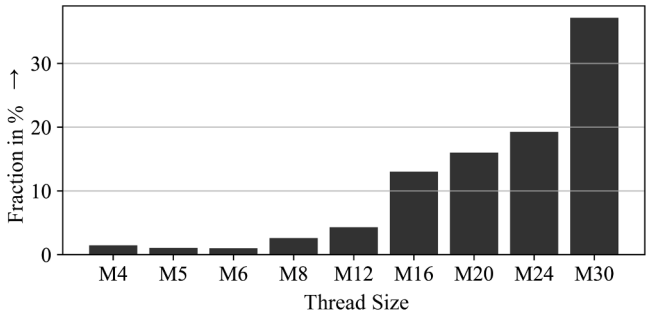

Fig. 12. Resulting Thread size distribution in the training data set.

default to this size regardless of the input. Whether a more balanced dataset would yield improved results remains an open question.

Models that were trained with a higher rank ($r = 16$), an increased number of trainable target modules, and a learning rate of 5×10^{-4} consistently outperformed the reference model in terms of dialogue performance and eval_loss. These improvements came with no substantial increase in training time, as the models were already operating near full hardware utilization. As shown in Fig. 13, dataset size had the most pronounced impact on training duration; GPT2-medium models took 2.87 times longer to train than their GPT2-small counterparts.

For time-sensitive applications, a GPT2-small model trained on approximately 1000 entries using aggressive hyperparameter settings may suffice, though overfitting is a risk.

Early stopping thresholds of 1 %, 2 %, and 3 % for relative eval_loss reduction led to significantly shortened training durations, particularly for GPT2-medium (see Fig. 14). At the 3 % threshold, for example, training time was reduced to one-third of the original. However, accelerated training is only useful if the model quality is preserved.

As shown in Fig. 15, the evaluation results revealed that models trained with early stopping, especially those employing higher learning rates and more target modules, achieved comparable or even better eval_loss than the fully trained reference models.

Nevertheless, general recommendations cannot be made without considering the complexity of the specific task, quality demands, and hardware constraints.

The CD and CCC pipelines were introduced to overcome the limitations observed in AIO. These architectures divide complex tasks, such as dialogue, parameter extraction, calculation, and code generation, into specialized models.

Despite their simplicity, both codegen-350M-mono and GPT2-small, handled code generation tasks effectively GPT, with the latter not even

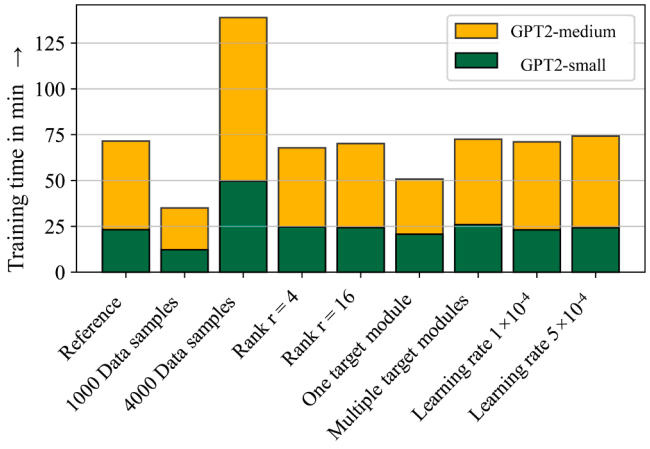

Fig. 13. Training times for GPTGPT2-small vs. GPT2-medium for different training settings. For each variation one parameter was varied at a time relatively to the reference.

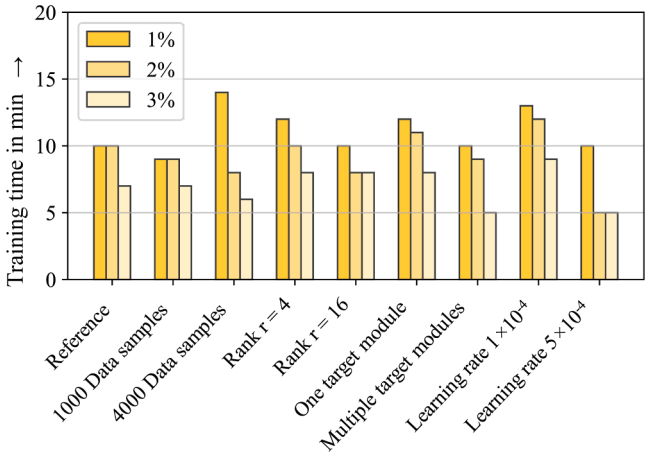

Fig. 14. GPT2-medium training times for the different early stopping thresholds 1 %, 2 %, and 3 %.

being pre-trained on code.

Specialization significantly improved design performance. The CD architecture's Designbot achieved a 2 % error rate, and the CCC Calculator using codegen-350M-mono reached a 0 % error rate. These results confirm that focused models provide superior results for highly deterministic tasks, such as bolt sizing.GPT-2 models (GPT2-small and GPT2-medium) used for the same task underperformed, with error rates of 20.1 % and 2.3 %, respectively.

The Chatbot component in both architectures achieved a 2 % parameter extraction error rate. Based on insights from the AIO setup, this error rate could likely be reduced to 0 % with optimized hyperparameters (e.g., rank = 16).

In summary, the AIO approach is not viable with its 53 % total error rate, despite the short training time and simplified pipeline. If future improvements enable it to reliably perform bolt dimensioning, its low complexity and high efficiency would make it attractive.

In contrast, the CD and CCC architectures showed promising results, with total error rates of 3.96 % and 2.0 %, respectively. Both could likely be improved further by enhancing the Chatbot's training parameters. Hypothetically, the CD architecture could achieve a total error rate of 2 %, and CCC architecture could reach 0 %.

Although CCC outperforms CD in both accuracy and training time (approximately 1 hour vs. 2 h for CD), CD remains a valid alternative. It uses fewer models and requires less programming effort. Minor improvements in training configuration or dataset balance could allow CD to match CCC's performance.

4. Conclusion

This work successfully demonstrates the integration of generative AI (GenAI) into a CAD modeling pipeline for the automated design and parametric modeling of a bolt, in accordance with common mechanical engineering guidelines. Leveraging LLMs, a functional system was developed that extracts user-defined bolt parameters through natural language dialogue and translates them into fully parametric CAD models within FreeCAD.

Among the evaluated system architectures, the CCC architecture – comprising three task-specific LLMs for dialogue handling, design computation, and code generation – emerged as the most effective. It achieved error-free design performance with a short training time by decomposing the overall task into narrowly defined subtasks.

The results show that small-scale models, such as GPT2-small and codegen-350M-mono, can perform complex engineering-related tasks such as parameter extraction and bolt dimensioning, when the task scope is clearly defined. Notably, even highly imbalanced datasets with as few as 2000 entries can suffice when combined with a modular system

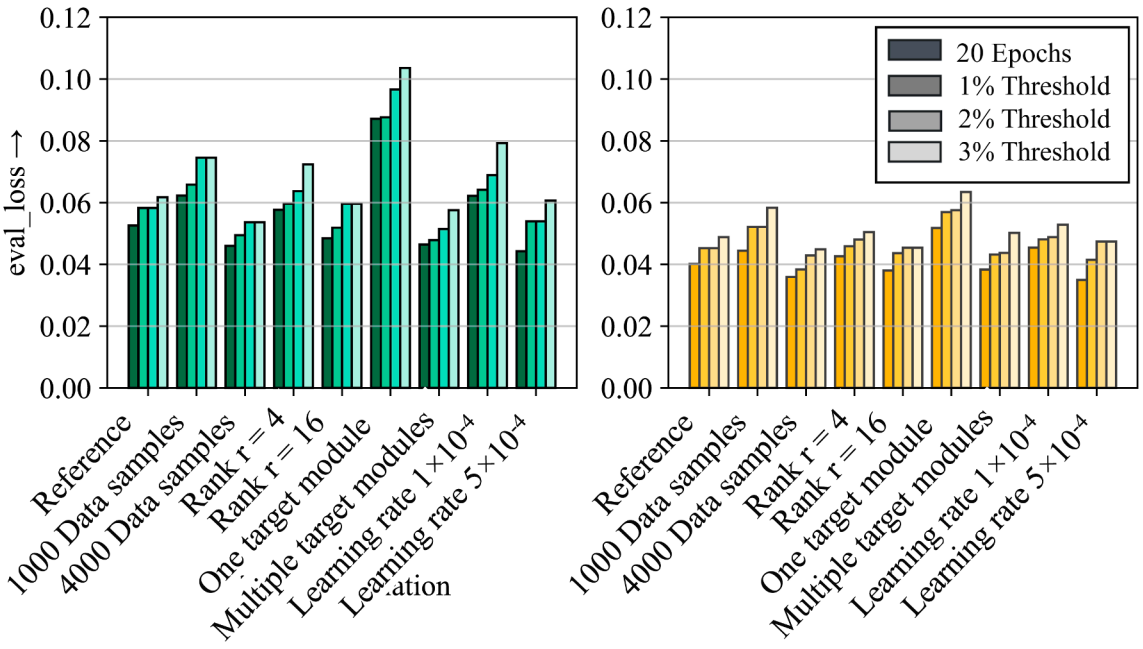

Fig. 15. Comparison of the eval_loss with and without early stopping for GPT2-small (green) and GPT2-medium (yellow) models.

architecture.

While the present study focuses on technical evaluation of AI-CAD system performance, the practical impact for users is a key consideration. The current results demonstrate that the modular architecture, particularly the CCC pipeline, supports robust and reproducible bolt connection generation with minimal configuration effort. Although formal user studies were not conducted, the workflow is expected to reduce design time, lower the risk of manual errors in repetitive CAD tasks, and offer a more accessible entry point for non-expert users.

Furthermore, the study provides insight into fine-tuning strategies. Increasing the learning rate, rank, and number of trainable target modules beyond typical defaults led to improved model performance without significantly increasing training time. Additionally, applying early stopping based on relative validation loss proved effective in reducing training duration while preserving output quality. One of the advantages of the proposed modular pipeline is its ability to decompose the generation process into distinct, reusable steps. Specifically, the parameter list extracted by the Chatbot serves as an intermediate representation that can be directly reused and modified without repeating the initial user dialogue, which enables efficient design iteration. Currently, the full bolt connection is computed and modeled via a single user interaction dialogue, while more experienced users can parse all needed information within one prompt, the architecture currently reminds the user that some parameters are missing, so that new user need up to 5 interactions with the Chatbot, before obtaining the CAD part. Further, the approach allows user unfamiliar with operating CAD systems to use it more efficiently. While dynamic post-hoc edits such as triggering the workflow from a modified FMC have not been systematically tested, they are technically feasible due to the structure of the generated macro steps.

Although the presented bolt dimensioning task was deliberately framed as a simplified use case, it encompasses non-trivial engineering dependencies and safety-relevant design logic. The modular architecture is broadly extensible to other parametric components such as brackets, flanges or gussets. By adjusting the underlying parameter schema, training data and macro templates, the proposed pipeline can be adapted to alternative design tasks that follow a similar dialogue-to-parameter logic. The purpose of this abstraction was to systematically explore GenAI model behavior in a controlled setting, thereby establishing a replicable and extensible foundation for future AI-assisted CAD design processes. Specifically, it offers guidance on dataset generation, model architecture selection, and fine-tuning strategies. These findings can be directly applied to future use cases involving more advanced engineering components and higher design complexity.

Funding

This research was partially funded by the Bavarian Research and Transformation Foundation within the context of the Research Consortium FORAnGen und the Grant Agreement number AZ 1625–24 and partially funded by the European Regional Development Fund within the project ASSiST under Grant Agreement number ROF-SG20-3066–3–12–13.

CRediT authorship contribution statement

Tobias Rosnitschek: Writing – original draft, Visualization, Validation, Methodology, Formal analysis, Data curation, Conceptualization. **Jan Walschewski:** Writing – original draft, Visualization, Software, Methodology, Investigation. **Peter Grohmann:** Writing – original draft, Validation, Investigation, Formal analysis. **Sascha Eckardt:** Writing – review & editing, Validation, Formal analysis, Conceptualization. **Malte Stonis:** Writing – review & editing, Resources. **Bettina Alber-Laukant:** Writing – review & editing, Supervision, Resources. **Stephan Tremmel:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors greatly acknowledge the continuous support of the University of Bayreuth and the Institut für Integrierte Produktion Hannover gGmbH.

Data availability

https://doi.org/10.5281/zenodo.18269442 (All relevant dataset and code are made publicly available on Zenodo)

References

[1] Ślusarczyk B. Industry 4.0 – ARE we ready? PJMS 2018;17:232–48. https://doi.org/10.17512/pjms.2018.17.1.19.

[2] Ikubanni PP, Adeleke AA, Agboola OO, Christopher CT, Ademola BS, Okonkwo J, et al. Present and future impacts of computer-aided design/computer-aided manufacturing (CAD/CAM). JESA 2022;55:349–57. https://doi.org/10.18280/jesa.550307.

[3] Santamaría-Peña J, Benito-Martín M, Sanz-Adán F, Arancón D, Martinez-Calvo M. Reliable low-cost alternative for modeling and rendering 3D objects in Engineering Graphics Education. In: Eynard B, Nigrelli V, Oliveri SM, Peris-Fajarnes G, Rizzuti S, editors. Advances on mechanics, design engineering and manufacturing. Cham: Springer International Publishing; 2017. p. 923–30. https://doi.org/10.1007/978-3-319-45781-9_92.

[4] Ghosh AK, Rentzsch M, Ura S. Fostering empathy in AI-driven Co-creation of CAD models. Procedia CIRP 2025;134:741–6. https://doi.org/10.1016/j.procir.2025.02.206.

[5] Pelau C, Dabija D-C, Ene I. What makes an AI device human-like? The role of interaction quality, empathy and perceived psychological anthropomorphic characteristics in the acceptance of artificial intelligence in the service industry. Comput Hum Behav 2021;122:106855.

[6] Gursoy D, Chi OH, Lu L, Nunkoo R. Consumers acceptance of artificially intelligent (AI) device use in service delivery. Int J Inf Manage 2019;49:157–69. https://doi.org/10.1016/j.ijinfomgt.2019.03.008.

[7] Sarker IH. AI-based modeling: techniques, applications and research issues towards automation, intelligent and smart systems. Sn Comput Sci 2022;3:158. https://doi.org/10.1007/s42979-022-01043-x.

[8] Nti IK, Adekoya AF, Weyori BA. Nyarko-Boateng O. Applications of artificial intelligence in engineering and manufacturing: a systematic review. J Intell Manuf 2022;33:1581–601. https://doi.org/10.1007/s10845-021-01771-6.

[9] Kim S-G, Yoon SM, Yang M, Choi J, Akay H, Burnell E. AI for design: virtual design assistant. CIRP Annals 2019;68:141–4. https://doi.org/10.1016/j.cirp.2019.03.024.

[10] Akay H, Kim S-G. Design transcription: deep learning based design feature representation. CIRP Annals 2020;69:141–4. https://doi.org/10.1016/j.cirp.2020.04.084.

[11] Akay H, Kim S-G. Extracting functional requirements from design documentation using machine learning. Procedia CIRP 2021;100:31–6. https://doi.org/10.1016/j.procir.2021.05.005.

[12] Akay H, Kim S-G. Measuring functional independence in design with deep-learning language representation models. Procedia CIRP 2020;91:528–33. https://doi.org/10.1016/j.procir.2020.02.210.

[13] Tadokoro F, Ghosh AK, Ura S. On the effective co-creation of CAD models by leveraging generative artificial intelligence. Procedia CIRP 2025;134:663–8. https://doi.org/10.1016/j.procir.2025.02.181.

[14] Picard C., Regenwetter L., Nobari A.H., Srivastava A., Ahmed F. Generative optimization: A perspective on AI-enhanced problem solving in engineering 2025. https://arxiv.org/abs/2412.13281.

[15] Regenwetter L, Srivastava A, Gutfreund D, Ahmed F. Beyond statistical similarity: rethinking metrics for deep generative models in engineering design 2025. https://arxiv.org/abs/2302.02913.

[16] Zhu Q, Luo J. Generative transformers for design concept generation 2022. https://arxiv.org/abs/2211.03468.

[17] Lee J-S, Kim T-H, Jeon S-H, Park S-H, Kim S-H, Lee E-H, et al. Automation of trimming die design inspection by zigzag process between AI and CAD domains. Eng Appl Artif Intell 2024;127:107283. https://doi.org/10.1016/j.engappai.2023.107283.

[18] Li K-Y, Huang C-K, Chen Q-W, Zhang H-C, Tang T-T. Generative AI and CAD automation for diverse and novel mechanical component designs under data constraints. Discov Appl Sci 2025;7:315. https://doi.org/10.1007/s42452-025-06833-5.

[19] Wdowik R, Bełzo A. Artificial intelligence-based design of assemblies in the FreeCAD software. TiAM 2025;127:74–82. https://doi.org/10.7862/tiam.2025.1.6.

[20] Khan M.S., Sinha S., Sheikh T.U., Stricker D., Ali S.A., Afzal M.Z. Text2CAD: Generating sequential CAD models from beginner-to-expert level text prompts 2024. https://doi.org/10.48550/arXiv.2409.17106.

[21] Badagabettu A, Yarlagadda SS, Farimani AB. Query2CAD: generating CAD models using natural language queries 2024. https://arxiv.org/abs/2406.00144.

[22] Idrisov B, Schlippe T. Program code generation with generative AIs. Algorithms 2024;17:62. https://doi.org/10.3390/a17020062.

[23] Nijkamp E, Pang B, Hayashi H, Tu L, Wang H, Zhou Y, et al. CodeGen: an open large language model for code with multi-turn program synthesis 2023. https://arxiv.org/abs/2203.13474.

[24] Nijkamp E., Hayashi H., Xiong C., Savarese S., Zhou Y. CodeGen2: lessons for training LLMs on programming and natural languages 2023. https://arxiv.org/abs/2305.02309.

[25] Zhang L, Le B, Akhtar N, Lam S-K, Ngo D. Large language models for computer-aided design: a survey. ACM Comput Surv 2026;58:3787499. https://doi.org/10.1145/3787499.

[26] https://www.freecad.org/index.php.

[27] Machado F, Malpica N, Borromeo S. Parametric CAD modeling for open source scientific hardware: comparing OpenSCAD and FreeCAD Python scripts. PLoS ONE 2019;14:e0225795. https://doi.org/10.1371/journal.pone.0225795.

[28] Anisuzzaman DM, Malins JG, Friedman PA, Attia ZI. Fine-tuning large language models for specialized use cases. Mayo Clin Proc: Digit Health 2025;3:100184. https://doi.org/10.1016/j.mcpdig.2024.11.005.

[29] Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al. LoRA: low-rank adaptation of large language models 2021. https://arxiv.org/abs/2106.09685.

[30] Dettmers T., Pagnoni A., Holtzman A., Zettlemoyer L. QLoRA: Efficient finetuning of quantized LLMs 2023. https://arxiv.org/abs/2305.14314.

[31] Diara F. Open source HBIM and OpenAI: review and new analyses on LLMs integration. Heritage 2025;8:149. https://doi.org/10.3390/heritage8050149.

[32] Chung HW, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, et al. Scaling instruction-finetuned language models 2022. https://arxiv.org/abs/2210.11416.

[33] Budzianowski P, Vulić I. Hello, it's GPT-2 - how can I help you? Towards the use of pretrained language models for task-oriented dialogue systems. In: Proceedings of the 3rd workshop on neural generation and translation. Hong Kong: Association for Computational Linguistics; 2019. p. 15–22. https://doi.org/10.18653/v1/D19-5602.

[34] Ham D, Lee J-G, Jang Y, Kim K-E. End-to-End neural pipeline for goal-oriented dialogue systems using GPT-2. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online: Association for Computational Linguistics; 2020. p. 583–92. https://doi.org/10.18653/v1/2020.acl-main.54.

[35] Rosnitschek T, Walschewski J. From dialogue to design: genAI-based automation of parametric modeling and sizing tasks in CAD workflows - code and datasets. Zenodo. https://doi.org/10.5281/ZENODO.18269442.

[36] https://grok.com.

[37] Lample G, Charton F. Deep learning for symbolic mathematics 2019. https://arxiv.org/abs/1912.01412.

[38] Saxton D, Grefenstette E, Hill F, Kohli P. Analysing mathematical reasoning abilities of neural. models 2019. https://arxiv.org/abs/1904.01557.

[39] Decker KH, Kabus K, Rieg, Frank and Weidermann, Frank and Engelken, Gerhard and Hackenschmidt, Reinhard and Alber-Laukant, Bettina. Maschinenelemente. 21st. München: Hanser; 2024.

[40] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style. High-Performance Deep Learning Library 2019.

[41] Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, et al. HuggingFace's Transformers: state-of-the-art Natural language processing. https://arxiv.org/abs/1910.03771.

[42] Mangrulkar S., Gugger S., Debut L., Belkada Y., Paul S., Bossan B. PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods. 2022. https://github.com/huggingface/peft.

[43] Devlin J, Chang M-W, Lee K, Toutanova KBERT. Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the north. Minneapolis, Minnesota: Association for Computational Linguistics; 2019. p. 4171–86. https://doi.org/10.18653/v1/N19-1423.

[44] Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I. Language models are unsupervised multitask learners. 2019. https://api.semanticscholar.org/CorpusID:160025533.