

# Applied AlphaEvolve

Dmitrii Beresnev<sup>2</sup>, Roman Khalikov<sup>5</sup>, Alexander Tolmachev<sup>3,4</sup>, Ivan Ulitin<sup>6</sup>, Ainura Zakirova<sup>2</sup>

**Project curators:** Vladimir Makharev<sup>1,2</sup>, Petr Anokhin<sup>1</sup>

<sup>1</sup> Artificial Intelligence Research Institute, Moscow, Russia

<sup>2</sup> Innopolis University, Innopolis, Russia

<sup>3</sup> Skolkovo Institute of Science and Technology, Moscow, Russia

<sup>4</sup> Moscow Institute of Physics and Technology, Moscow, Russia

<sup>5</sup> Joint-Stock Company Rotec Digital Solutions, Moscow, Russia

<sup>6</sup> Institute of Precision Mechanics and Control Problems of the Russian Academy of Sciences, Saratov, Russia



## Abstract

Recent advances in automated discovery, exemplified by Google’s AlphaEvolve framework, demonstrate the effectiveness of integrating large language models (LLMs) with evolutionary search for complex optimization tasks. AlphaEvolve sets a new standard in this domain through candidate code generation and rigorous evaluation mechanisms. In this study, we run and analyze *OpenEvolve* agent, the open-source implementation of AlphaEvolve. We further extend our analysis to the applied Computer-Aided Design (CAD) reconstruction problem, demonstrating the approach’s versatility and establishing a comprehensive experimental benchmark for this task. Besides, we explore the possibility of the AlphaEvolve appliance to the combinatorial geometry problems. Our source-code is available at GitHub: <https://github.com/crogs-foundation/openevolve-smiles25>.

**Index Terms:** AlphaEvolve, OpenEvolve, LLM, CAD reconstruction, combinatorial geometry

## 1 Introduction

CAD is a cornerstone of modern engineering and manufacturing, enabling the creation and modification of intricate 3D models. Traditionally, this has been a manual, labor-intensive process requiring specialized software and expertise. However, the rise of LLMs and advanced code generation techniques has opened up new possibilities for automating CAD model creation from natural language descriptions. This research explores an innovative approach to this problem by adapting AlphaEvolve, an evolutionary agentic framework, for the CAD Reconstruction task. We use OpenEvolve agent [7], an open-source implementation of AlphaEvolve, to generate and optimize CAD models from textual prompts.

This study focuses on a specific and challenging aspect of CAD: translating abstract, text-based descriptions into precise and manufacturable 3D models. While traditional LLMs have shown some success in this domain through a process known as zero-shot generation (creating a model from a single prompt), they often struggle with complex geometries, precise dimensions, and combinatorial operations. Our approach posits that an evolutionary framework can systematically overcome these limitations by iteratively refining and evolving generated code. By applying the principles of evolutionary computation, the OpenEvolve framework can not only produce high-quality solutions but also explore a wide, diverse range of valid designs, leading to more robust and accurate results than those achieved by a one-off, zero-shot approach. Fur-

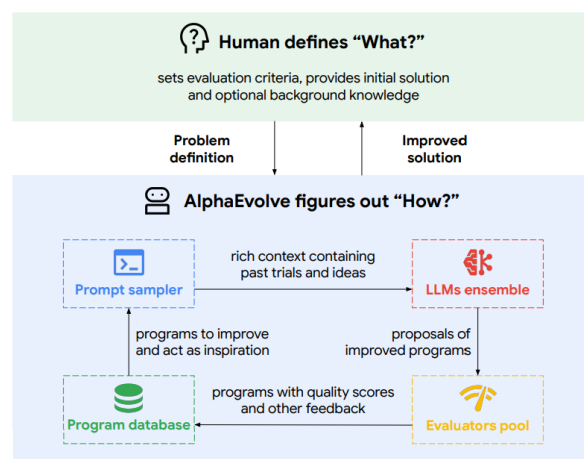


Figure 1. Overview of AlphaEvolve framework

thermore, the framework’s adaptability is demonstrated by its application to combinatorial geometry problems, a class of tasks that involve finding optimal configurations from a finite set of geometric objects, such as set partitioning or tiling.

### 1.1 Hypothesis

This work explores the hypothesis that the AlphaEvolve agent [6], implemented within the OpenEvolve framework [7], can be adapted to the CAD Reconstruction task and other applied domains, achieving robust performance that surpasses the zero-shot capabilities of state-of-the-art LLMs or task-specific baselines. Building upon prior research on AlphaEvolve applications, this study focuses on executing and analyzing OpenEvolve—an open-source implementation of AlphaEvolve, with the goal of contributing to the CAD Reconstruction task and improving the framework’s core components where possible. The CAD experiments involve benchmarking on standard datasets (e.g., Text2CAD, DeepCAD) using metrics such as Chamfer Distance, Intersection over Union, Invalidity Ratio, and custom evaluation criteria. Additionally, the applicability of the AlphaEvolve framework to combinatorial geometry problems is explored.

## 2 Background & Literature Review

The field of AI-driven design and generation has advanced significantly, especially with large language models (LLMs) and evolutionary algorithms. A key development is AlphaEvolve, an evo-

lutionary coding agent that uses LLMs for general-purpose algorithm discovery and optimization [6]. AlphaEvolve employs multiple LLMs to generate diverse code solutions, which are then refined through an evolutionary process. This method has succeeded in complex tasks like data center scheduling, AI model training, and hardware design, showing its ability to evolve entire codebases rather than just single functions.

This approach fits into the broader framework of multi-agent systems (MAS), where autonomous agents work together to solve problems [2]. Another related concept is the Darwin Gödel Machine (DGM), a self-improving system that updates its own code, representing progress in open-ended evolution [12]. However, DGM has a complex structure and lacks open-source implementations, while AlphaEvolve offers accessible tools (e.g., OpenEvolve<sup>1</sup>, OpenAlpha\_Evolve<sup>2</sup>).

The CAD Reconstruction problem, converting inputs like text or point cloud into CAD model, is an important challenge in engineering. What is more crucial in industry, is to recover a modeling script from an existing mesh file of a 3D object, enabling further refinement, parametric control, or iterative modification of the model. This also called CAD reverse engineering problem, which goal is to reconstruct of boundary representation (B-Rep). Standard datasets like DeepCAD and Text2CAD have supported progress by providing construction sequences and text descriptions [11], [3]. A recent breakthrough is CADrille, a multi-modal model that fine-tunes an LLM using reinforcement learning, achieving top results by optimizing both geometric accuracy and code validity [5]. However, CADrille depends on a single LLM’s knowledge, which may limit robustness and creativity. Evolutionary methods like AlphaEvolve could help explore a wider solution space, leading to more diverse and validated outcomes.

Evolution-based approaches for large language models (LLMs), including AlphaEvolve and its open-source implementation OpenEvolve, provide powerful tools for discovering structures or objects with desirable properties when their “quality” can be quantified through metrics that depend monotonically on their underlying characteristics. This makes LLM-driven evolutionary methods particularly promising for problems in combinatorial geometry, which studies geometric objects with combinatorial constraints—such as maximizing edge/vertex counts in graphs, finding optimal partitions or coverings, and analyzing graph configurations. The key idea involves efficiently verifying constrained objects through computable metrics (e.g., set diameter lengths, graph invariants, etc.). In this work, we explore the application of AlphaEvolve to the problem of partitioning sets into subsets of smaller diameter. This task is deeply connected to fundamental questions in combinatorial geometry, including the *Borsuk conjecture* [1] and the *Hadwiger–Nelson problem* [8].

Specifically, we consider partitioning the  $n$ -dimensional unit-diameter ball (i.e., a ball of radius  $r = 1/2$ ) into  $(n + 1)$  subsets while minimizing the maximum diameter among them. Theoretically, optimal bounds for such partitions can be derived from a regular simplex inscribed in the boundary of the unit ball. Our experiments evaluate the effectiveness of evolutionary methods for this partition task and compare computational results against known theoretical values. Although solving this problem does not produce new bounds for combinatorial geometry, it serves as an

important benchmark for evolutionary approaches in this domain and lays the groundwork for future research.

### 3 Methods

#### 3.1 OpenEvolve agent

The AlphaEvolve is an evolutionary agentic framework to generate a state-of-the-art solutions for a user-defined task, as depicted in Figure 1. Basically it uses two agents to generate a code and to perform an evaluation and feedback generation. Original implementation employed Gemini 2.0 Flash and 2.0 Pro versions respectively. OpenEvolve is an open-source implementation of the AlphaEvolve. It also proposes additional improvements to the AlphaEvolve, for example, it allows to employ any number of open-source models as evaluators using local inference and/or APIs.

**3.1.1 Key components** To initiate the evolutionary process using the OpenEvolve framework, three key components must be defined: the initial program, the evaluator, and the configuration. The initial program typically consists of a text file containing a code snippet written in any supported programming language that serves as the starting point for evolutionary improvement via OpenEvolve. The evaluator is a Python script that assesses the quality of candidate programs using user-defined evaluation metrics. These metrics are designed such that higher values correspond to better performance, guiding the evolutionary algorithm toward more optimal solutions.

The configuration file plays a central role by specifying critical parameters for the evolutionary task. It includes the hyperparameters of the evolutionary algorithm, settings for the agents for code generation, the number of evolutionary iterations, and a system prompt that provides high-level instructions for the intended CAD model generation. This modular design allows OpenEvolve to be adapted to a wide range of code synthesis and optimization tasks, including the refinement and evolution of CAD modeling scripts.

**3.1.2 Evolution process** OpenEvolve leverages the MAP-Elites algorithm as its core evolutionary engine to achieve a balance between performance optimization and behavioral diversity. MAP-Elites is a quality-diversity algorithm that maintains a structured archive of elite solutions across a discretized feature space, allowing the system to simultaneously explore a broad range of solution types while refining high-performing individuals. In OpenEvolve, the feature space is defined by interpretable dimensions such as program complexity and functional diversity, though it can be extended to task-specific metrics. This structured search encourages the discovery of not only top-performing CAD programs but also diverse and creative alternatives. The framework supports island-based evolution, where multiple sub-populations (or islands) evolve in parallel with periodic migration, promoting cross-fertilization of ideas and preventing premature convergence.

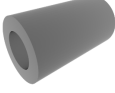




#### 3.2 Inference Resources

For our local inference experiments, we used a mix of specialized hardware and cloud services. Our main resources provided by school organizers included Huawei’s ModelArts with Ascend TPUs and a Yandex DataSphere GPU service. We also used the

<sup>1</sup> <https://github.com/codelion/openevolve>

<sup>2</sup> [https://github.com/shyamsaktawat/OpenAlpha\\_Evolve](https://github.com/shyamsaktawat/OpenAlpha_Evolve)

**Table 1**  
Shapes used in the current research

Figure name	Picture	Text description
Tube		A tall vertical cylinder (116mm diameter, 200mm height) is partially subtracted by a smaller offset cylinder (66mm diameter, 200mm height) in the center of the cylinder.
Gear		Gear wheel: inner radius of 10 mm, outer radius of 40 mm, thickness of 20 mm, with 6 rectangular cogs. The cogs are as thick as the gear, protruding 10 mm outwards and 20 mm wide. The cogs are inserted 2 mm into the gear body.
Open Box		A box with a length of 150mm, a width of 100mm, and a bottom thickness of 10mm. The walls are 40mm high. The walls along the length have a thickness of 15mm, and the walls along the width have a thickness of 30mm.
Ladder		The resulting object is a solid, monolithic block measuring 60 mm wide, 80 mm high, and 50 mm deep. It has a two-step profile on its front face composed of 40 mm risers and 30 mm treads. The block is bisected by a full-width planar cut on its right side that connects the top-back edge to the bottom-front edge, creating a new face angled at approximately 58 degrees relative to the base.
Spheres		A large sphere with a radius of 100mm is modified by subtracting a smaller sphere (radius 80mm) shifted 20mm up and 70mm left from the center. Another small sphere (radius 50mm) touches the bottom of the large sphere from the outside.

OpenRouter<sup>3</sup>, GPT4Free<sup>4</sup>, and Mistral AI<sup>5</sup> for quicker tests. For a complete list of all the hardware and services, see Appendix 6.1.

### 3.3 CAD Reconstruction

**3.3.1 Dataset** For our 3D CAD reconstruction task, we used the Text2CAD dataset [4], which consists of pairs of natural language descriptions and corresponding Python code that uses the CadQuery library to generate precise 3D models. Each sample includes a detailed textual explanation of the object’s dimensions, shape, and geometric operations, along with the Python code required to model it.

Along with each description, the corresponding script provided in the dataset serves as the ground truth and is used for evaluation. We compare the 3D model produced by the LLM-generated script against the ground truth model by rendering both and computing similarity metrics, presented in the Section 3.3.2.

For our test set, we selected five shapes — tube, gear, open box, ladder, and spheres (see Table 1). These shapes were intentionally chosen to represent varying levels of complexity: the tube is relatively simple, while the ladder and spheres more detailed and complex geometric descriptions. This progression allowed us to evaluate how well OpenEvolve performs across different difficulty levels.

Although the original descriptions in the dataset are highly detailed, we have decided to modify them to better suit the needs of our specific task. Rather than using full text description, we abstracted them to include only the key geometric information—such

as the overall shape, dimensions, and main features. Despite the simplification, the modified descriptions still provide all the essential details needed for the model to correctly infer the modeling procedure. The main motivation for description simplification is to make the OpenEvolve approach practical for real-world use. In reality, users are unlikely to provide highly detailed, fine-grained descriptions, even though such detail would make the task easier for the language model. Table 4 shows original description for the 10-sided polygon prism and its simplified version.

**3.3.2 Scoring and Metrics** To quantitatively evaluate the similarity between a predicted mesh ( $M_{pred}$ ) and its corresponding ground-truth mesh ( $M_{gt}$ ), we compute a comprehensive set of geometric and structural metrics. To ensure scale and position invariance, all meshes are first normalized: they are centered at the origin and uniformly scaled to fit within a unit cube. This normalization is crucial as it provides a canonical space for comparison and ensures our distance-based metrics are well-behaved. The metrics are organized into three distinct categories:

**Volumetric:** These metrics assess the overlap in 3D space.

- *Intersection over Union (IoU)* is a standard metric that measures the ratio of the intersection volume to the union volume of the two meshes. It provides a highly accurate measure of volumetric overlap but can be computationally expensive and sensitive to non-watertight meshes
- *Voxelized Intersection over Union (VIoU)* serves as a robust and efficient approximation of IoU. To compute it, we voxelize both meshes into a  $64 \times 64 \times 64$  grid and calculate the IoU on the resulting binary voxel representations. This approach is more resilient to topological errors in the input meshes and

<sup>3</sup> <https://openrouter.ai/>

<sup>4</sup> <https://github.com/xtekky/gpt4free>

<sup>5</sup> <https://mistral.ai/>

**Table 2**

Evaluation metrics with corresponding short names (aliases) and formulae.  $M_{pred}$  and  $M_{gt}$  are the predicted and ground-truth meshes.  $P_{pred}$  and  $P_{gt}$  are point clouds sampled from their respective surfaces.

Metric	Alias	Category	Formula
Intersection over Union	iou	Volumetric	$\frac{\text{vol}(M_{pred} \cap M_{gt})}{\text{vol}(M_{pred} \cup M_{gt})}$
Voxelized IoU	viou	Volumetric	$\frac{ V_{pred} \cap V_{gt} }{ V_{pred} \cup V_{gt} }$ , where $V$ is the set of occupied voxels.
Chamfer Distance (Similarity)	cd	Point-Based	$1 - \frac{1}{2} \left( \frac{1}{ P_{pred} } \sum_{x \in P_{pred}} \min_{y \in P_{gt}} \ x - y\ _2^2 + \frac{1}{ P_{gt} } \sum_{y \in P_{gt}} \min_{x \in P_{pred}} \ y - x\ _2^2 \right)$
Hausdorff Distance (Similarity)	hd	Point-Based	$1 - \max \left( \sup_{x \in P_{pred}} \inf_{y \in P_{gt}} \ x - y\ _2, \sup_{y \in P_{gt}} \inf_{x \in P_{pred}} \ y - x\ _2 \right)$
Wasserstein Distance (Similarity)	wd	Point-Based	$1 - \text{EMD}^2(P_{pred}, P_{gt})$ , computed with a squared Euclidean cost matrix.
Area Similarity	as	Property-Based	$1 - \frac{ \text{Area}(M_{pred}) - \text{Area}(M_{gt}) }{\text{Area}(M_{pred}) + \text{Area}(M_{gt})}$
Inertia Similarity	is	Property-Based	$1 - \frac{\ I_{pred} - I_{gt}\ _F}{\ I_{pred}\ _F + \ I_{gt}\ _F}$ , where $I$ is the inertia tensor and $\ \cdot\ _F$ is the Frobenius norm.

significantly faster

**Point-Based:** These metrics evaluate similarity by comparing point clouds sampled from the mesh surfaces. For each metric, a distance  $d(P_{pred}, P_{gt})$  is first computed between the point cloud  $P_{pred}$  sampled from  $M_{pred}$  and  $P_{gt}$  from  $M_{gt}$ . Since all meshes are normalized to a unit cube, the distances are bounded. We convert this distance  $d$  into a similarity score  $s \in [0, 1]$  via the transformation  $s = 1 - d$ , where a score of 1 indicates perfect similarity.

- *Chamfer Distance (CD)* measures the average squared distance from each point in one cloud to its nearest neighbor in the other. We sample  $N = 5000$  points from each mesh for this calculation. It provides a good overall measure of surface alignment.
- *Hausdorff Distance (HD)* measures the maximum distance between the two point clouds, effectively identifying the worst-case mismatch. This makes it sensitive to outliers and significant geometric deviations. It is computed on the same  $N = 5000$  point samples.
- *Wasserstein Distance (WD)*, also known as the Earth Mover’s Distance (EMD), quantifies the minimum cost required to transform one point distribution into the other. We compute the squared EMD using a squared Euclidean cost matrix between point clouds of size  $N = 1000$ . This metric is particularly effective at capturing structural differences and is less sensitive to small misalignments than CD.

**Property-Based:** These metrics compare intrinsic physical or geometric properties of the meshes.

- *Area Similarity (AS)* compares the total surface area of the two meshes, calculated as a normalized similarity score based on their relative difference.
- *Inertia Similarity (IS)* compares the 3D shapes’ mass distribution by calculating the similarity between their moment of inertia tensors. The similarity is defined as one minus the normalized Frobenius norm of the difference between the tensors, providing a descriptor of rotational symmetry and mass distribution.

Table 2 provides a summary of all metrics, their short names, and their mathematical formulations. *Combined score* — overall score — is computed as sum of the seven described metrics. As

all the described metrics are in  $[0, 1]$  range, the combine score 7.0 corresponds to a perfect sahpes match, while 0.0 — to total mismatch.

**3.3.3 Zero-shot baseline** To validate the applicability of OpenEvolve to CAD reconstruction tasks, we conducted a series of preliminary experiments using canonical test cases. The designed 3D model descriptions was inputed to the selected open-source models Qwen2.5-72b, Qwen2.5-Coder-32b, and closed-source GPT-o4-mini.

**3.3.4 OpenEvolve Implementation** Based on provided example runs, we implemented the core components of our reconstruction framework, adapting them to generate parametric CAD code. As an initial benchmark, we evaluated the reconstruction of a simple geometric shape—box with a hole (Figure 7, left).

The evolutionary search was configured with 100 iterations, a choice made to strike a balance between available time and achieving meaningful performance results. The remaining parameters, including a population size of 50 and 3 parallel islands, were set by default as per the MAP-Elites algorithm. Consistent with the original OpenEvolve setup, two large language models were employed for code generation: mistral-small-latest (assigned to 80% of the population) and mistral-large-latest (20%). This mixture of models enabled a balance between fast exploratory sampling and higher-quality code generation, contributing to population diversity and search efficiency.

To run OpenEvolve, we constructed an evaluator prompt and an initial program to evolve, which are presented in the Appendix 6.2 and Appendix 6.3 respectively. The evaluator prompt was carefully designed to provide a comprehensive and unambiguous evaluation framework by containing a role, requirements, metrics description, and task description. This structure guides the evaluator toward a consistent assessment, while the initial program, a very simple CadQuery shape, provides a foundational yet flexible starting point for the evolutionary process to explore a broad range of design solutions.

### 3.4 Sets partitions in combinatorial geometry

Evolution-based approaches (AlphaEvolve [6], OpenEvolve et.al) have a broad area of applications not only for practical problems (such as CAD reconstruction), but also for scientific problems. In this paper we explore the appliance of OpenEvolve approach to combinatorial geometry problems that potentially promised for such evolutions and haven't been considered before, for instance as one of benchmarks in the paper [6]. One of the central problems in combinatorial geometry related to the Borsuk hypothesis stated by K. Borsuk in 1933:

**Borsuk hypothesis:** *can be any set with unit diameter  $A \subset \mathbb{R}^n$  be divided into  $n + 1$  subsets of smaller diameter (less than 1) ?*

This hypothesis has attracted a lot of attention from researchers over the world, and was conjectured in .... Nowadays, this hypothesis was conjectured started from  $n = 64$ , i.e. for any  $n_0 \geq n$  exists such set in  $\mathbb{R}^{n_0}$  that *cant* be divided into  $(n_0 + 1)$  parts of smaller diameter. Saying about smaller dimensions ( $n < 63$ ), the following definition is especially sufficient:

**Definition:** Let  $b(n)$  be the minimum number of parts such that any bounded subset  $A \subset \mathbb{R}^n$  can be divided into  $b(n)$  parts of smaller diameter. The quantity  $b(n)$  is called a *Borsuk number*.

Without loss of generality, we can consider only subsets  $A$  with unit diameter ( $\text{diam}(A) = 1$ ) and its partitions into subsets of smaller diameter. For small dimensions, precise values are known:  $b(1) = 2$ ,  $b(2) = 3$  and  $b(3) = 4$ , however if  $n \geq 4$  the Borsuk number can be greater than  $n + 1$  that makes this problem significantly more complex. The next definition clarifies the optimization problem for partition-like tasks:

**Problem:** Let  $d_{n,k} \in \mathbb{R}_+$  is the smallest number such that every subset  $A \subset \mathbb{R}^n$  of unit diameter can be divided into  $k$  parts  $A = A_1 \sqcup A_2 \sqcup \dots \sqcup A_k$  such that  $\forall i \in \{1, \dots, k\} \quad \text{diam}(A_i) \leq d_{n,k}$ .

Currently, exact values of  $d_{n,k}$  have been obtain only for some special cases and small dimensions. One of the way to find upper bounds for these quantities relies on the Young theorem:

**Theorem (H. Young, 1901):** Every set  $A \in \mathbb{R}^n$  with unit diameter can be covered by the ball  $B \subset \mathbb{R}^n$  with radius  $r = \sqrt{\frac{n}{2n+2}}$ .

**Definition:** The set  $U \subset \mathbb{R}^n$  called a universal covering set if and only if an every subset of  $\mathbb{R}^n$  with unit diameter can be covered by  $U$ .

Hence, if we divided some universal covering set into  $n$  parts with diameters of all parts not more than  $d_0$  in  $k$ -dimensional case, we prove that  $d_{n,k} \leq d_0$ . This fact allows to obtain upper bounds of  $d_{n,k}$  values via the partitioning of the universal covering sets into parts of smaller diameter [9, 10].

Following the Young theorem, the ball with radius  $r = \sqrt{\frac{n}{2n+2}}$  is the universal covering set in  $n$ -dimensional case. From Young theorem follows:

**Corollary:** The side length of a regular  $n$ -dimensional simplex inscribed in the unit ball  $B_n$  equals  $\rho_d = \sqrt{\frac{n+1}{2n}}$ .

Hereandafter, the unit ball denotes the ball with unit diameter, i.e. with radius  $r = 1/2$ .

**3.4.1 Ball partitions into parts of smaller diameter.** Despite the existence of more complex universal covering systems, in our initial experiment we firstly will explore the partitions of the unit ball  $B_n \subset \mathbb{R}^n$  centered in  $\vec{0}$  into  $(n + 1)$  parts of smaller diameter. Each partition has been parametrized via the  $(n + 1)$  vectors  $\vec{v}_1, \dots, \vec{v}_{n+1}$  and the sphere partition is represented via polyhedral

cones centered in  $\vec{0}$  based on vectors  $\vec{v}_1, \dots, \vec{v}_{n+1}$ . During evolution process the program computed the diameter of each part and find that maximum value among all parts. According to latter corollary, the theoretical estimation of the maximum diameter (in case of partition into  $n$  parts) equals  $\rho_d = \sqrt{\frac{n+1}{2n}}$ , because one of  $n$  parts contains at least two simplex vertices according to Dirichlet principle and each two regular simplex vertices distanced at  $\rho_d$  (see Figure 2).

This task has been chosen for this analysis, because it's quite simple to find the precise theoretical value of the estimated quantity ( $\rho_d$ ), but at the same time it's quite complex to find optimal directions in  $n$ -dimensional case (directions corresponding to regular  $n$ -dimensional simplex) via analytic optimizers in high-dimensional setup. It's expected that evolution-based approaches find the optimal directions that result in the optimal value that will be very closed to theoretical estimate.

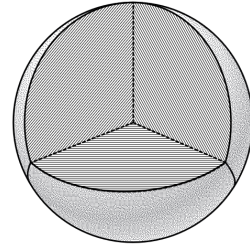


Figure 2. Regular simplex based partition of unit ball in  $\mathbb{R}^3$  into 4 parts

## 4 Results

### 4.1 Ascend experiments

Currently, we have prepared the virtual environment and tested some basic code. To upload a local model to the environment, it is required to download a model from the Hugging Face portal. We faced a problem with connection stability. Right now, we've solved this problem and are preparing to upload a Qwen3-32B model to Huawei's OBS.

We was not able to run quantized Qwen3-32B models, AWQ quantization requires libraries which only works on NVIDIA or AMD GPU (IPEX backend or triton library). Unquantized models requires big amount of memory and it was hard to download, archive and upload, besides that we was able to employ a variety of models including Qwen3-32B through API.

### 4.2 CAD reconstruction

**4.2.1 Zero-shot task** As an initial approach to the CAD reconstruction problem, we explored the zero-shot capabilities of LLMs, including the instruction-following Qwen2.5-72b, the reasoning model Qwen2.5-Coder-32b, and GPT-o4-mini. The goal of this experiment was to assess in which cases OpenEvolve is necessary and to explore whether zero-shot methods alone could provide a correct solution.

Our initial results showed that vanilla shapes, like a basic box or sphere, can be easily reproduced by zero-shot LLMs. Therefore, we focused on the more complex figures (see Table 1) that zero-shot methods struggled to reconstruct accurately.

The test results for the zero-shot approach are summarized in

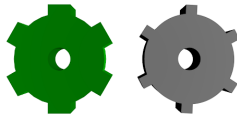
**Table 3**  
Combined Scores for Different LLMs and Shapes<sup>1</sup> (values reported as *mean ± std*)

LLMs	Type	Combined Score				
		Tube	Gear	Open box	Ladder	Spheres
Qwen2.5-72b	Zero-Shot	4.950 ± 0.220	3.988 ± 1.402	4.138 ± 0.088	2.994 ± 1.579	3.128 ± 0.058
Qwen2.5-Coder-32b	Zero-Shot	2.749 ± 2.905	0	3.802 ± 0.527	3.802 ± 0.122	2.830 ± 0.002
GPT-o4-mini	Zero-Shot	5.651 ± 0.306	3.460 ± 1.840	3.777 ± 0.763	2.991 ± 1.623	2.435 ± 1.809
Qwen2.5-72b (t=0.7) 80%	OpenEvolve	<b>6.728 ± 0.205</b>	<b>4.779 ± 0.982</b>	5.199 ± 0.190	<b>3.637 ± 1.035</b>	<b>6.528 ± 0.176</b>
Qwen2.5-72b (t=0.2) 20%						
Qwen2.5-Coder-32b 50%	OpenEvolve	6.682 ± 0.238	4.529 ± 1.021	<b>5.426 ± 0.214</b>	3.417 ± 1.118	5.227 ± 0.221
Qwen2.5-72b 50%						

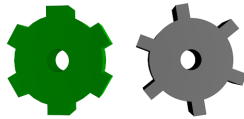
<sup>1</sup> The shapes are described in Table 1.

Some evolutionary experiment visual results are presented in Appendix 6.7.

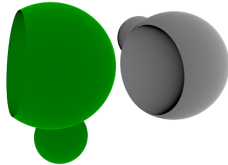
Table 3. Qwen2.5-Coder-32b failed to generate runnable code for some figures, which led to errors and a zero score in the metrics. In contrast, GPT-o4-mini successfully generated runnable code across all tasks. Its performance, however, was uneven: it achieved relatively strong results on the Tube shape ( $5.651 \pm 0.306$ ), outperforming Qwen2.5-72b ( $4.950 \pm 0.220$ ), but performed worse on the Gear and Spheres shapes ( $3.460 \pm 1.840$  and  $2.435 \pm 1.809$ , respectively), where Qwen2.5-72b achieved higher scores ( $3.988 \pm 1.402$  and  $3.128 \pm 0.058$ ). On the Open box and Ladder shapes, both models showed comparable performance, with GPT-o4-mini slightly lagging behind in stability due to higher variance. These results suggest that while GPT-o4-mini demonstrates a capacity to generate runnable code consistently, its effectiveness is strongly shape-dependent and does not uniformly surpass Qwen2.5-72b in zero-shot performance.



**Figure 3.** The Qwen2.5-72b model zero-shot results for the gear.



**Figure 4.** The GPT-o4-mini model zero-shot results for the gear.



**Figure 5.** The GPT-o4-mini model zero-shot results for the spheres.

In terms of visual similarity, Figure 3 shows the gear generated by Qwen2.5-72b, while Figure 4 presents the gear generated by

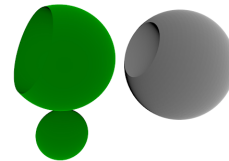
GPT-o4-mini. The gear from the Qwen2.5-72b model appears visually closer to the ground truth, that is also confirmed by the evaluation metrics in the Table 3. On the other hand, for the spheres figure, the solution generated by GPT-o4-mini (Figure 5) looks more accurate compared to the Qwen2.5-72b model results (Figure 6). This is demonstrated by the combined score, which is higher for GPT-o4-mini.

Overall, these zero-shot results highlight areas for improvement, which we address using the OpenEvolve method, as described in the next Section 4.2.2.

**4.2.2 OpenEvolve** We prepared the pipeline for CAD Reconstruction task for the OpenEvolve as described in 3.3.4 and run initial benchmark. After five iterations of the algorithm, an ideal-match 3D model with the maximum value of metrics was obtained (Fig. 7, right). Fig. 8 shows the visualized program evolution provided by OpenEvolve.

The OpenEvolve method significantly improved the performance of the models on the CAD reconstruction task. As shown in Table 3, OpenEvolve consistently produced higher combined scores for most complex shapes compared to the zero-shot approach. Specifically, the combined Qwen2.5-72b model configuration achieved the best scores for the Tube, Gear, Ladder, and Spheres figures, outperforming all other models, including GPT-o4-mini. The combined score for the Spheres figure saw a particularly notable improvement of over 0.7 from the zero-shot result. Additionally, OpenEvolve successfully addressed the failures of the Qwen2.5-72b and Qwen2.5-Coder-32b models, which failed to generate runnable code for some shapes in the zero-shot tests.

The combination of Qwen2.5-Coder-32b and Qwen2.5-72b also yielded strong results, achieving the highest combined score for the Open box figure. This suggests that mixing different models within the evolutionary loop can be an effective strategy, leveraging the strengths of each model to produce better outcomes. Over-



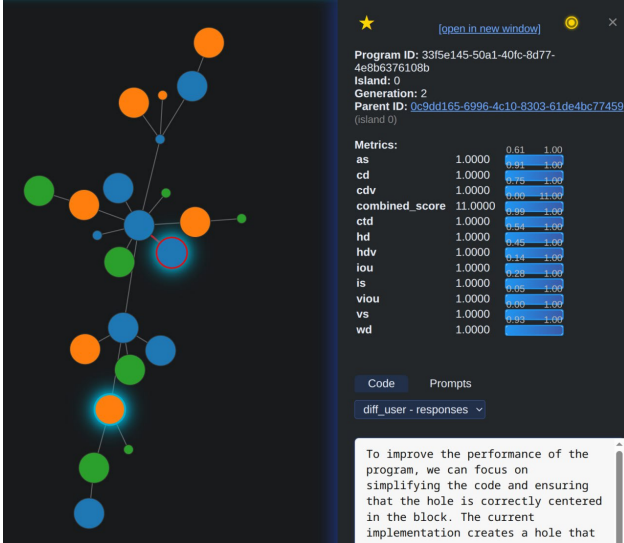
**Figure 6.** The Qwen2.5-72b model zero-shot results for the spheres.



all, these results confirm that the iterative, evolutionary process of OpenEvolve is crucial for accurately reconstructing complex CAD models, moving beyond the limitations of single-pass, zero-shot methods. Visual results of some of these experiments are provided in Appendix 6.7.



**Figure 7.** Example of built CAD meshes. Green model to the left - the Ground Truth for the experiment, gray model in the middle - the CAD mesh proposed by OpenEvolve on iteration 4, red model in the right - the final perfect match CAD mesh proposed by OpenEvolve on iteration 5.



**Figure 8.** Program evolution graph for specific checkpoint during experiment on the initial benchmark.

#### 4.2.3 Analysis of Error Modes and Evolutionary Pathway

To understand the significant performance gap between zero-shot generation and the OpenEvolve framework, we must analyze the types of errors that occur in programmatic CAD modeling. These errors can be broadly categorized into two types:

- **Structural Errors:** These are fundamental, topological mistakes where the core components or operations of the design are incorrect. Examples include failing to create a hole, generating a solid block instead of an open box, or omitting a key feature like the teeth on a gear. These errors represent a misunderstanding of the object's basic structure and cannot be fixed by simply adjusting numbers.
- **Parametric Errors:** These occur when the model's overall structure is correct, but its dimensions, positions, or other numerical attributes are inaccurate. Examples include a gear with the correct number of teeth but the wrong radius, or a hole that is correctly placed but has an incorrect diameter.

These errors can be fixed by tuning the parameters within an already correct program structure.

Our analysis reveals that zero-shot LLMs frequently commit parametric errors in case of easily describable models (e.g. Figures 3, 4 and 5). As seen in Figure 6, the zero-shot attempt by Qwen2.5-72b on the 'Spheres' task resulted in a both parametric and structural failures. The model lacks distinct, correctly formed additional sphere and a properly parametrized subtraction sphere. This failure mode is common in single-pass generation, where a single logical misstep in the program's control flow leads to a completely incorrect topology. It appears more frequently as the complexity of the 3D models increases. Mostly it is caused by the wrong translations of the model's part in relation to other parts. In some cases it is even impossible to construct the model. The Qwen2.5-72b failed to construct the 'Ladder' model.

The OpenEvolve framework excels precisely because its evolutionary process creates a pathway to solve both types of errors in sequence. As illustrated in Figure 9, the process unfolds in two distinct phases:

1. **Structural Discovery (Early Generations):** In the initial generations, the population of programs is highly diverse. Most attempts contain structural errors, like shown in Figure 9a. However, through mutation and selection guided by the evaluation metrics, the system tries to discover correct structural elements like in Figure 9b. Figure 9c shows a critical "structural leap" where a mutation successfully introduces the concept of cogs. Even though these cogs are parametrically incorrect (wrong position), their existence provides a massive improvement in the evaluation score. The evolutionary search prioritizes this correct topology, ensuring it becomes the foundation for future generations.
2. **Parametric Refinement (Later Generations):** Once a structurally sound program template is established within the population, the evolutionary search shifts its focus to parametric optimization. The LLM, now prompted to make small modifications to a high-quality parent program, generates variations that adjust dimensions. The program evolves to have the correct position and shape of cogs. Guided by the continuous feedback from the scoring function, most of the later generations tries to fine-tune these values. It took most of iterations to properly fix the position. The solution converges on the ground truth, as shown in Figure 9d.

In summary, evolution provides a robust, two-stage search mechanism that single-pass generation lacks. It first navigates the vast and difficult space of program structures to find a viable topology and then efficiently optimizes the parameters within that structure. The feedback provided by the evaluator agent and random mutations are main reasons for better performance on complex design tasks in comparison to the simple one-shot approach.

#### 4.3 Ball partitioning into parts of smaller diameter

Let  $B_n$  be the  $n$ -dimensional ball centered at  $O$  with radius  $r = \frac{1}{2}$ . We want to find optimal partition of this ball into parts of smaller diameter. Formally, we estimate the quantity

$$d_n = \sup_{A \subset \mathbb{R}^n, \text{diam}(A)=1} \inf \{x \in \mathbb{R}_+ : \exists A_1, \dots, A_{n+1} \subset \mathbb{R}^n : A_1 \sqcup A_2 \sqcup \dots \sqcup A_{n+1} = B_n \text{ and } \text{diam}(A_i) \leq x \quad \forall i \in \{1, 2, \dots, n+1\}\}.$$



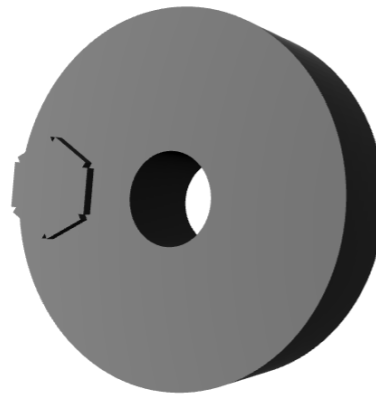
```

1 # Simple disk is created
2 result = cq.Workplane("XY")
3         .circle(10).circle(40).extrude(20)
4

```

**Listing (1) Structural Error:** The initial program completely misses the core concept of gear teeth. There was an attempt to generate it, but it was unsuccessful.

(a) Iteration 1: Structural Error



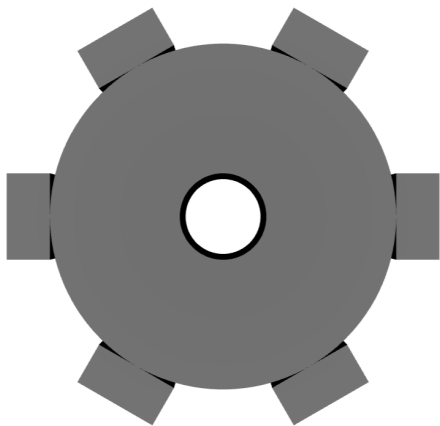
```

1 # Loop fixed to add visible protrusions
2 for i in range(6):
3     # ... (code for a triangular wedge)
4

```

**Listing (2) Protrusions improvement:** Evolution tries to improve the protrusions, it still structurally wrong.

(b) Iteration 2: Protrusions improvement



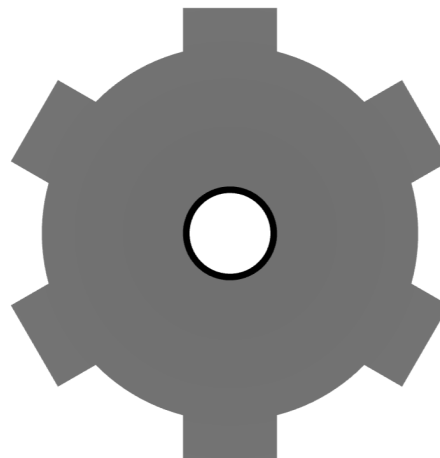
```

1 # Parameters closer but not correct
2 cog = cq.Workplane("XY")
3     .rect(20, 10)
4     .extrude(cog_height + 2)
5     .translate((0, outer_radius +
6               cog_length / 2, -2))

```

**Listing (3) Structural Leap:** The proper cog structure is found, but position (as parameter) need refinement.

(c) Iteration 16: Structural leap



```

1 # Final, correct parameters
2 .extrude(cog_height)
3 .translate((0, outer_radius + cog_length / 2 -
4           2)) # proper insertion
5 # ... (rotation and union of cog and gear
6       models)

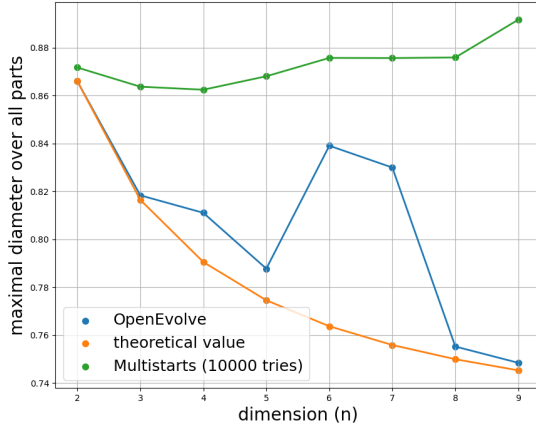
```

**Listing (4) Converged Solution:** Both structure and parameters match the target, achieving a high score.

(d) Iteration 98: Converged Solution

**Figure 9. Evolutionary Pathway for the Gear Shape, Demonstrating Correction of Structural and Parametric Errors.** The evolutionary process is shown in four stages. (a) Initial attempts often result in structural errors. (b) The process tries to fix the topology but make mistakes. (c) A key structural leap introduces the correct topology, even with incorrect parameters. (d) Finally, the process converges to a solution that is both structurally and parametrically correct.

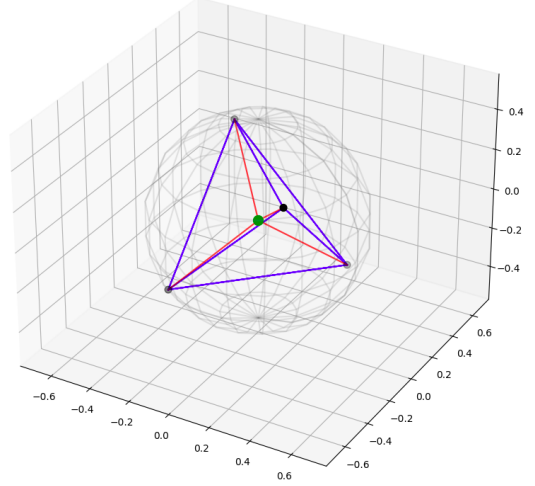




**Figure 10.** Comparison of the theoretical value for optimal partitions and two experimental ways: via OpenEvolve and via the best partition over multiple runs (multistart strategy).

The partition can be parametrized via  $n + 1$  points  $C_1, C_2, \dots, C_{n+1} \in \partial B_n$  located at the ball's surface. If the  $O \notin \text{conv}(C_1, \dots, C_{n+1})$ , then the maximal diameter of partitions based on  $C_1, \dots, C_{n+1}$  equals 1, else these points parametrized the balls partitions into cones centered at  $O$  of smaller diameter (the construction of these partitions have been obtained from facets of the above mentioned convex hull in  $\mathbb{R}^n$ ). We apply OpenEvolve approach to find the  $(n + 1)$  points on  $\partial B_n$  parametrized the optimal ball's partition and the goal is to minimize the maximum over the diameters of the partition cones. As we mention in the Methodology section, this value theoretically equals  $d_n = \sqrt{\frac{n+1}{2n}}$  and achieved on the regular simplex based construction (for instance, regular triangle vertices in planar case or regular tetrahedron in  $\mathbb{R}^3$ ). During this experiment we apply OpenEvolve approach to find optimal balls partitions for each dimension from 2 to 9 separately. The results of the comparison of OpenEvolve based estimations with theoretical results and Monte Carlo sampling methods (best partition over 10000 samples of points  $C_1, C_2, \dots, C_{n+1}$  uniformly distributed on  $\partial B_n$ ) are presented at the Figure 10. The prompt for this task has been provided at the Appendix of this paper.

Interestingly to note, that experimental results coincides with the theoretical estimations, and the model learns the optimal regular simplex based structure of partitions (Figure 11 illustrates the regular simplex vertices that have been learned via this approach in case of  $n = 3$ ). It's important to highlight, that in the experiment we slightly change the prompt to generalize our task and apply it for  $n = 13$ , then the program learns the optimal partition structure (vertices of regular simplex) for all dimension values from  $n = 2$  to  $n = 13$ . This example shows the generalization ability of evolution based approaches and highlights the promising way to use evolution-based approaches like OpenEvolve for more complicated partition problems in the combinatorial geometry. Particularly, it can be applied for the study of partitions of other universal covering sets (not only the ball form Young's theorem) into parts of smaller diameter and for other problems re-



**Figure 11.** Optimal partition of 3-dimensional ball with unit diameter into 4 parts based on the regular simplex. The red lines (segments between  $O$  and  $C_1, C_2, C_3, C_4$ ) indicates the partition cones edges.

lated to the Borsuk problem [1]. One of the main difficulties here related to the complex structure of evaluation part of OpenEvolve pipeline - for high-dimensional non-trivial universal covering sets the computation of the diameter of partition sets is a hard problem, however this issue will be mitigated during further research work on this topic.

## 5 Conclusion

The open-source implementation of AlphaEvolve, OpenEvolve, proves to be a versatile and effective framework for complex optimization tasks, outperforming single LLMs by iteratively refining a population of solutions to systematically achieve more robust outcomes. Our study successfully applies this LLM-driven evolutionary search approach to a new domain: CAD reconstruction. By doing so, we not only create a new benchmark for this specific problem but also demonstrate the broader potential of this methodology for solving other complex challenges, such as combinatorial geometry problems.

## References

- [1] Karol Borsuk. "Drei Sätze über die  $n$ -dimensionale euklidische Sphäre". In: *Fundamenta Mathematicae* 20 (1933), pp. 177–190.
- [2] Weiqiang Jin et al. *A Comprehensive Survey on Multi-Agent Cooperative Decision-Making: Scenarios, Approaches, Challenges and Perspectives*. 2025. arXiv: 2503.13415 [cs.MA]. URL: <https://arxiv.org/abs/2503.13415>.
- [3] Mohammad Sadil Khan et al. "Text2CAD: Generating Sequential CAD Designs from Beginner-to-Expert Level Text Prompts". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: <https://openreview.net/forum?id=5k9XeHIK3L>.
- [4] Mohammad Sadil Khan et al. "Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 7552–7579.
- [5] Maksim Kolodiaznyi et al. *cadrille: Multi-modal CAD Reconstruction with On-line Reinforcement Learning*. 2025. arXiv: 2505.22914 [cs.CV]. URL: <https://arxiv.org/abs/2505.22914>.
- [6] Alexander Novikov et al. *AlphaEvolve: A coding agent for scientific and algorithmic discovery*. 2025. arXiv: 2506.13131 [cs.AI]. URL: <https://arxiv.org/abs/2506.13131>.

- [7] Asankhaya Sharma. *OpenEvolve: an open-source evolutionary coding agent*. 2025. URL: <https://github.com/codelion/openevolve>.
- [8] Alexander Soifer. “The Hadwiger-Nelson Problem”. In: *Open Problems in Mathematics*. 2016.
- [9] A. D. Tolmachev and D. S. Protasov. “Covering Planar Sets”. In: *Doklady Mathematics* (2021). DOI: 10.1134/S1064562421040141.
- [10] A. D. Tolmachev, D. S. Protasov, and V. A. Voronov. “Coverings of planar and three-dimensional sets with subsets of smaller diameter”. In: *Discrete Applied Mathematics* 320 (Oct. 2022), pp. 270–281. DOI: 10.1016/j.dam.2022.06.016.
- [11] Rundi Wu, Chang Xiao, and Changxi Zheng. “DeepCAD: A Deep Generative Network for Computer-Aided Design Models”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 6772–6782.
- [12] Jenny Zhang et al. *Darwin Godel Machine: Open-Ended Evolution of Self-Improving Agents*. 2025. arXiv: 2505.22954 [cs.AI]. URL: <https://arxiv.org/abs/2505.22954>.

## 6 Appendix

### 6.1 Inference Resources Details

**6.1.1 Ascend TPU** To experiment with local inference a remote host with Huawei Ascend TPUs was provided. We used Huawei’s ModelArts platform to develop and execute the inference workflow. ModelArts is a one-stop AI development platform providing managed notebook, training, and deployment services support. In particular, we created a ModelArts Notebook instance with the Ascend-enabled image, giving us an interactive Python environment. This notebook instance came preconfigured with the necessary Ascend drivers and frameworks. All models and data were stored in Huawei’s cloud storage: we used Object Storage Service (OBS) to host the model files. OBS provides scalable bucket storage for larger model assets. We mounted these storage resources into the notebook so that our model files could be loaded at runtime. Importantly, since our work involves inference only, we did not perform any training or fine-tuning on these models – we simply loaded them from storage into the Ascend-accelerated runtime. We ran all computations on a virtual machine with eight Ascend Snt9B3 NPUs. Each Ascend Snt9B3 card provides 32 GB of high-bandwidth memory, it gives up to 256 GB. This amount of high-bandwidth memory provided opportunity to load few models with more than 70 billions parameters and a large context size. The host node uses a 192-core ARM processor with 1.536 TB of RAM.

**6.1.2 Yandex DataSphere GPU** As an alternative we used a Yandex DataSphere service for the same purposes. It offers up to 32 GB of VRAM which may be not enough for our experiments.

**6.1.3 Inference API** For quick experiments we employed OpenRouter API, GPT4Free, and Mistral API. Initial experiments included usage of DeepSeek R1 671B, mistral-small-latest, and mistral-large-latest models. Finally, we applied GPT-o4-mini, Qwen2.5-72b, and Qwen2.5-Coder-32b as backbone LLMs for our experiments with OpenEvolve.

### 6.2 Dataset details

As outlined in Section 3.3.1, we used the Text2CAD dataset [4] as our starting point. We later realized that we did not need the Text2CAD reference code, because our experiments compare the final 3D models rather than the code. Based on this, our dataset includes (i) a short test description, example of which is shown

in Table 4, (ii) the figure name, and (iii) a Python script using the `trimesh` library to recreate the ground-truth 3D model. In addition to figures taken from Text2CAD, we also created several figures ourselves to cover missing cases.

Our dataset is available at link.

The original Text2CAD dataset is at link.

**Table 4**

Comparison of Prism CAD Model Descriptions

Initial Text2CAD Description
Create a new coordinate system with Euler angles set to 0 degrees for the first 2 axes and -90 degrees for the third axis. Set the translation vector to 0 for the first and third axes and 0.6666 units for the second axis. Draw a 2-dimensional sketch on the first face. In this sketch, start by drawing the first loop which consists of 11 lines. The first line starts at coordinates (0, 0.3566) and ends at (0.0716, 0.1362). Continue drawing the subsequent lines according to the specified coordinates until the eleventh line, which connects back to the starting 0. After completing the first loop, apply a scale factor of 0.75 to the entire 2-dimensional sketch. Transform the scaled 2-dimensional sketch into 3 dimensions by rotating it using the same Euler angles and translating it using the same translation vector. Extrude the transformed sketch 0.6666 units along the normal direction to create a solid body. The final dimensions of the rounded cube base are approximately 0.75 units in length, 0.7133 units in width, and 0.6666 units in height
Our Simplified Description
Prism with height 200mm with right 10-sided polygon base. Length of one side of polygon is 20mm

### 6.3 OpenEvolve evaluator prompt for CAD Reconstruction

```

1 You are an expert in parametric 3D modeling using CadQuery and Python. Your task
  is to write a Python function using the CadQuery library that generates
  a 3D model matching a reference information about the shape as closely as
  possible.
2
3 Requirements:
4 1. The code must use CadQuery primitives and operations.
5 2. The function should return final CadQuery solid object ('cq.Workplane' with 3
   D geometry).
6 3. The script must be executable in a standard Python environment with CadQuery
   installed (no other packages).
7 4. Remove all comments and descriptions from the solution code
8
9 You have a text shape description with important information.
10
11 Here are description of metrics on which
12 the result will be evaluated (with respect to original object):
13 - 'iou' (Intersection over Union) measures the overlap between two meshes based
   on their volume.
14
15 - 'viou' (Voxelized IoU) measures the IoU on a voxelized representation of the
   meshes
16
17 - 'cd' (Inverse Chamfer Distance) measures the average squared distance between
   nearest points on the two mesh surfaces.
18 Concept: '1 - (mean_dist(P1 -> P2))^2 + mean_dist(P2 -> P1)^2)'
19
20 - 'hd' (Inverse Hausdorff Distance) measures the maximum "worst-case" distance
   between the two surfaces. It finds the point on one surface furthest from
   any point on the other. Calculated on 5000 sampled points.
21 Concept: '1 - max(max_dist(P1 -> P2), max_dist(P2 -> P1))'
22
23 - 'wd' (Inverse Wasserstein Distance) measures the minimum "cost" to transform
   one surface's point cloud into the other.
24 Concept: '1 - EMD(P1, P2)'
25
26 - 'as' (Surface Area Similarity) compares the scalar surface area of two meshes.
27 Formula: '1 - |Area1 - Area2| / max(Area1, Area2)'
28
29 - 'is' (Inertia Similarity) compares the moment of inertia tensors, which
   describe the mass distribution and rotational properties of the objects.
30 Formula: '1 - ||Inertia1 - Inertia2|| / (||Inertia1|| + ||Inertia2||)'
31
32 Shape description:
33 A 80x60x10 block (length x width x height) with central hole of radius 22

```

### 6.4 OpenEvolve initial program for CAD Reconstruction

```

1 # EVOLVE-BLOCK-START
2 """Function that builds 3D figure by text description"""
3
4 import cadquery as cq
5
6
7 def build_3d_figure() -> cq.Workplane:
8     return cq.Workplane()
9
10
11 # EVOLVE-BLOCK-END

```

### 6.5 OpenEvolve evaluator prompt for the ball partition

```

1 You are an expert in combinatorial geometry and Python.
2 Your task is to write a Python function that generates an optimal k-parts
3 partition of a n-dimensional ball of given radius
4 where k = (n + 1), so as to partition the ball into (n + 1) conical regions
5 whose maximal diameter is as small as possible.
6 You can change function in all possible ways as you want as long as it fits
7 requirements and constraints and improves evaluation metrics.
8
9 Requirements:
10 1. The code must use only standard Python and NumPy primitives and
11 operations.
12 2. It must return a NumPy array of shape (k_points, n_dim) whose rows are
13 the coordinates of the k_points points on the sphere of radius `radius`.
14
15 Constraints:
16 - Points must lie on the surface of the n-dimensional ball of the
17 given `radius`.
18 - The points define a convex hull. If the center of the ball lies inside
19 this convex hull, the configuration is invalid (validity = 0).
20 - The partition is formed by cones whose apex is at the center of the ball
21 and whose bases are the facets (triangles) of the convex hull formed by
22 the points.
23
24 Evaluation metrics (computed externally):
25 - `validity`: 1.0 if the shape of the point cloud matches the requirements (
26 see above), otherwise 0.0.
27 - `max_diam`: the maximum diameter among all conical regions (i.e., the
28 largest distance between any pair of points in each partitions cone),
29 this value cannot exceed the ball diameter, it is necessary to minimize
30 this value.
31 - `target_ratio`: ratio of the theoretical minimum possible diameter to the
32 actual `max_diam` (higher is better, capped at 1).
33 - `combined_score`: validity x target_ratio (your main optimization
34 objective).
35
36 Your goal:
37 Maximize the `combined_score`, which is equal to `target_ratio` if
38 the packing is valid and zero otherwise. Max_diam should be minimize

```

### 6.6 OpenEvolve initial program for the ball partition

```

1 # EVOLVE-BLOCK-START
2 """Constructor-based the partition of n-dimensional sphere
3 into (n + 1) parts of smaller diameter"""
4
5 import numpy as np
6 from scipy.spatial import ConvexHull, distance
7 import matplotlib.pyplot as plt
8 from itertools import combinations
9 import scipy.stats as sps
10
11
12 def construct_packing(n_dim, k_points, radius=0.5):
13     """
14     Construct a random partition
15
16     Returns:
17     ... points
18
19     # Initialize points via random sampling from the uniform sphere distribution
20
21     points = sps.uniform(loc=-1, scale=2).rvs(
22         (k_points, n_dim)
23     )
24     points = points / np.linalg.norm(points, axis=1, keepdims=True)
25     points *= radius
26
27     return points
28
29 # EVOLVE-BLOCK-END

```

### 6.7 Evolutionary experiment visual results

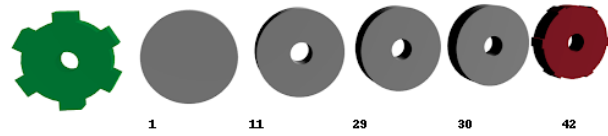


Figure 12. OpenEvolve evolution results for the gear figure using the Qwen2.5-72b model with each stage labeled by its iteration number below.



Figure 13. OpenEvolve evolution of the gear figure generated by alternating between Qwen2.5-72b and Qwen2.5-Coder-32b (50% each), with each stage annotated by its iteration number below.

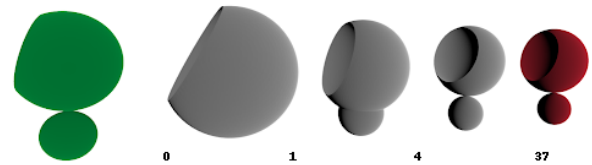


Figure 14. OpenEvolve evolution results for the spheres figure using the Qwen2.5-72b model with each stage labeled by its iteration number below.

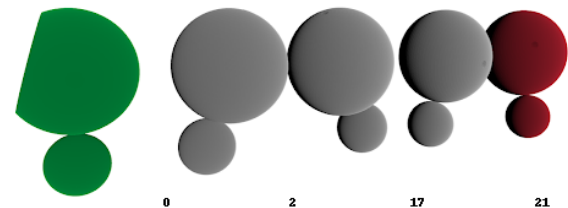


Figure 15. OpenEvolve evolution of the spheres figure generated by alternating between Qwen2.5-72b and Qwen2.5-Coder-32b (50% each), with each stage annotated by its iteration number below.

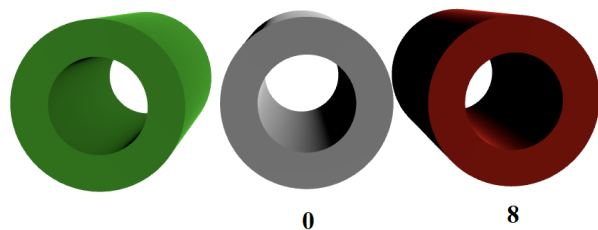
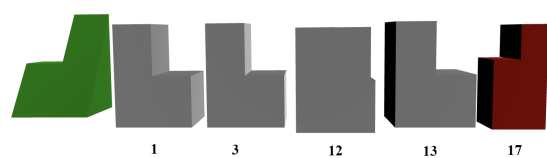
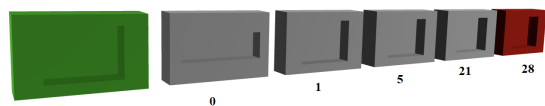


Figure 16. OpenEvolve evolution of the tube figure generated by alternating between Qwen2.5-72b and Qwen2.5-Coder-32b (50% each), with each stage annotated by its iteration number below.



**Figure 17.** OpenEvolve evolution of the ladder figure generated by alternating between Qwen2.5-72b and Qwen2.5-Coder-32b (50% each), with each stage annotated by its iteration number below.



**Figure 18.** OpenEvolve evolution of the open box figure generated by alternating between Qwen2.5-72b and Qwen2.5-Coder-32b (50% each), with each stage annotated by its iteration number below.