BRICK DANCE: CHOREOGRAPHING LEGO BUILDS FROM MOTION SPECIFICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models write working software, yet they cannot reliably build working machines. The hard part of mechanical engineering is rarely a single part: it is an assembly of many parts whose connections must produce the motion a specification demands. We study this gap in LEGO, since its parts are fixed, expert-designed assemblies are plentiful, and, as we show, a machine can check whether an assembly holds together and moves as asked. Prior work on LEGO treats a build as a static shape, examining only whether its bricks connect and stand. We treat it as a mechanism, whose parts must move as specified: a door that opens $140°$ and carries twenty-five pieces with it. Such requirements are much closer to real mechanical engineering. To evaluate them, we build a decompiler that recovers a LEGO model's connections, its rigid bodies, and how far each joint turns before it collides, from its raw geometry. It reads 1,464 human-built models, and from any of them it derives a functional requirement specification without human authoring, which is what Brick Dance grades against, to our knowledge the first LEGO benchmark to evaluate motion and editing. Finally, we propose Brick Dance Copilot, a coding agent that uses our decompiler as tools and feedback: holding the model and the budget fixed, it submits a valid assembly on 89% of runs against 47% for an unmodified coding agent, and the gap is almost entirely physical.

1 INTRODUCTION

Coding agents have had a remarkable impact on software engineering, but are yet to be useful for mechanical engineering. Recent work has begun to apply language models to CAD, generating parametric models and sketches from text, image and code (Khan et al., 2024; Wang et al., 2025; Zhang et al., 2025; Li et al., 2025). Despite these successes, having coding agents build mechanical engineering *assemblies* remains a challenge, especially subject to user requirements and specifications.

In this work we examine how good today's coding agents are at producing valid mechanical assemblies. We use LEGO[1] to evaluate this ability. LEGO lets us evaluate assembly specifically, without entangling it with a model's ability to produce CAD geometry. LEGO has two further advantages: a large body of data is available online from official sets and from amateur and hobbyist builders, and a finished build can be cheaply assembled and verified in the real world, unlike game-based environments. We propose that a model that can produce a valid mechanical assembly should also be able to produce a valid LEGO assembly, and so seeing the limit of an AI system's ability to build with LEGO is informative about the more general AI mechanical engineering ability.

We ask a specific question: can a coding agent act as a mechanical engineer— take in requirements and produce a design that satisfies them?

Our answer is a benchmark. The obstacle to building one is that a LEGO file, like a CAD file, states only where parts are, so there is nothing in it to grade against: a door that swings and a door fused shut are the same list of coordinates. We therefore build the benchmark and the machinery that makes it gradeable together. Brick Dance poses tasks that are *generated* rather than written. Our

[1]LEGO® is a trademark of the LEGO Group, which does not sponsor, authorise or endorse this work and is not affiliated with it. We use the mark only to identify the physical building system our data describes.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

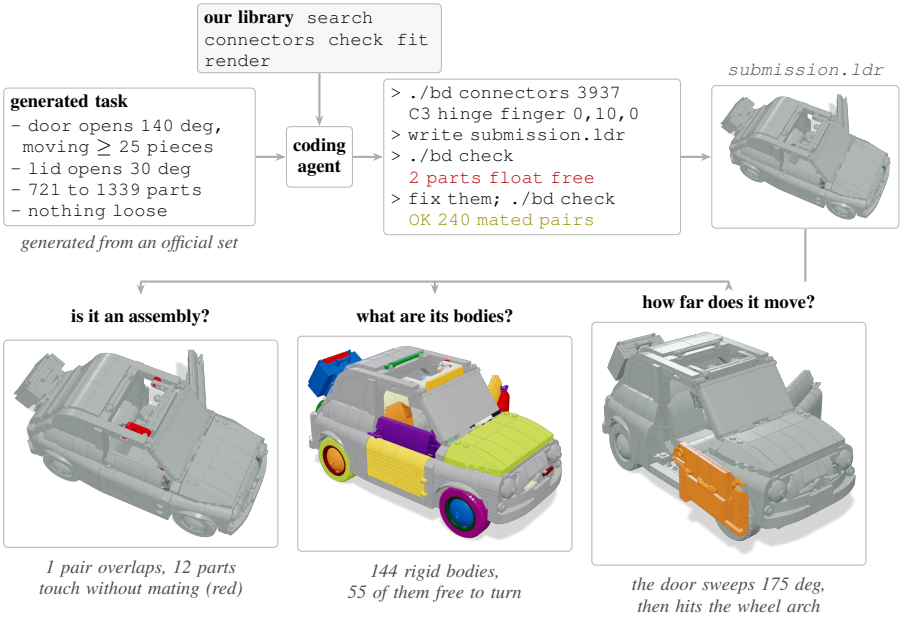


Figure 1: **Brick Dance.** We generate build specifications from human built LEGO sets, build a grader to evaluate submissions according to requirements, and see how well coding agents perform

decompiler reads 1,464 official LEGO sets—designed by LEGO, sold as products, and therefore known to be buildable—and turns any of them into requirements that state what the model must *do*, such as a door that opens 140° carrying at least twenty-five pieces, and it poses each set on four tracks that differ only in how much of the answer is shown. The machinery underneath is a decompiler that recovers typed connections between parts, merges them into rigid bodies and the joints between them, and sweeps each joint to a measured limit of travel; on top of it sits a grader that answers the three questions a raw LDraw file cannot (Figure 1), scoring validity and motion on separate axes so that a physical failure cannot hide behind a plausible silhouette. To our knowledge this is the first LEGO benchmark whose specifications state required motion and whose grader verifies that motion on the produced assembly, and the first to pose *editing*: finishing a half-built model without breaking what already works.

The benchmark's first result is that, for a fixed model, the environment roughly doubles how often a build is a real assembly. Holding the agent, the model, the clock and the spending cap fixed and varying only what the environment offers, an unmodified coding agent produces a valid assembly in 47% of runs and one given our connection library in 89%, at the same cost per run; the gap is almost entirely physical—parts driven into one another, or joined to nothing. None of it is visible in a rendering the agent can see. Because the harness asks only for an LDraw file, it accepts a submission from any system.

2 Related Work

Generating LEGO with learned models. The past few years have seen growing interest in using learned models to build LEGO constructions. Table 1 summarises it. BrickGPT emits brick-by-brick build sequences from text and enforces buildability during decoding with a validity check and a physics-aware rollback (Pun et al., 2025); Brick-by-Brick had earlier cast the problem as reinforcement learning over a variable-sized action space (Chung et al., 2021); BrickAnything conditions on geometry and tokenises structure by local attachment relations (Ni et al., 2026); rollback-free generation trains the stability constraint into the policy so that no simulator is needed at inference (Xu et al., 2026); and post-training recipes target the spatial and physical reasoning such generation demands (Yuan et al., 2026). Stability has been studied directly by StableLego, which infers the structural stability of block-stacking assemblies from force balance (Liu et al., 2024). Closest to

System	Parts	Conn.	Stab.	Motion	Edit
Legolization (Luo et al., 2015)	cuboid, grid	—	✓	—	—
Technic (Xu et al., 2019)	beams, gears	✓	✓	annot.	—
Brick-by-Brick (Chung et al., 2021)	unit bricks	✓	—	—	—
Break and Make (Walsman et al., 2022)	>1,000 shapes	✓	—	—	rebuild
StableLego (Liu et al., 2024)	cuboid, grid	—	✓	—	—
BrickGPT (Pun et al., 2025)	cuboid 20^3	✓	✓	—	—
BrickAnything (Ni et al., 2026)	brick library	✓	✓	—	—
Rollback-free (Xu et al., 2026)	cuboid, grid	✓	—	—	—
BrickNet (Kulits & Schmid, 2026)	1000s of types	✓	—	—	—
Brick Dance (ours)	full LDraw library	✓	✓	✓	✓

Table 1: Prior work on LEGO construction, as each work reports it. *Conn.*: a real connection between parts is checked, not voxel adjacency. *Motion*: motion is specified and verified. *Edit*: it operates on an existing assembly.

our representation is BrickNet, which generates build sequences over thousands of part types—rather than cuboid bricks on a grid—by way of a graph-based program that parametrises structure through *connectivity* (Kulits & Schmid, 2026). We arrived at a connectivity-first representation independently and for a different purpose: BrickNet uses it to make generation physically grounded, whereas we use it to *decompile* existing sets, so that a specification and a grader can be derived from models that are known to be buildable. What is missing throughout is any notion of a successful build beyond valid connections and stability, which leaves out the vehicles, doors and mechanisms that make up much of real LEGO—and that are what make it a proxy for mechanical assembly at all.

Rebuilding, optimising, assembling. Other work builds LEGO without learning from text prompts. Break and Make asks an agent to disassemble a model and prove its understanding by rebuilding it in the LTRON simulator, over more than a thousand brick shapes (Walsman et al., 2022); its objective is reconstruction of a known target rather than satisfaction of a stated requirement. From the graphics community, Legolization lays out bricks to realise a target shape under a force-based stability metric (Luo et al., 2015), and, most relevant to us, Computational LEGO Technic Design synthesises beams, axles and gears from user sketches with optional *motion annotations* (Xu et al., 2019)—but only from extensive human input.

Outside LEGO, JoinABLe learns the joint that mates two parametric CAD parts and releases the Fusion 360 Gallery assembly dataset (Willis et al., 2022). Language models reached CAD recently and first for single parts, from text (Khan et al., 2024), from an image (Wang et al., 2025), or through parametric code (Zhang et al., 2025; Li et al., 2025); assemblies have followed within the year, as ASSEMCAD compiles a natural-language brief into typed parts and executable mates (Dong et al., 2026) and ArtiCAD generates articulated assemblies, declaring joint parameters before any geometry is built (Shui et al., 2026). What none of them evaluates is an agent going from a stated requirement to the joints that satisfy it, and then checking that those joints move. Functional evaluation has been performed in game worlds. BesiegeField scores machine programs by simulated physical behaviour and reports that strong models struggle with spatially precise assembly (Zhang et al., 2026), BuildArena spans static and dynamic mechanics (Xia et al., 2026).

Across all of this, motion is either absent, supplied by hand as an annotation, or checked only inside a bespoke simulator with its own parts and its own physics. Brick Dance asks for motion in the specification, verifies it on the assembly the agent produced, and does so over the real part library against ground truth that was manufactured and sold.

3 Methodology and Data

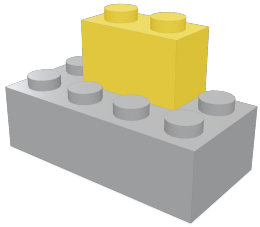
3.1 Representing Brick Assemblies

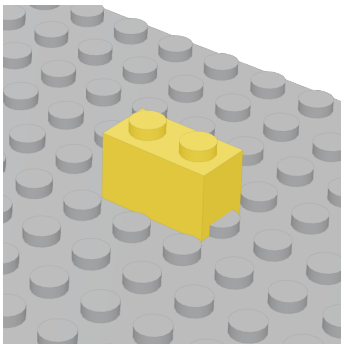
How would a code-writing model build a mechanical LEGO assembly at all? The universally adopted way to specify LEGO through code is LDraw, an open format that the community has

```
1 71 0 0 0    1 0 0 0 1 0 0 0 1 3001.dat
1 14 0 -24 10 1 0 0 0 1 0 0 0 1 3004.dat
```

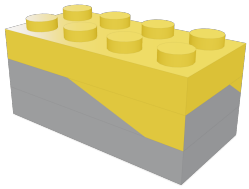
type, colour	position $x\ y\ z$ (LDU)	rotation (row-major 3×3)	part file

The first line a grey 2×4 brick (`3001`) at the origin; the second places a yellow 1×2 brick (`3004`) 24 LDU higher, which happens to correspond to a valid connection.

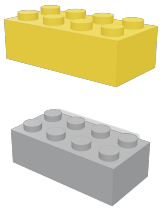




(a) it almost mates. `10.0 LDU (0.0 along the axis, 10.0 across) and 0.0° off`



(b) it is inside the other part. `interpenetrates (4580 LDU³ deep)`



(c) it is not attached. `floats free --- no mate and no contact`

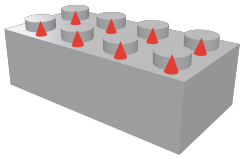
Figure 2: LDraw specifies *placements*, not connections. Top: a two-line file and the model it denotes. Bottom: three models that can be specified in LDraw, but do not correspond to a physically valid build.

used for decades to describe official and custom models. A model is a list of placements. Each line names a part file, a colour, an x, y, z position in LEGO distance units (LDU) and a 3×3 rotation matrix; a part file gives the exact geometry of that piece.

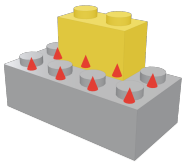
LDraw offers a flexible representation and is easy to render. Its limitation, as a language for *building*, is that it captures no information about the logical connections between parts. A syntactically valid line can specify a placement that is not a valid LEGO joint at all, or that drives one part through another, and these failures are impossible to see from LDraw code alone and difficult to see from renders because of occlusion. The problem has been addressed in prior work such as (Pun et al., 2025; Kulits & Schmid, 2026) through a *connection-based* representation, and we address it in a similar way by building a connection language and a decompiler. Every part in the library carries typed connectors—round cylinders (studs, pins, axles, and the holes that receive them), clips, hinge fingers and generic mounts—each with a local position, an orientation, a gender and a section profile giving its radius and length. We use these to determine all the places on two parts that might be in connection, and then test each candidate. Over the $1{,}464$ sets of our corpus this recovers a median of 84% of the seatings a set uses, with failures concentrated in very large Technic models and in flexible elements such as hoses and chains that have no rigid seating to find. What is not recovered is reported as unverifiable rather than accepted, and a submission is only penalised for defects the original does not already have, so a gap in the part library costs an agent nothing. What we get back is a set translated into the language of connections, and a language a model can build in: instead of a raw position it states how a part mates to those already placed, over the full range of LEGO connections rather than brick-on-brick stacking alone.

3.2 Grading a build, and the benchmark it grades

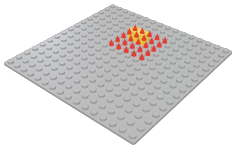
Our grader takes two things: a raw LDraw file and a specification—"a door that turns $140°$"—and answers whether the file satisfies it. We commit to raw LDraw as the input so that the benchmark



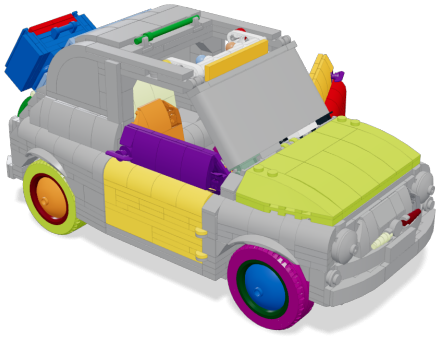
(a) every connector, placed. Typed features from the part library: studs and pins, sockets and holes, clips, hinge fingers. Lattices are expanded, so a mate is between two features and not two parts.



(b) a mate. Axes opposed, axis lines coincident, radii equal, profiles overlapping. The type fixes the constraint: a stud is rigid, a round bore is revolute.



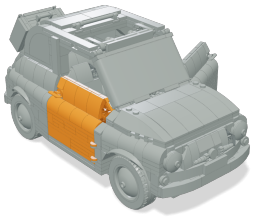
(c) not a mate. The same test fails by 10 LDU across the axis. Recorded as a contact, never silently as a joint.



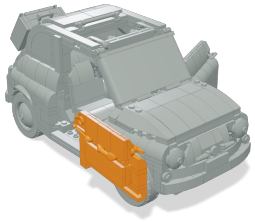
(d) rigid bodies. `10271-1`, the Fiat 500.

parts	1,030
rigid bodies	144
largest body (chassis)	581
movable joints	55

Grey is the chassis, which is every part the rigid mates weld into one piece. Everything that can move relative to it gets a colour of its own: the front lid, the door, each wheel with its hub, the seats. Nothing here is labelled by hand.



(e) shut, $0°$**.** 51 parts turn with this bearing.



(f) open, $-175°$**.** The measured limit. The door lies back along the front wing.

```
% (g) why the arc ends
+5°   3005 Brick 1 x 1 into
11211 Brick 1 x 2 w/ studs
1224 LDU^3 deep (tol 60)
-180° 99780 Bracket 1 x 2 into
62361 Wheel Arch
91 LDU^3 deep (tol 60)
```

arc $= [-175°,\ 0°]$, 51 parts moving

Figure 3: The decompiler, end to end, on one official set. Algorithm 1, part I, recovers connections (a–c); part II merges them into rigid bodies and the joints between them (d); part III drives each joint until it collides and reports the arc together with the pair of parts that ended it (e–g). Both ends of this door's arc are contacts, so both are reported. That last number is what a requirement such as "a door that opens $140°$" is checked against.

is open to any approach: a language model, a search procedure or a human can submit, and nothing about how the file was produced is inspected.

To answer that question we have to answer three others in order. Does the file represent a build at all, which is to say, can it be read as connections? If it can, what rigid bodies does it have? And do those bodies move as far as was asked? A submission fails at the first of these it fails, and the grader reports which, so a failure is locatable rather than a score. Figure 3 shows all three stages on one official set.

Connections. We place every connector in world coordinates, expanding lattices so that the 4×2 studs of a 2×4 brick are eight features rather than one, and bucket them into a cubic grid at 8 LDU, with long axles and bars rasterised along their span. Every pair of connectors sharing a cell

Algorithm 1 From an LDraw file to a mechanism: connections, rigid bodies, motion.

I. CONNECTIONS

place every connector of every part in world coordinates, expanding lattices
bucket them into an 8 LDU grid, rasterising long axles and bars along their span
for each pair (a, b) in one cell or its neighbours, of opposite gender and compatible type **do**
    if $|\cos\theta| \geq 0.995$, axis lines within 0.75 LDU, radii within 0.6 LDU, and profiles overlapping by ≥ 4 LDU
    then record a MATE; **else if** the parts touch, record a CONTACT
end for

II. RIGID BODIES AND JOINTS

classify each mate RIGID (stud, keyed axle, friction pin) or BEARING (round bore, clip–bar, hinge fingers)
repeat ▷ welds can be created by welds
    union the two sides of every RIGID mate
    reclassify as RIGID any BEARING whose two sides now share a body
until no bearing changed class
for each remaining bearing j between bodies (A, B) **do**
    j.pinned $\leftarrow$ **not** IsBridge(j) ▷ a second load path holds it still
end for

III. RANGE OF MOTION

for each unpinned joint j, largest moving side first **do**
    base$[m, s] \leftarrow$ DeepOverlap(m, s) at rest, for every near pair ▷ a bearing interpenetrates by construction
    for each direction $d \in \{+1, -1\}$ **do**
        step θ by $d \cdot 5^\circ$, rotating j's subtree about its axis and pivot, until some pair exceeds base$[m, s] + \tau$
        $\text{limit}_d \leftarrow$ the last free angle; $\text{stop}_d \leftarrow$ the colliding pair
    end for
end for
return mates and contacts; bodies and joints with axis and pivot; per joint an arc $[\text{limit}_{-1}, \text{limit}_{+1}]$, its moving mass, and what stopped it

or a neighbouring one is a candidate, and a geometric test decides whether it is a connection and which seating it uses (Algorithm 1). A pair that touches but fails the test is a *contact*; a pair with no readable mate is *unverifiable*. Neither is ever recorded silently as a joint, and that three-way verdict is what lets us tell a near-miss from a mate.

Bodies, and how far they turn. A mate's type fixes a constraint. Studs, keyed axles and friction pins resist rotation; round bores, clips on bars and hinge fingers leave one rotational degree of freedom. Merging the rigid mates is not enough, because two parts held by two bearings whose axes are *not* colinear cannot turn about either: a door on two hinges in a line swings, the same door on two hinges out of line does not. We therefore alternate merging and reclassifying to a fixpoint, and a bearing with a second load path through the body graph is marked pinned. Each unpinned joint is then driven from its rest pose in 5° steps until the swept parts hit something. What comes back is an arc in degrees, the number of pieces that travel with it, and the pair of parts that ended it.

What a requirement checks. A kinematic requirement is a count over that mechanism: at least n unpinned bearings whose swept arc is at least θ, each carrying at least k pieces, about a vertical or horizontal axis. Every term is measured from the submitted file, which is what makes the requirement decidable by anyone holding it; the human-readable name, such as "door", records which part of the original the thresholds came from. The same pass checks part count, footprint and height as bands around the original, the number of separate objects, and the absence of any collision or loose part beyond those the official model itself carries. On the tracks that supply them it also checks that every part came from the bag the agent was given and that the half-built model it started from is still in place. Appearance is not scored.

Corpus and tasks. Our corpus is the official LEGO models published as LDraw files by the Official Model Repository, of which our decompiler reads $1{,}464$ in under a minute. These are not machine-generated designs: each was designed by LEGO, sold as a product and transcribed brick by brick, so each is known to be buildable because it was built. From any of them we generate a task rather than write one. We run the decompiler, convert its measurements into typed requirements, and render the model from four fixed cameras for a vision–language model to describe; the requirements and that description form the prompt. A requirement is kept only if the original passes it, so no specification demands more than the model it came from, and joints that are articulation rather than mechanism are excluded by part identity. That filtering matters: the Lincoln Memorial yields 46 raw joints and none is a function of the model, while a service station yields 13 of which one is.

Sample and tracks. Running every set is neither affordable nor informative, so our experiments draw a stratified sample with a fixed seed from the 603 eligible sets, those the decompiler reads with between 80 and 400 parts. Below that a set poses no design problem; above it the exercise measures throughput. We stratify by size tercile crossed with whether the model has any moving function. We drew ten sets and report nine: one places its parts through LDraw sub-part references our part index does not resolve, so no specification could be generated and no agent ever saw it, and we report the draw rather than redrawing it. Each set is posed four times, differing only in how much of the answer is shown, and judged by the same grader. **T1** gives the specification alone. **T2** adds four renders of the finished original. **T3** adds the exact bag of parts it was built from, which the build may not exceed. **T4** adds the first half of the original's own build order, already assembled, which the agent must preserve while finishing the model. T4 is the editing track, and it is the one no existing LEGO benchmark poses.

3.3 Agents

We evaluate two agents. Both are the same coding agent in the same harness with the same task, the same wall-clock budget and the same spending cap. They differ only in what the environment offers. We run the vendor's own agent as shipped, in print mode with its native tools, in a working directory outside our repository, and we record the instructions, the tools granted, the command line and the full event stream. Every run is capped at twenty minutes and two dollars, and the agent is told that whatever is on disk when the time runs out is what will be graded. The cap is uniform, so the comparison is fair, but it is a real constraint: the results are what these agents achieve for two dollars, not what they could achieve without limit.

The **raw agent** gets no tools of ours. It receives the prompt and the official part library as files and must produce the LDraw file itself, reading part geometry out of the `.dat` files and computing every position and rotation matrix on its own. This is not a straw man. It is what a competent coding agent does when handed this problem, and in practice it rebuilds the environment it was denied. In one run it wrote its own geometry parser, its own overlap test, its own hinge sweep and its own rasteriser before it was satisfied with its work.

Brick Dance Copilot gets the connection library and a way to interrogate what it has built. It can search the part catalogue, ask a part for its studs, sockets, holes, clips and hinge fingers with positions and axes, render its current file from any camera, try a placement without committing it, and run the connection report of Section 3.2 over its own submission. It does not receive the grader, the task's postconditions or any score. It is told the requirements, since they are the prompt, but it has no way to measure them: no joint sweep, no moving-mass count, no grader, so whether a door turns 140° is something it must reason about rather than check. Its connection report does run the same parse, geometry and connectivity code as the grader's validity rungs, and we state that plainly because it is the point rather than a leak: the question we ask is what happens when an agent can see whether its placements are connections, which is exactly what a person building the model would see.

4 Experiments and Results

We run both agents over the sampled sets and all four tracks. That is nine sets, four tracks, two agents and one model, Claude Opus, for 72 runs, each under the same twenty-minute and two-dollar cap. Table 2 gives every run and Figure 4 shows what was actually built.

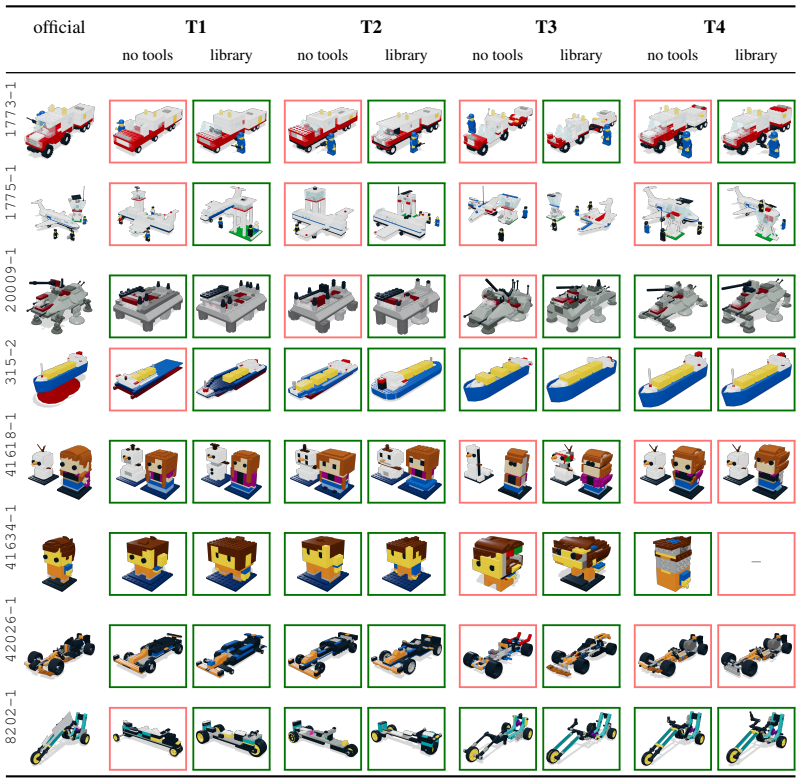


Figure 4: Every submission in the grid. Each row is one sampled set: the official model on the left, then the four tracks, each showing what the agent built without our tools and with them. A red frame means the file is not a valid assembly; a green frame means it is valid and met at least 50% of its requirements; an unframed picture is a valid build that met fewer. A dash is a run that submitted nothing. All renders share one camera and one scale, so the only thing that differs between two pictures is the model.

Table 2: Requirements met, run by run: every sampled set against every track, without our tools (–) and with them (+). A cell is the fraction of that task's requirements the submission satisfied; red marks a submission that is not a valid assembly, whatever else it met, and a dash is a run that submitted nothing.

set	T1		T2		T3		T4	
	−	+	−	+	−	+	−	+
1773-1	0/9	7/9	1/9	7/9	0/10	8/10	2/11	11/11
1775-1	1/7	4/7	1/7	5/7	0/8	3/8	2/9	8/9
20009-1	3/6	4/6	1/6	3/6	1/7	6/7	7/8	6/8
315-2	1/5	3/5	3/5	3/5	5/6	5/6	7/7	7/7
40138-1	1/6	4/6	5/6	1/6	—	—	6/8	6/8
41618-1	4/5	3/5	4/5	4/5	1/6	4/6	2/7	7/7
41634-1	3/5	3/5	4/5	4/5	1/6	5/6	5/7	—
42026-1	6/7	6/7	4/7	4/7	1/8	6/8	9/9	9/9
8202-1	1/11	10/11	10/11	10/11	9/12	10/12	12/13	12/13

Half of what the baseline builds is not an assembly. We call a submission *valid* when it is a thing that could exist: every part is real, no two parts occupy the same space, everything is joined through a recognised connection, and it stands. The agent without our tools clears that bar on 47% of its runs. The same agent, same model, same time and money, with our connection library available, clears it on 89%. The requirements themselves ask for things like a door that opens so far, or a part count inside a band. The two arms meet 46% of them and 70%. The baseline is not running out of

resources: it takes a median of 13 turns against the library arm's 22, and stops because it believes it is done. Its failures are also not gross: across 36 runs it drives parts into one another eleven times and leaves parts attached to nothing seven times, and that is usually a handful of placements out of two hundred.

The agents can design; they cannot check. Neither agent often satisfies a task outright: over 36 runs each, the baseline meets every requirement twice and the library arm four times, and in each arm half of those are marked invalid because they inherit a defect from the set they were derived from. The point is not that either agent solves these tasks. Table 2 shows where the two arms come apart. On the smaller sets they are level. On the larger and more articulated ones the baseline's builds stop being assemblies: on the fire-truck set it is invalid on all four tracks, while the library arm is valid on all four and on the editing track satisfies all eleven requirements. Nothing about the second agent makes it a better designer. It can ask whether the thing it has drawn is real, and the first agent cannot.

Mechanisms are reached; the assembly under them is not. One sampled set contains a gear train, and both agents rebuild a working one on every track. The library arm's version is a 24-tooth gear driving two 8-tooth gears. Our grader finds the meshes from centre distance and pitch radii, turns the driver through a full revolution in 10° steps without the teeth clashing, and measures the ratio at exactly 3.000 on both output shafts; the baseline reaches the same bar with three 16-tooth gears in a line. In one of those runs the train works inside a submission that is not a valid assembly at all, which is exactly the separation this benchmark is built to expose. Whatever is failing in the other runs, it is not the mechanism.

More information does not always help. Each track hands the agent more of the answer. Renders help: adding four pictures of the finished original and nothing else (T2) lifts the baseline from 44% valid to 67%. The bag of parts does not. T3 gives strictly more information than T2, and both agents drop —to 22% and 89%—because the bag is a constraint as much as a hint: the build may use only those pieces, and submissions shrink from 130 and 106 parts to 75 and 77. Handing over a half-built model (T4) is where the library arm satisfies the most requirements, 81%, and where it finishes whole sets, but also where its validity is lowest, for a reason that is not its doing.

Two failures are ours, not the agents'. On T3 both agents submitted nothing for one set: the bag is derived from the original's own parts, but the generator emitted it without the requirements that make the task answerable, leaving an inventory and nothing to satisfy. Separately, two of the nine original sets fail our absolute geometry or connectivity checks as LEGO shipped them. One drives two mated parts $2{,}080$ LDU^3 into each other and the other leaves five parts floating free. A T4 submission that keeps those parts inherits the defect, so it is marked invalid while still satisfying every requirement. The *sound* requirement asks for no *new* breakage, and there is none. We report both rather than relaxing the threshold, and leave the rows in.

5 Discussion and Limitations

Simulating interconnected motion. The joints we recover are measurable one at a time, but their motions interact, and the recovered bodies, joints and limits are enough to export a model to a physics simulator and drive it. We have done this for a 3:1 gear train in MuJoCo; a raw LDraw file, being only geometry, cannot be asked the question at all.

Limitations. We ran few experiments, because each costs money, and we evaluate one model under one harness, so we cannot say how the gap varies across model families. Motion is verified one joint at a time against a static rest of the model, so a mechanism whose parts must move together is reported joint by joint. And we grade what a build *is* and what it *does*, not whether it looks like the original.

AI use statement

All ideas in this work are the authors' own. The system was developed jointly with an AI coding assistant, which was used to implement and test parts of the decompiler, the benchmark harness and

the evaluation code, to help locate related work, and to polish prose. It was not used to generate the paper's claims. Every reference was checked by the authors against the primary source, and every empirical statement in the paper is backed by an artifact in the released repository. The agents under evaluation are, of course, AI systems; their outputs are the object of study rather than a contribution of the paper.

Ethics statement

This work involves no human subjects and no personal data. Our corpus is the Official Model Repository, a community-maintained collection of LDraw files transcribing LEGO sets that were manufactured and sold; the transcriptions are released by their authors for free use, and we redistribute only the derived specifications and measurements rather than the part geometry itself. LEGO is a trademark of the LEGO Group, which is not affiliated with this work and does not endorse it. The systems we evaluate are Anthropic's Claude Opus, accessed through its published API under its terms of use, driven by the vendor's own command-line agent as shipped; we report the version of both. Every run executed in a directory outside our repository and we audited afterwards that nothing outside it was read, so no third-party code or data was touched by the agents under test. The benchmark measures whether a machine-produced design is physically buildable, which we take to be a safety-relevant property rather than a hazardous capability: a grader that catches a part driven through another is useful precisely because a renderer does not.

Reproducibility statement

Every number here is computed from a stored artifact. The decompiler, grader and harness are one repository; the sample is drawn by a seeded script; the specifications are generated files; and each of the 72 runs keeps its prompt, its granted tools, its full event stream, its `submission.ldr` and the grader's verdict. One script regenerates every table and one every figure. Section 3.2 gives the thresholds and Appendix B reproduces a run end to end. The code is withheld only for anonymity and will be released with the run archive at camera-ready.

References

Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W. Taylor, Jaesik Park, and Minsu Cho. Brick-by-brick: Combinatorial construction with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2110.15481.

Yurui Dong, Shu Zou, Siqi Li, Nianchen Deng, Hongbin Zhou, Xuemeng Yang, Pinlong Cai, Licheng Wen, Xinyu Cai, and Botian Shi. ASSEMCAD: Production-ready CAD assembly generation from natural language. *arXiv preprint arXiv:2607.05123*, 2026. Introduces AssemBench.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. Spotlight. arXiv:2409.17106.

Peter Kulits and Cordelia Schmid. Bricknet: Graph-backed generative brick assembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. arXiv:2604.22984.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. CAD-Llama: Leveraging large language models for computer-aided design parametric 3D model generation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573, 2025. arXiv:2505.04481.

Ruixuan Liu, Kangle Deng, Ziwei Wang, and Changliu Liu. Stablelego: Stability analysis of block stacking assembly. *IEEE Robotics and Automation Letters*, 9(11):9383–9390, 2024. arXiv:2402.10711.

Sheng-Jie Luo, Yonghao Yue, Chun-Kai Huang, Yu-Huan Chung, Sei Imai, Tomoyuki Nishita, and Bing-Yu Chen. Legolization: Optimizing lego designs. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 34(6), 2015.

Zhengyang Ni, Feng Yan, Yu Guo, and Fei Wang. Brickanything: Geometry-conditioned buildable brick generation with structure-aware tokenization. *arXiv preprint arXiv:2605.26182*, 2026.

Ava Pun, Kangle Deng, Ruixuan Liu, Deva Ramanan, Changliu Liu, and Jun-Yan Zhu. Generating physically stable and buildable brick structures from text. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 14798–14809, 2025. arXiv:2505.05469; released as LegoGPT, renamed BrickGPT.

Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. ArtiCAD: Articulated CAD assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

Aaron Walsman, Muru Zhang, Klemen Kotar, Karthik Desingh, Ali Farhadi, and Dieter Fox. Break and make: Interactive structural understanding using lego bricks. In *European Conference on Computer Vision (ECCV)*, 2022. arXiv:2207.13738; LTRON simulator.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. CAD-GPT: Synthesising CAD construction sequence with spatial reasoning-enhanced multimodal LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7880–7888, 2025. arXiv:2412.19663.

Karl D. D. Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Joinable: Learning bottom-up assembly of parametric cad joints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. arXiv:2111.12772; Fusion 360 Gallery assembly dataset.

Tian Xia, Tianrun Gao, Wenhao Deng, Long Wei, Xiaowei Qian, Chenglei Yu, and Tailin Wu. Buildarena: A physics-aligned interactive benchmark of llms for engineering construction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. arXiv:2510.16559.

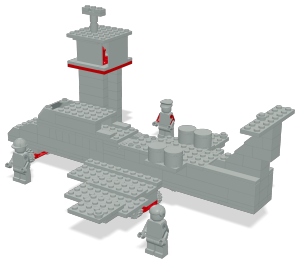
Chenhui Xu, Ziyue Bai, Fuxun Yu, Heng Huang, and Jinjun Xiong. Rollback-free stable brick structures generation. *arXiv preprint arXiv:2605.06947*, 2026.

Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. Computational lego technic design. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 38(6), 2019. arXiv:2007.02245.

Yuhuan Yuan, Zhouliang Yu, Minghao Liu, Weiyang Liu, and Ge Lin Kan. Sample-efficient post-training for lego spatial-physics reasoning. *arXiv preprint arXiv:2606.07602*, 2026.

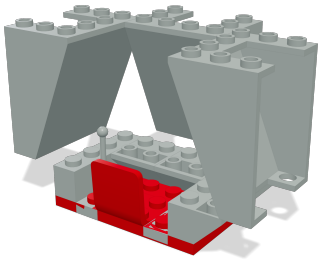
Wenqian Zhang, Yangyi Huang, Weiyang Liu, and Zhen Liu. Compositional machine design as program synthesis with llms. In *Proceedings of EMNLP (Main Conference)*, 2026. arXiv:2510.14980; BesiegeField benchmark.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. FlexCAD: Unified and versatile controllable CAD generation with fine-tuned large language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2411.05823.



(a) T1, no tools — 224 parts, 1/7.

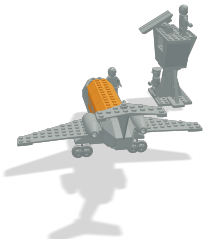
```
15 part(s) in 14 piece(s) float free
3× 4624 is 12.0 LDU along the axis
from the pin it should sit on
```



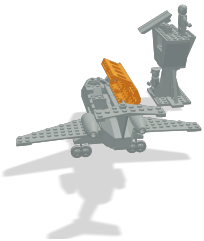
(b) T4, no tools — 162 parts, 2/9.

```
#110 3032 interpenetrates #116 4079
(4045 LDU³ deep, 8446 shared)
```

the seat is inside the plate



(c) shut, 0°



(d) open, 115°

```
% (c,d) T4, with the library
% 152 parts, valid, 8/9
joint J217:  6 parts move
arc 0 to 115 deg
stops when 4857 hits 4868a
(260 LDU³, tol 60)
[ok] hinged-part-1:  2
horizontal-axis
    bearing(s) swinging ≥ 90 deg
```

Figure 5: One set, 1775-1, three outcomes. Without our library the agent produces models that read correctly and are not assemblies: (a) fifteen parts attached to nothing and three wheel rims sitting 12 LDU short of their pins, (b) a minifigure seat driven 4,045 LDU^3 into the plate beneath it. Red is what the checker named; everything else is ghosted. With the library, the same agent on the same set builds a valid model whose cargo door is a real bearing: (c,d) show it shut and at the far end of its *measured* arc, 115°, against a requirement of 90°.

A Failure and success, on one set

Figure 5 shows both kinds on one set, the airport 1775-1. The T1 submission places 224 parts into something with wings, a tower and three minifigures. Fifteen of those parts are attached to nothing at all, and three wheel rims sit exactly 12 LDU short of the pins they were meant to go on. That distance is what the checker reports, not an estimate. The T4 submission fails the other way, with a minifigure seat driven 4,045 LDU^3 into the plate below it. Both files render as plausible models. This is why a renderer is not a checker. The agent looked at its own work and saw a finished aeroplane.

On the same set the agent with the library produces a valid 152-part model that satisfies eight of its nine requirements, missing only the footprint band by 0.2 studs. Its cargo door is a real bearing rather than a door-shaped hole. Our grader sweeps it from shut to 115°, where the door catches on the fuselage, against a requirement of 90° (Figure 5c,d). That is the comparison the benchmark exists to make. Same agent, same model, same task, and the difference between a picture of an aeroplane and an aeroplane.

B A COMPLETE AGENT TRACE

One run, end to end, exactly as the harness recorded it. Absolute paths are replaced by `$SANDBOX`; where the agent opened a rendered image, the base64 payload is replaced by a note; long outputs are cut with the number of lines dropped. Nothing else is altered. The full trace, with every byte of output and the submitted model, is in the released repository under this run's identifier.

Two verdicts in it are worth reading carefully, because the grader's wording is terse. STABILITY passes while reporting that an eleven-part sub-assembly tips: the rung fails a build whose *whole* centre of mass leaves its support hull, and reports a detached display piece that would topple on its own as a minor finding rather than a failure of the model. Likewise MECHANISM passes while naming a bearing that cannot turn: a joint frozen by its neighbours is a fact about the design, not a violation, and only becomes a failure when a requirement asks that joint to move. Parts with no connector or voxel data are excluded from the rungs that would need them and reported as unverifiable; they are never counted as having passed.

task	`bd-1773-1-T4` (track T4: finish a half-built model)
agent	Brick Dance Copilot (`parts-check`), claude-code 2.1.280 (Claude Code)
model	`claude-opus-5-5`
caps	120 turns, $2, 20 minutes
spent	$1.20, 21 turns, 27935 output tokens
tools granted	`Task,Bash,Edit,Glob,Grep,Monitor,NotebookEdit,Read,TaskStop,Write`
escaped sandbox	false

Turn 1 — `Bash`.

```
cd $SANDBOX; ls; cp seed.mpd submission.ldr; ./bd check seed.mpd | tail -30; for p in 4600 4084
 4624 3641 3795 3962b 2508 4485 4592c01 3815; do grep -m1 "^0 " ldraw/parts/$p.dat; done
```

```
PARTS.txt
TASK.md
TOOLS.md
bd
front.png
ldraw
rear.png
seed.mpd
side.png
src
three-quarter.png
OK   parse
  ! 3 part(s) have no connector data -- their mates cannot be checked: 4862, 4084
  ! 4 part(s) have no voxel model -- collisions involving them are unchecked
  * 55 parts, 30 distinct
OK   geometry
  * 0 interpenetrations, 0 marginal overlaps, 11 contact-skin pairs below the noise floor
OK   connectivity
  ! 3 part(s) in 3 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attachm
ent unverifiable
  ! no part rests on the ground plane (y = 0)
  * 68 mated pairs, 4 piece(s), 0 grounded part(s)
... (13 more lines)
```

Turn 2 — `Read`.

```
$SANDBOX
```

```
[an image was returned to the agent]
```

Turn 3 — `Read`.

```
$SANDBOX
```

```
[an image was returned to the agent]
```

Turn 4 — Bash.

```
cd $SANDBOX; cat PARTS.txt | head -70; python3 - <<'E'
import re,collections
inv=collections.Counter()
for l in open('PARTS.txt'):
    m=re.match(r'\s*(\S+)\s+(\d+)\s+x(\d+)',l)
    if m: inv[(m[1],m[2])]+=int(m[3])
for l in open('seed.mpd'):
    t=l.split()
... (4 more lines)

# the 111 pieces the original was built from -- part, colour, how many
3024 4 x8
4624 15 x8
4864a 15 x6
3070b 36 x4
3641 0 x4
3788 4 x4
4070 15 x4
4084 0 x4
4600 0 x4
3710 4 x3
4315 15 x3
2357 15 x2
2377 15 x2
2412b 7 x2
3022 0 x2
3023 4 x2
3024 47 x2
3062b 46 x2
3068b 4 x2
3070b 15 x2
3070bp08 14 x2
... (63 more lines)
```

Turn 5 — Bash.

```
cd $SANDBOX; ./bd connectors 4600 --all | head -40; ./bd connectors 4624 --all | head -30; ./d
bd connectors 2508 --all; ./bd connectors 3795 | head -30

C1    stud socket    cyl    F    pos -10,8,-10    axis 0,1,0    opens down (+y)    secs "R 6 4"
caps=one
C2    stud socket    cyl    F    pos -10,8,10    axis 0,1,0    opens down (+y)    secs "R 6 4" c
aps=one
C3    stud socket    cyl    F    pos 10,8,-10    axis 0,1,0    opens down (+y)    secs "R 6 4" c
aps=one
C4    stud socket    cyl    F    pos 10,8,10    axis 0,1,0    opens down (+y)    secs "R 6 4" ca
ps=one
C5    tube (bore)    cyl    F    pos 0,8,0    axis 0,1,0    opens down (+y)    secs "R 6 4" caps
=one
C6    bar/pin shaft (r4, 11 LDU)    cyl    M    pos 22,5,0    axis -1,0,0    points +x    secs "
R 4 11   _L 4.25 1" caps=one
C7    bar/pin shaft (r4, 11 LDU)    cyl    M    pos -22,5,0    axis 1,0,0    points -x    secs "
R 4 11   _L 4.25 1" caps=one
C8    stud    cyl    M    pos -10,0,10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C9    stud    cyl    M    pos -10,0,-10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C10    stud    cyl    M    pos 10,0,10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C11    stud    cyl    M    pos 10,0,-10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C1    bar/pin socket (r4, blind)    cyl    F    pos 0,0,8    axis 0,0,1    opens +z    secs "R 4
 10   R 5 2" caps=one
C2    generic mount (wheel/tyre or ball joint), male    gen    M    pos 0,0,0    axis 0,0,1    a
xis along z    secs "-"
... (34 more lines)
```

Turn 6 — Bash.

```
cd $SANDBOX; head -1 ldraw/parts/3730.dat; ./bd connectors 3730 --all; head -1 ldraw/parts/2508
.dat; ./bd connectors 4084 --all; ./bd connectors 3641 --all

0 Plate  2 x  2 with Towball Socket
C1    stud socket    cyl    F    pos -10,8,-10    axis 0,1,0    opens down (+y)    secs "R 6 4"
caps=one
C2    stud socket    cyl    F    pos -10,8,10    axis 0,1,0    opens down (+y)    secs "R 6 4" c
```

```
aps=one
C3    stud socket    cyl    F    pos 10,8,-10    axis 0,1,0    opens down (+y)    secs "R 6 4" c
aps=one
C4    stud socket    cyl    F    pos 10,8,10    axis 0,1,0    opens down (+y)    secs "R 6 4" ca
ps=one
C5    unknown (female cyl, profile "S 16 4")    cyl    F    pos 0,8,0    axis 0,1,0    opens dow
n (+y)    secs "S 16 4" caps=one
C6    tube (bore)    cyl    F    pos 0,8,0    axis 0,1,0    opens down (+y)    secs "R 6 4" caps
=one
C7    stud    cyl    M    pos 10,0,10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C8    stud    cyl    M    pos -10,0,10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C9    stud    cyl    M    pos 10,0,-10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=one
C10    stud    cyl    M    pos -10,0,-10    axis 0,1,0    points up (-y)    secs "R 6 4" caps=on
e
C11    generic mount (wheel/tyre or ball joint), female    gen    F    pos 0,4,-30    axis 0,1,0
    axis along y    secs "-"
0 Plate  1 x  2 with 3L Extension and Towball
no connector data for 4084
... (2 more lines)
```

Turn 7 — Bash.

```
cd $SANDBOX; for p in 3641 4084 4624 2508 3031 2357 4864a 3010 3788 4213 4315 2412b 3070bp08 371
0; do echo "== $p $(head -1 ldraw/parts/$p.dat)"; ./bd part $p | sed -n '2,3p'; done

== 3641 0 Tyre  6/ 50 x  8 Offset Tread
size 2 x 1 studs * 40 x 20 x 18 LDU (x by z by height) * off the stud grid
origin: centre of the TOP face. +y is DOWN, so the body runs y 0 -> +18. Positions below are par
t-local LDU.
== 4084 0 Tyre  8/ 75 x  8 Offset Tread

SHAPE / PLACEMENT NOTES
== 4624 0 Wheel Rim  6.4 x  8
size 1 x 1 studs * 20 x 20 x 10 LDU (x by z by height) * off the stud grid
origin: centre of the TOP face. +y is DOWN, so the body runs y 0 -> +10. Positions below are par
t-local LDU.
== 2508 0 Plate  1 x  2 with 3L Extension and Towball
size 2 x 5 studs * 40 x 100 x 12 LDU (x by z by height) * off the stud grid
footprint centre sits 0,-44 LDU from the origin (x,z)
== 3031 0 Plate  4 x  4
size 4 x 4 studs * 80 x 80 x 8 LDU (x by z by height)
origin: centre of the TOP face. +y is DOWN, so the body runs y 0 -> +8 and studs rise to y -4. P
ositions below are part-local LDU.
== 2357 0 Brick  2 x  2 Corner
size 2 x 2 studs * 40 x 40 x 24 LDU (x by z by height)
footprint centre sits 10,10 LDU from the origin (x,z)
== 4864a 0 Panel  1 x  2 x  2 with Solid Studs
... (26 more lines)
```

Turn 8 — Bash.

```
cd $SANDBOX; mkdir -p src; cat > src/bbox.py <<'E'
import sys,os
L='ldraw'
def find(n):
    n=n.lower().replace('\\','/')
    for d in ['parts','p','parts/s']:
        f=os.path.join(L,d,n)
        if os.path.exists(f): return f
... (24 more lines)

3641 [(-18.0, 18.0), (-18.0, 18.0), (-8.0, 8.0)]
4084 [(-25.0, 25.0), (-25.0, 25.0), (-12.0, 8.0)]
4624 [(-10.0, 10.0), (-10.0, 10.0), (-8.0, 8.0)]
2508 [(-20.0, 20.0), (-4.0, 12.0), (-98.0, 10.0)]
3962b [(-24.0, 4.0), (-55.0, 18.0), (-8.0, 4.0)]
3730 [(-20.0, 20.0), (-7.8, 14.8), (-37.5, 20.0)]
4600 [(-34.0, 34.0), (-4.0, 10.0), (-20.0, 20.0)]
3795 [(-60.0, 60.0), (-4.0, 8.0), (-20.0, 20.0)]
```

Turn 9 — Read.

```
$SANDBOX

[an image was returned to the agent]
```

Turn 10 — Bash.

```
cd $SANDBOX; for p in 4213 4070 2357 4864a 3788; do echo "== $p"; ./bd connectors $p | sed -n '
/MAPS/,$p' | head -40; done; python3 src/bbox.py 4213 4070 2357 4864a 3788

== 4213
MAPS  (cell = symbol + C-number; * = nothing)
  O stud   o stud socket   ? tube

  TOP  (x -> right, z ? * 1 col = 10 LDU (? stud))
       -30 -20 -10   0  10  20  30
  z-30  O23   * O22   * O21   * O20
  z-20    *   *   *   *   *   *   *
  z-10  O19   * O18   * O17   * O16
  z0      *   *   *   *   *   *   *
  z10   O15   * O14   * O13   * O12

  BOTTOM  (x -> right, z ? * 1 col = 10 LDU (? stud))
      -30 -20 -10   0  10  20  30
  z-30   o1   *  o3   *  o5   *  o7
  z-20    * ?10   *   *   * ?11   *
  z-10   o2   *  o4   *  o6   *  o8

NOT DRAWN ON THE MAPS (no symbol of their own: bars, clips, hinges, balls, angled features):
  C9 hinge finger at 0,4,36 * axis along x

`./bd connectors <part> --all` prints the flat per-connector table.
... (71 more lines)
```

Turn 11 — Bash.

```
cd $SANDBOX; for p in 973p26 3815 3816 3818 3819 3820 3962b 3626bp01 4485 4070; do echo "== $p";
 ./bd connectors $p --all; done 2>&1 | cut -c1-150

== 973p26
C1    unknown (female cyl, profile "R 5 2   R 6 5")    cyl    F    pos -15.6,9,0    axis -0.99,-
0.17,0    opens -x    secs "R 5 2   R 6 5" caps=one
C2    unknown (female cyl, profile "R 5 2   R 6 5")    cyl    F    pos 15.6,9,0    axis 0.99,-0.
17,0    opens +x    secs "R 5 2   R 6 5" caps=one
C3    pin (12 LDU)    cyl    M    pos 0,0,0    axis 0,1,0    points up (-y)    secs "R 6 12" cap
s=one
C4    stud socket (deep well)    cyl    F    pos -10,32,0    axis 0,1,0    opens down (+y)    se
cs "R 6 11" caps=one
C5    stud socket (deep well)    cyl    F    pos 10,32,0    axis 0,1,0    opens down (+y)    sec
s "R 6 11" caps=one
== 3815
C1    pin (11 LDU)    cyl    M    pos -10,0,0    axis 0,1,0    points up (-y)    secs "R 6 11" c
aps=one
C2    pin (11 LDU)    cyl    M    pos 10,0,0    axis 0,1,0    points up (-y)    secs "R 6 11" ca
ps=one
C3    bar/pin shaft (r3, 8 LDU)    cyl    M    pos -2,12,0    axis 1,0,0    points -x    secs "R
 3 8" caps=one
C4    bar/pin shaft (r3, 8 LDU)    cyl    M    pos 2,12,0    axis -1,0,0    points +x    secs "R
 3 8" caps=one
== 3816
C1    bar/pin socket (r3, blind)    cyl    F    pos -2,0,0    axis 1,0,0    opens +x    secs "R
... (34 more lines)
```

Turn 12 — Bash.

```
cd $SANDBOX; cat > src/gen.py <<'E'
import numpy as np
seed=[l.rstrip('\n') for l in open('seed.mpd') if l.startswith('1 ')]
out=[]
def f(v):
    v=round(float(v),3); return str(int(v)) if v==int(v) else str(v)
def P(c,pos,R,part):
    R=np.array(R,dtype=float).reshape(3,3)
... (53 more lines)

Exit code 1
105
OK   parse
  ! 6 part(s) have no connector data -- their mates cannot be checked: 4862, 4084
```

```
  ! 10 part(s) have no voxel model -- collisions involving them are unchecked
  * 105 parts, 47 distinct
OK   geometry
  ! #87 4213 interpenetrates #91 2412b (76 LDU^3 deep, 336 shared)
  * 0 interpenetrations, 1 marginal overlaps, 13 contact-skin pairs below the noise floor
FAIL connectivity
  [x] 1 part(s) touch the main assembly with 2 connector pair(s) that almost mate but do not: #9
1 2412b cyl is 12.8 LDU (8.0 along the axis, 10.0 across) and 0.0 deg off #85 4864a
  [x] 1 part(s) touch the main assembly with 1 connector pair(s) that almost mate but do not: #9
2 3070b cyl is 8.0 LDU (8.0 along the axis, 0.0 across) and 0.0 deg off #85 4864a
  [x] 1 part(s) touch the main assembly with 1 connector pair(s) that almost mate but do not: #9
3 3070b cyl is 8.0 LDU (8.0 along the axis, 0.0 across) and 0.0 deg off #86 4864a
  ! 1 separate object(s) stand on the ground apart from the main assembly (11 parts)
  ! 6 part(s) in 6 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attachm
ent unverifiable
  * 117 mated pairs, 11 piece(s), 2 grounded part(s)

3 connection problem(s). Part numbers are indices into your file's type-1 lines, counting from 0
... (1 more lines)
```

Turn 13 — Bash.

```
cd $SANDBOX; python3 - <<'E'
s=open('src/gen.py').read()
s=s.replace("P(15,(-40,-112,240),I,'4213')\nP(15,(-40,-112,190),I,'3710')","P(15,(-40,-112,220),
I,'4213')\nP(15,(-40,-112,270),I,'3710')")
s=s.replace("P(7,(-40,-120,260),I,'2412b')","P(7,(-40,-120,270),I,'2412b')")
open('src/gen.py','w').write(s)
E
python3 src/gen.py && ./bd check submission.ldr

105
OK   parse
  ! 6 part(s) have no connector data -- their mates cannot be checked: 4862, 4084
  ! 10 part(s) have no voxel model -- collisions involving them are unchecked
  * 105 parts, 47 distinct
OK   geometry
  * 0 interpenetrations, 0 marginal overlaps, 13 contact-skin pairs below the noise floor
OK   connectivity
  ! 1 separate object(s) stand on the ground apart from the main assembly (11 parts)
  ! 6 part(s) in 6 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attachm
ent unverifiable
  * 120 mated pairs, 8 piece(s), 2 grounded part(s)

Every part is a real part, nothing is inside anything else, and everything is joined. (This does
 not check that it looks right, that it stands up, or that anything moves.)
```

Turn 14 — Bash.

```
cd $SANDBOX; for x in -88 -86 -84; do ./bd fit 3070bp08 $x -94 210 0 1 0 0 0 1 1 0 0 | head -5;
 done

trying 3070bp08 at (-88, -94, 210) as index 105:
  it fits: no interpenetration, and it is joined to what is already there.
  drew fit.png: the part you are trying is MAGENTA, with everything within 140 LDU of it. View i
t with the Read tool.
(nothing was written -- this was a trial)
trying 3070bp08 at (-86, -94, 210) as index 105:
  it fits: no interpenetration, and it is joined to what is already there.
  drew fit.png: the part you are trying is MAGENTA, with everything within 140 LDU of it. View i
t with the Read tool.
(nothing was written -- this was a trial)
trying 3070bp08 at (-84, -94, 210) as index 105:
  it fits: no interpenetration, and it is joined to what is already there.
  drew fit.png: the part you are trying is MAGENTA, with everything within 140 LDU of it. View i
t with the Read tool.
(nothing was written -- this was a trial)
```

Turn 15 — Bash.

```
cd $SANDBOX; python3 - <<'E'
s=open('src/gen.py').read()
s=s.replace("# roof","P(14,(-88,-94,210),[[0,1,0],[0,0,1],[1,0,0]],'3070bp08')\nP(14,(8,-94,210)
```

```
,[[0,-1,0],[0,0,-1],[1,0,0]],'3070bp08')\n# roof")
open('src/gen.py','w').write(s)
E
python3 src/gen.py && ./bd check submission.ldr | head -12; ./bd render --lat 25 --lon 50 --ou
t r1.png; ./bd render --lat 10 --lon 90 --out r2.png

107
OK   parse
  ! 6 part(s) have no connector data -- their mates cannot be checked: 4862, 4084
  ! 10 part(s) have no voxel model -- collisions involving them are unchecked
  * 107 parts, 48 distinct
OK   geometry
  * 0 interpenetrations, 0 marginal overlaps, 13 contact-skin pairs below the noise floor
OK   connectivity
  ! 1 separate object(s) stand on the ground apart from the main assembly (11 parts)
  ! 6 part(s) in 6 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attachm
ent unverifiable
  * 122 mated pairs, 8 piece(s), 2 grounded part(s)

Every part is a real part, nothing is inside anything else, and everything is joined. (This does
 not check that it looks right, that it stands up, or that anything moves.)
wrote r1.png (camera lat 25 deg, lon 50 deg). View it with the Read tool.
wrote r2.png (camera lat 10 deg, lon 90 deg). View it with the Read tool.
```

Turn 16 — Read.

```
$SANDBOX

[an image was returned to the agent]
```

Turn 17 — Read.

```
$SANDBOX

[an image was returned to the agent]
```

Turn 18 — Bash.

```
cd $SANDBOX; grep -v "^0" ldraw/parts/4864a.dat | head -30

1 16 0 0 0 1 0 0 0 1 0 0 0 1 s\4864s01.dat

4 16 20 0 10 20 48 10 -20 48 10 -20 0 10
1 16 10 0 0 1 0 0 0 1 0 0 0 1 stud.dat
1 16 -10 0 0 1 0 0 0 1 0 0 0 1 stud.dat
```

Turn 19 — Bash.

```
cd $SANDBOX; sed -i '' "s/P(15,(-70,-104,240),L,'4864a'); P(15,(-10,-104,240),Rt,'4864a')/P(15,(
-70,-104,240),Rt,'4864a'); P(15,(-10,-104,240),L,'4864a')/" src/gen.py; python3 src/gen.py && ./
bd check submission.ldr | sed -n '5,9p'; ./bd render --lat 20 --lon 230 --out r3.png

107
OK   geometry
  * 0 interpenetrations, 0 marginal overlaps, 13 contact-skin pairs below the noise floor
OK   connectivity
  ! 1 separate object(s) stand on the ground apart from the main assembly (11 parts)
  ! 6 part(s) in 6 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attachm
ent unverifiable
wrote r3.png (camera lat 20 deg, lon 230 deg). View it with the Read tool.
```

Turn 20 — Read.

```
$SANDBOX

[an image was returned to the agent]
```

What the agent said when it stopped.

```
I've finished the build and written the whole model to `./submission.ldr`: 107 parts, the 55 ori
ginal parts unchanged plus 52 new ones. The connection check passes: every part exists in the li
brary, no parts overlap, and everything is joined. The file has 2 separate objects, the truck-an
d-trailer and the minifig. I can't test here whether it stands up, whether the wheels really tur
n a full circle, or how the reviewer will judge its looks.

**What I added:**
- **Truck:** four white wheels, plus the three large black tyres that were missing.
- **Trailer:**
  - A black 2x6 frame hooked to the truck by the black drawbar, whose towball sits in the truck'
s towball socket.
  - Two axles 2 studs apart, each on a wheel-pin plate under a red mudguard, carrying four white
 wheels with small black tyres. These four wheels are the moving parts the spec asks for. None o
f them touches anything, so they should turn freely, but I couldn't spin them to confirm.
  - A white box on a white 4x4 plate, with a red strip along the front edge. It has corner brick
s at the front, smooth panels on the sides and back, and a 1x4 brick.
  - A yellow tile on each side, and a roof with two red tiles, a gray vent and two transparent-r
ed rear corner lights.
- **Minifigure:** all blue with a blue cap, holding a black walkie-talkie, standing beside the t
ruck.

**Differences from the original you should know about:**
- **Trailer shape:** the box is 4 studs wide and 5 long, not the roughly 6-long box in the refer
ence pictures. The back is plain panels rather than the studs pointing backward, and the red lig
hts sit on the roof rather than on the back face.
- **Opening:** there is a small gap at the trailer's rear side corner, where a panel's open edge
 shows.
- **Ground contact:** the tyres don't quite touch the table. The truck's sit 3 units above it an
d the trailer's 1 unit, so the whole thing will likely settle and tilt very slightly toward the
truck.
- **Unused parts:** a red 1x4 plate, two red 1x1 plates and the hinge plate with roof holder.

**Size:** it measures about 11 x 22 studs and about 160 units tall, within the allowed ranges.

The file is generated by `src/gen.py`.
```

The grader's verdict on the submitted file. The agent never saw this.

```
PASS  parse
      6 part(s) have no connector data -- their mates cannot be checked: 4862, 4084
      10 part(s) have no voxel model -- collisions involving them are unchecked
      107 parts, 48 distinct
PASS  geometry
      0 interpenetrations, 0 marginal overlaps, 13 contact-skin pairs below the noise floor
PASS  connectivity
      1 separate object(s) stand on the ground apart from the main assembly (11 parts)
      6 part(s) in 6 piece(s) have no connector/voxel data (tyres, chains, stickers...) -- attac
hment unverifiable
      122 mated pairs, 8 piece(s), 2 grounded part(s)
PASS  stability
      structure of 11 part(s), 3.7 g: CoM is 1.4 LDU outside its ground footprint -- it tips ove
r (minor)
      mass 78 g, CoM margin -1.4 LDU inside the support hull
PASS  mechanism
      31 rigid bodies (largest 48 parts), 12 movable bearings, 0 pinned
      bearing J155 (#7 3730 <-> #62 2508) cannot turn at all -- blocked by neighbouring parts
      12 bearings swept: free arcs 0-360 deg
SKIP  actuation
      no gear trains to actuate
PASS  task
      [ok] parts 107 >= 77
      [ok] parts 107 <= 145
      [ok] height 160 >= 136 LDU
```