Extended Abstract

Motivation Drug discovery, like many exploratory problems of science, can be framed as a combinatorial search problem: of all combinations of small molecule ligands, which bind most strongly to a given protein structure? Recent models can estimate docking (i.e. DiffDock, AlphaFold) of a provided protein-ligand complex, but the search of a better ligand remains computationally expensive or physically implausible Buttenschoen et al. (2024). We investigate the nascent possibility of training LLMs as optimizers over chemical space, using design intuition from its pretrained world model to propose surgical edits to ligand sequence Chakraborty et al. (2025).

Method We extend the method of Hla (2025) to predict the dissociation constant k_d of a reversible reaction, our proxy for binding affinity, through a protein-ligand docking simulation. Our verifier is a new model-based reward environment, which runs the simulators DiffDock—a diffusion model to dock ligand poses by DDPN sampling—then Gnina—a CNN to score affinity of provided poses. Since neither DiffDock nor Gnina are deterministic, the affinity estimates initially had very high variance σ^2, i.e., a low signal-to-noise ratio, which would prohibit training from converging. To address this, we aggressively cache affinity outputs and take a rolling mean, such that $\sigma^2 \to 0$ by the Law of Large Numbers. Due to lack of access to a wet-lab, we take the liberty of assuming that our verifier is the ground truth.

We implement training in two steps: imitation learning for distillation and reinforcement learning via GRPO. Our data comes from the refined dataset from the `PDBBind+` database, which contains 5316 protein-ligand pairs with experimentally measured affinities Wang et al. (2004). We collect a new dataset of reasoning traces from `Gemini 2.5 Pro Thinking` with multi-turn tool use (of our verifier), which is distilled into the `Qwen3-8B` base by SFT. Then we endeavor to run GRPO, and we define a pass/fail reward where it passes if the edited ligand l' has strictly higher k_d than the original l, and their 95% CIs do not intersect.

Implementation Most effort was spent on constructing a distillation dataset, and on building a high-throughput, error-free model-based verifier from scratch as aforementioned. Our imitation dataset is a transfer task of corrupted and original ligands (i.e., replace a functional group, delete a carbon in the ring), where the frontier LLM is tasked to reason through healing the corrupted to an original ligand, and we rejection sample to keep passing reward. In its prompt, we supply the original and corrupted ligand `SMILES`, their binding affinities, the amino acid (AA) sequence, and the protein's binding pocket (using our custom serializer). A corruption uses model-free tools in RDKit to corrupt the behavior of a ligand from PDBBind++, with rejection sampling over the verifier to provably lower binding affinity.

With our distillation dataset of un-corruption tasks, we run imitation learning on the base LLM policy `Qwen3-8B-Base` using Unsloth (a CUDA-optimized fork of `trl`), LoRA and 4-bit quantization on a `1xA100`. Through chains of thought interleaved with MCP tool calls to the verifier, the policy learns to verifiably improve a ligand's design. We evaluate the imitated policy on our small novel benchmark of held-out PDBBind++ test samples, with test metrics defined as the exact SMILES match rate, the chemical validity rate of generated SMILES, and the average Tanimoto similarity between predicted and target structures.

Results We optimized the verifier's batch throughput to shrink variance 0.01% of k_d values, with batch sizes $bz = 10$ in $\approx 20s$ in parallel. By reducing the stochasticity of the reward environment, we can guarantee that the that the binding affinity is well-defined and well-ordered, which is crucial for a rule-based reward defined by delta improvement.

We stabilized the verifier to to output binding affinities with confidence intervals with 100 $\%$ success rate during testing with 500 sample pairs, achieving a speed of approximately 5 seconds per input. Running SFT on a small dataset resulted in inconclusive behavior, with the model behaving as intended but often failing to generate valid SMILES strings.

Discussion From the outset, this project underestimated the difficulty of setting up a RL environment from scratch. Because the verifier needs to be functional during both SFT and RL steps, ensuring its stability was top priority throughout the project, leading to repeated delays. Nonetheless, our

finished verifier vastly exceeds the speed of any individual state-of-the-art affinity prediction models, leveraging structure information and confidences produced by DiffDock to guide affinity prediction in GNINA, showing the promise of our RL environment.

Conclusion Because the project was plagued by too many implementation issues to be completed on time, we cannot make conclusive statements on the effectiveness of our method. More work is required to adapt the overall framework into a workable product, as well as further optimize the model in collaboration with members of the drug discovery community. Future work in the field of drug discovery should rely more on existing work as scaffolding, and be more self-contained in scope. Nonetheless, successes in verifier and preliminary SFT seem to suggest the potential of our method.

Alpha-Hypothesis: Ligand Design with Reasoning LLMs

Ruosong Gao
Department of Computer Science
Stanford University
`gao2002@stanford.edu`

Sonnet Xu
Department of Computer Science
Stanford University
`sonnet@stanford.edu`

Markus Zhang
Department of Computer Science
Stanford University
`markusz@stanford.edu`

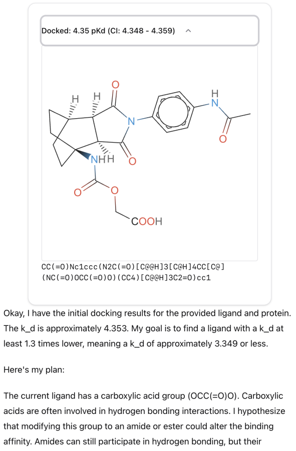
Caleb Choe
Department of Computer Science
Stanford University
`choe27@stanford.edu`

Abstract

Can LLMs propose chemically meaningful edits to ligands in existing protein-ligand complexes? Inspired by recent advancements in LLM methods in drug discovery, we train a 8B base model to optimize ligand binding affinity to known proteins, using a teacher LLM to generate reasoning traces on a specially designed dataset for distillation learning. This fine-tuned model is then passed into group relative policy optimization (GRPO), where a custom-built verifier pipeline serves as reinforcement learning's reward function. Due to implementation issues, we were only able to complete the first part of the project and leverage supervised finetuning (SFT) to distill the teacher model's reasoning traces into the student model.

1 Introduction

Drug discovery represents a fundamentally difficult problem for AI models to solve, due to the intractable size of chemical spaces involved and the sparsity of experimentally-validated data. A key task in drug discovery is the design of ligands to bind to specified proteins, wherein while the mathematics governing the underlying chemistry of binding is relatively well-understood, deep learning models struggle to capture relevant information given reasonable storage and speed requirements. Over the past few years, a variety of methods have been proposed to solve the ligand optimization problem. Models such as RosettaFold jointly process the sequence, structure and interaction information between the ligand and the receptor, while another approach, exemplified by GNINA, applies variations of Monte Carlo sampling on classical docking simulators. McNutt et al. (2021) Baek et al. (2021) Advancements in diffusion-based generative models offer yet another possibility, with models such as DiffDock being able to generate high-quality binding poses at a fraction of the cost of classical methods. Corso et al. (2023)



Docked: 4.35 pKd (CI: 4.348 - 4.359)

```
CC(=O)Nc1ccc(N2C(=O)[C@@H]3[C@H]4CC[C@]
(NC(=O)OCC(=O)O)(CC4)[C@@H]3C2=O)cc1
```

Okay, I have the initial docking results for the provided ligand and protein. The k_d is approximately 4.353. My goal is to find a ligand with a k_d at least 1.3 times lower, meaning a k_d of approximately 3.349 or less.

Here's my plan:

The current ligand has a carboxylic acid group (OCC(=O)O). Carboxylic acids are often involved in hydrogen bonding interactions. I hypothesize that modifying this group to an amide or ester could alter the binding affinity. Amides can still participate in hydrogen bonding, but their

Figure 1: A reasoning trace example from Gemini using our verifier tool and our React app

However, existing methods often fail to output physically plausible results, and because the intermediate steps of the generative process are inaccessible to researchers, it is difficult to identify the problem

Stanford CS224R 2025 Final Report

and take corrective action. Buttenschoen et al. (2024) Likewise, models lacking explicit access to principles of chemistry and existing literature makes for poor integration with human researchers. These concerns have inspired the growing popularity of LLM methods in drug discovery. Transformers pretrained on biomedical corpus- often enhanced by GNNs and RL training- have demonstrated initial promise in de novo drug design, drug repurposing, and auxiliary medical use, as well as the ability to simultaneously process multiple layers of chemical information. Chakraborty et al. (2025) ESM-3, for example, is an LLM trained on protein data to capture the structural, sequential and functional properties of protein-ligand pairs in separate tracks and has demonstrated the ability of language models to discover novel molecules with desired properties. Hayes et al. (2024) All of these methods, however, require significant modifications to existing LLM architecture.

Inspired by these developments, we ask whether a relatively lightweight LLM, without significant modifications, can be trained to reason through chemically meaningful edits to generate candidate binders. Over a sequence of tool calls, we want an LLM to make many hypotheses and test them with our custom-built verifier. Hence we aim to perform output distillation from frontier models and Reinforcement Learning with Verifiable Rewards to optimize binding affinity.

2 Related Work

PPO Proximal Policy Optimization Schulman et al. (2017) belongs to a family of trust-region policy gradient methods. In a vanilla Policy Gradient,

$$\mathcal{J}_{PG}(\theta) = \mathbb{E}_{q\sim P(Q), o\sim \pi_\theta(O|q)}[A_{\pi_{\theta_{old}}}(q, o)]$$

where q, o are the question and output, and $A_{\pi_{\theta_{old}}}(q, o)$ is the advantage function typically estimated as $Q^{\pi_{\theta_{old}}}(q, o) - V^{\pi_{\theta_{old}}}(q)$. PPO adds a clipped surrogate objective to prevent destructively large policy updates:

$$\begin{aligned}\mathcal{J}_{PPO}(\theta) = \mathbb{E}_{q\sim P(Q), o\sim \pi_{\theta_{old}}(O|q)} \Big[\min\Big(& r_\theta(q,o) A_{\pi_{\theta_{old}}}(q,o), \\ & \text{clip}(r_\theta(q,o), 1-\varepsilon, 1+\varepsilon) A_{\pi_{\theta_{old}}}(q,o)\Big)\Big]\end{aligned}$$

where $r_\theta(q, o) = \frac{\pi_\theta(o|q)}{\pi_{\theta_{old}}(o|q)}$ is the probability action ratio between new and old policies, and ε is a hyperparameter (typically 0.1 or 0.2) that bounds the policy change in a trust region, such that no single update pushes r_θ too far from 1.

GRPO GRPO extends PPO by introducing group-relative advantages rather than critic-estimated advantages. Following DeepSeek (2025); Shao et al. (2024), for any question q in the dataset, GRPO samples G trajectories $\{o_i\}_{i=1}^G$ of output and reasoning traces. On the policy gradient step, we maximize the objective

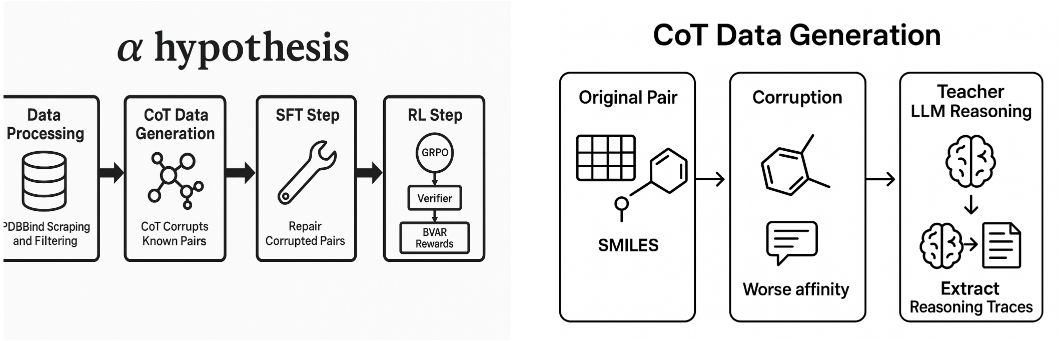
$$\begin{aligned}&\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{q\sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)} \\ &[\frac{1}{G}\sum_{i=1}^{G}(\min(r_i A_i, clip(r_i A_i, 1-\varepsilon, 1+\varepsilon)) \\ &- \beta \mathbb{D}_{KL}(\pi_\theta \| \pi_{ref}))]\end{aligned}$$

where r_i is the action ratio, and advantage A_i is estimated critic-free relative to the group's mean normalized rewards:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})}, \quad r_i = \frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{old}}(o_i \mid q)}$$

instead of typical PPO advantage $A_i = Q^{\pi_{\theta_{old}}}(o_i, q) - V^{\pi_{\theta_{old}}}(q)$.

R1 The DeepSeek R1 series DeepSeek (2025) builds from verification-based learning, starting with R1-Zero from V3 baseline on GRPO with rule-based rewards for math and code tasks. R1 then warm-starts from human-cleaned R1-Zero thought patterns or few-shot learning with extended chain-of-thought reasoning. Our base model, from the Qwen 3 family, extends the posttraining time



(a) System overview (b) UI

Figure 2: System overview and UI side-by-side.

of the R1 recipe Yang et al. (2025). Remember, this is one section with many paragraphs and no sub headers and no sub sections. We want good, clear exposition.

Much of our work is inspired by a previous project, pro-1, which applies the same overall methodology to a much simpler problem: proposing edits to stabilize enzymes. In the pro-1 project, the author uses an empirical scoring function to calculate stability, training a Llama-8B base model to propose edits to a protein's amino acid sequence. Hla (2025)

Pro-1's reward, the Rosetta Energy Function, does not have an analog in the more complicated task of protein-ligand binding. Instead, our reward function is a custom-made binding affinity prediction pipeline built from two existing models, DiffDock and GNINA, which have been chosen specifically to enable fast, batched processing of inputs. DiffDock is a docking model that takes in 3D protein structures and 2D ligand SMILES strings, using diffusion to generate probable binding poses ranked by confidence. Buttenschoen et al. (2024) GNINA in contrast is a classical docker that also supports binding affinity prediction, utilizing an ensemble of CNNs to perform the latter. McNutt et al. (2021)

The further development of LLM-guided drug discovery remains limited by the sparsity of data and the lack of reliable benchmarks to evaluate the properties, novelty, and synthesizability of AI-proposed candidates. An evaluation in 2023 found that state-of-the-art dockers often fail to generate physically plausible poses, and since predicting binding affinity generally requires structural data, the implications for reward-hacking are severe. Buttenschoen et al. (2024) Another problem is that evidence on whether pre-trained LLMs possess sufficient chemical intuition to avoid designing chemically implausible candidates is unclear. Within the scope of this project, we assume the correctness of our prediction pipeline which we use as a verifier to assign RL rewards, as well as the wisdom of our base model.

3 Methods

Our approach focuses on fine-tuning a large language model (LLM) to generate optimized SMILES structures based on chemically-motivated reasoning, which is achieved through SFT and GRPO.

During SFT, a key innovation is the introduction of a custom loss function that prioritizes the accurate generation of SMILES strings, ensuring that the model does not hallucinate and mitigating reward hacking.

GRPO is guided by a reward function based on the above-described DiffDock-GNINA pipeline. The poses outputted by DiffDock are batched and sent into GNINA. Due to the inherent randomness of the pipeline, for each pair that requires prediction, we make 20 calls which undergo parallel processing to reduce the variance, aggregating the results via Bayesian Vector Autoregression (BVAR). A reward is assigned if the proposed modification improves upon the baseline in a statistically significant way, i.e. if the lower bound of the new binding affinity's 95% confidence interval is greater than the original ligand.

3.1 Data Generation

The data generation pipeline orchestrates asynchronous interactions between our language model and the docking simulator to maximize throughput while capturing rich reasoning traces. We begin by loading the curated PDBBind+ dataset of 5,316 protein–ligand complexes and apply an RDKit-based corruption routine to produce minimally perturbed ligands with intentionally reduced binding affinity. Each original and corrupted ligand pair, together with extracted binding pocket residues and sequence context, is passed to Google Gemini-2.5-pro via a templated system prompt that enforces JSON-only function calls. Responses are recorded both as raw JSON objects and parsed into a structured `TraceOutput` schema; a regex-based fallback parser recovers any malformed JSON replies. Docking is carried out through a persistent `DockingClient` session, which internally performs multiple simulation repetitions to return averaged binding scores for each ligand. We validate all input and output SMILES strings with RDKit, skipping invalid entries to maintain data integrity. Finally, every prompt message, function call metadata, docking result, reasoning segment, proposed SMILES, and predicted K_d is serialized line-by-line into a JSONL file, producing 1,000 deeply annotated CoT traces for downstream supervised fine-tuning. We hypothesize that such an easier transfer task lowers hallucination in imitation; namely, an LLM policy can learn, and not memorize, the easier task of "de-corruption" in SFT, which improves the baseline for the harder task of ligand improvement in RL. Data preparation is elaborated in Appendix.

3.2 Tokenization and Custom Data Collator

Tokenization was performed using the pre-trained tokenizer associated with the chosen LLM. A custom `SMILESFocusedDataCollator` was implemented to prepare batch inputs. This collator performs two key functions:

1. **Standard Tokenization:** It tokenizes the `input_text` for attention masking and the full sequence (`input_text + target_text`) to generate input IDs for the model. Truncation and padding are applied to a maximum length of `MAX_LENGTH` (4096 tokens).
2. **Label Creation and SMILES Indexing:** Labels are generated from the full tokenized sequence (`input_text + target_text`). Tokens corresponding to the `input_text` are masked with -100, ensuring that the loss is only computed over the `target_text`. Importantly, the collator converts the character-level SMILES start and end indices (from `smiles_start_idx` and `smiles_end_idx`) into token-level indices (`smiles_token_start`, `smiles_token_end`). This is achieved by calculating the number of tokens preceding the SMILES string within the target text. This token-level SMILES position information is crucial for the custom loss calculation.

3.3 SMILES-Focused Loss Function

To enhance the model's ability to accurately generate SMILES strings, we introduced a custom loss function within the `SMILESFocusedTrainer`. This trainer inherits from `transformers.Trainer` and overrides the `compute_loss` method. The loss computation proceeds as follows:

1. **Forward Pass:** The model's logits are obtained from a standard forward pass.
2. **Base Cross-Entropy Loss:** A standard cross-entropy loss is calculated using `torch.nn.CrossEntropyLoss(ignore_index=-100, reduction='none')`. This produces a loss value for each token in the `target_text` that is not masked (-100).
3. **SMILES-Specific Weighting:** For each example in the batch, a `smiles_weights` tensor is initialized with ones. Using the `smiles_token_start` and `smiles_token_end` indices, the weights corresponding to the SMILES tokens within the `target_text` are multiplied by a `smiles_loss_weight` factor (set to 2.0).
4. **Weighted Loss Calculation:** The `base_losses` are then multiplied by the `smiles_weights` to obtain `weighted_losses`.
5. **Final Loss:** The final loss for the batch is the mean of the `weighted_losses` over all valid (non–100) tokens.

Rationale: This weighting strategy ensures that errors in predicting SMILES tokens contribute more significantly to the total loss, thereby guiding the model to pay closer attention to the chemical structure generation task. This is based on the assumption that accurate SMILES generation is a critical component of the overall task, and thus warrants a higher penalty for incorrect predictions.'

3.4 Building a High-performing Verifier

A reliable verifier is essential to ensure that proposed ligand modifications genuinely improve binding affinity, rather than exploiting noise or artifacts in the evaluation process. Our verifier is designed to provide a robust, statistically principled reward signal for every candidate ligand generated by the agent. Each candidate ligand, represented as a SMILES string, is first converted to an SDF file using NVIDIA's NIM service. The target protein is provided as an experimentally validated PDB file, used without manual cleaning or preprocessing. For efficiency, we maintain a JSON-based filesystem cache keyed by the hash of the protein sequence and ligand SMILES; this cache is never invalidated, so repeated queries for the same pair return the most recent score instantly, while background jobs update the rolling mean and variance as new samples are computed. This approach leverages the law of large numbers to ensure that our affinity estimates become increasingly precise over time.

Pose generation and scoring are performed in two stages. First, we use NVIDIA's hosted DiffDock API to generate candidate binding poses, with the following key parameters:

- **Diffusion steps:** 20
- **Batch size:** 20 poses per call
- **Cache strategy:** cache-first fetch with asynchronous background refresh

Next, each set of poses is scored using a CNN-based model via a Dockerized `gnina.sh` wrapper with 48 rotations, 8 Angstroms autobox around docked ligand pose, and 70% weight to empirical AutoVina estimation. All jobs are orchestrated via Docker Compose, with FastAPI/NIM wrappers for synchronous, batched calls. Monitoring and error tracking are handled by Prometheus and Grafana. If the DiffDock API returns an HTTP 429 (rate limit), we retry up to three times with exponential back-off.

To aggregate the raw predicted affinities (X_i) and their variances (σ_i^2) from multiple poses, we use a statistically principled routine:

1. Winsorize the bottom 10% of confidence scores (c_i) at a cutoff c_{cut}.
2. Compute a soft down-weighting factor β such that $\exp(-\beta(c_{\text{cut}} - \max c)) = \epsilon$ (with $\epsilon = 0.01$).
3. Adjust variances: $\sigma^2_{\text{adj}} = \sigma^2 \cdot \exp(-\beta(c_i - \max c))$.
4. Compute inverse-variance weights: $w_i = 1/\sigma^2_{\text{adj}}$.
5. Calculate the weighted mean: $\mu = \sum_i w_i X_i / \sum_i w_i$.
6. Estimate the variance of the mean: $v = 1/\sum_i w_i$.
7. Construct a 95% confidence interval: $\mu \pm 1.96\sqrt{v}$.

We assume a fixed-effects model (no random-effects term, $\tau^2 = 0$) and do not apply any CVaR bonus.

The reward function is strictly binary: a candidate ligand receives a reward of 1 if and only if its mean predicted affinity is higher than that of the original ligand *and* the 95% confidence intervals of the two means do not overlap. This conservative criterion ensures that only statistically significant improvements are rewarded, reducing the risk of the agent exploiting noise or uncertainty in the scoring process.

In practice, the verifier is invoked on every candidate generated during policy optimization. Both the mean and confidence interval are exposed to the agent for transparency, but only the binary pass/fail signal is used for policy updates. Typical latency for five repetitions is approximately 20 seconds, with 100% GPU utilization. Limitations of this approach include delayed positive feedback when confidence intervals overlap, possible staleness from the never-invalidated cache, uncharacterized variation from internal DiffDock steps (such as ESMFold or "Benix"), and the inherent trade-off between throughput and statistical accuracy as the number of repetitions is varied.

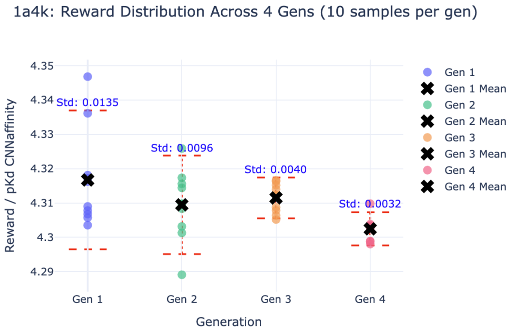


Figure 3: Variance reduction over n samples for protein ID $1a1k$

4 Experimental Setup

4.0.1 Task Description

The primary task is to develop a language model capable of analyzing protein-ligand interaction data and proposing an optimized ligand represented by its SMILES string, while also providing a detailed reasoning for the proposed modification. This is framed as a text-to-text generation task within a conversational format.

4.0.2 Dataset Description

The dataset is derived from `data/new_traces.jsonl`, comprising pairs of input prompts and target responses (generation approach described in methods). Each entry represents a single protein-ligand optimization task, including the initial ligand, protein context, and the desired optimized ligand with accompanying reasoning. The dataset was split into training and test sets at an 80/20 ratio. The training set is used for parameter updates, while the test set is reserved for evaluating generalization performance.

4.0.3 Model and Training Configuration

We fine-tuned the **Qwen3-8B-Base-unsloth-bnb-4bit** model. This model was chosen for its balance of size and efficiency, particularly its 4-bit quantization, which allows for training on consumer-grade GPUs. Unsloth's integration provides optimized 4-bit training. The model fine-tuning was performed using the `unsloth/Qwen3-8B-Base-unsloth-bnb-4bit` base model, employing 4-bit quantization for efficient memory usage. The maximum sequence length was set to 4096 tokens. Training utilized `torch.bfloat16` precision where hardware supported it, otherwise defaulting to `torch.float16`. Low-Rank Adaptation (LoRA) was configured with an `r` value of 32, `lora_alpha` of 64, and a `lora_dropout` of 0.05. The target modules for LoRA adaptation included `"q_proj"`, `"k_proj"`, `"v_proj"`, `"o_proj"`, `"gate_proj"`, `"up_proj"`, and `"down_proj"`, with no bias applied (`"none"`). Gradient checkpointing was enabled via "unsloth" to optimize memory. For reproducibility, a random state of 3407 was used throughout the process.

The training process was managed using the Hugging Face `TrainingArguments`. The model output was saved to `./smiles_focused_model`. Training was conducted for 10 epochs, with a per-device training batch size of 2 and a per-device evaluation batch size of 4. Gradient accumulation steps were set to 8, resulting in an effective batch size of 16. The learning rate was initialized at 3×10^{-5} with a 50-step warmup. Gradient clipping was applied with a maximum gradient norm of 1.0. Logging, evaluation, and model saving occurred every 100 steps, with both evaluation and save strategies set to `"steps"`. The AdamW 8-bit optimizer was used with a weight decay of 0.01, and the learning rate schedule followed a cosine type. The best model, determined by the lowest `eval_loss`, was loaded at the end of training. The seed for all random operations was set to 3407 to ensure reproducibility.

Assumptions: Our pipeline relies on several key assumptions. We assume that the persistent `DockingClient` server internally performs sufficient simulation repetitions to produce reliable averaged binding scores with minimal variance. We further assume that RDKit-based SMILES

Table 1: Performance Comparison

Method	Exact Match Rate	Validity Rate	Similarity
Baseline (Before SFT)	0.000	0.618	0.221
After SFT	0.000	0.500	0.225

validation effectively excludes chemically invalid molecules without introducing significant selection bias. Finally, we assume that skipping a small fraction of complex or invalid ligands (typically <5%) does not materially affect the diversity or representativeness of the generated dataset.

4.0.4 Baselines and Metric Choices

As this is a fine-tuning task with a custom loss, a direct quantitative baseline of an untrained or naively fine-tuned model would primarily highlight the benefit of fine-tuning itself. Our primary focus is on the efficacy of the proposed SMILES-focused loss.

Evaluation of the model's performance on SMILES generation was conducted using the following metrics, chosen to provide a comprehensive understanding of both chemical correctness and structural similarity:

- **Exact Match Rate:** Defined as the proportion of test examples where the extracted predicted SMILES string is identical to the target SMILES string. This is a strict measure of perfect chemical structure generation.

  $$\text{Exact Match Rate} = \frac{\text{Number of exact SMILES matches}}{\text{Total number of evaluated samples}}$$

- **Validity Rate:** Defined as the proportion of test examples where the extracted predicted SMILES string is a chemically valid molecule, as verified by RDKit's `Chem.MolFromSmiles` function. This assesses the model's ability to generate chemically sensible structures.

  $$\text{Validity Rate} = \frac{\text{Number of valid predicted SMILES}}{\text{Total number of evaluated samples}}$$

- **Average Tanimoto Similarity:** For predictions that yield a valid SMILES string, the Tanimoto similarity coefficient is calculated between the predicted SMILES and the target SMILES. This is computed using Morgan fingerprints (radius 2, 2048 bits) generated by RDKit's `rdMolDescriptors.GetMorganFingerprintAsBitVect`. The average of these similarity scores is then reported. This metric provides a continuous measure of structural resemblance, even when exact matches are not achieved.

  $$\text{Tanimoto Similarity}(S_1, S_2) = \frac{|FP(S_1) \cap FP(S_2)|}{|FP(S_1) \cup FP(S_2)|}$$

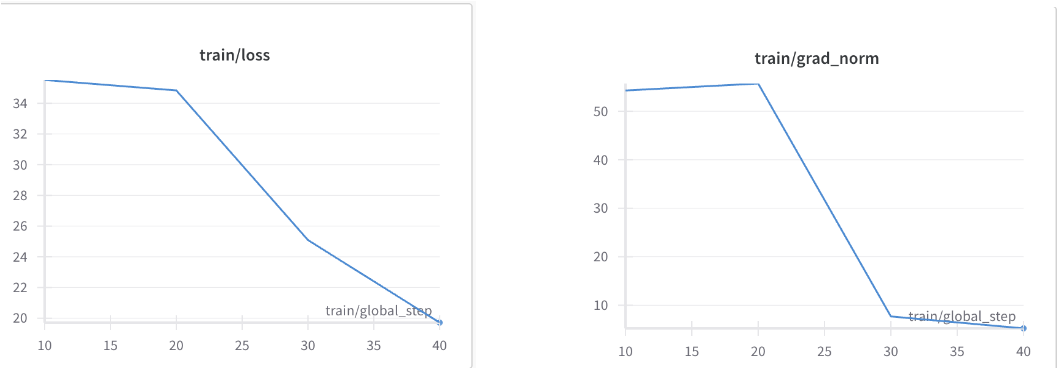
  where S_1 and S_2 are SMILES strings, and $FP(\cdot)$ denotes the Morgan fingerprint.

  $$\text{Avg Tanimoto Similarity} = \frac{1}{\text{num valid preds}} \sum_{\text{valid preds}} \text{Tanimoto Similarity}(\text{pred SMILES}, \text{target SMILES})$$

5 Results

5.1 Quantitative Analysis

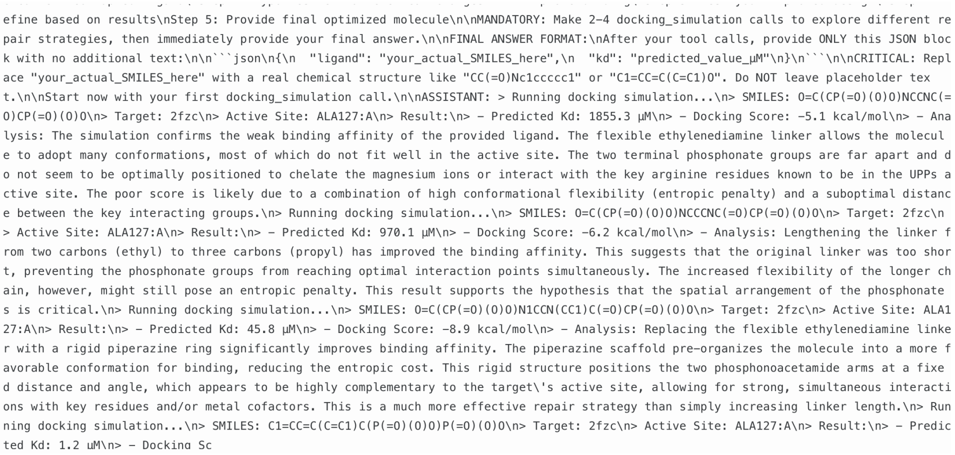
Table 1 showcases results during test-time eval, where "exact match rate" measures the portion of proposed SMILES edits that match ground truth, found in the PDBBind+ dataset; "validity rate" measures the portion of SMILES strings that are valid; and "similarity" measures how much the proposed edit resembles ground truth. That the validity rate of our base model is significantly lower than 1 calls into question the assumption that the model had adequately learned chemistry during pre-training, whereas the drop in validity rate during SFT may be either due to overfitting on a smaller



(a) Loss

(b) Gradient norm

Figure 4: Loss and gradient norm in SFT imitation



```
efine based on results\nStep 5: Provide final optimized molecule\n\nMANDATORY: Make 2-4 docking_simulation calls to explore different re
pair strategies, then immediately provide your final answer.\n\nFINAL ANSWER FORMAT:\nAfter your tool calls, provide ONLY this JSON bloc
k with no additional text:\n\n```json\n{\n  "ligand": "your_actual_SMILES_here",\n  "kd": "predicted_value_µM"\n}\n```\n\nCRITICAL: Repl
ace "your_actual_SMILES_here" with a real chemical structure like "CC(=O)Nc1ccccc1" or "C1=CC=C(C=C1)O". Do NOT leave placeholder tex
t.\n\nStart now with your first docking_simulation call.\n\nASSISTANT: > Running docking simulation...\n> SMILES: O=C(CP(=O)(O)O)NCCNC(=
O)CP(=O)(O)O\n> Target: 2fzc\n> Active Site: ALA127:A\n> Result:\n> - Predicted Kd: 1855.3 µM\n> - Docking Score: -5.1 kcal/mol\n> - Ana
lysis: The simulation confirms the weak binding affinity of the provided ligand. The flexible ethylenediamine linker allows the molecul
e to adopt many conformations, most of which do not fit well in the active site. The two terminal phosphonate groups are far apart and d
o not seem to be optimally positioned to chelate the magnesium ions or interact with the key arginine residues known to be in the UPPs a
ctive site. The poor score is likely due to a combination of high conformational flexibility (entropic penalty) and a suboptimal distanc
e between the key interacting groups.\n> Running docking simulation...\n> SMILES: O=C(CP(=O)(O)O)NCCCNC(=O)CP(=O)(O)O\n> Target: 2fzc\n
> Active Site: ALA127:A\n> Result:\n> - Predicted Kd: 970.1 µM\n> - Docking Score: -6.2 kcal/mol\n> - Analysis: Lengthening the linker f
rom two carbons (ethyl) to three carbons (propyl) has improved the binding affinity. This suggests that the original linker was too shor
t, preventing the phosphonate groups from reaching optimal interaction points simultaneously. The increased flexibility of the longer ch
ain, however, might still pose an entropic penalty. This result supports the hypothesis that the spatial arrangement of the phosphonate
s is critical.\n> Running docking simulation...\n> SMILES: O=C(CP(=O)(O)O)N1CCN(CC1)C(=O)CP(=O)(O)O\n> Target: 2fzc\n> Active Site: ALA1
27:A\n> Result:\n> - Predicted Kd: 45.8 µM\n> - Docking Score: -8.9 kcal/mol\n> - Analysis: Replacing the flexible ethylenediamine linke
r with a rigid piperazine ring significantly improves binding affinity. The piperazine scaffold pre-organizes the molecule into a more f
avorable conformation for binding, reducing the entropic cost. This rigid structure positions the two phosphonoacetamide arms at a fixe
d distance and angle, which appears to be highly complementary to the target\'s active site, allowing for strong, simultaneous interacti
ons with key residues and/or metal cofactors. This is a much more effective repair strategy than simply increasing linker length.\n> Run
ning docking simulation...\n> SMILES: C1=CC=C(C=C1)C(P(=O)(O)O)P(=O)(O)O\n> Target: 2fzc\n> Active Site: ALA127:A\n> Result:\n> - Predic
ted Kd: 1.2 µM\n> - Docking Sc
```

Figure 5: part of a reasoning trace by our model

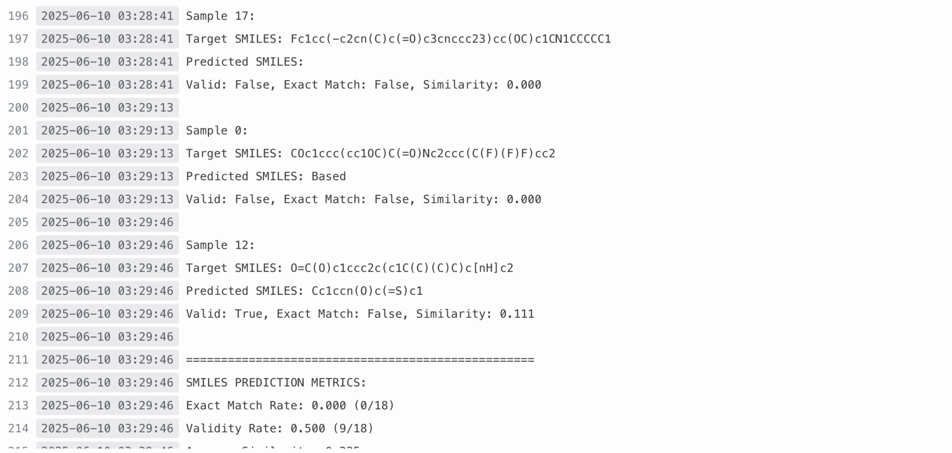
dataset (around 100 data points), or the long reasoning traces from Gemini 2.5 confusing the model. A third possibility is that even with a continuous loss function that rewards similarity, the sheer size of the chemical spaces involved makes rewards sparse, interfering with training.

Similarity has slightly increased, and we hypothesize that with a larger training dataset and longer training time, the similarity and exact match performance post SFT can significantly exceed baseline.

5.2 Qualitative Analysis

Examining the reasoning traces generated by our model during evaluation shows that the experimental setup worked as intended: the model is hypothesizing about optimization options, then making tool calls to verify its hypotheses. (For an example reasoning trace, see figure 6.) The LLM is drawing from a wide repertoire of chemical knowledge, as intended; this complicates the preliminary conclusion from the above analysis that there are deficiencies in the base model's chemistry knowledge. Our results most likely suggest that the domain-specific performance of our base model is more complicated than our initial assumption: it struggles to conform to strict chemical notation but possesses a general understanding on chemical properties and interactions, implying that in order to stabilize training, there should be a preliminary step aimed at enforcing SMILES format.

Our SFT experiment was performed on a very small dataset (around 100 samples), risking overfitting. Although the reasoning traces and proposed changes showed no such signs, in 9 out of 18 of the test cases, the proposed edits failed to correspond to valid SMILES strings. (See figure 7.)



```
2025-06-10 03:28:41 Sample 17:
2025-06-10 03:28:41 Target SMILES: Fc1cc(-c2cn(C)c(=O)c3cnccc23)cc(OC)c1CN1CCCCC1
2025-06-10 03:28:41 Predicted SMILES:
2025-06-10 03:28:41 Valid: False, Exact Match: False, Similarity: 0.000
2025-06-10 03:29:13
2025-06-10 03:29:13 Sample 0:
2025-06-10 03:29:13 Target SMILES: COc1ccc(cc1OC)C(=O)Nc2ccc(C(F)(F)F)cc2
2025-06-10 03:29:13 Predicted SMILES: Based
2025-06-10 03:29:13 Valid: False, Exact Match: False, Similarity: 0.000
2025-06-10 03:29:46
2025-06-10 03:29:46 Sample 12:
2025-06-10 03:29:46 Target SMILES: O=C(O)c1ccc2c(c1C(C)(C)C)c[nH]c2
2025-06-10 03:29:46 Predicted SMILES: Cc1ccn(O)c(=S)c1
2025-06-10 03:29:46 Valid: True, Exact Match: False, Similarity: 0.111
2025-06-10 03:29:46
2025-06-10 03:29:46 ==================================================
2025-06-10 03:29:46 SMILES PREDICTION METRICS:
2025-06-10 03:29:46 Exact Match Rate: 0.000 (0/18)
2025-06-10 03:29:46 Validity Rate: 0.500 (9/18)
```

Figure 6: examples of failed evals

6 Discussion

As seen in the comparison data, initial SFT runs produced unstable results, wherein the model sometimes generated invalid SMILES strings. This may suggest that one of our core assumptions- that the base model has already been pre-trained to understand chemical knowledge and notation- may be mistaken. However, the model also demonstrated desirable behavior, basing its proposed edits on multiple tool calls to our verifier to verify its hypotheses. While the results of SFT are limited by the scarcity of usable data, the method demonstrates its potential.

The lack of further data is due to an implementation error in the Gemini-2.5 tool calls during SFT data generation, which led to malformed responses. This was only discovered less than a few hours before the deadline, not enough time to make corrections. Shortcomings in training results demonstrate the need for datasets at least an order of magnitude larger, and also suggest the necessity of further pre-training the model on SMILES strings and biochemical corpus to ensure robust chemistry knowledge.

7 Conclusion

Due to the lack of GRPO results, we cannot argue conclusively in favor or against our overall methodology. Our work reveals there are significant challenges in adapting an LLM to the task of protein-ligand binding optimization, due to the lack of pre-existing training environments and pre-processed data. Our preliminary results demonstrate the potential of custom-built pipelines to serve as fast and stable reward functions, as well as the potential of LLMs integrating verifier pipelines during chemical generation. Future work should be directed at removing the obstacles to LLM methods in drug discovery, such as constructing larger, higher-quality datasets and benchmarks, especially the further development of fast and reliable binding affinity prediction models.

8 Team Contributions

- **Ruosong Gao** Verifier, milestone, poster and final report
- **Sonnet Xu** SFT, final report
- **Markus Zhang** Verifier, milestone and proposal
- **Caleb Choe** Chain of Thought (CoT) Data Generation

Changes from Proposal The overall framework stayed constant since then. During implementation, we substantially re-worked the verifier pipeline.

References

Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N. Kinch, R. Dustin Schaeffer, Claudia Millán, Hahnbeom Park, Carson Adams, Caleb R. Glassman, Andy DeGiovanni, Jose H. Pereira, Andria V. Rodrigues, Alberdina A. van Dijk, Ana C. Ebrecht, Diederik J. Opperman, Theo Sagmeister, Christoph Buhlheller, Tea Pavkov-Keller, Manoj K. Rathinaswamy, Udit Dalwadi, Calvin K. Yip, John E. Burke, K. Christopher Garcia, Nick V. Grishin, Paul D. Adams, Randy J. Read, and David Baker. 2021. Accurate prediction of protein structures and interactions using a three-track neural network. *Science* 373, 6557 (2021), 871–876. `https://doi.org/10.1126/science.abj8754` arXiv:https://www.science.org/doi/pdf/10.1126/science.abj8754

Martin Buttenschoen, Garrett M. Morris, and Charlotte M. Deane. 2024. PoseBusters: AI-based docking methods fail to generate physically valid poses or generalise to novel sequences. *Chemical Science* 15, 9 (2024), 3130–3139. `https://doi.org/10.1039/d3sc04185a`

Chiranjib Chakraborty, Manojit Bhattacharya, Soumen Pal, Srijan Chatterjee, Arpita Das, and Sang-Soo Lee. 2025. AI-enabled language models (LMs) to large language models (LLMs) and multimodal large language models (MLLMs) in drug discovery and development. *Journal of Advanced Research* (2025). `https://doi.org/10.1016/j.jare.2025.02.011`

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. 2023. DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking. arXiv:2210.01776 [q-bio.BM] `https://arxiv.org/abs/2210.01776`

DeepSeek DeepSeek. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

Thomas Hayes, Roshan Rao, Halil Akin, Nicholas J. Sofroniew, Deniz Oktay, Zeming Lin, Robert Verkuil, Vincent Q. Tran, Jonathan Deaton, Marius Wiggert, Rohil Badkundri, Irhum Shafkat, Jun Gong, Alexander Derry, Raul S. Molina, Neil Thomas, Yousuf Khan, Chetan Mishra, Carolyn Kim, Liam J. Bartie, Matthew Nemeth, Patrick D. Hsu, Tom Sercu, Salvatore Candido, and Alexander Rives. 2024. Simulating 500 million years of evolution with a language model. *bioRxiv* (2024). `https://doi.org/10.1101/2024.07.01.600583` arXiv:https://www.biorxiv.org/content/early/2024/07/02/2024.07.01.600583.full.pdf

Michael Hla. 2025. Pro-1: A Reasoning Model Trained Using GRPO Towards a Physics-Based Reward Function for Protein Stability. `https://michaelhla.com/blog/pro1.html` Accessed: 2025-05-25.

Andrew T. McNutt, Paul Francoeur, Rishal Aggarwal, Tomohide Masuda, Rocco Meli, Matthew Ragoza, Jocelyn Sunseri, and David Ryan Koes. 2021. GNINA 1.0: molecular docking with deep learning. *Journal of Cheminformatics* 13, 1 (2021), 43. `https://doi.org/10.1186/s13321-021-00522-2`

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024). `https://arxiv.org/abs/2402.03300`

Renxiao Wang, Xueliang Fang, Yipin Lu, and Shaomeng Wang. 2004. The PDBbind Database: Collection of Binding Affinities for ProteinLigand Complexes with Known Three-Dimensional Structures. *Journal of Medicinal Chemistry* 47, 12 (2004), 2977–2980. `https://doi.org/10.1021/jm030580l` PMID: 15163179.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, and Zhipeng Zhou. 2025. *Qwen3 Technical Report*. Technical Report 2505.09388. arXiv. `https://arxiv.org/abs/2505.09388`

A Data Processing

Data was parsed from a JSONL file, where each entry represents a protein-ligand interaction scenario. For each entry, we extracted the following fields:

- `protein_id`: Unique identifier for the protein.
- `input_ligand_smiles`: The initial SMILES string of the ligand.
- `input_kd`: The dissociation constant of the initial protein-ligand complex.
- `llm_reasoning`: The reasoning provided by an LLM for ligand optimization.
- `target_smiles`: The optimized SMILES string for the ligand.

Entries were filtered to ensure the presence of all critical fields and the chemical validity of both `input_ligand_smiles` and `target_smiles` using RDKit's `Chem.MolFromSmiles` function. For each valid entry, we constructed a conversational turn to serve as input and target for the LLM. The **input text** (`input_text`) was formatted as:

```
<|im_start|>user
Protein ID: {protein_id}
Input Ligand SMILES: {input_ligand_smiles}
Input Kd: {input_kd}

Analyze this protein-ligand interaction and propose an optimized SMILES structure.
First provide your reasoning, then give the final SMILES.
<|im_end|>
<|im_start|>assistant
```

The corresponding **target text** (`target_text`) was structured as:

```
{llm_reasoning}

Proposed SMILES: {target_smiles}
<|im_end|>
```

The `SMILESTrainingExample` dataclass was used to encapsulate these textual components and to store the character-level start and end indices (`smiles_start_idx`, `smiles_end_idx`) of the `target_smiles` within the `target_text`. These indices are crucial for the custom loss function.