RASO

Below is a step-by-step guide detailing how you can re-implement the RASO pipeline described in the paper “Generate then Select: Open-ended Visual Question Answering Guided by World Knowledge” (Fu et al., 2022) along with insights drawn from the appendix.

---

1. Overview

The RASO approach is built on a two-stage “generate-then-select” strategy:

• **Stage 1 – Answer Choices Generation:**
A pre-trained language model (PLM) is prompted (using two variants of prompts) to generate a long list of possible answers for each image–question pair. The image is first converted into a textual context via image captioning and object tagging. In-context examples (selected via similarity, e.g. using CLIP embeddings) are provided so that the PLM (e.g. GPT-J, UL2, OPT, or Codex) outputs diverse answer candidates.

• **Stage 2 – Answer Selection:**
A lightweight answer selection model then picks the correct answer from the candidate list. Importantly, the model is guided by a chain-of-thought (CoT) rationale generated using a fixed prompt (see Appendix A.1 in the paper), which helps the model reason over the image, question, and candidate answers.

The authors show that this method greatly expands external world knowledge coverage (by over 20% compared to previous approaches) and improves VQA performance on benchmarks like OK-VQA.

---

2. Data Preparation

a. Image to Text Conversion

For every image–question pair in your dataset (e.g. OK-VQA):

1. **Generate a Caption:**
   Use an image captioning model (e.g. BLIP-2 or any state-of-the-art captioner) to produce a natural language description of the image.

2. **Obtain Object Tags:**
   Use an image tagging API (e.g. Microsoft Azure Cognitive Services) to extract a list of relevant object tags.
3. **Combine into Context:**
   Merge the caption and tags to form a textual context that will be fed into your prompts.

3. Answer Choices Generation

a. Prompt Construction

There are two prompts:

- **PromptQ:** Uses only the question and a few in-context examples.
- **PromptQC:** Uses both the question and the additional textual context (caption + tags).

Both prompts include a clear instruction (e.g. “Please list all the possible answers to the question.”) and are augmented with in-context examples drawn from the training set. These examples are selected based on similarity (e.g. using CLIP embeddings) so that the examples are most relevant to the current image–question pair. The paper suggests using 16-shot examples on test data (4-shot during training, if you have multiple gold answers).

b. PLM Generation

Using a PLM (set at a low temperature, e.g. 0.001, and with a maximum token length of 15), generate answer candidates from both prompts. Then, merge the candidate lists (or use them separately for ensembling).

Notes:

- **In-context examples selection:** Use a similarity measure (e.g. via CLIP embeddings) to pick examples most similar to your current instance.
- **Merging candidates:** You can simply concatenate the lists and remove duplicates or rank them by PLM-provided probabilities.

---

4. Answer Selection

a. Chain-of-Thought (CoT) Rationale

Before selecting the final answer, generate a chain-of-thought rationale to guide the selection process. A fixed prompt (detailed in Appendix A.1 of the paper) is used for this purpose. The CoT rationale is generated by feeding the question (with or without the image context) along with the candidate answers into a PLM set to a higher temperature (e.g. 0.7) and a moderate max token length (e.g. 80).

b. Training the Selection Model

The next step is to train a lightweight answer selection model that takes as input:

- The textual context (caption + tags)
- The original question
- The generated CoT rationale
- The candidate answer list

The input is formatted with sentinel tokens to clearly demarcate different sections. The model's target output is the correct answer (assuming it is within the generated candidate list).

You can experiment with several text-generation models (e.g. KAT, UnifiedQA, or even fine-tuning a GPT-2 variant) to act as the answer selector.

c. Training Details

- **Loss & Optimizer:**
  Use a cross-entropy loss (if the selection model is formulated as classification/generation) and the AdamW optimizer. Learning rates in the paper are around 3e-5 for KAT, 5e-5 for UnifiedQA, and 2e-5 for ClipCap.
- **Hyperparameters:**
  - For PLM answer generation: Temperature = 0.001, max token length = 15
  - For CoT generation: Temperature = 0.7, max token length = 80
  - Number of in-context examples: 16-shot for test; 4-shot for training (when multiple gold answers exist)

- **Hardware:**
  The experiments were conducted on a multi-GPU setup (e.g. 8 NVIDIA V100 GPUs). You can start on a smaller scale if needed.

5. Additional Considerations from the Appendix

- **Chain-of-Thought Prompt Details (Appendix A.1):**
  The fixed prompt used to generate CoT rationales includes several examples (seven are used in the paper, with two shown explicitly). You can replicate this by curating several demonstration examples that explain the reasoning process. Adjust the temperature and token limits accordingly during inference.
- **Ablation Studies and Iterative Prompting:**
  The paper also experiments with iterative prompting (where the PLM is asked to select between its own outputs), but results indicate that using a dedicated selection model with CoT rationale tends to work better.
- **Ensembling PLMs:**
  Although the paper experiments with combining answer choices from multiple PLMs (e.g. GPT-J, UL2, OPT, Codex), note that increasing the number of candidate answers may sometimes hurt the performance of the selection model if it is not robust enough.

---

7. Code and Resources

- **Third-party Implementations:**
  There are also publicly available implementations for parts of the pipeline, such as CLIP-based similarity (using OpenAI's CLIP repository) and image captioning (e.g. using BLIP-2 from the Hugging Face hub).
- **Existing PLM Codebases:**
  For running generation with GPT-J or OPT, consider using the Mesh-Transformer-JAX repository (https://github.com/kingoflolz/mesh-transformer-jax) or Hugging Face's Transformers library.

---

8. Conclusion

In summary, to implement the RASO pipeline:

1. **Preprocess your data** by generating captions and tags for each image.
2. **Generate a diverse set of candidate answers** by prompting a PLM using two complementary prompt designs with in-context examples.
3. **Generate a chain-of-thought rationale** using a fixed prompt to guide the reasoning.
4. **Train or fine-tune a lightweight answer selection model** that, given the context, question, rationale, and candidate answers, outputs the correct answer.
5. **Tune hyperparameters and experiment** with ensembling, iterative prompting, and various PLMs to optimize performance.

This approach has been shown to boost external knowledge coverage significantly and improve end-task performance on challenging VQA benchmarks like OK-VQA (achieving a new state-of-the-art performance).

Feel free to adapt and expand upon this outline using the official code and additional resources available online. Happy coding!