Targeted control of fast prototyping through domain-specific interface

Yu-Zhe Shi [*1,2] **Mingchen Liu** [*3] **Hanlu Ma** [4] **Qiao Xu** [1] **Huamin Qu** [2,4] **Kun He** [3]
Lecheng Ruan [1] **Qining Wang** [1]

Abstract

Industrial designers have long sought a natural and intuitive way to achieve the targeted control of prototype models—using simple natural language instructions to configure and adjust the models seamlessly according to their intentions, without relying on complex modeling commands. While Large Language Models have shown promise in this area, their potential for controlling prototype models through language remains partially underutilized. This limitation stems from gaps between designers' languages and modeling languages, including mismatch in abstraction levels, fluctuation in semantic precision, and divergence in lexical scopes. To bridge these gaps, we propose an interface architecture that serves as a medium between the two languages. Grounded in design principles derived from a systematic investigation of fast prototyping practices, we devise the interface's operational mechanism and develop an algorithm for its automated domain specification. Both machine-based evaluations and human studies on fast prototyping across various product design domains demonstrate the interface's potential to function as an auxiliary module for Large Language Models, enabling precise and effective targeted control of prototype models.

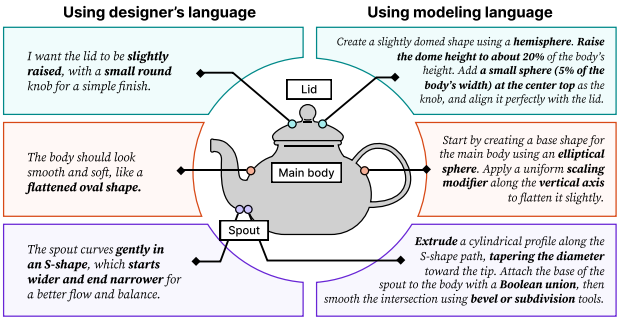

Figure 1. **Designers' language *vs*. modeling language for fast prototyping.** This comparison clearly demonstrates that the designers' language for teapot design is significantly more intuitive than its modeling counterpart, providing a focused creation flow.

1. Introduction

Creating physical prototypes of the target products in a rapid iteration fashion, known as *fast prototyping* (Burns, 1993; Hallgrimsson, 2012), is one of the most critical processes in professional industrial design (Camburn et al., 2017). It helps maintain project timelines, alleviates design fixation, and enables product testing against clear criteria. Unlike production-ready modeling conducted by engineers (Barnhill, 1992), which focuses on refining prototypes into master models for mass-production, fast prototyping emphasizes exploring possibilities, extremes, and feasibility, providing essential references for final design choices. During this process, industrial designers actively and frequently make configurations and adjustments to prototype models, driven by their unpredictable intuitions, insights, and inspirations. Limited by cognitive bandwidth (Lieder & Griffiths, 2020; Griffiths, 2020), industrial designers expect immediate feedback on how their interventions affect the prototype's appearance without the need to switch attention to refining the current version of design (Monsell, 2003), such as manually tuning the parameters of the Computer-Aided Design (CAD) model through the interactive Graphical User Interface (GUI) of modeling engines. Such interruptions may disturb their continuous flow of thinking and creation (Cross, 2023). Consequently, industrial designers have been on the quest of an *interface* for the *targeted* control of prototype models—one

[*]Equal contribution [1]Department of Advanced Manufacturing and Robotics, College of Engineering, Peking University, Beijing, China [2]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR [3]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China [4]Division of Emerging Interdisciplinary Areas, The Hong Kong University of Science and Technology, Hong Kong SAR. Correspondence to: Lecheng Ruan <ruanlecheng@ucla.edu>, Qining Wang <qiningwang@pku.edu.cn>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

that seamlessly implements their intentions as though the modeling controller were an extension of their clay-playing hands, mirroring their thoughts, as shown in Fig. 1. By providing simple Natural Language (NL)-based instructions, the interface should ensure the appropriate realization of their intentions, avoiding under-specification, where interventions are inadequately detailed, and over-specification, where unnecessary details distort the desired outcome.

Recent advancements in Artificial Intelligence (AI) have positioned Large Language Models (LLMs) as a promising medium for interactions between industrial designers and modeling engines. These models can flexibly interpret designers' intentions through NL-based instructions, comprehend and reason over the instructions, and generate corresponding modeling commands. Researchers have begun exploring LLM-based tools to support CAD model creation and editing (Wu et al., 2023; Yuan et al., 2024). However, the potential of LLMs for fast prototyping with its targeted control demands has yet to be fully elicited, particularly in the context of the targetedness-demand fast prototyping (Uusitalo et al., 2024). In the practice of fast prototyping via these tools, the designers' intuitive, subjective, and convention-driven instructions for property assignment and adjustment are often not comprehended precisely, thus the prototype models may not be targetedly controlled. These drawbacks stem from the substantial gaps between the designers' languages for describing their intention, and the underlying programming languages that drives the modeling process through modeling commands.

Designers' languages exhibit three key characteristics: they are high-level, ambiguous, and domain-specific. First, these languages employ semantic references to describe both component parts and their relationships. For example, in teapot design, designers refer to specific parts like "pot lid" and "spout", as well as relationships such as the "relative position of pot lid and spout." This semantic approach ensures the language remains intuitive for designers' practical use (Hannah, 2002). Second, these languages inherently contain ambiguity (Russell, 1923). Consider how the term "length" carries different meanings when applied to different parts—the "length" of a "pot lid" refers to a distinctly different physical property than the "length" of a "spout" (Pei et al., 2011). This ambiguity extends to subjective operational instructions, such as "make the shape of the spout sharper", where the intended degree of modification remains uncertain. Third, these languages demonstrate strong domain-specificity. Each product category employs its own semantic vocabulary. Teapot design uses terms like "pot lid", "spout", and relationships like "articulated" and "fused". In contrast, wardrobe design employs different terminology such as "base plate", "column", "inner", and "beneath". This semantic tailoring extends across all product categories (Micheli et al., 2012). Following design convention, we use *category* to denote products sharing similar functionality, while adopting the term *domain-specificity* from knowledge representation theory to describe this category-specific variation in semantics (Barsalou, 2008)—a convention we maintain throughout this paper. In essence, designers' languages reflect human commonsense knowledge and intuitive thinking patterns. They approach product design through a top-down methodology, breaking down products into semantic components and their interrelationships, while considering their physical properties. This approach aligns with designers' natural cognitive processes and practical needs.

In contrast to designers' languages, programming languages of modeling engines are low-level, precise, and domain-agnostic[1]. First, the atomic constructs of modeling commands—both their concepts and associated operations—contain no abstract meanings beyond basic graphical semantics (Wilkinson, 2012). These commands operate solely on fundamental geometric elements such as points, curves, surfaces, shapes, intersections, and angles. This focus on basic geometric primitives ensures the commands maintain sufficient expressiveness for model construction. Second, modeling commands employ precise quantization and parameterization, diverging from the ambiguous descriptive adjectives common in natural language, such as "big" or "sharp". They also avoid relative comparatives like "larger" or "sharper" that humans typically use when assigning or adjusting properties (Kennedy, 2007). Third, the absence of abstract semantics in modeling commands renders them domain-agnostic rather than domain-specific in industrial design applications (Mernik et al., 2005). While some encapsulated functions may provide abstract operations, such as batch processing similar parts, these functions remain bound by atomic operations and reflect the modeling engine's rendering mechanisms rather than addressing domain-specific design requirements (Shi et al., 2023b). In essence, modeling languages follow the logic of graphics description languages and avoid maintaining prefabricated components for specific requirements. Instead, they require employing a bottom-up approach and building meaningful semantic functions through the composition of atomic elements.

To bridge the gaps between designers' languages and modeling languages, we propose an interface built upon LLMs that translates designers' high-level, ambiguous, and domain-specific intentions into low-level, precise, and domain-agnostic modeling commands. Our design principles emerge from a systematic study of fast prototyping practices and their cognitive foundations. The first principle establishes the interface as an intermediate-level

[1]The readers may refer to a commonly used, open-source CAD modeling language, FreeCAD. Visit the website at https://www.freecad.org/

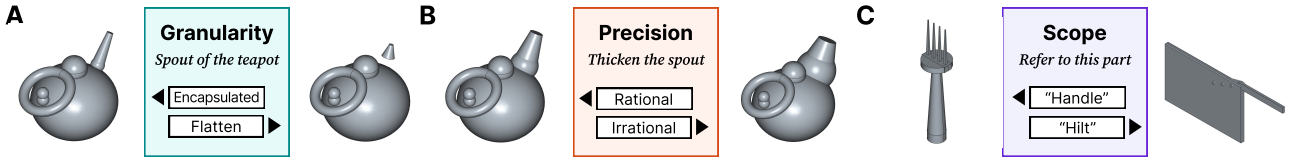

Figure 2. **Illustrations of the gaps against the targeted control.** **(A)** Designers' language contains semantically rich, encapsulated concepts. Improper handling of abstraction levels can lead to undesired semantic flattening. **(B)** Designers' language relies heavily on subjective references that require commonsense knowledge for proper interpretation. Improper specification of instructions may result in physically or functionally inappropriate operations. **(C)** Designers' language is inherently domain-specific, where similar physical properties may be referenced by different semantic identifiers across product design domains.

Domain-Specific Language (DSL) between the two languages (Fowler, 2010). This DSL uses designers' concepts and relationships as semantic identifiers, with each language construct encapsulating the atomic modeling commands needed to render parts or execute operations based on context. The abstraction level of this DSL must carefully balance between the finest granularity of designers' languages and the broadest semantics of modeling languages (Abelson & Sussman, 1996). The second principle addresses targeted prototype control through precise interpretation of designers' intentions. The interface DSL must bind concepts (including parts and relationships) to their permissible operations. For instance, the "length" of a "spout" permits operations like "increase" or "decrease," but not "twist" or "smooth"—establishing a binding between a concept and its allowable second-order adjustments. Similarly, the "relative position of pot lid and spout" can be specified through directional descriptions, creating a binding between a relationship and its permissible first-order assignments (Büring, 2005). These bindings ensure the interface remains both expressive and intuitive (Shi et al., 2024d). The third principle acknowledges the domain-specific nature of the interface. While the DSL offers benefits of hierarchical encapsulation and precise knowledge representation, each DSL instance serves only one product design domain, as concepts and operations vary significantly across domains. Manual creation of domain-specific DSLs requires substantial effort and becomes impractical for design agencies given the diversity of consumer products (Shi et al., 2024a). Therefore, to ensure practical utility, we propose an automated mechanism for domain-specific interface specification.

This study identifies the gaps between designers' and modeling languages in fast prototyping and proposes an interface to bridge these gaps. Our work makes three primary contributions: (i) we conduct a systematic analysis of the gaps between designers' languages and modeling languages in practice (Sec. 2), deriving the design principles for the interface grounded in cognitive theory; (ii) we develop the interface architecture, detail its operational mechanisms, and design an algorithm for automated domain specification (Sec. 3); (iii) we evaluate the interface through both machine-based assessments and human studies (Sec. 4), demonstrating its potential to bridge communication gaps between diverse-minded experts while serving as an auxiliary module for LLM-based interaction tools.

2. The gaps against the targeted control

In this section, we systematically study the practices of fast prototyping in industrial design and analyze the three gaps between the designers' and modeling languages, within the three dimensions: level of abstraction (Sec. 2.1), semantic precision (Sec. 2.2), and domain-specificity (Sec. 2.3). According to these investigations, we propose the corresponding design principles of the interface (Sec. 2.4).

2.1. Level of abstraction and granularity

This gap pertains to the level of abstraction and granularity of the language. In semantic theory, this aspect critically influences the conceptual density and operational granularity of language (Levinson, 2003). Designers employ a high-level, semantically rich language that encapsulates complex concepts and relationships inherent to specific design contexts. This language is closer to human natural language, which is inherently high-level and contextually enriched. On the other hand, modeling commands utilize a low-level, syntactically oriented language primarily focused on the precise execution of graphical operations. These commands operate at a lower level of abstraction, dealing directly with the geometrical and mathematical representations required to render objects, without the semantic enrichment seen in designers' language. As shown at Fig. 2A, the designer's semantic *"spout of teapot"* essentially encapsulates the modeling descriptions *"a hollow cylinder thick at the proximal and thin at the distal"*.

2.2. Semantic precision and determinacy

This gap refers to the semantic precision and determinacy of the language. In the realm of semantics, this aspect is critical as it affects the interpretability and operational

specificity of the language (Lyons, 1995). Designers' language exhibits a higher degree of semantic indeterminacy, allowing for contextual and subjective interpretation and flexibility. This ambiguity is typical in natural language, where meanings can shift based on context, usage, and individual perception. In contrast, modeling commands demonstrate a semantic determinacy, with each command having a fixed and unequivocal meaning, devoid of contextual interpretation. This precision is necessary for the accurate execution of graphical tasks, where ambiguity could lead to errors in design implementation. As shown at Fig. 2B, the designer's intention of *"thicken the spout"* implies a rational grounding of the diameter-related parameters, which can be irrational if not properly specified.

2.3. Lexical scope and semantic applicability

This gap highlights the lexical scope and semantic applicability of the language. From a semantic perspective, this involves the domain specificity or generality of the language (Saeed, 2015). Designers' language is characterized by domain-specific lexicons, tailored to particular design fields or product categories. This specialization enables designers to communicate effectively within their specific context, using terminology and concepts that are relevant to their particular field. Conversely, modeling commands are domain-agnostic, designed to be universally applicable across various design fields without modification. This universality ensures that modeling engines remain versatile and broadly applicable, at the cost of not providing specialized tools that cater to the nuances of specific design domains. As shown at Fig. 2C, referring to parts with similar physical properties within different product design domains require distinct semantic identifiers, such as *"handle" in "fork"* and *"knit" in "cleaver knife"*.

2.4. Design principles for the interface

The three gaps highlight fundamental differences in design conceptualization and communication. While designers' language is semantically rich, contextually flexible, and specialized, modeling commands are syntactically precise, context-independent, and generalized. To bridge these gaps, we propose three design principles derived from the tacit knowledge of industrial design practices.

Balanced hierarchy of abstraction This principle addresses the alignment between designers' semantic language and modeling systems' rigid syntax through an intermediate-level DSL. This DSL is designed to preserve *naturalness* by maintaining the conceptual richness of designers' language (Myers et al., 2004; Hindle et al., 2016), such as terms like "organic form" or "balanced proportions", while abstracting low-level geometric operations like extrusion tolerances or fillet radii into executable functions. Crucially, the interface ensures that granular modeling details remain transparent to designers, allowing them to focus on intent rather than implementation. For instance, an instruction to *"soften the edges for an ergonomic feel"* maps to parametric functions adjusting fillet radii without exposing the underlying modeling commands. To achieve this balance, the DSL employs *function-like natural language constructs* that encapsulate atomic graphical elements into reusable semantic identifiers (Bryant & O'Hallaron, 2011; Gulwani et al., 2017). This is achieved through a reciprocal development process: (i) top-down decomposition of designers' domain-specific terminology (*e.g.*, breaking "part handle" into geometric primitives like "angled cylinders with blended intersections") establishes a semantic anchor for translation; while (ii) bottom-up encapsulation aggregates atomic modeling commands (*e.g.*, Boolean union, surface lofting) into higher-order functions (*e.g.*, "generate grip geometry") that mirror designers' mental models. By harmonizing these bidirectional strategies, the DSL operates at a "meso-scale" abstraction—neither oversimplifying creative intent nor overcomplicating execution—thereby sustaining designers' cognitive flow while guaranteeing technical precision.

Articulated concepts and operations This principle ensures precise translation of designers' intentions through concept-operation bindings that reflect domain-specific design logic. This is achieved by binding concepts (*i.e.*, geometric parts or relational configurations) to spaces of permissible operations that reflect domain-specific design logic. These bindings enforce semantic coherence by restricting operations to contextually valid transformations, thereby preventing mismatches between intent and execution. Hence, the interface employs a *hierarchical syntax* where operations are linked to governing concepts (Chomsky, 1957; Shi et al., 2024c). The effective field of an operation is confined to the attributes of the associated concept. First-order operations (*e.g.*, static descriptors like "smooth" or "short") directly modify attribute values, while second-order operations (*e.g.*, dynamic comparatives like "smoother" or "shorter") adjust the incremental magnitude of these modifications (Emde & Rollinger, 1983). Critically, the system maintains spatial-temporal continuity (Kuipers, 1986): attributes remain stable until explicitly modified, and operational effects stay consistent until recalibrated. This structure balances expressive power with cognitive clarity (Felleisen, 1991), enabling precise intent articulation while avoiding excessive modeling choices.

Automated domain specification This principle confronts the practical challenge of scaling the interface's utility across diverse product design domains, where manual crafting of DSLs becomes prohibitively costly due to the vast heterogeneity of concepts and operations (Pei et al.,

2011). The interface integrates a mechanism that autonomously derives domain-specific linguistic and operational landscapes by modeling the joint distribution of designers' language and modeling command patterns. Drawing from cognitive theories of *concept acquisition* (Kemp & Tenenbaum, 2008), where humans construct mental models by iteratively sampling and generalizing from exemplars (Tenenbaum et al., 2011), the mechanism leverages LLMs as repositories of commonsense knowledge to build generative models of design concepts (Yildirim & Paul, 2024). By framing LLM interactions as a stochastic sampling process, the system emulates human-like concept formation, where repeated exposure to diverse examples, mediated by the LLM's latent knowledge, gradually defines the boundaries and operational rules of domain-specific concepts. This approach not only captures the breadth of real-world design variation but also ensures that the synthesized DSL remains grounded in both technical feasibility and designers' linguistic norms. This approach transforms manual specification into a scalable, knowledge-driven workflow (Shi et al., 2024a; 2025), ensuring comprehensive coverage while maintaining practicability.

3. The interface for the targeted control

In this section, we describe our proposed methodology for the target control of fast prototyping. We first present the problem formulation (Sec. 3.1), followed by the interface representation (Sec. 3.2), and we describe the algorithm for the domain adaptation of the interface (Sec. 3.3; Fig. 3).

3.1. The intention communication problem

Original formulation We consider the original problem of communicating the designers' intention with the modeling engine as a mapping $f : \mathcal{I}_c \mapsto \mathcal{M}$ that transforms the space of intention $\mathcal{I}_c$ to the space spanned by the possible command constructs of the modeling engine $\mathcal{M}$. Each instance of intention $I \in \mathcal{I}_c = \{I_l, I_p\}$ is translated to a sequence of modeling commands $M \in \mathcal{M} = \langle m_1, \ldots, m_{|M|} \rangle$, where I_l represents the intention of configuration on the global shape, structure, and layout of the set of parts of the target product, and I_p denotes the intention of configuration on the parts of the target product; m_k represents a program indicating a modeling command. Hence, the objective of modeling according to the designers' intention can be maximizing the likelihood

$$\arg\max_{M} \; p(M \mid I). \tag{1}$$

Integrating interface into the formulation Directly estimating the likelihood $p(M \mid I)$ can be intractable, given the substantial gaps between the designers' and modeling languages. We introduce the intermediate-level DSL, the interface $\mathcal{L}_c$ into the problem, decomposing the original mapping into a combination of mappings $g : \mathcal{I}_c \mapsto \mathcal{L}_c$ and $h : \mathcal{L}_c \mapsto \mathcal{M}$, where $f = h \circ g$. The generative process can then be formulated as $\mathcal{I}_c \xrightarrow{g} \mathcal{L}_c \xrightarrow{h} \mathcal{M}$. The interface DSL $\mathcal{L}_c = \{\mathcal{S}_c, \Lambda_c\}$, where $\mathcal{S}_c$ represents the set of DSL constructs regarding the global shape, structure, and layout of the set of parts of the target product, and Λ_c denotes the set of DSL constructs regarding the parts of the target product. An arbitrary permissible program generated by the DSL can thereby be represented as $L \in \mathcal{L}_c^* = \{\text{prog} \mid \text{prog} \in \mathcal{S}_c^* \cup \Lambda_c^*\}$. Treating the interface as a latent variable in the likelihood maximization objective, we have

$$\begin{aligned} &\arg\max_{M} p(M \mid I) \\ &= \arg\max_{L} \sum_{L \in \mathcal{L}_c^*} p(M \mid L) p(L \mid I), \end{aligned} \tag{2}$$

where $p(L \mid I)$ and indicates $p(M \mid L)$ generative functions g and h respectively. Leveraging this latent variable formulation, we obtain a relatively tractable estimation of $p(M \mid I)$ with the interface as the medium.

3.2. Representation of the interface

Syntax of the DSL The local part construct is defined as $\Lambda_c = \langle \lambda, a_\lambda, o_\lambda, v_\lambda, v'_\lambda \rangle$, where λ denotes the semantic identifier of the referred part, $a_\lambda = (a_{\lambda,i}, a_{\lambda,p})$ indicates the physical subpart $a_{\lambda,i}$ (*e.g.*, a handle or its constituent cylinder) and its associated physical property $a_{\lambda,p}$ (*e.g.*, length, diameter), o_λ represents the operation that is permissible to apply on the physical property, v_λ denotes the quantitative value of the physical property (*i.e.*, first-order value assignment), and v'_λ indicates the quantitative value of the extent of the operation (*e.g.*, second-order value adjustment). The global construct is defined as $\mathcal{S}_c = \langle s, a_s, o_s, v_s, v'_s \rangle$, where s denotes the semantic identifier of the referred relationship between parts, a_s indicates the joint descriptor of the physical subparts within the two parts (*e.g.*, the relative position of spout and pot lid), o_s represents the combinatorial operation over the relationship (*e.g.*, make the relative position closer), o_s represents the combinatorial operation applied to the relationship, and v'_s combines the values of the two operations linked to the physical properties. The syntax of these language constructs mirrors the designer's intention, with each unit maintaining an identical structural pattern.

Hierarchical modeling According to our semantic binding design principle, we describe the DSL that generates language constructs using a top-down approach. In this hierarchy, lower-level elements are controlled by their immediate higher-level elements: parts are generated based on the intention, physical subparts are generated from the part,

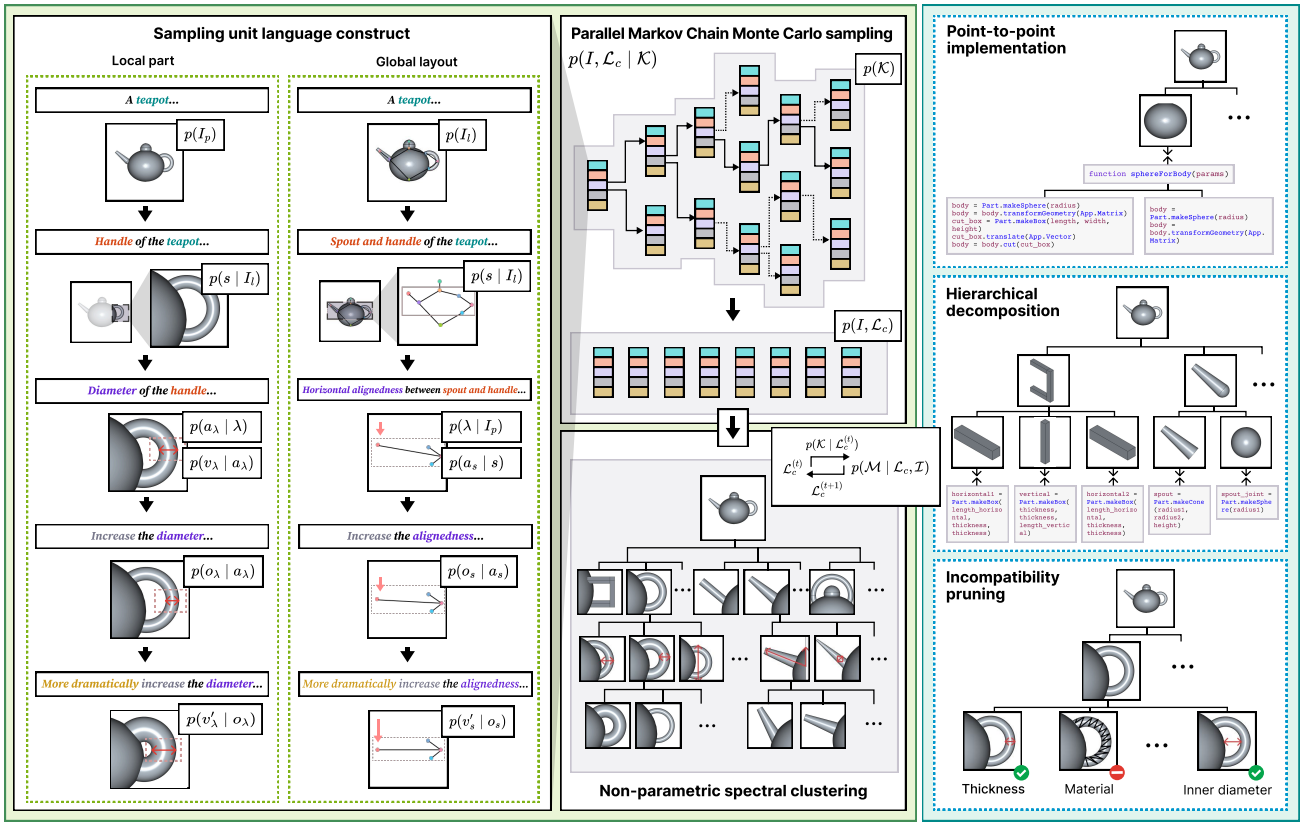

Figure 3. **Representation and domain adaptation of the interface.** This illustration conveys a running example of adapting the interface to the teapot design domain. The iterative optimization alternates **construct expansion (left)** with **feasibility validation (right)**.

operations are determined by physical properties, quantitative values are assigned according to their corresponding physical properties and operations. To maintain DSL expressivity that accommodates the high variability in designers' intentions for the same target product (Shi et al., 2024b), we model the relationships between adjacent levels as conditional probabilistic generation (Wu et al., 2019). This results in a tree-structured hierarchical probabilistic model, which can be formulated as

$$\begin{aligned} & p(I, \mathcal{L}_c) \\ =& p(I_l, I_p, s, a_s, o_s, v_s, v_s', \lambda, a_\lambda, o_\lambda, v_\lambda, v_\lambda') \\ =& p(v_\lambda' \mid o_\lambda) p(v_s' \mid o_s) p(v_\lambda \mid a_\lambda) p(o_\lambda \mid a_\lambda) \\ & p(v_s \mid a_s) p(o_s \mid a_s) p(a_\lambda, a_s \mid \lambda, s) p(s, \lambda \mid I_l, I_p), \end{aligned} \tag{3}$$

representing the overall joint distribution of the designers' language and the interface.

3.3. Domain adaptation of the interface

The target of domain adaptation is to automate the design of the interface DSL $\mathcal{L}_c$ given any domain of design c (*e.g.*, teapot, sofa, fence, racket, *etc.*). According to our design principle of domain adaptation, we regard interacting with LLMs as a Bayesian inference from the commonsense knowledge base $\mathcal{K}$ with prior distribution $p(\mathcal{K})$, serving as the prior of a latent generative model

$$p(I, \mathcal{L}_c, \mathcal{K}) = p(I, \mathcal{L}_c \mid \mathcal{K}) p(\mathcal{K}), \tag{4}$$

where $\mathcal{L}_c$ is iteratively refined through stochastic sampling. To achieve this objective, wherein the interface DSL must accept the designers' language and generate the modeling language, we derive the iterative algorithm with a reciprocative optimization framework (see Figs. 3 and 4).

Sampling from commonsense priors To initialize $\mathcal{L}_c$, we sample language constructs Λ_c and $\mathcal{S}_c$ from the LLM-mediated prior $p(\mathcal{K})$, following their hierarchical architecture level-by-level. This is implemented as a Markov Chain Monte Carlo (MCMC) process. The MCMC is initialized by drawing M seed sample constructs $\{L_1^{(0)}, \ldots, L_M^{(0)}\}$ via LLM queries. For each seed sample $L_i^{(0)}$, N Metropolis-Hastings steps are performed to explore the neighborhood of $\mathcal{K}$, specified as

$$L^{(t+1)} \sim q(L' \mid L^{(t)}) \min\left(1, \frac{p(L' \mid \mathcal{K})}{p(L^{(t)} \mid \mathcal{K})}\right), \tag{5}$$

where the proposal distribution q is defined by LLM-generated perturbations. This process yields $M \times N$ samples $\{L_i^{(t)}\}$, ensuring diversity while preserving domain coherence of the samples. The sampled constructs are clustered into semantically coherent units exploiting a non-parametric Dirichlet Process Mixture Model (DPMM) to handle the unknown and variable number of domain concepts. Let θ_k denote the parameters of the k-th cluster (*e.g.*, one type of the handle within teapot), with a Chinese Restaurant Process prior (Blei et al., 2010), each sample is assigned to cluster k with probability proportional to its likelihood under $p(L_i^{(t)} \mid \theta_k)$ modeled as a multinomial distribution over the elements $\langle \lambda, a_\lambda, o_\lambda, \dots \rangle$ and $\langle s, a_s, o_s, \dots \rangle$. The DPMM induces a hierarchical tree structure, where nodes closer to the root represent macro concepts (*e.g.*, part), and nodes closer the leaves encode fine-grained variants (*e.g.*, physical properties).

Reciprocative optimization of granularity Though the sampling drawing and spectral clustering processes constructs the initial structure of the interface, there is one side missing—the interface not only accepts the designers' language, but also outputs the modeling language. The actual granularity of the interface DSL is determined by the both sides. Therefore, we exploit an Expectation-Maximization (EM) Algorithm-like reciprocative optimization that alternates between *construct expansion* (*i.e.*, Expectation) and *feasibility validation* (*i.e.*, Maximization). The Expectation step expands the DPMM tree structure by sampling new constructs and proposing new clusters using the current interface $\mathcal{L}_c^{(t)}$ as a prior. The Maximization validates the constructs of the interface DSL via simulated synthesize over the programming library of the modeling language. The validation phase categorizes each construct into three feasibility classes: (i) *point-to-point implementation*, which can be implemented by atomic operations or predefined functions within the modeling language, then the construct should be maintained; (ii) *hierarchical decomposition*, which may be implemented by some encapsulated functions within the modeling language, the construct sample should be regenerated at a finer granularity; (iii) *incompatibility pruning*, which cannot be implemented by the modeling language, such as the physical property "material", the construct should be removed. After this Maximization-step, the algorithm goes into the next iteration. The likelihood is maximized by iterating

$$\mathcal{L}_c^{(t+1)} = \arg\max_{\mathcal{L}_c^{(t)}} \mathbb{E}_{p(\mathcal{K} \mid \mathcal{L}_c^{(t)})}[\log p(\mathcal{M} \mid \mathcal{L}_c, \mathcal{I})], \quad (6)$$

where $p(\mathcal{K} \mid \mathcal{L}_c^{(t)})$ indicates the mechanism to focus exploration of $\mathcal{K}$ based on the interface's current state. Intuitively, this mechanism indicates that given the current interface state, what areas of the commonsense knowledge are most relevant to refine or expand it, mirroring human designers' iterative refinement—prior knowledge is recalled in context of current progress.

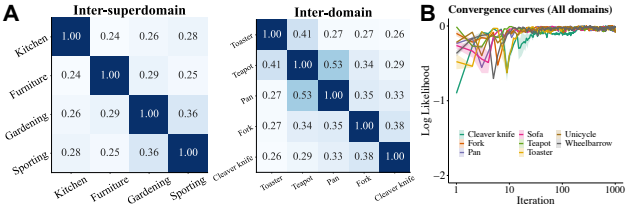

Figure 4. **Running status of the domain adaptation algorithm. (A)** Distinctions on language construct distributions cross super-categories and domains. **(B)** Convergence curves of the domain adaptation algorithm on eight domains.

4. Experiments and results

In this section, we report and discuss the results of our experiments for assessing the utility of the interface for targeted control. We start from describing our designer-participated fast prototyping tasks (Sec. 4.1), along with the metrics to measure the targetedness of the control of designers' intentions on the modeling engine (Sec. 4.2). Afterwards, we introduce the alternative methods equipping integrating different interfaces used for comparison (Sec. 4.3). Finally, we report and analyze the experimental results both quantitatively and qualitatively (Sec. 4.4).

4.1. Experimental setups

Experimental protocol We evaluate the targetedness of modeling control across two dimensions: rendering consistency and information clarity. We design a realistic fast prototyping scenario to simulate the targetedness-demanding design practices. Given the fundamental differences between fast prototyping and production-ready modeling (Hallgrimsson, 2012), we limit the iteration count to ten rather than allowing unlimited iterations until the modeling becomes subjectively "satisfying". At the beginning, participants receive information about their target domain of product design. During each iteration, participants provide one natural language instruction to the interface and receive images of rendered models generated by our interface and alternative methods. Participants then rank these images based on how closely they match their intended effect. We collect these rankings for consistency measurement. Additionally, we gather the modeling programs generated during this process for machine-based evaluation.

Participants The human study includes 50 participants, each holding at minimum a Bachelor's degree in industrial design or related fields. All participants receive standardized task instructions and provide informed consent. For

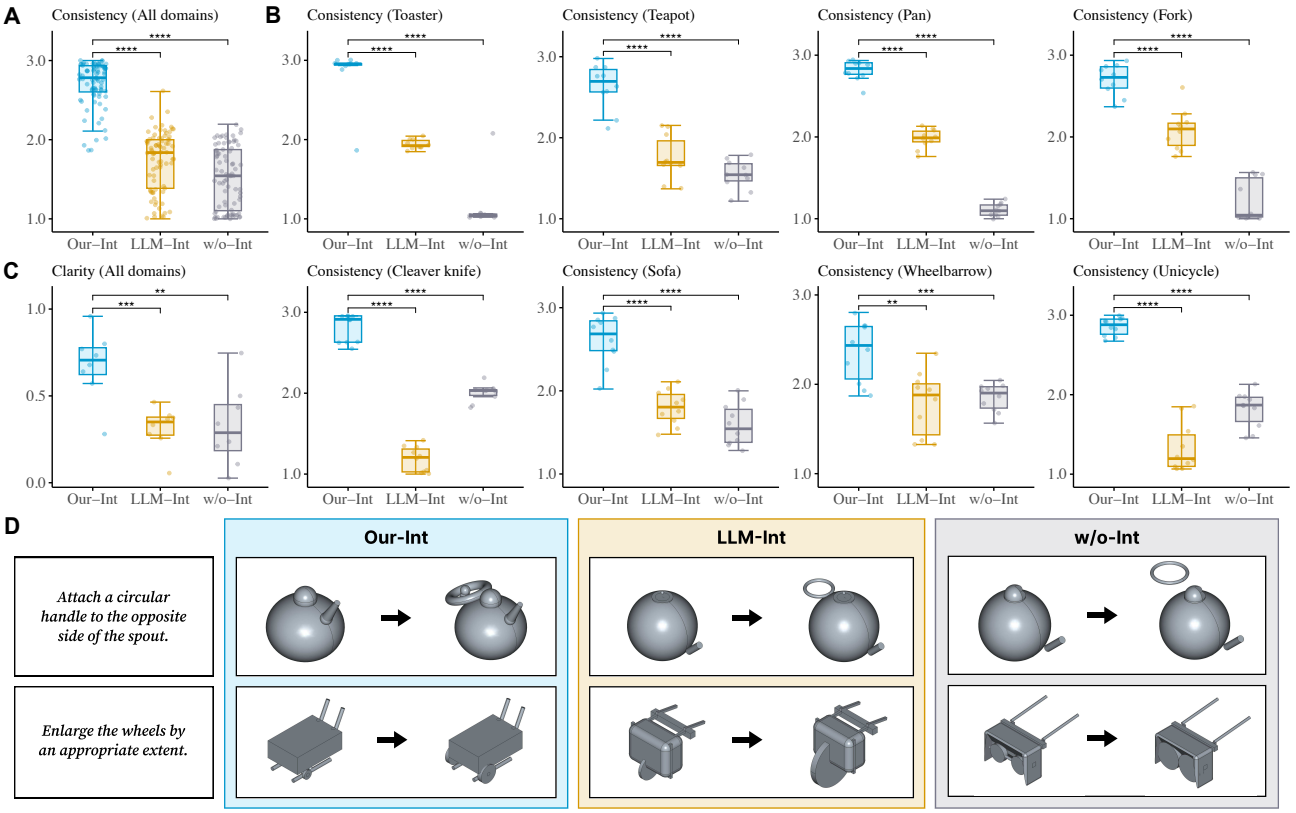

Figure 5. **Quantitative and qualitative results.** **(A)** Results on rendering consistency assessment across all domains. **(B)** Isolated domain-wise results on rendering consistency assessment. **(C)** Results on information clarity assessment across all domains. **(D)** Showcases collected from the fast prototyping tasks. The leftmost text is the designer's instruction. Each pair of visualizations indicates the models before (left to the arrow) and after (right to the arrow) the intervention.

ethical considerations, please refer to Appx. B.

Materials To ensure experimental generality, we select eight product design targets from major consumer market supercategories (Pei et al., 2011): *kitchen appliances* (*i.e.*, teapot, pan, fork, toaster, and cleaver knife), *furniture* (*i.e.*, sofa), *gardening equipment* (*i.e.*, wheelbarrow), and *sporting goods* (*i.e.*, unicycle). We choose products across these supercategories to examine inter-supercategory differences and selected multiple products within supercategories to study intra-supercategory domain-wise variations. To thoroughly test interface utility, we intentionally select challenging products with irregular geometric features, such as teapots, sofas, wheelbarrows, and unicycles.

4.2. Evaluation metrics

Rendering consistency assessment We measure the consistency between participant intentions and modeling results using the ranked order of results from each intervention step, following established methodology in similar human studies (Lake et al., 2015). We treat the iterative process as a Markov Process, considering only transitions between adjacent design versions, as each intervention represents a state transition. Therefore, we apply uniform weighting to step-wise result rankings across each design task, with higher scores indicating better performance.

Information clarity assessment For each participant intervention step, we analyze the corresponding modeling programs for uncertainty by examining the modeling program library documentation (Chaitin, 1977). We analyze function Abstract Syntax Trees (ASTs) and calculate uncertainty based on the cumulative depth of specified program constructs along the resulting AST. Greater cumulative depth indicates lower uncertainty and higher information clarity. Higher scores represent better performance.

4.3. Alternative methods

Our comparison with alternative methods are required to answer two key questions. First, is our interface architecture necessary for the targeted control, specifically the transformation from standard likelihood maximization to our latent variable interface approach? Second, are our interface implementation principles, particularly the hier-

archical DSL-based representation, essential for the targeted control? To address the first question, we develop `w/o-Int`, an alternative model that processes designers' language directly without interfaces. For the second question, we created `LLM-Int`, an interface implemented through pure LLM-based prompt engineering, as a direct comparison to our proposed interface. We refer to our DSL-represented interface model as `Our-Int`. Each domain-specific instance of `Our-Int` is automatically designed through the algorithm described in Sec. 3.3. All methods are built upon state-of-the-art text-to-modeling program generators (Wu et al., 2023; Yuan et al., 2024).

4.4. Results

Rendering consistency Paired samples t-tests reveal that `Our-Int` significantly outperforms both alternative methods across all eight domains in rendering consistency (`Our-Int` *vs.* `LLM-Int`: $t(7183) = 80.60, p < .0001$; `Our-Int` *vs.* `w/o-Int`: $t(7246) = 63.81, p < .0001$; see Fig. 5A)[2]. Domain-specific analyses also show significant superiority of our approach in isolated comparisons (see Fig. 5B). Furthermore, `LLM-Int` demonstrates significant overall improvement compared to `w/o-Int` ($t(7127) = 12.37, p < .0001$; see Fig. 5A), supporting the necessity of the interface structure. These findings validate our approach from a black-box outcome perspective.

Information clarity Paired samples t-tests demonstrate that `Our-Int` significantly outperforms both alternative methods across all eight domains in information clarity (`Our-Int` *vs.* `LLM-Int`: $t(14) = 4.413, p < .001$; `Our-Int` *vs.* `w/o-Int`: $t(14) = 3.277, p < .01$; see Fig. 5C). These results validate our proposal from a white-box intermediate result perspective.

Discussion Comparison between `Our-Int` and alternative methods reveals significant advantages in targeted control: (i) while alternative methods only modify parameters at a coarse level, `Our-Int` successfully handles fine-grained instructions such as "attach" and "opposite" (see Fig. 5D); (ii) `Our-Int` precisely interprets subjective references to values and operation extents, and also domain-specific references to parts and physical properties, while alternative models often reference incorrect objects for operations, apply inappropriate operation types, and generate physically implausible results (*e.g.*, oversized wheels conflicting with the framework in the wheel size adjustment case; see Fig. 5D). These qualitative observations further support our interface's utility. We notice that the increased complexity of generated modeling programs led to higher probabilities of run-time errors during modeling engine execution. However, this limitation is relatively minor, as current LLM-based program fixers can iteratively refine programs based on error messages (Madaan et al., 2023).

[2]The varying degrees of freedom in different t-tests result from some participants only selecting results with highest consistency without completing the full ranking. Such incomplete data cases represent less than 2% of all collected data.

5. General discussion

In this study, we propose an interface to enhance targeted control of fast prototyping by bridging the gap between designers' languages and modeling languages. Experimental results demonstrate our interface's effectiveness and suggest its potential as an auxiliary module for LLMs in broader human instruction specification tasks.

On the cost of inter-domain communication Communication costs present a significant challenge in collaborative efforts. Domain experts from different backgrounds come with diverse mindsets, thereby using distinct DSLs accordingly, which often remain misaligned without extensive trial-and-error collaboration (Shi et al., 2023a). This challenge is particularly evident in industrial design and manufacturing, where designers must interpret high-level Part-A indicators, requirements, and preferences; while modeling engineers and their tools must implement low-level designs within spatial, graphical, and physical constraints. The substantial language differences between these groups traditionally result in high communication costs. Our interface aims to reduce these costs, potentially enabling a more balanced and reciprocal relationship between design creation and prototype implementation. The broader prospect of our methodology is to provide human experts with a *natural language programming* interface that enables direct high-level instructions, while low-level executions are encapsulated under semantic identifiers within the interface, remaining transparent to domain experts.

On the domain generalizability of the interface A key consideration is whether interface representations should be specific or general across product design domains. While a universal interface might seem appealing, creating a truly general *one-size-fits-all* solution for all product design domains presents significant challenges. Although theoretically possible to develop a comprehensive designers' language covering all requirements, such a system would become overwhelmingly complex, with overlapping semantics and intertwined references that designers would find impractical. Conversely, attempts to simplify this universal interface would inevitably compromise its representational capabilities, leading to the expressivity-complexity dilemma (Abelson & Sussman, 1996). Therefore, rather than pursuing elusive generality, a more pragmatic approach may be to focus on developing and automating the adaptation of domain-specific interfaces.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China under Grants 52475001 and RGC GRF Grant 16210321. Q. Xu is a visiting student at Peking University from University of Science and Technology of China.

Impact statement

This work explores bridging high-level industrial design instructions and low-level modeling implementations in a cost-efficient manner for fast prototyping of industrial designs through the automatic design and deployment of domain-specific interfaces. Our proposed methodology extends beyond industrial design to address a fundamental challenge in smart manufacturing: aligning requirements from design teams (Part-A) with production capabilities (Part-B). This alignment has long been a costly bottleneck that limits manufacturing efficiency, particularly given the current trend toward small-batch, multi-variety production. The challenge manifests in three critical gaps: level of abstraction, semantic precision, and domain-specificity, which together hinder rapid product iteration.

In response to these challenges, this work highlights the importance of managing tacit knowledge in manufacturing. The *natural DSL* used by specific Part-A or Part-B teams should be *standardized toward a sound and complete DSL* within their respective domains to avoid insufficient detail, ambiguity, and excessive jargon. Interfaces serve as information-preserving translators that map standardized Part-A DSL programs to executable Part-B DSL programs, enabling customized point-to-point communication between any pair of Part-A and Part-B components.

From a system-level perspective of the entire manufacturing supply chain, all suppliers are interconnected through a communication *busline*. The interfaces provide suppliers with access to this busline while incorporating the unified requirements of the Original Equipment Manufacturer (OEM) that governs the supply chain. These interfaces can be customized for each supplier at low cost because they are designed through highly automated processes. This automation is practical given the substantial volume of historical communication data records between Part-A and Part-B components, combined with LLMs for manufacturing that serve as commonsense knowledge bases. This standardization methodology has the potential to improve both the quality and efficiency of manufacturing supply chains.

References

Abelson, H. and Sussman, G. J. *Structure and interpretation of computer programs*. The MIT Press, 1996.

Barnhill, R. E. *Geometry processing for design and manufacturing*. SIAM, 1992.

Barsalou, L. W. Grounded cognition. *Annual Review of Psychology*, 59(1):617–645, 2008.

Blei, D. M., Griffiths, T. L., and Jordan, M. I. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*, 57(2):1–30, 2010.

Bryant, R. E. and O'Hallaron, D. R. *Computer systems: a programmer's perspective*. Prentice Hall, 2011.

Büring, D. *Binding theory*. Cambridge University Press, 2005.

Burns, M. *Automated fabrication: improving productivity in manufacturing*. Prentice-Hall, Inc., 1993.

Camburn, B., Viswanathan, V., Linsey, J., Anderson, D., Jensen, D., Crawford, R., Otto, K., and Wood, K. Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science*, 3:e13, 2017.

Chaitin, G. J. Algorithmic information theory. *IBM Journal of Research and Development*, 21(4):350–359, 1977.

Chomsky, N. *Syntactic Structures*. Mouton de Gruyter, 1957.

Cross, N. *Design thinking: Understanding how designers think and work*. Bloomsbury Publishing, 2023.

Emde, W. and Rollinger, C.-R. The discovery of the equator or concept driven learning. In *International Joint Conference on Artificial Intelligence*, 1983.

Felleisen, M. On the expressive power of programming languages. *Science of computer programming*, 17(1-3): 35–75, 1991.

Fowler, M. *Domain-specific languages*. Pearson Education, 2010.

Griffiths, T. L. Understanding human intelligence through human limitations. *Trends in Cognitive Sciences*, 24(11): 873–883, 2020.

Gulwani, S., Polozov, O., Singh, R., et al. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

Hallgrimsson, B. *Prototyping and modelmaking for product design*. Laurenceking, 2012.

Hannah, G. G. *Elements of design: Rowena Reed Kostellow and the structure of visual relationships*. Princeton Architectural Press, 2002.

Hindle, A., Barr, E. T., Gabel, M., Su, Z., and Devanbu, P. On the naturalness of software. *Communications of the ACM*, 59(5):122–131, 2016.

Kemp, C. and Tenenbaum, J. B. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

Kennedy, C. Vagueness and grammar: The semantics of relative and absolute gradable adjectives. *Linguistics and Philosophy*, 30:1–45, 2007.

Kuipers, B. Qualitative simulation. *Artificial Intelligence*, 29(3):289–338, 1986.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Levinson, S. C. *Space in language and cognition: Explorations in cognitive diversity*. Cambridge University Press, 2003.

Lieder, F. and Griffiths, T. L. Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and Brain Sciences*, 43:e1, 2020.

Lyons, J. *Linguistic semantics: An introduction*. Cambridge University Press, 1995.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Mernik, M., Heering, J., and Sloane, A. M. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, 2005.

Micheli, P., Jaina, J., Goffin, K., Lemke, F., and Verganti, R. Perceptions of industrial design: The "means" and the "ends". *Journal of Product Innovation Management*, 29(5):687–704, 2012.

Monsell, S. Task switching. *Trends in Cognitive Sciences*, 7(3):134–140, 2003.

Myers, B. A., Pane, J. F., and Ko, A. J. Natural programming languages and environments. *Communications of the ACM*, 47(9):47–52, 2004.

Pei, E., Campbell, I., and Evans, M. A taxonomic classification of visual design representations used by industrial designers and engineering designers. *The Design Journal*, 14(1):64–91, 2011.

Russell, B. Vagueness. *The Australasian Journal of Psychology and Philosophy*, 1(2):84–92, 1923.

Saeed, J. I. *Semantics*. John Wiley & Sons, 2015.

Shi, Y.-Z., Li, S., Niu, X., Xu, Q., Liu, J., Xu, Y., Gu, S., He, B., Li, X., Zhao, X., et al. PersLEARN: Research Training through the Lens of Perspective Cultivation. In *Annual Meeting of the Association for Computational Linguistics*, 2023a.

Shi, Y.-Z., Xu, M., Hopcroft, J. E., He, K., Tenenbaum, J. B., Zhu, S.-C., Wu, Y. N., Han, W., and Zhu, Y. On the complexity of Bayesian generalization. In *International Conference on Machine Learning*, 2023b.

Shi, Y.-Z., Hou, H., Bi, Z., Meng, F., Wei, X., Ruan, L., and Wang, Q. AutoDSL: Automated domain-specific language design for structural representation of procedures with constraints. In *Annual Meeting of the Association for Computational Linguistics*, 2024a.

Shi, Y.-Z., Li, H., Ruan, L., and Qu, H. Constraint representation towards precise data-driven storytelling. In *IEEE Visualization and Visual Analytics Gen4DS Workshop*, 2024b.

Shi, Y.-Z., Meng, F., Hou, H., Bi, Z., Xu, Q., Ruan, L., and Wang, Q. Expert-level protocol translation for self-driving labs. In *Advances in Neural Information Processing Systems*, 2024c.

Shi, Y.-Z., Xu, Q., Meng, F., Ruan, L., and Wang, Q. Abstract Hardware Grounding towards the Automated Design of Automation Systems. In *International Conference on Intelligent Robotics and Applications*, 2024d.

Shi, Y.-Z., Liu, M., Meng, F., Xu, Q., Bi, Z., He, K., Ruan, L., and Wang, Q. Hierarchically Encapsulated Representation for Protocol Design in Self-Driving Labs. In *International Conference on Learning Representations*, 2025.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

Uusitalo, S., Salovaara, A., Jokela, T., and Salmimaa, M. " clay to play with": Generative ai tools in ux and industrial design practice. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*, 2024.

Wilkinson, L. *The grammar of graphics*. Springer, 2012.

Wu, S., Khasahmadi, A., Katz, M., Jayaraman, P. K., Pu, Y., Willis, K., and Liu, B. Cad-llm: Large language model for cad generation. In *NeurIPS 2023 Workshop on Machine Learning for Creativity and Design*, 2023.

Wu, Y. N., Gao, R., Han, T., and Zhu, S.-C. A tale of three probabilistic families: Discriminative, descriptive, and generative models. *Quarterly of Applied Mathematics*, 77(2):423–465, 2019.

Yildirim, I. and Paul, L. From task structures to world models: what do LLMs know? *Trends in Cognitive Sciences*, 2024.

Yuan, Z., Shi, J., and Huang, Y. OpenECAD: An efficient visual language model for editable 3D-CAD design. *Computers & Graphics*, 124:104048, 2024.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

A. Additional remarks

A.1. Rationale for the proposed methodology

Notably, this work is not an alternative to LLM-based CAD generators, but rather an interface to improve the performance of LLM-based CAD generators by bridging designers' language with modeling engineers' language. The fundamental challenge is that while modeling languages are hierarchically documented, designers' language is not standardized in the wild. This necessitates a systematic representation of concepts described in designers' language—from product categories to structures, components, attributes, and operations. This requirement aligns with word learning in cognitive science, where humans learn systems of interrelated concepts rather than isolated terms.

Drawing inspiration from cognitive development, we mirror how people learn concept networks by sampling from the environment and organizing these samples into hierarchical structures through DPMM. This spectral clustering approach captures multi-level attributes rather than clustering based on overall similarity. In our approach, we substitute environmental sampling with LLM-generated samples, as LLMs are recognized repositories of commonsense knowledge. We then apply DPMM to cluster these samples into a hierarchy of structures, components, attributes, and operations. This systematic representation allows natural instructions from designers to be decomposed into fine-grained elements that align with modeling language constructs, enabling targeted control in fast-prototyping.

A.2. Rationale for the selected evaluation metrics

Fast prototyping allowing designers to explore brainstormed ideas without elaborating their instructions into modeling engineers' language. Our approach is explicitly two-stage, with our proposed interface serving as the first stage and LLM-based CAD generators as the second. Our metrics were selected to measure specific aspects of the process. Specifically, we use rendering consistency to measure how well the interface captures designers' intentions, essentially evaluating if desired elements appear and undesired ones are suppressed. Meanwhile, information clarity metrics quantify how effectively information transfers between designers' high-level language and modeling engineers' fine-grained requirements.

A.3. Rationale for the protocol of design study

Fast prototyping differs from full design. While full design requires master models for mass production, fast prototyping serves as an exploration for primary design ideas. Design studies often use two setups: (i) counting iterations to reach certain results; or (ii) evaluating improvements within a fixed iteration count. We adopt the latter to assess the improvements of each iteration with the same instruction, therefore relatively invariant to the number of iterations. The current choice of 10 is informed by discussions with professional industrial designers to capture the typical lifecycle of fast prototyping, and would be further investigated in further study.

Also, we would like to clarify that we aim to directly assess the targetedness of each individual design instructions, *e.g.*, "make the spout narrower". This measure is somewhat subjective, as designers must evaluate whether desired changes were implemented while undesired changes were suppressed. Our work requires step-by-step assessment of each instruction within a sequence leading to a final product, whereas existing datasets typically evaluate only the final result after multiple instructions. This fundamental difference means that while conventional methods can create groundtruth references in advance, our approach relies on designers' on-the-fly instructions, making it impractical to prepare ground truth data (*e.g.*, point clouds) beforehand.

A.4. Rationale for not comparing with more baselines

We would like to first clarify that our work is not proposing an alternative to LLM-based CAD generators, but rather an interface to improve the performance of those generators. Our approach is explicitly two-stage, with our interface serving as the first stage and LLM-based CAD generators as the second. To our knowledge, there are no established state-of-the-art baselines for direct comparison of the entire two-stage pipeline. Comprehensive baselines exist for the second stage, so we have adopted the current state-of-the-art to evaluate the first stage, subject to our proposed interface.

A.5. Relations with program synthesis

Our automated interface design shares conceptual similarities with program synthesis in achieving hierarchical abstraction through iterative sampling and refinement. However, two key distinctions exist. The first lies in the perspective of

knowledge representation. Program synthesis relies on structured knowledge, using exemplar DSL programs to generate higher-level libraries. In contrast, our approach samples from unstructured commonsense knowledge bases (*e.g.*, LLMs) and organizes knowledge into a DSL. The second comes from the perspective of machine learning paradigm. Program synthesis follows a supervised approach, leveraging I/O pairs with task specifications, whereas our method is unsupervised, emerging from commonsense knowledge bases.

B. Ethics statement

The human studies included in this work has been approved by the Institutional Review Board (IRB) of Peking University. We have been committed to upholding the highest ethical standards in conducting this study and ensuring the protection of the rights and welfare of all participants. We paid the participants a wage of $22.5/h, which is significantly higher than the standard wage.

We have obtained informed consent from all participants, including clear and comprehensive information about the purpose of the study, the procedures involved, the risks and benefits, and the right to withdraw at any time without penalty. Participants were also assured of the confidentiality of their information. Any personal data collected (including name, age, and gender) was handled in accordance with applicable laws and regulations.

C. Experimenting details

C.1. Resulting models from fast prototyping

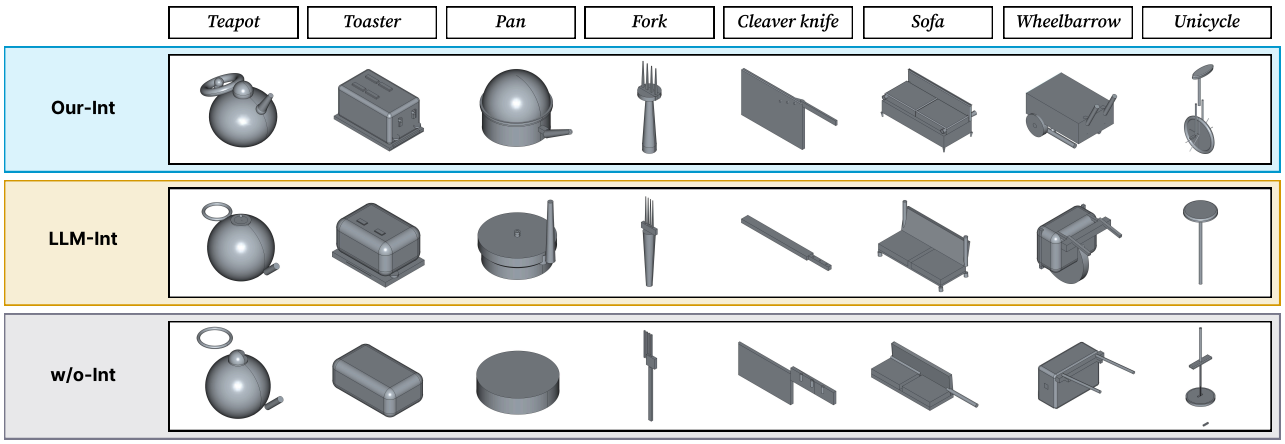

Figure A1. **Visualization of the modeling results from the fast prototyping tasks**

C.2. Showcases of designers' instructions

There is a sample of ten-iteration instructions by one participant in the fast prototyping task of teapot design as follows.

```
1. Make the main body a rounded sphere.
2. Flatten the sphere slightly.
3. Create a short cylindrical neck at the top.
4. Attach a dome-shaped lid to the neck.
5. Extend a spout from the side of the body.
6. Make the spout curved.
7. Make the spout narrower toward the tip.
8. Attach a torus handle to the opposite side of the spout.
9. Align the spout, body, and handle along the same horizontal axis.
10. Keep the handle and the spout symmetrical.
```

There is a sample of ten-iteration instructions by one participant in the fast prototyping task of toaster design as follows.

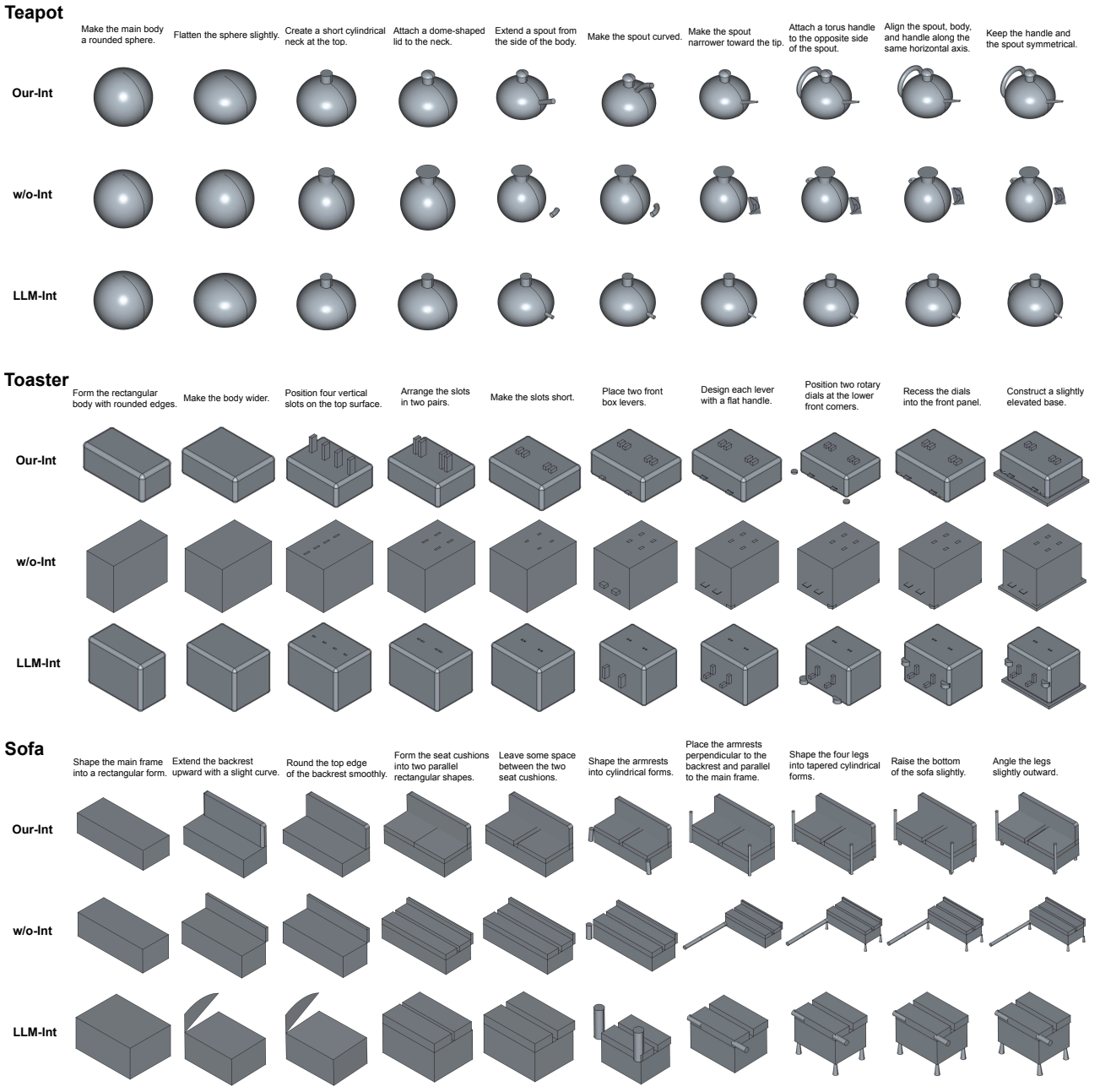

Figure A2. **Step-by-step rendering results of some samples from the fast prototyping tasks**

```
1. Form the rectangular body with rounded edges.
2. Make the body wider.
3. Position four vertical slots on the top surface.
4. Arrange the slots in two pairs.
5. Make the slots short.
6. Place two front box levers.
7. Design each lever with a flat handle.
8. Position two rotary dials at the lower front corners.
9. Recess the dials into the front panel.
10. Construct a slightly elevated base.
```

There is a sample of ten-iteration instructions by one participant in the fast prototyping task of sofa design as follows.

```
1. Shape the main frame into a rectangular form.
2. Extend the backrest upward with a slight curve.
3. Round the top edge of the backrest smoothly.
4. Form the seat cushions into two parallel rectangular shapes.
5. Leave some space between the two seat cushions.
6. Shape the armrests into cylindrical forms.
7. Place the armrests perpendicular to the backrest and parallel to the main frame.
8. Shape the four legs into tapered cylindrical forms.
9. Raise the bottom of the sofa slightly.
10. Angle the legs slightly outward.
```

C.3. Showcases of interface DSL programs

There is a sample DSL program collected from the fast prototyping of teapot design as follows.

```
{
  "Parts": {
    "body": {
      "sphere_0": ["radius"]
    },
    "neck": {
      "cylinder_0": ["height", "radius", "diameter"]
    },
    "lid": {
      "sphere_0": ["height"]
    },
    "knob": {
      "sphere_0": ["height"]
    },
    "spout": {
      "cone_0": ["height", "angle", "radius"],
      "cylinder_1": ["diameter", "radius", "length", "height"]
    },
    "handle": {
      "torus_0": ["length", "height", "radius"]
    }
  },
  "Relationships": {
    "body <-> neck": ["top center"],
    "neck <-> lid": ["flush"],
    "lid <-> knob": ["top center"],
    "body <-> spout": ["side", "aligned_horizontal", "tilted_upward"],
    "body <-> handle": ["side", "opposite_spout", "aligned_horizontal", "higher"]
  }
}
```

There is a sample DSL program collected from the fast prototyping of toaster design as follows.

```
{
  "Parts": {
    "body": {
```

```
      "rounded_box_0": ["height", "length"]
    },
    "slot_1": {
      "rectangle_0": ["width", "length", "height"]
    },
    "slot_2": {
      "rectangle_0": ["width", "length", "height"]
    },
    "slot_3": {
      "rectangle_0": ["width", "length", "height"]
    },
    "slot_4": {
      "rectangle_0": ["width", "length", "height"]
    },
    "border_slot_1": {
      "thin_box_0": ["width", "length", "height"]
    },
    "border_slot_2": {
      "thin_box_0": ["width", "length", "height"]
    },
    "border_slot_3": {
      "thin_box_0": ["width", "length", "height"]
    },
    "border_slot_4": {
      "thin_box_0": ["width", "length", "height"]
    },
    "lever_1": {
      "curved_flat_handle_0": ["height", "radius"]
    },
    "lever_2": {
      "curved_flat_handle_0": ["height", "radius"]
    },
    "guide_rails_lever_1": {
      "vertical_thin_box_0": ["length", "height", "width"]
    },
    "guide_rails_lever_2": {
      "vertical_thin_box_0": ["length", "height", "width"]
    },
    "dial_1": {
      "cylinder_0": ["height", "radius"]
    },
    "dial_2": {
      "cylinder_0": ["height", "radius"]
    },
    "base": {
      "slightly_elevated_box_0": ["length", "width", "height"]
    },
    "crumb_tray": {
      "thin_box_0": ["length", "height", "width"]
    },
    "storage_area": {
```

```
      "cuboid_0": ["height", "radius"]
    },
    "power_cord": {
      "cylinder_0": ["height", "radius"]
    }
  },
  "Relationships": {
    "body <-> slot_1": ["top", "aligned"],
    "body <-> slot_2": ["top", "aligned"],
    "body <-> slot_3": ["top", "aligned"],
    "body <-> slot_4": ["top", "aligned"],
    "slot_1 <-> border_slot_1": ["surround"],
    "slot_2 <-> border_slot_2": ["surround"],
    "slot_3 <-> border_slot_3": ["surround"],
    "slot_4 <-> border_slot_4": ["surround"],
    "lever_1 <-> guide_rails_lever_1": ["beside"],
    "lever_2 <-> guide_rails_lever_2": ["beside"],
    "lever_1 <-> slot_1": ["aligned"],
    "lever_2 <-> slot_3": ["aligned"],
    "body <-> dial_1": ["front_bottom_corner", "recessed"],
    "body <-> dial_2": ["front_bottom_corner", "recessed"],
    "dial_1 <-> increments_dial_1": ["around"],
    "dial_2 <-> increments_dial_2": ["around"],
    "body <-> base": ["beneath", "contrast"],
    "base <-> crumb_tray": ["top", "slide_outward"],
    "body <-> storage_area": ["rear"],
    "storage_area <-> power_cord": ["extend_from"],
    "body <-> top_sides_base": ["seamless"]
  }
}
```

There is a sample DSL program collected from the fast prototyping of wheelbarrow design as follows.

```
{
  "Parts": {
    "basin": {
      "box_0": ["height", "radius"],
      "sphere_1": ["height", "radius"]
    },
    "front_edge": {
      "box_0": ["height", "radius"],
      "cylinder_1": ["height", "radius"]
    },
    "handles": {
      "cylinder_0": ["radius", "height"],
      "cylinder_1": ["radius", "height"]
    },
    "supports": {
      "cylinder_0": ["radius", "height"],
      "cylinder_1": ["radius", "height"]
    },
    "frame": {
```

```
      "u_shape_frame_0": ["radius", "height"]
    },
    "wheels": {
      "cylinder_0": ["radius", "height"],
      "cylinder_1": ["radius", "height"]
    },
    "axle": {
      "cylinder_0": ["radius", "height"]
    }
  },
  "Relationships": {
    "basin <-> front_edge": ["sloped_downward"],
    "basin <-> handles": ["attached_to_rear"],
    "handles <-> supports": ["connected"],
    "frame <-> basin": ["underneath"],
    "frame <-> wheels": ["symmetrical"],
    "wheels <-> axle": ["through"],
    "axle <-> frame": ["horizontal"],
    "axle <-> basin": ["behind_front_edge"],
    "handles <-> frame": ["rear_higher"]
  }
}
```

C.4. Success and failure cases

Success cases demonstrate the advancements of our interface and failure cases shape its boundary.

Success cases: (i) `Our-Int` effectively maintains and guides the transformation of basic shapes and their modifications, mapping abstract designer instructions to concrete shape changes (see Fig. A3A); (ii) `Our-Int` can automatically infer a commonsense spatial distribution of components when designer instructions lack explicit spatial constraints (see Fig. A3B); and (iii) `Our-Int` translates modifications at a finer granularity, identifying which component and which attributes are affected (see Fig. A3C).

Failure cases: (i) for qualitative spatial constraints like "vertical" or "parallel", LLMs sometimes fail to map them correctly to precise positions and orientations due to their weak spatial reasoning (see Fig. A4A); and (ii) for certain abstract and complex instructions—such as operations involving Bezier curves—current methods sometimes fail to capture the correct approach (see Fig. A4B).

D. Implementation details

D.1. Computational cost

We leverage OpenAI's GPT-4o API as the backbone LLM for both domain adaptation and runtime execution of the interface. Automatically designing a domain-specific interface incurs an average cost of approximately $10 per domain, with full adaptation across eight distinct domains. Operational costs during prototyping remain economical: executing the interface for targeted control consumes $0.3 per ten refinement iterations.

D.2. Implementation of MCMC sampling

The MCMC sampling process for construct expansion leverages LLMs as a dynamic proposal generator. For each seed construct $L^{(t)}$, the proposal distribution $q(L' \mid L^{(t)})$ is instantiated through LLM prompts that request domain-aware perturbations, such as: *"Modify [current construct: `handle length`] to explore ergonomic variations, considering material constraints."* The LLM generates candidate perturbations L', which are parsed into valid DSL construct samples. The acceptance probability is estimated using the LLM's likelihood scores for L' versus $L^{(t)}$ under domain-specific con-

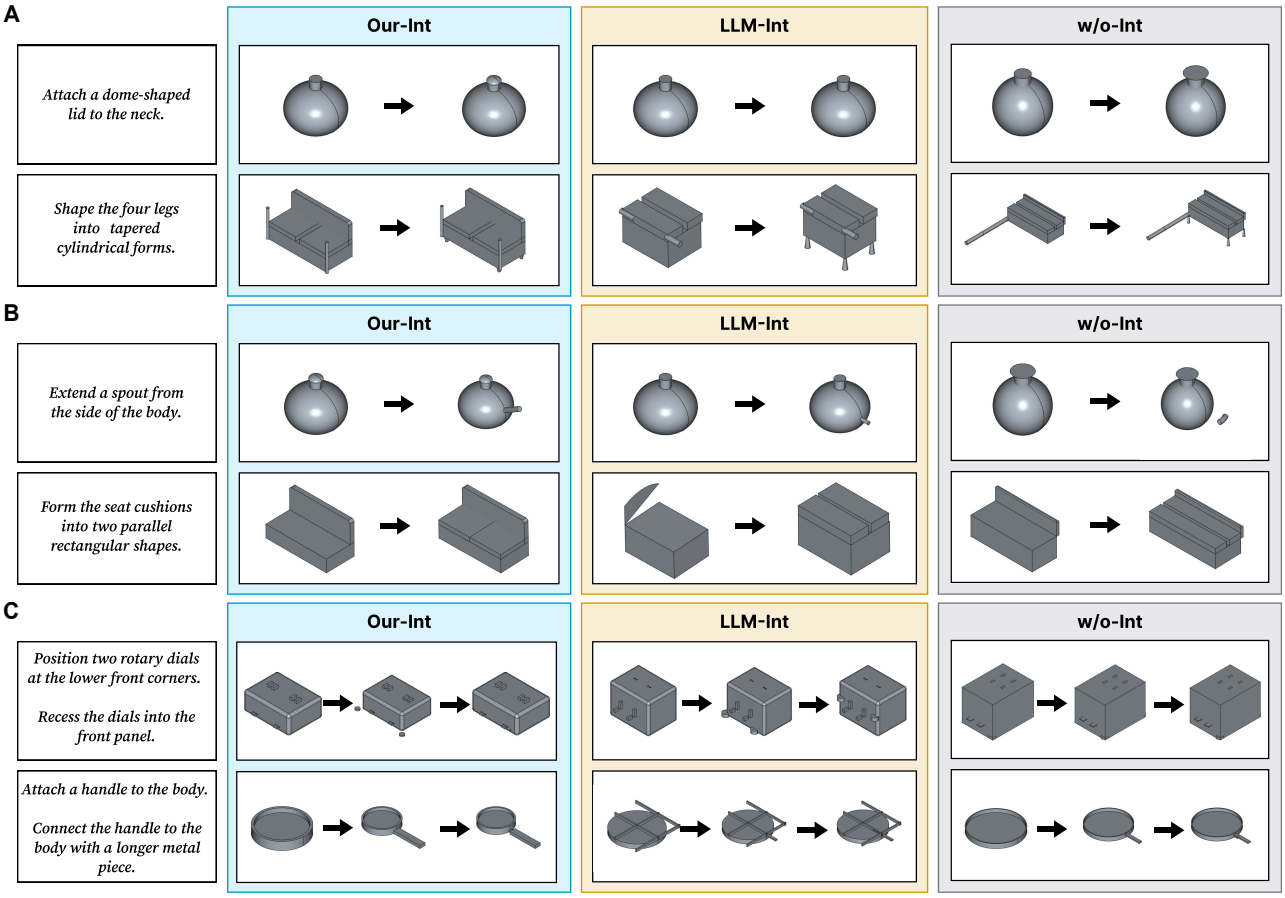

Figure A3. **Demonstrations of success cases**

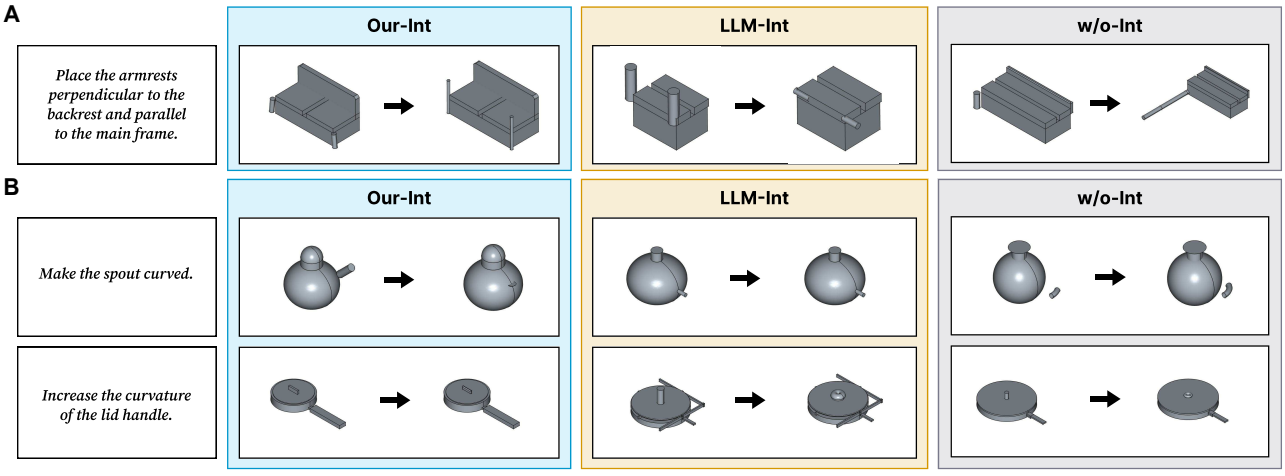

Figure A4. **Demonstrations of failure cases**

texts. To ensure detailed balance, we maintain a buffer of rejected proposals for delayed acceptance. By initializing M parallel chains and performing N steps per chain, the sampling process efficiently explores the combinatorial space of $\mathcal{K}$ while preserving semantic diversity across domains.

D.3. Feasibility validation via documentation-guided RAG

The validation phase employs a Retrival-Augmented Generation (RAG) system built from the modeling engine's programmatic documentation. First, we generate structured documentation by parsing the FreeCAD library's source code (hosted at https://wiki.freecad.org/Category:Developer_Documentation) using Doxygen (as suggested by the FreeCAD official website), extracting function signatures, parameter constraints, and geometric operation specifications. This documentation is chunked into text segments and indexed in a vector store. During validation, each interface construct is converted to a natural language query (*e.g.*, *"Adjust cylindrical surface diameter"*) and used to retrieve the top-k relevant documentation entries. This RAG-driven process ensures the interface remains grounded in the modeling engine's actual capabilities, closing the loop between conceptual abstraction and technical executability.

Specifically, the mechanism of feedback generation evaluates constructs through both LLM-as-a-judge analysis and CAD engine constraints (Zheng et al., 2023). It assesses three feasibility aspects and provides feedback to refine heuristics for subsequent iterations, all with minimal human intervention. First, designer language constructs (*e.g.*, a "ring-shaped teapot body") cannot be translated into modeling operations. If a construct is unsupported, heuristics such as pruning incompatible geometric primitives are applied and the LLM is prompted to propose alternative base shapes (*e.g.*, torus segments, since CAD engines do not directly support a "ring"). A high frequency of such cases indicates that the constructs are overly abstract. Second, modeling constructs (*e.g.*, sofa cushions formed by fusing two cylinders) lack equivalent high-level design terms. For missing constructs, heuristics like generating composite-shape directives (*e.g.*, "create cushion from combined cylinders") are triggered in the next sampling iteration. A high occurrence of these cases suggests insufficient diversity in the designer's language. Third, no overlap between the finest designer constructs and the coarsest modeling operations. For mismatches (*e.g.*, unsupported "material texture" operations), heuristics such as incompatibility pruning are employed, permanently removing non-viable constructs.

D.4. Updating of iteration

The optimization of Eq. (6) is achieved through iterations alternating construct expansion and feasibility validation. During the construct expansion phase, the heuristics adjust exploration strategies based on the interface's state and feedback from prior iterations. When diversity is insufficient (*e.g.*, limited variation in designs), the heuristics broaden exploration breadth by prompting the LLM with directives like *"generate diverse handle configurations for teapots"*. Conversely, if diversity is high, the heuristics narrow exploration breadth by prompting with constraints. When constructs are overly abstract (*e.g.*, *"refine shape"*), heuristics increase exploration depth by decomposing the constructs into atomic operations. If constructs are excessively granular, the heuristics reduce exploration depth by encapsulating low-level commands into functions (*e.g.*, *"smooth contour"*). In the feasibility validation phase, heuristics enforce alignment with the modeling engine's capabilities by pruning constructs incompatible with CAD engine's constraints. This is now discussed in the revised manuscript.

Three intermediate metrics are leveraged to monitor the iterative process: (i) soundness, ensuring all language constructs used by designers can be implemented in the modeling process; (ii) completeness, ensuring all modeling process constructs are represented in designers' language; (iii) granularity alignment, ensuring proper overlap between the finest-grained constructs in designers' language and the coarsest-grained constructs in the modeling process.

Under these intermediate metrics, we explore several alternative interfaces during the iterative process. The interfaces we examined are framed as (i) Single-MCMC: single-scale sampling without updating, sampling and clustering constructs within a single chain but lacks multi-chain diversity; (ii) Multi-MCMC: multi-scale sampling with updating, exploring diverse constructs via parallel chains but omits iterative optimization; and (iii) Ours: extending the previous two alternative interfaces by integrating multi-scale sampling and converged iterative refinement. Quantitative results support the design rationale of our interface (see Fig. A5). This superior systematic representation directly contributes to the improved final performance when integrated with an LLM for CAD.

E. Reproducibility

The project page with supplementary files for reproducing the results of this paper will be available at https://autodsl.org/concept/papers/icml25shi.html.

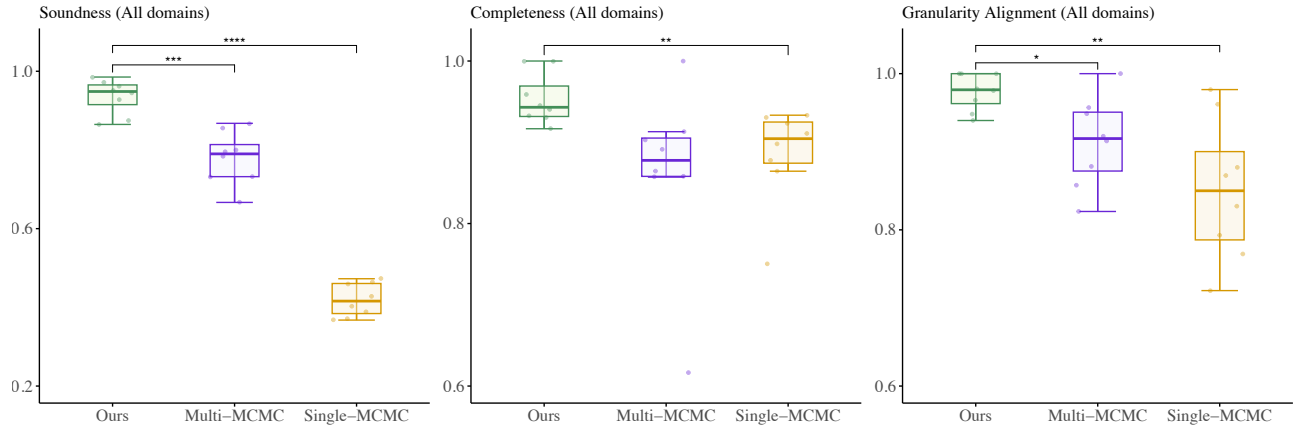

Figure A5. **Quantitative results of the intermediate metrics**

F. Limitations

As a representation of interface designed for a relatively under-researched problem, the design and evaluation of the proposed methodology come with limitations, leading to further investigations:

- We currently model the interaction between industrial designers and the interface as a Markov Process, where each step of adjustment is only conditioned on the last step. Can we model the interactive process in a more sophisticated fashion, thereby taking the temporal dynamics along the process into consideration, such as Bayesian updating, working memory, and the Aha! moment?
- We presently regard NL instructions as the vehicle conveying the intentions of industrial designers. Can we include more types of medium for industrial designers' ideas, such as primal sketches on 2D plane, gesture trajectories in 3D space, or digital clays with surfaces covered by tactile sensors in a higher degree-of-freedom, as the input of our proposed interface?
- Creating groundtruth-level modeling commands for target products requires the dedication of considerable efforts of experts. Can we employ an approach with a higher extent of automation to reduce the cost groundtruth annotation, thereby broadening the scope of the products in the testing set, toward some small group domains of product?

With many questions unanswered, we hope to explore more on building interface for the targeted control of fast prototyping and beyond.