AI Physicist White Paper

Author: Vineet Gattani

AI Physicist: Supervised Fine-Tuning + Dual-Stage RL with Tool Calling for Evaluation

This project builds an AI Physicist: a specialized LLM for physics reasoning, trained in three stages. We first domain-adapt the base model to physics via supervised fine-tuning (SFT). Next, we refine its outputs using labeled preferences (hypotheses) with Direct Preference Optimization (DPO). Finally, we train an RL agent (via PPO) to evaluate generated hypotheses using external tools (search, math solvers, code). In combination, this "dual-stage RL" approach (DPO + PPO) aligns the model to human-like physics reasoning. (In fact, DPO "bypasses reward modeling" by tuning directly on preference data).

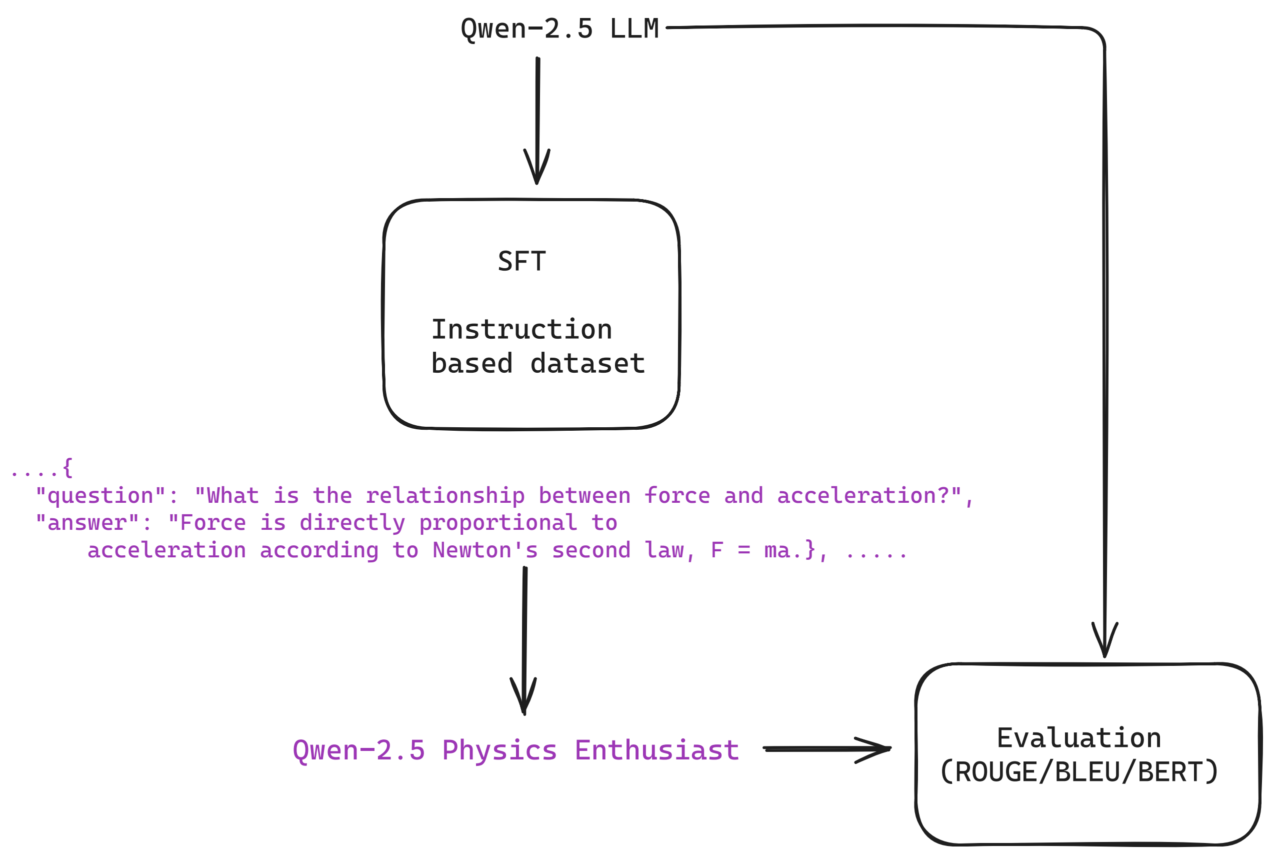
Phase 1: Supervised Fine-Tuning (SFT)

In Phase 1 we **fine-tune** a pre-trained LLM on curated physics data. Supervised fine-tuning uses labeled examples to teach the model desired behavior. Here the dataset (`data/sft_physics_data.json`) contains many instruction–response pairs from physics Q&A (e.g. a Physics StackExchange dump). Each entry looks like:

```
{ "instruction": "Explain the difference between general relativity and special
relativity.",
"output": "General relativity accounts for gravity as curvature of spacetime, whereas
special relativity does not include gravitational effects." }
```

Training on such examples (using `train_sft_physics.py`) adapts the model's weights, transforming it into a *physics-aware* question-answering system. In other words, SFT injects domain knowledge by example, enabling the LLM to learn the characteristic style of physics problem-solving from data.

To quantitatively evaluate the outputs of the fine-tuned model, we can use a combination of surface-level and semantic-level metrics. **BLEU** and **ROUGE** scores are useful for measuring the degree of overlap between the model's generated response and the reference answer in terms of shared n-grams, providing insight into how closely the model matches the expected phrasing. However, these metrics can be overly sensitive to wording variations, especially in scientific explanations where multiple phrasings can be equally valid. To address this, we also compute **BERTScore**, which uses contextual embeddings from a pre-trained language model to assess semantic similarity between the generated and reference responses. This allows us to capture the underlying meaning of an answer even when the exact wording differs, offering a more robust evaluation of scientific correctness and relevance.


Qwen-2.5 LLM
SFT
Instruction based dataset
....{
"question": "What is the relationship between force and acceleration?",
"answer": "Force is directly proportional to acceleration according to Newton's second law, F = ma.}, ...
Qwen-2.5 Physics Enthusiast
Evaluation (ROUGE/BLEU/BERT)

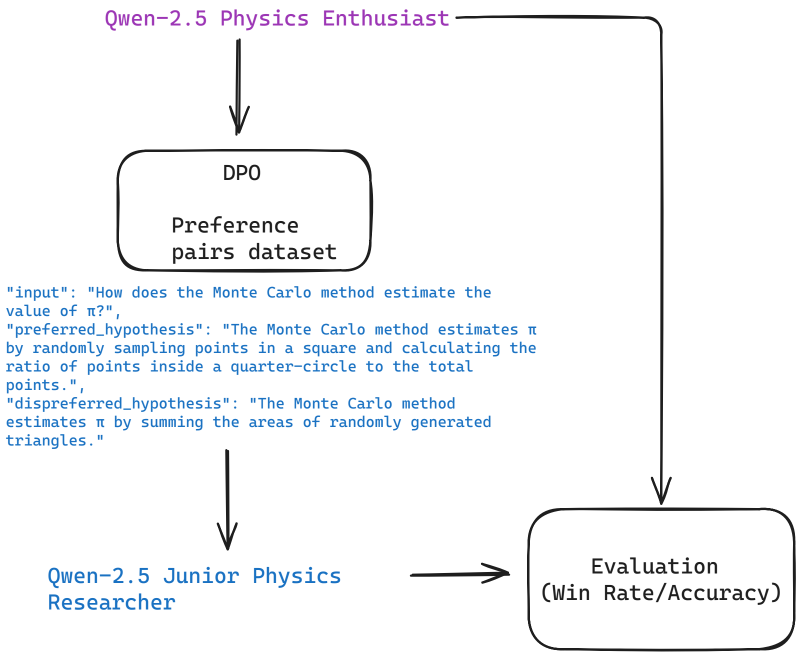
Phase 2: Hypothesis Generation via DPO

After SFT, we generate and refine candidate answers (hypotheses) to physics problems. We prepare a dataset of **preference pairs**: for each physics question, two possible solutions and a label indicating which one is better. These pairs go into `data/stage1_hypothesis_pairs.json`. We then apply **Direct Preference Optimization (DPO)** to train the model on this data (script `train_stage1_hypothesis_dpo.py`).

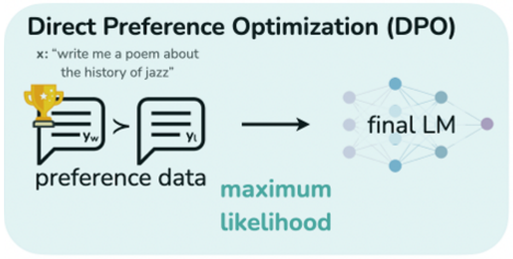
DPO directly optimizes the LLM policy using these binary preferences, *without* training a separate reward model. In effect, the model is updated to give higher likelihood to preferred (correct) solutions than to suboptimal ones. For example, if answer A is judged better than answer B for a question, DPO will adjust the model so that $\log P(A) - \log P(B)$ is maximized. As one blog explains, DPO replaces the usual reinforcement reward model with a **classification-style loss** on preference data (This is known to be simpler and often as effective as RLHF when good preference annotations exist.)

In practice, we run the DPO trainer on `stage1_hypothesis_pairs.json`, which may include additional grounding (e.g. arXiv references) to justify which hypothesis is correct. The result is a model that generates more *plausible physics hypotheses*.

To evaluate **Phase 2: Hypothesis Generation via DPO**, we measure how well the model aligns with human-labeled preferences in unseen hypothesis pairs. A standard metric is **pairwise accuracy**, which quantifies how often the model assigns higher likelihood to the preferred hypothesis in a held-out test set. Additionally, **win-rate evaluations** can be conducted by comparing outputs from the DPO model and a baseline (e.g., SFT-only) in side-by-side human judgments.



```
"input": "How does the Monte Carlo method estimate the
value of π?",
"preferred_hypothesis": "The Monte Carlo method estimates π
by randomly sampling points in a square and calculating the
ratio of points inside a quarter-circle to the total
points.",
"dispreferred_hypothesis": "The Monte Carlo method
estimates π by summing the areas of randomly generated
triangles."
```


Direct Preference Optimization (DPO)
x: "write me a poem about the history of jazz"
preference data
maximum likelihood
final LM

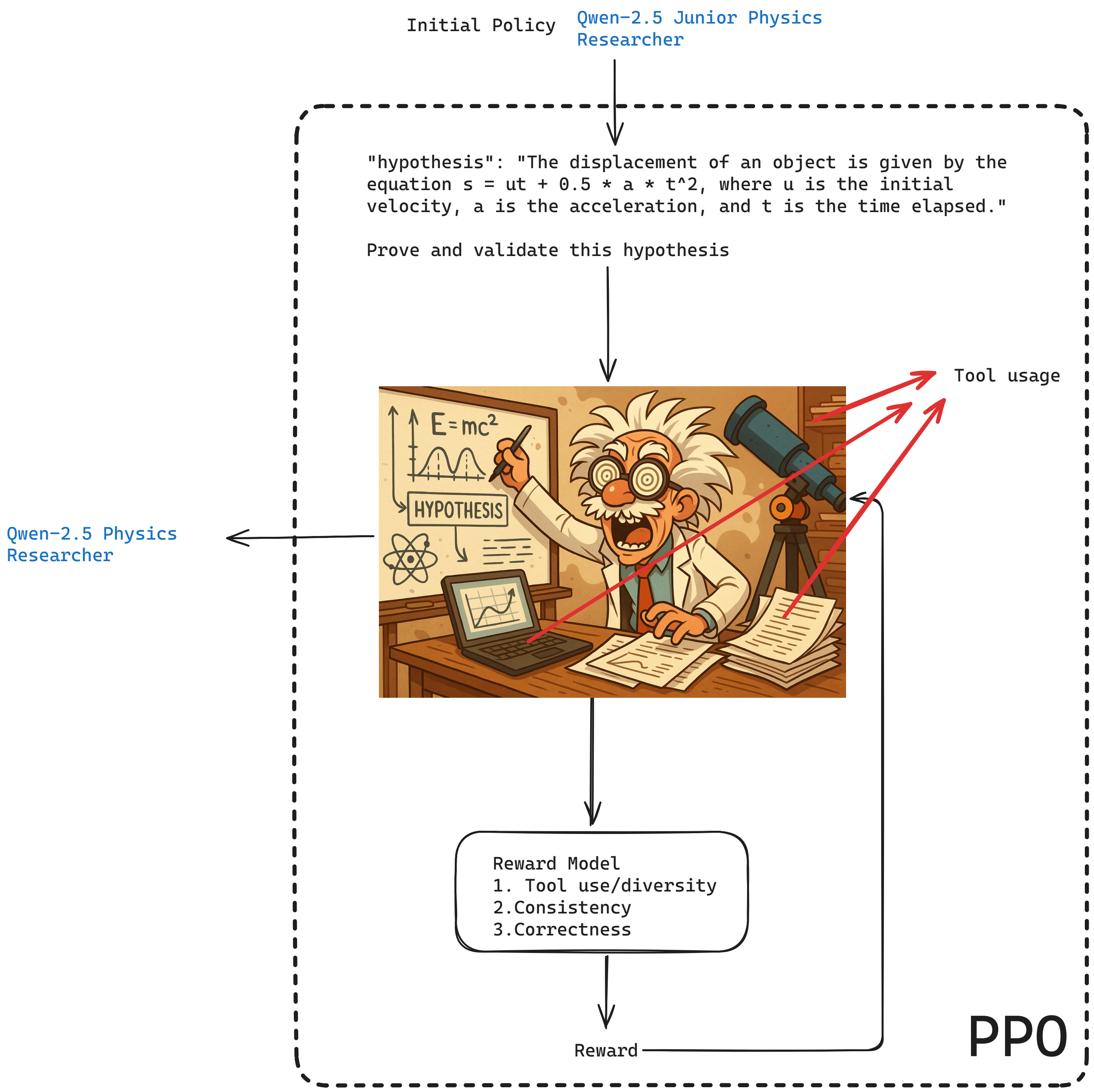
Phase 3: Evaluation Strategy via Tool-Assisted PPO

Phase 3 treats each hypothesis as a decision-making task solved by an **LLM agent** using external tools. Concretely, for a given physics hypothesis, we let the model iteratively plan an evaluation strategy over multiple turns (a "conversation" with itself). At each turn, the agent can invoke tools from `tools/` – e.g. a document retrieval tool, a symbolic math solver, or a code interpreter – and then receive the tool's output as new context. These turn-by-turn actions form a chain-of-thought with tool execution. For example, the model might (1) retrieve relevant formulas from a physics paper, (2) use a calculator to compute a value, and (3) run a short code snippet to check a scenario.

This multi-turn process is modeled as a Markov Decision Process (MDP) where the *state* includes the conversation history (including tool outputs) and the *action* is the next tool call or reasoning step. We train the LLM policy with **Proximal Policy Optimization (PPO)** to maximize a custom reward. Notably, prior work has shown that giving LLM agents access to tools (search, calculators, code, etc.) "can significantly extend their capabilities beyond pure text-based reasoning". Likewise, GRPO/PPO are common choices for training such LLM agents in multi-turn reasoning tasks.

Our reward design encourages effective tool use. We give a bonus of +1.0 if *any* tool is used (to incentivize assistance) and subtract a small penalty (−0.2 each) for redundant use of the same tool. In code form:

```
score = 1.0 if tools_used
else 0.0 score -= 0.2 * (redundancy_count)
```



Initial Policy
Qwen-2.5 Junior Physics Researcher
"hypothesis": "The displacement of an object is given by the equation s = ut + 0.5 * a * t^2, where u is the initial velocity, a is the acceleration, and t is the time elapsed."
Prove and validate this hypothesis
Tool usage
E=mc²
HYPOTHESIS
Qwen-2.5 Physics Researcher
Reward Model
1. Tool use/diversity
2.Consistency
3.Correctness
Reward
PPO

This way the agent is rewarded for leveraging tools but discouraged from unnecessary repeats. The PPO trainer (`train_ppo.py`) uses this reward to adjust the policy. Over many episodes, the model learns to construct evaluation strategies that judiciously call external tools to check or refine hypotheses.

Summary: In Phase 3 we essentially train the fine-tuned LLM to reason *with tools* (ReTool). The multi-turn rollout and PPO optimization lead to an agent that systematically evaluates physics answers using references, math, and code. This completes our pipeline: starting from an SFT-aligned model, we generate hypotheses via DPO, then vet them through a learned tool-assisted evaluation.