CAN AI SPEAK THE LANGUAGE OF ORIGAMI?

Anonymous authors
Paper under double-blind review

ABSTRACT

Building AI systems that can plan, act, and create in the physical world requires more than pattern recognition. Such systems must reason about the generative mechanisms and constraints that govern physical processes, using structured representations that connect observations, actions, and their effects. Yet existing benchmarks often study these capabilities in isolation, focusing either on visual recognition or on abstract symbolic and programmatic reasoning. Origami offers a natural testbed for integrating these abilities: constructing shapes through sequences of folds requires visual perception, reasoning about geometric and physical constraints, and multi-step planning, while remaining sufficiently structured for systematic evaluation. We introduce *OrigamiBench*, a benchmark for evaluating programmatic understanding of the mechanisms underlying origami synthesis and the ability to induce such mechanisms in a high-level language of physically grounded fold actions. Experiments with modern vision-language models show that increasing model scale alone does not reliably improve reasoning about physical transformations. Moreover, models struggle to ground programmatic representations in visual observations, indicating that their visual and linguistic representations remain only weakly integrated.

1 INTRODUCTION

Building AI systems that can plan, act, and create in the physical world requires more than pattern recognition. Such systems must reason about the generative mechanisms and constraints governing physical processes in order to make sequential decisions. This requires structured representations that connect observations, actions, and their effects on the environment, while capturing how complex outcomes arise from sequences of simpler transformations. Yet, many existing benchmarks study these capabilities separately, focusing either on recognizing visual states or on reasoning over abstract symbolic or programmatic representations. This raises a fundamental question: *can models learn to reason about how physical outcomes are generated through structured representations that connect observations, actions, and their effects?* Origami provides a natural setting in which perception, physical validity, and symbolic composition can be studied jointly. A sheet of paper is transformed through a sequence of folds constrained by geometry and foldability (Akitaya et al., 2013). Despite relying on a single primitive action, repeated folding generates a rich space of designs through an inherently compositional and recursive synthesis process (Gibbons, 2003). Existing computational approaches, however, predominantly represent origami through crease patterns (Lang, 1996; Tachi, 2010; 2009; Akitaya et al., 2013; Zahavy et al., 2026; Huang et al., 2026), which describe the geometry of a design but do not explicitly encode the ordered composition of folds that produces it. They therefore make the *outcome* of synthesis explicit while leaving the underlying *sequential generative mechanism* implicit.

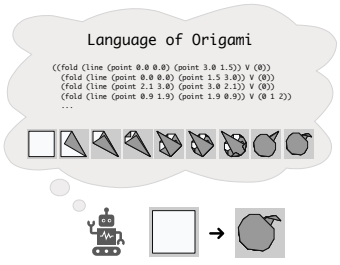

Figure 1: Visual Induction Problem: AI must induce the underlying generative process from visual perceptual data.

In this work, we make the synthesis process itself the object of representation and reasoning. To this end, we introduce *OrigamiBench*, a benchmark for quantifying how well modern vision-language foundation models (VLMs) understand the mechanisms underlying origami synthesis and induce

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.

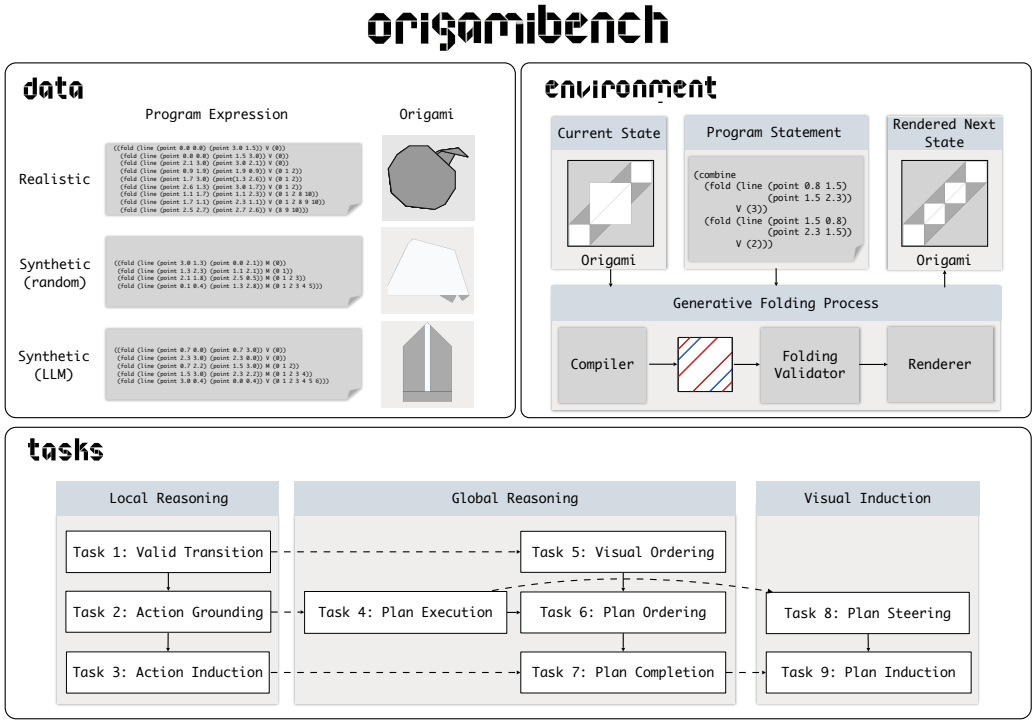

Figure 2: Visual summary of *OrigamiBench*. On the left (**Data**), representative realistic, random and LLM-generated examples are shown as program expressions paired with their rendered origami states. On the top right (**Environment**), a program statement is executed from the current state through the generative folding process: the compiler converts the statement into a crease-pattern representation, the folding validator checks the resulting state, and the renderer produces the rendered next state. At the bottom (**Tasks**), a curriculum of tasks and their dependence relationships are shown. The curriculum is designed to interpolate between single step (local) reasoning to full visual induction.

the underlying programmatic descriptions, as shown in Fig. 1. At its core is a high-level language in which origami designs are expressed as compositions of physically grounded fold actions, bridging visual states with the sequential transformations that generate them. The language abstracts away low-level geometric implementation of crease patterns while preserving the structure required to reason about ordering, composition, and physical validity. This enables us to precisely annotate folding trajectories, generate physically valid synthesis problems with controllable compositional complexity, and construct evaluation tasks that probe whether models can ground symbolic transformations in visual observations. Experiments on *OrigamiBench* reveal three key findings: scaling model size alone does not reliably improve reasoning about physical transformations; models struggle to ground structured programmatic information in visual observations; and this difficulty is particularly pronounced for fine-grained program elements such as fold points, fold types, and paper layers. Overall, these results suggest that visual and language representations remain weakly integrated.

2 ORIGAMIBENCH

2.1 ORIGAMI ACTION LANGUAGE

We represent an origami design as a program consisting of a sequence of folding actions applied to a paper state. Unlike geometric representations such as FOLD (Demaine et al., 2016), which encode crease patterns and folded states, our language specifies the transformations that produce them. Programs consist of atomic folds, compound folds, paper turns, and reusable folding primitives, and can be compiled into FOLD graphs for foldability checking and rendering. The syntax is defined in Figure 3.

```
<program>   ::= (<statement>*)
<statement> ::= <fold> | <combine> | <library> | (turn h) | (turn v)
<fold>      ::= (fold <line> <fold_type> <levels>)
<line>      ::= (line <point> <point>)
<point>     ::= (point <coord> <coord>)
<coord>     ::= [-]<number>
<number>    ::= <integer> | <digit>+.<digit>+
<integer>   ::= <digit>+
<digit>     ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<fold_type> ::= M | V
<levels>    ::= (<integer>+)
<combine>   ::= (combine <fold> <fold>+)
<library>   ::= (<signature> <args>)
<signature> ::= pleat | crimp | inrev | outrev | rabear | flatpock
<args>      ::= (<point>+ <levels>)
```

Figure 3: Grammar of the origami action language. Programs are Lisp-like expressions composed of atomic folds, compound folds, library operations, and paper turns.

A `program` is an ordered sequence of `statements` that transform the current paper state. Statements are evaluated sequentially and may be atomic `folds`, compound `combine` operations, calls to reusable `library` operations, or turns of the paper about its horizontal `(turn h)` or vertical `(turn v)` axes.

An atomic `fold` is parameterized by a line, a fold direction, and the paper layers affected by the operation. The line is specified by two 2D points, and the fold direction is either mountain (`M`) or valley (`V`). The `levels` argument identifies the indexed paper layers participating in the fold. Explicit layer selection is necessary to create detailed, complex origamis, where folds are not applied arbitrarily to the entire origami but only to specific layers to create specific shapes. Layer indices are updated after each folding operation to reflect the current layers in the origami. Importantly, an atomic action does not necessarily correspond to a single crease edge: depending on the current paper state, one fold action may induce several mountain and valley crease edges (Figure 4).

A `combine` expression groups folds that are evaluated against the same paper state. It may differ semantically from executing the folds sequentially, where each fold acts on the state produced by the previous one. This distinction is necessary for folding actions whose constituent creases must be formed together and cannot be represented as a sequence of independent folds (as the case od some libraries).

A `library` expression represents a reusable folding primitive, such as a pleat, crimp, or reverse fold. Each primitive expands deterministically into base fold and `combine` actions according to Table 3 in Appendix A. Library operations thus provide a compact hierarchical representation without extending the semantics of the base language. The number of point arguments is determined by the primitive signature (cf. Appendix A).

The `turn` statements reorient the current paper state about its horizontal (`turn h`) or vertical (`turn v`) axis before subsequent actions are applied.

Example. Figure 4 shows an eleven-statement program for constructing a whale. The first two statements are libraries, followed by a sequence of valley folds and turn operations. The complete program is shown below:[1]

```
(
 (rabear ((point 2.0 2.1) (point 0.2 2.8) (point 2.8 2.8)
             (point 2.8 0.2) (point 2.0 2.8) (0)))
 (rabear ((point 0.9 1.0) (point 0.2 2.8) (point 0.2 0.2)
             (point 2.8 0.2) (point 0.2 1.0) (0)))
 (fold (line (point 1.3 1.9) (point 1.4 2.1)) V (2 3))
 (fold (line (point 1.2 1.7) (point 0.9 1.4)) V (5 6))
```

[1]where coordinates are rounded to the first decimal for simplicity of visualization.

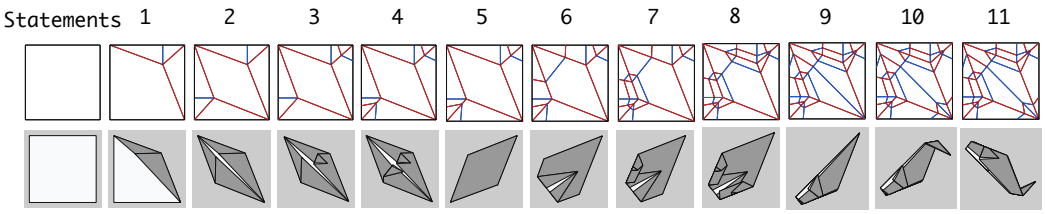

Figure 4: Execution of an origami action program that constructs a whale. Each column corresponds to one program statement. The statement is compiled into a crease graph (middle row), checked for foldability, and executed to obtain the updated rendered paper state (bottom row). Thus, statements in the language correspond directly to folding actions rather than individual crease edges. In particular, a single atomic action may induce several crease edges: statement 6, for example, produces two valley and one mountain crease.

```
  (turn v)
  (fold (line (point 1.2 0.6) (point 0.6 1.2)) V (0 3 8))
  (fold (line (point 0.3 0.8) (point 0.8 2.1)) V (7 8 9 10 11 13))
  (fold (line (point 2.0 0.8) (point 0.8 0.3)) V (2 3 4 10 11 12 13))
  (fold (line (point 0.5 0.5) (point 2.8 2.8)) V (9 10 11 13))
  (fold (line (point 2.1 2.3) (point 2.3 2.5)) V (4 7 25 26))
  (turn h)
)
```

2.2 Environment Workflow

We implement an environment that executes programs written in the origami action language introduced in Section 2.1. Starting from an unfolded sheet, the environment processes each program statement sequentially, updating the paper state accordingly. Each statement undergoes three stages: compilation, foldability checking, and rendering, as shown in Figure 2.

Compilation. The paper state is represented internally as a FOLD graph (Demaine et al., 2016). The compiler, implemented using Rabbit Ear,[2] maps each statement and its corresponding paper state to a candidate successor graph. Atomic folds are resolved with respect to the selected layers and may introduce multiple crease edges when the fold line intersects several faces or overlapping layers (Figure 4). For `combine` operations, all constituent folds are compiled against the same pre-action state. Library primitives are first expanded according to Table 3. `turn` operations update the paper orientation and layer-index mapping.

Foldability checking. The candidate FOLD graph is passed to Flat-Folder,[3] which checks local flat-foldability conditions, including Maekawa's and Kawasaki's theorems, and then searches for a globally consistent folded configuration. If no feasible configuration is found, the action is marked invalid and the state remains unchanged; otherwise, the candidate graph becomes the new state.

Rendering. For a feasible statement, the validated folded configuration is rendered to produce the updated folded-paper view and the corresponding 2D crease pattern.

2.3 Data Generation and Organization

OrigamiBench is organized as a collection of folding trajectories represented in the grammar introduced in Section 2.1. Each trajectory contains a program expression, its intermediate FOLD states, and the corresponding rendered observations, as shown in Figure 2, enabling supervision at both the program and individual state-transition levels. A key advantage of our action language is that it provides a scalable way to generate physically meaningful crease patterns while abstracting away low-level physical constraints. Rather than specifying crease geometry directly, each action transforms the current paper state, producing creases consistent with its geometry and layer configuration while preserving the intermediate states that lead to the final pattern.

[2] `https://github.com/rabbit-ear/rabbit-ear`

[3] `https://github.com/origamimagiro/flat-folder`

Table 1: Composition and program characteristics of *OrigamiBench*. Program-level statistics are minimum / mean / maximum across trajectories. Operation categories are mutually exclusive: `fold` counts standalone folds, `combine` excludes library and base operations, and `statement` includes all operations, including turnovers.

Subset	Trajectories	# `statement` per traj.	# `fold` per traj.	# `combine` per traj.	# `library` per traj.
Manually Annotated	100	3 / 7.4 / 14	0 / 5.8 / 12	0 / 0.3 / 3	0 / 0.6 / 8
LLM-generated	1,000	2 / 8.6 / 15	0 / 5.8 / 14	0 / 1.6 / 5	0 / 0.9 / 3
Randomly Generated	10,000	1 / 7.5 / 15	0 / 4.7 / 14	0 / 1.3 / 7	0 / 1.1 / 5
Total	11,100	1 / 7.6 / 15	0 / 4.8 / 14	0 / 1.4 / 7	0 / 1.1 / 8

As shown in Figure 5, randomly sampled statements are substantially more robust to increasing compositional complexity than direct crease-pattern generation, retaining an 18.6% validity rate at eight folds while the latter approaches zero; full experimental details are provided in Appendix B. Building on this representation, we generate data in three ways: through manual annotation, random program sampling, and LLM-based program generation. Table 1 summarizes the number of trajectories and program-length statistics for each subset.

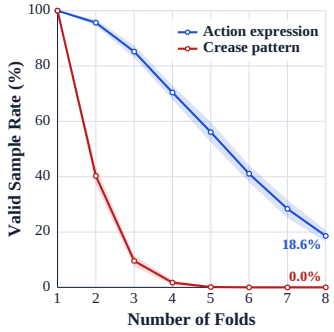

Figure 5: **Validity under increasing compositional complexity.** Valid sample rates of action expressions and crease patterns across 1 to 8 folds, evaluated on 50,000 paired random samples (2,000 per seed across 25 random seeds). Shaded regions show mean ± standard deviation across 25 random seeds.

Manually annotated data. To ground *OrigamiBench* in realistic, human-designed folding procedures, we construct the manually annotated subset from step-by-step origami instructions.[4] We build an interface around the *OrigamiBench* environment (cf. Appendix C.1) to interactively reproduce each instruction by specifying fold actions on the current paper state and observing the resulting states. For each step, we record the program expression, intermediate crease pattern, and rendered paper state (front and back views), forming an executable folding trajectory.

Randomly generated data. To broaden the coverage of the program space and introduce diverse folding trajectories, we generate programs by randomly sampling statements from our grammar. The sampled programs are executed sequentially in our environment workflow, producing both the complete program and its intermediate folding states. We select 80 valid trajectories from the random representation validity experiment in Figure 5 for inclusion in *OrigamiBench*. Further details of the sampling and validation procedure are provided in Appendix C.3.

LLM-generated data. To expand the coverage of recognizable flat-folded shapes, we use manually annotated examples as in-context demonstrations and prompt an LLM to generate new data together with natural-language instructions and corresponding program expressions. The LLM can directly interact with our environment workflow to execute its generated programs and iteratively revise invalid program expressions based on execution feedback. Through this process, we obtain 80 valid trajectories in total. Further details of the prompting and generation procedure are provided in Appendix C.2.

2.4 Evaluation Tasks

OrigamiBench introduces a suite of tasks aiming to assess the local, global and visual induction capabilities of models. Tasks also have variants with progressively increased difficulty. Tasks and variants evaluate a broad range of capabilities, from local single-step reasoning over folding

[4]These are obtained from free online resources, including 100origami, origami.me, origamiUSA and origami200. However, the programmatic descriptions and the images in *OrigamiBench* are original material, created by the annotation tool. Only program descriptions and the environment will be publicly released upon acceptance.

Task	Variants	Input	Output	Type	Metrics
1. Valid Transition	a: Assoc.; b: Causal	S_t	S_{t+1}	C	Acc
2. Action Grounding	a: No; b: Weak; c: Prog.	S_t, A_t	S_{t+1}	C	Acc
3. Action Induction	a: Prog.	S_t, S_{t+1}	A_t	C	Acc
4. Plan Execution	a: Assoc.; b: Causal; c: Prog.	S_t, $A_{t:T-1}$	S_T	C	Acc
5. Visual Ordering	–	shuffle($S_{t:T}$)	$\pi(S_{t:T})$	O	PAcc, K τ
6. Plan Ordering	–	S_t, S_T, shuffle($A_{t:T-1}$)	$\pi(A_{t:T-1})$	O	MPAcc, K τ
7. Plan Completion	a: $\lvert I\rvert$=1; b: $\lvert I\rvert$=2	S_t, S_{t+n}, mask$_I(A_{t:t+n-1})$	A_I	G	CPD
8. Plan Steering	a: No; b: Prog.; $i \in \{\lfloor n/2 \rfloor, \lfloor n/3 \rfloor\}$	S_t, S_{t+n}	$A_{t:t+i}$	C	Acc
9. Plan Induction	–	S_t, S_{t+n}	$A_{t:t+n-1}$	G	CPD

Legend. **Variants:** Assoc. = Associative, No = No Grounding, Prog. = Programmatic. **Type:** C = Classification, O = Ordering, G = Generative. **Metrics:** Acc = Accuracy, PAcc = Pairwise Accuracy, MPAcc = Mean Position Accuracy, K τ = Kendall's τ, CPD = Crease Pattern Dissimilarity (Oh et al., 2015). $\pi(\cdot)$ denotes the recovered ordering. mask$_I(A_{t:t+n-1})$ denotes the action sequence $A_{t:t+n-1}$ with the actions at index set $I \subset \{t, \ldots, t+n-1\}$ ($\lvert I\rvert$=1 or $\lvert I\rvert$=2) hidden from the model; the hidden actions A_I are the target to be predicted in Task 7.

Table 2: Overview of *OrigamiBench* evaluation tasks. Tasks 1-4 and 8 are classification problems over four candidate choices; Tasks 5-6 require recovering the correct ordering; Tasks 7 and 9 require free-form synthesis of actions. Classification tasks are designed using different candidate choice types, including associative, causal and programmatic distractors. Associative distractors are sampled from unrelated trajectories, causal distractors are sampled from other states or actions of the same trajectory. Programmatic distractors are obtained by intervening on the abstract syntax tree of the action and re-executing it from the same state. Full task descriptions are provided in Appendix E, while detailed definitions of the evaluation metrics are provided in Appendix D.

transitions, action grounding, and action induction, to multi-step plan reasoning through execution, ordering, and completion, ultimately progressing to plan steering and full plan induction from visual states. Table 2 summarizes the definitions of the tasks and their variants, including information about the input/output, the problem type and the evaluation metrics. More detailed information is provided in Appendix E and Appendix D. We use S_t to denote the rendered origami state after t folding steps, and A_t to denote the program expression (action) applied at step t.

Local Reasoning (Tasks 1-3). These tasks probe single-step understanding of folding transitions. Task 1 asks whether a model can recognize a valid successor state from the current state. Task 2 asks whether it can ground a given action in its geometric effect, while Task 3 asks the inverse question: whether the model can infer the action that explains an observed transition.

Global Reasoning (Tasks 4-7). These tasks extend transition understanding to multi-step trajectories. Task 4 asks whether a model can track cumulative state change over an action sequence. Tasks 5 and 6 ask whether it can recover the correct ordering of a shuffled trajectory over states and actions, respectively. Task 7 asks whether it can complete a plan by recovering actions masked from an otherwise observed sequence.

Goal-Directed Planning (Tasks 8-9). These tasks move toward planning from visual start and goal states. Task 8 asks whether a model can identify which candidate partial plan provides the most promising start toward the target. Task 9 asks the fully generative question: whether the model can induce a complete action sequence directly from the start and target states.

3 Related Work

Visual and physical reasoning benchmarks. Existing benchmarks probe complementary aspects of visual, spatial, algorithmic, and physical reasoning. RAVEN (Zhang et al., 2019) and ARC-AGI (Chollet, 2019; 2026) test abstract rule induction; CLEVR compositional reasoning (Johnson et al., 2017); Bongard-LOGO visual concept induction (Nie et al., 2020); and SMART-101 abstraction and spatial reasoning (Cherian et al., 2023). Physion evaluates physical outcome prediction (Bear et al., 2021), while PhyBlock studies physically grounded multi-step planning (Ma et al., 2025). In

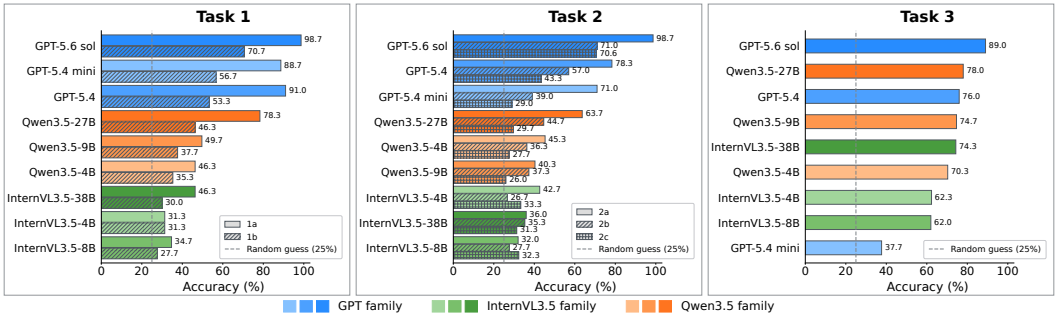

Figure 6: Accuracy (%) on Tasks 1-3.

contrast, *OrigamiBench* evaluates recovery of an executable generative procedure that reconstructs a target object through state-transforming actions.

Constructive solid geometry and procedural generation. Executable programs are increasingly used to represent visual and geometric structure. D^{2}CSG reconstructs CAD shapes as CSG compositions (Yu et al., 2023); ArcPro and PyTorchGeoNodes recover procedural shape programs from 3D observations (Huang et al., 2025; Stekovic et al., 2025); and VLMaterial predicts procedural material graphs from images (Li et al., 2025). In the forward direction, Infinigen and ProcTHOR procedurally generate large-scale environments (Raistrick et al., 2023; Deitke et al., 2022), while CaptainCook4D and Ego-Exo4D model the temporal structure of multi-step activities (Peddi et al., 2024; Grauman et al., 2024). Unlike these largely declarative, generative, or recognition-oriented settings, *OrigamiBench* requires an action-oriented sequential program whose execution progressively transforms the origami state.

Origami reasoning benchmarks. ORIGAMISPACE evaluates Multimodal Large Language Models (MLLMs) on pattern prediction, multi-step reasoning, spatial relations, and crease-pattern code generation (Xu et al., 2025), while GamiBench studies 2D-to-3D spatial reasoning through folded-state prediction, viewpoint validation, and impossible-pattern detection (Spencer et al., 2025). These benchmarks reason primarily over crease patterns or folded states. *OrigamiBench* instead evaluates synthesis of the folding procedure itself through an explicit executable action language.

Origami generation systems. Computational origami design predates recent learning-based approaches. Lang introduced algorithmic origami-base design and TreeMaker (Lang, 1996), while Tachi developed Freeform Origami and Origamizer for generating crease patterns realizing prescribed 3D shapes (Tachi, 2010; 2009). Akitaya et al. recover folding instructions from flat-foldable crease patterns (Akitaya et al., 2013). More recently, COrigami generates crease patterns from natural-language descriptions (Zahavy et al., 2026), while Learn2Fold formulates origami synthesis as conditional program induction with learned and symbolic components (Huang et al., 2026). Whereas prior systems focus on geometric design, crease-pattern generation, or instruction recovery, *OrigamiBench* provides a systematic evaluation of executable sequential folding-program synthesis.

4 Experimental Results

We evaluate a set of proprietary and open-weight vision-language models (VLMs) on Tasks 1-6 of *OrigamiBench*, covering both Local Reasoning (Tasks 1-3), Global Reasoning (Tasks 4-6) and Visual Induction (Tasks 8-9). We report results for three proprietary models accessed through API (`GPT-5.4-mini`, `GPT-5.4`, `GPT-5.6-sol`) and two families of open-weight models at increasing scale, `InternVL3.5` (4B/8B/38B) and `Qwen3.5` (4B/9B/27B). For all classification tasks (Tasks 1-4), the random-guessing baseline over four candidates is 25%.

4.1 Can Models Understand a Single Transition? (Tasks 1-3)

Fig. 6 reports accuracy on Tasks 1-3 (300 instances each). Task 1 asks a model to identify a valid successor S_{t+1} from S_t alone; Task 2 additionally conditions on the generating action A_t; Task 3 inverts the problem, recovering A_t from (S_t, S_{t+1}).

Understanding. All models exceed the 25% random baseline, but performance drops sharply once associative distractors are replaced by causal ones (1a→1b, 2a→2b: -20 to -40 points for GPT,

-10 to -30 for Qwen3.5, -0 to -20 for InternVL3.5 families). This suggests that models rely on visual pattern matching to solve the tasks rather than having a causal understanding of the involved transition. In other words, models suffer from *visual shortcuts*. Conditioning on A_t (Task 2 vs. Task 1) does not significantly help: five of eight models score lower on 2a than 1a, and accuracy drops further still under programmatic distractors (2c) with several open-weight models falling back to near-chance. Together, these results indicate that models largely ignore the symbolic action and struggle to *ground it visually*. Scale alone is not a reliable predictor of performance: Qwen3.5 improves fairly monotonically with size, InternVL3.5 does not (e.g., 1b: 31.33→27.67→30.00 from 4B to 38B), and even GPT-5.6-sol loses ∼30 points from 2a to 2c.

Transition Selection. Task 3 highlights two phenomena. Firstly, the gap in performance between proprietary and open-weight models reduces compared to the gap observed in Task 1 and Task 2. Secondly, there is an asymmetry in performance between Task 3 and Task 2c, suggesting that vision-to-program (induction) and program-to-vision (grounding) are not equally difficult.

Tasks 1-3 show that scale alone does not improve physical-transformation reasoning, visual shortcuts persist even in the strongest proprietary models, and grounding programmatic information visually remains largely unsolved.

4.2 Can Models Reason Globally? (Tasks 4-6)

Task 4: Plan Execution. Task 4 evaluates whether a model can predict the final state S_T from an intermediate state S_t and the remaining action sequence A_t, under associative, causal, and programmatic distractor settings (Appendix H.2). Performance degrades sharply with trajectory length (Fig. 7). At n=1, all models perform well, reaching up to 99% for GPT-5.6-sol and 64–79% for GPT-5.4 and Qwen3.5-27B. By n=3, however, every open-weight model falls to or below the 25% random-guessing baseline, and several remain near or below chance as trajectories grow longer (e.g., InternVL3.5-4B reaches 10% at n=8). Only GPT-5.4, GPT-5.6-sol, and Qwen3.5-27B remain consistently above chance across the evaluated range, although performance is not strictly monotonic with length; for example, GPT-5.4 recovers to 66% at n=8. These results suggest that Task 4 is challenging primarily because it requires composing multiple state transformations, rather than because of any particular distractor construction. The sharp drop after a single action indicates that current models can often execute an isolated transformation but struggle to propagate its consequences reliably over multiple steps. Aggregate results across distractor variants (Appendix H.2, Table 4) support the same conclusion: GPT-5.4 and GPT-5.6-sol exceed chance by a significant margin, whereas open-weight models remain close to random guessing overall.

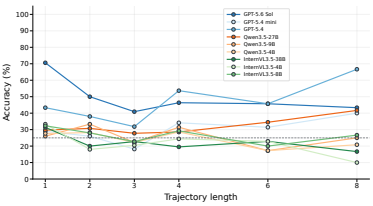

Figure 7: **Task 2c and 4c performance by trajectory length**. Trajectory length of 1 is obtained from Tasks 2c; Trajectory lengths 2-8 are from Tasks 4c.

Tasks 5&6: Visual and Plan Ordering. Figure 8 compares mean Kendall's τ across models for Task 5 (ordering rendered states) and Task 6 (ordering actions given only start/end states). For every model, ordering degrades sharply from Task 5 to Task 6: GPT-5.6-sol drops from 0.853 to 0.658, GPT-5.4 from 0.796 to 0.473, and Qwen3.5-27B from 0.822 to 0.300, confirming that recovering the order of actions is markedly harder than recovering the order of intermediate states, even for the strongest models. InternVL3.5 shows the opposite pattern: both variants achieve a negative τ on Task 5 (-0.126/-0.436 for 4B/8B), indicating orderings systematically anti-correlated with the ground truth, but become weakly positive on Task 6 (0.131/0.109), suggesting that the visual cues driving erroneous state orderings are less influential when the model instead orders actions. Only GPT-5.6-sol, GPT-5.4, and Qwen3.5-27B

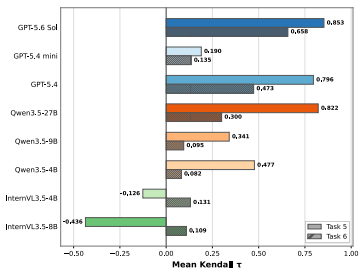

Figure 8: **Performance across Tasks 5 and 6** This chart compares model performance on Tasks 5, 6, where a score closer to 1 represents a more accurate ordering sequence.

retain a strong rank correlation ($\tau \geq 0.3$) on the harder action-ordering task, while the remaining models cluster near $\tau \approx 0.1$ or below on both tasks.

Task 7: Plan Completion. Table 7, provided in the Appendix, reports performance for recovering one (7a) or two (7b) masked actions from a partially observed folding plan, given the start and end states. Unlike the preceding tasks, Task 7 requires free-form generation under a constrained program syntax and output format, so failures can arise from instruction or format adherence in addition to incorrect action prediction. The GPT models show almost no such failures and maintain near-complete coverage across both variants, whereas several open-weight models frequently fail to return scoreable actions. Among valid predictions, however, `Qwen3.5-9B` achieves the highest action-type accuracy (62.7% on 7a and 67.7% on 7b) and ranks better by CPD than the GPT models. Moving from one to two masked actions does not consistently reduce the correctness of scoreable predictions, but it lowers coverage for several open-weight models. The same trend appears with trajectory length: action correctness remains relatively stable when outputs are valid, while coverage often decreases for longer trajectories.

4.3 Can Models Perform Visual Induction? (Tasks 8-9)

Task 8: Plan Steering. Table 8, provided in the Appendix, reports accuracy for selecting a partial action sequence $A_{t:t+i}$ that steers an initial state S_t toward a target state S_{t+n}, using prefixes covering either one half ($i = 1/2$) or one third ($i = 1/3$) of the trajectory. Performance is substantially higher on Task 8b than on 8a across all models: `GPT-5.6-sol` improves from 44.5% to 78.0% overall, while `Qwen3.5-27B` reaches 67.9% on 8b, making it the strongest open-weight model. Models also generally benefit from longer action prefixes, with `GPT-5.6-sol` reaching 83.2% on 8b at $i = 1/2$, compared with 72.8% at $i = 1/3$. This effect is not universal, however, as `Qwen3.5-27B` remains nearly unchanged across prefix lengths. Performance on 8a also tends to decrease with longer source trajectories, while 8b is markedly more stable across trajectory lengths for the strongest models. Overall, Task 8 shows that plan steering benefits from additional plan context, and remains sensitive to the trajectory length.

Task 9: Full Plan Generation. Table 9, provided in the Appendix, reports performance for generating a complete folding trajectory from the initial and target visual states. The GPT models achieve complete coverage, while several open-weight models fail to return valid complete programs, most notably `InternVL3.5-8B` with only 28.3% coverage. Among models with full coverage, `GPT-5.6-sol` ranks best by CPD, while `Qwen3.5-27B` obtains competitive CPD on its covered subset but covers only 86.7% of trajectories. Performance also degrades strongly with trajectory length: CPD increases for nearly all models as the target plan becomes longer, and coverage additionally drops for some open-weight models. Overall, Task 10 shows that full plan generation becomes increasingly difficult for longer trajectories.

5 Discussion

We have introduced a high-level language for describing the synthesis process of flat-foldable origami and, building on it, developed a benchmark to systematically probe the capabilities of modern VLMs. Future work will extend the evaluation to learning abstractions, as well as to orthogonal directions such as 3D origami and videos (Moriya et al., 2026). More broadly, benchmarks of this kind can help advance causal machine learning, physical AI, and neuro-symbolic programming by providing controlled settings in which models must connect structured representations with the mechanisms governing physical transformations. Such capabilities are relevant to domains including manufacturing, robotics, graphics, and protein folding.

AI use statement

In this work, we used generative AI tools (LLMs) for synthesizing data as described in the Data Generation and Organization section. Data is formally verified through our developed Origami environment. We used LLMs to polish the text in the paper and reviewed all AI-assisted work. We take full responsibility for the content of this work.

Ethics statement

The data and the software in *OrigamiBench* are original material. This includes the program descriptions to render the origamis and the rendering software. All material is subject to copyright and will be released to the community under MIT license upon acceptance.

Reproducibility statement

Data and software will be released publicly upon acceptance. Detailed information about data collection, organization, tasks creation and evaluation methodology is discussed in the main paper and in the appendices.

References

Hugo A Akitaya, Jun Mitani, Yoshihiro Kanamori, and Yukio Fukui. Generating Folding Sequences from Crease Patterns of Flat-Foldable Origami. In *SIGGRAPH*, 2013.

Daniel M Bear, Elias Wang, Damian Mrowca, Felix J Binder, Hsiao-Yu Fish Tung, RT Pramod, Cameron Holdaway, Sirui Tao, Kevin Smith, Fan-Yun Sun, et al. Physion: Evaluating Physical Prediction from Vision in Humans and Machines. In *NeurIPS*, 2021.

Anoop Cherian, Kuan-Chuan Peng, Suhas Lohit, Kevin A Smith, and Joshua B Tenenbaum. Are Deep Neural Networks SMARTer than Second Graders? In *CVPR*, 2023.

François Chollet. On the Measure of Intelligence. *arXiv*, 2019.

François Chollet. Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI), 2026. URL `https://arcprize.org/arc-agi/3/`.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *NeurIPS*, 2022.

Erik D Demaine, Jason S Ku, and Robert J Lang. A New File Standard to Represent Folded Structures. In *Fall Workshop Computational Geometry*, 2016.

Jeremy Gibbons. Origami Programming. *Mathematics in Computer Science*, 2003.

Kristen Grauman, Andrew Westbury, Lorenzo Torresani, Kris Kitani, Jitendra Malik, Triantafyllos Afouras, Kumar Ashutosh, Vijay Baiyya, Siddhant Bansal, Bikram Boote, et al. Ego-Exo4D: Understanding Skilled Human Activity from First-and Third-Person Perspectives. In *CVPR*, 2024.

Qirui Huang, Runze Zhang, Kangjun Liu, Minglun Gong, Hao Zhang, and Hui Huang. ArcPro: Architectural Programs for Structured 3D Abstraction of Sparse Points. In *CVPR*, 2025.

Yanjia Huang, Yunuo Chen, Ying Jiang, Jinru Han, Zhengzhong Tu, Yin Yang, and Chenfanfu Jiang. Learn2Fold: Structured Origami Generation with World Model Planning. In *arXiv*, 2026.

Thomas C. Hull. Origami Design Secrets: Mathematical Methods for an Ancient Art. *Mathematical Intelligencer*, 2005.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In *CVPR*, 2017.

Robert J Lang. A Computational Algorithm for Origami Design. In *Annual Symposium on Computational Geometry*, 1996.

Beichen Li, Rundi Wu, Armando Solar-Lezama, Changxi Zheng, Liang Shi, Bernd Bickel, and Wojciech Matusik. VLMaterial: Procedural Material Generation with Large Vision-Language Models. In *ICLR*, 2025.

Liang Ma, Jiajun Wen, Min Lin, Rongtao Xu, Xiwen Liang, Bingqian Lin, Jun Ma, Yongxin Wang, Ziming Wei, Haokun Lin, et al. PhyBlock: A Progressive Benchmark for Physical Understanding and Planning via 3D Block Assembly. In *NeurIPS*, 2025.

Maya Moriya, Sigal Raab, Yael Vinker, and Tali Dekel. FoldingAgent: Inferring Parametric Origami Procedures from Demonstration Videos. In *SIGGRAPH Asia*, 2026.

Weili Nie, Zhiding Yu, Lei Mao, Ankit B Patel, Yuke Zhu, and Anima Anandkumar. Bongard-LOGO: A New Benchmark for Human-Level Concept Learning and Reasoning. In *NeurIPS*, 2020.

Seung Man Oh, Godfried T. Toussaint, Erik D. Demaine, and Martin L. Demaine. A dissimilarity measure for comparing origami crease patterns. In *Proceedings of the International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, pp. 386–393. INSTICC, SciTePress, 2015. ISBN 978-989-758-076-5. doi: 10.5220/0005291203860393.

Rohith Peddi, Shivvrat Arya, Bharath Challa, Likhitha Pallapothula, Akshay Vyas, Bhavya Gouripeddi, Qifan Zhang, Jikai Wang, Vasundhara Komaragiri, Eric Ragan, et al. CaptainCook4D: A Dataset for Understanding Errors in Procedural Activities. In *NeurIPS*, 2024.

Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite Photorealistic Worlds Using Procedural Generation. In *CVPR*, 2023.

Ryan Spencer, Roey Yaari, Ritvik Vemavarapu, Joyce Yang, Steven Ngo, and Utkarsh Sharma. GamiBench: Evaluating Spatial Reasoning and 2D-to-3D Planning Capabilities of MLLMs with Origami Folding Tasks. In *arXiv*, 2025.

Sinisa Stekovic, Arslan Artykov, Stefan Ainetter, Mattia D'Urso, and Friedrich Fraundorfer. PyTorchGeoNodes: Enabling Differentiable Shape Programs for 3D Shape Reconstruction. In *CVPR*, 2025.

Tomohiro Tachi. Origamizing polyhedral surfaces. *IEEE transactions on visualization and computer graphics*, 2009.

Tomohiro Tachi. Freeform Variations of Origami. In *ICGG*, 2010.

Rui Xu, Dakuan Lu, Zicheng Zhao, Xiaoyu Tan, Xintao Wang, Siyu Yuan, Jiangjie Chen, and Yinghui Xu. ORIGAMISPACE: Benchmarking Multimodal LLMs in Multi-Step Spatial Reasoning with Mathematical Constraints. In *NeurIPS*, 2025.

Fenggen Yu, Qimin Chen, Maham Tanveer, Ali Mahdavi Amiri, and Hao Zhang. D2CSG: Unsupervised Learning of Compact CSG Trees with Dual Complements and Dropouts. In *NeurIPS*, 2023.

Tom Zahavy, Shaobo Hou, Thomas Tumiel, James Doran, Francesco Faccio, Xidong Feng, Alex Havrilla, Igor Khytryi, Chenglei Li, Lisa Schut, et al. COrigami: An AI Pipeline for Co-Designing Flat-Foldable Visually Recognisable Origami. In *arXiv*, 2026.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A Dataset for Relational and Analogical Visual REasoNing. In *CVPR*, 2019.