Implementation Approach Summary

1. Classifier-Based Reasoning Modules

- **Classifiers:**

  - Gap-Filling: Detects incomplete shapes using template matching/morphological analysis and completes -- patterns with rule-based or CNN-driven extrapolation.
  - Color-Matching: Infers color mappings by comparing training examples; applies mappings with iterative - - per-cell or object-based replacement.
  - Diagonal Pattern: Uses directional filters or coordinate checks.
  - Object Transformation: Object Movement/Rotation/Mirroring, etc..
  - more…

- **Interface:** Each classifier implements a method (predict(input_grid, train_examples)) to output a grid and a confidence score.

2. Decision-Making Strategy

- **Confidence Scoring:**

  - Each classifier verifies consistency across training examples, quantifies pattern specificity, and adjusts via heuristics.
  - Scores are normalized (range [0,1]).

- **Decision:**

  - Selection: Choose the candidate with the highest confidence if it exceeds a predefined threshold.
  - Multiple candidates: If outputs are close, merge multiple classifiers (combine color changes with gap-filling).
  - Fallback: Call an LLM with Chain-of-Thought module when all classifiers are uncertain (under the threshold). A possible problem/solution is when the LLM itself performs better than our specific classifiers in general.

3. Chain-of-Thought (CoT) Model Integration

- **Model Choice & Prompting:**

  - Use an LLM with chain-of-thought prompting.
  - Encode grids in a simple text format (rows of digits or coordinate lists) or a screenshot of the grid and - send as a picture, alongside high-level summaries.

- **Iterative Refinement:**

  - Generate an initial CoT solution.
  - Run classifiers to critique the output (check if rotation or color mapping matches training examples).
  - Feed back specific feedback to the CoT model and iterate.

4. Implementation -Examples-

Training Phase (classifiers):

```
for classifier in [gap_fill, color_match, diagonal, object_transform, ...]:
    training_data = collect_training_tasks_for(classifier)
    X = []
    y = []
    for task in training_data:
        features = extract_features(task.input_grids,
task.output_grids)
        label = 1  # since these tasks are of the classifier's type
        X.append(features); y.append(label)
        # also collect some negative examples
        neg_tasks = sample_other_tasks(not_type=classifier.type)
        for t in neg_tasks:
            features = extract_features(t.input_grids, t.output_grids)
            X.append(features); y.append(0)
    # Train a classifier or model on X, y
    model = train_model(X, y)  # could be CNN, sklearn model, anything
    save_model(classifier.name, model)
```

Inference Phase:

```
def solve_task(train_inputs, train_outputs, test_input):
    task_features = extract_features(train_inputs, train_outputs)
    #  Run classifiers (potentially filtered by a quick guess?)
    candidates = []
    for classifier in [gap_fill, color_match, diagonal, object_transform]:
        if classifier.is_applicable(task_features):  # the optional quick check
            output_grid = classifier.solve(test_input, train_inputs,
train_outputs)
            confidence = classifier.score_confidence(train_inputs,
train_outputs, output_grid)
            candidates.append((output_grid, confidence, classifier.name))
    # select best candidate
    if not candidates:
        # No classifier applied (should not normally happen if is_applicable is not strict)
        return use_chain_of_thought(train_inputs, train_outputs, test_input)
    # pick candidate with max confidence
    candidates.sort(key=lambda x: x[1], reverse=True)
    best_output, best_conf, best_name = candidates[0]
    # check if second best is close and conflicting?
    if len(candidates) > 1 and abs(best_conf - candidates[1][1]) <
CONFIDENCE_MARGIN? DO
        merged = try_merge(best_output, candidates[1][0], train_inputs, train_outputs)
        if merged:
            best_output = merged
            best_name = best_name + "+" + candidates[1][2]
    # if confidence is below threshold or conflict unresolved, use CoT
    if best_conf < CONFIDENCE_THRESHOLD:
        cot_output = use_chain_of_thought(train_inputs, train_outputs, test_input)
        return cot_output
    else:
        return best_output
```

Thoughts:

Vectorized Feature Encoding: For certain pattern recognitions, especially for simpler classifiers, a fixed-length feature vector can be effective. We compile relevant features into a vector – for example, [number of objects, count of each color, symmetry flag (0/1), max object size, etc]. This vector can feed into a classical machine learning classifier to predict the task type or parameters. One use-case: a small logistic regression on such features could act as a task-type classifier that predicts which of the specialized classifers should handle the task (this complements the confidence mechanism discussed earlier).

CNNs: we can feed the raw grid (or a normalized version of it?) into a CNN for pattern recognition. Each grid treated as a small image. A small CNN can be trained to detect specific pattern types (CNN outputs a probability that the pattern is a diagonal line, another might detect if an object has been rotated). We use CNNs in a targeted way: each specialized classifier (gap-filling, color-matching, etc.) could include a mini-CNN that assists in its detection step. The gap-filling classifier might use a CNN to classify what shape is incomplete, the diagonal classifier might have a CNN to strengthen detection of diagonal arrangements, etc. By training these on synthetic grid patterns or on the ARC training data labeled by type, we make the classifiers "learned".

Classifiers uncertainty: if color-matching and object-movement both have moderate confidence and each partially explains the task, we can apply the color mapping rule first, then the movement rule, and compare to the training outputs. This essentially tests if a sequence of classifiers solves the puzzle (useful when we have 3+ moderate scores?)

HOW?: how to handle a problem if two classifiers want opposite goals (one cannot happen with the other)?

Efficiency: Attempting all classifiers on every task can be costly. If that's a major problem a preliminary quick classification of task type can suggest which "experts" to run (for instance, skip the diagonal classifier if no diagonal lines are detected at all by a quick scan). Not sure how to implement this yet.

CoT Feedback: The system uses an iterative loop where the CoT model initially produces a solution, which is then critiqued by our classifiers. The classifiers provide targeted feedback that is fed back to the CoT model via updated prompts (Here we can use another LLM to provide the feedback, but prehab this is too much? A stupid one can be used here), allowing the solution to be refined iteratively until it matches training examples, but if confidence remains low, the system recognizes the task as unsolved.