1. Pairwise Preference Generation Script

`build_pairwise_prefs.py`, generates **pairwise preference data** from a dataset of queries, code, and descriptions. These preference pairs `(prompt, chosen, rejected)` are useful for training **reward models** in RLHF (Reinforcement Learning with Human Feedback) pipelines or similar tasks.

Overview

The script performs the following steps:

1. Retrieve relevant context for each query from a **vector database**.
2. Generate multiple candidate responses using a **large language model (LLM)**.
3. Score each candidate using a **grounding-first heuristic** combining context relevance, keyword coverage, and length penalty.
4. Select the best and worst candidate as a pair: `(chosen, rejected)`.
5. Save the generated pairs in a JSONL file for downstream training.

Major Components

1. Configuration

- **DATA_CSV**: Input CSV containing queries, code, and descriptions.
- **VDB_PATH**: Path to the local FAISS vector store.
- **BASE_LLM**: Pretrained model used for candidate generation.
- **USE_4BIT**: Whether to use 4-bit quantization for memory efficiency.

- **MAX_NEW_TOKENS**: Maximum number of tokens generated per candidate.
- **DEVICE**: `"cuda"` if GPU is available, otherwise `"cpu"`.
- **SEEDS**: Seeds for generating diverse candidates.
- **NUM_CANDIDATES**: Number of responses generated per prompt.
- **OUT_JSONL**: File to save the preference pairs.

---

2. Retrieval

- Uses **FAISS** for efficient vector similarity search.
- Embeddings are generated with a **sentence-transformer**.
- Retrieves the top `k` documents (`k=1` in this case, since documents are dense/self-contained).
- These retrieved documents form the **context** for the LLM.

---

3. Prompt Construction

- The function `make_prompt(context, question)` formats the input for the LLM:
  - Includes retrieved context.
  - Adds the user query.
  - Wraps the text with instruction tags.
  - Ensures the LLM answers based on context or says "I don't know" if unsure.

---

4. Model Loading

- Supports **4-bit quantized LLMs** for memory efficiency.
- The Mistral-7B-Instruct model is loaded using `AutoModelForCausalLM`.
- Optional **bfloat16** computations improve numeric stability.
- Tokenizer is loaded and ensures a padding token is set for generation.

---

5. Candidate Generation

- For each prompt, multiple candidates are generated for diversity.
- Uses **sampling** with temperature and top-p (`do_sample=True, temperature=0.7, top_p=0.9`).
- Multiple seeds ensure different outputs for the same prompt.
- The function `extract_answer` extracts the generated text after the instruction tags.
- Optimized batching reduces generation time significantly.

---

6. Scoring Candidates

Each candidate is scored using a **heuristic function**:

1. **Grounding Score**: Cosine similarity between candidate and retrieved context.
2. **Keyword Coverage**: Fraction of task-relevant keywords present in the candidate.
3. **Length Penalty**: Mild penalty for overly long outputs.

- Scoring formula:
  `score = (W_GROUND * grounding) + (W_KEYWD * keyword_coverage) - (W_LEN * length_penalty)`
- Weights: W_GROUND = 0.7, W_KEYWD = 0.3, W_LEN = 0.05.

---

7. Main Loop: Building Preference Pairs

- Iterates through each query in the CSV.
- Retrieves context from FAISS.
- Generates multiple candidates.
- Scores candidates and sorts them.
- Selects **best (`chosen`)** and **worst (`rejected`)** candidate.
- Writes a JSON object per pair in the output JSONL file.

Output JSONL structure:

- `"prompt"`: LLM-formatted prompt
- `"chosen"`: Best candidate according to the heuristic
- `"rejected"`: Worst candidate according to the heuristic

Key Features

- **Retrieval-Augmented Generation (RAG)**: Ground answers in relevant context.
- **Diverse Candidate Generation**: Multiple seeds and sampling strategies.
- **Heuristic Scoring**: Ensures contextually correct and relevant chosen answers.
- **Memory Efficient**: 4-bit quantization and device-aware model loading.
- **Extensible**: Adjust `NUM_CANDIDATES`, seeds, or scoring weights to tune output.

Notes

- Generating candidates for 500 queries takes roughly 3–4 hours on a GPU with 4-bit quantization.
- Tokenizer is ensured to have a padding token to avoid generation errors.
- Output JSONL can directly be used to train reward models for RLHF pipelines.