Phase 2 Documentation: Preference-Based Alignment of Phi-2

Author: Rishav Tewari

Date: 16-01-2026

Abstract This document provides a complete and technically accurate description of Phase 2 of the project. Phase 2 builds directly on the execution guarantees established in Phase 1 and introduces preference-based alignment of a large language model using an offline, script-driven pipeline. The objective of this phase is to align the model so that generated subgoals are semantically correct, executable, and consistent with expert behavior under a strict execution contract. The documentation is intended for internal research review, reproducibility audits, and as supporting material for a future conference submission.

Objectives of Phase 2 Phase 2 is designed to answer the following research question. Can a pretrained language model be aligned using preference signals derived from expert traces so that it reliably produces executable symbolic subgoals under deterministic execution constraints.

The concrete objectives of Phase 2 are as follows:

- Define a reproducible preference dataset over context and subgoal pairs derived from expert MiniGrid trajectories.
- Align the Phi-2 language model using Direct Preference Optimization with LoRA adapters.
- Train a reward model that assigns scalar scores to candidate subgoals.
- Validate alignment using the execution pipeline established in Phase 1.
- Clearly document architectural and hardware constraints observed during implementation.

Codebase structure and primary artifacts Phase 2 is implemented as a structured Python codebase rather than a notebook-based workflow.

The top-level structure is organized as follows:

- Phase2/data/ – expert_traces_train.json – expert_traces_val.json – expert_traces_test.json – dpo_val.jsonl – dpo_test.jsonl
- Phase2/src/data_gen/ – generate_traces.py – minigrid_solver.py – oracle.py – create_preference_pairs.py
- Phase2/src/llm/ – train_dpo.py – train_rlhf.py

- Phase2/src/rm/ – train_rm.py
- Phase2/src/experiments/ – evaluate_pipeline.py

All experiments are driven via Python scripts and configuration parameters. Artifacts such as trained LoRA adapters, reward model weights, and serialized datasets are treated as immutable once generated. Execution environment and reproducibility Phase 2 experiments are executed as script-based runs in a controlled Python environment.

Reproducibility is ensured through the following measures:

- Explicit random seed control for Python, NumPy, and PyTorch.
- Deterministic MiniGrid solvers implemented in minigrid_solver.py.
- Fixed dataset splits for training, validation, and testing.
- Separation of data generation, model training, and evaluation into independent scripts.

Reproducing Phase 2 requires running the scripts in the order documented below using compatible library versions and identical random seeds. No interactive or notebook-specific assumptions are made. System overview Phase 2 extends the Phase 1 execution pipeline with a preference-based alignment layer.

The end-to-end pipeline is as follows:

- Deterministic expert trajectories are generated using scripted MiniGrid solvers.
- Each expert decision is transformed into preference pairs over symbolic subgoals.
- The Phi-2 base model is aligned using Direct Preference Optimization with LoRA adapters.
- A reward model is trained on top of the aligned backbone.
- Generated subgoals are evaluated by executing them in the environment using the Phase 1 executor.

Each stage is validated independently before full pipeline evaluation.

Expert trace generation Expert trajectories are generated using generate_traces.py in conjunction with minigrid_solver.py and oracle.py. These components implement deterministic policies for MiniGrid tasks.

Each trace records:

- Environment identifier and random seed.
- A sequence of environment states.
- The canonical expert subgoal associated with each state.

Traces are serialized to JSON and split into training, validation, and test sets.

Preference data construction Preference datasets are constructed using create_preference_pairs.py. For each expert decision, a canonical subgoal is paired with one or more controlled negative alternatives.

Controlled perturbations include:

- Incorrect object or target selection.
- Invalid ordering of subgoals.
- Omission of required intermediate actions.

Each preference sample follows a strict schema:

- context: textual description of the environment state.
- chosen: expert subgoal.
- rejected: plausible but incorrect subgoal.
- metadata: environment id, seed, and timestep.

Preference data is serialized in JSONL format and split deterministically.

Direct Preference Optimization The Phi-2 language model is aligned using Direct Preference Optimization as implemented in train_dpo.py. The base model parameters are frozen and LoRA adapters are introduced to limit the number of trainable parameters.

Key design decisions include:

- Freezing the base model to preserve pretrained knowledge.
- Training only LoRA adapters to reduce memory and compute requirements.
- Using conservative learning rates and small batch sizes.
- Optimizing a pairwise objective that increases the likelihood of chosen subgoals relative to rejected alternatives.

The output of this stage is a set of LoRA adapter checkpoints encoding preference-aligned behavior. Reward modeling A reward model is trained using train_rm.py.

The DPO-aligned backbone is frozen and a lightweight linear head maps the final hidden state to a scalar reward.

Design rationale:

- Prevent instability by freezing the backbone.
- Minimize additional parameters.
- Produce a reusable scoring function for subgoal evaluation and future RLHF.
- Reward model weights are stored separately from the backbone.

Evaluation End-to-end evaluation is performed using evaluate_pipeline.py. The aligned model generates subgoals for held-out environments, and these subgoals are executed using the Phase 1 executor.

Evaluation metrics include:

- Subgoal validity under the canonical grammar.
- Successful execution of subgoals in the environment.
- Consistency with expert behavior across trajectories.

Comparisons are performed between the base model, the DPO-aligned model, and reward-model-scored outputs.

Failure modes and mitigations Observed failure modes include:

- Dataset leakage between splits.
- Tokenizer or dependency mismatches.
- Reward model overfitting.
- Infeasibility of PPO due to memory requirements.

Mitigations include strict data separation, explicit dependency pinning, validation splits, and conservative model updates.

Limitations Phase 2 excludes PPO-based reinforcement learning. In the TRL framework, PPO requires multiple full copies of the model to reside in GPU memory, which exceeds the capacity of the available hardware. This limitation is documented explicitly and PPO is deferred to future work.

Phase 2 outcome Phase 2 demonstrates that preference-based alignment using DPO and reward modeling can reliably shape a language model to produce executable symbolic subgoals that satisfy a strict execution contract. The resulting artifacts form a stable and extensible foundation for future work on full RLHF, scaling, and continual learning.