ORIGAMI AS A SPATIAL AND GEOMETRIC REASONING BENCHMARK

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal language models perform well on visual recognition and on tool use, but these abilities do not transfer directly to spatial and geometric reasoning. Geometric structure does not reduce to text or to a single image: exact angles and incidences, which surface lies in front of which, and the order in which operations were applied are not recoverable from a semantic description of a scene. We introduce **CP2Seq**, a benchmark for evaluating geometric reasoning and sequential planning in multimodal language models through origami. The benchmark contains 600 procedurally generated samples, each requiring a model to produce a valid sequence of folds from a crease pattern and a target folded state. Samples are produced by a deterministic folding engine. Ground truth is recorded during generation. We release the generation and evaluation framework to support additional samples and experiments across reference depths and action models. During evaluation, the engine executes proposed folds or rejects invalid actions with explicit feedback, while the model remains responsible for selecting actions and searching for a solution. We evaluate multimodal language models under progressively greater tool assistance, and against a deterministic breadth-first search over the same action set. Candidate sequences are assessed by executing them and comparing the resulting folded state with the target, allowing for planar translations, rotations, and reflections. Performance falls off sharply with sequence depth: in the Luna ablations no configuration solves a medium or hard sample (11 or more folds), while some-layers solves rise from 6/30 to 18/30 with higher reasoning effort. Failures are dominated by search rather than by illegal moves: in the broader evaluation, 48% of episodes end by revisiting earlier states. A breadth-first search over the same actions, under its own time budget, solves 54.7% of easy-group attempts, so short sequences are within reach of systematic search.

1 INTRODUCTION

Geometric planning requires a model to connect a description of a target with the operations needed to construct it. In origami, this involves choosing a fold line, identifying which parts of the sheet should move, and tracking how the resulting reflections change their positions and layer order. A locally valid fold may lead away from the target, so completing a sequence also requires selecting among alternatives and revising earlier decisions. We study whether multimodal language models can use visual and geometric observations to make these decisions over successive folds.

Origami provides a procedural setting for studying these decisions. A folding engine can generate instances across reference depths, record intermediate states, and check proposed actions under a specified folding model. Sequence planning also arises in engineering: robotic origami folding requires choosing the order of simple folds (Balkcom & Mason, 2008), self-folding machines fold flat sheets into functional structures (Felton et al., 2014), and deployable arrays must accommodate panel thickness when stowed (Zirbel et al., 2013). CP2Seq isolates the combinatorial part of this problem; physical feasibility, including thickness and contact, is outside the current model (Section 6).

Existing origami benchmarks ask questions about folds or generate crease patterns (Spencer et al., 2025; Xu et al., 2025). CP2Seq evaluates whether a model can translate geometric information into

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.

a complete, executable folding sequence, with each action checked for validity and the resulting construction checked against the target.

We introduce **CP2Seq**, a benchmark of 600 procedurally generated origami samples and a framework for generating, executing, and evaluating folding sequences. Each task supplies a crease pattern and a target folded state; the reference sequence and intermediate reference states are withheld. Starting from an unfolded square, the model must produce a valid sequence of simple folds under the specified action model. The crease pattern constrains which creases must be produced, while the target specifies the arrangement of the folded sheet, including layer order.

The task combines *geometric reasoning*, *spatial state tracking*, and *sequential planning*. A fold line is specified in the current folded coordinates, whereas the target crease pattern describes the original sheet. Connecting these representations requires tracking reflections and the correspondence between paper regions. Selecting a fold also requires reasoning about overlapping layers and their connectivity. Subsequent decisions depend on the state produced by earlier actions, so increasing sequence depth extends the planning horizon.

We generate examples procedurally by applying legal folds and recording the resulting sequence, crease pattern, and folded state. This supplies reference solutions without manual annotation and lets us vary reference depth and the permitted folding operations. Releasing the generator also enables fresh evaluation instances beyond the published corpus, reducing reliance on fixed examples that future models may encounter during training (Section 3).

We also provide an evaluation framework that presents images of the input crease pattern, the target folded state, and the current state after each accepted edit. Tools support adding and removing folds, inspecting geometry, and returning to a specified earlier step to explore another sequence. The engine checks proposed edits and returns feedback, while the model directs the search. Different levels of tool assistance allow us to study how this support affects performance (Section 4.3).

Evaluation checks both individual actions and the completed construction. Candidate sequences are replayed and assessed for crease-pattern agreement and agreement with the target folded state. Folded-state comparison accounts for a global translation, rotation, or turnover of the completed object, with the corresponding changes to layer order and face orientation (Section 4.2). We also provide a deterministic breadth-first search baseline using the same candidate-fold enumerator and terminal-state comparator. This baseline shows what systematic search achieves with the same filtering.

Our contributions are:

1. **A dataset and generation framework.** CP2Seq contains 600 samples with crease patterns, target states, recorded fold sequences, and intermediate states. Configurable generation supports experiments across reference depths and action models.
2. **An interactive evaluation and verification pipeline.** The framework connects visual and geometric inputs to tools for constructing and revising sequences, step-level legality checks, and final replay-based scoring. It includes corpus checks and a folded-state comparator. Experiments vary access to legal-fold enumeration and target-comparison feedback.
3. **A search baseline and an empirical analysis of model behavior.** We provide breadth-first search over the same candidate actions, report performance by difficulty group, and analyze episode termination. A broader evaluation of `gpt-5.6-luna` is complemented by smaller ablations of model version, reasoning effort, and tool assistance.

In ablations over model version, reasoning effort, and tool assistance, models solve some short sequences but no medium or hard sample; in the broader evaluation, most unsuccessful episodes (215 of 384) end by cycling through earlier states rather than by illegal moves (Section 5).

2 Related work

2.1 Origami benchmarks and generation

GamiBench (Spencer et al., 2025) evaluates multimodal models through visual questions about folded configurations, viewpoints, and impossible crease patterns. OrigamiSpace (Xu et al., 2025)

evaluates pattern prediction, multi-step spatial reasoning, spatial relationships, and crease-pattern code generation. It also provides an interactive environment for the code-generation task, in which the model writes a crease pattern; CP2Seq instead gives the crease pattern and asks for the fold sequence that produces it, verified one fold at a time. These benchmarks connect origami geometry to visual reasoning and generation, with different requirements on the model's output.

Other recent systems generate origami rather than answer questions about it. Agarwal et al. (2026) provides an interactive environment for synthesising flat-foldable patterns over 366 designs collected from Flat-Folder, and evaluates query efficiency, defined as the proportion of fold steps that contribute to the final model, together with geometric and semantic similarity. Zahavy et al. (2026) pairs a generator with a designer in a pipeline for flat-foldable and visually recognisable models, using a vision-language model for aesthetic and semantic evaluation and reporting the flat-foldability of generated crease patterns separately. Huang et al. (2026) plans structured origami generation against a learned world model. Moriya et al. (2026a) works in the opposite direction from generation and the closest to ours: it reconstructs a folding process from instructional video, using the sequence-level corpus of Moriya et al. (2026b). That corpus is the nearest existing collection to what CP2Seq needs, and we neither train nor evaluate on it: its sequences are human-authored and its 27 sequences over 337 frames are too few to stratify, whereas ours are generated to order and correct by construction.

2.2 Computational origami and sequence planning

A *simple fold* is the operation introduced by Arkin et al. (2004) for map folding: the sheet is folded along one line at a time, by $\pm 180^\circ$. The models are distinguished by how much of the stack that line carries, namely a single layer, a contiguous run from the top or bottom (some-layers), or the entire stack (all-layers). A crease pattern is *simply foldable* in a given model when some sequence of such folds flattens the sheet and leaves exactly its creases. Deciding that is a studied question whose difficulty depends on the shape of the sheet, the directions the creases take, and which of the models is permitted (Arkin et al., 2004; Akitaya et al., 2017).

CP2Seq uses the some-layers and all-layers variants; the all-layers model is also the one that describes sheet-metal bending, where it has been studied in its own right (Akitaya et al., 2020), and the decision problem has been examined for further restrictions on the crease directions (Akitaya et al., 2023). Work on flat-folded states studies layer ordering as well as planar geometry (Bern & Hayes, 1996; Akitaya et al., 2026; Ku et al., 2026), motivating our explicit representation of layers in the state and final comparison; the space of flat foldings has also been studied as a graph in its own right (Eppstein, 2019).

A folded state is not a folding motion. Every well-behaved folded state of a simple polygon of paper is reachable by some continuous motion (Demaine et al., 2004), extending an earlier result for rectangular paper (Demaine & Mitchell, 2001). Those results do not carry over to a restricted sequence of simple folds; each target here is generated by a recorded legal sequence, so what makes a sequence hard is not whether the paper can get there but which discrete steps are permitted along the way. That is the structure this benchmark measures, and it is why the environment rules on one fold at a time rather than on a final configuration. Earlier work also addresses generating folding sequences from crease patterns (Akitaya et al., 2013); sequence recovery is therefore an established computational origami problem. We use it to evaluate multimodal models through visual observations, executable actions, and feedback.

3 CP2Seq: task and dataset

3.1 Task and action models

Given a crease pattern C and a target folded state $S^\star$, the model must construct a sequence of legal folds $a_1, \ldots, a_T$ starting from an unfolded square. Replaying the sequence must reproduce the crease pattern and reach the target state under the equivalence defined in Section 4.2. Each state records paper geometry and layer ordering; the reference sequence and its intermediate states are withheld, and alternative valid sequences are accepted. Figure 1 summarizes the task.

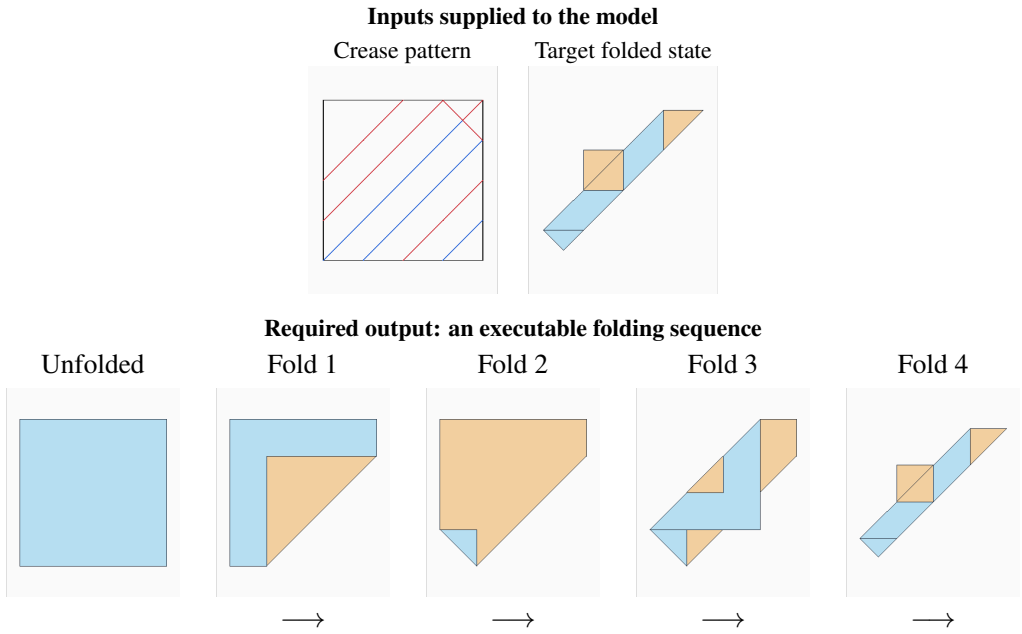


Figure 1: CP2Seq task illustrated by all-layers/easy-0001. The model receives the crease pattern and target state and must produce a valid sequence from the unfolded sheet. The lower row shows the recorded reference sequence, which is withheld during evaluation.

We use two simple-fold action models (Arkin et al., 2004): all-layers folds move the paper on the selected side of a fold line across the full stack, while some-layers folds select a contiguous run from the top or bottom of the stack. Both use flat 180° folds, with connectivity checks to reject moves that would tear the sheet; state representations and action constraints are detailed in Appendix A.

3.2 Dataset construction

Starting from a square sheet, the generator samples a legal fold under the chosen action model, applies it, and records the action and resulting state. This process continues to the requested reference depth; attempts with no available legal continuation are discarded. The accumulated creases are mapped to the original sheet to form the input crease pattern, with edges split at intersections. The last state supplies the target, and the recorded actions supply a reference solution. We group samples by reference fold count, hypothesizing that longer sequences generally require more search and are therefore harder to recover. These groups describe expected difficulty; reference length need not equal the shortest valid solution length. We also record coupling, the number of creases created by a fold, to characterize the geometry of each instance.

Generation pseudocode is provided in Appendix B.3.

We provide inspection software to visualize crease patterns and folded states, replay reference sequences, and examine verification feedback. These views support manual review to identify samples with inconsistent geometry or failed checks for exclusion from the dataset. Example interfaces and rejection feedback appear in Appendix B.

3.3 Dataset statistics

CP2Seq contains 600 samples: 400 all-layers examples grouped by reference depth and 200 some-layers examples with three to six reference folds (Table 1).

The model receives the crease pattern and target folded state; reference actions and intermediate reference states are withheld. Sample contents are documented in Appendix B.7.

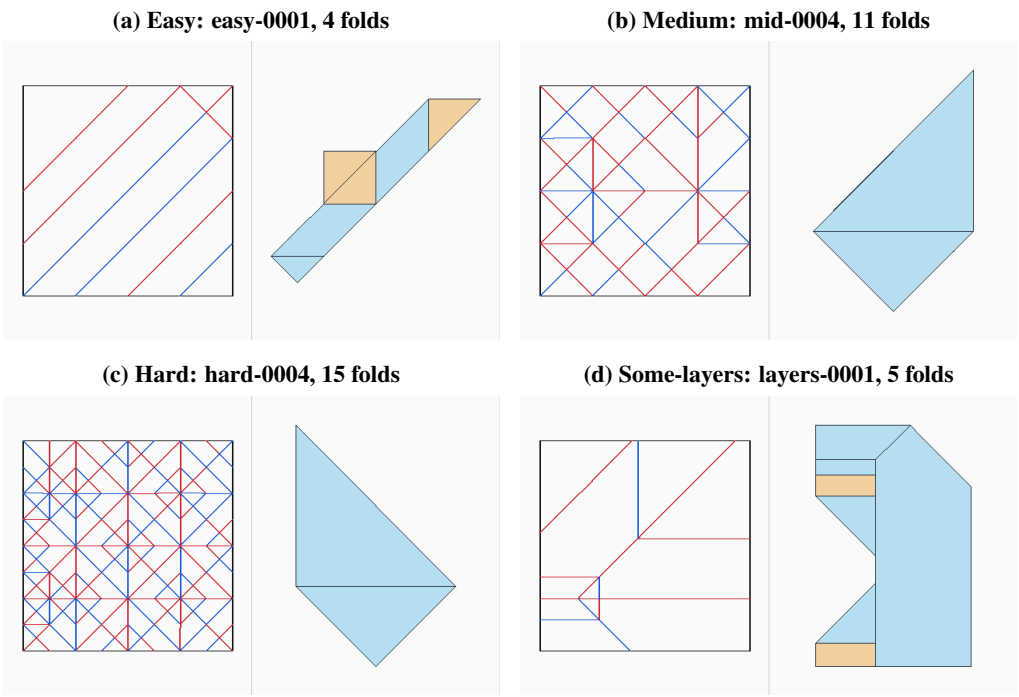


Figure 2: CP2Seq examples: each panel pairs the crease pattern (left) with the target top view (right). Panels (a–c) use all-layers folds; (d) is from some-generated-d5. Counts are reference folds. Red and blue denote mountain and valley creases.

Table 1: Dataset composition. Fold counts refer to recorded reference sequences.

Action model	Group	Samples	Reference folds
All-layers	Easy	200	4–10
All-layers	Medium	100	11–13
All-layers	Hard	100	14–19
Some-layers	Depth 3	50	3
Some-layers	Depth 4	50	4
Some-layers	Depth 5	50	5
Some-layers	Depth 6	50	6
Total		600	3–19

The two replay-based corpus checks pass on all 600 samples; their shared-engine validation scope is discussed in Section 6.

4 Experimental Setup

4.1 Environment

The model edits a folding sequence through the tools in Table 5. For each proposed fold, the engine reflects the selected paper across the fold line, updates layer order and orientation, and checks sheet connectivity and compatibility with the input crease pattern. Accepted edits update the current state; rejected edits return an error code and preserve the previous state.

For a some-layers fold, moving and stationary regions may share an edge away from the hinge. Reflecting only the moving region would separate that shared edge, so the engine rejects the proposal as `would-tear`. A `no-crease` rejection means that paper was selected to move but the operation did not create a crease segment. Other rejections cover an empty moving region, an incompatible over/under direction, and creases absent from the target pattern. That last check is a form of as-

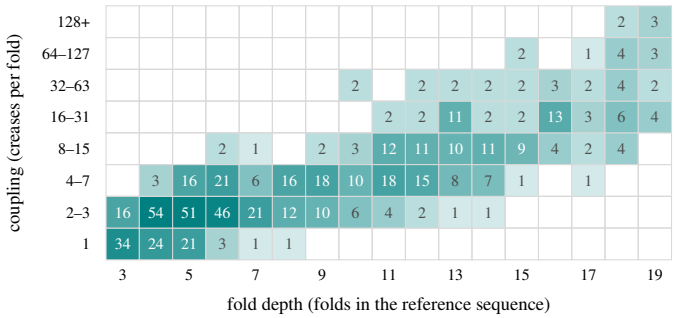


Figure 3: **What the corpus covers, over fold depth and coupling.** Each cell holds the number of samples at that depth and coupling, where coupling is creases made per fold, binned by doublings because an all-layers fold that crosses a doubled stack makes twice the creases. The two axes are not independent: a fold of all layers thickens the stack, so a long sequence cannot keep cutting few layers. No sample reaches 15 folds at coupling below 4, and the deepest such sample has 14; reaching that corner in practice takes pre-creasing, which this action space excludes. The corpus is therefore thinner than real folding in a way the difficulty groups alone do not show.

sistance: it prevents a class of unrecoverable moves, and the rates reported here are therefore rates under that constraint. Detailed conditions are listed in Appendix A.7.

The environment never refuses a fold for self-intersection. A fold moves either the whole stack or a contiguous run taken from the top or the bottom, and such a move cannot drive paper through the layers it leaves behind, so the action space excludes self-intersection by construction.

Observation tools are listed in Table 5. Exploded views use illustrative layer separation.

4.2 Evaluation metrics

We accept any executable sequence that produces the required crease pattern and folded state.

4.2.1 Verification

We execute the proposed sequence and compare its accumulated creases with the input pattern, including mountain–valley assignments. We then compare the final stack with the target after alignment by a single global translation, rotation, or turnover. A turnover reflects the geometry, reverses the stack order, and flips face orientations. The same transformation must align all layers in their corresponding stack order. A successful construction must satisfy both crease-pattern and folded-state agreement.

The crease comparator and the shared folded-state comparator both match positions within 2×10^{-6}; Table 3 lists every tolerance in the pipeline, and Appendix A.6 describes the alignment and representation details.

4.2.2 Reported metrics

We report solve rate by fold-count group, with the number of attempts and the applicable run budgets. Termination reasons distinguish successful submission, unsuccessful completion, budget limits, and repetition-related stopping. Query counts measure calls consumed before termination.

We measure CP edit distance between the accumulated crease pattern in the last recoverable state and the supplied pattern, in original-sheet coordinates. Collinear mountain and valley intervals are merged separately, excluding sheet boundaries. If m intervals match under the crease comparator, the insertion/deletion distance is $d_{\mathrm{CP}} = |C_{\mathrm{target}}| + |C_{\mathrm{candidate}}| - 2m$. Each missing or extra interval costs one; changing an assignment costs two. Partially overlapping intervals do not match unless their endpoints agree within tolerance. We also report $d_{\mathrm{CP}}/|C_{\mathrm{target}}|$, which can exceed one. Normalized distance is undefined for an empty target. Means weight attempts equally and exclude missing state artifacts, whose counts are reported separately.

Comparator implementation and validation cases are described in Appendix A.6.

4.3 Evaluation protocol

The model interface uses OpenAI Codex CLI (OpenAI, 2026) through a custom episode harness. Software version evidence is documented in Appendix B.1.

4.3.1 Episode procedure

One episode is one sample attempted by one model under one tier of assistance. The loop is the same for every arm and every model, and the only thing that varies between arms is what the model is allowed to see and call.

Each query contains one tool call. Accepted edits update the current sequence; rejected edits preserve it, and both consume budget. Removing a fold replays the remaining actions; returning to an earlier step truncates the sequence, while revision restoration can recover a previous branch. The reference sequence is withheld until scoring.

Episode pseudocode and the evaluation pipeline appear in Appendix B.4.

All conditions provide fold editing, backtracking, and state observations. Additional conditions provide legal-fold enumeration and target feedback (Section 4.4). The full tool interface and loading configuration appear in Appendix B.5.

4.3.2 Models and budgets

The recent ablations evaluate GPT-5.6 Luna at low reasoning effort and GPT-6 Luna at low and high effort, using ten samples per group and configuration. Runs use an 80-turn limit, a 300-second call timeout, and full image history. Earlier runs have different coverage and budgets and are reported separately (Section 5).

We include breadth-first search over the same enumerated legal actions as a systematic-search baseline (Appendix B.2).

4.4 Assistance conditions

The ablations compare three conditions with common sample IDs and budgets: *basic tools* (the interface of Section 4.3.1), *legal enumeration* (adding `list_legal_folds`), and *automatic feedback* (adding tier-3 target feedback after every edit). Earlier runs instead let the model request tier-3 comparison on demand.

The automatic-feedback condition supplies layer counts, aligned layer discrepancies, and suggested corrections after edits (Appendix B.5.1). Because those corrections do much of the target comparison for the model, we treat this condition as an upper bound on what tool support can achieve, not as a measure of the model's own geometric reasoning; conclusions about the model rest on the basic-tools condition. Even that condition is assisted, since folds that would crease outside the target pattern are rejected (Section 4.1).

5 Results

We report the completed Codex runs in the inventory generated on 24 September 2026, separating the broader GPT-5.6 Luna evaluation from the recent ablations. The inventory counts completed attempts, including unsuccessful submissions and stopping-limit terminations. For the ablations, all 360 planned sample–configuration pairs enter the denominator. Execution errors and omissions after blocking errors count as unsuccessful evaluations, with causes marked separately. We retain the first completed attempt per sample and configuration when available; retries do not add denominator entries.

Earlier GPT-5.6 Luna runs used different samples and budgets per condition, so we list them without comparison (Table 8); tool effects are measured only in the ablations.

5.1 TOOL AND REASONING-EFFORT ABLATIONS

The recent study uses the first ten samples of each group in every configuration: `easy-`, `mid-` and `hard-0001-0010` from the all-layers corpus and `layers-0001-0010` from the some-layers corpus; the some-layers samples all have five reference folds. It compares GPT-5.6 Luna at low effort with GPT-6 Luna at low and high effort. Tool conditions and run settings are given in Sections 4.4 and 4.3.2. Of the 360 planned sample–configuration pairs, five are protocol failures counted as unsolved (Appendix B.13).

Table 2: Recent Luna ablations. Cells show solved/evaluated samples; each group has ten intended samples. B: basic tools; L: legal enumeration; A: legal enumeration with automatic tier-3 target feedback. $*$ includes one input-size error; † includes one service prompt rejection; ‡ includes two omissions after blocking errors. All five count as unsuccessful evaluations. In two A cells (easy, GPT-5.6 low and GPT-6 high) one wrong submission followed a false *matches* from the comparison tool (Appendix B.13).

Model	Effort	Tools	Easy	Medium	Hard	Some-layers	Total
GPT-5.6 Luna	Low	B	2/10	0/10	0/10	0/10	2/40
	Low	L	1/10	0/10	0/10	5/10	6/40
	Low	A	4/10	$0/10^{*}$	0/10	6/10	$10/40^{*}$
GPT-6 Luna	Low	B	0/10	0/10	0/10	0/10	0/40
	Low	L	1/10	$0/10^{*\dagger}$	0/10	1/10	$2/40^{*\dagger}$
	Low	A	3/10	$0/10^{\ddagger}$	0/10	5/10	$8/40^{\ddagger}$
GPT-6 Luna	High	B	3/10	0/10	0/10	3/10	6/40
	High	L	3/10	0/10	0/10	7/10	10/40
	High	A	3/10	0/10	0/10	8/10	11/40

Tools. Summing the Total column of Table 2 over the three model settings gives, for the same 40 samples in each, 8/120 solves with B, 18/120 with L and 29/120 with A (two L and three A pairs are protocol failures counted as unsolved); the order B<L<A holds in each setting. By group, easy gives 5, 5 and 10 of 30 and some-layers 3, 13 and 19 of 30; medium and hard give none.

Reasoning effort. Pooled over the three tool conditions, GPT-6 Luna solves 6/30 some-layers samples at low effort and 18/30 at high effort. Per condition the increases are 0/10 to 3/10, 1/10 to 7/10 and 5/10 to 8/10; with ten samples per cell only the second is significant on its own (two-sided Fisher exact test, $p \approx 0.02$; $p \approx 0.21$ and 0.35 for the others), and a 95% Wilson interval spans roughly ± 0.3 (e.g. 0.11–0.60 for 3/10). Pairing outcomes by sample and condition, every discordant pair favours high effort (12 to 0; exact McNemar test, $p < 0.001$), so the effect is consistent even where single cells are underpowered. It is measured on five-fold some-layers samples; on the easy group, which spans four to ten folds, solves rise less, from 4/30 to 9/30.

Depth. No Luna ablation solves a medium or hard sample, and the breadth-first baseline solves 1/40 medium and 0/16 hard attempts (Appendix B.2), against 366/669 on the easy group. Within the easy group, the model solve rate falls from 32% at four folds to 0–15% at eight to ten (Table 10); beyond ten reference folds the benchmark is currently out of reach for these models and nearly so for systematic search. This matches the cost of search itself: breadth-first search time grows exponentially with depth, by a fitted factor of 3.9, 4.6 and 12.4 per fold on the easy, medium and hard groups (Figure 4, Appendix B.2). Depth is not isolated from other geometric differences between groups.

5.2 FAILURE ANALYSIS

Rejected actions never end an episode, although they consume turns, so episodes end on search-related stopping rules rather than on rule violations. In the earlier 447-attempt GPT-5.6 Luna snapshot, 167 episodes submitted a sequence (63 of them correct), 215 stopped after cycling through previously visited states, 18 stopped on repetition and 47 exhausted the turn budget. Most unsuccessful episodes (215 of 384) therefore end with the model returning to states it has already seen instead of committing to a new branch. The cycling rule stops an episode when the same fold is played from the same position three times; we have not tested whether looser limits would turn

some of these stops into solves. The ablations show the same pattern: 191 of their 300 unsuccessful episodes end in state cycling (Table 9).

The broader evaluation, ablations, BFS snapshot and CP-distance scan cover different attempt sets with different budgets; we report their denominators separately and do not pool them into a single controlled comparison.

6 LIMITATIONS

The evaluation covers Luna and BFS, and the ablations use ten samples per group and configuration. This study is currently privately funded, and these limits follow from the cost of evaluation: with full image history, one Luna episode consumed about 3.1 million input tokens on average over 20 episodes. In pilots, Qwen3.5-9B (via Tinker) cycled without solving either of two samples, and one 80-turn GPT-6 Astra episode cost about US$400. We are seeking research funding to extend the evaluation to further model families and to the full corpus.

The action space covers all-layers and selected some-layers simple folds, excluding pre-creasing; the some-layers corpus currently extends to six reference folds. The environment represents zero-thickness flat states and does not model material mechanics or continuous collisions during folding.

Termination labels record stopping conditions; attributing failures to individual capabilities requires further trajectory analysis. Revisit and budget limits constrain the observed search. Training overlap with the public corpus is unknown.

Corpus replay checks assess consistency with the shared generation engine. The terminal comparator compares current-plane polygons, parity and rank, and does not explicitly match original-sheet region identities, as detailed in Appendix A.6; equal shapes at different original-sheet locations can therefore require a correspondence check this path does not perform.

7 CONCLUSION AND FUTURE WORK

CP2Seq pairs 600 generated instances and an open-ended generator with an environment that rules on each fold and a scorer that accepts any valid sequence. The experiments support three observations. First, difficulty rises steeply with sequence depth: models solve some easy and some-layers samples, but no ablation configuration solves a medium or hard sample, and breadth-first search over the same actions solves 366/669 easy attempts but only 1/40 medium and 0/16 hard, in line with a search cost that grows exponentially with depth. Second, unsuccessful episodes end mostly under the cycling stop rule, with the model returning to states it has already visited, rather than on illegal moves, which are rejected without ending the episode. Third, assistance and reasoning effort help within reach but do not move that reach: on matched samples, legal-fold enumeration, target feedback and higher effort raise solve rates on short sequences, while medium and hard samples stay unsolved in every condition. Because each ablation result rests on ten samples per cell, we report these as trends with their uncertainty rather than as rankings of models.

Because the generator produces fresh instances with exact, programmatically checkable success, CP2Seq can also serve as an environment for training rather than only testing, a use we have not yet evaluated. Next steps are to extend the evaluation to other model families and to a text-only condition that isolates the contribution of images; to add a random-legal baseline and a symbolic solver such as xkevio (2026); to compare models and search under matched budgets; and to analyse trajectories decision by decision, so that failures can be attributed to state tracking, target comparison or backtracking. A natural extension is to check generated sequences with a bar-and-hinge simulator that models contact (Zhu & Filipov, 2021), connecting the benchmark to engineering folding problems.

REPRODUCIBILITY STATEMENT

The task, action models and state representation are specified in Section 3.1 and Appendix A; the generation procedure in Section 3.2 and Appendix B.3; the scoring rule, comparator and every numerical tolerance in Section 4.2 and Appendix A.6; the episode loop, tool interface, tool schemas

and prompt in Appendices B.4 and B.5; model versions, reasoning effort, turn and timeout limits in Section 4.3.2; the software version in Appendix B.1; and the accounting of every planned ablation case, including failures, in Appendix B.13. The evaluated models are closed and served through the Codex CLI, so their weights and sampling cannot be pinned; we release the saved runs with each attempt's model identifier and reasoning effort, so reported numbers can be audited if not regenerated exactly. The corpus generator, environment and verifier, corpus checks, state comparator and evaluation harness are released under the BSD 3-Clause licence, with links omitted for anonymous review. Every sample is synthesised by folding a square, so the data carry no personal, human-subject or scraped content; Flat-Folder's example collection is neither used nor redistributed.

USE OF LARGE LANGUAGE MODELS

Language models are evaluated as agents in the folding task. AI assistants were also used for implementation, literature-search assistance, and manuscript drafting and editing. Claude Code was used for coding assistance. AI-generated figure drafts served as composition references; dataset geometry and experimental results must be rendered from the corresponding artifacts. The authors are responsible for the manuscript and released implementation.

ACKNOWLEDGMENTS

Withheld for anonymous review.

REFERENCES

Naaisha Agarwal, Yihan Wu, Yichang Jian, Yikuan Hu, Nishad Mansoor, Mohan Li, Yifei Peng, Wang-Zhou Dai, Yao-Xiang Ding, and Emanuele Sansone. OrigamiBench: An interactive environment to synthesize flat-foldable origamis. *arXiv preprint arXiv:2603.13856*, 2026.

Hugo Akitaya, Josh Brunner, Erik D. Demaine, Della Hendrickson, Victor Luo, and Andy Tockman. Complexity of simple folding of mixed orthogonal crease patterns. *Thai Journal of Mathematics*, 21(4):1025–1046, 2023. URL `https://erikdemaine.org/papers/MixedSimpleFolds_TJM/`.

Hugo A. Akitaya, Jun Mitani, Yoshihiro Kanamori, and Yukio Fukui. Generating folding sequences from crease patterns of flat-foldable origami. In *ACM SIGGRAPH 2013 Posters*, pp. 1. Association for Computing Machinery, 2013. doi: 10.1145/2503385.2503407.

Hugo A. Akitaya, Erik D. Demaine, and Jason S. Ku. Simple folding is really hard. *Journal of Information Processing*, 25:580–589, 2017. doi: 10.2197/ipsjjip.25.580.

Hugo A. Akitaya, Cordelia Avery, Joseph Bergeron, Erik D. Demaine, Justin Kopinsky, and Jason S. Ku. Infinite all-layers simple foldability. *Graphs and Combinatorics*, 36(2):231–244, 2020. doi: 10.1007/s00373-019-02079-2.

Hugo A. Akitaya, Erik D. Demaine, and Jason S. Ku. Computing flat-folded states. In Guoxing Lu, Zhong You, and Michael Assis (eds.), *Origami8, Volume III*, pp. 201–219. Springer, 2026. doi: 10.1007/978-981-96-6561-7_14.

Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry*, 29 (1):23–46, 2004.

Devin J. Balkcom and Matthew T. Mason. Robotic origami folding. *The International Journal of Robotics Research*, 27(5):613–627, 2008. doi: 10.1177/0278364908090235.

Marshall Bern and Barry Hayes. The complexity of flat origami. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 175–183, 1996.

Erik D. Demaine and Joseph S. B. Mitchell. Reaching folded states of a rectangular piece of paper. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG)*, 2001.

Erik D. Demaine, Satyan L. Devadoss, Joseph S. B. Mitchell, and Joseph O'Rourke. Continuous foldability of polygonal paper. In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG)*, 2004.

Erik D. Demaine, Jason S. Ku, and Robert J. Lang. The FOLD file format, 2026. URL https://github.com/edemaine/fold. Format specification and software repository, accessed 25 September 2026.

David Eppstein. Realization and connectivity of the graphs of origami flat foldings. *Journal of Computational Geometry*, 10(1):257–280, 2019. doi: 10.20382/jocg.v10i1a10.

Samuel Felton, Michael Tolley, Erik Demaine, Daniela Rus, and Robert Wood. A method for building self-folding machines. *Science*, 345(6197):644–646, 2014. doi: 10.1126/science.1252610.

Evgueni T. Filipov, Ke Liu, Tomohiro Tachi, Mark Schenk, and Glaucio H. Paulino. Bar and hinge models for scalable analysis of origami. *International Journal of Solids and Structures*, 124: 26–45, 2017. doi: 10.1016/j.ijsolstr.2017.05.028.

Yanjia Huang, Yunuo Chen, Ying Jiang, Jinru Han, Zhengzhong Tu, Yin Yang, and Chenfanfu Jiang. Learn2Fold: Structured origami generation with world model planning. *arXiv preprint arXiv:2603.29585*, 2026.

Thomas C. Hull and Inna Zakharevich. Flat origami is Turing complete. *arXiv preprint arXiv:2309.07932*, 2023.

Thomas C. Hull, Marcus Michelen, and Corrine Yap. On random locally flat-foldable origami. *arXiv preprint arXiv:2502.04279*, 2025.

John E. Koehler. Folding a strip of stamps. *Journal of Combinatorial Theory*, 5:135–152, 1968.

Jason S. Ku. Flat-Folder: A crease pattern solver, 2022. URL https://github.com/origamimagiro/flat-folder. Software repository, accessed 25 September 2026.

Jason S. Ku and Erik D. Demaine. Folding flat crease patterns with thick materials. *Journal of Mechanisms and Robotics*, 8(3):031003, 2016. doi: 10.1115/1.4031954.

Jason S. Ku, Akira Terao, and Kenji N. Terao. An algebraic approach to layer ordering constraints for origami flat-foldability. In Guoxing Lu, Zhong You, and Michael Assis (eds.), *Origami8, Volume III*, pp. 317–333. Springer, 2026. doi: 10.1007/978-981-96-6561-7_21.

Stéphane Legendre. Foldings and meanders. *arXiv preprint arXiv:1302.2025*, 2013.

Weichen Liu, Qiyao Xue, Haoming Wang, Xiangyu Yin, Boyuan Yang, and Wei Gao. Spatial reasoning in multimodal large language models: A survey of tasks, benchmarks and methods. *arXiv preprint arXiv:2511.15722*, 2025.

Édouard Lucas. *Théorie des Nombres*, volume 1. Gauthier-Villars, Paris, 1891.

W. F. Lunnon. Multi-dimensional map-folding. *The Computer Journal*, 14(1):75–80, 1971. doi: 10.1093/comjnl/14.1.75.

Jun Mitani. Development of origami pattern editor (ORIPA) and a method for estimating a folded configuration of origami from the crease pattern. *IPSJ Journal*, 48(9):3309–3317, 2007. URL https://mitani.cs.tsukuba.ac.jp/dl/IPSJ-JNL4809039.pdf. In Japanese.

Maya Moriya, Sigal Raab, Yael Vinker, and Tali Dekel. FoldingAgent: Inferring parametric origami procedures from demonstration videos. *arXiv preprint arXiv:2609.00377*, 2026a. URL https://arxiv.org/abs/2609.00377.

Maya Moriya, Sigal Raab, Yael Vinker, and Tali Dekel. PurelandFold, 2026b. URL https://huggingface.co/datasets/mayaweiz/PurelandFold. Dataset accompanying FoldingAgent; CC-BY-4.0; accessed 25 September 2026.

Chihiro Nakajima. A spin model for global flat-foldability of random origami. *arXiv preprint arXiv:2403.07306*, 2024.

OEIS Foundation Inc. The on-line encyclopedia of integer sequences, A000136: Number of ways of folding a strip of n labeled stamps, 2026. URL https://oeis.org/A000136. Accessed 25 September 2026.

OpenAI. Codex CLI. Open-source software, version 0.155.1, 2026. URL https://github.com/openai/codex/tree/be2951ea34f0d295ed0becf97079f92fa5f6950e. Release tag rust-v0.155.1; accessed 25 September 2026.

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. *arXiv preprint arXiv:2512.22207*, 2025. URL https://arxiv.org/abs/2512.22207.

Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. Mind's eye of LLMs: Visualization-of-thought elicits spatial reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a45296e83b19f656392e0130d9e53cb1-Abstract-Conference.html. arXiv:2404.03622.

xkevio. Creasy, 2026. URL https://github.com/xkevio/Creasy. Software repository, accessed 25 September 2026.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://neurips.cc/virtual/2025/poster/115182.

Tom Zahavy, Shaobo Hou, Thomas Tumiel, James Doran, Francesco Faccio, Xidong Feng, Alex Havrilla, Igor Khytryi, Chenglei Li, Lisa Schut, Vivek Veeriah, Arijan Abrashi, Michał Kosmulski, Robert J. Lang, Nick Robinson, Brandon Wong, Marcus Chiam, Gloria Fang, and Satinder Singh. COrigami: An AI pipeline for co-designing flat-foldable visually recognisable origami. *arXiv preprint arXiv:2606.26299*, 2026.

Yi Zhu. Sim-FAST: Simulator for active structures, 2026. URL https://github.com/zzhuyii/Sim-FAST. Software repository, accessed 25 September 2026.

Yi Zhu and Evgueni T. Filipov. Sequentially working origami multi-physics simulator (SWOMPS): A versatile implementation. In *Proceedings of the ASME 2021 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE), Volume 8B: 45th Mechanisms and Robotics Conference*, pp. V08BT08A034, 2021. doi: 10.1115/DETC2021-68042.

Shannon A. Zirbel, Robert J. Lang, Mark W. Thomson, Deborah A. Sigel, Phillip E. Walkemeyer, Brian P. Trease, Spencer P. Magleby, and Larry L. Howell. Accommodating thickness in origami-based deployable arrays. *Journal of Mechanical Design*, 135(11):111005, 2013. doi: 10.1115/1.4025372.

A Folding model and verification

A.1 Flat-folded states

A state represents the positions of paper regions in the plane, their orientations, and their layer ordering. Original-sheet coordinates identify each region and preserve its connectivity to other regions as folds change its current position. Overlapping regions can occupy the same planar area while remaining distinct layers. Their original-sheet identities therefore matter both when validating a move and when comparing the completed construction with the target. Layer ordering is also a central component of computational treatments of flat-folded states (Akitaya et al., 2026).

A.2 SUPPORTED ACTIONS

The simple-fold literature distinguishes operations by the layers they move (Arkin et al., 2004). Our all-layers action selects the paper on one side of a fold line throughout the stack. Our some-layers action restricts that selection to a contiguous run from the top or bottom. An action specifies the line in current folded coordinates, the moving side, the layer selection, and whether the moving paper folds over or under the stationary paper. The selected paper is reflected across the line, with the corresponding orientation and stack-order updates. In the implemented some-layers model, a top run folds over and a bottom run folds under.

A.3 CONNECTIVITY AND TEARING

A move must preserve the connections between regions of the original sheet. If moving paper is attached to stationary paper along a segment away from the proposed hinge, the operation would separate connected paper and is rejected with a `would-tear` response. The check uses original-sheet connectivity together with the proposed motion; overlap in the current plane alone cannot establish whether two regions are attached. Rejection leaves the current state unchanged.

The engine also checks the action parameters and compatibility with the target crease pattern. It represents zero-thickness flat states and does not test continuous collisions during the folding motion.

A.4 DEPTH AND SEARCH DIFFICULTY

Reference depth is the number of generation steps. Search effort also depends on the branching structure and alternative solutions. For random proposal, the probability of selecting a legal move depends on both the legal set and the proposal distribution.

A.5 COMPUTATIONAL COMPLEXITY

The sequence-generation method of Akitaya et al. (2013) and the hardness proofs concern different contributions. For a precise hardness result, Akitaya et al. (2017, Theorem 3) prove that deciding simple foldability of square paper with assigned creases at multiples of 45° is strongly NP-complete in their some-layers and all-layers models. The input is an arbitrary crease pattern; the question is whether any permitted sequence folds it. Strong NP-hardness persists when numerical magnitudes are polynomially bounded.

The reduction starts from 3-PARTITION: divide integers into triples with a common sum. In the orthogonal-paper construction (Theorem 1), pleating a staircase encodes choices of integers. A bar must pass through successive cage slots; the geometry forces each group of choices to contain three integers with the required sum. A partition therefore yields a folding sequence, and a complete folding yields a partition. Theorem 3 embeds this construction in square paper using additional creases that enforce the preparatory folds. The construction has polynomial size, and a proposed folding order can be checked in polynomial time.

For CP2Seq, each target is generated by a recorded legal sequence, so reachability is promised rather than decided. The model must recover a sequence that also matches a supplied terminal state, under the implemented action rules and numerical tolerances. The theorem above does not by itself prove hardness of this promised reconstruction problem or of our sample distribution. Its relevance is that general sequence selection can encode combinatorial choices even when individual actions can be checked efficiently. In an episode, the practical search burden depends on the available actions, solution depth, and the number of unsuccessful branches explored; fold count alone does not determine that burden.

A.6 COMPARISON IMPLEMENTATION

The engine compares creases in original-sheet coordinates after grouping and merging collinear intervals by assignment. Interval endpoints are matched within 2×10^{-6}.

The shared terminal comparator first checks layer counts. It considers direct alignment and turnover, with turnover reversing layer order and face parity. Correspondences between vertices of the first

layer propose global alignments; a candidate alignment is accepted only if every layer polygon and parity matches its target rank. Polygon matching uses a positional tolerance of 2×10^{-6}, short-edge cleanup at 10^{-7}, and a collinearity threshold of 10^{-9}.

Table 3 collects every numerical tolerance the reported results depend on. Each result record carries the values it was scored under, so a re-scoring can be distinguished from a comparison at the same settings.

Table 3: Numerical tolerances used by the reported checks.

Value	Where used	What it admits
2×10^{-6}	Crease comparator; terminal comparator	Interval endpoint and polygon vertex positions, after an arbitrary global isometry.
10^{-7}	Terminal comparator	Short-edge cleanup before polygon matching.
10^{-9}	Terminal comparator	Collinearity threshold for vertex removal.
2×10^{-6}	CP edit distance	Interval endpoint agreement.
2.44×10^{-4}	CP edit distance	Line quantization when grouping collinear intervals.
10^{-5}	CP edit distance	Gap below which two intervals on one line are merged.

The shared implementation represents current-plane polygons, parity, and rank. It does not explicitly compare original-sheet region identities. Any identity-aware offline comparator must be distinguished from this path, and the comparator version used for each reported result must be recorded. In particular, equal shapes at different original-sheet locations can require a stronger correspondence check than current-plane geometry alone.

Comparator test cases cover translations, rotations, turnovers, and equivalent polygon representations, including changes in start vertex and winding. Negative cases include incorrect stack order or face orientation, missing layers, scaling, inconsistent per-layer transformations, and empty stacks. Asymmetric polygons distinguish reflections from rotations. The corpus-side comparator has a separate negative control.

The repository contains comparison and verification tests; executed test results and the exact comparator revision must accompany any reported validation.

A.7 Action rejection conditions

Table 4: Geometric and action constraints checked by the implementation. Rejected edits preserve the previous state.

Code	Condition
`would-tear`	A moving region and a stationary region share an original-sheet edge whose current position is not on the fold line. Reflecting one side would separate the join.
`no-crease`	A nonempty moving selection produces no crease segment in the paper. This condition is checked separately from tearing.
`nothing-to-move`	The selected half-plane contains no selected paper, or the requested partial selection contains no layers.
`direction-impossible`	The action requests folding a top run under the stack or a bottom run over it, contrary to the supported layer-selection rule.

The independent Flat-Folder wrapper uses layer-order constraints, including taco-taco, taco-tortilla, tortilla-tortilla, and transitivity constraints (Akitaya et al., 2026; Ku, 2022). It tests whether the pattern admits a flat-folded state.

B SOFTWARE AND SUPPLEMENTARY EXAMPLES

B.1 SOFTWARE VERSIONS

The inspected Codex CLI installation is version `0.155.1` (OpenAI, 2026). Its upstream release tag `rust-v0.155.1` resolves to commit `be2951ea34f0d295ed0becf97079f92fa5f6950e`.

The saved run configurations record the executable path but do not record its version or binary hash; the installation was checked on 25 September 2026.

B.2 BREADTH-FIRST SEARCH ANALYSIS

The saved BFS evaluation contains 725 attempts: 366/669 solved on easy samples (54.7%), 1/40 on medium, and 0/16 on hard. These attempts include repeated samples and different budgets and are not paired with the recent ten-sample ablations. BFS uses wall-clock, state-expansion, and depth limits, while model runs use turn and call-timeout limits.

A fold is specified by a line in current-plane coordinates together with a layer selection, and both are drawn from small finite sets. Candidate lines are read off the crease pattern itself: the distinct lines through its mountain and valley edges, carried forward through each distinct face transform into current coordinates, then deduplicated. On a box-pleated pattern thousands of crease edges collapse to a few dozen lines. Layer selections are not arbitrary subsets either, because a partial fold must be a contiguous run at one extremity of the stack; the selections are the whole stack, the top k, and the bottom k, which is about $2N$ for N layers rather than 2^N, with the over or under direction forced for partials. Each surviving candidate is judged by the same function the `add_fold` tool calls, so a listed fold cannot be one the tool would then reject, and it is dropped if it fails any material check of Section 4.1 or would crease the sheet where the target pattern carries no crease.

Across 73,750 enumerations recorded in the saved runs, this evaluates a mean of about 2,010 candidate actions per state and returns 3.51 of them, a ratio of approximately 570 between evaluated and returned candidates. The baseline expands with the same enumerator the model is given, tests goals with the same terminal comparator that decides `solved`, reads no reference sequence, and runs under a per-sample wall-clock budget with caps on states expanded and depth.

The supplementary cost model approximates search time as the product of two exponential factors: the frontier grows with the branching factor, and the cost of expanding one node grows with depth as well, because one enumeration evaluates four candidates per crease line per facet and a fold of all layers at most doubles the facets. Fitting both from the saved searches gives a per-fold multiplier of 3.9 on the easy group, 4.6 on medium and 12.4 on hard (Figure 4). These fitted curves extrapolate beyond the observed depths; they are not measured solution times or lower bounds. The hard-group fit uses only two samples.

B.3 DATASET GENERATION

```
GENERATE(reference_depth, action_model):
    state <- unfolded square
    actions, frames <- [], []
    for step in 1..reference_depth:
        candidates <- LEGAL_FOLDS(state, action_model)
        if candidates is empty: return FAILED
        action <- SAMPLE(candidates)
        state <- APPLY(state, action)
        actions.append(action)
        frames.append(state)
    cp <- CREASE_PATTERN_IN_ORIGINAL_SHEET(state)
    cp <- SPLIT_EDGES_AT_INTERSECTIONS(cp)
    if not REPLAY_CHECK(cp, actions, state): return FAILED
    return (cp, state, actions, frames, METADATA(actions))
```

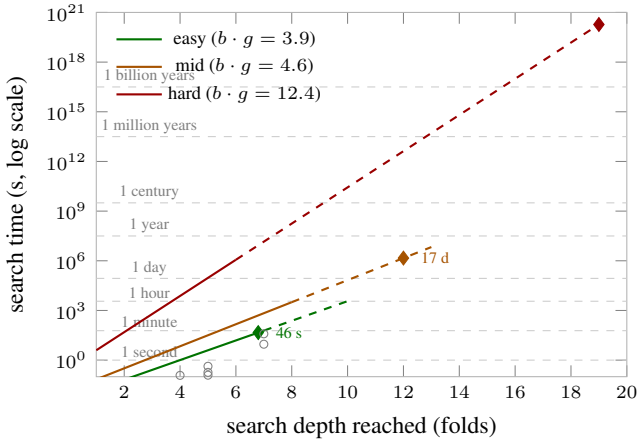


Figure 4: **Breadth-first search time against search depth, by difficulty group.** The vertical axis is logarithmic, so a straight line is exponential growth and its slope is $\log(b \cdot g)$. Curves are $t(d) = c_0(b \cdot g)^d$, the product of two exponentials: the frontier grows as b^d for a deduplicated branching factor b, measured as generated over expanded during search, while the cost of expanding one node grows as g^d, because one enumeration evaluates $4 \times$ (distinct crease lines) $\times$ (facets) candidates and a fold of all layers at most doubles the facets. g and c_0 are fitted by least squares on per-depth enumeration timings. Each curve is solid only where that group was observed and dashed beyond, and spans only the depths its own corpus reaches. Diamonds are each group's mean reference depth; the hard group's sits at 10^{20} seconds, which is why the axis runs that far. Circles are observed solves. b is fitted from budget-exhausted runs: 7 over 3 samples for easy, 12 over 3 for medium, and 8 over 2 for hard, across 5 time budgets. The hard extrapolation rests on two samples. Extrapolated values are model projections, not measured solution times or lower bounds.

Group (reference folds)	Median states expanded	Solved / attempts
Easy (4–10)	65	366/669
Medium (11–13)	1,051.5	1/40
Hard (14–19)	532	0/16

Figure 5: Observed BFS search effort in the saved aggregate. Bar lengths use a linear scale and show median states expanded across attempts, including unsuccessful attempts. Budgets and repeated samples are not standardized by this aggregate; 358 of 725 attempts terminated in timeout.

B.4 Evaluation pipeline

```
EPISODE(sample, model, condition, query_budget):
    cp, target <- LOAD_INPUTS(sample)
    session <- INITIALIZE(cp, target, unfolded_square)
    history <- []
    for query in 1..query_budget:
        obs <- OBSERVE(session, condition)
        call <- model(cp, target, obs, history, condition)
        if call is finish: break
        # Edits: add/remove fold, go to step, restore revision.
        # Reads: state, images, candidates, target feedback.
        result <- DISPATCH_TRANSACTIONALLY(session, call)
        # Rejected edits leave session unchanged but consume a query.
        # Read calls consume a query without changing the sequence.
        history.append(call, result)
        if STOPPING_RULE(session, history): break
    return SCORE(REPLAY(session.actions), REFERENCE_AT_SCORING(sample))
```

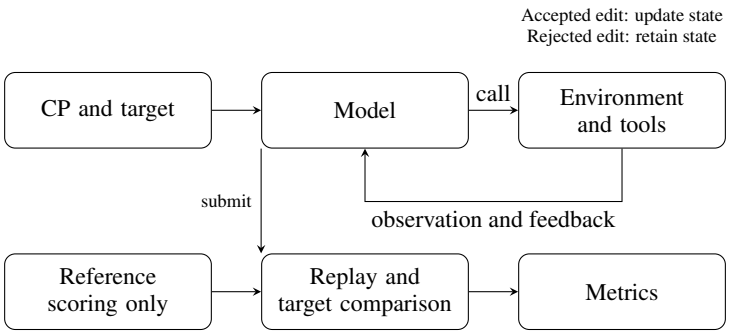


Figure 6: Evaluation pipeline. The model edits and inspects its current construction. The reference sequence is withheld from the model and used only for scoring.

B.5 Tool interface and feedback

Basic tools comprise all entries in Table 5 except the optional `list_legal_folds` and `compare_to_target` tools. Loading and configuration are described in Appendix B.10.

Table 5: Tools available to the evaluation harness; exposure depends on run configuration.

Tool	Operation
`add_fold`	Append a checked fold; reject invalid actions without changing state.
`remove_fold`	Remove a specified fold and replay the remainder transactionally.
`go_to_step`	Retain the first specified number of folds.
`restore_revision`	Restore a previously recorded branch state.
`get_state`	Return the current sequence and layer geometry.
`get_images`	Render top, oblique, reverse, X-ray, and exploded views of a completed step; exposed views depend on run configuration.
`list_legal_folds`	Optionally enumerate legal candidates compatible with the target CP.
`compare_to_target`	Optionally return feedback at the configured level (Section 4.4).
`finish`	End the episode and evaluate the current sequence.

B.5.1 Target-feedback levels

The run configuration selects one of three feedback levels for `compare_to_target`. The tool compares the current and target stacks.

Table 6: Information returned at each target-feedback level.

Level	Information returned
1: Counts	Current and target layer counts, their difference, and a count-based summary.
2: Diagnosis	Level 1 plus aligned layer matches, mismatched ranks, and a diagnosis of count, shape, orientation, or ordering disagreement.
3: Corrections	Level 2 plus suggested corrections for up to twelve mismatched ranks, such as orientation changes or a different target rank for a layer shape.

B.6 Tool schemas and prompt

The following listings document the tool schemas and folding prompt; tool exposure is configured per run.

```
"""Schemas supplied to the native Qwen3.5 tool-aware vision renderer."""
import json
```

```
def tool(name, description, properties=None, required=None):
    return {"type": "function", "function": {"name": name, "description": description,
            "parameters": {"type": "object", "properties": properties or {},
                           "required": required or [], "additionalProperties": False}}}


TOOLS = [
    tool("add_fold", "Append a 180-degree fold. Choose angle_index OR angle_degrees. Partial
     ↪ top/bottom runs must not tear connected paper.", {
        "angle_index": {"type": "integer", "enum": [0, 1, 2, 3]},
        "angle_degrees": {"type": "number", "description": "Line angle counterclockwise from +
     ↪ x; offset uses unit normal (-sin(theta), cos(theta))."},
        "selection_mode": {"type": "string", "enum": ["all", "top", "bottom"]},
        "layer_count": {"type": "integer", "minimum": 1, "description": "Required for top/
     ↪ bottom, omit for all."},
        "offset": {"type": "number"}, "move_positive": {"type": "boolean"},
        "over": {"type": "boolean"}}, ["offset", "move_positive", "over"]),
    tool("remove_fold", "Remove 1-based fold; replay the rest transactionally.",
         {"step": {"type": "integer", "minimum": 1}}, ["step"]),
    tool("go_to_step", "Keep the first N folds; zero clears the sequence.",
         {"step": {"type": "integer", "minimum": 0}}, ["step"]),
    tool("restore_revision", "Restore a returned revision, including abandoned branches.",
         {"revision": {"type": "integer", "minimum": 0}}, ["revision"]),
    tool("get_images", "Render top, two oblique, X-ray, and exploded-stack PNGs at a completed
     ↪  step.",
         {"step": {"type": "integer", "minimum": 0}}),
    tool("get_state", "Inspect the current sequence and bottom-to-top layer polygons."),
    tool("list_legal_folds", "List the folds that are legal in the current state and on-target
     ↪  for the CP. "
         "Read-only. Every listed action is accepted verbatim by add_fold while the state is
     ↪ unchanged.", {
        "max_results": {"type": "integer", "minimum": 1},
        "selection_filter": {"type": "string", "enum": ["any", "all", "top", "bottom"]},
        "include_rejected": {"type": "boolean"}}),
    tool("compare_to_target", "Compare the current stack with the target folded state. Read-
     ↪ only. "
         "Detail is fixed by the run's configured tier, not by you. Uses the target state only
     ↪ ; "
         "the reference fold sequence is never consulted."),
    tool("finish", "End this episode and evaluate the current sequence."),
]

# The Codex loop exposes only a subset in its baseline condition; the enumerator changes
# what the benchmark measures, so it must be requested explicitly. See TODO.md.
ENUMERATION_TOOLS = ("list_legal_folds",)
COMPARE_TOOLS = ("compare_to_target",)


def _whole_stack_only(tools):
    """Strip partial folds from the schemas, so nothing offers what ToolSession will refuse.

    Two edits, and both are needed. add_fold loses selection_mode and layer_count, or the
    model can request a run the simulator then rejects; list_legal_folds loses
    selection_filter, whose remaining values would all be lies once the run forces "all".
    """
    out = []
    for t in tools:
        name = t["function"]["name"]
        if name not in ("add_fold", "list_legal_folds"):
            out.append(t)
            continue
        copy = json.loads(json.dumps(t))
        props = copy["function"]["parameters"]["properties"]
        for key in ("selection_mode", "layer_count", "selection_filter"):
            props.pop(key, None)
        if name == "add_fold":
            copy["function"]["description"] = (
                "Append a 180-degree fold of the whole stack. Choose angle_index OR
     ↪ angle_degrees.")
        out.append(copy)
    return out


def tools_for(mode, compare_tier=0, action_space="any"):
    """The action set for one experimental condition.

    compare_to_target appears only when a tier is configured, so the default arms are
    byte-identical to every run recorded before the comparison tool existed. action_space
    defaults to "any" for the same reason: nothing changes unless a run asks for it.

    action_space is a property of the corpus, not a preference. On release/all-layers every
```

```
    one of the 4180 reference folds is whole-stack, so partial folds cost 3-87x in search and
    buy nothing; on some-verified-d3 and some-generated-d6 they are what 14% and 34% of the
    samples require. Whichever is chosen must be given to the search arm too, or the
    comparison measures the setting rather than the solver.
    """
    if mode == "base":
        chosen = [t for t in TOOLS if t["function"]["name"] not in ENUMERATION_TOOLS]
    elif mode == "legal-folds":
        chosen = list(TOOLS)
    else:
        raise ValueError(f"Unknown tool mode: {mode}")
    if not compare_tier:
        chosen = [t for t in chosen if t["function"]["name"] not in COMPARE_TOOLS]
    if action_space == "all-layers":
        chosen = _whole_stack_only(chosen)
    elif action_space != "any":
        raise ValueError(f"Unknown action space: {action_space}")
    return chosen

Recover a folding sequence from the supplied crease pattern (CP) and final folded
state. The sequence starts empty. Use one tool call at a time, then inspect its
result before proposing the next fold. Do not return an entire sequence in prose.
The environment writes your accepted actions into seq.json.

You receive only geometry and images of the CP, final target, and current state.
No reference folding sequence is supplied. All folds are flat 180-degree simple
folds. Select all layers (default), or a contiguous run from the top or bottom.

Coordinates are x right, y up. Fold axes are in CURRENT folded coordinates:

| angle_index | fold line | positive side |
| --- | --- | --- |
| 0 | y = offset | y > offset |
| 1 | y = x + offset | y - x > offset |
| 2 | x = offset | x > offset |
| 3 | x + y = offset | x + y > offset |

For arbitrary line angles, supply angle_degrees instead of angle_index, never
both. With theta counterclockwise from +x, the line is
-sin(theta)*x + cos(theta)*y = offset; offset is signed perpendicular distance.
move_positive selects the greater-than side. The folding motion remains 180 degrees.
selection_mode is all (default), top, or bottom. Top/bottom requires layer_count;
omit layer_count for all. Top must fold over; bottom must fold under.
The engine tracks original-sheet connectivity. A partial fold returns would-tear
if selected paper is attached to stationary paper away from the proposed hinge.
For example, selecting three of four layers may tear a paired flap. This check
runs before target-CP checking and leaves state unchanged on rejection. Inspect
sheet_polygon coordinates and exploded views when choosing a connected run.

move_positive selects which half-plane moves. over=true puts the reflected moving
stack above the stationary stack; false puts it below. A flipped stack reverses
its internal order. Offset is not a normalized distance for diagonal axes.
The tool rejects extra creases or incompatible mountain/valley assignments and
leaves the state unchanged on error. This is a zero-thickness geometric engine;
it does not check continuous collisions during motion.

add_fold appends ONE fold. remove_fold removes a 1-based fold and replays the
remaining actions; an invalid replay leaves the sequence unchanged. go_to_step
keeps the first N folds (0 empties the sequence). restore_revision restores a
previous tool state, even after changing branches; revision 0 is the empty sheet.
These editing operations create new revision IDs. get_state returns your current
sequence and geometry. get_images returns PNG views of any completed step in the
current branch without changing it. Use the returned revision IDs when backtracking.

Images: CP red = mountain, blue = valley, dark boundary = paper edge. Top and
two oblique views show paper shape with a small illustrative layer separation.
Oblique views look down from opposite directions, never exactly edge-on. X-ray
is a top projection: more overlapping layers produce darker pixels. Each view
is fitted independently; use geometry for coordinates and scale, not pixel size.
Exploded views separate layers upward for inspection: layer 1 is the bottom,
and p0/p1 labels give face parity. Separation is illustrative, not paper thickness.
Initial images include the target's exploded view; feedback images include the
current state's exploded view after edits.

Try to reproduce both the CP and the supplied target, including face orientation
and layer order. Terminal matching accepts any global translation and any rotation
angle, including 30 degrees. It also accepts turning the whole model upside down:
reflect the geometry, reverse the bottom-to-top stack, and flip every face parity.
These operations can be combined. One common rigid transformation must match all
layers; arbitrary layer rearrangements, scaling, or independently moving layers
do not count. CP matching still uses the supplied original-sheet coordinates and
```

```
mountain/valley assignments. Call finish when you are done. This ends the episode and
    ↪ evaluates your
current sequence; it does not automatically claim success.
```

B.7 SAMPLE CONTENTS

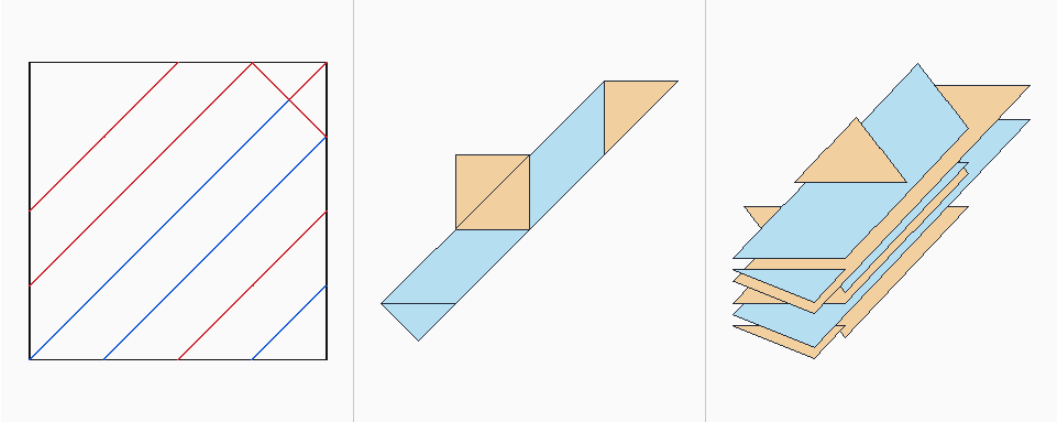
Each sample contains the input crease pattern, a reference state sequence, reference fold actions, and metadata (Table 7).

Table 7: Contents of a CP2Seq sample.

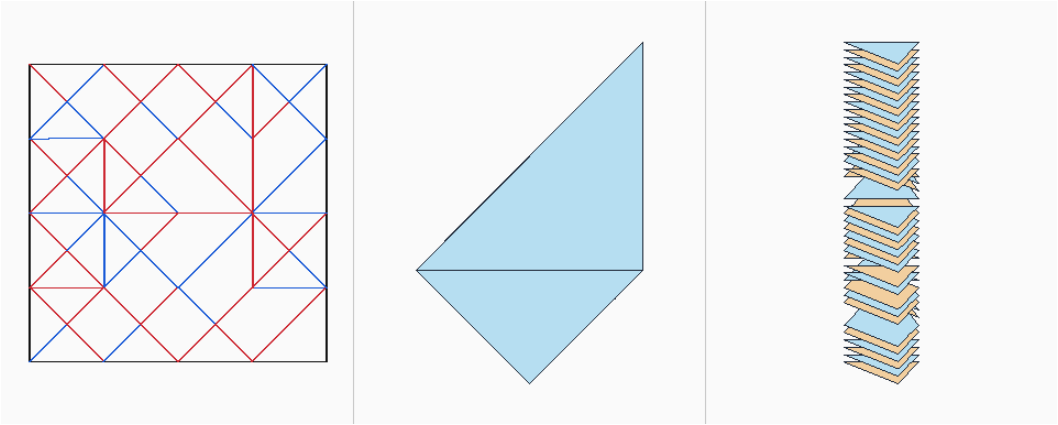
Artifact	Contents
Crease pattern	Input crease geometry.
State sequence	Folded states after each reference action; only the final state is supplied as the target.
Reference actions	Withheld fold lines, directions, layer selections, and creases created.
Metadata	Sample descriptors and generator version.

B.8 DATASET GALLERY AND REFERENCE SEQUENCES

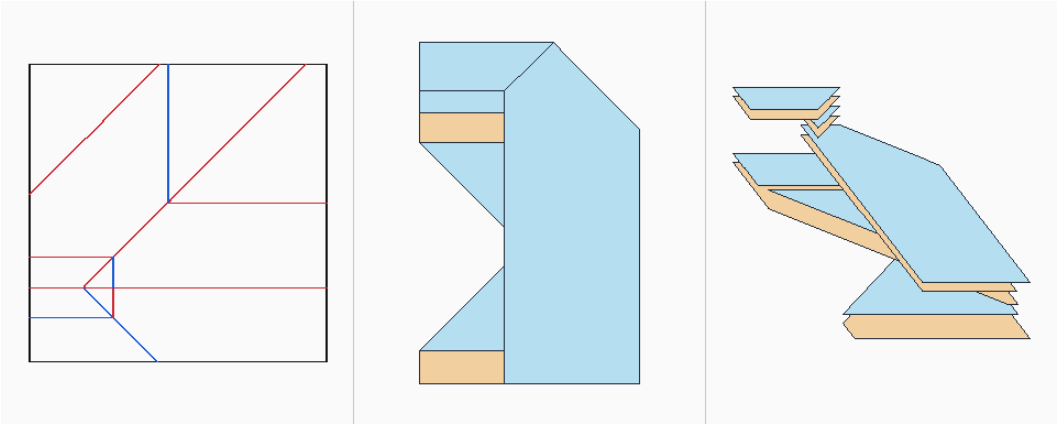
Each rendered panel contains the crease pattern, top view, and an oblique view with illustrative layer separation.
all-layers/easy-0001 (all-layers, 4 reference folds)



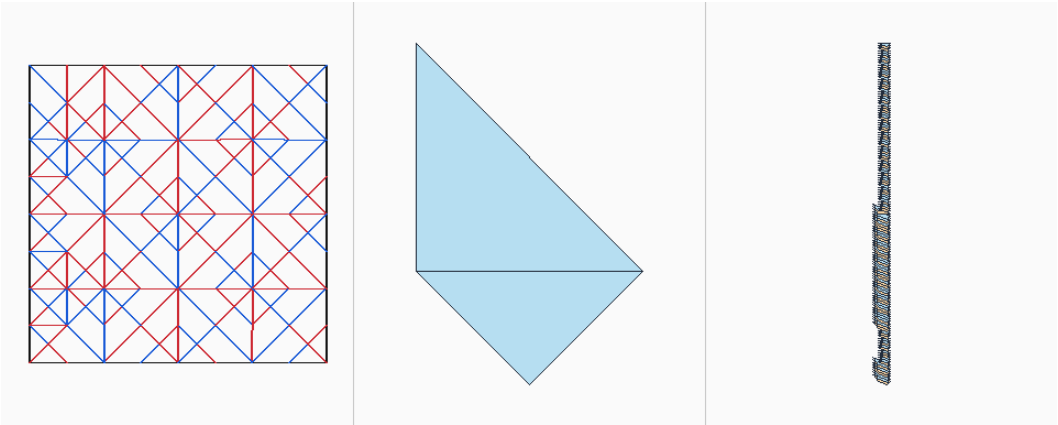
all-layers/mid-0004 (all-layers, 11 reference folds)



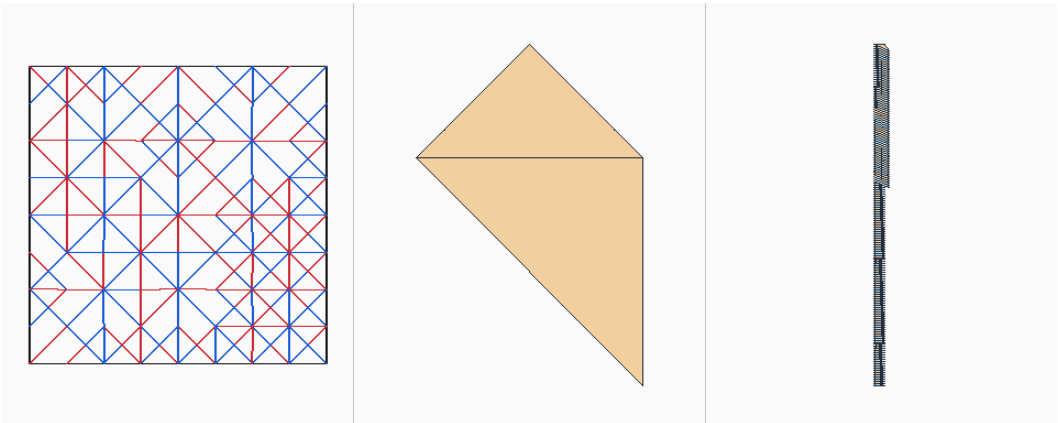
some-generated-d5/layers-0001 (some-layers, 5 reference folds)



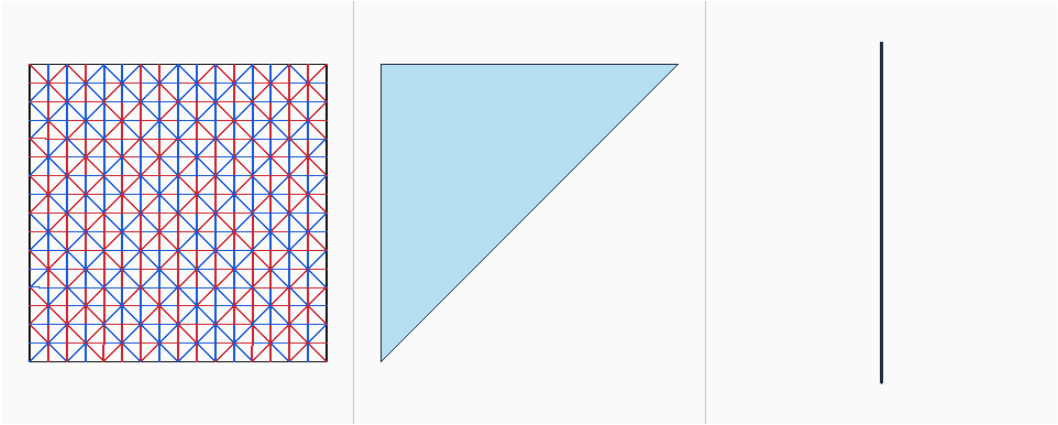
all-layers/hard-0004 (all-layers, 15 reference folds)



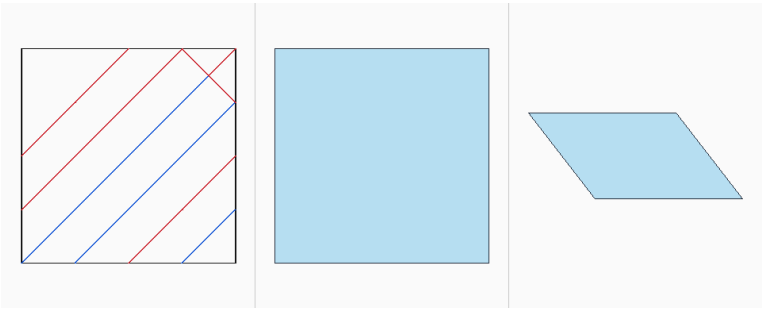
all-layers/hard-0005 (all-layers, 15 reference folds)



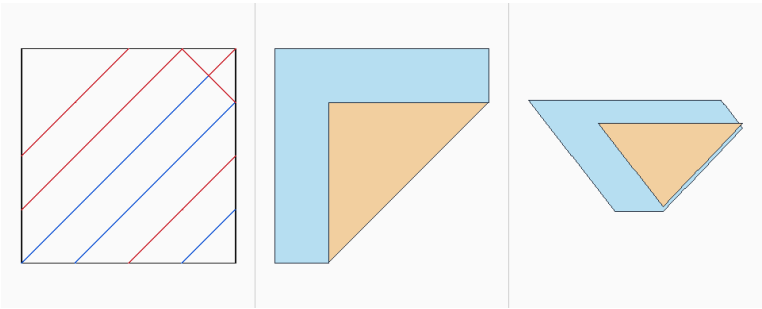
all-layers/hard-0006 (all-layers, 15 reference folds)



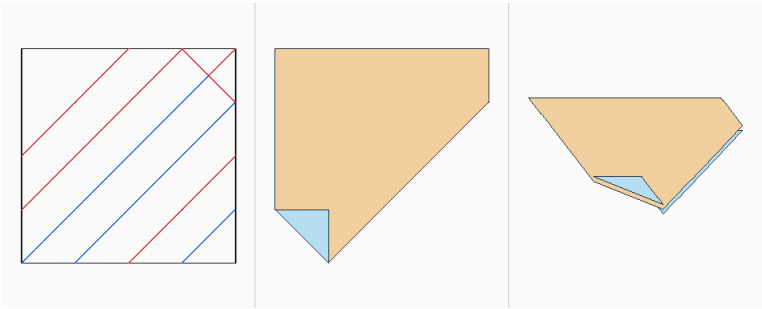
all-layers/easy-0001 (all-layers, 4 reference folds) Stored state 0



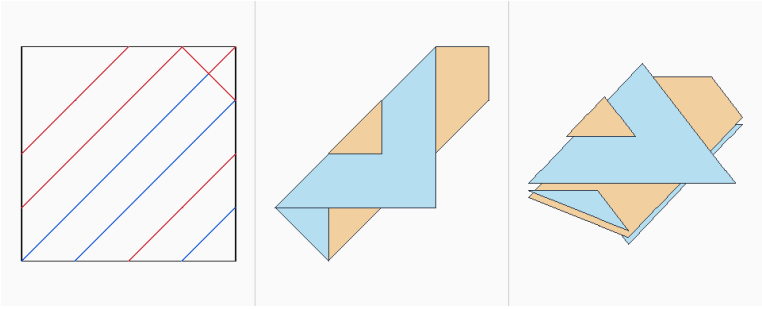
Stored state 1



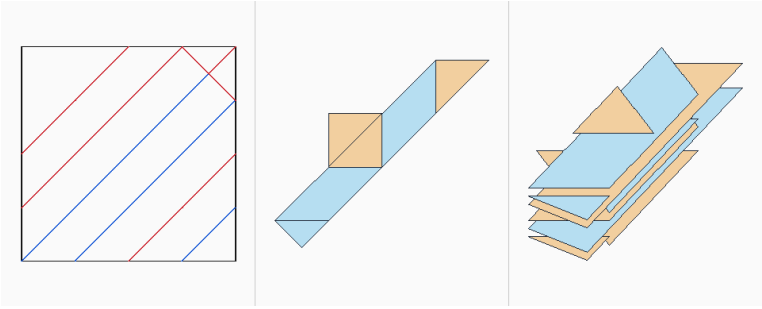
Stored state 2



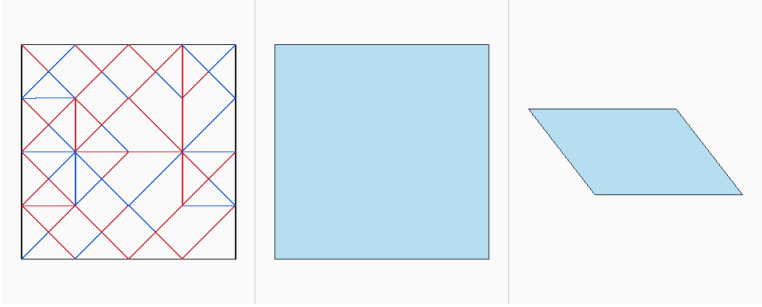
Stored state 3



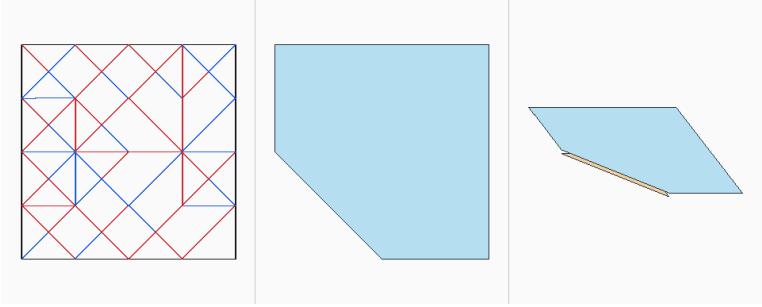
Stored state 4



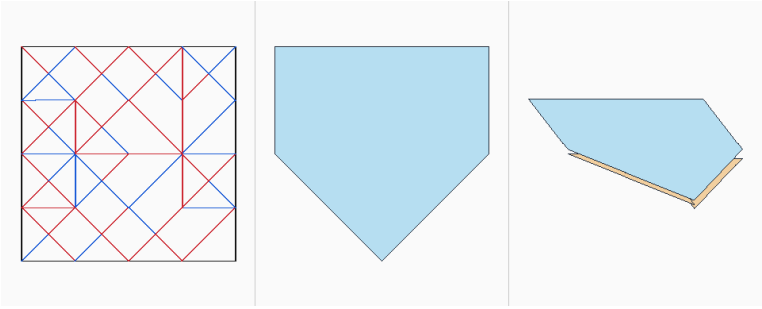
all-layers/mid-0004 (all-layers, 11 reference folds) Stored state 0



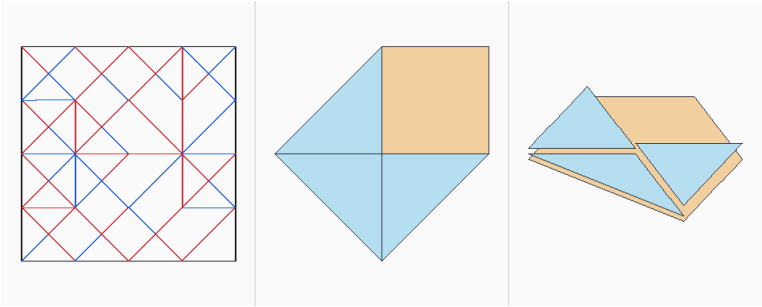
Stored state 1



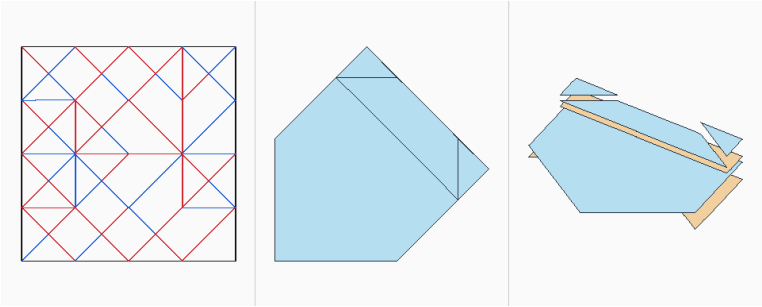
Stored state 2



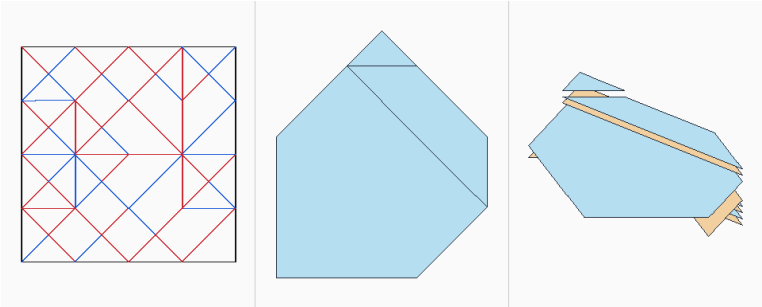
Stored state 3



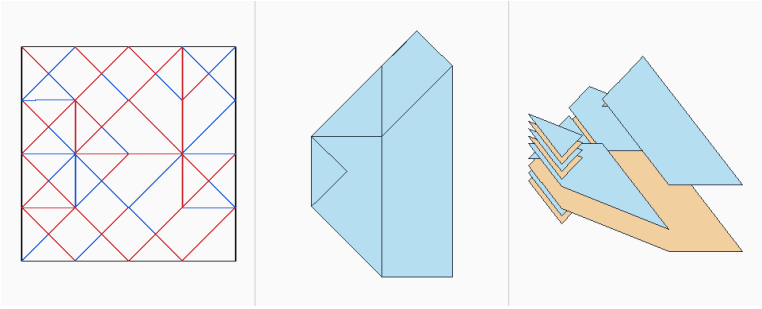
Stored state 4



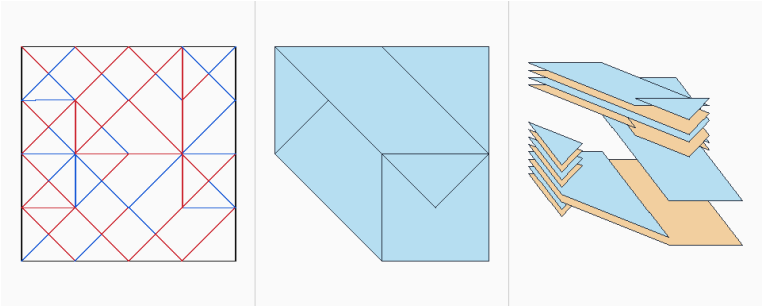
Stored state 5



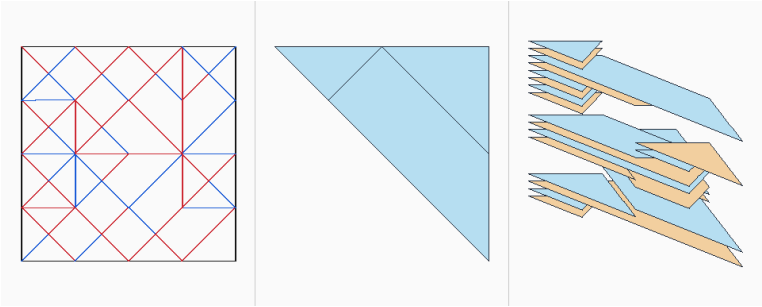
Stored state 6



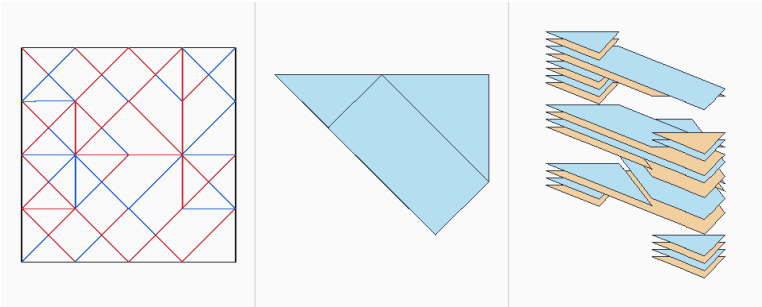
Stored state 7



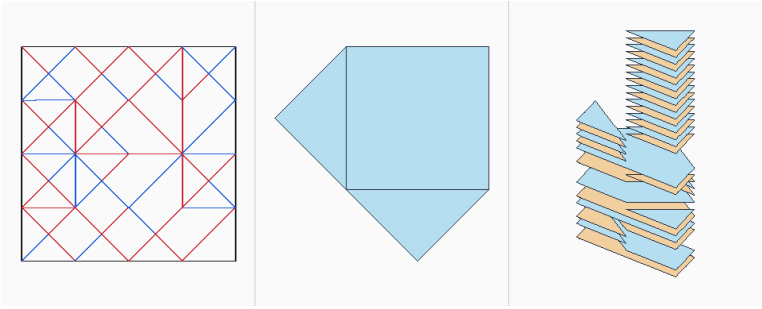
Stored state 8



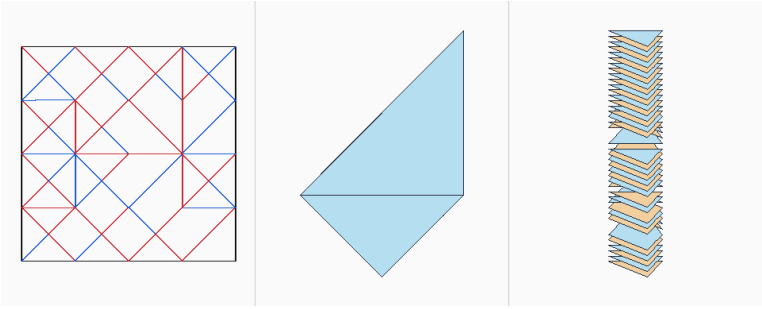
Stored state 9



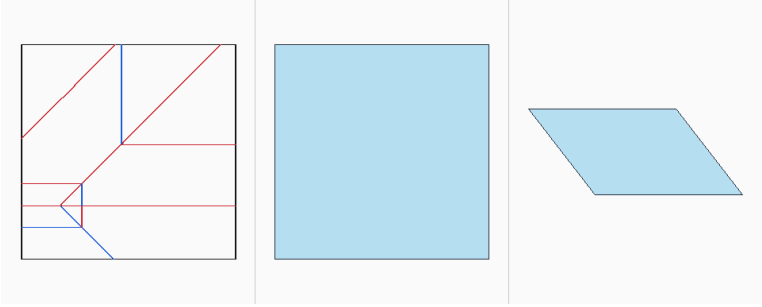
Stored state 10



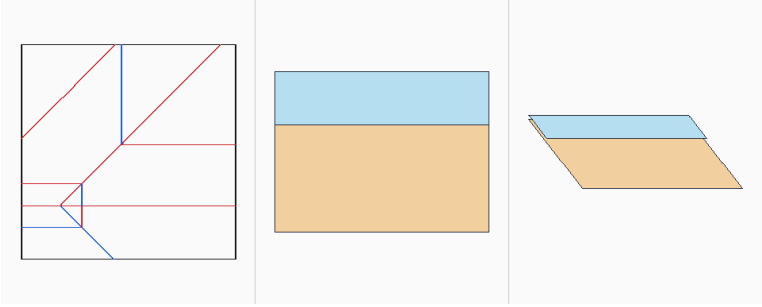
Stored state 11



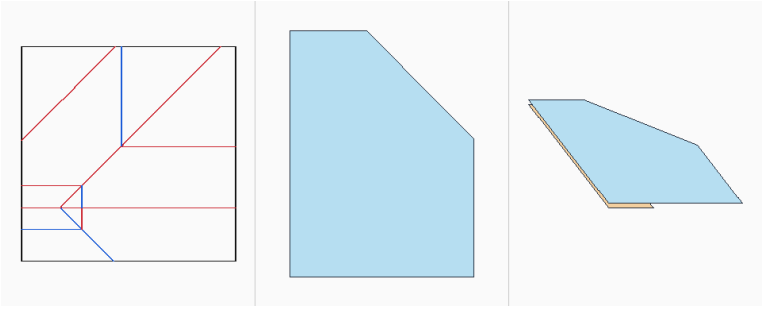
some-generated-d5/layers-0001 (some-layers, 5 reference folds) Stored state 0



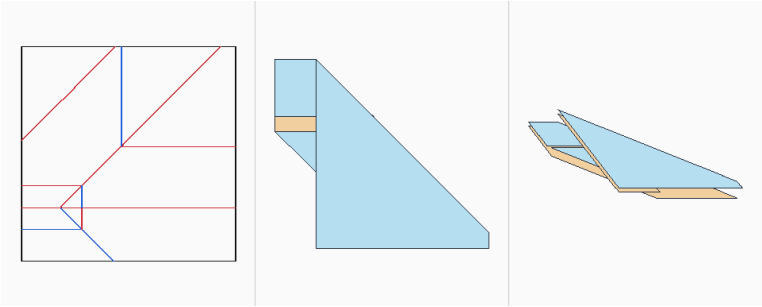
Stored state 1



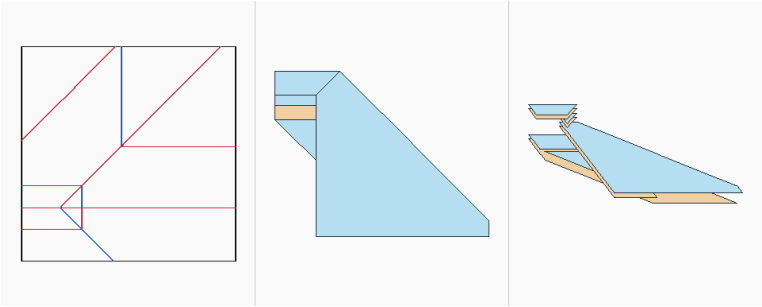
Stored state 2



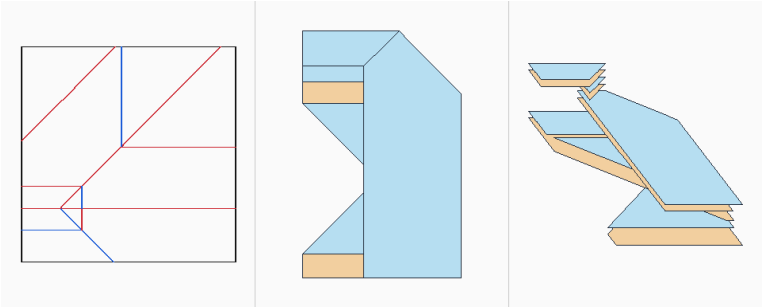
Stored state 3



Stored state 4



Stored state 5



B.9 TRACE-ANALYSIS INTERFACE

We provide a trace-inspection interface for browsing saved runs, replaying recorded folding trajectories, and examining model actions alongside tool feedback and rendered states. This supports qualitative analysis of successful construction, repeated exploration, and unsuccessful termination. A command-line trace viewer additionally supports filtering by task, trial, session, role, and event type.

Figures 7 and 8 show the dataset browser and trace viewer, respectively. These interfaces support inspection of reference sequences and recorded model episodes.

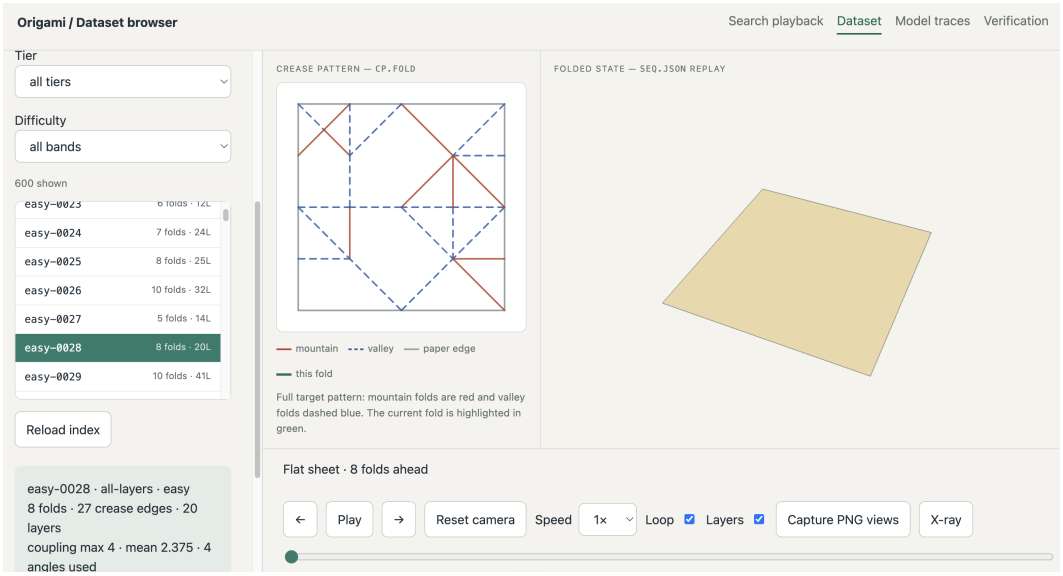


Figure 7: Dataset browser with `easy-0028` selected. The interface displays sample metadata, the target crease pattern, and reference-sequence playback controls. The right panel shows the unfolded sheet at step zero.

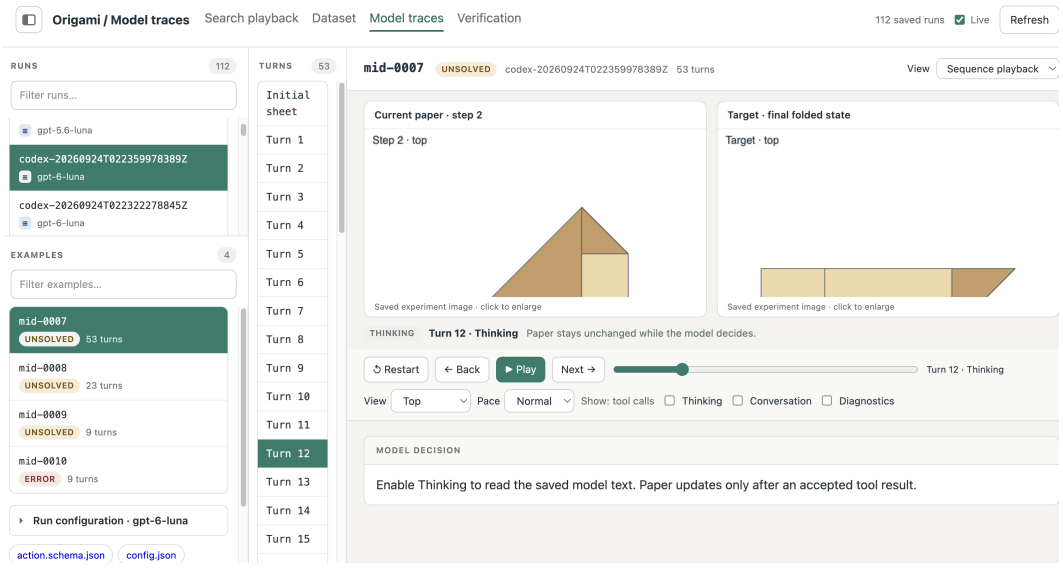
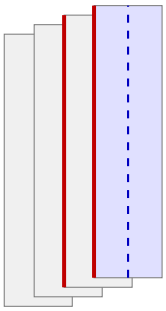


Figure 8: Trace viewer for a saved GPT-6 Luna attempt on `mid-0007`. Run and turn navigation appear beside the current and target states. The selected episode is marked unsolved; the displayed frame is at turn 12. Playback controls allow inspection of saved actions and feedback.

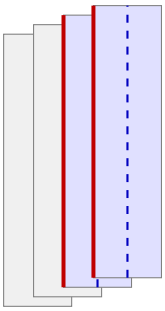
(a) Rejected proposal



`would-tear`

Top leaf selected; state unchanged.

(b) Accepted proposal



`ok: true`

Connected top pair selected.

Figure 9: Rejection feedback and a legal alternative from the same state. Dashed blue lines mark the proposed hinge; red segments mark the shared material edge identified by the tearing diagnostic. Blue shading identifies selected layers. Selecting only the top leaf would separate it from its connected neighbour away from the hinge; selecting the connected top pair permits the fold. Both panels show the pre-action state, with illustrative layer separation. The example is rendered from verification fixtures.

B.10 Tool loading and configuration

The schema builder selects the tools exposed in each run. Basic mode excludes candidate enumeration; filtered mode includes it. Target comparison is added only when a comparison level is configured. For an all-layers action model, partial-layer selection arguments are removed from the exposed action schemas. The run configuration therefore records tool mode, comparison level, and action model separately.

B.11 Rejection feedback

Figure 9 illustrates the geometric evidence associated with a `would-tear` response using the shared verification fixtures. The comparison holds the initial state and hinge fixed and changes the selected layers. The export script records both engine responses and checks that rejection preserves the state.

B.12 Supplementary results

Table 8: Earlier GPT-5.6 Luna evaluation at low reasoning effort. Counts are solved attempts over completed attempts, including repeats across samples and budgets.

Condition	Solved / completed	Solve rate
Basic tools	4/67	6.0%
Legal enumeration	31/272	11.4%
Legal enumeration + on-demand comparison	28/108	25.9%

The inventory also contains small exploratory runs: GPT-6 Sol with legal enumeration solves 1/3 low-effort attempts and 1/1 high-effort attempt; GPT-6 Astra with basic tools solves 1/1 low-effort attempt. Earlier basic-tool runs record 0/7 for GPT-5.6 Sol and 0/1 for GPT-5.6 Terra. One additional completed basic-tool attempt has no recorded model identity and is unsolved. These exploratory runs are reported separately from the Luna ablations.

A separate scan of 1,421 saved attempts with recoverable states gives a mean crease-pattern insertion/deletion distance of 45.5, or 0.55 of the target crease count (Table 11). These averages pool models, baselines, configurations and repeated samples, and predate the ablations.

These attempt-weighted averages pool saved configurations and repeated samples; they are descriptive and do not constitute matched model comparisons. The CP-distance scan may cover more runs than the earlier solve-rate snapshot.

Table 9: How each Luna ablation episode ended. Solved: correct submission. Wrong: incorrect submission. Cycling: stopped after revisiting earlier states. Dead end: no recoverable legal continuation. The five protocol failures of Appendix B.13 have no completed episode and are not listed, so rows can sum to fewer than 40.

Model	Effort	Tools	Solved	Wrong	Cycling	Repetition	Turn budget	Dead end
GPT-5.6 Luna	Low	B	2	6	14	3	5	10
	Low	L	6	12	22	0	0	0
	Low	A	10	2	27	0	0	0
GPT-6 Luna	Low	B	0	2	12	20	1	5
	Low	L	2	10	25	1	0	0
	Low	A	8	0	29	0	0	1
GPT-6 Luna	High	B	6	4	13	5	5	7
	High	L	10	8	21	0	1	0
	High	A	11	1	28	0	0	0
Total			55	45	191	29	12	23

Table 10: Easy-group solve rate by reference fold count, over all completed model attempts in the inventory. Attempts include repeats and differing budgets, so rates are descriptive.

Reference folds	4	5	6	7	8	9	10
Solved / attempts	19/59	35/115	12/46	10/97	3/60	0/33	7/47
Solve rate	32.2%	30.4%	26.1%	10.3%	5.0%	0.0%	14.9%

B.13 Protocol failure accounting

All 360 planned Luna ablation cases enter the solve-rate denominator. Of these, 355 have completed episodes. Two cases end at the request input-size limit: `mid-0006` for GPT-5.6 Luna low with automatic feedback and GPT-6 Luna low with legal enumeration. GPT-6 Luna low with legal enumeration on `mid-0010` ends in a service prompt rejection. For GPT-6 Luna low with automatic feedback, `mid-0006` and `mid-0010` were omitted from the restarted batch after these blocking errors. These two cases are classified as omissions after blocking errors; no direct service error is recorded for them in that condition. All five cases count as unsuccessful evaluations under the planned protocol.

Comparison-tool defect. The target-comparison tool aligns stacks by their bottom layer. When no rigid motion aligned the two bottom layers and the layer counts were equal, it reported the diagnosis *matches* with zero matching layers. Scoring uses a separate comparator and was not affected, so every reported solve is correct. Replaying the final state of every incorrect submission made under a comparison condition shows that five followed this false report: `easy-0001` in the automatic-feedback ablations for GPT-5.6 Luna (low) and GPT-6 Luna (high), and `easy-0001`, `easy-0058` and `easy-0095` in the earlier on-demand comparison runs. These episodes are reported as measured; the comparison conditions are therefore slightly understated: by at most one solve in each affected ablation cell, and at most three of 108 attempts in the earlier on-demand condition. The defect is fixed in the released harness, and all 536 solved episodes in the supplementary runs still receive *matches* from the corrected tool.

C Extended related work

This appendix collects the literature the benchmark rests on, grouped by the role each work plays rather than alphabetically. Section 2 discusses the works the design depends on directly; the rest are recorded here.

Origami benchmarks and systems for language models. Spencer et al. (2025) releases both code and data, and evaluates viewpoint consistency and the recognition of infeasible patterns. Xu et al. (2025) evaluates pattern prediction, multi-step spatial reasoning and crease-pattern code generation, and ships an interactive environment for the last of these. Agarwal et al. (2026) synthesises flat-foldable patterns in an interactive environment over 366 designs collected from Flat-Folder, and evaluates query efficiency, geometric similarity through intersection over union, and semantic similarity through CLIP embeddings; its query efficiency is the proportion of fold steps that contribute to the final model, not the number of queries to a solution. It is one source of the metrics surveyed while our scoring was designed. Zahavy et al. (2026) co-designs flat-foldable and recognisable models, using a vision-language model for aesthetic and semantic evaluation and reporting the flat-foldability of generated crease patterns separately. Huang et al. (2026) plans structured generation against a learned world model. Moriya et al. (2026a) reconstructs folding processes from instructional video, over the sequence-level corpus of Moriya et al. (2026b).

Table 11: CP insertion/deletion distance over saved attempts. Means exclude unscored attempts. Normalization divides each distance by its target merged crease count before averaging.

Model	Attempts	Scored	Mean distance	Mean normalized
claude-sonnet-5	5	5	12.800	0.481
codex-cli-chatgpt	1	1	8.000	1.000
deterministic-bfs	725	725	38.810	0.494
deterministic-reverse	203	203	24.212	0.764
gpt-5.6-luna	447	447	67.159	0.538
gpt-5.6-sol	7	7	15.857	0.586
gpt-5.6-terra	1	1	0.000	0.000
gpt-6-astra	1	1	0.000	0.000
unknown	32	31	43.065	0.804
All attempts	1422	1421	45.454	0.553

Complexity and theory of flat folding. Bern & Hayes (1996) carries the claim the task rests on: geometry does not determine layer order, and deciding the overlap order of a flat folding is NP hard even when a valid mountain-valley assignment is given. Arkin et al. (2004) defines simple foldability and the one-layer, some-layers and all-layers models this action space is drawn from, and Akitaya et al. (2017) settles the complexity of deciding it for square paper with creases at multiples of 45°. Akitaya et al. (2020) treats the all-layers model, which is also the model of sheet-metal bending, and Akitaya et al. (2023) treats mixed orthogonal crease patterns. Akitaya et al. (2026) gives a method for deciding whether a pattern admits a flat-folded state at all, a problem that is NP hard in general. Ku et al. (2026) gives an algebraic treatment of the layer-ordering constraints, and Eppstein (2019) studies the graphs of flat foldings themselves. Demaine et al. (2004) and its predecessor Demaine & Mitchell (2001) separate a folded state from a continuous folding motion, for simple polygonal and rectangular paper respectively. Those results concern continuous motion; they do not establish that a folded state is reachable by the restricted sequence of simple folds used here, and the discrete step structure is the question this benchmark poses. Hull & Zakharevich (2023) establishes that flat origami is Turing complete, which bounds what any general claim about foldability can promise.

The crease-pattern-to-sequence problem itself. Akitaya et al. (2013) named the problem and supplied the tools the area still uses: reflection paths, graph rewriting, and a step sequence graph whose paths yield folding sequences. xkevio (2026) is an open-source implementation of that line and is usable as a symbolic baseline, which we have not yet run.

Counting folded states. The number of ways to fold a strip of n labelled stamps runs 1, 2, 6, 16, 50, 144, 462, 1392, 4536, 14060 for $n = 1, \ldots, 10$ (OEIS Foundation Inc., 2026). The problem was posed by Lucas (1891) in 1891 and has been attacked repeatedly since (Koehler, 1968; Lunnon, 1971); the related enumeration of meanders is surveyed by Legendre (2013), which reports that no closed formula was known. The relevance here is that the number of valid states grows fast enough that scoring against any single reference sequence would be wrong, which is why Section 4.2 scores by execution instead.

Spatial reasoning context. Liu et al. (2025) surveys spatial reasoning in multimodal models. Wu et al. (2024) names the mental-imagery question this benchmark turns into a measurement: whether a model can maintain and update a spatial representation across steps rather than describe one.

Statistical physics of random flat foldability. Nakajima (2024) represents layer-order constraints as spin interactions, and Hull et al. (2025) proves that for random locally flat-foldable mountain-valley assignments on the square grid, global flat foldability is exponentially unlikely under its sampling distribution. Neither result concerns the number of terminal states of the patterns in this corpus, and we do not extrapolate them to it.

Tools, formats and data. Ku (2022) is the implementation that decides and enumerates flat-folded states, for the method of Akitaya et al. (2026); together they are the source of the four constraint classes cited in Section 4.1. Demaine et al. (2026) is the de facto standard used by every artifact we ship. Mitani (2007) is a crease-pattern editor with folded-form estimation. Moriya et al. (2026b) is the closest existing sequence-level corpus, with 27 sequences and 337 annotated keyframes released under CC-BY-4.0 as the ground truth for FoldingAgent; it is not used here as training or evaluation data.

Scope, and what is left out. Thick folding (Ku & Demaine, 2016), bar-and-hinge mechanics (Filipov et al., 2017) and its simulators (Zhu & Filipov, 2021; Zhu, 2026), and the wider creased-sheet mechanics literature belong to a different line of work, one in which the sheet has thickness and the verifier is a simulation

rather than a decision. That line asks a question this benchmark does not: the states here are zero-thickness and flat, and the environment rules on a fold in one pass rather than integrating a deformation. Those works are therefore not treated as neighbouring work, and the limits of the model used here are stated in Section 6.