ARC-AGI: Combined Approaches to ARC Puzzle Solving

> Three standalone pipelines demonstrating language- and vision-driven methods to solve ARC (Abstraction and Reasoning Corpus) tasks.

Table of Contents

Repository Structure

```
arc-agi/
├── arc_agi(gpt_4).py      # GPT-4 natural language reasoning
├── arc_agi(bert).py       # BERT-based rule discovery
├── arc_agi(cnn+gnn).py    # Hybrid CNN + GNN vision pipeline
├── functions.py           # Primitive grid transformations
├── requirements.txt       # Python dependencies
└── README.md              # This document
```

Setup

1. **Clone the repository**

   ```
   git clone https://github.com/<your-org>/arc-agi.git
   cd arc-agi
   ```

2. **Install dependencies**

   ```
   pip install -r requirements.txt
   ```

3. **Configure API keys** (if applicable)

   ```
   export OPENAI_API_KEY="<your_openai_key>"
   export HF_TOKEN="<your_huggingface_token>"
   ```

4. **Data placement**

    - Place ARC JSON task files under `data/arc/` or adjust paths in scripts.

Approaches

1. GPT-4 Approach

Script: `arc_agi(gpt_4).py`

Goal: Leverage GPT-4 to generate natural-language descriptions of grid transformations.

Workflow:

1. Load example pairs of input/output grids.
2. Send few-shot prompts to GPT-4 for each training example.
3. Parse GPT-4's textual output into actionable rules (via regex or manual logic).
4. Apply parsed rules to test inputs and compute accuracy.

```
python arc_agi\(gpt_4\).py
```

> 💡 *Tip:* Customize prompts and parsing logic directly within the script.

---

2. BERT Approach

Script: `arc_agi(bert).py`

Goal: Fine-tune BERT as a token-classification model to identify transformation operations.

Workflow:

1. Serialize input→output transformations into a token sequence.
2. Train a BERT model to label tokens corresponding to specific operations.
3. At inference, run BERT on new token sequences to predict operations.
4. Apply predicted operations to generate solution grids.

```
python arc_agi\(bert\).py
```

> 🔑 *Note:* This pipeline is self-contained and does **not** call GPT-4.

---

3. CNN + GNN Approach

Script: `arc_agi(cnn+gnn).py`

Goal: Combine hand-crafted primitives with a learned vision+graph model to maximize coverage.

Workflow:

1. **Primitive Matching Analysis**
   - Apply each function in `functions.py` to training examples.
   - Count how many examples each primitive solves.
2. **Data Preparation**
   - Normalize grid dimensions and map colors to indices.
   - Construct PyTorch `Dataset`/`DataLoader` for training and evaluation.
3. **Model Architecture**
   - **CNN** layers for local feature extraction.
   - **GNN** layers (e.g., GraphConv) for relational reasoning over pixel nodes.
4. **Training**
   - Optimize cross-entropy loss on pixel-wise predictions.
   - Track train/validation accuracy.
5. **Inference**
   - First attempt primitives in descending order of match count.
   - Fall back to CNN+GNN predictions for unmatched tasks.

```
python arc_agi\(cnn+gnn\).py
```