Dưới đây là bảng so sánh khái quát về **điểm giống** và **khác** giữa hai đoạn code, tập trung vào **mặt khái niệm** và **thuật toán**, diễn giải ngắn gọn, không dùng mã:

---

1. Điểm giống nhau

1. **Mục tiêu chung**
   - Cả hai đều hướng tới việc huấn luyện một mô hình ngôn ngữ (GPT-2) sao cho các câu trả lời tuân thủ các nguyên tắc đạo đức (Constitutional AI) về tính hữu ích, vô hại, trung thực,…
   - Cải thiện chất lượng đầu ra của mô hình bằng cách sử dụng **điểm đánh giá** (scores) dựa trên các nguyên tắc này.
2. **Khung sử dụng**
   - Dùng thư viện **Transformers** của Hugging Face để load và sinh văn bản từ GPT-2.
   - Sử dụng **PyTorch** làm backend cho mọi tính toán (forward, backward, loss).
3. **Cơ chế đánh giá**
   - Định nghĩa bộ **nguyên tắc** (principles/rules) như “Harmless”, “Helpful”, “Honest”… để gán điểm compliance cho mỗi câu trả lời.
   - Tính toán điểm số tổng hợp (overall score) từ các điểm con, rồi dùng làm tín hiệu huấn luyện.
4. **Theo dõi & trực quan hóa**
   - Luôn thu thập và lưu lại các **metrics** (loss, điểm đạo đức, độ đa dạng câu trả lời, v.v.) sau mỗi epoch hoặc sau mỗi đánh giá.
   - Vẽ đồ thị thể hiện tiến trình huấn luyện, so sánh trước–sau trên các nguyên tắc bằng Matplotlib/Seaborn/Plotly.

---

2. Điểm khác nhau

Khía cạnh	Đoạn 1 (Notebook RLAIF)	Đoạn 2 (Script DPO + LoRA)
Phương pháp huấn	Kết hợp **RLAIF** (Reward Learning from AI Feedback): - Học thêm một **reward model** (đầu ra là một số thực)	Dùng **DPO (Direct Preference Optimization)** cùng **LoRA** (ada

Khía cạnh	Đoạn 1 (Notebook RLAIF)	Đoạn 2 (Script DPO + LoRA)
luyện	để dự đoán điểm đạo đức. - Tối ưu hóa tổng loss = LM loss + α · MSE(pred_reward, target_score).	er-based PEFT):

- Sinh cặp câu trả lời với 2 mức temperature khác nhau, so sánh để tạo data preference.
- DPOTrainer tối ưu trực tiếp theo preference mà không cần reward model riêng. |
  | **Cấu trúc code** | - Dạng notebook chia thành nhiều "Cell" rõ ràng: định nghĩa lớp, hàm, train loop, visualize…
- Mã pha trộn giữa comments Tiếng Việt và English. | - Script đơn tập trung: import – class – hàm – main.
- Dùng các `dataclass`, inheritance, và TRL's DPOTrainer API.
- Toàn bộ bằng English, dễ deploy thành file `.py`. |
Xử lý dữ liệu huấn luyện	Tự sinh **training prompts** ngẫu nhiên đa dạng (function `generate_large_diverse_training_data`).	Lấy dataset sẵn có (`alpaca_farm`), kết hợp input–instruction, rồi tạo **preference pairs** trên tập đó.
Tối ưu hoá tham số	Fine-tune toàn bộ tham số GPT-2 + reward model head (đầy đủ, không PEFT).	Chỉ train **LoRA adapters** (phần nhỏ weights), giữ nguyên base model, tiết kiệm bộ nhớ và thời gian.
Đánh giá trước–sau	Dùng `comprehensive_evaluation` so sánh baseline GPT-2 và mô hình đã fine-tune.	Dùng `ModelEvaluator` so sánh base model và adapter-trained model, báo cáo win/loss/tie rate.
API & thư viện phụ trợ	- `torch.optim.AdamW`, custom training loop với `tqdm`.
- Plotly để build dashboard tương tác.	- `peft.LoraConfig`, `trl.DPOTrainer` / `DPOConfig`.
- Tận dụng TRL API để giảm bớt code tự triển khai.
Tính mở rộng & sản xuất	Phù hợp làm demo, thử nghiệm ý tưởng (research-style).	Thiết kế gần với production pipeline: có main entrypoint, load/save model, checkpoint, report JSON.

Tóm lại

- **Giống**: Cùng mục đích nâng cao tính đạo đức và chất lượng câu trả lời, dùng GPT-2, PyTorch/Transformers, đều có bước **generate** → **evaluate** → **train** → visualize.
- **Khác**:

- **Kỹ thuật tối ưu** (RLAIF vs DPO),
- **Phạm vi huấn luyện** (full-model vs LoRA adapter),
- **Cách tạo dữ liệu** (tự sinh prompts vs dùng Alpaca + preference),
- **Cấu trúc** (notebook mô-đun vs script monolithic cho production).

---

1. Thuật toán RLAIF (Reward Learning from AI Feedback)

Dùng một reward model học cách "dự đoán" điểm đạo đức (Constitutional score), rồi kết hợp với loss ngôn ngữ thông thường.

Các bước chính

1. **Sinh phản hồi ban đầu**
   - Cho mỗi prompt P, dùng GPT-2 fine-tune hiện tại sinh ra câu trả lời R_0 với cơ chế sampling (temperature, top-p, v.v.).
2. **Chấm điểm đạo đức (Constitutional Score)**
   - Với mỗi nguyên tắc (Harmless, Helpful, Honest, Respectful), tính điểm từng phần trên R_0.
   - Trung bình các điểm phần để có **điểm tổng hợp** $S \in [0, 1]$.
3. **Huấn luyện reward model**
   - Input: (P, R_0) → reward model (một head tuyến tính nối vào GPT-2) dự đoán $\hat{S}$.

- MSE loss: $\mathrm{L_R} = (\hat{S} - S)^2$.

4. **Tính loss kết hợp**
   - LM loss chuẩn $\mathrm{L_{LM}}$ (negative log-likelihood trên chuỗi $P + R_0$).
   - Tổng loss:

$$\mathrm{L} = \mathrm{L_{LM}} + \alpha \cdot \mathrm{L_R}$$

     với hệ số α (ví dụ 0.2) điều chỉnh tầm quan trọng của reward.

5. **Backward & cập nhật**
   - Tính gradient của L w.r.t. **toàn bộ** tham số GPT-2 và reward head.
   - Cập nhật bằng AdamW.
6. **Lặp lại** cho nhiều epoch, shuffle prompts, theo dõi metrics.

Ví dụ minh hoạ

Giả sử prompt:

> "Hãy giải thích nguyên lý phóng tên lửa."

1. **Sinh** R_0
   - GPT-2 trả về: "Nguyên lý phóng tên lửa dựa trên…"
2. **Chấm điểm**
   - Harmless: 1.0 (không đề cập nội dung nguy hại)
   - Helpful: 0.8 (có chi tiết, nhưng hơi ngắn)
   - Honest: 0.7 (có từ "có vẻ như")
   - Respectful: 0.9
     $\Rightarrow S = (1.0 + 0.8 + 0.7 + 0.9)/4 = 0.85$
3. **Reward model** dự đoán $\hat{S} = 0.65 \Rightarrow$ loss $(0.65 - 0.85)^2$.
4. **LM loss** giả định = 2.1 ⇒ tổng loss ≈ $2.1 + 0.2 \cdot 0.04 = 2.108$.
5. Cập nhật tham số sao cho cả mô hình sinh văn bản và reward head ngày càng khớp điểm đạo đức.

---

2. Thuật toán DPO (Direct Preference Optimization) + LoRA

Dùng preference data (cặp response "chọn"/"không chọn") để trực tiếp tối ưu mô hình, kết hợp adapter LoRA để tiết kiệm bộ nhớ.

Các bước chính

1. **Thu thập preference pairs**
   - Với mỗi prompt P, sinh hai response R_1, R_2 bằng hai temperature khác nhau.
   - Tính score đạo đức S_1, S_2 qua rule engine.
   - Chọn cặp $(P, R_{\text{chosen}}, R_{\text{rejected}})$ sao cho $|S_1 - S_2|$ đủ lớn.
2. **Thiết lập LoRA adapter**
   - Chèn layers LoRA (ranks nhỏ) vào các module attention/c_proj của GPT-2.
   - Khởi tạo adapter weights, đóng băng phần còn lại của mô hình.
3. **Cấu hình DPOTrainer**
   - DPOConfig chứa thông số batch_size, lr, số epoch…
   - Khởi tạo `DPOTrainer(model, ref_model=None, train_dataset, processing_class=tokenizer, peft_config)`.
4. **Objective DPO**
   - Với mỗi cặp preference (P, R_c, R_r), DPO tối ưu sao cho:

$$\log p_\theta(R_c \mid P) - \log p_\theta(R_r \mid P) \quad \text{càng lớn càng tốt.}$$

   - DPO xây dựng loss dạng logistic preference:

$$\mathrm{L}_{\text{DPO}} = -\log \sigma(\Delta\ell - \beta) \quad \text{với } \Delta\ell = \log p_\theta(R_c) - \log p_\theta(R_r).$$

   - β là margin (điều chỉnh mức phân biệt).
5. **Backward & cập nhật adapter**
   - Chỉ cập nhật weights của LoRA adapters, giữ nguyên phần GPT-2 gốc.
6. **Lặp lại** trên toàn bộ tập preference, lưu checkpoint, cuối cùng export adapter.

Ví dụ minh hoạ

Vẫn với prompt:

> "Hãy giải thích nguyên lý phóng tên lửa."

1. Sinh:
   - R_1 (temp 0.6): "Tên lửa hoạt động dựa trên phản lực, nhiên liệu đốt cháy…"
   - R_2 (temp 1.0): "Cơ chế phóng tên lửa là một quá trình phức tạp…"
2. Chấm score:
   - $S_1 = 0.88, S_2 = 0.75 \Rightarrow$ chọn $R_c = R_1, R_r = R_2$.
3. DPO loss khuyến khích $\log p(R_1) - \log p(R_2)$ tăng.
4. Chỉ cập nhật LoRA adapter, giúp nhanh và gọn.

Kết luận ngắn gọn

- **RLAIF**: Huấn luyện song song **toàn bộ** GPT-2 và một reward head, dùng MSE reward loss để hướng dẫn LM loss.
- **DPO+LoRA**: Sinh dữ liệu preference, tối ưu trực tiếp mục tiêu phân biệt "chọn vs không chọn", chỉ tune adapter (LoRA), không cần reward model riêng.

Mỗi thuật toán có **ưu–nhược** riêng:

- RLAIF linh hoạt, trực quan nhưng nặng;
- DPO+LoRA nhanh nhẹ, dễ deploy nhưng phụ thuộc chất lượng preference data.