

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 GIFARC: SYNTHETIC DATASET FOR LEVERAGING HUMAN-INTUITIVE ANALOGIES TO ELEVATE AI REA- SONING

006 **Anonymous authors**

007 Paper under double-blind review

011 ABSTRACT

013 The Abstraction and Reasoning Corpus (ARC) poses a stringent test of general AI
014 capabilities, requiring solvers to infer abstract patterns from only a handful of ex-
015 amples. Despite substantial progress in deep learning, state-of-the-art models still
016 achieve accuracy rates of merely 40-55% on the 2024 ARC Competition, indicative
017 of a significant gap between their performance and human-level reasoning. In this
018 work, we aim to bridge that gap by introducing an analogy-inspired dataset called
019 GIFARC. Leveraging vision-language models (VLMs), we synthesize new ARC-
020 style tasks from a variety of GIF images that include analogies. Each new task is
021 paired with a ground-truth analogy, providing an explicit mapping between visual
022 transformations and everyday concepts. By embedding robust human-intuitive
023 analogies into ARC-style tasks, GIFARC guides AI agents to adopt analogical rea-
024 soning approaches, facilitating more concise and human-understandable solutions.
025 We empirically demonstrate that GIFARC improves task-solving performance by
026 aligning model reasoning with human analogical problem-solving strategies.¹²

027 1 INTRODUCTION

030 Despite recent advances in AI, many of current AI models remain limited in their capacity for flexible,
031 general-purpose reasoning, which is a core characteristic often associated with artificial general
032 intelligence (AGI). In this context, the Abstraction and Reasoning Corpus (ARC) was introduced
033 as a benchmark aiming to measure the ability of artificial general intelligence systems to abstract
034 visual patterns and reason about them in a manner akin to human cognition (Chollet, 2019). This
035 requirement closely mirrors the cognitive flexibility inherent to human task-solving, thus making
036 ARC an essential playground for probing the fundamental limitations and capabilities of current AI
037 systems in their pursuit of human-level AGI.

038 In the 2024 ARC Prize competition and semi-evaluation, various deep neural networks achieved
039 strong performance, such as J.Berman (Berman, 2025) and E.Pang (Pang, 2025) reached 79.6%
040 and 77.1% respectively. However, most models saw their performance reduced by more than half
041 on ARC-AGI-2, which lacks data generators like ReARC (Hodel, 2024) for training on similar
042 tasks. This reveals a fundamental limitation: while deep neural networks excel with ample data, they
043 struggle in example-deficient tasks requiring abstract and compositional reasoning (Dziri et al., 2023;
044 Lee et al., 2024). ARC thus provides strong evidence that data-driven approaches alone do not yet
045 capture the breadth of human reasoning and generalization.

046 Why does this limitation exist? One crucial factor lies in how humans leverage analogies to map
047 new visual or conceptual clues onto existing knowledge. When people see a grid transformation as
048 in Figure 1, they can instantly map this visual pattern onto familiar concepts such as gravity and
049 color change. These priors, accumulated from a lifetime of diverse visual and conceptual experiences,
050 provide immediate insight into the task constraints and sharply reduce the complexity of the tasks.
051 Rather than searching blindly through every possible transformation, humans can reject lengthy or
052 counterintuitive candidates and focus on the small subset of patterns consistent with their analogies.

¹Tasks generated with GIFARC are visualized in <https://gifarc.vercel.app/>.

²Full dataset is available at https://huggingface.co/datasets/DumDev/gif_arc.

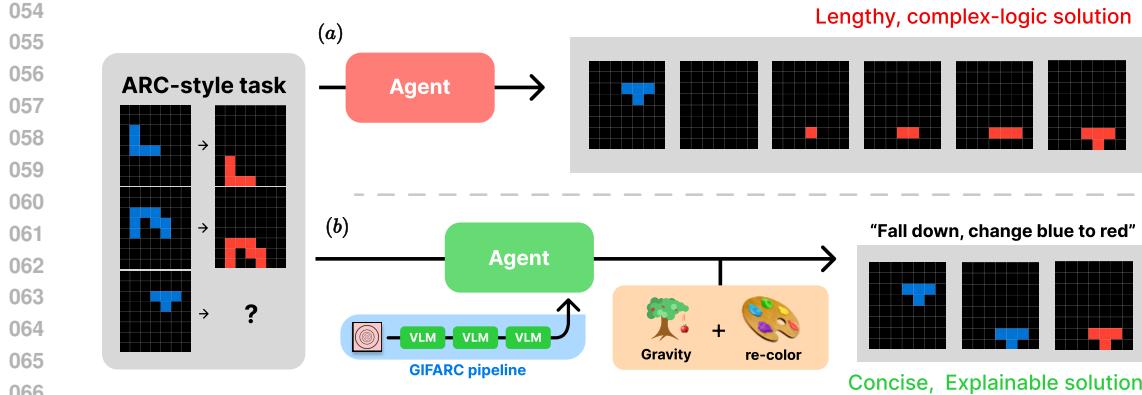


Figure 1: Illustration of two different solutions of ARC-style task found with or without analogic approach. In general, humans will perceive the solution of (b) better than (a), as it is more concise and easier to understand. When the agent has been guided to think analogically with GIFARC dataset, as depicted in (b), it seeks a more concise and human-intuitive solution.

In contrast, deep learning models rarely have an explicit mechanism for analogy-making (Mitchell, 2021). Even advanced neural networks typically rely on statistical correlations learned from large training sets, lacking the means to “tag” or “reframe” new tasks in terms of previously understood scenarios (Bober-Irizar & Banerjee, 2024). Consequently, these models miss a powerful heuristic: an analogy-driven search strategy that narrows down the solution space by linking visual abstractions to real-world events.

Need of analogy-contained dataset. To help bridge this gap between human cognition and AI, the agent should obtain human-level amount of analogies in prior to any task-specific training. Since the models we aim to train are predominantly data-driven, the need for training dataset that consists of useful analogies naturally arise. The dataset we seek contains following criteria;

- (i) *ARC-style examples*: the dataset should contain analogies in forms of ARC-style examples. Since the agent is evaluated via ARC benchmark, the agent has the world perception in form of 2D grids. In addition, the analogies one needs to know should be expressible on an ARC-style 2D grid. Thus, the analogy we intend to train should also be expressed and given in the 2D grid in order for the agents to understand and utilize them during evaluation.
- (ii) *Executable codes*: For the agent to capture how to manipulate 2D grids in accordance with the analogy, the transformation should be provided as lines of executable code. With these 2D grids and codes, the agent should be able to correlate ARC-style transformations with their underlying analogies and identify them during evaluation.

In this paper, we propose an analogy-inspired synthetic ARC dataset, GIFARC. Drawing on vision-language model (VLM), we developed a framework to synthesize ARC-style tasks from online GIF images by extracting various visual patterns and the analogies they evoke. These GIFs encompass a range of scenarios (e.g., snowfall, blooming flowers, moving objects), each providing a different type of visual transformation. We encode these transformations in newly generated ARC-style tasks and, crucially, supply ground-truth analogy labels that clarify the underlying conceptual mapping (e.g., “water flow blocked by an obstacle”).

Role of GIF as the basis of ARC-style task generation. Using GIFs as the starting ingredient of ARC-style tasks, our GIFARC synthesis pipeline takes great advantages in picking human-understandable and visualizable analogies, as GIFs are series of images that are intentionally drawn to visualize certain ideas. In addition, GIF also has advantages in holding a sufficiently small number of analogies. In terms of analogy extraction simplicity, it is critical for the analogies to be implied in the image series without any entanglement to one another. Longer image series such as videos are more likely to hold excessive number of analogies and consequently have a higher risk of entanglement in between. Thus, GIF is an adequate form of analogy-implying image series.

108 These new ARC-style tasks generated from GIFs will possess a variety of intuitive analogies. The
 109 agent trained with this dataset will be guided to gain the grasp of analogical approaches and utilize
 110 them in the solving step. By incorporating diverse analogies into the dataset, model trained with
 111 GIFARC is guided to “think analogically” before searching for solutions, thereby reducing the
 112 complexity of task-solving. In addition, by converting the visual data to verbose expression before
 113 generating the ARC-style tasks, the agent will specifically gain the ability to verbally point out the
 114 analogy intended in the ARC-style task.

115 Running this pipeline on motion-rich GIFs produces 10k unique high-quality ARC-style tasks
 116 spanning 20 analogy categories. We empirically validated GIFARC through two complementary
 117 approaches. First, in-context learning experiments demonstrated that GIFARC effectively guides
 118 LLMs toward human-like analogical reasoning, as models using full analogical descriptions achieved
 119 better alignment with ground-truth analogies than those using analogy-removed descriptions. Second,
 120 supervised fine-tuning demonstrated quantitative improvements, with accuracy increasing from 0.2%
 121 to 2.9% on ARC-AGI-1 and negative log-likelihood decreasing by 94.5% on ARC-AGI-2.

2 RELATED WORK

123 **Abstraction and Reasoning Corpus.** The Abstraction and Reasoning Corpus (ARC), introduced by
 124 Chollet (2019), is an open-ended benchmark designed to evaluate broad generalization from minimal
 125 demonstrations. Its goal is to test how efficiently an agent can acquire and apply abstract patterns
 126 with just a few examples. In 2020, there have been many attempts to solve ARC through Domain
 127 Specific Language (DSL) based brute force programs or various search algorithms. The limitation of
 128 these methodologies was that they required manually written DSLs that humans deemed necessary.
 129 Since the release of GPT-4 in 2023, the performance of LLMs has been improved significantly,
 130 leading to various attempts to solve ARC problems using LLMs (Mirchandani et al., 2023; Xu et al.,
 131 2024; Lee et al., 2024). In 2024, there were two LLM utilized approaches. The first approach,
 132 similar to previous methods, involved LLMs directly predicting test outputs. ARCitect (Franzen
 133 et al., 2024) won first place in the 2024 ARC Prize by fine-tuning models using data augmented with
 134 ReARC (Hodel, 2024) and employing specialized decoding strategies for predicting test outputs.
 135 Additionally, OpenAI’s reasoning models, such as o1 and o3, showed significant performance despite
 136 only applying CoT, unlike the approaches from 2023. The second approach using LLMs generated
 137 Python code solutions with LLMs (Li et al., 2024; Berman, 2024). Berman (2024) generated over 250
 138 Python codes from LLMs for a single problem and refined them to create an optimal solution code.
 139 However, this approach has the disadvantage of requiring high costs due to the need to generate many
 140 code candidates. In 2025, a new version of the ARC benchmark named ARC-AGI-2 (Chollet et al.,
 141 2025) was released, which requires more abstract and analogical thinking compared to the existing
 142 ARC-AGI-1. Former ARC-AGI-1 winning models struggled to solve the new version benchmark,
 143 implying that the prior approaches still have room for development.

144 **Synthetic ARC Generation.** To address the data scarcity inherent in ARC, which hinders both
 145 symbolic program synthesis and the training of neural models, researchers have turned to generating
 146 programmatic data using human-curated, AI-generated, or hybrid pipelines (LeGris et al., 2024;
 147 Opiełka et al., 2024; Qi et al., 2021). One such effort is ReARC, which manually scripts generative
 148 procedures for each task to produce an effectively infinite set of in-distribution examples (Hodel,
 149 2024). LARC (Language-Complete ARC) crowd-sources natural language explanations of tasks,
 150 enabling models to learn from natural programs instead of raw grids (Acquaviva et al., 2022). Other
 151 benchmarks such as ConceptARC, Mini-ARC, and 1D-ARC use manually crafted rules to produce
 152 diverse yet interpretable task distributions (Kim et al., 2022; Moskvichev et al., 2023; Xu et al.,
 153 2024). Recent approaches automate this process at scale. For instance, BARC generates 400,000
 154 training examples by prompting GPT-4 to mutate a curated set of 160 Python-based solvers (Li et al.,
 155 2024). Similarly, MC-LARC transforms ARC tasks into multiple-choice formats, enabling scalable
 156 evaluation of language models’ intermediate reasoning stages (Shin et al., 2024).

157 Different from existing in-distribution resampling or code-mutation pipelines, our work introduces
 158 GIFARC, a new dataset that mines human-intuitive GIFs, distills their analogical structure with
 159 vision-language models, and auto-compiles each analogy into an executable ARC-domain specific
 160 language (ARC-DSL) code, explicitly teaching models how to think by analogy rather than simply
 161 giving them more examples to memorize.

162 3 METHOD

163 3.1 OVERVIEW

164

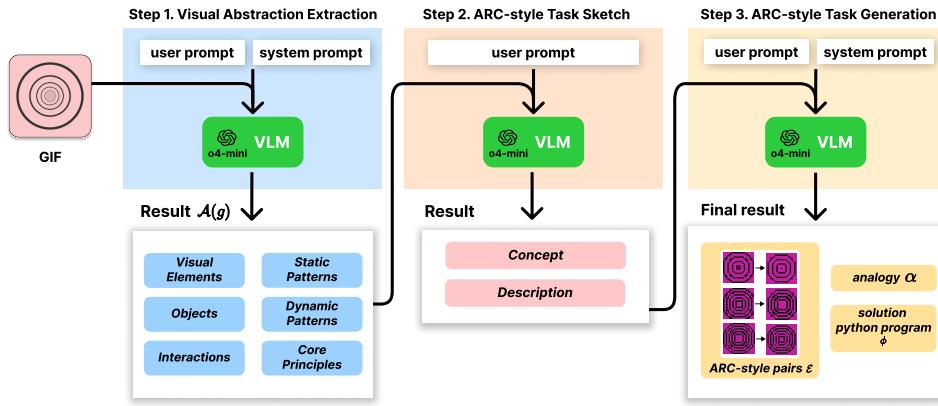
165 **Problem formulation.** Given a set of GIF images $\mathcal{G} = \{g_j\}_{j=1}^N$, our goal is to construct an
 166 analogy-grounded ARC dataset, GIFARC, containing ARC-style tasks and represented as triples
 167 $\mathcal{T} = (\mathcal{E}, \alpha, \phi)$ where (i) $\mathcal{E} = \{(x_i, y_i)\}_{i=1}^K$ is a set of K input-output grid pairs of ARC-style task.
 168 Here, $x_i \in \{0, \dots, 9\}^{h_{x,i} \times w_{x,i}}$ is the input grid, $y_i \in \{0, \dots, 9\}^{h_{y,i} \times w_{y,i}}$ is the output grid, with
 169 $h_{G,i}, w_{G,i}$ respectively being the height and width of the G grid (either grid x or y) in i -th pair of \mathcal{E} ;
 170 (ii) α is a short natural-language analogy description (e.g., “blocked water flow”). (iii) ϕ is a Python
 171 program implementing $\mathcal{F} : \mathcal{X} \rightarrow \mathcal{Y}$, the latent deterministic transformation such that $y_i = \mathcal{F}(x_i)$;
 172 GIFARC ($\mathcal{D} = \{\mathcal{T}_j\}_{j=1}^N$) therefore pairs each ARC task with its provenance and analogy.
 173

174 **Pipeline summary.** Our data synthesis pipeline is a three-stage process that transforms raw GIFs
 175 into fully executable, analogy-grounded ARC-style tasks (see Figure 2 for illustrative examples).
 176



179 Step 1. A VLM (o4-mini) analyzes each GIF g and outputs a structured JSON record that captures
 180 its objects, static/dynamic patterns, interactions, and core reasoning principles $\mathcal{A}(g)$.
 181 Step 2. A VLM (o4-mini) converts the JSON abstraction into a concise ARC-style task sketch: a
 182 set of concepts with a natural-language description that implies the extracted patterns.
 183 Step 3. Conditioned on the sketch and a handful of retrieved ARC task examples, the VLM (o4-mini)
 184 synthesizes Python code implementing `generate_input` (stochastic examples) and
 185 `main` (deterministic transformation). The output is a complete ARC-style task comprising
 186 (i) an input-output demonstration set \mathcal{E} , (ii) the Python solution program ϕ , and (iii) its
 187 analogy label α .
 188

189 Details of Step 1 appear in Section 3.2, Step 2 in Section 3.3, and Step 3 in Section 3.4. All model
 190 calls use Azure OpenAI endpoints. The GIFARC dataset generated by this pipeline is analyzed in
 191 Appendix A.



200 Figure 2: Illustration of GIFARC data synthesis pipeline that transforms a single GIF into a corre-
 201 sponding ARC-style task and supplementary data. In step 1, o4-mini digests a GIF file and outputs a
 202 detailed text expression $\mathcal{A}(g)$ of the visual transformation implied in the GIF in a JSONL format. In
 203 step 2, o4-mini reads step 1 result JSONL and outputs a text sketch an ARC-style task. In step 3,
 204 o4-mini reads the step 2 sketch result and generates an ARC-style task \mathcal{E} , an implied analogy α , and
 205 the Python solution program ϕ . The example prompts used in each step are reported in Appendix B.
 206

213 3.2 STEP 1. EXTRACTING VISUAL ABSTRACTIONS FROM GIFS

214

215 Our first stage converts a raw GIF into a structured, readable summary that captures the analogy-laden
 216 visual logic hidden in the clip. Given an animated GIF g depicting a short visual phenomenon (e.g.,

216 snow piling up, light reflecting, pipes clogging), we wish to automatically extract the key elements
 217 $\mathcal{A}(g)$ that a human would use when reasoning by analogy: what the scene contains, which parts move
 218 or remain fixed, how objects interact, and which general principle explains the observed dynamics.
 219 The output, called a visual abstraction, serves as the input for the later stage.
 220

221 **GIF crawl.** To maximize coverage of distinct visual regularities, we first assemble a visual transfor-
 222 mation corpus of GIFs. Using the GIPHY API, we query a curated tag list {nature, geometry,
 223 pattern, relational_pattern, sequence} where these types of clips often exhibit
 224 dynamic change. In particular, we crawled those with G-rated ratings, to generate nonharmful
 225 ARC-style tasks and prevent possible refusal from the following LLM APIs throughout the pipeline.
 226 After removing duplicates, we obtained 10k GIFs in total.
 227

228 **Prompt for extracting visual abstractions.** We then submit every GIF to the vision-language
 229 model o4-mini under a prompt intentionally designed around two objectives: (i) *granularity* - to
 230 extract as many diverse visual analogies as possible from the GIF, (ii) *structure* - the reply must be
 231 valid JSON, and (ii) *coherence* - to ensure that each extracted visual analogy maintains alignment
 232 with the original GIF. Full prompt used in this step is reported in Appendix B.1. The system message
 233 states the structural constraint (“return JSON with keys {visual elements ... core principles} and
 234 use bullet lists only”). The user message enumerates the six fields and attaches concrete bullet-level
 235 instructions.
 236

237 These six fields in $\mathcal{A}(g)$ serve complementary purposes: (i) `visual_elements` stores the raw,
 238 static pixel facts; (ii) `objects` binds memorable names to entities so later prompts can re-
 239 ference them; (iii) `interactions` records explicit or implicit relations (e.g., “snowflake col-
 240 lides with ground”, “fluid flow is blocked”) among objects. (iv) `static_patterns` and (v)
 241 `dynamic_patterns` distinguish invariants and variants from transformations, respectively; (vi)
 242 `core_principles` captures the very abstraction (e.g., gravity, reflection, occlusion, ...) that we
 243 ultimately hope an ARC solver will recall as an analogy.
 244

245 This step is an extraction step to pull out intuitive analogies from GIF image. Converting pixel data
 246 into a more lightweight text descriptions lets the pipeline to efficiently digest the analogy faster
 247 through the remaining synthesis stages without having to handle heavy image data along. Also, in
 248 case that GIF holds multiple visual analogies, explicitly pointing out which visual analogy to focus
 249 on lets the agents in later stages interpret the target analogy without possible confusion. Furthermore,
 250 this stage is critical in abiding the GIPHY API terms of right. By irreversibly conceptualizing visual
 251 data into text description, we have generated a distinct visual analogy data pool that does not contain
 252 any explicit GIFs.
 253

254 3.3 STEP 2. FROM VISUAL ABSTRACTION TO TASK SKETCH

255 The visual abstraction $\mathcal{A}(g)$ obtained in previous step is rich but verbose; it contains a lot of observable
 256 details of the GIF, many of which may be irrelevant for task design (e.g., background textures, camera
 257 shakes). Before we can synthesize executable code, we therefore distill $\mathcal{A}(g)$ into a compact *task*
 258 *sketch* composed of two parts: (i) **Concepts:** 3-5 keyword phrases that name the core visual ideas
 259 (e.g., `blocked_flow`, `rotational_symmetry`, `incremental_accumulation`, ...); (ii) **Description:** a single
 260 paragraph that narrates how those concepts manifest on an ARC-style grid.
 261

262 **Prompt for task sketch.** We obtain each sketch from a single completion of the vision language
 263 model o4-mini using a three-block prompt: **Block 1** provides in-context demonstrations; we draw 75
 264 concepts/description pairs at random from the 160 human-written example tasks released with the
 265 BARC corpus (Li et al., 2024) and paste them. Because these examples cover symmetry, counting,
 266 physical analogy, topological change, and more, they implicitly teach the model what “looks like” a
 267 solvable ARC-style task while leaving room for new content. **Block 2** provides the six fields of $\mathcal{A}(g)$
 268 from the previous step in bullet form. **Block 3** has a message that freezes the output format, exactly
 269 two comment headers for concepts and description.
 270

271 We further instruct the model to reflect the full trajectory from initial to final state, maintaining
 272 semantic fidelity between the GIF dynamic behavior and the input-output structure of ARC-style grid.
 273 Full prompt used in this step is reported in Appendix B.2.
 274

270 This selection step is crucial in two aspects. First, we expect VLM to filter out abstractions that
 271 cannot be turned into deterministic grid transformations (e.g., ‘clouds forming random shapes’).
 272 Second, it forces the retained content into the exact format expected by our code-generation prompt
 273 via in-context learning, thereby lowering entropy and reducing failure cases downstream.
 274

275 3.4 STEP 3. GENERATING EXECUTABLE ARC-STYLE TASKS

277 The final stage transforms each task sketch produced in Step 2 into a fully executable ARC-style
 278 task. The output comprises two parts: (i) `main`, a deterministic Python function implementing the
 279 latent transformation \mathcal{F} (rendered as Python code ϕ); (ii) `generate_input`, a stochastic Python
 280 function that draws fresh grid examples from the same distribution to generate input grids.

281 From this output, input grids generated with `generate_input` function are fed to the `main`
 282 function to produce corresponding output grids, thereby constructing a set of input-output grid
 283 pairs \mathcal{E} . The analogy α is copied from the task sketch created in Step 2, forming a ARC-style task
 284 $\mathcal{T} = (\mathcal{E}, \alpha, \phi)$.
 285

286 **Prompt for generating executable ARC-style tasks.** Our code-generation prompt inherits the
 287 overall structure introduced in BARC (Li et al., 2024), but differs in that the new task is grounded in the
 288 GIF-derived concepts and description produced in Step 2. To tighten this grounding while preserving
 289 the coding style that has been proven effective in earlier literature, we again adopt an in-context
 290 learning strategy, along with retrieval augmentation. Specifically, we embed both our description
 291 and the 160 human-authored example ARC-style tasks with `text-embedding-ada-002` model.
 292 The cosine similarity between sketch and examples is computed, and the top-4 semantically closest
 293 examples are pasted into the prompt before the new sketch. Each example already contains a
 294 working `generate_input` / `main` pair, so these few-shot examples show the model how high-
 295 level concepts and narrative prose translate into concise Python code. The full prompt example for
 296 generating executable ARC tasks is reported in Appendix B.3.
 297

298 4 APPLICATIONS

300 In this section, we conducted three experiments to examine: (1) how in-context learning with
 301 appropriately selected guiding examples influences the task-solving strategy of a LLM; (2) how this
 302 assists in identifying ground-truth analogies; (3) how it enhances the model performance in ARC-AGI
 303 evaluation tasks.
 304

305 4.1 VERIFICATION CHANGES IN REASONING STEPS

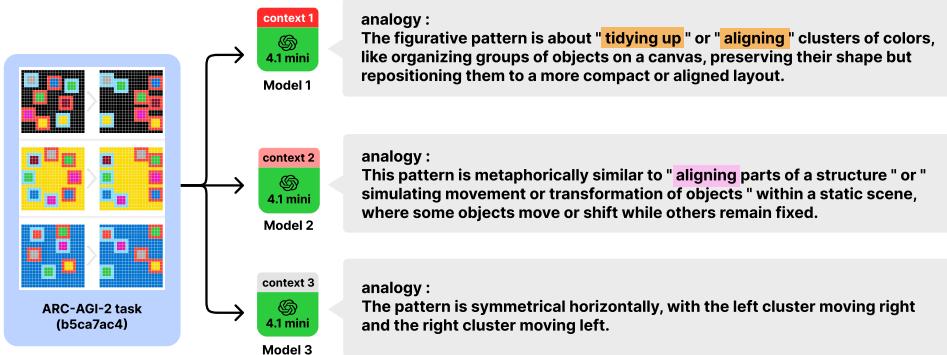
306 Is the components in GIFARC really meaningful in analogical understanding of ARC-AGI tasks?
 307 In order to check the importance of each element in the dataset, we ‘flattened’ some key parts in
 308 GIFARC dataset sample and tested if the presence of components in GIFARC show significant
 309 difference in model understanding of the task.
 310

311 We prepared three types of processed GIFARC data subset with different levels of components
 312 included:
 313

- (i) **full description:** 15 well-figuratively described data $(\mathcal{E}, \alpha, \phi)$ chosen from GIFARC through
 314 iterative refinement with LLM and human.
- (ii) **description without analogy:** 15 data $(\mathcal{E}, \alpha_{\text{flat}}, \phi_{\text{flat}})$ from full description but gone through a
 315 ‘flattening’ process where researchers replaced analogic terms in α and ϕ into their correspond-
 316 ing lower-level synonyms.
- (iii) **description without analogy and without solution:** 15 data $(\mathcal{E}, \alpha_{\text{flat}})$ from description without
 319 analogy but without solution ϕ_{flat} .

321 Note that smaller numbered description contains richer analogies. These in-context learned models
 322 were then prompted to solve the tasks of ARC-AGI-2. In order to effectively check whether an
 323 analogic approach was used, models were instructed to specify two points in the result; first identify
 the underlying analogy in the given task, and implement it as a detailed solution.
 324

324 For each different sampled GIFARC data as a context, GPT 4.1-mini was instructed to return the
 325 thought process while solving ARC-AGI-2 public evaluation dataset. As described above, since
 326 ARC-AGI-2 dataset requires a more diverse approach than the predecessor, analogic approach in the
 327 task-solving process would be more clearly shown and used for the comparison of the reasoning step.
 328



341 Figure 3: Experiment 1 case study: the LLMs learned in-context with richer analogy context result
 342 in more analogic interpretation of the ARC-AGI-2 task. Context 1, 2, 3 respectively stands for full
 343 description sample, description sample without analogy, description sample without analogy and
 344 without solution.

345 As a result of the solution comparison between models, we could confirm instances that analogies
 346 in GIFARC significantly helped identify the analogy of ARC-AGI-2 task. GPT 4.1-mini with full
 347 description expressed the analogy applied to the ARC-AGI-2 problem using an intuitive analogy.
 348 Figure 3 depicts one of the well-performed result cases. While GPT 4.1-mini with description without
 349 analogy and without solution mainly described the task analogy with grid-level terms (symmetrical,
 350 horizontal, left, right), GPT 4.1-mini with full description was able to describe with words from
 351 several domains (such as ‘tidying up’ and ‘organizing’). In particular, it was observed that GPT
 352 4.1-mini perceived full description as not only as an ‘analogy phrase pool’, but also recognized it as
 353 an ‘example of analogic approach’ and could use external domain terminologies those are not present
 354 in context. This result shows that the analogic terms that were wiped during the flattening process
 355 are critical for the LLMs to get the grasp of analogic approach needed in solving ARC-style tasks.
 356

357 4.2 ALIGNMENT BETWEEN CONTEXT-GENERATED AND GROUND-TRUTH ANALOGIES

358 We have now confirmed that the components are analogically meaningful. Then are those analogies
 359 understandable and aligned to human perception? This experiment was conducted to verify if LLMs
 360 with GIFARC-sampled context can successfully identify underlying analogy in ARC-style tasks.
 361

362 Here, we used full description and analogy-removed description, which we further removed the
 363 analogical expressions that remained in description (such as “gravity”, “spiral spin”, and “flicker”)
 364 and replaced them with grid-level descriptions (such as “move down”, “rotate”, and “color change”).
 365 Next, GPT 4.1-mini with full description and GPT 4.1-mini with analogy-removed description were
 366 instructed to find the analogy implied in the 12 ARC-style tasks. Along with two models, three ARC
 367 expert humans also collected analogies implied in the 12 tasks. The outputs of two models and three
 368 humans were then compared with the ground-truth analogies of the GIFARC task.
 369

370 We first instructed o3-mini to evaluate semantic similarities by providing a one-shot guideline about
 371 measuring similarity. This evaluator LLM evaluated the semantic similarities between analogies
 372 generated by solvers (two in-context learned LLMs and three humans) and the ground truth analogy.
 373

374 Figure 4a depicts the similarity result done by LLM. It resulted that the output of GPT 4.1-mini with
 375 full description showed 0.138 similarity, while the output of GPT 4.1 mini with analogy-removed
 376 description only measured 0.050. In order to cross-check the reliability of the evaluation done by
 377 LLM, we re-conducted the experiment using human evaluators instead of an LLM-based evaluator.
 378 Five human evaluators ranked the analogy answers of five agents (three humans and two models)
 379 for 12 GIFARC tasks, ordering them based on their similarity to the original GIFARC analogies.
 380 As a result, the analogy of three human agents ranked on 1st place for all evaluators, the model
 381

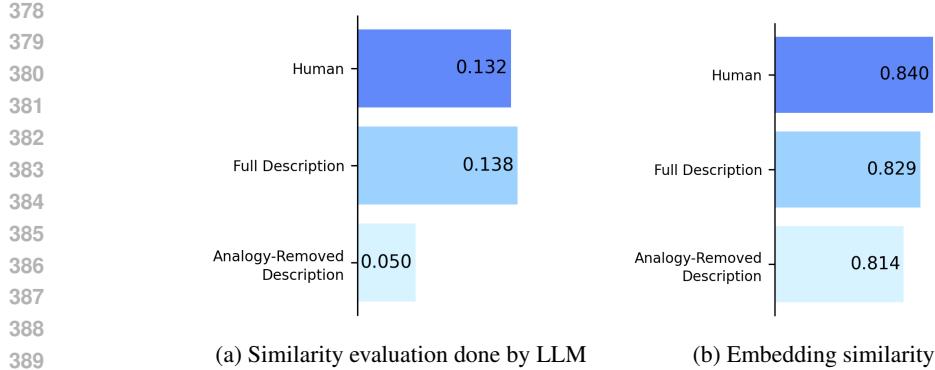


Figure 4: Similarity analysis between task-implied analogy found by GIFARC-trained LLMs and its corresponding hand-crafted analogy identified by three ARC experts. Figure 4a illustrates that the analogy generated by GPT 4.1 mini with full description showed better similarity than that of GPT 4.1 mini with analogy-removed description by 8.7 pp, and embedded cosine similarity by 1.4 pp, as depicted in Figure 4b.

trained with full analogies through in-context learning ranked at 2.33, and the model trained with analogy-removed descriptions ranked at 2.5 on average. These results showed consistency with the rankings obtained using the LLM evaluator in the original experiment.

Secondly, we embedded each outputs and compared the cosine similarity between. It resulted that the output of GPT 4.1-mini with full description showed 0.829 similarity, while the output of GPT 4.1 mini with analogy-removed description only scored 0.814. This shows that context sample from GIFARC played a significant role in changing the task-solving approach of LLM to align with human-level analogic approach. Since GIFARC is a superset of the tested samples, it contains a much various types of analogies. Thus, this experiment shows the potential of GIFARC as the ‘analogic approach’ guiding dataset.

4.3 PERFORMANCE GAINS THROUGH GIFARC FINE-TUNING

Through Section 4.1 and Section 4.2, we confirmed that GIFARC provides analogical reasoning guidance to models. In this section, we verify whether this analogical approach actually leads to improved performance in solving complex reasoning tasks in ARC-AGI. To quantitatively assess whether GIFARC improves reasoning capabilities beyond analogical understanding, we conducted supervised fine-tuning experiments on a language model. We then compared the performance of the GIFARC-trained model against the baseline model on the ARC-AGI.

We conducted comprehensive evaluations using both ARC-AGI-1 and ARC-AGI-2. The ARC-AGI-1 evaluation set contains 400 tasks with a total of 419 test pairs, while the ARC-AGI-2 set contains 120 tasks with 167 test pairs. To quantitatively verify whether GIFARC improves reasoning performance, we compared the performance of Mistral-NeMo-Mintron-8B-Base with the model fine-tuned on GIFARC on both evaluation sets. The fine-tuning was conducted using SFT with LoRA on 1,000 GIFARC tasks for 4 epochs.

We evaluated model performance using two metrics. First, we measured accuracy where a prediction is considered correct only when all elements in the predicted 2D grid exactly match the ground-truth grid; a single incorrect element results in a failed prediction. Second, we assessed model performance using Negative Log Likelihood (NLL), which measures how likely each model is to generate the correct grid. We computed NLL for all test pairs in each evaluation set. NLL is defined as $NLL = -\sum_{i=1}^n \log P(t_i|t_1, \dots, t_{i-1}, \text{prompt})$ where t_1, \dots, t_n are tokens of the ground-truth response. NLL values indicate a higher likelihood of the model generating the correct answer, representing better performance.

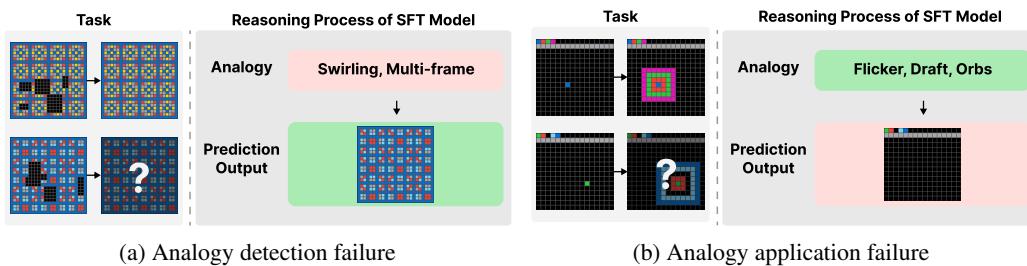
Table 1 shows the quantitative evaluation results. The GIFARC-trained model demonstrates improvement across both metrics. In terms of accuracy, although both models failed to solve any tasks in ARC-AGI-2, reflecting its extreme difficulty, the ARC-AGI-1 results show more promise.

432 The dramatic decreases in NLL indicate that the model assigns significantly higher probability to
 433 correct answers after fine-tuning, suggesting improved understanding of the task structure. Although
 434 the absolute accuracy remains low due to ARC-AGI’s stringent evaluation criterion, where even a
 435 single incorrect grid element results in task failure, both the accuracy gains on ARC-AGI-1 and the
 436 substantial NLL improvements across both evaluation sets demonstrate that the model is learning
 437 meaningful patterns and moving closer to correct solutions.

438
439 Table 1: Base and SFT model performance on ARC-AGI evaluation sets.

Benchmark	Metric	Base Model	SFT Model
ARC-AGI-1	NLL	134	117
	Accuracy	0.2% (1/419)	2.9% (12/419)
ARC-AGI-2	NLL	3724	204
	Accuracy	0% (0/167)	0% (0/167)

447 Quantitative analysis of Table 1 revealed that for ARC-AGI-1 evaluation, the model fine-tuned with
 448 GIFARC successfully solved additional problem types such as grid partitioning, object abstraction,
 449 and occlusion handling. For ARC-AGI-2 evaluation, problem types including object arrangement,
 450 object composition, and symbolic pattern application showed substantial NLL reductions. However,
 451 Figure 5 reveals that the current utilization of GIFARC is incomplete. Figure 5 shows that when
 452 examining the problem-solving process of the SFT model, there are cases where the model fails to
 453 detect appropriate analogies for tasks, and even when appropriate analogies are detected, they are not
 454 properly applied. These findings confirm that GIFARC is a useful learning resource, but suggest that
 455 more sophisticated learning methodologies are needed to improve analogy detection and application
 456 in order to maximize its effectiveness.



464 Figure 5: Two failure modes in the SFT model’s analogical reasoning. Figure 5a shows a case where
 465 the model produces the correct answer despite an incorrect analogy. The task requires identifying a
 466 repeating pattern and filling in black regions accordingly. However, the model incorrectly interprets
 467 this as a multi-frame sequence with a swirling effect. Figure 5b demonstrates a case where the model
 468 identifies an appropriate analogy but fails to apply it correctly. The model correctly reasons that the
 469 task involves creating stable orbs through flicker and draft effects, but cannot properly apply this
 470 analogy to generate the test output grid.

5 CONCLUSION

471 We introduced GIFARC, a large-scale synthetic ARC-style dataset grounded in human-intuitive
 472 analogies extracted from GIF images. GIFARC framework narrows the human-AI reasoning gap
 473 by embedding explicit analogy labels and executable solutions that help LLMs to map abstract grid
 474 transformations with everyday experiences. This approach has potential to improve AI-assisted
 475 education, scientific discovery, and decision-making support systems. Empirical results on the
 476 original ARC benchmark confirm that both fine-tuning on GIFARC and leveraging analogy cues
 477 in the reasoning step boost solver accuracy. However, our data generation pipeline is inherently
 478 dependent on a single GIF for analogy extraction, which places a limitation on the diversity and
 479 scope of analogies. In future work, we will scale GIFARC beyond motion-rich GIFs to videos and
 480 3D simulations, extend evaluation to additional reasoning suites, and explore the fusion of multiple
 481 visual analogies to generate much more complex ARC tasks.

486 REPRODUCIBILITY STATEMENT
487

488 Our work introduces a new dataset for leveraging human-intuitive analogies to elevate AI reasoning.
489 To ensure reproducibility and promote further research, we have made the dataset publicly available
490 at https://huggingface.co/datasets/DumDev/gif_arc. Pipeline codes used for
491 generation and its instructions are available at <https://bit.ly/GIF-ARC>.

493 REFERENCES
494

495 Samuel Acquaviva, Yewen Pu, Marta Kryven, Theodoros Sechopoulos, Catherine Wong, Gabrielle
496 Ecanow, Maxwell Nye, Michael Tessler, and Joshua B. Tenenbaum. Communicating Natural
497 Programs to Humans and Machines. In *NeurIPS*, 2022.

498 Jeremy Berman. How I came in first on ARC-AGI-Pub using Sonnet 3.5 with Evolutionary
499 Test-time Compute, December 2024. URL <https://jeremyberman.substack.com/p/how-i-got-a-record-536-on-arc-agi>.

500
501
502 Jeremy Berman. How i got the highest score on ARC-AGI again swapping Python
503 for English: Using multi-agent collaboration with evolutionary test-time com-
504 pute. Substack, 2025. URL <https://jeremyberman.substack.com/p/how-i-got-the-highest-score-on-arc-agi-again>. Accessed: 2025-11-21.

505
506 Mikel Bober-Irizar and Soumya Banerjee. Neural networks for abstraction and reasoning. *Scientific
507 Reports*, 14(1):27823, 2024.

508
509 François Chollet. On the Measure of Intelligence. *arXiv:1911.01547*, 2019.

510
511 Fransis Chollet, Mike Knoop, and Henry Pinkard Kamradt, Gregory Bryan Landers. ARC-AGI-2: A
512 New Challenge for Frontier AI Reasoning Systems. *arXiv:2505.11831*, 2025.

513 Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean
514 Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of
515 transformers on compositionality. In *NeurIPS*, 2023.

516 Daniel Franzen, Jan Disselhoff, and David Hartmann. The LLM ARCithect: Solving ARC-AGI Is
517 A Matter of Perspective, 2024. URL [https://github.com/da-fr/arc-prize-2024/
blob/main/the_architects.pdf](https://github.com/da-fr/arc-prize-2024/blob/main/the_architects.pdf).

518
519 Michael Hodel. Addressing the Abstraction and Reasoning Corpus via Procedural Example Genera-
520 tion. *arXiv:2404.07353*, 2024.

521
522 Subin Kim, Prin Phunyaphibarn, Donghyun Ahn, and Sundong Kim. Playgrounds for Abstraction
523 and Reasoning. In *NeurIPS Workshop on Neuro Causal and Symbolic AI*, 2022.

524
525 Seungpil Lee, Woochang Sim, Donghyeon Shin, Sanha Hwang, Wongyu Seo, Jiwon Park, Seokki
526 Lee, Sejin Kim, and Sundong Kim. Reasoning Abilities of Large Language Models: In-Depth
527 Analysis on the Abstraction and Reasoning Corpus. *arXiv:2403.11793*, 2024.

528
529 Solim LeGris, Wai Keen Vong, Brenden M Lake, and Todd M Gureckis. H-ARC: A Robust Estimate
530 of Human Performance on the Abstraction and Reasoning Corpus Benchmark. *arXiv preprint
arXiv:2409.01374*, 2024.

531
532 Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M Dunn,
533 Hao Tang, Michelangelo Naim, Dat Nguyen, et al. Combining induction and transduction for
534 abstract reasoning. *arXiv preprint arXiv:2411.02272*, 2024.

535
536 Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas,
537 Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large Language Models as General Pattern
538 Machines. In *CoRL*, 2023.

539 Melanie Mitchell. Abstraction and analogy-making in artificial intelligence. *Annals of the New York
Academy of Sciences*, 1505(1):79–101, 2021.

540 Arseny Moskvichev, Victor Vikram Odonard, and Melanie Mitchell. The ConceptARC Benchmark:
541 Evaluating Understanding and Generalization in the ARC Domain. *Transactions on Machine*
542 *Learning Research*, 2023.

543 Gustaw Opiełka, Hannes Rosenbusch, Veerle Vijverberg, and Claire E Stevenson. Do Large Language
544 Models Solve ARC Visual Analogies Like People Do? In *CogSci*, 2024.

545 CT Pang. [ARC-AGI-2 SoTA] efficient evolutionary program synthesis.
546 Substack, 2025. URL <https://ctpang.substack.com/p/arc-agi-2-sota-efficient-evolutionary>. Accessed: 2025-11-21.

547 Yonggang Qi, Kai Zhang, Aneeshan Sain, and Yi-Zhe Song. PQA: Perceptual Question Answering.
548 In *CVPR*, 2021.

549 Donghyeon Shin, Seungpil Lee, Klea Lena Kovacec, and Sundong Kim. From Generation to Selection
550 Findings of Converting Analogical Problem-Solving into Multiple-Choice Questions. In *EMNLP*,
551 2024.

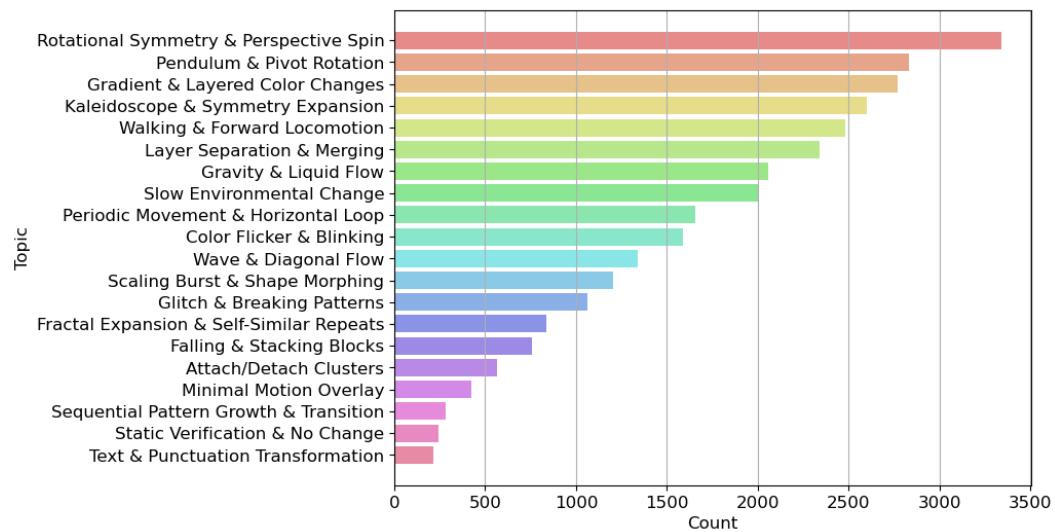
552 Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias B Khalil. LLMs and the
553 Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-Based
554 Representations. *Transactions on Machine Learning Research*, 2024.

555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593

594 APPENDIX
595596 A DATASET ANALYSIS
597

598 **Descriptive Statistics.** GIFARC currently contains 10,000 tasks, each accompanied by 103,357
599 train-test input/output pairs and an executable solution. The average input grid is 420 ± 262 cells,
600 while target grids are at $1,125 \pm 2,895$. On average, a task uses 4.4 ± 1.9 distinct colors, indicating
601 that most tasks require multi-object reasoning or contain objects with some complexity, rather than
602 binary foreground/background segmentation.
603

604 **Distribution of Task Types.** To analyze the semantic concepts and human-intuitive analogies
605 represented in the dataset, we visualized the distribution of task types. Each keyword was mapped by
606 GPT-4o into one of 20 coarse types, and Figure 6 presents the resulting histogram. Although the raw
607 GIFs were initially sampled uniformly across 20 categories (e.g., nature, geometry, animation, ...),
608 the distribution of task types in the final dataset diverges from this initial sampling. The generated
609 problems cover a wide range of task types, with ‘Rotational Symmetry & Perspective Spin’ being the
610 most dominant, and many tasks exhibiting a mixture of multiple types.
611



629 Figure 6: Histogram of task types occurrence in GIFARC. We classified the tasks in the ARC-AGI-2
630 training set using the 20 category tags defined in the GIFARC dataset.
631

632 **Comparing Taxonomy Distributions.** In the ARC-AGI-2 training dataset, Layer Separation &
633 Merging accounted for the largest share (11.3%), followed by Gravity & Liquid Flow (9.6%) and
634 Color Flicker & Blinking (9.3%), whereas in GIFARC, Rotational Symmetry is the dominant category.
635 The average number of tasks per category was 36 (SD = 26). Although the distributions of GIFARC
636 and ARC-AGI-2 differ, GIFARC achieves 90.7% category coverage. Moreover, because GIFARC
637 translates diverse visual elements derived from GIFs into object concepts on the ARC grid and
638 composes patterns from them, it can supplement task types that are absent from the ARC-AGI-2
639 training set.
640

641 **Task Complexity.** We assessed task complexity by measuring the complexity of the code used to
642 generate GIFARC’s grids. To this end, we report metrics such as the average number of lines of code
643 (Lines of Code), the number of branching points in the code (Cyclomatic Complexity), the maximum
644 depth of nested blocks in the code (Nesting Depth), and the number of unique operators used in the
645 code (Unique Ops Count). These metrics make it easy to gauge how complex the processes required
646 by GIFARC tasks are.
647

Code complexity metrics measured for GIFARC tasks are as follows: The average number of lines of
code is 38.43 ± 10.55 , while the cyclomatic complexity averages 6.59 ± 3.62 . The maximum nesting

648
649
650 Table 2: Code Complexity of GIFARC Tasks compared to BARC.
651
652
653
654
655
656

	GIFARC		BARC	
	Mean	Std.	Mean	Std.
Lines of Code	38.43	10.55	30.20	8.13
Cyclomatic Complexity	6.59	3.62	4.00	2.07
Nesting Depth	4.37	1.55	3.66	1.53
Unique Ops Count	7.03	0.28	7.42	1.07

657
658
659 depth in the code is 4.37 ± 1.55 , and the number of unique operators averages 7.03 ± 0.28 . These
660 metrics quantitatively demonstrate that GIFARC contains tasks with diverse levels of complexity.
661662
663
664
665
666
667
668 GIFARC has a longer average Lines of Code and a higher Cyclomatic Complexity score than BARC.
The standard deviations of both metrics are also greater compared to BARC. This indicates that
GIFARC encompasses problems with a relatively diverse range of code lengths and control flows,
from simple branching to complex control structures. In contrast, Nesting Depth and Unique Ops
Count does not show significant difference between the two datasets. Therefore, GIFARC can be
considered a dataset that, compared to BARC, is similar to existing tasks in terms of nesting structures
and variety of operations, while extending difficulty in code length and branching complexity -
making it more complex and diverse in problem distribution than the original BARC.669
670
671
672
673
674
675
676 **Generation Fidelity.** We measure the fidelity of our generation pipeline by evaluating the pass ratio
at each stage. A task is considered a failure if it fails to compile or does not produce outputs in the
expected format. Table 3 reports the success rates, showing that the overall pass ratio is consistently
high across stages. The 19-20% failure rates mentioned are stage-specific figures, and since we only
need to retry the failed stage rather than the entire pipeline, the actual additional API costs are very
limited. Our analysis of the GIFARC data generation pipeline revealed two main failure causes: (1)
API content policy-related errors and (2) non-conforming VLM responses, both of which can be
addressed through simple retry logic and enhanced response validation.677
678 Table 3: Success rate of generation.
679
680
681
682
683

	Rate (%)
Visual abstraction stage	94.12
Task sketch stage	80.54
Executable ARC task stage	81.82

684
685
686
687
688
689
690
691
692
693
694 **Qualitative Quality Evaluation** To evaluate whether the synthetically generated GIFARC tasks
meaningfully incorporate the visual transformations extracted from the original GIFs, we conducted
three human evaluation studies on 1,000 randomly sampled tasks. In our generation pipeline, step 2
produces task-defining concepts and descriptions that are explicitly designed to capture the core visual
transformations present in the GIFs and to guide the synthesis of the final tasks. Human evaluators
were then asked to assess whether the resulting synthetic ARC-style tasks faithfully reflected these
step-2 concepts and descriptions. Our evaluation shows that 85.7% of the sampled tasks exhibit a
clear alignment between the final task and its underlying concepts and descriptions. This suggests that,
for the majority of GIFARC tasks, the dynamic visual elements captured at the concept-extraction
stage are successfully integrated into the final task structure.695
696
697
698
699
700
701

702 **B EXAMPLES OF PROMPTS**
703704 **B.1 FULL PROMPT EXAMPLE FOR EXTRACTING VISUAL ABSTRACTIONS**
705706 The extraction of visual abstractions constitutes the first step in the data genera-
707 tion process. This process involves extracting comprehensive information from GIFs,
708 including `visual_elements`, `objects`, `static_patterns`, `dynamic_patterns`,
709 `core_principles`, and `interactions`. The specific content requirements for these six types
710 of information and the response format for the VLM are detailed in both the user prompt and system
711 prompt below. The following content presents the complete specification of the aforementioned
712 prompts.
713713 **User Prompt: Extracting Visual Abstractions from GIFs**
714715 Please analyze the given GIF by extracting core reasoning elements. For each section below, provide
716 your response as a list of clearly separated items (not long-form paragraphs).
717717 **[visual_elements]**718

- 719 • Identify the key visual elements, objects, and actors in the GIF.
- 720 • List the major visual components in the scene (e.g., objects, colors, spatial arrangement,
721 shapes).

722 **[objects]**723

- 724 • List all visual objects appearing in the GIF using this format:
725

```
{  
726     "name": "object name",  
727     "type": "explicit" or "implicit"  
728 }
```
- 729 • explicit = physical objects with clear boundaries
- 730 • implicit = patterns or structures formed by multiple elements
- 731 • Identify 2-8 key objects.

732 **[interactions]**733

- 734 • List object interactions using this format:
735

```
{  
736     "objects_involved": ["object1", "object2"],  
737     "interaction_type": "clear/ambiguous/constraint",  
738     "interaction_parameters": ["parameter1", "parameter2"]  
739 }
```
- 740 • interaction_type definitions:
 - 741 • **clear**: distinct physical interactions (collisions, contact, etc.)
 - 742 • **ambiguous**: indirect or unclear interactions
 - 743 • **constraint**: interactions that establish limitations or boundaries
- 744 • Record 2-6 key interactions, including only those directly observed in the GIF.

745 **[static_patterns]**746

- 747 • Identify the elements or relationships that remain consistent throughout the GIF.
- 748 • Describe repeating spatial arrangements, consistent backgrounds, or fixed design features.
- 749 • List all patterns or objects that remain constant throughout the GIF.

750 **[dynamic_patterns]**751

- 752 • Describe how the elements interact and change over time.
- 753 • Consider whether the changes occur gradually (step-by-step), abruptly, or through multi-stage
754 transformations (e.g., directional shifts, scaling, rotation).
- 755 • List the distinct changes or interactions that occur over time.

756 **[core_principles]**

756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809

- Identify all general reasoning principles or mechanisms that explain **how and why** the dynamic elements change over time.
- Each principle should be **generalizable beyond the current GIF** and help abstract the reasoning structure behind the transformation.
- List one or more such principles that account for the observed dynamics.
- After listing them, **summarize the single most fundamental principle in one concise sentence.**
- Examples include: physical forces (e.g., gravity causes downward movement), goal-oriented behaviors (e.g., movement toward a target), causal chains (e.g., one event triggers another), symmetry-based transformations (e.g., reflection or alignment), and repetitive or cyclic patterns.

System Prompt: Extracting Visual Abstractions from GIFs

You are an analysis assistant. Your role is to extract structured and reproducible information from a visual GIF using the five categories below.

You **MUST** focus on observable phenomena, transformation structures, and etc.

Do **NOT** invent unobservable events or make abstract generalizations. Describe only what can actually be seen in the GIF.

For each category below, provide a plain list of items (one per line or bullet point). Avoid paragraph-style narration. Focus only on concrete, observable phenomena and transformation patterns.

Return your response strictly in the following JSON format:

```
{
  "visual_elements": [
    "<list of observable objects, colors, spatial arrangements, or
    notable visual traits>",
    ],
  "objects": [
    {
      "name": "<object name>",
      "type": "explicit/implicit"
    }],
  "interactions": [
    {
      "objects_involved": ["<object1>", "<object2>"],
      "interaction_type": "clear/ambiguous/constraint",
      "interaction_parameters": ["<parameter1>", "<parameter2>"]
    }],
  "static_patterns": [
    "<list of all objects or properties that remain unchanged
    throughout the GIF>",
    ],
  "dynamic_patterns": [
    "<list of all distinct transformations or movements that occur
    over time>",
    ],
  "core_principles": [
    "<list of general reasoning principles behind the transformation
    (e.g., gravity causes vertical motion)>"
  ]
}
```

810 B.2 FULL PROMPT EXAMPLE FOR TASK SKETCH
811812 The task sketch represents the second step in the data generation process. The objective of this step
813 is to design ARC-style tasks by dividing them into concepts and description components,
814 utilizing the six types of information extracted in Step 1. The user prompt provides the detailed
815 specifications for this process.816 User Prompt: Task Sketch
817818 You've generated these on previous requests:
819 {examples}820 Based on your previous GIF analyses, I'd like you to create a new concept that incorporates
821 the key elements from this analysis:822 objects: {objects}
823 static_patterns: {static_patterns}
824 dynamic_patterns: {dynamic_patterns}
825 interactions: {interactions}
826 core_principles: {core_principles}

827 Brainstorm one more.

828 If the above informations involve gradual changes over time or a process of reaching a clear
829 goal state, the input and output grids should not merely depict a short-term or single-frame
830 transition. Instead, they should be designed to capture the entire transformation process from
831 the initial state to the final state.832 When temporal progression or cumulative change is central to the analogy, construct the task
833 code so that this flow and its outcome are clearly reflected through the difference between
834 input and output.835 In this case, please create concepts and descriptions that effectively incorporate the above
836 visual elements, static patterns, dynamic patterns, and core principles, following the format
837 below, while referencing previous examples:838 # concepts:
839 # <concepts in your new generation>
840 # description:
841 # <description of your new generation>842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

864
865

B.3 FULL PROMPT EXAMPLE FOR GENERATING EXECUTABLE ARC TASKS

866
867
868
869

We generated ARC-style tasks using the analogies extracted from Step 1 and Step 2. Step 3 utilizes original BARC seeds that were used to generate input and output grid pairs. The following prompt contains the detailed explanation for Step 3.

870

871

User Prompt: Puzzle Implementation from Description

872
873
874
875

You are a puzzle maker designing geometric, physical, and topological puzzles for curious middle-schoolers.

876
877
878

Each puzzle consists of uncovering a deterministic rule, pattern, procedure, algorithm, or transformation law that maps inputs to outputs. Both the inputs and outputs are 2D grids of colored pixels. There are 10 colors, but the order of the colors is never relevant to the puzzle.

879
880
881
882
883
884

The middle schoolers are trying to discover this deterministic transformation, which can be implemented as a Python function called `main`. Designing a puzzle involves also creating example inputs, which can be implemented as a Python function called `generate_input`. Unlike `main`, the `generate_input` function should be stochastic, so that every time you run it, you get another good example of what the transformation can be applied to.

885
886
887

Here is a overview of the puzzle you are designing:

```
{description}
```

888
889

Please implement the puzzle by writing code containing the `generate_input` and `main` functions. Use the following standard library (`common.py`):

```
{common_lib}
```

890
891
892
893

Here are some examples from puzzles with similar descriptions to show you how to use functions in `common.py`:

```
{examples}
```

894
895
896

Your task is to implement the puzzle, following these steps:

897
898
899
900
901
902
903

1. Inspect the example puzzle implementations, making note of the functions used and the physical/geometric/topological/logical details
2. Inspect the new puzzle's description
3. Brainstorm a possible implementation for the new puzzle
4. Generate a code block formatted like the earlier examples with a comment starting `# concepts:` listing the concepts and `# description:` describing the inputs and transformation from the given description.

904
905
906
907
908

When implementing code, please avoid using float type variables, numbers with decimal points, and the math library. These elements make it difficult to intuitively identify patterns between input and output grids. Instead, use only integer operations and basic arithmetic operators to clearly reveal the essence of the pattern. Please strictly follow this constraint when implementing your code.

909
910
911
912
913

Also, When implementing code, please use the minimum number of lines possible. As code gets longer, its complexity increases, and if it becomes too detailed and complicated, people will find it difficult to intuitively understand the puzzle's rules just by looking at the input and output grids. Situations where one needs to analyze the code to understand the rules should be avoided. Please write concise and efficient code that clearly reveals the core pattern.

914
915

Be sure to make the transformation `main` deterministic. Follow the description closely.

916
917

918
919

System Prompt: Generating Executable ARC Task (Version 1)

920
921
922
923

You need to help me write code containing the `generate_input` and `main` functions according to the given puzzle design. You must use the standard library (`common.py`). Create an appropriate puzzle following the given puzzle design concepts and description.

924
925
926
927
928
929
930
931
932
933
934
935

When writing code, please use variable names that are meaningfully related to the core concepts of the problem. For example, if the problem involves snow falling phenomena, use variable names like `snowflake`, `precipitation`, `accumulation`, `gravity`, `obstacle`, etc. Specifically, when implementing the `generate_input` function and `main` function, make sure each variable name is directly associated with the concepts in the problem. For instance, use `gravity_strength` for a variable representing the intensity of gravity, and `obstacle_positions` for storing the locations of obstacles — choose names that clearly reveal the role and meaning of each variable.

Additionally, in the `generate_input` function, please restrict the grid size to be between `1x1` and `30x30`. Do not create grids larger than `30x30`. Implement the `generate_input` function that creates inputs appropriate for the problem and the `main` function that utilizes them while following these constraints.

When implementing code, please avoid using float type variables, numbers with decimal points, and the math library. These elements make it difficult to intuitively identify patterns between input and output grids. Instead, use only integer operations and basic arithmetic operators to clearly reveal the essence of the pattern. Please strictly follow this constraint when implementing your code.

Also, when implementing code, please use the minimum number of lines possible. As code gets longer, its complexity increases, and if it becomes too detailed and complicated, people will find it difficult to intuitively understand the puzzle rules just by looking at the input and output grids. Situations where one needs to analyze the code to understand the rules should be avoided. Please write concise and efficient code that clearly reveals the core pattern.

When doing this, please output your solution following the JSON format specified below.

```
{
  "library": "<Write only the libraries used in the code.  
Ex. from common import* \n import numpy as np \n ....>",
  "main_code": "<Write the main code part.>",
  "generate_input_code": "<Write the generate input code part.>",
  "total_code": "<Write total code including libraries, main, generate_input and given concepts and description>"
}
```

956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

972 **B.4 FULL PROMPT EXAMPLE FOR GENERATING ANALOGY GIVEN TASKS**
 973
 974 In the Section 4.2, we used the following prompt to generate analogies for the given task.
 975
 976 User Prompt: Generating Analogy
 977
 978 **TOP-LEVEL INSTRUCTIONS – READ FIRST**
 979 • The section between **Few-Shot Examples** and <task> is an example block that demonstrates
 980 how to solve the task using analogies.
 981 • The lines that follow <task> present the task you must solve, consisting of Training
 982 Examples and Test Examples.
 983 **Few-Shot Examples**
 984
 985 <additional_information>
 986 This information provides helpful context to solve the given ARC
 987 task.
 988
 989 <description w/ analogy or description w/ analogy and solution -
 990 1>
 991 ...
 992 <description w/ analogy or description w/ analogy and solution - 15>
 993
 994 <task>
 995
 996 </task>
 997
 998 <additional_information>
 999 This information provides helpful context to solve the given ARC task.
 1000 {description w/ analogy or description w/ analogy and solution - 1}
 1001 ...
 1002 {description w/ analogy or description w/ analogy and solution - 15}
 1003
 1004 </additional_information>
 1005
 1006 <task>
 1007 ...
 1008 </task>
 1009
 1010 Please solve task thoroughly and structure your response with the following sections:
 1011
 1012 **# THINKING ANALOGY**
 1013 [Let's think the analogical pattern about task step by step]
 1014 Let's think step by step about the analogical pattern in the task, and
 1015 describe what kind of analogical, metaphorical, or figurative pattern
 1016 it is
 1017
 1018 **# THINKING PROCESS**
 1019 [Let's think the task step by step by leveraging thinking analogy and
 1020 task analogy]
 1021 Let's think the task leveraging the analogical, metaphorical, or figu-
 1022 rative patterns observed in the 15 examples to uncover and apply the
 1023 underlying analogy
 1024
 1025 **# SOLUTION CODE**
 1026 [Write the final solution code for the task based on the analogical
 1027 approach identified in the thinking process]
 1028
 1029 Let's solve the problem step by step.
 1030
 1031

1026 **C IMPLEMENTATION DETAILS**
10271028 **C.1 MODEL HYPERPARAMETERS**
10291030 In this pipeline, we set the temperature to 0 in order to build deterministic datasets. Note that
1031 reasoning models like OpenAI o-series do not support temperature configuration, so this parameter
1032 was not set for those models. For more detailed hyperparameters, refer to Table 4.
10331034 **Table 4: OpenAI API Hyperparameters**
1035

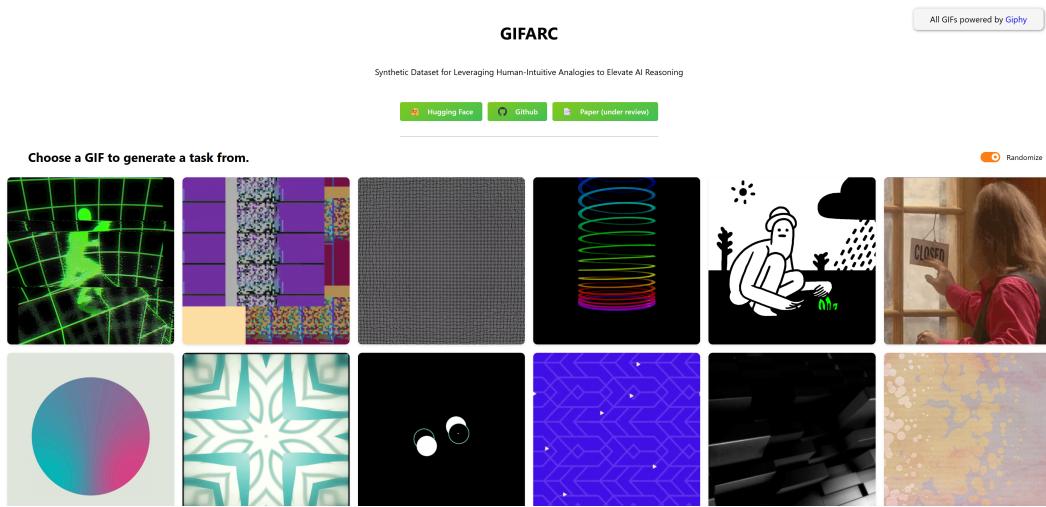
Process Name	Role	Model Name	Max Tokens	Top p	Temperature
Method Step 1	Data Generator	o4-mini	2,048	1.0	-
Method Step 2	Data Generator	o4-mini	2,048	1.0	-
Method Step 3	Data Generator	o4-mini	2,048	1.0	-
Application	LLM Evaluator	o3-mini	40,000	1.0	-
Application	LLM Solver	GPT 4.1-mini	32,768	1.0	0.0

1045 **C.2 DATA GENERATION**
10461047 GIFARC builds on the data generation process established by the prior work BARC (Li et al., 2024).
1048 The original BARC framework generated tasks using manually crafted functions called seeds. A key
1049 difference from BARC is that GIFARC employs o4-mini to extract six key pieces of information
1050 from GIFs in Step 2, which are then used to design tasks in Step 3. Generation is based on the
1051 `Description`. Based on the designed tasks, we use o4-mini to generate code in Step 3 that creates
1052 inputs and the `main` function that produces outputs. We create input and output grids and apply a
1053 filtering process using this generated code.1054 The filtering criteria consist of ten conditions adapted from BARC, as shown in Table 5. Input-output
1055 grid pairs that meet any of these conditions are filtered out. In GIFARC, creating tasks that incorporate
1056 analogies often results in complex code. Therefore, we set a maximum time limit of 300 seconds per
1057 task to provide sufficient processing time. Cases requiring more time are considered to be caught
1058 in infinite loops and are filtered out as timeout failures. This filtering process aims to enhance the
1059 quality of GIFARC-generated tasks.
10601061 **Table 5: Filtering Conditions**
1062

Filtering Condition	Meaning
Non-Deterministic	The output grid differ when the same transformation is applied multiple times to the same input.
Non Color Invariant	1. Cases where the transformation function itself fails during the color permutation process. 2. Either the permuted grid or the input grid is not well-formed. 3. The results differ even when only the colors are changed.
Identity	The input and output are completely identical.
Non-Well Formed Output	The transformation result is not well-formed (i.e., not a 2D list with equal row lengths and integer values between 0-9).
Black Output	The output consists entirely of 0s (black pixels).
Timeout	Overall time limit exceeded.
Non-Well Formed Input	The input is not well-formed (i.e., not a 2D list with equal row lengths and integer values between 0-9).
Duplicate Input	The generated input is duplicated with existing ones.

1080 C.3 APPLICATION
10811082 We conducted experiments on Section 4.1 which is “Verification Changes in Reasoning Steps” and
1083 Section 4.2 which is “Alignment between Context-Generated and Ground-Truth Analogies”. For
1084 these experiments, we developed various prompt context conditions: full description, description
1085 without analogy, and description without analogy and without solution.1086 The full description was generated using o3-mini to enhance the analogies in both the α and ϕ of the
1087 GIFARC dataset. To prevent the inclusion of irrelevant analogies, we performed manual refinement
1088 of the generated content. For the description without analogy, we utilized o3-mini to remove analogy-
1089 related information from the full description. Subsequently, we converted any remaining analogical
1090 elements in both α and ϕ to grid-level representations. The description without analogy and without
1091 solution used the same analogical framework as the description without analogy. Across all kinds of
1092 descriptions, the grid information remained consistent.1093 In our experiments examining Section 4.1 and Section 4.2, we employed both the full description and
1094 the analogy-removed description. The full description was identical to that used in the Section 4.1
1095 application. The analogy-removed description was a refined version of the description without
1096 analogy used in the Section 4.1 experiment, where we completely eliminated any remaining analogical
1097 information by either removing it entirely or converting it to grid-level representations, thus ensuring
1098 the complete absence of analogical content.1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

1134 **D GUIDELINES FOR HUMAN EVALUATORS ASSESSING THE GIVEN TASKS**
11351136 In the section 4.2, we conducted human evaluation on 12 tasks. Three experts familiar with ARC
1137 were asked to describe the analogies for 12 GIFARC tasks. To obtain responses of sufficient quality
1138 for our experiment, we provided minimal guidelines. The following is a brief outline of the guidelines
1139 provided to the human evaluators.1140 **Guidelines for Human Evaluators**
11411142 Please write the pattern analogy in English sentences for the provided 12 tasks (input-output
1143 pairs), referring to the content below:1144
1145 1. What the provided input-output pairs are visualizing
1146 2. What kind of analogy the pattern changing from input to output can be related to or
1147 expressed as
1148 3. What the colors or object shapes shown in the input/output symbolize or represent
1149 4. Please write the expected rules if you don't know the task's analogy.
1150 Ex-1) - It appears to be spreading from the center.
1151 Ex-2) - The movement seems to change randomly and the colors are changing.1152
1153 [Working Example]
1154 {Example task}1155
1156
1157 Analogy: In this task the grid represents a scene where small buds (colored gray: 5) transform
1158 into blooming flowers. The transformation simulates the process of blooming: each bud
1159 expands its presence by turning its neighboring cells (up, down, left, right) into petal cells
1160 (colored gray: 5) while the original bud cell becomes the highlight of the bloom (colored
1161 yellow: 4).1162
1163 [GIFARC Task 1]
1164 {Input-Output Pairs}1165
1166 :1167
1168 [GIFARC Task 12]
1169 {Input-Output Pairs}1170
1171 Based on this guideline, participants described the analogies for the given GIFARC tasks. The
1172 information collected contained only content about the analogies, and no personal or sensitive
1173 information was gathered.1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

1188 E EXAMPLE VISUALIZATIONS OF GIFARC
1189
1190
1191
1192
1193
1194
11951206
1207 Figure 7: Landing Page of GIFARC Visualization website. It is publicly released at <https://gifarc.vercel.app/>
1208
12091210 All 10,000 tasks generated with GIFARC are visualized in <https://gifarc.vercel.app/>.
1211 Following are example visualizations of GIFARC-generated ARC-style tasks of each task type.
12121213 E.1 EXAMPLE TASK OF TYPE 1 - ROTATIONAL SYMMETRY & PERSPECTIVE SPIN
1214
12151228
1229 Figure 8: Snapshot of GIF used for generating task 962.
1230
1231
1232

- 1233 • Concepts : fractal expansions, symmetrical pulsations, radial transformations, iterative growth
- 1234 • Description : In the input, you will see a sequence of grids (frames) on a black background. Each
1235 grid depicts: - A large circular region in the center, with multiple concentric rings and radial
1236 lines diverging outward. - A stacked-curve fractal anchored at the bottom-left corner. - A set of
1237 spire fractals anchored at the bottom-right corner. As you move from one frame to the next in
1238 the input, these elements undergo iterative transformations: - The radial lines repeatedly shift in
1239 thickness and visual intensity, while preserving rotational symmetry. - The concentric rings in the
1240 circular region pulsate by alternately expanding and contracting. - The stacked-curve fractal on the
1241 bottom-left changes its curve density, fractally adding or removing segments. - The right-side spire
fractals expand and contract in repeated vertical segments. Your task is to replicate and apply these

1242 transformations for the entire sequence, and produce the final frame of the animation as the output:
 1243 1) Ensure the anchoring of the fractals at the bottom edge remains the same. 2) Preserve the radial
 1244 symmetry of the central circular region and its common center point. 3) For each pulsation step
 1245 in the input, magnify or contract the rings, lines, and fractals accordingly. 4) Continue until the
 1246 final pulsation step is reached. That final state is your output grid. The essential principle is that
 1247 symmetry-based fractal transformations and repeated cyclical expansions/contractions produce the
 1248 overall pulsating effect. By reflecting each iterative change step by step, you reconstruct the final
 1249 pulsating pattern visible at the end of the sequence.

1250 • Full web view is available at <https://gifarc.vercel.app/task/962>.

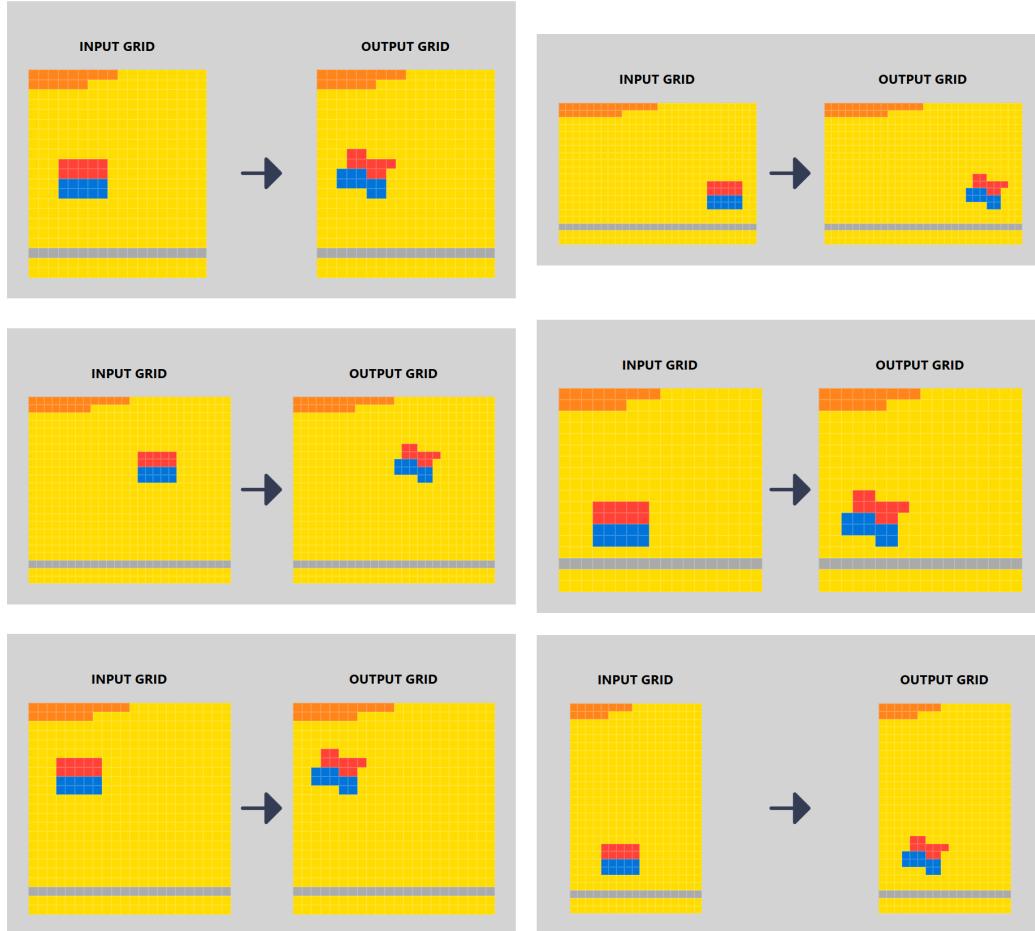


Figure 9: GIFARC-generated task 962.

Listing 1: Python code solution for task 962.

```

1296
1297
1298 1 from common import *
1299 2 import numpy as np
1300 3 import random
1301 4
1302 5 # concepts:
1303 6 # rotational motion, center-of-mass shifting, multi-step transformation,
1304 7 # static background
1305 8 # description:
1306 9 # The input grid shows a paved surface (Color.YELLOW) with static
1307 # features: a fence (Color.GREY) drawn near the bottom and mountains
1308 10 (Color.ORANGE) at the top.
1309 # A dynamic object-a wheelchair with its occupant-is represented by a
1310 # 4x5 sprite: the top two rows are the occupant (Color.RED) and the
1311 # bottom two rows the wheelchair (Color.BLUE).
1312 # The wheelchair rotates around an axle (computed as the average
1313 # position of the blue pixels) so that the occupant's mass shifts
1314 # further off support.
1315 # The transformation in main erases the original dynamic pixels and
1316 # re-blits them using an integer-approximated clockwise rotation
1317 13 (about 22 degrees) around the axle, while leaving the static
1318 # background unchanged.
1319 14
1320 15 def main(input_grid):
1321 16     out=input_grid.copy()
1322 17     dyn=[]
1323 18     for i in range(len(input_grid)):
1324 19         for j in range(len(input_grid[0])):
1325 20             if input_grid[i,j] in [Color.BLUE,Color.RED]:
1326 21                 dyn.append((i,j,input_grid[i,j]))
1327 22                 out[i,j]=Color.YELLOW
1328 23             if not dyn: return out
1329 24             ax_i=sum(p[0] for p in dyn)//len(dyn)
1330 25             ax_j=sum(p[1] for p in dyn)//len(dyn)
1331 26             cos_val, sin_val = 92, 38
1332 27             for i,j,col in dyn:
1333 28                 di=i-ax_i; dj=j-ax_j
1334 29                 ni=ax_i+((cos_val*di+sin_val*dj)//100)
1335 30                 nj=ax_j+((-sin_val*di+cos_val*dj)//100)
1336 31                 if 0<=ni<len(out) and 0<=nj<len(out[0]): out[ni,nj]=col
1337 32             return out
1338 33
1339 34 def generate_input():
1340 35     H,W=random.randint(15,30),random.randint(15,30)
1341 36     grid=np.full((H,W),Color.YELLOW)
1342 37     for j in range(W): grid[H-3,j]=Color.GREY
1343 38     for i in range(2):
1344 39         for j in range(W//(2+i)):
1345 40             grid[i,j]=Color.ORANGE
1346 41     sprite=np.full((4,5),Color.BLACK)
1347 42     sprite[:2,:]=Color.RED; sprite[2:,:]=Color.BLUE
1348 43     x,y=random_free_location_for_sprite(grid,sprite,background=
1349 44             Color.YELLOW)
1350 45     blit_sprite(grid,sprite,x,y)
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423

```

1350 E.2 EXAMPLE TASK OF TYPE 2 - KALEIDOSCOPE & SYMMETRY EXPANSION
1351

- 1352 Concepts: kaleidoscopic transformation, mirrored folding, cyclical animation, vertical symmetry
- 1353 Description : In this puzzle, the input is a multi-frame sequence depicting a pale-yellow corridor-like structure with a central rectangular opening and mirrored arches aligned along a vertical axis of symmetry. Over several frames, the corridor's walls fold inward and outward in a smooth, kaleidoscopic motion, always returning to the initial configuration. The output is the full cycle of transformations repeated in a loop, preserving the corridor's pale-yellow hue and central frame while continuously reflecting the mirrored arch patterns around the vertical axis. This cyclical approach showcases the corridor morphing into itself, completing each transformation stage and then re-starting at the original symmetrical configuration.
- 1354
- 1355
- 1356
- 1357
- 1358
- 1359
- 1360
- 1361
- 1362
- 1363
- 1364
- 1365
- 1366
- 1367
- 1368
- 1369
- 1370
- 1371
- 1372
- 1373
- 1374
- 1375
- 1376
- 1377
- 1378
- 1379
- 1380
- 1381
- 1382
- 1383
- 1384
- 1385
- 1386
- 1387
- 1388
- 1389
- 1390
- 1391
- 1392
- 1393
- 1394
- 1395
- 1396
- 1397
- 1398
- 1399
- 1400
- 1401
- 1402
- 1403

1394 Figure 10: GIFARC-generated task 7442.
1395
1396
1397
1398
1399
1400
1401
1402
1403

1404

E.3 EXAMPLE TASK OF TYPE 3 - PENDULUM & PIVOT ROTATION

1405

- Concepts : cyclical rotation, pivot anchoring, animation frames

1406

• Description : Inspired by previous examples that repeated or translated objects across multiple frames, in this puzzle the input is a single snapshot showing a blue curved shape on a black background. For the output, construct a multi-frame grid depicting the same shape completing a full 360-degree rotation around a central pivot. Each frame rotates the shape by a fixed angle (e.g., 15° increments), and the frames are arranged in a 3x3 or 4x4 grid until the shape returns to its original orientation. The black background and the blue color remain consistent across frames, and the shape never goes outside the canvas boundary. This cyclical rotation from start to finish, then repeating, is the puzzle's core concept.

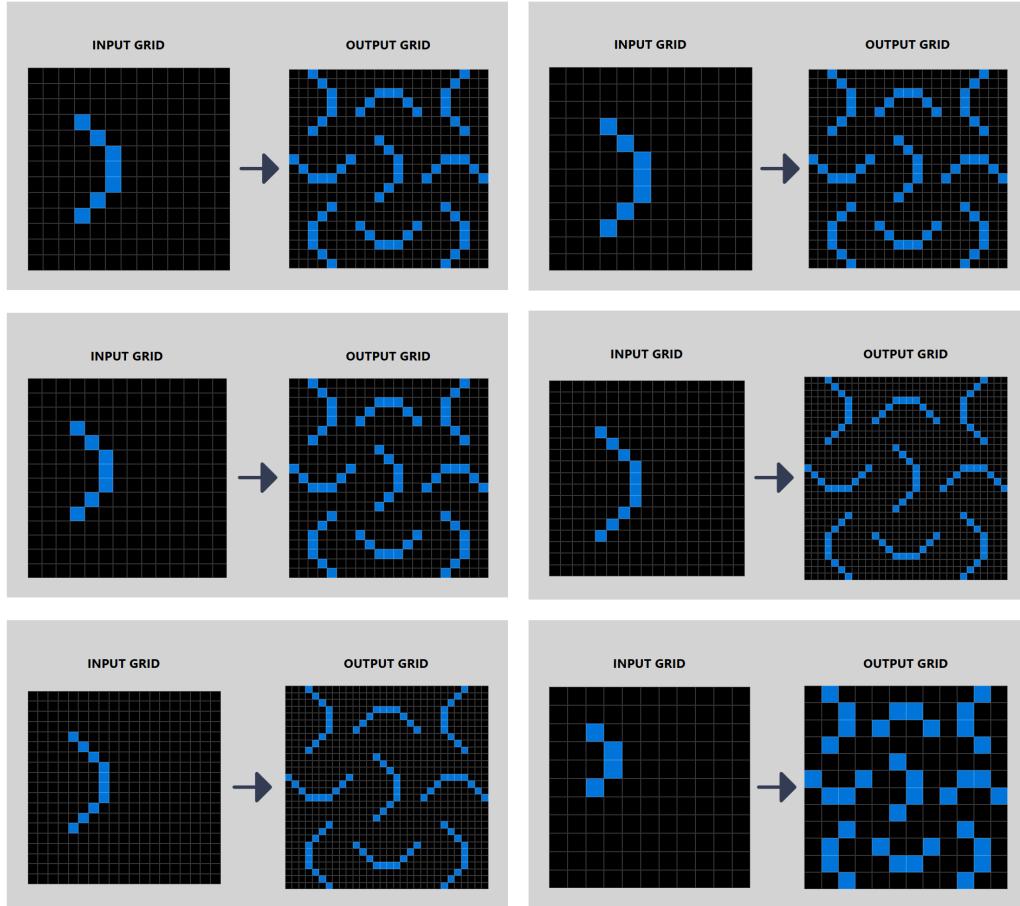
1414

- Full web view is available at <https://gifarc.vercel.app/task/1037>.

1415

1416

1417



1418

1419

1420

1421

1422

1423

1424

1425

1426

1427

1428

1429

1430

1431

1432

1433

1434

1435

1436

1437

1438

1439

1440

1441

1442

1443

1444

1445

1446

1447

Figure 11: GIFARC-generated task 1037.

1448

1449

1450

1451

1452

1453

1454

1455

1456

1457

1458

E.4 EXAMPLE TASK OF TYPE 4 - WALKING & FORWARD LOCOMOTION

1459

- Concepts : cyclical motion, repetition, background constraint

1460

- Description : In the input you will see an animation (or multiple frames) of pink human-shaped figures in a light-blue background. White wavy lines span the background and shift subtly as the pink figures travel diagonally from the top-left to the bottom-right. When a pink figure leaves the bottom-right edge, it reappears at the top-left edge at the same vertical position, making a continuous loop. The translucent blue overlay patterns remain visually consistent on top of the background while the white lines gently oscillate as the figures pass. To produce the output, replicate this cyclical motion fully at least once, preserving continuity. The final frame must line up so that if we start again from there, the pink figures continue repeating the same path through the light-blue background in a seamless loop.

1461

- Full web view is available at <https://gifarc.vercel.app/task/2767>.

1462

1463

1464

1465

1466

1467

1468

1469

1470

1471

1472

1473

1474

1475

1476

1477

1478

1479

1480

1481

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494

1495

1496

1497

1498

1499

1500

1501

1502

1503

1504

1505

1506

1507

1508

1509

1510

1511

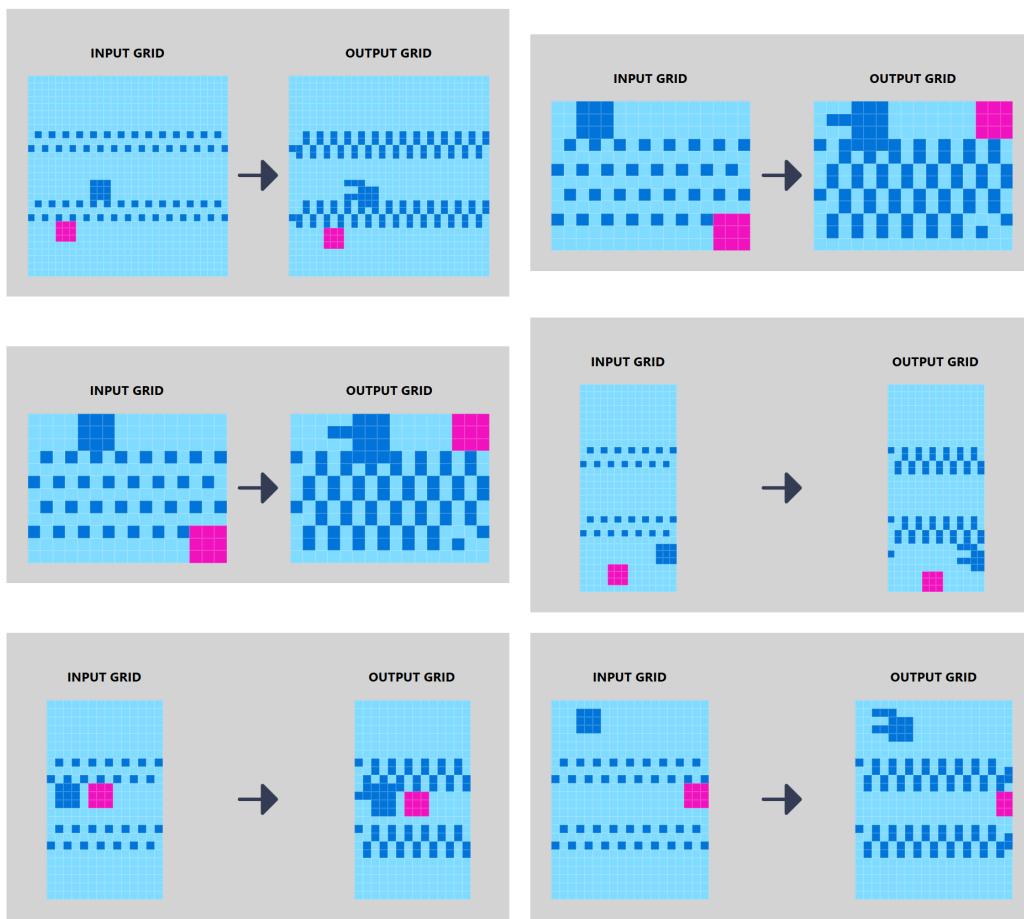
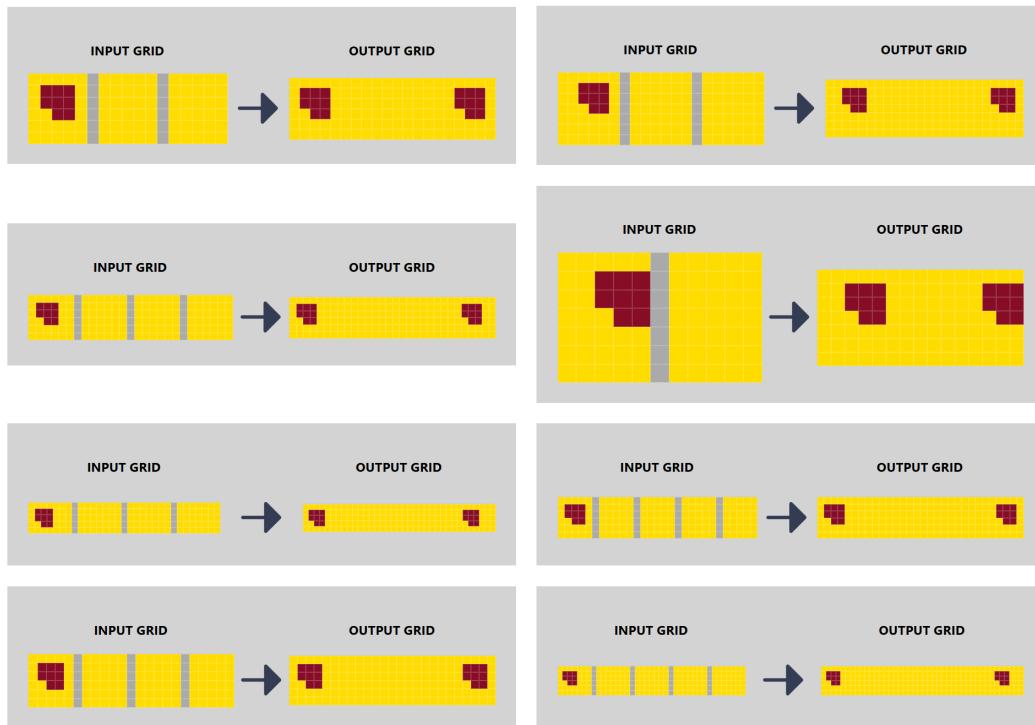


Figure 12: GIFARC-generated task 2767.

1566 E.6 EXAMPLE TASK OF TYPE 6 - PERIODIC MOVEMENT & HORIZONTAL LOOP
1567

- 1568 Concepts : cyclical motion, frame-based animation, horizontal repetition, background preservation
- 1569 Description : In the input you will see several frames depicting a single brown triceratops, side-profile, walking across a beige background. The frames show a gradual shift in the triceratops' position from left to right while its legs cycle through a walking gait. To create the output, stack or tile these frames into a continuous strip (or series) that repeats seamlessly. The background stays beige and does not change between frames, and the triceratops remains the same brown color and orientation. The final output shows a horizontally looping animation strip where the last frame transitions smoothly back into the first, preserving the cyclical walking motion and the uniform background.
- 1570
- 1571
- 1572
- 1573
- 1574
- 1575
- 1576
- 1577
- 1578
- 1579
- 1580
- 1581
- 1582
- 1583
- 1584
- 1585
- 1586
- 1587
- 1588
- 1589
- 1590
- 1591
- 1592
- 1593
- 1594
- 1595
- 1596
- 1597
- 1598
- 1599
- 1600
- 1601
- 1602
- 1603
- 1604
- 1605
- 1606
- 1607
- 1608
- 1609
- 1610
- 1611
- 1612
- 1613
- 1614
- 1615
- 1616
- 1617
- 1618
- 1619

1602 Figure 14: GIFARC-generated task 3690.
1603

1620

E.7 EXAMPLE TASK OF TYPE 7 - COLOR FLICKER & BLINKING

1621

- Concepts : cyclical color flicker, hinge constraints, multi-frame puzzle, looping animation

1622

• Description : The input consists of multiple frames showing a hand-drawn scene: a large circular shape with concentric scribbles/colors in its center, topped by a smaller hinged circle, with various scribbles at the right (ladder-like), left (darker scribble), and near the hinge (green). Over time, the concentric scribbles in the center circle flicker through different color states while all objects remain in the same relative positions.

1623

- Full web view is available at <https://gifarc.vercel.app/task/385>.

1624

1625

1626

1627

1628

1629

1630

1631

1632

1633

1634

1635

1636

1637

1638

1639

1640

1641

1642

1643

1644

1645

1646

1647

1648

1649

1650

1651

1652

1653

1654

1655

1656

1657

1658

1659

1660

1661

1662

1663

1664

1665

1666

1667

1668

1669

1670

1671

1672

1673

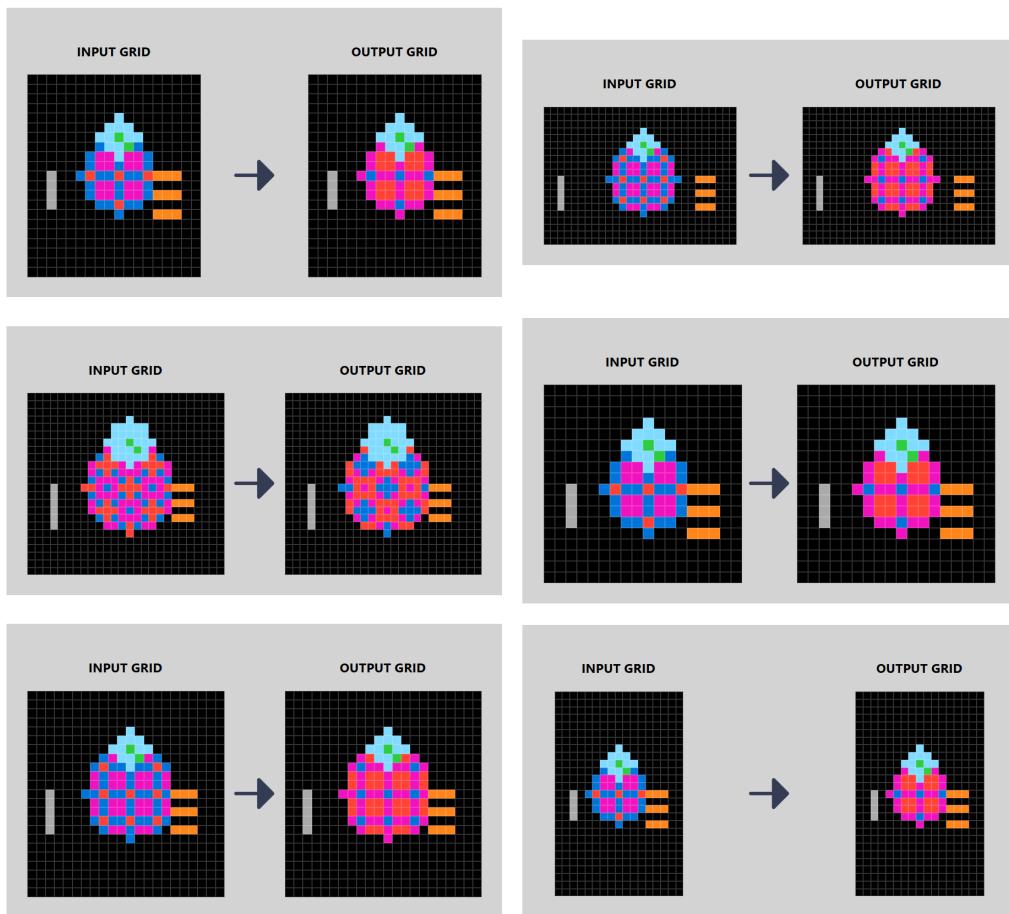
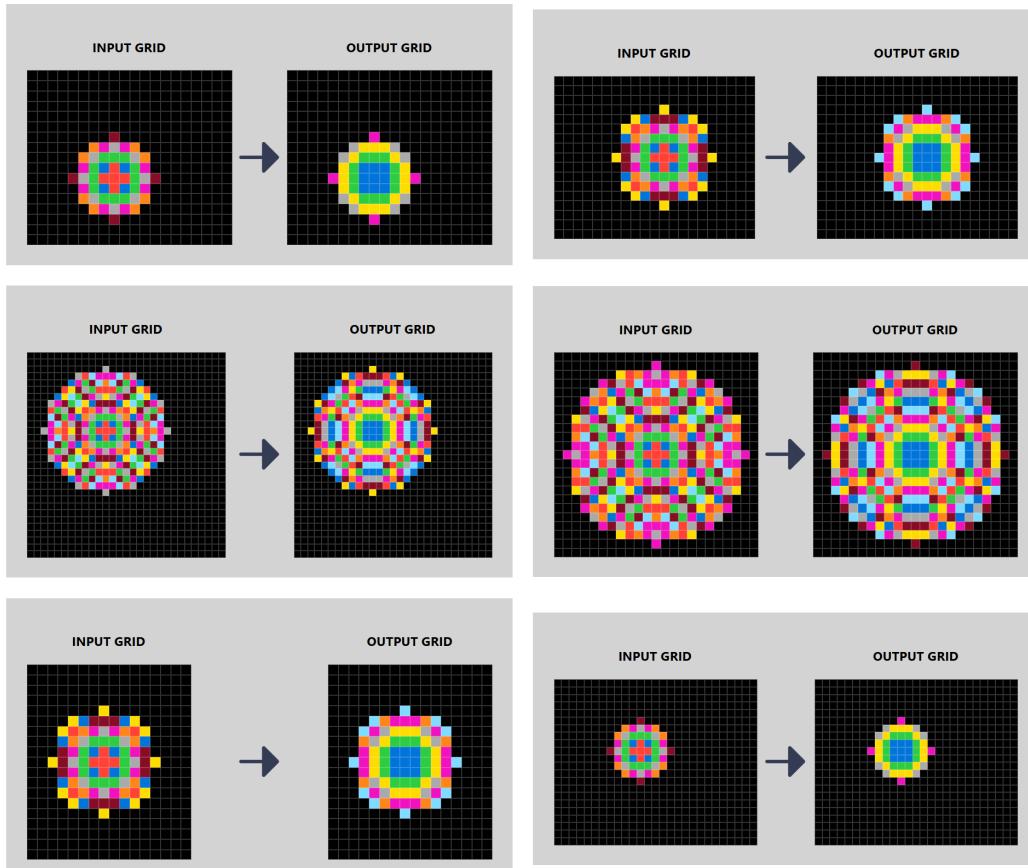


Figure 15: GIFARC-generated task 385.

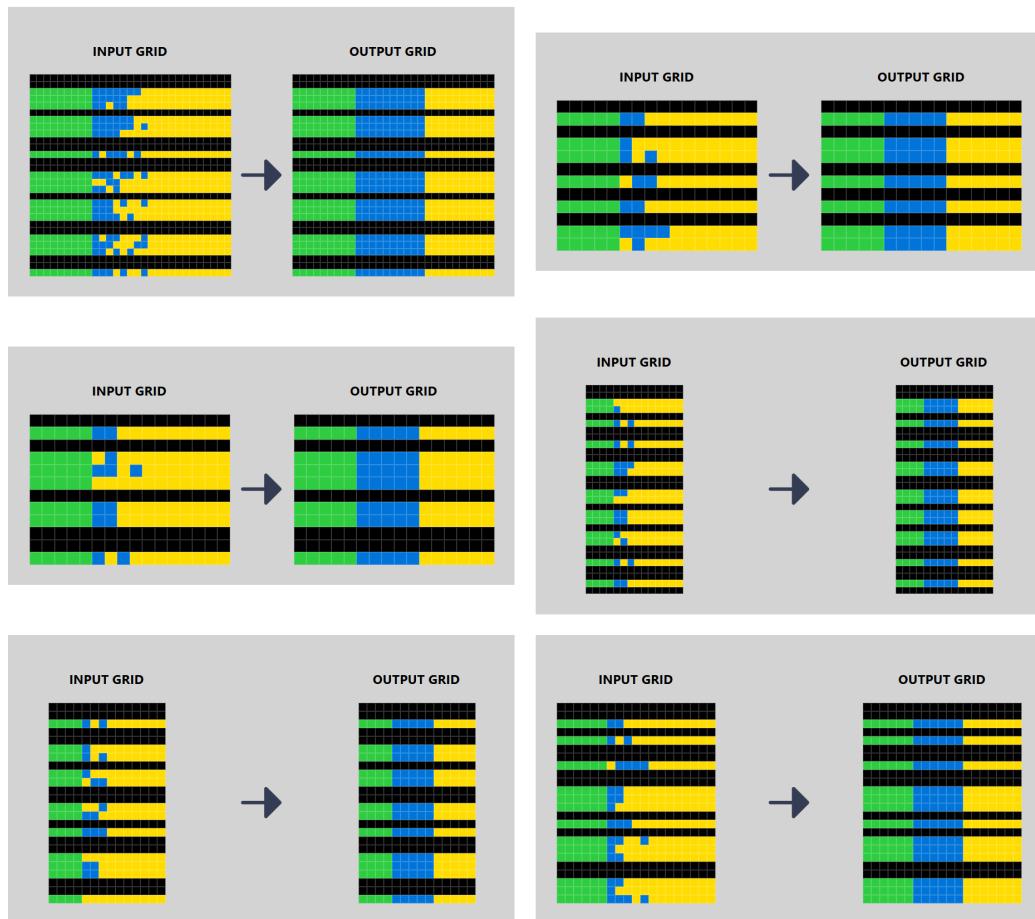
1674 E.8 EXAMPLE TASK OF TYPE 8 - GRADIENT & LAYERED COLOR CHANGES
1675

- 1676 Concepts : radial color cycling, symmetry, continuous transformations, layering
- 1677 Description : In the input, you will see a single circular shape on a purely black background. Inside
1678 the circle is a radial, symmetrical pattern composed of multiple colors. Each concentric ring of the
1679 circle shifts colors in a periodic cycle, from an inner ring (earliest phase) to an outer ring (later
1680 phase), repeatedly. The circle retains its size and position, and the background remains black.
- 1681 Full web view is available at <https://gifarc.vercel.app/task/117>.

1711 Figure 16: GIFARC-generated task 117.
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727

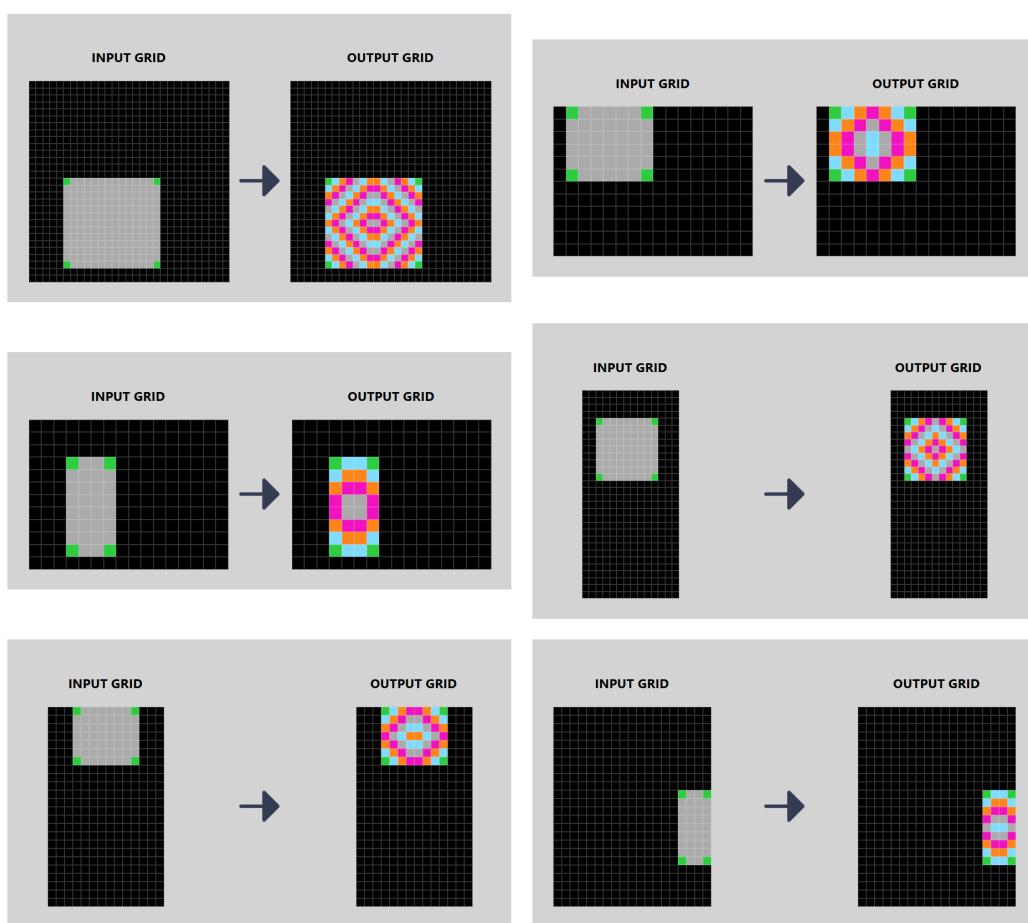
1728 E.9 EXAMPLE TASK OF TYPE 9 - GLITCH & BREAKING PATTERNS
1729

- 1730 Concepts : glitch patterns, color flickering, repetitive animation, partial stability
- 1731 Description : In the input you will see multiple horizontal bars stacked vertically over a black
1732 background. Each bar is composed of green, purple, and yellow pixels arranged in rectangular
1733 stripes. Some small clusters of yellow pixels flicker like digital noise, intensifying toward the center
1734 and right side of each bar. The bars remain in the same positions throughout the sequence.
- 1735 Full web view is available at <https://gifarc.vercel.app/task/410>.

1767 Figure 17: GIFARC-generated task 410.
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781

1782 E.10 EXAMPLE TASK OF TYPE 10 - WAVE & DIAGONAL FLOW
1783

- 1784 Concepts : cyclical pulsing, polygonal wave propagation, time-lapse transformation
- 1785 Description : In the input, you will receive a sequence of frames representing a dark polygonal
1786 surface with green glowing points at each polygon vertex. Each frame gradually shifts the brightness
1787 of the polygons in wave-like motions, causing shimmering highlights that travel along the connected
1788 polygon faces. The initial frame and the final frame share the same luminous intensity, forming
1789 a seamless loop of pulsing and returning. Your task: 1. Identify the green points at the vertices,
1790 which remain spatially fixed throughout the sequence. 2. Track the wave-like shimmering across
1791 the polygon faces. 3. Ensure that the final frame's brightness pattern matches the initial frame's,
1792 capturing the cyclical nature of the pulse. 4. Output the entire transformation as a collection
1793 of frames (or a metadata structure) that visually loops back to the start. The key principles are:
1794 - The polygon mesh arrangement does not change; only the luminous intensity on the polygon
1795 faces and green vertices fluctuates. - Shimmering waves spread across the surface repeatedly, then
1796 recede, creating a looping timeline. - The result is a time-lapse style output in which the final state
1797 seamlessly resets to the first state, preserving the thought of continuous pulsing.
- 1798 Full web view is available at <https://gifarc.vercel.app/task/1354>.

1829 Figure 18: GIFARC-generated task 1354.
1830
1831
1832
1833
1834
1835

1836 E.11 EXAMPLE TASK OF TYPE 11 - GRAVITY & LIQUID FLOW
1837

- 1838 • Concepts : time-lapse, incremental changes, layering, chart visualization
- 1839 • Description : In the input you will see multiple "frames" of a chart with vertical bars and a pink
- 1840 line, each frame on a separate grid. The black background, rectangular grid, and bounding box
- 1841 remain constant across these frames. In each subsequent frame, the vertical bars shift in height and
- 1842 flicker in intensity, while the pink line oscillates up and down with a net upward trend.
- 1843 • Full web view is available at <https://gifarc.vercel.app/task/1413>.

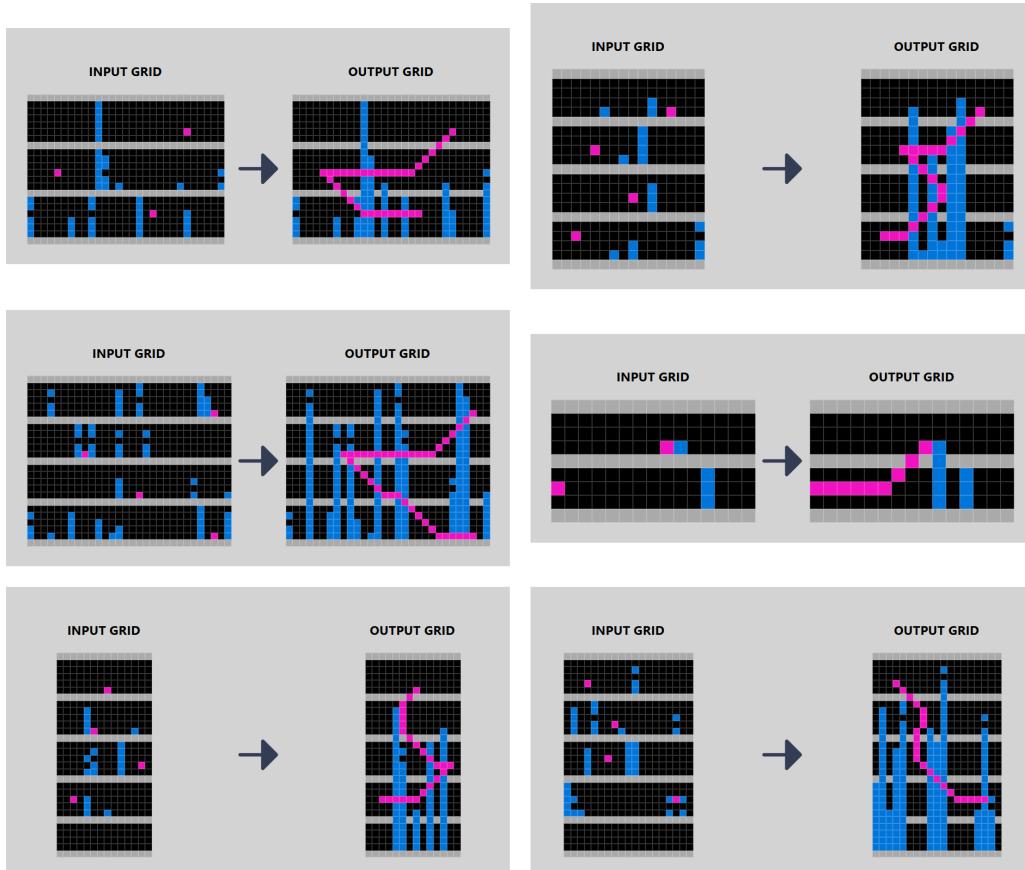


Figure 19: GIFARC-generated task 1413.

1890

E.12 EXAMPLE TASK OF TYPE 12 - SLOW ENVIRONMENTAL CHANGE

1891

- Concepts : time-lapse, incremental changes, layering, chart visualization

1892

- Description : In the input you will see multiple "frames" of a chart with vertical bars and a pink line, each frame on a separate grid. The black background, rectangular grid, and bounding box remain constant across these frames. In each subsequent frame, the vertical bars shift in height and flicker in intensity, while the pink line oscillates up and down with a net upward trend.

1893

- Full web view is available at <https://gifarc.vercel.app/task/1554>.

1894

1895

1896

1897

1898

1899

1900

1901

1902

1903

1904

1905

1906

1907

1908

1909

1910

1911

1912

1913

1914

1915

1916

1917

1918

1919

1920

1921

1922

1923

1924

1925

1926

1927

1928

1929

1930

1931

1932

1933

1934

1935

1936

1937

1938

1939

1940

1941

1942

1943

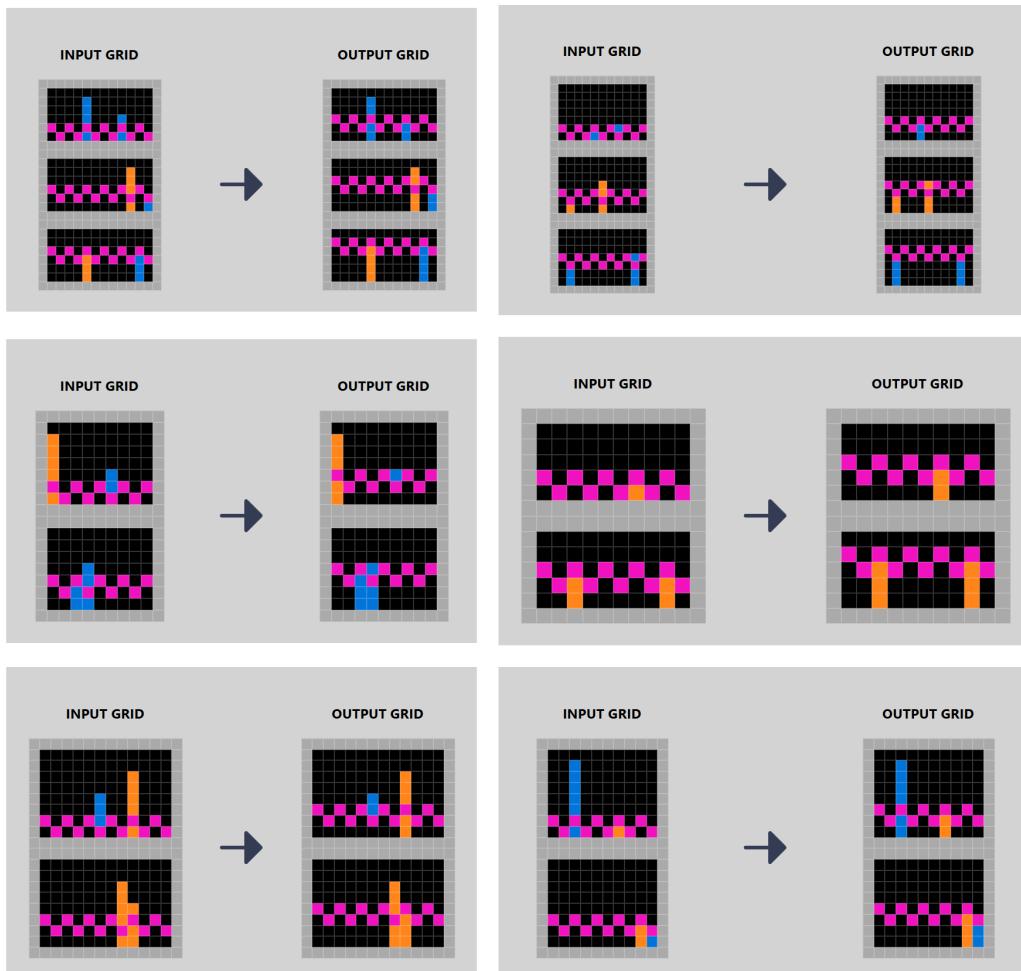


Figure 20: GIFARC-generated task 1554.

1998

E.14 EXAMPLE TASK OF TYPE 14 - ATTACH/DETACH CLUSTERS

1999

- Concepts : multi-frame transformation, attachment/detachment, motion, object grouping

2000

• Description : The input represents a time-sequence (e.g., multiple "frames") showing a suited host seated at a desk with several static background objects (gift bag, city backdrop, building spire). The host reaches for a black mug, and we see that the tie is inadvertently attached to the mug (one frame shows the tie moving upward with the mug). In subsequent frames, the host detaches the tie from the mug and returns both to their normal positions. The core principle to illustrate is that if two objects are "attached" they move together until they become detached. Your output is the final frame where the tie is no longer stuck to the mug. The puzzle solution must show how the tie becomes free again, returning to its intended resting position, while the mug is back on the desk.

2001

- Full web view is available at <https://gifarc.vercel.app/task/7128>.

2002

2003

2004

2005

2006

2007

2008

2009

2010

2011

2012

2013

2014

2015

2016

2017

2018

2019

2020

2021

2022

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

2033

2034

2035

2036

2037

2038

2039

2040

2041

2042

2043

2044

2045

2046

2047

2048

2049

2050

2051

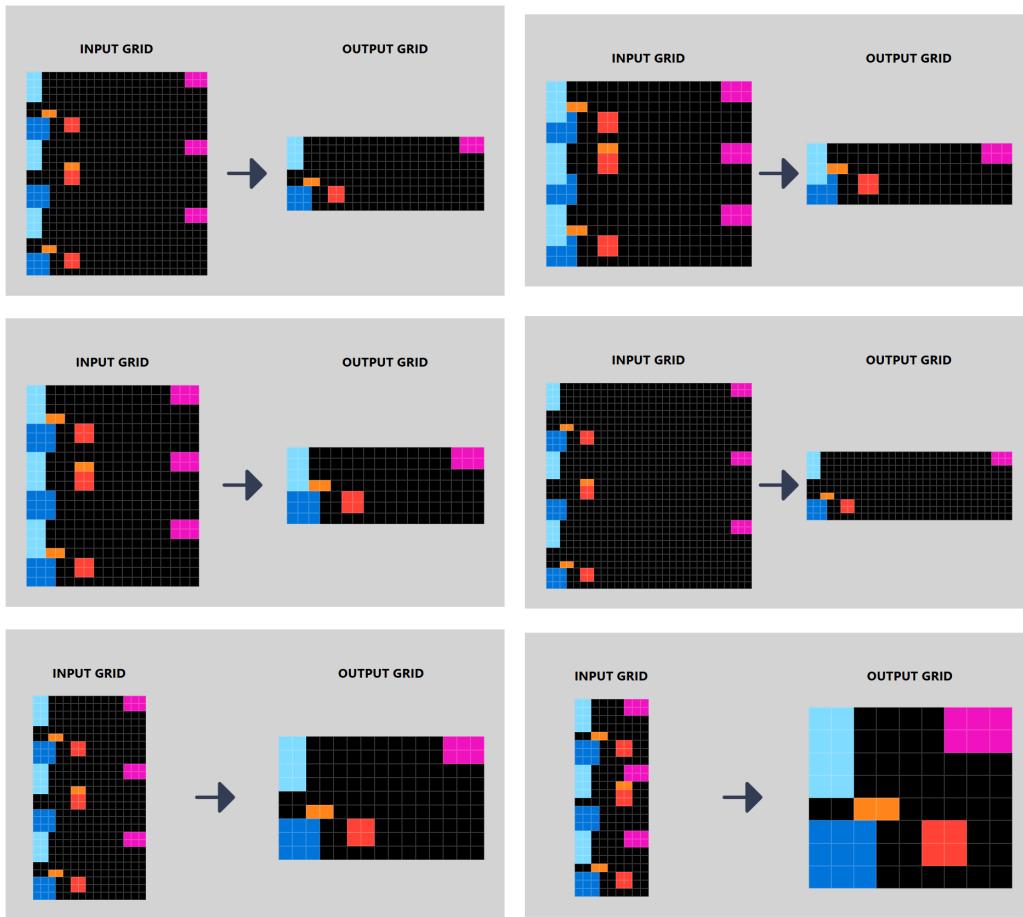


Figure 22: GIFARC-generated task 7128.

2052 E.15 EXAMPLE TASK OF TYPE 15 - LAYER SEPARATION & MERGING
2053

- 2054 Concepts : color gradient, layered squares, temporal progression, partial transparency
- 2055 Description : In the input you will see a multi-frame sequence of an overlapping grid of squares
2056 starting in dark blues at the top, transitioning gradually to lighter, more yellow tones at the bottom.
2057 The squares remain in a consistent arrangement, but their colors shift slightly from frame to frame.
2058 Overlaps create partial transparency effects that slightly modify the underlying tones. To produce
2059 the output, you must show the final frame in which the color gradient transitions all the way to
2060 lighter shades near the bottom. Capture the entire transformation by preserving the arrangement of
2061 squares, ensuring that no abrupt color changes occur between consecutive frames, and maintaining
2062 a seamless progression of hues through the final arrangement.
- 2063 Full web view is available at <https://gifarc.vercel.app/task/214>.

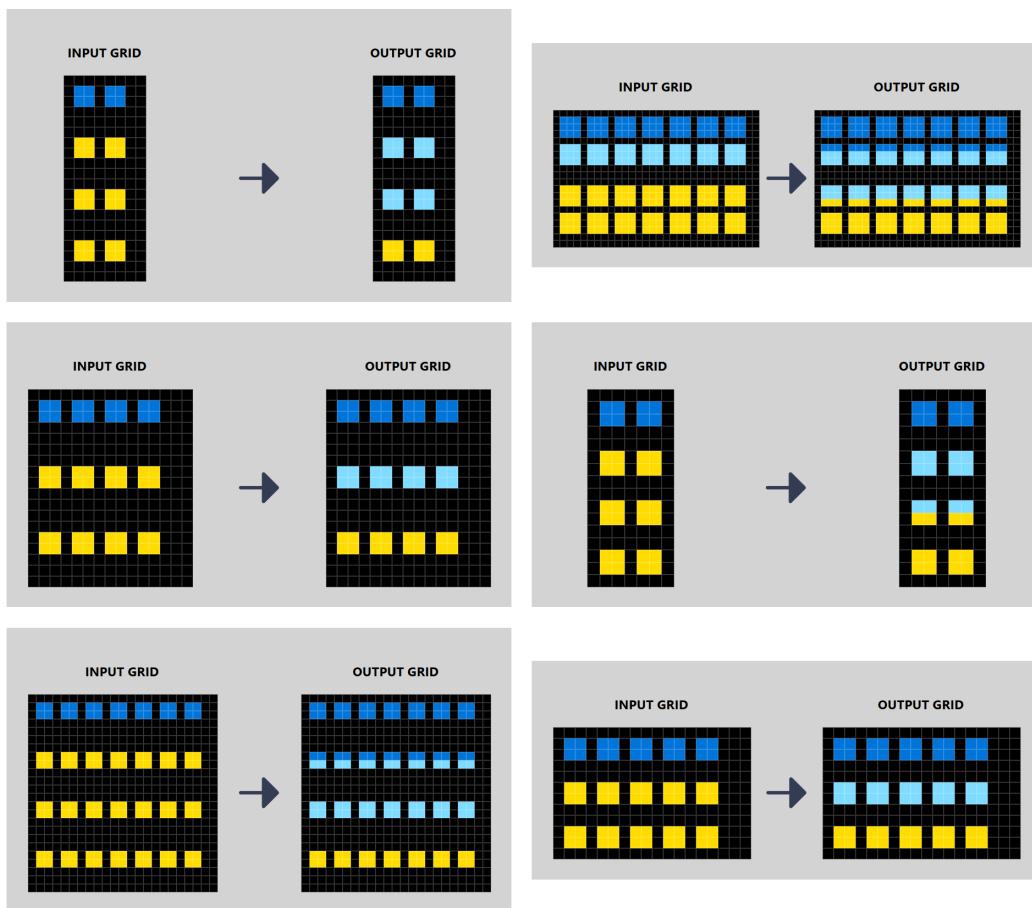


Figure 23: GIFARC-generated task 214.

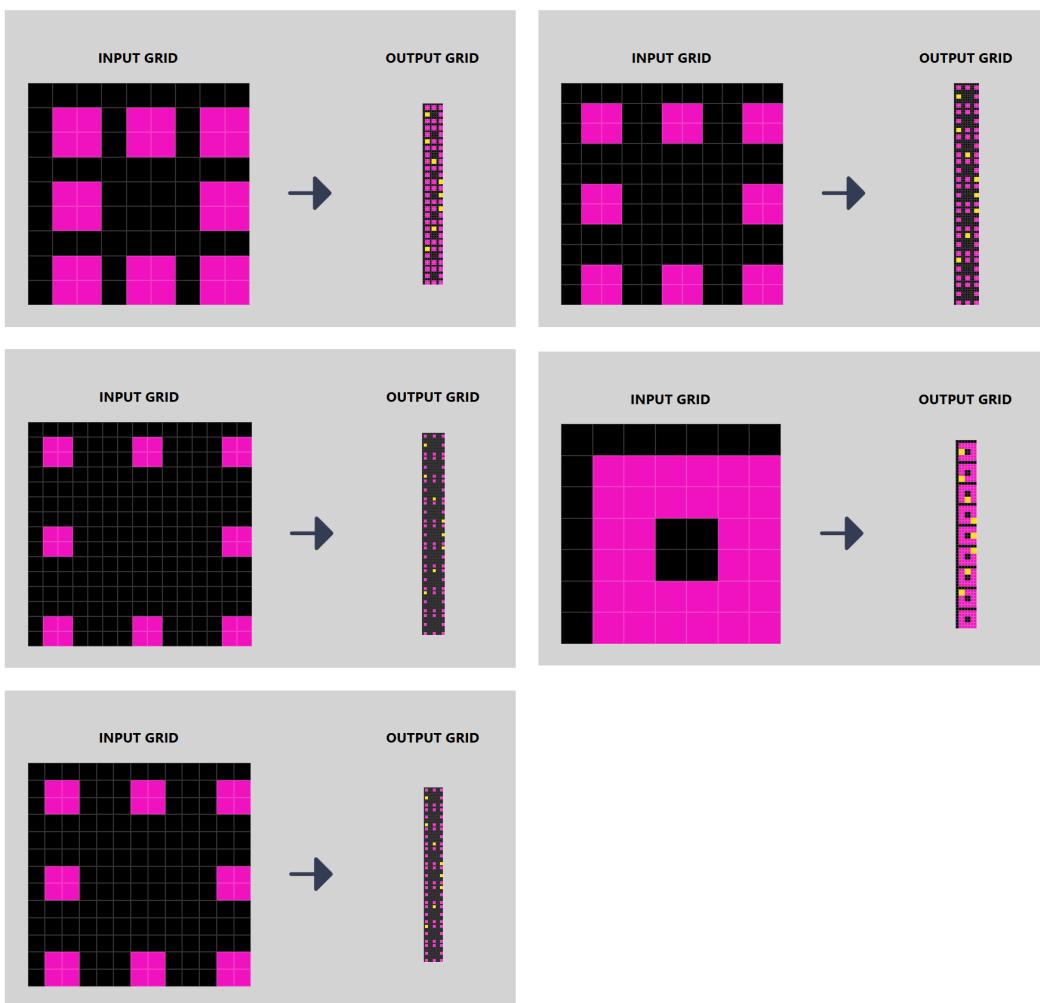
2106 E.16 EXAMPLE TASK OF TYPE 16 - TEXT & PUNCTUATION TRANSFORMATION
2107

- 2108 • Concepts : reflection, wave distortion, layering
- 2109 • Description : In the input grid, you will see: 1. A forest background occupying the top portion.
2110 2. The word "RELAX" in large, two-toned letters at the center of the grid. 3. A black area
2111 (representing water) at the bottom. To make the output: 1. Reflect both the forest background and
2112 the "RELAX" text onto the black water region, mirroring them vertically. 2. Apply a horizontal
2113 wave distortion row by row to the mirrored content, simulating water ripples. 3. Preserve the
2114 original forest and text unchanged in their original positions, keeping them static above the water.
2115 The core principle is to create a continuous rippling motion in the lower part that reflects the static
2116 background and text above.
- 2117 • Full web view is available at <https://gifarc.vercel.app/task/3896>.

2151 Figure 24: GIFARC-generated task 3896.
2152
2153
2154
2155
2156
2157
2158
2159

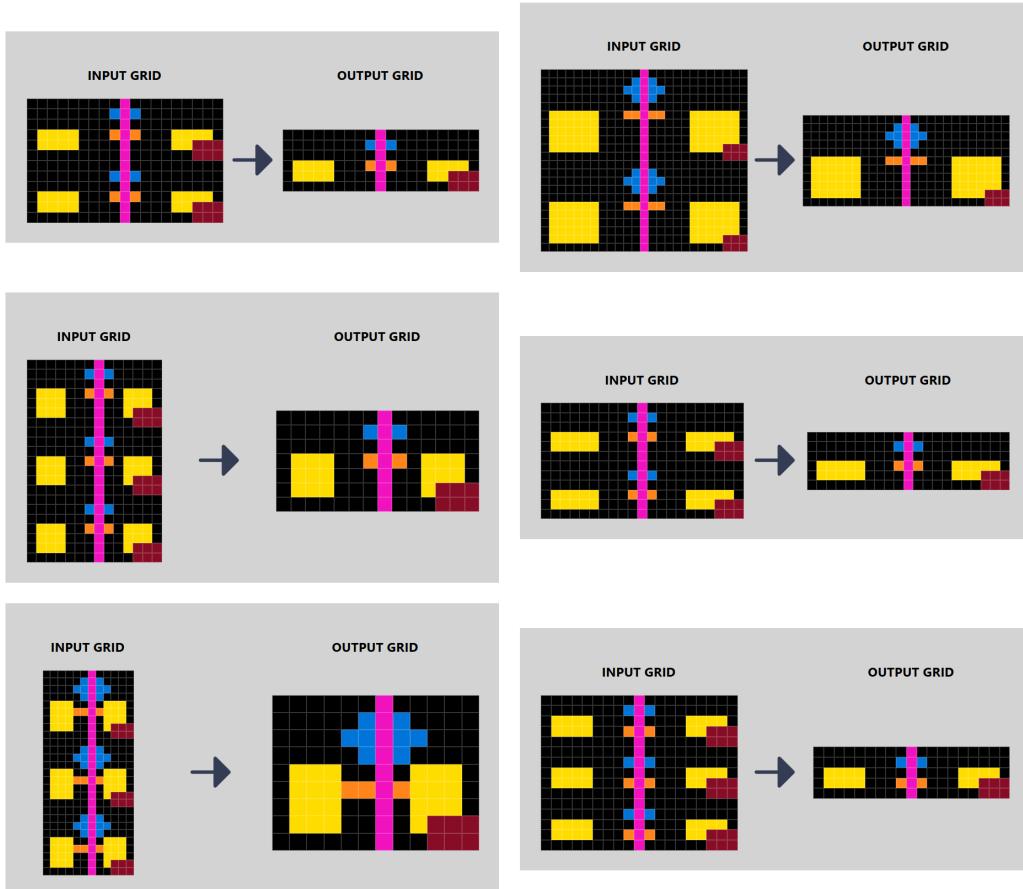
2160 E.17 EXAMPLE TASK OF TYPE 17 - MINIMAL MOTION OVERLAY
2161

- 2162 Concepts : ring arrangement, sequential flipping, cyclical motion
- 2163 Description : In the input you will see a grid composed of multiple rows (or columns), each
2164 representing a separate "frame" of a green cross-like shape (with a purple outline) on a blue
2165 background. 1. The shape is supposed to expand and contract in a symmetrical, cyclical pattern
2166 around its center, but the given frames may be scrambled or have missing corners/edges. 2. Reorder
2167 these frames (and fix any missing corners/edges) to restore a clean, cyclical sequence where the
2168 cross-like shape starts in its initial state, morphs outward, and returns symmetrically to a state
2169 matching the initial frame. 3. The shape should retain its green fill, purple outline, and be fully
2170 contained in the blue background across all frames. 4. The final output is a single grid (or list of
2171 frames) showing the corrected sequence in order, with all frames centered and fully symmetrical,
2172 completing one full expansion-contraction cycle.
- 2173 • Full web view is available at <https://gifarc.vercel.app/task/9842>.

2196 Figure 25: GIFARC-generated task 9842.
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213

2214 E.18 EXAMPLE TASK OF TYPE 18 - STATIC VERIFICATION & NO CHANGE
2215

- 2216 Concepts : multi-frame static verification, scene consistency, no-change detection
- 2217 Description : In the input, you will see several frames that together depict a stylized scene: - A
2218 top sphere positioned at the upper center - A curved band placed directly below the sphere - Two
2219 large symmetrical side shapes in yellow - A vertical pink line in the center - A signature text in
2220 the bottom right corner These elements are repeated exactly across all frames with no movement
2221 or color transitions. To make the output, you must check if all frames are truly identical. If they
2222 are, return a single-frame grid replicating the scene exactly. If there is any discrepancy (in shape,
2223 color, or position) among the frames, then the output should be a black canvas of the same size,
2224 indicating the scenes are not perfectly static.
- 2225 Full web view is available at <https://gifarc.vercel.app/task/413>.

2256 Figure 26: GIFARC-generated task 413.
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267

2268

E.19 EXAMPLE TASK OF TYPE 19 - FRACTAL EXPANSION & SELF-SIMILAR REPEATS

2269

- Concepts : fractal expansions, symmetrical pulsations, radial transformations, iterative growth
- Description : In the input, you will see a sequence of grids (frames) on a black background. Each grid depicts: - A large circular region in the center, with multiple concentric rings and radial lines diverging outward. - A stacked-curve fractal anchored at the bottom-left corner. - A set of spire fractals anchored at the bottom-right corner. As you move from one frame to the next in the input, these elements undergo iterative transformations: - The radial lines repeatedly shift in thickness and visual intensity, while preserving rotational symmetry. - The concentric rings in the circular region pulsate by alternately expanding and contracting. - The stacked-curve fractal on the bottom-left changes its curve density, fractally adding or removing segments. - The right-side spire fractals expand and contract in repeated vertical segments. Your task is to replicate and apply these transformations for the entire sequence, and produce the final frame of the animation as the output: 1) Ensure the anchoring of the fractals at the bottom edge remains the same. 2) Preserve the radial symmetry of the central circular region and its common center point. 3) For each pulsation step in the input, magnify or contract the rings, lines, and fractals accordingly. 4) Continue until the final pulsation step is reached. That final state is your output grid. The essential principle is that symmetry-based fractal transformations and repeated cyclical expansions/contractions produce the overall pulsating effect. By reflecting each iterative change step by step, you reconstruct the final pulsating pattern visible at the end of the sequence.

2270

- Full web view is available at <https://gifarc.vercel.app/task/155>.

2271

2272

2273

2274

2275

2276

2277

2278

2279

2280

2281

2282

2283

2284

2285

2286

2287

2288

2289

2290

2291

2292

2293

2294

2295

2296

2297

2298

2299

2300

2301

2302

2303

2304

2305

2306

2307

2308

2309

2310

2311

2312

2313

2314

2315

2316

2317

2318

2319

2320

2321

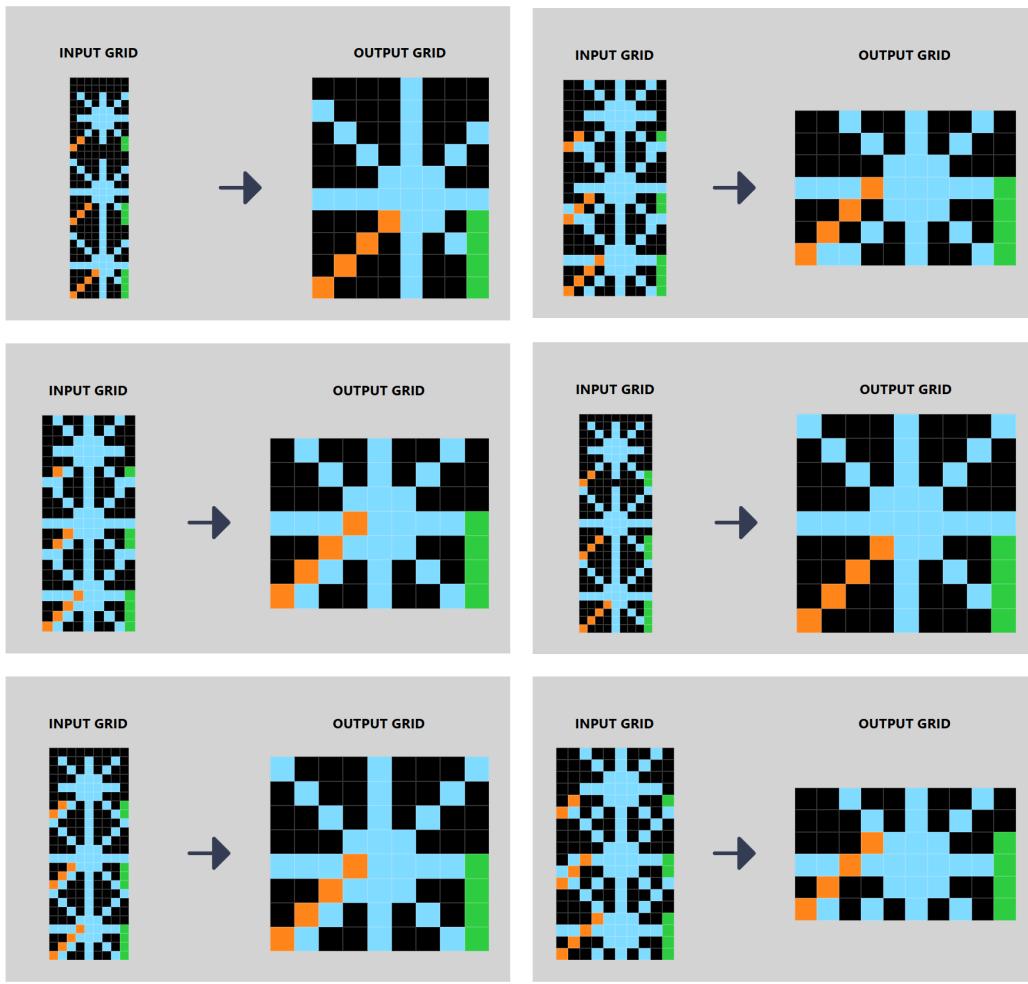
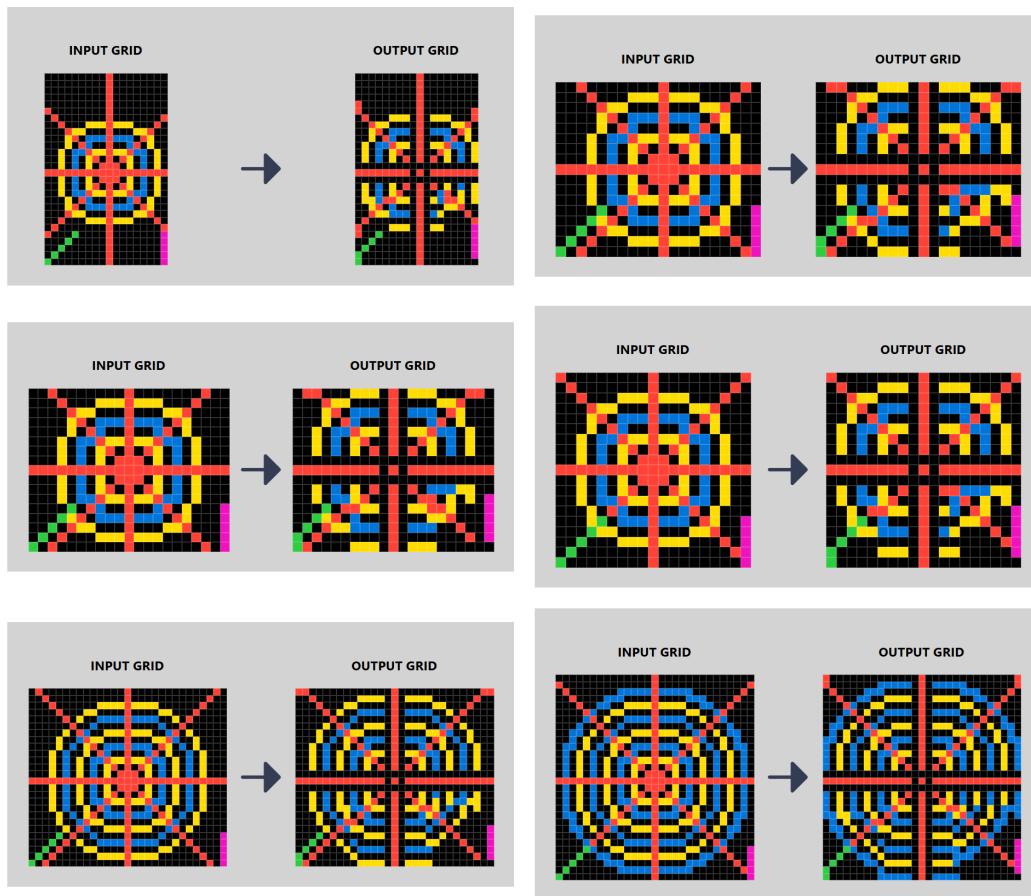


Figure 27: GIFARC-generated task 155.

2322 E.20 EXAMPLE TASK OF TYPE 20 - SEQUENTIAL PATTERN GROWTH & TRANSITION
2323

- 2324 Concepts : fractal expansions, symmetrical pulsations, radial transformations, iterative growth
- 2325 Description : In the input, you will see a sequence of grids (frames) on a black background. Each
2326 grid depicts: - A large circular region in the center, with multiple concentric rings and radial
2327 lines diverging outward. - A stacked-curve fractal anchored at the bottom-left corner. - A set of
2328 spire fractals anchored at the bottom-right corner. As you move from one frame to the next in
2329 the input, these elements undergo iterative transformations: - The radial lines repeatedly shift in
2330 thickness and visual intensity, while preserving rotational symmetry. - The concentric rings in the
2331 circular region pulsate by alternately expanding and contracting. - The stacked-curve fractal on the
2332 bottom-left changes its curve density, fractally adding or removing segments. - The right-side spire
2333 fractals expand and contract in repeated vertical segments. Your task is to replicate and apply these
2334 transformations for the entire sequence, and produce the final frame of the animation as the output:
2335 1) Ensure the anchoring of the fractals at the bottom edge remains the same. 2) Preserve the radial
2336 symmetry of the central circular region and its common center point. 3) For each pulsation step
2337 in the input, magnify or contract the rings, lines, and fractals accordingly. 4) Continue until the
2338 final pulsation step is reached. That final state is your output grid. The essential principle is that
2339 symmetry-based fractal transformations and repeated cyclical expansions/contractions produce the
2340 overall pulsating effect. By reflecting each iterative change step by step, you reconstruct the final
2341 pulsating pattern visible at the end of the sequence.

- 2342 • Full web view is available at <https://gifarc.vercel.app/task/2061>.

2372 Figure 28: GIFARC-generated task 2061.
2373
2374
2375