SMDD-BENCH: CAN LLMS SOLVE REAL-WORLD SMALL MOLECULE DRUG DESIGN TASKS?

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents have incredible potential for scientific discovery applications. However, the performance of LLM agents on real-world, small molecule drug design (SMDD) tasks across diverse chemistries and targets is unclear. Current evaluation methods are either ad hoc, too simple for real-world discovery, limited in scale, or restricted to single-turn question answering. In effort to standardize the evaluation of LLM agents on small molecule design, we introduce SMDD-Bench, a challenging, multi-turn, long-horizon agentic benchmark consisting of 502 guaranteed-solvable task instances spanning 5 task types: 2D Pharmacophore Identification, Interaction Point Discovery, Scaffold Hopping, Lead Optimization, and Fragment Assembly. SMDD-Bench tasks span a wide region of chemical space and involve 102 unique protein targets. Completely solving the benchmark would require having strong chemical and biological reasoning and 3D intuition, understanding specialized tool use, and displaying planning expertise over a limited number of oracle calls. We benchmark 7 frontier open and closed source LLMs and find even the most performant LLM, GPT5.4, solves only 40.2% of tasks. We hope SMDD-Bench provides a standardized testbed to invigorate the field towards training and evaluating LLM agents for fully autonomous computational drug design.

1 INTRODUCTION

LLMs have seen considerable success in many scientific discovery tasks such as synthesizing scientific literature, orchestrating chemistry experiments, and even performing end to end applied machine learning research (Boiko et al., 2023; Lu et al., 2026; Asai et al., 2026). Within the drug discovery and medicinal chemistry space, LLMs have been benchmarked on tasks including single step retrosynthesis prediction, molecular description, de novo molecular generation, and property prediction (Yu et al., 2024; Mirza et al., 2025; Huang et al., 2021; Bartmann et al., 2026). However, these benchmarks are either restricted to single-turn question answering, are rudimentary and unrealistic when compared to the complexity of real-world computational drug discovery, or are limited to only a small non-diverse handful of tasks and protein targets. Although such benchmarks are useful in understanding an LLM's ability to understand basic chemistry principles, they are not as useful for understanding and improving an LLM **agent's** ability to handle the geometric and chemical complexity and diversity of drug discovery tasks facing expert human medicinal chemists.

In this work, we present SMDD-Bench, an agentic benchmark designed to evaluate LLM agents, multi-agent systems, and agentic workflows on complex and realistic small molecule drug design (SMDD) tasks facing human medicinal chemists. SMDD-Bench is comprised of 5 task types with a total of 502 task instances. The task types are 2D Pharmacophore Identification, Interaction Point Discovery, Scaffold Hopping, Lead Optimization, and Fragment Assembly. Each task is deeply embedded into real-world drug discovery workflows and requires deep chemical intuition, biological reasoning ability, expertise in selecting and running specialized cheminformatics tools, hypothesis generation and validation, and understanding 3D geometries in 3D space. We also introduce the notion of witness-aware task generation, allowing us to generate a large number of chemically valid and guaranteed-solvable task instances procedurally and without a human in the loop.

Real world small molecule drug discovery requires the LLM to be able to write RDKit code, analyze protein-ligand interactions in 3D space, propose targeted molecular edits, understand how

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

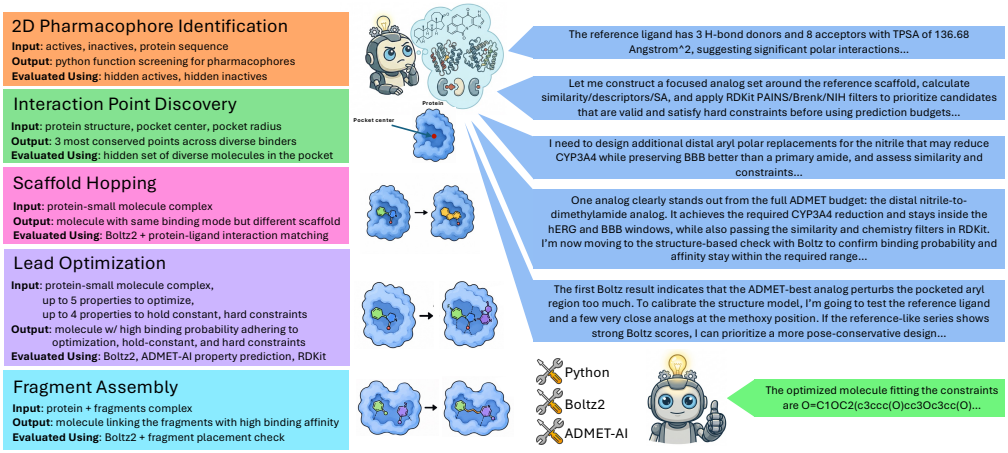

Figure 1: Overview of SMDD-Bench's task types and example reasoning turns.

those molecular edits affect predicted properties, binding affinities, and 3D conformations, and run specialized cheminformatics tools. These skills go far beyond the simple single-turn question answering tasks that previous benchmarks study and require the development of a new and challenging agentic benchmark consisting of a diverse spread of chemistries and protein targets.

However, creating real-world drug discovery tasks can be difficult due to the requirement for there to be at least one known solution to the task. For example, in lead optimization, while it is tempting to create a task instance by sampling a protein-ligand pair before sampling a set of properties to optimize as well as a set of properties to hold constant, there is no guarantee there exists a solution that adheres to the set of provided objectives and constraints. As a result, when creating task instances, in SMDD-Bench, we create a *witness-aware* task generation pipeline, allowing us to generate difficult, chemically valid, and guaranteed-solvable task instances while simultaneously generating an example solution molecule – all without requiring a human expert in the loop.

We evaluate 7 frontier open and closed source LLM agents on SMDD-Bench and find that, although LLMs achieve up to 50% on simple lead optimization tasks, more challenging lead optimization tasks as well as other task types have near 0% success rates – highlighting significant room for growth for LLM applications as autonomous medicinal chemists. We also introduce **SMDD-Bench Diversity**, a subset of SMDD-Bench along with a principled framework for studying the diversity of SMDD agents' outputs.

We hope that the release of SMDD-Bench standardizes and invigorates research into training and evaluating LLMs for real-world autonomous drug discovery.

2 SMDD-BENCH

SMDD-Bench consists of 502 instances of 5 task types: 2D Pharmacophore Identification (25 tasks), Interaction Point Discovery (25 tasks), Scaffold Hopping (52 tasks), Lead Optimization (340 tasks), and Fragment Assembly (60 tasks). Each task is generated in a **witness-aware** fashion, where a witness molecule that solves the task is simultaneously generated along with the starting molecule – ensuring the task instance is guaranteed to be solvable and has at least one solution. Each task instance is also evaluated purely computationally, without a human in the loop, using tools such as RDKit, PLIP, OpenBabel, Boltz2, and ADMET-AI (Passaro et al., 2025; Swanson et al., 2024; Landrum, 2010; Salentin et al., 2015; O'Boyle et al., 2011). Boltz2 is used for protein-ligand co-folded structure prediction, binding affinity[1], and binding probability prediction while the ADMET-AI model is used for property prediction. An overview of the benchmark can be found in Figure 1.

[1]Note that Boltz2 predicts binding affinity as $log_{10}(\text{IC50 in } \mu\text{M})$. As a result, a decrease in the Boltz2 predicted affinity value leads to stronger binding (Passaro et al., 2025)

2.1 TASK TYPES

Further details on the generation and evaluation of each task type can be found in Appendix B. Only the *Scaffold Hopping*, *Lead Optimization*, and *Fragment Assembly* tasks require **witness-aware** generation due to the *2D Pharmacophore Identification* and *Interaction Point Discovery* tasks having ground truth solutions by default.

1. 2D Pharmacophore Identification
Task: Given a target protein sequence as well as the SMILES strings associated with 10 experimentally active molecules and 10 experimentally inactive molecules from the ChEMBL database (Mendez et al., 2019), create a 2D pharmacophore in the form of a python function, which returns True if an inputted SMILES string matches the pharmacophore and False otherwise. See Appendix B.1 for more details.

Evaluation: The python function is graded on its recall and specificity performance on a held-out set of hidden, experimentally validated actives and inactives.

Generation: We filtered ChEMBL activity records for high-confidence, direct-binding assays, split the candidates into actives and inactives as determined by potency, and then pruned with MaxMin diversity picking to ensure the visible set spans chemical space and introduces binding mode diversity.

This task assesses the agent's ability to extract a generalizable, discriminative structural hypothesis from a small set of actives and inactives. In real world applications, the pharmacophore produced by the agent can be used as a first-filter for the screening of additional candidate compounds.

2. Interaction point discovery
Task: Given a PDB file of the protein structure as well as the location of the pocket, specify the 3D coordinates of the 3 interaction points within a protein pocket most likely to be conserved across a diverse set of small molecule binders. The agent must output a csv file of 3D coordinates as well as a label for the type of interaction point. The types are chosen as donor, acceptor, aromatic, hydrophobic, cation or anion interaction points.

Evaluation: The predicted points are assigned to the ground-truth points via the Hungarian algorithm. Each prediction is filtered to a ground truth point whose type is compatible (exact match, or a permitted cross-type match, see Appendix B.2 for more details), then the Hungarian algorithm computes the optimal one-to-one assignment under a distance cutoff. All three predictions must match a distinct conserved ground truth point in order for the task to be considered a success.

Generation: We determine ground truth points by fetching thousands of co-crystal structures per target from the PDB (Berman et al., 2000), aligning binding sites, extracting per-ligand protein interactions, and run density-based clustering with a conservation threshold. Steric clashes with the agent-provided prototype PDB are filtered out and ambiguous type labels are merged. As a result, it is possible for there to be more than three ground truth points of well-conserved interaction points. The agent only needs to output points that match to three of these ground truth points.

This task assesses the agent's ability to identify valuable hotspots within a protein pocket. In real world applications, these hotspots can then be targeted during fragment-based drug discovery campaigns, serve as guardrails during lead optimization, or used as 3D pharmacophores during pharmacophore-based virtual screening.

3. Scaffold hopping
Task: Given a Boltz2-folded protein and reference ligand complex, propose a new SMILES with a distinct scaffold that reproduces the reference ligand's binding interactions with the protein pocket.

Evaluation: The submitted SMILES string must have a low Tanimoto similarity with the reference molecule. The Murcko scaffolds of the submitted and reference molecules must also have a maximum common substructure which is only a portion of the overall heavy atom count of the Murcko scaffold of the reference molecule. Both of these checks ensure that a proper scaffold hop has been performed.

In order to check if the submitted molecule adheres to the same binding mode as the reference molecule, Boltz2 is used to co-fold the submitted SMILES string against the target using the same pocket contacts as the reference molecule. Potential stochasticity within the Boltz2 co-folded structure is mitigated by matching binding interactions via an ensemble of 10 Boltz2 co-folded structures.

Witness-Aware Generation: We first mine experimentally known actives for a specific pocket. Afterwards, we consider all potential pairs of actives. Every pair which the two ligands satisfy the Tanimoto distance and scaffold substructure constraint is co-folded via Boltz to the shared target. If both members of the pair share the same interaction fingerprint with the protein pocket and have high predicted binding probability, the pair and target are created into a task instance. One molecule within the pair is labeled as the reference while the other molecule is labeled as the witness. The witness molecule ensures the existence of at least one solution and guarantees the solvability of the task instance.

This task assesses the agent's ability to understand bioisosterisms as well as internally reason through binding poses before committing to making limited, computationally heavy oracle calls. In real world applications, scaffold hopping is used for rescuing lead molecules with problematic scaffolds, mitigating off-target binding effects, enabling the extension of SAR exploration beyond a depleted search, and avoiding IP infringement.

4. Lead optimization

Task: Given a reference molecule that is Boltz2 co-folded with a target protein, optimize the reference molecule into a new molecule that improves a set of ADMET and binding affinity objectives while holding other properties constant, satisfying drug-likeness constraints, maintaining high binding probability, and maintaining close Tanimoto similarity to the reference molecule (due to this being a lead optimization task).

Evaluation: The submitted molecule must satisfy the set of specified ADMET and binding affinity objectives. RDKit is used to measure drug-likeness, ADMET properties are calculated using the ADMET-AI library, and Boltz2 is used for binding probability and binding affinity prediction.

Witness-Aware Generation: For each selected protein target, we build a pool of pocket-compatible molecules that pass drug-likeness filters and a Boltz2 binding probability gate. We consider all pairs of molecules from this pool that are similar via Tanimoto similarity. From these pairs, if there exists a molecule whose ADMET/binding affinity profile dominates the other molecule on some subset of the 8 properties in SMDD-Bench (found in Table 7), the task's optimization objectives and hold-constant constraints are derived from the measured property differences between the two molecules. The dominating molecule becomes the witness while the dominated molecule becomes reference to provide to the agent to start lead optimization. The task instance is only instantiated if the witness's profile satisfies every objective and constraint. As a result, the witness solves the task instance by construction and the task becomes guaranteed solvable. More details on this can be found in Appendix B.4.

This task assesses the agent's ability to reason through how certain edits may affect complicated, intertwined relationships between molecular properties. In real world applications, multi-property optimization with optimization objectives and hold-constant constraints define the core loop of any lead optimization campaign. Lead optimization is also where the bulk of time is spent in most drug discovery programs. We envision a setting where hundreds of thousands or millions of high-performing agents are instantiated in parallel with the task of discovering a diverse array of optimized molecules given a single starting reference molecule.

5. Fragment Assembly

Task: Given either one or two 3D molecule fragments placed in a protein pocket, design a single drug-like molecule that contains each fragment as a substructure, covalently links the fragments when more than one fragment is provided, achieves high binding probability and affinity, and preserves the input fragment poses when the submitted molecule is re-docked.

Evaluation: The submitted molecule must incorporate each of the provided fragments as distinct substructures as well as pass drug-likeness checks. The submitted molecule is also co-folded with the protein via Boltz2 and must have a high predicted binding probability and binding affinity with respect to the witness molecule. Finally, the fragment substructures within the submitted molecule must be similar to the provided 3D coordinates of the input fragments after pocket alignment is performed. This is to ensure a molecule of the same binding mode as the input fragments was designed. Otherwise, it would be possible for a submitted molecule to contain the input fragments but have the fragments be in nonessential locations for binding. More details can be found in Appendix B.5.

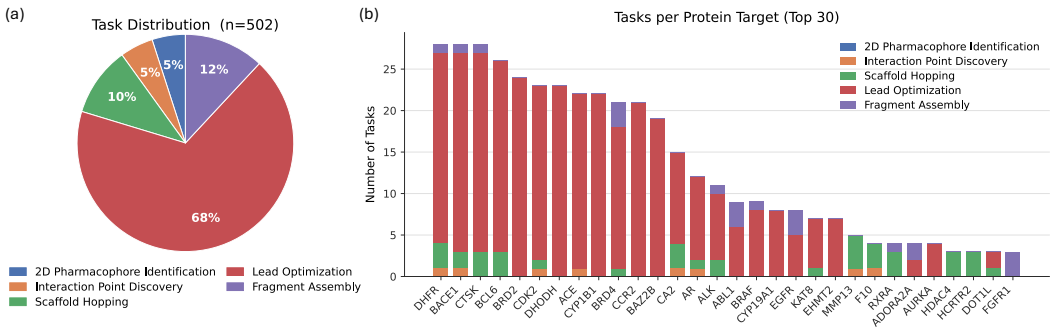

Figure 2: (a) The distribution of SMDD-Bench tasks across the five task types. Each task type is generated to be guaranteed-solvable by construction. (b). The top 30 most frequent protein targets found through all task types, broken down by task type.

Witness-Aware Generation: We co-fold known actives via Boltz2 and keep the co-folded structures which pass a binding probability gate. We perform BRICS decomposition (Degen et al., 2008) on the molecule to separate the molecule into its synthesizable fragments. Each input fragment must be large enough to be a meaningful pharmacophore, have at least one direct interaction with the protein pocket, and be in close proximity to the protein itself. Fragments are then sampled from the same molecule to create the task instance. As a result, the co-folded active molecule in which fragments are sampled from becomes the witness molecule by construction – ensuring the guaranteed solvability of the task instance. Task instances with 2 fragments have the additional constraint for fragments to be sufficiently far and not directly linked together.

This task assesses the agent's ability to reason through 3D geometry and fragment-pocket complementarity. In real world applications, fragment assembly is the central step of fragment-based drug discovery campaigns, where weakly-binding fragments identified via crystallographic or biophysical screening must be elaborated or linked into a single high-affinity lead molecule.

2.2 Benchmark Setup and Variations

SMDD-Bench Agent Setup. To evaluate the agent's innate medicinal chemistry reasoning ability, internet access is strictly prohibited, and agents have no access to academic literature, patent filings, textbooks, or any other source of non-parametric knowledge. We benchmark current frontier LLMs using a strong agent baseline (Yao et al., 2022) with a minimal harness, providing access to an isolated filesystem, a python interpreter with relevant biology and chemistry packages installed, and a limited number of Boltz2 and ADMET-AI oracle calls (the same oracles used to evaluate agent submissions). To minimize memorization, all identifying information such as PDB codes, ligand codes, Uniprot IDs, and target names are obfuscated or removed. More details on the agent harnesses can be found in Appendix C.

Benchmark Statistics. Across 502 task instances, SMDD-Bench contains 102 unique protein targets and 855 unique small molecules. Figure 2 shows the distribution of task types and protein targets across the benchmark. Figure 3(a) and (b) report, respectively, the frequencies with which each of the 9 properties appears in Lead Optimization tasks and the joint distribution of task complexity. Additional details on benchmark diversity are provided in Appendix A.

SMDD-Bench Lite. Due to the heavy computational overhead of running an agent on all 502 tasks, we release SMDD-Bench Lite, a performance-representative subset of 100 task instances designed to encourage broader adoption of the benchmark. SMDD-Bench Lite serves as a testbed for iterating on agent development against a difficulty-diverse and representative sample of the full benchmark. Details on how the SMDD-Bench Lite instances are selected can be found in Appendix E.

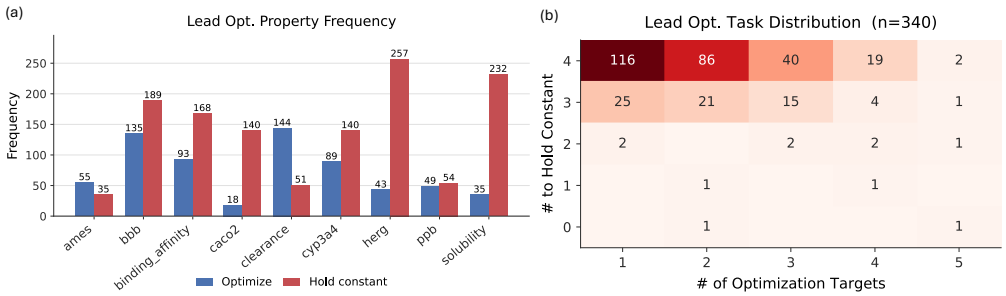

Figure 3: (a) Frequency with which each ADMET and binding affinity property appears as an optimization objective versus a hold-constant constraint across the 340 Lead Optimization instances. (b) Joint distribution of the number of optimization objectives and hold-constant properties across the 340 Lead Optimization task instances. Each cell reports the number of task instances with the given (hold-constant, optimization target) pair.

3 RESULTS

LLMs do not perform well on real-world small molecule drug discovery tasks. We evaluated 7 frontier LLMs on SMDD-Bench using the agent harness described in Section 2.2 and present results in Table 1. As described in Appendix C, each agent was provided with 8 Boltz2 oracle calls and 15 ADMET-AI oracle call – each ADMET-AI oracle call returns all 8 predicted ADMET properties. Closed source models such as Claude Sonnet 4.6, GPT-5.4 and Gemini 3.1 pro significantly outperform leading open source models such as Kimi K2.5 Thinking, Deepseek V3.2, and Qwen 3.5-397B-A17B with GPT5.4 setting the state of the art at 40.2% overall success rate.

However, most of the successes across all models are a result of performing well on the Lead Optimization task. Particularly, current LLMs perform poorly on tasks that require a heavy intuition and understanding of 3D geometries and interactions such as the Interaction Point Discovery, Scaffold Hopping and Fragment Assembly tasks. Considering the overwhelming challenges of multi-property, multi-constraint lead optimization, frontier LLMs perform surprisingly well at the Lead Optimization task. More specific results on LLM performance, such as cost breakdown and tool usage can be found in Appendix D.

Table 1: Success rate of 7 frontier closed and open source LLMs on SMDD-Bench, broken down by task type. Each LLM is evaluated using the agent harness outlined in Section 2.2 and Appendix C with a limited number of oracle calls.

Model	Success Rate (%)						Cost ($/instance)
	2D Pharma-cophore ID	Interaction Point Discovery	Scaffold Hopping	Lead Optimization	Fragment Assembly	Overall	
Claude Sonnet 4.6	**28.0**	0.0	**3.8**	53.5	0.0	38.0	1.31
GPT-5.4	12.0	0.0	**3.8**	**57.6**	**1.7**	**40.2**	0.78
Gemini 3.1 Pro	20.0	**4.0**	0.0	55.6	**1.7**	39.0	0.63
Kimi K2.5 Thinking	12.0	0.0	1.9	43.5	0.0	30.3	0.40
DeepSeek V3.2	8.0	0.0	**3.8**	34.7	0.0	24.3	0.43
MiniMax M2.7	16.0	0.0	1.9	27.1	0.0	19.3	**0.36**
Qwen3.5 397B A17B	4.0	0.0	1.9	40.0	0.0	27.5	0.75

The chemical novelty of the submissions of each agent, broken down into task type, are found in Table 2. Chemical novelty is defined as the % of total agent submissions which are not found in the ChEMBL36, SureChEMBL, PubChem, of BindingDB databases (Mendez et al., 2019; Papadatos et al., 2016; Kim et al., 2023; Liu et al., 2007; 2025b). These databases are chosen due to their potential inclusion of bioactivity data for small molecules with respect to protein targets. The novelty check we prescribe is a conservative measure of the agent's ability to design a molecule in the context of a protein target for which there doesn't exist pre-existing training data on.

LLMs are not diverse in their outputs. In real world applications, during lead optimization campaigns, researchers may want to instantiate many agents or agentic systems in parallel on the same task instance. The goal would be for the agents to propose a diverse and novel spread of successful solutions rather than converging on a single solution, allowing for a more robust exploration and

Table 2: Novelty of agent submissions, defined by % of submitted molecules not found in ChEMBL, PubChem, SureChEMBL, or BindingDB.

Model	Scaffold Hopping	Lead Optimization	Fragment Assembly
Claude Sonnet 4.6	88.5	77.9	82.8
GPT-5.4	**94.2**	75.3	68.3
Gemini 3.1 Pro	71.2	76.5	**83.3**
Kimi K2.5 Thinking	80.8	77.6	65.0
DeepSeek V3.2	**94.2**	83.2	69.1
MiniMax M2.7	82.7	**86.5**	67.2
Qwen3.5 397B A17B	82.7	77.9	61.7

discovery of lead molecules. In order to test for this behavior, we propose **SMDD-Bench Diversity**, a strict subset of 20 SMDD-Bench task instances for the Lead Optimization task type. The task instances represent the most challenging lead optimization tasks that are still solvable by current frontier LLMs. See Appendix F for more details on selecting SMDD-Bench Diversity.

For each task on SMDD-Bench Diversity, we run each agent 10 times and report the distribution of novel and diverse molecules which pass the evaluation requirements. In order to measure diversity, we measure the average number of unique successful solutions per task instance, the average pairwise tanimoto similarity of successful solutions, the percentage of successful solutions which are novel (using the same criteria as in Table 2), and the average number of successful solutions of each LLM on the subset. Results are found in Table 3.

Table 3: Statistics across 10 runs of each of the 20 task instances in the SMDD-Bench Diversity subset. Novelty is defined in Table 2. "Avg. Successful" and "Avg. Unique and Successful" categories are out of 10 due to the 10 runs performed on each task instance.

Model	Avg. Successful	Avg. Unique and Successful	Novel and Successful (%)	Pairwise Tanimoto Similarity
Claude Sonnet 4.6	**8.40**	3.70	**74.0**	0.823
GPT-5.4	7.90	2.75	64.6	0.863
Gemini 3.1 Pro	8.00	4.00	67.6	0.809
Kimi K2.5 Thinking	6.00	3.85	65.0	0.786
DeepSeek V3.2	5.35	3.85	68.4	**0.763**
MiniMax M2.7	6.00	**4.05**	73.1	**0.763**
Qwen3.5 397B A17B	7.25	3.55	67.2	0.814

Although Claude Sonnet 4.6 consistently achieves the highest number of successful submissions, the model converges to the same set of molecules over half the time. Surprisingly, Minimax M2.7 submits the most unique spread of successful molecules on SMDD-Diversity while maintaining strong % novelty across successful submissions. SMDD agents deployed in the wild must consistently achieve a high number of both unique and novel successful solutions when provided the same task instance.

Enumeration vs. Selection. Throughout the solve process, each LLM enumerates many potential candidates, reasons through those candidates, and then runs the most promising of the candidates through an oracle tool call. We investigate whether or not LLMs are capable of enumerating passing molecules but fail at the reasoning and selection step. In order to do so, we use GPT5.4 mini to extract every SMILES string mentioned in each of the agent's traces for the Scaffold Hopping, Lead Optimization, and Fragment Assembly tasks. We then, using the task instance's evaluation pipeline, evaluate all of the molecules that the agent did not perform an oracle call on – this set of molecules represents the molecules which the agent hypothesized could be strong candidates, but considered to be inferior. We then calculate the overall recovered success rate on SMDD-Bench had the LLM selected the best candidate from its enumeration and report the values in Table 4. More details on how we performed this experiment can be found in Appendix G.

Scaffold hopping sees a significant increase in recovered rate for Claude Sonnet 4.6, Gemini 3.1 Pro, and GPT5.4 – indicating that the agents are not as capable at reasoning through binding interactions of chemically different analogues without first performing oracle calls. On the other hand, fragment assembly, a task requiring a deep understanding of how the binding pocket and surrounding chemistry

Table 4: Original vs recovered success rates (%) and total number of enumerated molecules (#Mols) for all agents on SMDD-Bench. The recovered success rate is the rate the agent would have if it had selected to submit the ground truth best molecule that it enumerated during the reasoning process.

	Scaffold Hopping			Lead Optimization			Fragment Assembly			Overall		
Model	Orig. (%)	Recov. (%)	#Mols	Orig. (%)	Recov. (%)	#Mols	Orig. (%)	Recov. (%)	#Mols	Orig. (%)	Recov. (%)	#Mols
Claude Sonnet 4.6	3.8	32.7	2657	53.5	64.1	5872	0.0	3.3	4982	40.7	52.4	13 511
DeepSeek V3.2	3.8	11.5	1052	34.7	39.4	3413	0.0	0.0	1358	26.5	31.0	5823
Gemini 3.1 Pro	0.0	19.2	735	55.6	62.9	3328	1.7	3.3	1855	42.0	50.0	5918
GPT-5.4	3.8	15.4	850	57.6	61.2	4486	1.7	1.7	1945	44.0	48.0	7281
Kimi K2.5 Thinking	1.9	3.8	986	43.5	53.8	4940	0.0	1.7	2528	33.0	41.2	8454
MiniMax M2.7	1.9	5.8	1203	27.1	47.1	7231	0.0	0.0	2372	20.6	36.1	10 806
Qwen3.5 397B	1.9	3.8	1055	40.0	53.8	4367	0.0	0.0	1915	30.3	40.9	7337

affects the 3D geometry of the small molecule binding pose, sees very minimal overall improvement. Interestingly, MiniMax M2.7 nearly doubles its success rate for the Lead Optimization task, indicating its ability to enumerate solutions and inability to select from the enumerations.

SMDD-Bench Lite performance. The performance of each agent on the SMDD-Bench Lite subset are reported in Table 5. We also report the performance of Gemini 3.1 pro on high thinking mode.

Table 5: Success rate (%) by task type and overall cost per instance of 8 frontier LLMs on the SMDD-Bench Lite subset.

	Success Rate (%)						Cost
Model	2D Pharmacophore ID	Interaction Point Discovery	Scaffold Hopping	Lead Optimization	Fragment Assembly	Overall	Cost ($/instance)
Claude Sonnet 4.6	40.0	0.0	0.0	52.9	0.0	38.0	1.29
GPT-5.4 (medium)	0.0	0.0	0.0	54.4	0.0	37.0	0.77
Gemini 3.1 Pro (medium)	20.0	0.0	0.0	54.4	0.0	38.0	0.57
Gemini 3.1 Pro (high)	0.0	0.0	20.0	52.9	8.3	39.0	0.46
Kimi K2.5 Thinking	0.0	0.0	0.0	42.6	0.0	29.0	0.38
DeepSeek V3.2	0.0	0.0	10.0	33.8	0.0	24.0	0.40
MiniMax M2.7	20.0	0.0	0.0	25.0	0.0	18.0	0.34
Qwen3.5 397B A17B	0.0	0.0	0.0	38.2	0.0	26.0	0.79

For completeness, we also benchmark non-LLM baselines against the SMDD-Bench benchmark in Appendix H. Non-LLM baselines do not outperform their LLM agent counterparts.

4 Common Failure Modes

Beyond the issues regarding the novelty rate, diversity rate, and enumeration vs. selection performance gap, we investigated individual traces searching for other common failure modes in the frontier LLMs.

Absence of Cross-Turn Structure–Activity Relationship (SAR) Synthesis. A recurring failure across models is their inability to generalize SAR rules. When a molecule violates predefined constraints, a budget-aware agent should ask whether a specific structural feature caused this failure and apply this exclusion rule to later candidates. Instead, agents repeatedly test variants of already disqualified structures, effectively wasting budget on confirming the same rule multiple times. This reflects a lack of meta-cognitive reasoning: agents fail to convert individual failures into structural generalizations that could narrow down the search space. We provide an annotated reasoning trace in Appendix J.1.

Incoherent planning across multiple turns. We notice that a common failure mode involves candidate-generation turns being treated independently rather than as part of an iterative, sequential process. This results in the agent generating new molecules instead of selecting from candidates that passed earlier-stage criteria. Furthermore, the same molecule may be proposed multiple times across turns even though it had already been tested in a previous turn and did not meet proper thresholds. Refer to a representative trace in Appendix J.2.

Code Execution Errors. Agents also exhibit tool-specific coding mistakes. These include malformed calls to structure-prediction tools (e.g., incorrect argument types or missing required fields for Boltz binding predictions, cheminformatics rountines (e.g., `MolToSmiles` conversion), and interaction-analysis pipelines (e.g., PLIP).

5 Related Work

SMDD-Bench evaluates LLM agents on challenging, real-world small molecule drug design tasks. Existing chemistry and molecular benchmarks – ChemBench (Mirza et al., 2025), MolecularIQ (Bartmann et al., 2026), SmolInstruct (Yu et al., 2024), QCBench (Xie et al., 2025), and MuMOInstruct (Dey et al., 2025) – are restricted to single-turn question answering on relatively simple chemistry tasks and do not evaluate complex agentic systems on real-world problems. Prior work on multi-turn iterative molecular optimization (Chennakesavalu et al., 2026; Wang et al., 2024) likewise lacks realistic agentic tool calling, instead querying the LLM repeatedly in a loop over a small number of simple optimization objectives. In contrast, harnessing LLM agents as molecular designers lets them use tools to understand the 3D problem setting, analyze previous failed hypotheses, and run simulations and other cheminformatics tools before committing to oracle calls. Iterative prompting approaches are not faithful representations of the true potential for LLM agents in drug discovery.

Beyond single-turn QA, prior LLM agents for drug design and discovery (Ock et al., 2026; Li et al., 2026a;b; Zhang et al., 2024; Wang et al., 2024) either report results on a few hand-selected successes or use ad hoc, non-standardized evaluation setups, often retrofitting datasets built for de novo molecular generation such as ZINC, Therapeutics Data Commons, GuacaMol, PMO, and MOSES (Tingle et al., 2023; Huang et al., 2021; Brown et al., 2019; Gao et al., 2022; Polykovskiy et al., 2020). Adjacent domains like computational biology have produced agentic benchmarks such as DeepResearch Bench (Du et al., 2025), BioProBench (Liu et al., 2025c), BioLP-Bench (Ivanov, 2024), ABC-Bench (Liu et al., 2025a), Lab-Bench (Laurent et al., 2024), BixBench (Mitchener et al., 2025), and CompBioBench (Nair et al., 2026), but no prior work evaluates LLMs on agentic small molecule drug design.

The primary contributions of SMDD-Bench compared to previous work are as follows: (1) the first standardized benchmark for evaluating long-horizon, multi-turn, small molecule drug design **agents** on a chemically diverse array of challenging drug discovery tasks; (2) support for five relevant, real-world task types: 2D Pharmacophore Identification, Interaction Point Discovery, Scaffold Hopping, Lead Optimization, and Fragment Assembly; and (3) guaranteed solvability for all tasks, including the Scaffold Hopping, Lead Optimization, and Fragment Assembly tasks via the existence of a held-out witness molecule which passes all evaluation requirements.

6 Discussion and Conclusion

We present SMDD-Bench, the first agentic benchmark evaluating LLM agents on challenging, long-horizon, real-world small molecule drug design. SMDD-Bench tests scientific and chemical reasoning across diverse chemistries, task types, and protein targets, alongside domain-specific tool-calling capabilities, with every task instance generated to be guaranteed-solvable. Our hope is that SMDD-Bench unifies and standardizes the design, training, and evaluation of LLMs for small molecule drug design. The dominant approach to small molecule drug design today is conditional generative diffusion and flow models (Sun et al., 2025; Zeng et al., 2026; Zhang et al., 2025; Morehead et al., 2026; Klein et al., 2023; Hoogeboom et al., 2022), but LLM-based SMDD agents offer distinct advantages in flexibility and generality. An agent can perform lead optimization on properties unseen at training time, understand the failure modes of specific oracles, selecting the right tool for a given task, and adhere to constraints (such as maintaining protein-protein contacts when designing a PROTAC) that were never specified during training. Communicating objectives in natural language further frees the agent from the rigid train-time settings that bind diffusion and flow models.

Boltz2 and ADMET-AI. We deliberately provide a limited number of oracle calls to reflect the sparsity of real-world wet lab experiments. To enforce that the Boltz2 and ADMET-AI predicted distributions are the operative ground truth, all PDB-sourced complexes are re-folded via Boltz2 and re-scored by ADMET-AI during task generation, so the agent only sees structure-activity relationships and property distributions produced by these oracles. SMDD-Bench is intended as a testbed for developing training frameworks calibrated to the Boltz2 and ADMET-AI distributions. The same frameworks can later be interfaced with an autonomous laboratory, where the oracle becomes a real wet lab that synthesizes, characterizes, and tests candidates, **allowing agents to be re-trained under the true distribution of real-world structure-activity relationships**.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in implementing a small portion of the methods and providing feedback on research methodology or experiments. We have not used generative AI tools for developing theoretical models or conceptual frameworks, proposing or refining hypotheses, or supporting qualitative and thematic analysis or interpreting results, and generating synthetic datasets, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools for editing software code and identifying relevant literature. We have reviewed all AI-assisted work. For example, all LLM-generated code was verified and tested for correctness by at least 2 authors as well as an extensive set of test cases. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, Mike D'Arcy, et al. Synthesizing scientific literature with retrieval-augmented language models. *Nature*, 2026.

Mark Ashton, John Barnard, Florence Casset, Michael Charlton, Geoffrey Downs, Dominique Gorse, John Holliday, Roger Lahana, and Peter Willett. Identification of diverse database subsets using property-based and fragment-based molecular descriptions. *Quantitative Structure-Activity Relationships*, 21(6):598–604, 2002.

Jonathan B Baell. Feeling nature's pains: natural products, natural product drugs, and pan assay interference compounds (pains). *Journal of natural products*, 79(3):616–628, 2016.

Christoph Bartmann, Johannes Schimunek, Mykyta Ielanskyi, Philipp Seidl, Günter Klambauer, and Sohvi Luukkonen. Molecullariq: Characterizing chemical reasoning capabilities through symbolic verification on molecular graphs. *arXiv preprint arXiv:2601.15279*, 2026.

Guy W Bemis and Mark A Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996.

H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000. doi: 10.1093/nar/28.1.235.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Ruth Brenk, Alessandro Schipani, Daniel James, Agata Krasowski, Ian Hugh Gilbert, Julie Frearson, and Paul Graham Wyatt. Lessons learnt from assembling screening libraries for drug discovery for neglected diseases. *ChemMedChem: Chemistry Enabling Drug Discovery*, 3(3):435–444, 2008.

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3): 1096–1108, 2019.

Joseph M Cavanagh, Kunyang Sun, Andrew Gritsevskiy, Dorian Bagni, Yingze Wang, Thomas D Bannister, and Teresa Head-Gordon. Smileyllama: Modifying large language models for directed chemical space exploration. *Nature Computational Science*, pp. 1–14, 2026.

Shriram Chennakesavalu, Kirill Shmilovich, Hayley Weir, Colin Grambow, John Bradshaw, Patricia Suriana, Chen Cheng, and Kangway Chuang. Evaluating the progression of large language model capabilities for small-molecule drug design. *arXiv preprint arXiv:2604.16279*, 2026.

Jorg Degen, Christof Wegscheid-Gerlach, Andrea Zaliani, and Matthias Rarey. On the art of compiling and using'drug-like'chemical fragment spaces. *ChemMedChem*, 3(10):1503, 2008.

Vishal Dey, Xiao Hu, and Xia Ning. Gellm3o: Generalizing large language models for multi-property molecule optimization. *arXiv preprint arXiv:2502.13398*, 10, 2025.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*, 2025.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8, 2009.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pp. 226–231, 1996.

Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35: 21342–21357, 2022.

M Paul Gleeson. Generation of a set of simple, interpretable admet rules of thumb. *Journal of medicinal chemistry*, 51(4):817–834, 2008.

Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pp. 8867–8887. PMLR, 2022.

Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *arXiv preprint arXiv:2102.09548*, 2021.

Igor Ivanov. Biolp-bench: Measuring understanding of biological lab protocols by large language models. *bioRxiv*, pp. 2024–08, 2024.

Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, et al. Pubchem 2023 update. *Nucleic acids research*, 51(D1): D1373–D1380, 2023.

Leon Klein, Andreas Krämer, and Frank Noé. Equivariant flow matching. *Advances in Neural Information Processing Systems*, 36:59886–59910, 2023.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Greg Landrum. Rdkit: Open-source chemoinformatics. `http://www.rdkit.org`, 2010. Accessed: 2026-04-11.

Jon M Laurent, Joseph D Janizek, Michael Ruzo, Michaela M Hinks, Michael J Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D White, and Samuel G Rodriques. Lab-bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024.

Paul D Leeson and Brian Springthorpe. The influence of drug-like concepts on decision-making in medicinal chemistry. *Nature reviews Drug discovery*, 6(11):881–890, 2007.

Hao Li, He Cao, Shenyao Peng, Zijing Liu, Bin Feng, Yu Wang, Zhiyuan Yan, Yonghong Tian, Yu Li, and Li Yuan. Agentic reinforcement learning empowers next-generation chemical language models for molecular design and synthesis. *arXiv preprint arXiv:2601.17687*, 2026a.

Tao Li, Kaiyuan Hou, Tuan Vinh, Monika Raj, Zhichun Guo, and Carl Yang. Reinforcement learning with llm-guided action spaces for synthesizable lead optimization. *arXiv preprint arXiv:2604.07669*, 2026b.

Carmen Limban, Diana C Nuţă, Cornel Chiriţă, Simona Negreș, Andreea L Arsene, Marina Goumenou, Spyros P Karakitsios, Aristidis M Tsatsakis, and Dimosthenis A Sarigiannis. The use of structural alerts to avoid the toxicity of pharmaceuticals. *Toxicology reports*, 5:943–953, 2018.

Andrew Bo Liu, Samira Nedungadi, Bryce Cai, Alex Kleinman, Harmon Bhasin, and Seth Donoughe. Abc-bench: An agentic bio-capabilities benchmark for biosecurity. In *NeurIPS 2025 Workshop on Biosecurity Safeguards for Generative AI*, 2025a.

Tiqing Liu, Yuhmei Lin, Xin Wen, Robert N Jorissen, and Michael K Gilson. Bindingdb: a web-accessible database of experimentally determined protein–ligand binding affinities. *Nucleic acids research*, 35(suppl_1):D198–D201, 2007.

Tiqing Liu, Linda Hwang, Stephen K Burley, Carmen I Nitsche, Christopher Southan, W Patrick Walters, and Michael K Gilson. Bindingdb in 2024: a fair knowledgebase of protein-small molecule binding data. *Nucleic acids research*, 53(D1):D1633–D1644, 2025b.

Yuyang Liu, Liuzhenghao Lv, Xiancheng Zhang, Jingya Wang Li Yuan, and Yonghong Tian. Bioprobench: Comprehensive dataset and benchmark in biological protocol understanding and reasoning. *arXiv preprint arXiv:2505.07889*, 2025c.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026.

David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michal Nowotka, María Gordillo-Marañón, Fiona Hunter, Laura Junco, Grace Mugumbate, Milagros Rodriguez-Lopez, Francis Atkinson, Nicolas Bosc, Chris J Radoux, Aldo Segura-Cabrera, Anne Hersey, and Andrew R Leach. Chembl: towards direct deposition of bioassay data. *Nucleic acids research*, 47(D1): D930—D940, January 2019. ISSN 0305-1048. doi: 10.1093/nar/gky1075. URL `https://europepmc.org/articles/PMC6323927`.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, et al. A framework for evaluating the chemical knowledge and reasoning abilities of large language models against the expertise of chemists. *Nature Chemistry*, 17(7):1027–1034, 2025.

Ludovico Mitchener, Jon M Laurent, Alex Andonian, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology. *arXiv preprint arXiv:2503.00096*, 2025.

Alex Morehead, Miruna Cretu, Antonia Panescu, Rishabh Anand, Maurice Weiler, Tynan Perez, Samuel Blau, Steven Farrell, Wahid Bhimji, Anubhav Jain, et al. Zatom-1: A multimodal flow foundation model for 3d molecules and materials. *arXiv preprint arXiv:2602.22251*, 2026.

Surag Nair, Laura Gunsalus, Brian Orcutt-Jahns, Jordan Rossen, Avantika Lal, Carlo De Donno, Muhammed Hasan Celik, Kipper Fletez-Brant, Xiaoman Xie, Hector Corrada Bravo, et al. Agentic systems are adept at solving well-scoped, verifiable problems in computational biology. *bioRxiv*, pp. 2026–04, 2026.

Noel M O'Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. Open babel: An open chemical toolbox. *Journal of Cheminformatics*, 3:33, 2011. doi: 10.1186/1758-2946-3-33.

Janghoon Ock, Radheesh Sharma Meda, Srivathsan Badrinarayanan, Neha S Aluru, Achuth Chandrasekhar, and Amir Barati Farimani. Large language model agent for modular task execution in drug discovery. *Journal of Chemical Information and Modeling*, 66(4):2055–2068, 2026.

George Papadatos, Mark Davies, Nathan Dedman, Jon Chambers, Anna Gaulton, James Siddle, Richard Koks, Sean A Irvine, Joe Pettersson, Nicko Goncharoff, et al. Surechembl: a large-scale, chemically annotated patent document database. *Nucleic acids research*, 44(D1):D1220–D1228, 2016.

S Passaro, G Corso, J Wohlwend, M Reveiz, S Thaler, VR Somnath, N Getz, T Portnoi, J Roy, H Stark, et al. Boltz-2: Towards accurate and efficient binding affinity prediction. biorxiv 2025. *Google Scholar*, 2025.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020.

Sebastian Salentin, Sven Schreiber, V Joachim Haupt, Melissa F Adasme, and Michael Schroeder. Plip: fully automated protein–ligand interaction profiler. *Nucleic acids research*, 43(W1):W443–W447, 2015.

Kunyang Sun, Yingze Wang, Justin Purnomo, Joseph M Cavanagh, Giovanni Battista Alteri, and Teresa Head-Gordon. Linkllama: Enabling large language model for chemically reasonable linker design. *bioRxiv*, 2026.

Yan Sun, Lianghong Chen, Zihao Jing, Yan Yi Li, Dongkyu Kim, Jing-Yan Gao, Reza Noroozi, Grace Y Yi, Conrard Giresse Tetsassi Feugmo, Anna Klinkova, et al. Generative ai for the design of molecules: advances and challenges. *Journal of Chemical Information and Modeling*, 65(23): 12668–12690, 2025.

Kyle Swanson, Parker Walther, Jeremy Leitz, Souhrid Mukherjee, Joseph C Wu, Rabindra V Shivnaraine, and James Zou. Admet-ai: a machine learning admet platform for evaluation of large-scale chemical libraries. *Bioinformatics*, 40(7):btae416, 06 2024. ISSN 1367-4811. doi: 10.1093/bioinformatics/btae416. URL https://doi.org/10.1093/bioinformatics/btae416.

Benjamin I Tingle, Khanh G Tang, Mar Castanon, John J Gutierrez, Munkhzul Khurelbaatar, Chinzorig Dandarchuluun, Yurii S Moroz, and John J Irwin. Zinc-22– a free multi-billion-scale database of tangible compounds for ligand discovery. *Journal of chemical information and modeling*, 63(4): 1166–1176, 2023.

Manthena VS Varma, Bo Feng, R Scott Obach, Matthew D Troutman, Jonathan Chupka, Howard R Miller, and Ayman El-Kattan. Physicochemical determinants of human renal clearance. *Journal of medicinal chemistry*, 52(15):4844–4852, 2009.

Haorui Wang, Marta Skreta, Cher-Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, et al. Efficient evolutionary search over chemical space with large language models. *arXiv preprint arXiv:2406.16976*, 2024.

Michael J Waring. Lipophilicity in drug discovery. *Expert opinion on drug discovery*, 5(3):235–248, 2010.

Jiaqing Xie, Weida Wang, Ben Gao, Zhuo Yang, Haiyuan Wan, Shufei Zhang, Tianfan Fu, and Yuqiang Li. Qcbench: Evaluating large language models on domain-specific quantitative chemistry. *Journal of Chemical Information and Modeling*, 65(22):12268–12278, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Botao Yu, Frazier N Baker, Ziqi Chen, Xia Ning, and Huan Sun. Llasmol: Advancing large language models for chemistry with a large-scale, comprehensive, high-quality instruction tuning dataset. *arXiv preprint arXiv:2402.09391*, 2024.

Cheng Zeng, Jirui Jin, Connor Ambrose, George Karypis, Mark Transtrum, Ellad B Tadmor, Richard G Hennig, Adrian Roitberg, Stefano Martiniani, and Mingjie Liu. Propmolflow: property-guided molecule generation with geometry-complete flow matching. *Nature Computational Science*, pp. 1–10, 2026.

Odin Zhang, Jieyu Jin, Zhenxing Wu, Jintu Zhang, Po Yuan, Yuntao Yu, Haitao Lin, Haiyang Zhong, Xujun Zhang, Chenqing Hua, et al. Ecloudgen: leveraging electron clouds as a latent variable to scale up structure-based molecular design. *Nature Computational Science*, pp. 1–12, 2025.

Yu Zhang, Xiusi Chen, Bowen Jin, Sheng Wang, Shuiwang Ji, Wei Wang, and Jiawei Han. A comprehensive survey of scientific large language models and their applications in scientific discovery. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8783–8817, 2024.

APPENDIX CONTENTS

A Detailed Benchmark Diversity **15**

B Details on Task Type Generation **15**

B.1 2D Pharmacophore Identification 15

B.2 Interaction Point Discovery 18

B.3 Scaffold Hopping 19

B.4 Lead Optimization 20

B.5 Fragment Assembly 22

C Agent Setup **23**

C.1 Selecting Oracle Calls 23

D Additional Agent Results **24**

D.1 Indicators of the Difficulty for Lead Optimization Tasks 24

D.2 Analysis on inter-agent submission diversity 26

E Selecting SMDD-Bench Lite **28**

F Selecting SMDD-Bench Diversity **30**

G Enumeration vs. Selection Details **31**

H Non-LLM baselines **32**

I Example Task Instances, Trajectories, and Analysis **32**

I.1 2D Pharmacophore Identification 32

I.2 Interaction Point Discovery 40

I.3 Scaffold Hopping 46

I.4 Lead Optimization 54

I.5 Fragment Assembly 61

J Failure Mode Analysis **67**

J.1 Absence of Cross-Turn SAR Synthesis: Repeated Testing of a Disqualified Structure 67

J.2 Repeated Proposal of a Failed Candidate Across Tool Invocations 72

J.3 Recall of Known Molecules as a Substitute for De Novo Design 73

A Detailed Benchmark Diversity

The complete list of the 102 unique protein targets broken down into task types can be found in Figure 4. In Figure 5, we plot each of the reference molecule baseline property values for optimization properties and hold-constant properties in the Lead Optimization task. Blood brain barrier penetration can either be optimized to enhance penetration or to decrease penetration depending on the real world application. Both directions of optimization are included in SMDD-Bench.

In Figure 6, we plot the pairwise Tanimoto similarity between all pairs of reference molecules in the Scaffold Hopping and Lead Optimization tasks. The low distribution of Tanimoto similarity indicates that the bulk of the reference molecules aren't collapsed to similar points in chemical space – highlighting the small molecule diversity of SMDD-Bench.

Figure 7 shows the breakdown of the protein targets within each task instance into their respective families, highlighting the protein diversity of SMDD-Bench. For an agent to be successful, it must be able to generalize and reason across diverse chemistries, pockets, and biological targets.

B Details on Task Type Generation

In the following section, we provide the specific details for generating the task instances for each task type.

B.1 2D Pharmacophore Identification

Each instance of this task type asks the agent to identify a 2D pharmacophore that distinguishes a set of active molecules from a set of inactive molecules for a given protein target. Tasks are generated procedurally from ChEMBL.

Activity data extraction We restrict to direct binding assays by requiring `standard_type` $\in \{\text{IC}_{50}, K_i\}$, `standard_units` = nM, `standard_relation` = '=', and `confidence_score` ≥ 8. Only targets with `target_type` = 'SINGLE PROTEIN' are considered.

Compound normalization and deduplication. SMILES are canonicalized via RDKit. We discard multi-component salts/mixtures, molecules with exact molecular weight greater than 900 Da, and entries that fail RDKit parsing. Multiple measurements of the same canonical compound are collapsed by taking the per-compound median activity in nM.

Active/inactive labeling Compounds with median activity below 1,000 nM ($\text{IC}_{50}/K_i < 1$ μM) are labeled active; compounds with median activity above 10,000 nM (> 10 μM) are labeled inactive. Compounds in the 1–10 μM ambiguous zone are discarded, providing a $10\times$ potency gap between the two classes that excludes borderline molecules. Targets with fewer than 30 actives or fewer than 30 inactives after filtering are not eligible.

Diverse subset selection For each class, ten visible exemplars are selected via the MaxMin algorithm operating on Morgan fingerprints (radius = 2, 2048 bits, seed = 42) (Ashton et al., 2002). The remaining compounds form a held-out evaluation set that the agent never sees. MaxMin selection forces the visible exemplars to span chemical space, allowing the agent to see the most chemically diverse spread of actives as possible.

Agent inputs and outputs Each generated task directory contains: `actives.smi` and `inactives.smi` (the 10 visible exemplars per class, given to the agent), `protein.fasta` (the target sequence). The agent is required to emit `solution.py` defining `check_pharmacophore(smiles: str) -> bool`, which is run on every molecule of the held-out evaluation set during scoring.

Design rationale Using ChEMBL-measured inactives rather than synthetically constructed decoys (e.g. DUD-E) ensures that negatives are experimentally verified non-binders that may share scaffolds with the actives, ruling out solutions that rely on coarse topological dissimilarity. Providing both classes to the agent and requiring generalization to a held-out set forces extraction of a genuine substructural pattern rather than memorization of the visible exemplars.

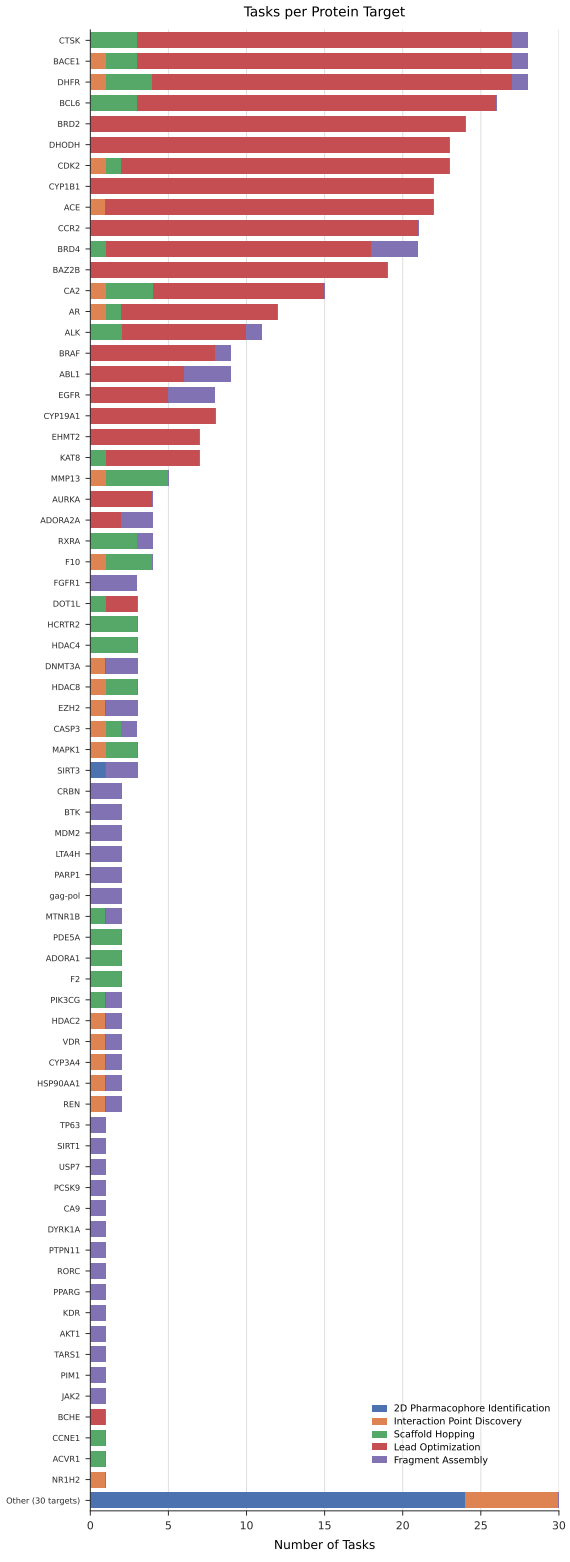

Figure 4: The complete breakdown of task instances into protein targets. There are 102 unique protein targets in SMDD-Bench.

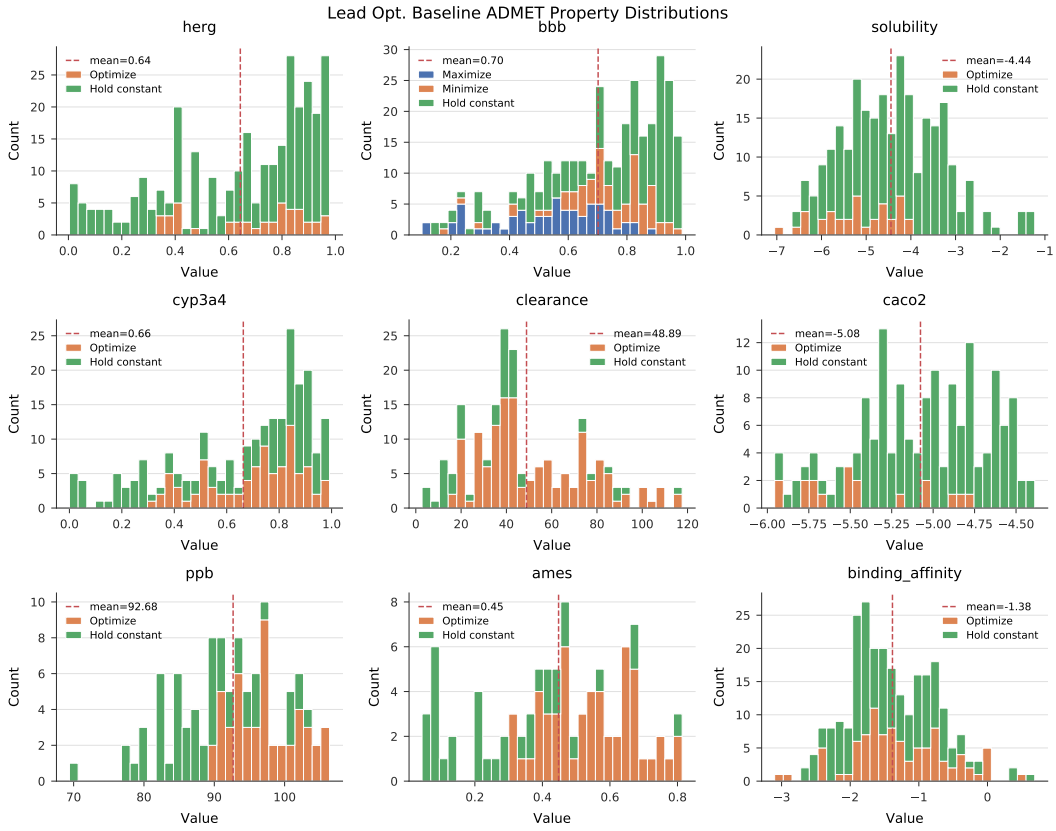

Figure 5: The baseline ADMET property values for all optimization objectives and hold-constant properties for the Lead Optimization task type. These are the values calculated for the reference molecules that the LLM agent must either optimize or hold constant. Note that blood brain barrier penetration (bbb) can either be a property that is minimized or maximized. All other properties are either only minimized or only maximized.

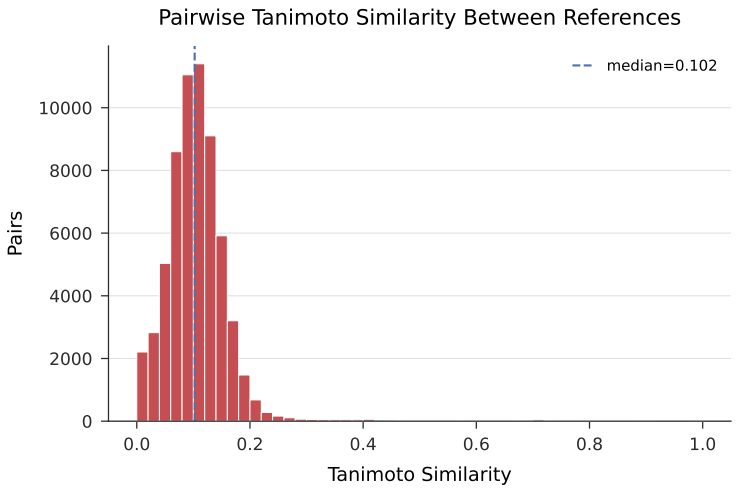

Figure 6: A histogram of the tanimoto similarities between all pairs of reference molecules provided to the LLM agent in the Scaffold Hopping and Lead Optimization tasks. The low median tanimoto similarity between pairs indicates the chemical diversity of SMDD-Bench.

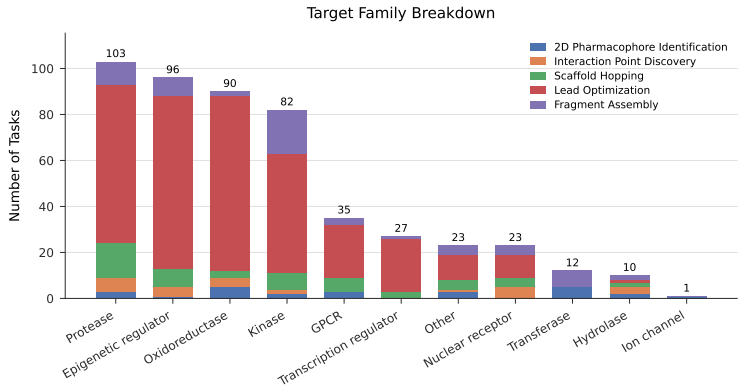

Figure 7: A breakdown of the task types of SMDD-Bench into the families of the protein targets associated with each task instance.

B.2 Interaction Point Discovery

Each instance of this task type asks the agent to predict, given a single apo-like protein structure and a pocket center, the 3D coordinates and pharmacophoric type of the three interaction points most likely to be conserved across diverse binders of that pocket. Tasks are generated procedurally from the RCSB PDB (Berman et al., 2000).

Ensemble construction For each specified target from the PDB, we fetch up to 500 unique X-ray crystal structures with resolution better than 2.5 Å. The first structure containing a drug-like ligand is taken as the *reference*; common crystallization additives (glycerol, PEG, DMSO, sulfate, and similar) are explicitly blacklisted. Within that reference structure, the largest non-blacklisted ligand by heavy-atom count is selected as the reference ligand. The reference receptor is stripped of all heteroatoms, waters, and ions to produce `receptor.pdb`, and the centroid of the reference ligand defines the pocket center provided to the agent.

Local structural alignment All ensemble members are superimposed onto the reference using only the C_α atoms of residues within 12 Å of the pocket center. Restricting the alignment to the local pocket suppresses contributions from global protein motion and induced fit – minimizing potential global variance in the 3D reference frame.

Pharmacophore feature extraction For each aligned structure, we select the ligand whose centroid lies closest to the reference pocket center (within 12 Å); waters, the same blacklist of crystallization additives and common ions used during reference selection, and small fragments (fewer than six heavy atoms) are excluded from candidacy. RDKit is then used to assign each atom or functional group of the chosen ligand to one of six pharmacophoric families: hydrogen-bond donor, hydrogen-bond acceptor, aromatic, hydrophobic, cation, or anion. Features further than 12 Å from the pocket center are discarded to exclude allosteric binders and crystallographic artifacts.

Consensus clustering Clustering proceeds in two stages. First, DBSCAN (Ester et al., 1996) is run independently per pharmacophore family at $\varepsilon = 1.0$ Å on the 3D points of that family alone, so that, e.g., donor and acceptor densities never collapse into a single cluster regardless of spatial proximity. A cluster is retained only if it is supported by at least 15% of the unique structures contributing to the ensemble; the conservation score of a cluster is the fraction of unique PDB entries supporting it. Cross-family relationships between surviving clusters are handled separately during ground-truth point selection.

Ground-truth point selection Surviving clusters are post-processed to yield a physically valid, non-redundant ground-truth set: (i) clusters within 2 Å of any heavy atom of the reference receptor are discarded as sterically clashing with the reference protein conformation given to the agent; (ii) clusters of different types (e.g. donor and cation arising from a basic amine, or donor and acceptor from a hydroxyl) whose centroids lie within 2 Å are merged into a single point with a set-valued

type label, reflecting and allowing for chemical ambiguity in the agent output. The full set of $N \geq 3$ surviving consensus points constitutes the ground truth.

Agent inputs and outputs The agent is provided `receptor.pdb` (the reference protein with all ligands and solvent removed) the pocket center coordinates and the 12 Å pocket radius where all interaction points are within. The agent must emit `solution.csv` with exactly three rows of the form `x, y, z, type`, where `type` is drawn from {`Donor`, `Acceptor`, `Aromatic`, `Hydrophobic`, `Cation`, `Anion`}.

Evaluation Predicted points are matched against the N ground-truth points by optimal bipartite assignment (Kuhn, 1955) on Euclidean distance, restricted to type-compatible pairs: a prediction is type-compatible with a ground-truth point if its label is in the ground-truth label set, with an additional aromatic/hydrophobic equivalence at scoring time, reflecting that aromatic ring features and hydrophobic features are partially co-tagged by RDKit and that the ground-truth construction may assign only one of the two labels at a position where both are chemically reasonable. This choice was made out of evaluation leniency. A match is counted if the assigned distance is below 2.5 Å. The task score is (number of matched predictions) / 3, and the task is considered solved only when all three predictions match (score $= 1.0$).

Design rationale The local-pocket alignment, conservation threshold, and steric clash filter were designed to ensure that the targets are simultaneously experimentally grounded, chemotype-agnostic, empirically solvable, and medicinally useful in the specific receptor conformation handed to the agent.

B.3 Scaffold Hopping

Each instance of this task type asks the agent to propose a novel small molecule that binds the same protein pocket as a given reference ligand and reproduces its protein–ligand interaction pattern, while presenting a chemically distinct scaffold. Tasks are generated procedurally from the RCSB PDB and ChEMBL, and every task is constructed around an explicit *witness* molecule that is known and evaluated a priori to satisfy the evaluation criteria – thus guaranteeing the solvability of the task instance.

Target and canonical structure selection Targets are sourced from a curated pool of ∼300 `(UniProt, PDB, ligand)` triplets spanning diverse protein families, grouped by UniProt identifier. For each UniProt, the PDB entry with the best X-ray resolution is chosen as the canonical receptor; its FASTA sequence and a single fixed pocket definition are reused throughout downstream steps.

Pocket definition The binding pocket is defined as the set of residues with any heavy atom within 6 Å of the canonical ligand. This pocket, expressed as `(chain, SEQRES position)` pairs, is held fixed for all downstream Boltz2 cofolding runs and is also handed to the agent as part of the task specification.

Candidate ligand collection For each target we assemble a candidate ligand pool from two sources: (i) all ligands co-crystallized with the same UniProt across the PDB, with bond orders corrected via Chemical Component Dictionary (CCD) templates and deduplicated by canonical SMILES; and (ii) ChEMBL actives against the target with reported $IC_{50}/K_i/K_d$ below 10 μM, pre-filtered to remove unparseable structures, molecules above 1,000 Da, and metal-containing species.

Scaffold-diversity pre-filter Bemis-Murcko scaffolds are extracted for every candidate, and for each unordered ligand pair we compute the Maximum Common Substructure (MCS) of the two scaffolds (Bemis & Murcko, 1996). A pair is retained only if the MCS covers less than 65% of the heavy atoms of the larger scaffold; the symmetric denominator ensures the filter does not depend on the eventual reference/witness role assignment.

Crystal-PLIP pre-ranking For pairs in which both members appear as PDB co-crystals, each source structure is superimposed onto the canonical receptor using C_α atoms of pocket residues within 12 Å, and a protein-ligand interaction fingerprint is extracted with PLIP (Salentin et al., 2015). Pairs are then ordered by descending crystal interaction-fingerprint similarity, with PDB-PDB pairs prioritized over pairs involving ChEMBL-only ligands. This ranking concentrates downstream compute on pairs that are most likely to clear the interaction-similarity gate during the co-folding screen.

Co-folding screen For each target, the top-ranked unique candidate ligands are co-folded with the canonical receptor using Boltz2 (Passaro et al., 2025), constrained to the fixed pocket definition with 10 diffusion samples per ligand. From the resulting ensemble of 10 conformations, we extract: (i) the Boltz2-predicted binding probability across samples; (ii) per-sample PLIP interaction fingerprints, where each interaction is coarsened to a `(chain, residue, broad type)` tuple over six interaction families (HBond, Aromatic, Ionic, Halogen, Metal, Hydrophobic); and (iii) a consensus interaction fingerprint defined as the set of tuples present in at least 60% of the diffusion samples. Any ligand whose predicted binding probability falls below 0.75 is discarded. The consensus interaction fingerprint is required during evaluation to offset variance induced by Boltz2-predicted binding poses.

Witness validation and role assignment A pair is promoted to a task only if both members survive the binding gate *and* the set Tanimoto between their consensus interaction fingerprints exceeds 0.75. Within a surviving pair, the ligand with the higher predicted binding probability is assigned as the *reference* (the molecule provided to the agent to hop away from) and the other becomes the witness (the hidden molecule whose existence guarantees the solvability of the task instance). We retain at most three tasks per UniProt target and require distinct reference molecules across tasks of the same target, preventing one well-behaved reference from dominating the benchmark.

Agent inputs and outputs The agent is given: `complex.pdb` (the Boltz2-predicted reference complex, with the ligand on chain L), `reference.sdf` (the reference ligand conformer extracted from the Boltz2 complex), `protein.fasta` (the canonical protein sequence), and the target ligand identifier and pocket contact residue list used during the task generation process. The agent is required to emit `solution.smi` containing a single SMILES string. The witness molecule is never exposed to the agent.

Evaluation A submission must clear five gates: (1) the SMILES is parseable by RDKit; (2) the Morgan fingerprint Tanimoto (radius $= 2$, 2048 bits) between the agent's molecule and the reference is below 0.5; (3) the Bemis–Murcko scaffold MCS, normalized by the heavy-atom count of the larger of the two scaffolds, is below 0.65; (4) the agent's molecule, co-folded under the same Boltz protocol used during generation (10 diffusion samples, identical pocket constraint), produces a consensus interaction fingerprint whose set Tanimoto with the reference's consensus fingerprint exceeds 0.75; and (5) the predicted binding probability of the agent's molecule across the same diffusion samples exceeds 0.7. Gates 1, 2, and 3 enforce 2D novelty; gates 4 and 5 enforce 3D mimicry, ensuring the scaffold hopped molecule adheres to the same pharmacophore, and binding competence.

Design rationale Every task is anchored to a concrete witness molecule that has already cleared the same evaluation pipeline the agent will be scored against, ensuring the existence of at least one solution for the task instance. The 2D-dissimilarity and 3D-interaction-similarity criteria rule evaluates for a true scaffold hopped molecule and that the submitted molecule matches the binding mode of the original reference molecule. High predicted binding probability of the submitted molecule is also enforced. We score interaction similarity against a consensus fingerprint over the 10-sample diffusion ensemble rather than against the single most confident pose to ensure robustness against variance in the Boltz2 poses. Requiring a tuple to appear in at least 60% of samples discards interactions that are artifacts of any one diffusion draw and retains only those the model is confident enough to recover under structural perturbation.

B.4 Lead Optimization

Each instance of this task type asks the agent to modify a known active molecule so that it improves a specified set of pharmacological optimization objectives, holds a separate set of properties within tolerance, and clears a fixed panel of hard drug-likeness constraints. As in the Scaffold Hopping task type, every task instance is constructed around an explicit witness molecule that has already been verified to satisfy every gate the agent will be scored against, guaranteeing the solvability of the task instance.

Target and pocket selection Targets are drawn from a curated set of UniProt entries with high-resolution co-crystal coverage. For each target, a single representative PDB is chosen; its FASTA sequence and the residues within 6 Å of the crystal ligand define a fixed pocket that is reused for all candidate co-folding within the target.

Table 6: List of hard-constraints for molecules evaluated in Lead Optimization and Fragment Assembly tasks.

Category	Constraint
SMILES validity	Valid SMILES only
Molecular weight	MW < 600 Da
Lipophilicity	$\log P \in [-1, 5]$
Topological polar surface area	TPSA < 140 Å^2
H-bond donors	≤ 5
H-bond acceptors	≤ 10
Rotatable bonds	≤ 10
Formal charge	$\in [-2, +2]$
Synthetic accessibility	SA score < 4.5 (Ertl & Schuffenhauer, 2009)
Structural alerts	No PAINS (Baell, 2016) or Brenk/NIH alerts (Brenk et al., 2008; Limban et al., 2018)

Table 7: List of ADMET properties used as optimization objectives and hold-constant constraints.

Property
hERG inhibition
Blood-brain barrier permeability
Aqueous solubility
CYP3A4 inhibition
Hepatocyte clearance
Caco-2 permeability
Plasma protein binding
Ames mutagenicity

Candidate pool collection For each target we aggregate experimentally reported actives from ChEMBL, BindingDB, and PubChem, deduplicated by canonical SMILES. The pool is then filtered through the same hard-constraint panel that will be applied to the agent's submission at evaluation time. The hard-constraint panel is found in Table 6.

ADMET annotation Every surviving molecule is scored on eight ADMET properties (hERG inhibition, blood-brain-barrier permeability, aqueous solubility, CYP3A4 inhibition, hepatocyte clearance, Caco-2 permeability, plasma protein binding, Ames mutagenicity) via the ADMET-AI ensemble (Swanson et al., 2024) trained on the corresponding TDC datasets (Huang et al., 2021).

Co-folding screen Each filtered candidate is co-folded against the canonical receptor with Boltz2 (Passaro et al., 2025), constrained to the fixed pocket via fixed pocket contacts with 10 diffusion samples. We retain the predicted binding probability and predicted $\log_{10}(\mathrm{IC}_{50})$ across samples, and discard any molecule whose predicted binding probability falls below 0.75. The pool of molecules surviving every filter constitutes the per-target validated pool from which both reference and witness are drawn.

Pair finding and task generation Within each target's validated pool, we compute pairwise Morgan fingerprint Tanimoto (radius $= 2$, 2048 bits) and consider only pairs with similarity ≥ 0.7.

For each filtered pair of molecules within the validated pool, we derive the optimization objectives and hold-constant set directly from the measured property differences between the two molecules. A property is eligible to be an optimization objective if the witness improves over the reference by at least a property-specific threshold. A property is eligible to be a hold-constant constraint only if the witness's value already lies within the property's tolerance band relative to the reference.

Up to five objectives and four hold-constant properties are selected greedily. The direction of BBB optimization is inferred from the witness properties rather than imposed externally. Furthermore, for the task instances without binding affinity as optimization objectives or hold-constant constraints (only a handful in SMDD-Bench), Boltz2 calculated binding probability is enforced as a hard constraint for the agent's submitted molecule to prevent the agent from throwing away binding in the pursuit of optimizing ADMET properties.

Agent inputs and outputs The agent is given: `complex.pdb` (the Boltz2-predicted reference complex, with the ligand on chain L), `reference.sdf` (the reference ligand conformer extracted

from the complex), `protein.fasta` (the canonical sequence), and the pocket residue list, the target ligand identifier, the list of optimization objectives with directions and thresholds, the list of hold-constant properties with tolerance bands, the full hard-constraint panel, the BBB direction, and the baseline property values measured on the reference. The agent is required to emit `solution.smi` containing a single SMILES string. The witness molecule is never exposed to the agent.

Evaluation A submission must clear five gates: (1) the SMILES is valid and sanitizable; (2) every hard constraint in the panel is satisfied, including a Tanimoto similarity to the reference ≥ 0.7 that anchors the submission to the same lead series; (3) every hold-constant property lies within its tolerance band relative to the reference baseline; (4) every optimization objective improves over the reference by at least its required threshold; and (5) Boltz2 co-folding of the agent's molecule under the same protocol used during generation produces a predicted binding probability above 0.7 together with the affinity checks implied by the binding-affinity objective or hold-constant constraint. ADMET properties are recomputed from the agent's SMILES with the same ADMET-AI ensemble used in generation.

Design rationale Deriving objectives and hold-constant sets from the actual reference–witness property delta, rather than imposing a fixed objective list per target, produces tasks that provably solvable. The high-Tanimoto pairing constraint enforces that the task captures local lead-series chemistry rather than de novo design, which is the regime in which the multi-objective trade-offs are most relevant. Because the entire hard-constraint panel and the ADMET model are applied identically at generation and at evaluation, the agent is scored under the exact filter that defined task feasibility, and any submission that clears the gates is at least as good as a known real molecule on every property the task names.

We investigate the indicators of the difficulty of the Lead Optimization tasks in Section D.1.

B.5 Fragment Assembly

Each instance of this task type asks the agent to design a single drug-like molecule that incorporates one or two given 3D molecular fragments at their specified poses inside a protein binding pocket, covalently links them when more than one is provided, and binds the target well. As in the previous two task types, every task instance is constructed around an explicit witness molecule – here, the known active from which the fragments were carved – that has already been verified to satisfy every gate the agent will be scored against, guaranteeing the solvability of the task instance.

Target and active selection Targets are sourced from UniProt entries that have at least two PDB entries containing distinct, non-buffer ligands of at least 10 heavy atoms, after filtering crystallization additives. For each (UniProt, PDB, ligand) triplet, the ligand SMILES is reconstructed via Chemical Component Dictionary templates and the binding pocket is defined as the protein residues with any heavy atom within 6 Å of the ligand.

Witness co-folding Each candidate active is co-folded against its source protein with Boltz2 (Passaro et al., 2025) under pocket constraints derived from the crystal pose, with the affinity head enabled and 10 diffusion samples. Only chains that participate in pocket contacts are retained for the co-fold; co-ligands and unrelated chains are excluded. Actives are kept only if their predicted binding probability exceeds 0.75. The resulting predicted $\log_{10}(\mathrm{IC}_{50})$ is recorded as the witness affinity baseline against which the agent's submission will be scored.

Fragment generation Surviving actives are decomposed via BRICS (Degen et al., 2008) into chemically reasonable fragments at retrosynthetically meaningful bonds. Each fragment is retained only if it (i) contains at least 6 heavy atoms after dummy-atom removal, (ii) makes at least one PLIP (Salentin et al., 2015) protein-ligand interaction with the receptor when scored on the witness's docked pose, and (iii) lies within 4 Å of the protein. The first criterion ensures the fragment is large enough to be a meaningful pharmacophore rather than a methyl-scale stub; the second and third together guarantee that the fragment is genuinely engaged with the pocket rather than solvent-exposed.

Pair sampling for two-fragment tasks When a task instance is to contain two fragments, the pair is drawn from a single source active and is required to have a closest-heavy-atom distance of at least 6 Å, this prevents the fragments from being spatially close enough to admit a short or simple linker.

Agent inputs and outputs The agent is given: `pocket.pdb` (the witness's Boltz2-predicted protein structure with the witness ligand stripped), one `fragment_i.sdf` per fragment (the 3D pose of each fragment as it sat in the witness's Boltz2 docked complex), the binding affinity baseline target for the submited molecule, and the full hard-constraint panel also used in the Lead Optimization task (see Table 6). The agent is required to emit `solution.smi` containing a single SMILES string. The witness is never exposed to the agent.

Evaluation A submission must clear five gates: (1) the SMILES is valid; (2) every fragment SMARTS pattern is a substructure of the agent's molecule; (3) the fragments admit a disjoint substructure assignment (no shared atoms across fragments) and are pairwise connected by a covalent path in the agent's molecule; (4) the agent's molecule clears the same hard-constraint panel specified in Table 6; and (5) Boltz2 co-folding of the agent's molecule against the source receptor under the same pocket constraint and 10 diffusion samples produces (a) a predicted binding probability above 0.7, (b) a predicted $\log_{10}(\text{IC}_{50})$ no more than 0.1 log units worse than the witness baseline, and (c) a per-fragment 3D pose RMSD below 2.0 Å. RMSD is computed by SMARTS-matching each fragment against the docked agent ligand, extracting the 3D coordinates of the matched atoms, aligning the docked complex to `pocket.pdb` by Kabsch superposition on the C_α atoms of the pocket residues, and selecting the best-aligned diffusion sample for the comparison.

Design rationale The single-active witness construction guarantees that for every task there exists at least one molecule – the original active itself – that contains all fragments at exactly the required poses, links them through known chemistry, and already satisfies the binding and drug-likeness panels under the same Boltz2 protocol used at evaluation time. The 3D pose RMSD threshold enforces the docked geometry to recapitulate the fragment poses given as input, which forces the linker chemistry to be physically compatible with the relative orientation of the fragments rather than just being topologically valid or bind well.

C Agent Setup

To isolate model capability from harness engineering, all baseline runs use a deliberately minimalist ReAct loop. The agent sees only the obfuscated task description and the tool schemas. All tools run in an isolate docker container with an isolated filesystem. The standard agent configuration can be found in Table 8.

Tools. The agent is provided four tools: (i) `python(code)`: executes Python in the sandbox with RDKit, OpenBabel, PLIP, BioPython, NumPy, SciPy, and scikit-learn preinstalled; (ii) `predict_boltz(smiles, protein_sequences, ...)`: runs Boltz-2 co-folding with affinity prediction; (iii) `predict_admet(smiles)`: runs ADMET-AI property predictors and returns all the values of all properties of the molecule in a single call; The Boltz/ADMET oracles live in an isolated venv that the agent's `python` tool cannot import, forcing all oracle calls through the budgeted interface.

Sandbox. Each task runs in a fresh `smdd-agent` container on an internal Docker network with no external routing; the only reachable endpoint is `api.colabfold.com:443` via an nginx TCP-passthrough sidecar (required by Boltz's MSA server). Task files (e.g. `complex.pdb`, `reference.sdf`) are obfuscated to remove target names, PDB codes, and ligand three letter codes to suppress data-leakage shortcuts via memorized identifiers.

C.1 Selecting Oracle Calls

By default, across all evaluations, we set the total number of available Boltz2 calls to 8 and the number of ADMET-AI calls to 15. Note that a single ADMET-AI call returns all 8 ADMET properties for the inputted SMILES string. The complete reasoning behind only providing the LLM with a limited number of oracle calls is outlined in Section 6. In short, our goal is to treat the oracle calls as though they were ground truth, real-world wet lab experiments, hence the limited number of oracle calls. Ultimately, SMDD-Bench can be considered a testbed for developing the right training strategies and agent harnesses for SMDD agents – even though agents trained on SMDD-Bench adhere to the Boltz2/ADMET-AI distribution of binding affinities, structures, and properties. Afterwards, the agents can be trained using those same strategies with a wet-lab-in-the-loop – where the oracle calls

Table 8: Baseline agent harness configuration. All harness values are constants across tasks and models. Boltz diffusion samples was set to 10 to mitigate stochasticity in Boltz outputs. We find empirically find settings above 3 diffusion samples to sufficiently reduce stochasticity across multiple runs.

Parameter	Value	Notes
Sampling		
Temperature	1.0	—
Sampling seed	42	—
Max ReAct turns	100	one LLM call per turn
Max cumulative tokens	10,000,000	input + output, summed across turns
Max consecutive refusals	15	terminates run if exceeded
Oracle budgets (per task)		
Boltz-2 calls	8	shared across rollout
ADMET-AI calls	15	shared across rollout
Boltz diffusion samples	10	matches evaluator setting

are directly made to an automated wet lab, performing the synthesis and then the pertinent assays completely autonomously.

The amount of Boltz2 and ADMET-AI calls must be tight enough to reward oracle-efficient reasoning instead of brute-force enumeration but generous enough such that a competent agent is not bottlenecked by oracle access.

In order to determine a standard number of oracle calls, we performed a sweep across oracle budget choices using a small calibration subset. We constructed the calibration split of 25 tasks drawn proportionally from each task type, with a minimum of one task per type (each task yielding $\max(1, \lfloor 25 \cdot n_t/N \rfloor)$, where n_t is the number of tasks of type t and N the total benchmark size). The Lead Optimization task instances were sampled stratified by problem complexity, defined as the sum of the number of optimization objectives and the number of hold-constant constraint, such that the calibration set spans the full difficulty range of the task type.

We swept each oracle independently while pinning the other to a generous ceiling:

- **Boltz sweep:** ADMET budget fixed at 35; Boltz budget varied over $\{5, 8, 12\}$.
- **ADMET sweep:** Boltz budget fixed at 12; ADMET budget varied over $\{15, 25, 35\}$.

The shared end-point (boltz $= 12$, admet $= 35$) was reused, giving five distinct (B, A) configurations in total. All other agent-harness settings were held at their defaults; the agent model used for calibration was `GPT5.4`.

For each sweep, we computed the per-type pass rate at every budget level and selected the "knee" of the plot, where the increase in performance began to saturate as the number of oracle calls increased.

D Additional Agent Results

Table 9 shows the cost breakdown of each of the LLMs evaluated on the full SMDD-Bench benchmark while Table 10 shows the average amount of wall-clock time required for task and model. Each task instance was run on either an A800 or A6000 GPU. Table 11 investigates the rate at which each LLM agent was able to recover exactly the witness that was found during task generatio out of all successful instances. Figure 8 and Figure 9 depict the number of task instances each LLM can uniquely get correct as well as the number of task instances all LLMs get correct for both SMDD-Bench and SMDD-Bench Lite respectively. Table 12 shows the number of Interaction Point Discovery task instances correctly discovering 0, 1, 2, or 3 of the most conserved interaction points. Table 13 depicts

D.1 Indicators of the Difficulty for Lead Optimization Tasks

We find that the number of optimization objectives and hold constant constraints are not indicative of the difficulty of a Lead Optimization task instance, as seen in Figure 10, where the difficulty of a task

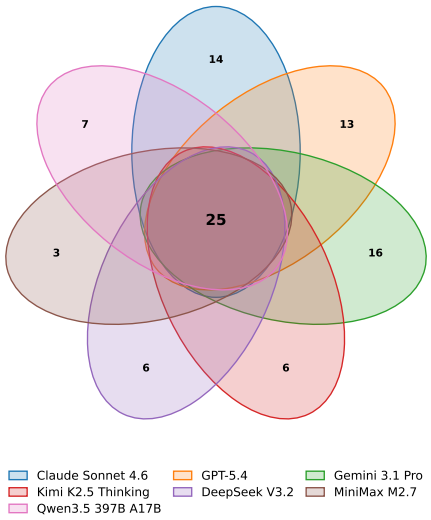

Figure 8: The number of task instances that each LLM agent gets correct as well as the number of task instances that each LLM **uniquely** gets correct on the full SMDD-Bench dataset.

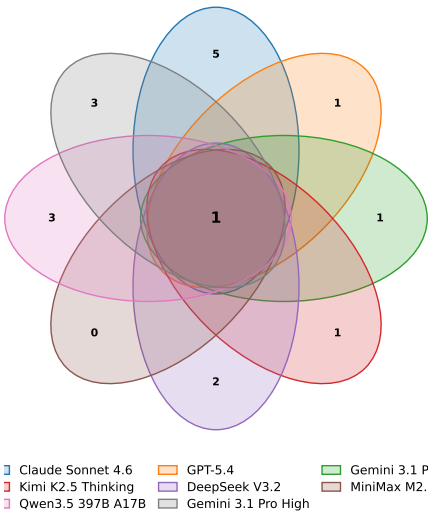

Figure 9: The number of task instances that each LLM agent gets correct as well as the number of task instances that each LLM **uniquely** gets correct on the full SMDD-Bench Lite dataset.

Table 9: Cost breakdown per instance for Task Types 1 through 5 and Overall averages across 7 frontier LLMs.

Model	Average Cost ($/task instance)					Overall
	2D Pharmacophore ID	Interaction Point Discovery	Scaffold Hopping	Lead Optimization	Fragment Assembly	
Claude Sonnet 4.6	1.09	0.30	0.92	1.01	3.85	1.31
GPT-5.4	0.61	0.72	0.62	0.72	1.31	0.78
Gemini 3.1 Pro	0.39	0.16	0.31	0.62	1.24	0.63
Kimi K2.5 Thinking	0.41	0.09	**0.18**	0.43	0.53	0.40
DeepSeek V3.2	**0.21**	0.09	0.29	0.47	0.57	0.43
MiniMax M2.7	0.41	**0.07**	0.31	**0.36**	**0.46**	**0.36**
Qwen3.5 397B A17B	1.09	0.15	0.63	0.78	0.81	0.75

Table 10: Average wall-clock time (minutes) per task, split by success status.

Model	Status	Average Time (minutes)					
		2D Pharmacophore ID	Interaction Point Discovery	Scaffold Hopping	Lead Optimization	Fragment Assembly	Overall
Claude Sonnet 4.6	Passed	8.1	–	9.3	14.9	–	14.6
	Failed	12.1	2.9	18.9	27.9	58.7	29.5
GPT-5.4	Passed	2.6	–	14.4	18.4	41.3	18.2
	Failed	5.2	5.7	27.4	28.4	39.9	26.9
Gemini 3.1 Pro	Passed	3.1	2.0	–	12.4	77.4	12.4
	Failed	3.7	1.9	13.5	25.9	41.0	23.4
Kimi K2.5 Thinking	Passed	8.0	–	4.4	21.9	–	21.5
	Failed	6.1	2.2	28.7	39.5	44.2	34.0
DeepSeek V3.2	Passed	10.0	–	25.9	39.0	–	38.3
	Failed	14.0	7.3	39.4	51.2	50.8	44.4
MiniMax M2.7	Passed	20.7	–	13.2	36.9	–	36.0
	Failed	27.6	8.0	54.2	54.8	49.2	49.6
Qwen3.5 397B A17B	Passed	9.0	–	4.1	16.4	–	16.2
	Failed	9.8	2.0	15.2	22.3	24.9	19.5

instance is defined by the average success rate of the 7 evaluated frontier LLM agents. This may be a byproduct of the greedy task generation algorithm used to generate Lead Optimization task instances – for every pair of molecules, all conserved and optimized properties of a molecule within the pair are used to create the task instance. Therefore, properties with synergistic effects can be simultaneously co-optimized via the same local molecular edit.

Rather than the number of optimization objectives and hold-constant constraints, we find that tasks with a large number of properties with clashing effects to be the greatest indicator for task difficulty. The incompatible, clashing property pairs can be found in Table 14 (Waring, 2010; Gleeson, 2008; Leeson & Springthorpe, 2007; Varma et al., 2009). These properties are chemically anti-correlated – standard local modifications intended to optimize for one property result in the degradation of the other. As a result, molecular modifications are typically multi-step and more complex in nature compared to task instances with fewer incompatible property pairs.

D.2 Analysis on inter-agent submission diversity

As seen in Figure 12, we find, perhaps intuitively, that an increase in the number of turns an LLM takes to solve a task is correlated with the increasing difficulty of the task (as indicated by success rate). We also observe that, as the LLM agents require more turns for more challenging tasks, the pairwise Tanimoto similarity between successful submissions of different LLMs for the same task increases while the fraction of unique molecules also increases. This is a surprising result, indicating that, for challenging tasks, the successful agents are converging onto the same narrow region in chemical space but via different approaches – leading to successful submissions with high Tanimoto similarity but also high unique fraction.

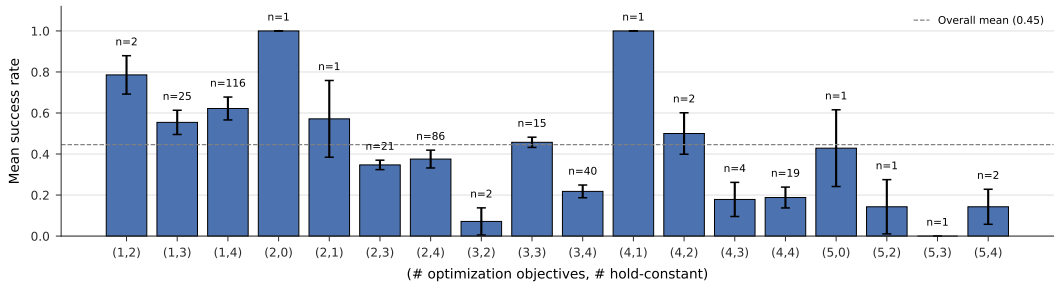

Figure 10: The average success rate across all 7 evaluated frontier LLM agents with respect to the number of optimization objectives and hold-constant constraints. There is no noticeable trend indicating the difficulty of the task instance increases as the number of objectives or constraints increases.

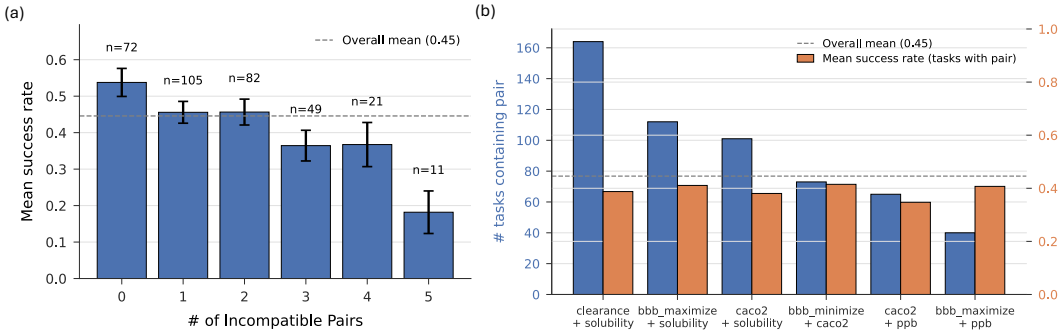

Figure 11: (a) The mean success rate across all 7 agents evalauted on SMDD-Bench plotted against the number of incompatible pairs in the optimization objectives or hold constant constraints. Pairs are not mutually exclusive, meaning one property may be incompatible with multiple other properties. (b) The mean success rate across all agents plotted against the specific incompatible pair. There is no relationship between the difficulty of the task and the specific pairing.

Table 11: Rate (%) at which the LLM agents were able to recover the witness molecule exactly. The denominator is the number of **successful** instances for the LLM on the specific task type.

Model	Scaffold Hopping	Lead Optimization	Fragment Assembly
Claude Sonnet 4.6	0.0%	16.5%	–
GPT-5.4	0.0%	21.9%	0.0%
Gemini 3.1 Pro	–	21.7%	0.0%
Kimi K2.5 Thinking	0.0%	16.9%	–
DeepSeek V3.2	0.0%	16.1%	–
MiniMax M2.7	0.0%	14.1%	–
Qwen3.5 397B A17B	0.0%	19.1%	–

Table 12: The number of task instances in the Interaction Point Discovery task type that each LLM was able to successfully pinpoint 0, 1, 2, or 3 interaction points for. Only instances where the LLM was able to find 3 points were considered a success. Note that there may be more than 3 ground truth points the LLM-discovered points are able to match to.

Model	**0/3**	**1/3**	**2/3**	**3/3**
Claude Sonnet 4.6	21	4	0	0
GPT-5.4	11	11	3	0
Gemini 3.1 Pro	14	9	1	1
Kimi K2.5 Thinking	23	2	0	0
DeepSeek V3.2	25	0	0	0
MiniMax M2.7	24	1	0	0
Qwen3.5 397B A17B	20	5	0	0

Conversely, simpler tasks lead to a low fraction of unique molecules; most likely resulting from the agents identifying and attempting the same set of straightforward strategies which passed the oracle calls on an early turn count.

E SELECTING SMDD-BENCH LITE

Running the full 502-task benchmark is expensive in wall clock time and GPU compute requirement (see Table 10). As a result, we introduce the SMDD-Bench Lite subset, a 100 task instance

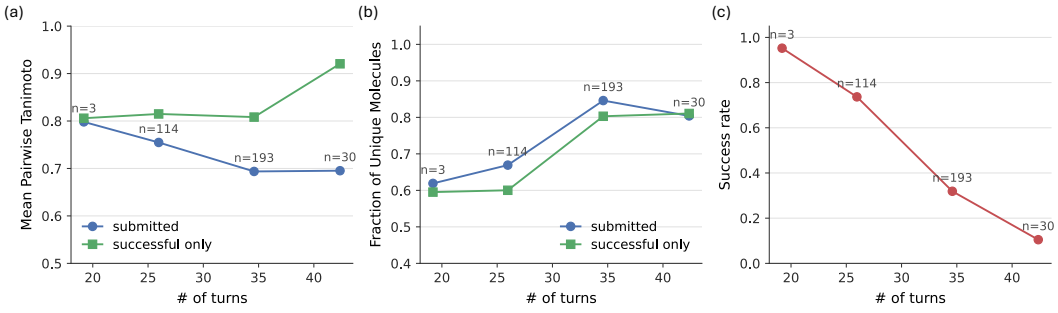

Figure 12: We investigate the relationship between the submissions of multiple different LLMs on the same task instance for the Lead Optimization task type. (a) The average pairwise Tanimoto similarity between the submissions of the 7 LLMs evaluated on the Lead Optimization task in SMDD-Bench plotting against the number of turns to solve the task instance. (b) The fraction of unique molecules submitted by the 7 LLMs with respect to number of turns. (c) The success rate of the 7 LLMs with respect to the number of turns.

Table 13: The total number of turns and tool calls, broken down by task type for the SMDD-Bench full runs. The Boltz2 oracle budget was 8 calls while the ADMET-AI oracle budget was 15 calls. Most of the turns were spent on visualization, analysis, and simulation using the python tool.

Model	Metric	2D Pharmacophore ID	Interaction Point Discovery	Scaffold Hopping	Lead Optimization	Fragment Assembly	Overall
Claude Sonnet 4.6	Total Turns	29.4	11.6	27.1	28.0	57.6	30.7
	python	28.4	10.6	23.4	20.0	50.6	23.9
	predict_boltz	0.0	0.0	3.8	3.0	7.8	3.3
	predict_admet	0.0	0.0	0.8	10.8	0.8	7.5
	other	0.0	0.0	0.0	0.0	0.0	0.0
GPT-5.4	Total Turns	13.8	13.8	15.0	14.6	21.4	15.4
	python	12.6	12.8	12.2	9.7	18.3	11.3
	predict_boltz	0.0	0.0	5.5	4.3	7.9	4.4
	predict_admet	0.0	0.0	0.8	12.7	0.5	8.7
	other	0.0	0.0	0.0	0.0	0.0	0.0
Gemini 3.1 Pro	Total Turns	14.2	7.4	11.1	19.3	31.4	19.1
	python	13.2	6.4	7.8	9.3	23.5	10.9
	predict_boltz	0.0	0.0	2.7	2.9	7.3	3.1
	predict_admet	0.0	0.0	0.1	10.7	0.8	7.4
	other	0.0	0.0	0.0	0.0	0.0	0.0
Kimi K2.5 Thinking	Total Turns	29.2	11.6	20.8	35.8	35.9	32.7
	python	28.2	10.6	13.9	17.8	24.8	18.4
	predict_boltz	0.0	0.0	5.2	5.2	7.7	5.0
	predict_admet	0.0	0.0	0.8	12.0	2.6	8.5
	other	0.0	0.0	0.0	0.0	0.0	0.0
DeepSeek V3.2	Total Turns	18.0	10.8	24.0	36.0	31.7	32.1
	python	17.0	9.8	13.6	15.7	19.5	15.7
	predict_boltz	0.0	0.0	6.8	5.3	7.3	5.2
	predict_admet	0.0	0.0	2.3	12.6	1.9	9.0
	other	0.0	0.0	0.0	0.0	0.0	0.0
MiniMax M2.7	Total Turns	40.0	14.0	39.2	43.4	48.8	42.0
	python	38.9	12.9	30.9	27.4	39.1	29.0
	predict_boltz	0.0	0.0	6.7	5.9	7.7	5.6
	predict_admet	0.0	0.0	0.3	12.8	0.5	8.8
	other	0.0	0.0	0.0	0.0	0.0	0.0
Qwen3.5 397B A17B	Total Turns	42.0	13.4	40.3	48.5	46.2	45.3
	python	37.2	11.0	31.7	26.8	36.0	28.2
	predict_boltz	0.0	0.0	6.8	5.7	7.7	5.5
	predict_admet	0.0	0.0	0.4	11.7	1.0	8.0
	other	0.0	0.0	0.0	0.0	0.0	0.0

Table 14: The incompatible property pairs defined in SMDD-Bench. These are pairs of properties which are not sampled for co-optimization targets but can be found in hold-constant constraints or have one property as an optimization objective and the other as a hold-constant constraint.

Property A (direction)	Property B (direction)	Rationale
Aqueous solubility (↑)	BBB penetration (↑)	Solubility favors polar/hydrophilic compounds; BBB penetration favors lipophilic compounds.
Plasma protein binding (↓)	BBB penetration (↑)	Low PPB requires reduced lipophilicity; BBB penetration is driven by lipophilicity.
Caco-2 permeability (↑)	Plasma protein binding (↓)	Caco-2 permeability tracks lipophilicity, which also drives plasma protein binding.
Aqueous solubility (↑)	Hepatic clearance (↓)	Increasing polarity to improve solubility tends to expose substrates to renal/metabolic clearance.
Aqueous solubility (↑)	Caco-2 permeability (↑)	Solubility favors polarity; Caco-2 permeability favors lipophilicity.
Caco-2 permeability (↑)	BBB penetration (↓)	Both are passive permeability measures; maximizing one while minimizing the other is contradictory.

representative subset of SMDD-Bench, making faster model and harness iteration faster and more unified.

Lite preserves the per-type mix of the full benchmark as seen in Table 15.

Table 15: The task breakdown for SMDD-Bench Lite compared to SMDD-Bench.

Task type	Full	Lite
2D Pharmacophore Identification	25	5
Interaction Point Discovery	25	5
Scaffold Hopping	52	10
Lead Optimisation	340	68
Fragment Assembly	60	12
Total	502	100

Selection proceeds independently within each task type, and combines stratified sampling with a greedy representativeness/diversity objective. For each task we define a stratum used as a difficulty/discrimination proxy:

- **Lead optimization**: the tuple $(n_{\text{opt}}, n_{\text{hold}})$ of the number of optimisation objectives and hold-constant properties.
- **2D Pharmacophore ID, Interaction Point Discovery, Scaffold Hopping, Fragment Assembly**: the integer success rate of the task across the seven evaluated models,

Stratum allocation. Given a per-type budget k and stratum sizes $\{n_s\}$, we allocate k slots across strata in proportion to n_s using the largest-remainder (Hamilton) rule: each stratum gets $\lfloor k \cdot n_s/N \rfloor$ slots, and the remaining slots are awarded to the strata with the largest fractional parts, breaking ties by stratum key.

Within-stratum greedy pick. Within each stratum we pick tasks one at a time. Let $\mathbf{r}_{\text{full}} \in [0,1]^6$ be the per-model pass-rate vector of the full type, and let $\mathbf{r}_S$ be the mean pass-rate vector of the currently-selected Lite tasks. At each step we pick the candidate t minimizing

$$\mathcal{L}(t) \;=\; \left\|\mathbf{r}_{S\cup\{t\}} - \mathbf{r}_{\text{full}}\right\|_1 \;+\; \lambda \cdot c(\text{target}(t)),$$

where $c(\cdot)$ counts how many already-selected Lite tasks share the same protein/target as t, and $\lambda = 0.15$. The target field is parsed from the task id. The protein-reuse term acts as a tie-breaker between near-equivalently representative candidates rather than overriding the representativeness objective; ties are broken deterministically by sorted `task_id`. Strata are processed in order of decreasing size so that the largest contributors to $\mathbf{r}_{\text{full}}$ shape the selection first.

F Selecting SMDD-Bench Diversity

During a lead optimization campaign, one potential use case involves running many SMDD agents on the same task in parallel in order to achieve a diverse spread of successful molecules. In order to test this, we develop the SMDD-Bench Diversity subset, a standardized 20 instance subset of SMDD-Bench meant for each task instance to be run multiple times.

In order to select the SMDD-Bench Diversity subset, we seek to find the most challenging consistently solvable instances. Task instances that are too simple are trivially solvable with minimal molecular edits. Task instances which are too challenging may be out of reach for the current frontier LLMs.

We first rank all Lead Optimization task instances solvable by 5/7 LLMs and rank them by turn count. We use turn count as a proxy for ranking the difficulty and chemical solution space of a task instance, following insight from Appendix D.2. We then hand-select the 20 instances with the highest average turn counts across the 7 LLMs while ensuring target diversity (no single target dominates the subset). This selection strategy ensures the tasks are at the edge of the realm of solvable tasks for current frontier LLM agents.

Note that, across the SMDD-Bench Diversity runs outlined in Table 3, none of the LLMs failed a task instance all 10 times the task instance was run.

G ENUMERATION VS. SELECTION DETAILS

A failure on SMDD-Bench can come from two qualitatively different shortcomings of the agent: (i) the agent never enumerated a passing molecule at all, or (ii) the agent enumerated a passing molecule somewhere in its trajectory but never selected it to be tested by an oracle call. The enumeration vs. selection experiments are designed to disentangle these shortcomings: for every failed task, we extract and evaluate every SMILES string enumerated by the agent but not selected for an oracle call. We define a "recovered" task to be one whose original submission fails evaluation but where at least one trajectory-resident SMILES would have passed the full evaluator.

We run the experiment for all seven evaluated models and the three generation-style task types: Scaffold Hopping, Lead Optimization, and Fragment Assembly. 2D Pharmacophore Identification and Interaction Point Discovery are excluded because their outputs are not molecules. In order to extract the SMILES string, we compact each trajectory to only the reasoning trace, LLM output, and python interpreter input before feeding the compacted trajectory to GPT5.4-mini-high.

Extraction Prompt The system prompt sent to GPT-5.4-mini is reproduced verbatim below:

```
You are a careful chemical-structure extractor.

You will receive a JSON list of trajectory events from an LLM
agent that
was working on a small-molecule drug-discovery task. Each event
has a
`turn` number, a `source` in {assistant_text, reasoning_content,
python_code}, and the raw text for that event.

Your job: extract EVERY SMILES string that appears in those events.
For
each occurrence, return:

- "turn":   the turn number (integer)
- "source": one of "assistant_text", "reasoning_content",
"python_code"
- "smiles": the raw SMILES string as it appears in the text
- "note":   a short (<=120 char) free-text note describing the
context
            (e.g. "proposed as analog A", "printed in a Python
            list of
            candidates", "mentioned while analyzing reference
            ligand").

Rules:
- Do NOT invent or canonicalize SMILES. Copy them verbatim from
the text.
- Report duplicates across turns / sources as separate
occurrences.
- Include SMILES that appear inside Python source code as string
literals.
- If a code snippet builds a SMILES via string concatenation and
you can
  statically determine the final value, include it with a note
  that says
  so. Otherwise, skip it.
- Skip protein sequences, fragment placeholders like `[*]`, and
reaction
  SMARTS.
- If you are unsure whether a token is a real SMILES, include it
---
  downstream RDKit validation will filter invalid ones.

Return a JSON object of the form:

{"occurrences": [{"turn": int, "source": str, "smiles": str,
"note":
```

```
    str}, ...]}

    Return an empty list if no SMILES are present. Do not include any
    other
    keys.
```

H NON-LLM BASELINES

For completeness, we run non-LLM baselines against each of the task types. Results can be found in Table 16.

Table 16: Success rates for non-LLM baselines on the full SMDD-Bench benchmark.

Task	Baseline / method	Exact recovery rate
Pharmacophore identification	Random forest on visible molecules	12%
Interaction point discovery	Random initialization within pocket volume	0%
Scaffold hopping	Retrieval baseline	5.8%
Lead optimization	SmileyLlama (Cavanagh et al., 2026)	1.5%
Fragment assembly	LinkLlama (Sun et al., 2026)	0%

Success rates are so low because SMDD-Bench evaluates models beyond simple lead optimization or fragment linking. For example, in the lead optimization task type, SMDD-Bench task instances consist of up to 5 optimization properties, 4 hold-constant properties, and 13 hard constraints including binding probability. Previous works focus only on a handful of constraints. Furthermore, in the fragment assembly task, strong baselines such as LinkLlama are capable of linking the fragments and pass RMSD checks on fragment poses; however, the model is not capable of accounting for the binding affinity and probability gates that SMDD-Bench requires.

For the random forest baseline, we used 2048 bit fingerprints with thresholds determined by leave-one-out cross validation. For interaction point discovery, we randomly sampled points within 9 angstroms of the pocket, accounting for steric clashes. For scaffold hopping, we performed retrieval, sorting over experimental potency, and filtering for the scaffold hopping hard constraints, over the ChEMBL database.

I EXAMPLE TASK INSTANCES, TRAJECTORIES, AND ANALYSIS

In this section, we show one successful LLM agent trajectory for each task type. Due to space considerations, each trajectory is limited to only the first 5 and last 5 turns. For all trajectories across all LLM runs, please refer to the official GitHub repository.

I.1 2D PHARMACOPHORE IDENTIFICATION

I.1.1 PASSING RUN: GPT-5.4 (MEDIUM REASONING)

Task ID: `smdd_001_Q99873_0` **Model:** GPT-5.4 (medium reasoning) **Run:** `gpt_54_med_full`

Evaluation result: PASSED.

Metric	Status	Result	Condition
Recall on Actives	**passed**	0.9880952380952381	{"gte": 0.8}
Specificity on Inactives	**passed**	0.8702290076335878	{"gte": 0.8}

Trajectory.

Showing the first 5 and last 5 of 12 agent turns; 2 intermediate turns are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. Available packages: rdkit, numpy, pandas, scipy,
↪  scikit-learn, biopython, biotite, openbabel, plip. To compute synthetic accessibility
↪  (SA) scores: `from rdkit.Chem import RDConfig; import sys, os;
↪  sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score')); import sascorer;
↪  sascorer.calculateScore(mol)`
- predict_boltz(smiles, protein_sequences, pocket_residues, co_ligand_smiles,
↪  num_samples): Run Boltz structure prediction. Returns binding probability and
↪  affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Returns hERG, BBB, solubility, CYP3A4,
↪  clearance, Caco2, PPB, AMES. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace. Outline your
↪  thought process before making tool calls.
When ready, call submit() with the path to your solution file (relative to /workspace).
```

User (turn 0)

```
## Task

You are provided with a set of active molecules and a set of inactive molecules for a
↪  protein target (Protein arginine N-methyltransferase 1) in 'actives.smi' and
↪  'inactives.smi', along with the target protein sequence in 'protein.fasta'.

Your task is to identify a 2D pharmacophore pattern that distinguishes actives from
↪  inactives.

Write a Python script 'solution.py' that defines a function `check_pharmacophore(smiles:
↪  str) -> bool`. This function should return True if the input SMILES matches your
↪  pharmacophore pattern, and False otherwise. You should rely primarily on RDKit in your
↪  solution.

Your workspace at /workspace contains the task files. Start by examining them.
```

Assistant (turn 1)

```
I'll first inspect the provided active/inactive sets and protein file, then analyze common
↪  substructures and simple 2D features in RDKit to identify a distinguishing
↪  pharmacophore pattern. After that, I'll implement `check_pharmacophore` in
↪  `solution.py` and validate it against the datasets before submitting.
```

Tool Call: python (turn 1, 0.14s)

Reasoning:

```
Inspect the workspace files and get a quick sense of the active/inactive molecule sets,
↪  including counts and example entries, to guide pharmacophore analysis.
```

Code:

```
import os, glob
from pathlib import Path
base=Path('/workspace')
print('Files:', [p.name for p in base.iterdir()])
for fn in ['actives.smi','inactives.smi','protein.fasta']:
    p=base/fn
    print('\n###',fn)
    print(p.read_text()[:2000])
```

Tool Output (turn 1)

```
Files: ['inactives.smi', 'actives.smi', '.tmp_script_0ec86da1.py', 'protein.fasta']

### actives.smi
CNCCN(C)Cc1nn2c(c1Cl1CCCCC1)CCC2
Nc1ncnc2c1ncn2[C@@H]1O[C@H](CSCC[C@H](N)C(=O)O)[C@@H](O)[C@H]1O
CNC(=N)NCCCNC(=O)Nc1ccc2cc(C(=O)OC)cc(O)c2c1
CN(CCN)Cc1c[nH]cc1-c1cccc(C(F)(F)F)c1
CCOCC1(COCC)CCC(Oc2ccccc2CN(C)C[C@@H](O)CNC)CC1
```

```
OC1CN=C2c3ccn([C@@H]4O[C@H]([C@H](O)c5ccc(Cl)cc5)[C@@H](O)[C@H]4O)c3N=CN21
CNCCNCc1c[nH]nc1-c1ccc(O[C@H]2C[C@H](OCCC3CCCCC3)C2)cc1
CNCCN(C)Cc1cn[nH]c1-c1ccc(OC(CO)C(C)C)cc1
CCOCC1(COCC)CCC(c2c(CNCCN)nn3c2CC(F)(F)C3)CC1
CC(C)Oc1ccc(-c2c[nH]cc2CNCCN)cc1


... [21 more lines elided]
```

Assistant (turn 2)

```
Next I'll compute basic RDKit descriptors and look for recurring scaffolds and
↪  substructures in the actives versus inactives. That should help me formulate a simple
↪  but selective 2D rule rather than overfitting to exact structures.
```

Tool Call: python (turn 2, 0.48s)

Reasoning:

```
Analyze active and inactive molecules with RDKit: parse molecules, compute descriptors,
↪  Murcko scaffolds, and inspect recurring functional motifs to identify a discriminative
↪  pharmacophore.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
from rdkit.Chem.Scaffolds import MurckoScaffold
from collections import Counter,defaultdict
from pathlib import Path

base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]
print('n actives',len(actives),'n inactives',len(inactives))

def summarize(smiles_list,name):
    rows=[]
    scaff=Counter()
    for smi in smiles_list:
... [25 more lines elided]
```

Tool Output (turn 2)

```
n actives 10 n inactives 10

 actives
MW           364.81
LogP           2.12
HBD            3.10
HBA            5.20
TPSA          86.46
RB             8.10
rings          2.80
aromrings      1.70
FCSP3          0.55
dtype: float64
Top scaffolds:
2 c1nn2c(c1C1CCCCC1)CCC2
... [30 more lines elided]
```

Assistant (turn 3)

```
The actives appear to include a dominant chemotype with an aminoethyl/diamino side chain
↪  attached to an N-containing 5-membered heteroaromatic ring, plus a few distinct
↪  substrate/cofactor-like mimics. I'll now quantify candidate SMARTS motifs across both
↪  classes to see which combinations are most discriminative.
```

Tool Call: python (turn 3, 0.29s)

Reasoning:

```
Test candidate SMARTS motifs and motif combinations across actives and inactives to
↪  identify a discriminative pharmacophore-style rule.
```

Code:

```
from rdkit import Chem
from pathlib import Path
from itertools import combinations
base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]
acts=[Chem.MolFromSmiles(s) for s in actives]
inas=[Chem.MolFromSmiles(s) for s in inactives]

patterns={
'diamine':'[NX3;H0,H1,H2][CX4][CX4][NX3;H0,H1,H2]',
'aminomethyl_diamine':'[CH2][NX3;H0,H1,H2][CX4][CX4][NX3;H0,H1,H2]',
'basic_N_count_2':'[N;!$(N=*);!$(N#*)]',
'5het_NH':'[nH]1cccc1',
'pyrazole_NH':'[nH]1ncc[c,n]1',
... [22 more lines elided]
```

Tool Output (turn 3)

```
diamine                   act 6/10 inact 1/10
aminomethyl_diamine       act 6/10 inact 1/10
basic_N_count_2           act 10/10 inact 8/10
5het_NH                   act 2/10 inact 1/10
pyrazole_NH               act 2/10 inact 1/10
azole_NH_generic          act 4/10 inact 2/10
imidazole_like            act 0/10 inact 0/10
pyrazole_like             act 2/10 inact 1/10
indole_like               act 4/10 inact 2/10
aryl_heteroaryl           act 4/10 inact 1/10
purine                    act 0/10 inact 0/10
ribose                    act 0/10 inact 0/10
guanidine                 act 1/10 inact 0/10
phenoxy                   act 0/10 inact 1/10
heteroaryl_CH2N           act 6/10 inact 0/10
... [3 more lines elided]
```

Assistant (turn 4)

(no assistant text emitted; tool call only)

Tool Call: python (turn 4, 0.26s)

Reasoning:

```
Identify which specific molecules match the most promising SMARTS motif so I can
↪ understand its scope and decide whether to extend it with additional pharmacophore
↪ branches.
```

Code:

```
from rdkit import Chem
from pathlib import Path
base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]
q=Chem.MolFromSmarts('[n,c]1[n,c][n,c][c,n][c,n]1[CH2][N][CH2][CH2][N]')
print('Actives matching:')
for i,s in enumerate(actives,1):
    m=Chem.MolFromSmiles(s)
    if m.HasSubstructMatch(q):
        print(i,s)
print('Inactives matching:')
for i,s in enumerate(inactives,1):
    m=Chem.MolFromSmiles(s)
    if m.HasSubstructMatch(q):
... [1 more line elided]
```

Tool Output (turn 4)

```
Actives matching:
1 CNCCN(C)Cc1nn2c(c1C1CCCCC1)CCC2
4 CN(CCN)Cc1c[nH]cc1-c1cccc(C(F)(F)F)c1
7 CNCCNCc1c[nH]nc1-c1ccc(O[C@H]2C[C@H](OCCC3CCCCC3)C2)cc1
```

```
8 CNCCN(C)Cc1cn[nH]c1-c1ccc(OC(CO)C(C)C)cc1
9 CCOCC1(COCC)CCC(c2c(CNCCN)nn3c2CC(F)(F)C3)CC1
10 CC(C)Oc1ccc(-c2c[nH]cc2CNCCN)cc1
Inactives matching:
```

Assistant (turn 5)

```
The dominant active motif is already fairly clear. I'm going to inspect the remaining
↪  outlier actives individually so I can decide whether to keep one core pattern or add a
↪  couple of chemically sensible auxiliary branches for the substrate/cofactor-like
↪  chemotypes.
```

Tool Call: python (turn 5, 0.26s)

Reasoning:

```
Inspect unmatched actives and compare to potentially confusing inactives, to define a
↪  small set of auxiliary pharmacophore branches if necessary.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import rdMolDescriptors
from pathlib import Path
base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]

main=Chem.MolFromSmarts('[n,c]1[n,c][n,c][c,n][c,n]1[CH2][N][CH2][CH2][N]')
print('Unmatched actives:')
for i,s in enumerate(actives,1):
    m=Chem.MolFromSmiles(s)
    if not m.HasSubstructMatch(main):
        print('\nACTIVE',i,s)
        print(' formula',rdMolDescriptors.CalcMolFormula(m),'rings',rdMolDescriptors.Calc
        ↪  NumRings(m),'HBD',rdMolDescriptors.CalcNumHBD(m),'HBA',rdMolDescriptors.CalcN
        ↪  umHBA(m),'TPSA',rdMolDescriptors.CalcTPSA(m))
        print(' ring info',m.GetRingInfo().AtomRings())
... [6 more lines elided]
```

Tool Output (turn 5)

```
Unmatched actives:

ACTIVE 2 Nc1ncnc2c1ncn2[C@@H]1O[C@H](CSCC[C@H](N)C(=O)O)[C@@H](O)[C@H]1O
 formula C14H20N6O5S rings 3 HBD 5 HBA 10 TPSA 182.62999999999997
 ring info ((1, 6, 5, 4, 3, 2), (7, 6, 5, 9, 8), (11, 10, 24, 22, 12))

ACTIVE 3 CNC(=N)NCCCNC(=O)Nc1ccc2cc(C(=O)OC)cc(O)c2c1
 formula C18H23N5O4 rings 2 HBD 6 HBA 5 TPSA 135.57000000000002
 ring info ((12, 26, 25, 15, 14, 13), (16, 17, 22, 23, 25, 15))

ACTIVE 5 CCOCC1(COCC)CCC(Oc2ccccc2CN(C)C[C@@H](O)CNC)CC1
 formula C24H42N2O4 rings 2 HBD 2 HBA 6 TPSA 63.19
 ring info ((4, 29, 28, 11, 10, 9), (14, 15, 16, 17, 18, 13))

ACTIVE 6 OC1CN=C2c3ccn([C@@H]4O[C@H]([C@H](O)c5ccc(Cl)cc5)[C@@H](O)[C@H]4O)c3N=CN21
... [19 more lines elided]
```

Elision

Turns 6-7 (2 agent turns) omitted.

Assistant (turn 8)

(no assistant text emitted; tool call only)

Tool Call: python (turn 8, 0.26s)

Reasoning:

```
Refine a generic ribose-like SMARTS that captures the nucleoside-like actives while
↪  allowing me to combine it with other features to exclude the SAM-like inactive.
```

Code:

```
from rdkit import Chem
from pathlib import Path
base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]
smarts_list=[
    '[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1',
    '[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[O,N]',
    '[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[*]',
    '[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[*]',
    '[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[O,N,#6]',
]
for sm in smarts_list:
    q=Chem.MolFromSmarts(sm)
    print('\nSMARTS',sm)
... [2 more lines elided]
```

Tool Output (turn 8)

```
SMARTS [O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1
act [2, 6]
inact [9]

SMARTS [O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[O,N]
act []
inact []

SMARTS [O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[*]
act [2, 6]
inact [9]

SMARTS [O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[*]
act [2, 6]
... [5 more lines elided]
```

Assistant (turn 9)

(no assistant text emitted; tool call only)

Tool Call: python (turn 9, 0.27s)

Reasoning:

```
Evaluate a combined multi-branch pharmacophore logic based on the dominant
↪  heteroaryl-diamine motif plus auxiliary branches for nucleoside-like, guanidinoamide,
↪  and aminoalomatic-aminoalcohol chemotypes.
```

Code:

```
from rdkit import Chem
from pathlib import Path
base=Path('/workspace')
actives=[line.strip().split()[0] for line in open(base/'actives.smi') if line.strip()]
inactives=[line.strip().split()[0] for line in open(base/'inactives.smi') if line.strip()]

A=Chem.MolFromSmarts('[n,c]1[n,c][n,c][c,n][c,n]1[CH2][N][CH2][CH2][N]')
B=Chem.MolFromSmarts('[O;r5]1[C;r5][C;r5]([O,N])[C;r5]([O,N])[C;r5]1[*]')
C=Chem.MolFromSmarts('NC(=N)NCCCNC(=O)N')
D=Chem.MolFromSmarts('a[CH2][N]([#6])[CH2][CH]([OH])[CH2][N]')
SP=Chem.MolFromSmarts('[S+]')

def rule(smi):
    m=Chem.MolFromSmiles(smi)
    return m.HasSubstructMatch(A) or (m.HasSubstructMatch(B) and not
    ↪  m.HasSubstructMatch(SP)) or m.HasSubstructMatch(C) or m.HasSubstructMatch(D)
... [8 more lines elided]
```

Tool Output (turn 9)

```
act 10 [True, True, True, True, True, True, True, True, True, True]
act 1 CNCCN(C)Cc1nn2c(c1C1CCCCC1)CCC2
act 2 Nc1ncnc2c1ncn2[C@@H]1O[C@H](CSCC[C@H](N)C(=O)O)[C@@H](O)[C@H]1O
act 3 CNC(=N)NCCCNC(=O)Nc1ccc2cc(C(=O)OC)cc(O)c2c1
act 4 CN(CCN)Cc1c[nH]cc1-c1cccc(C(F)(F)F)c1
```

```
act 5 CCOCC1(COCC)CCC(Oc2ccccc2CN(C)C[C@@H](O)CNC)CC1
act 6 OC1CN=C2c3ccn([C@@H]4O[C@H]([C@H](O)c5ccc(Cl)cc5)[C@@H](O)[C@H]4O)c3N=CN21
act 7 CNCCNCc1c[nH]nc1-c1ccc(O[C@H]2C[C@H](OCCC3CCCCC3)C2)cc1
act 8 CNCCN(C)Cc1cn[nH]c1-c1ccc(OC(CO)C(C)C)cc1
act 9 CCOCC1(COCC)CCC(c2c(CNCCN)nn3c2CC(F)(F)C3)CC1
act 10 CC(C)Oc1ccc(-c2c[nH]cc2CNCCN)cc1

inact 0 [False, False, False, False, False, False, False, False, False, False]
```

Assistant (turn 10)

```
I found a compact rule set that cleanly separates the provided actives from inactives. I'm
↪  now writing `solution.py` with a single `check_pharmacophore(smiles)` function
↪  implemented in RDKit using a small number of SMARTS-based pharmacophore branches and a
↪  sulfonium exclusion for the SAM-like inactive.
```

Tool Call: python (turn 10, 0.15s)

Reasoning:

```
Write solution.py implementing the pharmacophore classifier as RDKit SMARTS-based logic,
↪  with comments and a simple optional self-test block.
```

Code:

```
from pathlib import Path
code = r'''from rdkit import Chem

# PRMT1 actives in the supplied set fall into a few recognizable 2D pharmacophore
# families. The dominant motif is an N-heteroaryl system bearing a benzylic
# aminoethyl/diamine side chain. Additional active families are nucleoside-like
# cofactor/product mimics, a guanidino-propyl-amide motif, and a benzyl tertiary
# amino-alcohol terminating in a primary amine.
#
# The function below returns True when a molecule matches any of those active-like
# pharmacophore patterns.

# 1) Dominant PRMT1 inhibitor motif:
#    N-containing 5-membered heteroaryl/heterobicyclic core attached to a
#    benzylic aminoethyl-diamine side chain.
... [78 more lines elided]
```

Tool Output (turn 10)

```
Wrote /workspace/solution.py
from rdkit import Chem

# PRMT1 actives in the supplied set fall into a few recognizable 2D pharmacophore
# families. The dominant motif is an N-heteroaryl system bearing a benzylic
# aminoethyl/diamine side chain. Additional active families are nucleoside-like
# cofactor/product mimics, a guanidino-propyl-amide motif, and a benzyl tertiary
# amino-alcohol terminating in a primary amine.
#
# The function below returns True when a molecule matches any of those active-like
# pharmacophore patterns.

# 1) Dominant PRMT1 inhibitor motif:
#    N-containing 5-membered heteroaryl/heterobicyclic core attached to a
#    benzylic aminoethyl-diamine side chain.
... [51 more lines elided]
```

Assistant (turn 11)

(no assistant text emitted; tool call only)

Tool Call: python (turn 11, 0.28s)

Reasoning:

```
Run the solution script to verify that it executes cleanly and reproduces the intended
↪  separation on the provided datasets.
```

Code:

```
import subprocess, textwrap, os, sys
```

```
res=subprocess.run([sys.executable,'solution.py'],cwd='/workspace',capture_output=True,te
↪  xt=True)
print('returncode',res.returncode)
print('stdout:\n',res.stdout)
print('stderr:\n',res.stderr)
```

Tool Output (turn 11)

```
returncode 0
stdout:
 Actives matched: 10/10
Inactives matched: 0/10

stderr:
```

Assistant (turn 12)

```
The script validates cleanly on the provided files, matching all supplied actives and
↪  rejecting all supplied inactives. I'll submit `solution.py` now.
```

Tool Call: submit (turn 12, 0.07s)

file_path: `solution.py`

Tool Output (turn 12)

```
Submission accepted: solution.py
```

Submit (turn 12)

Submitted file: `solution.py`.

Submitted artifact (`solution.py`).

Submitted artifact: `solution.py`

```
from rdkit import Chem

# PRMT1 actives in the supplied set fall into a few recognizable 2D pharmacophore
# families. The dominant motif is an N-heteroaryl system bearing a benzylic
# aminoethyl/diamine side chain. Additional active families are nucleoside-like
# cofactor/product mimics, a guanidino-propyl-amide motif, and a benzyl tertiary
# amino-alcohol terminating in a primary amine.
#
# The function below returns True when a molecule matches any of those active-like
# pharmacophore patterns.

# 1) Dominant PRMT1 inhibitor motif:
#    N-containing 5-membered heteroaryl/heterobicyclic core attached to a
#    benzylic aminoethyl-diamine side chain.
_HETEROARYL_DIAMINE = Chem.MolFromSmarts(
... [73 more lines elided]
```

I.2 Interaction Point Discovery

I.2.1 Passing run: Gemini 3.1 Pro (medium reasoning)

Task ID: `smdd_002_P10275_0` **Model:** Gemini 3.1 Pro (medium reasoning) **Run:** `gemini_31_pro_med_full`

Evaluation result: PASSED.

Metric	Status	Result	Condition
3D Pharmacophore Match	passed	1.0	{"eq": 1.0}

Trajectory.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. Available packages: rdkit, numpy, pandas, scipy,
↪  scikit-learn, biopython, biotite, openbabel, plip. To compute synthetic accessibility
↪  (SA) scores: `from rdkit.Chem import RDConfig; import sys, os;
↪  sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score')); import sascorer;
↪  sascorer.calculateScore(mol)`
- predict_boltz(smiles, protein_sequences, pocket_residues, co_ligand_smiles,
↪  num_samples): Run Boltz structure prediction. Returns binding probability and
↪  affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Returns hERG, BBB, solubility, CYP3A4,
↪  clearance, Caco2, PPB, AMES. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace. Outline your
↪  thought process before making tool calls.
When ready, call submit() with the path to your solution file (relative to /workspace).
```

User (turn 0)

```
## Task

You are given a protein receptor structure (receptor.pdb). The binding pocket center is
↪  located at coordinates (0.266524, 31.234953, 4.470429) in Angstroms with a pocket
↪  radius of 12 Angstroms. Your task is to identify the 3 most conserved pharmacophoric
↪  interaction points within this binding pocket. Each point should be specified by its
↪  3D coordinates (x, y, z) and one interaction type. The valid interaction types are:
↪  Donor, Acceptor, Aromatic, Hydrophobic, Cation, Anion. Output a CSV file titled
↪  'solution.csv' with columns x,y,z,type containing exactly 3 rows (no header).

Your workspace at /workspace contains the task files. Start by examining them.
```

Assistant (turn 1)

(no assistant text emitted; tool call only)

Tool Call: python (turn 1, 0.30s)

Reasoning:

```
Identify residues within 12 Angstroms of the pocket center.
```

Code:

```
import numpy as np
from Bio.PDB import PDBParser

center = np.array([0.266524, 31.234953, 4.470429])
radius = 12.0

parser = PDBParser(QUIET=True)
structure = parser.get_structure('receptor', 'receptor.pdb')

pocket_residues = []
```

```
for model in structure:
    for chain in model:
        for residue in chain:
            for atom in residue:
                if np.linalg.norm(atom.coord - center) <= radius:
... [6 more lines elided]
```

Tool Output (turn 1)

```
Found 66 pocket residues.
VAL (' ', 685, ' ')
ALA (' ', 687, ' ')
PHE (' ', 697, ' ')
ALA (' ', 699, ' ')
LEU (' ', 700, ' ')
LEU (' ', 701, ' ')
SER (' ', 702, ' ')
SER (' ', 703, ' ')
LEU (' ', 704, ' ')
ASN (' ', 705, ' ')
GLU (' ', 706, ' ')
LEU (' ', 707, ' ')
GLY (' ', 708, ' ')
GLU (' ', 709, ' ')
... [52 more lines elided]
```

Assistant (turn 2)

(no assistant text emitted; tool call only)

Tool Call: python (turn 2, 17.01s)

Reasoning:

```
Generate pharmacophore features for the pocket using RDKit.
```

Code:

```
import numpy as np
from rdkit import Chem
from rdkit.Chem import ChemicalFeatures
from rdkit import RDConfig
import os

fdefName = os.path.join(RDConfig.RDDataDir, 'BaseFeatures.fdef')
factory = ChemicalFeatures.BuildFeatureFactory(fdefName)

mol = Chem.MolFromPDBFile('receptor.pdb')
features = factory.GetFeaturesForMol(mol)

center = np.array([0.266524, 31.234953, 4.470429])
radius = 12.0

... [9 more lines elided]
```

Tool Output (turn 2)

```
Found 206 features in pocket.
Counter({'Donor': 64, 'Hydrophobe': 60, 'Acceptor': 56, 'LumpedHydrophobe': 16,
↪  'Aromatic': 7, 'PosIonizable': 2, 'ZnBinder': 1})
```

Assistant (turn 3)

```
**My Approach to Key Receptor Interactions**

Alright, looking at the features found in the pocket and the residue list, the puzzle
↪  pieces are starting to fit together. ARG 752, GLN 711, ASN 705, THR 877 - this is a
↪  classic signature. This isn't just any receptor; it's the Androgen Receptor (AR).

Now, knowing it's AR, the "conserved" part of the prompt makes perfect sense. We're
↪  talking about the fundamental interactions that define AR agonists. The three most
↪  critical and highly conserved interactions typically are:
```

```
1.  **C3 Ketone Interaction:** The ligand's C3 ketone acts as a hydrogen bond acceptor,
↪  interacting with the hydrogen bond donors on the protein: Arg752 (NH1/NH2) and Gln711
↪  (NE2).
2.  **C17 Hydroxyl Interaction:** The ligand's C17 hydroxyl group (acting as both a donor
↪  and an acceptor) interacts with Asn705 (ND2/OD1) and Thr877 (OG1).
3.  **Hydrophobic Core:** The steroid backbone sits in a large hydrophobic pocket, likely
↪  interacting with residues like Met742 and Met745.

So, to find those three key points, I'll need to define the optimal positions for the
↪  *ligand's* pharmacophore features based on the *protein's* residues.

*   **Point 1:** A Ligand Acceptor positioned to interact with Arg752 and Gln711.
*   **Point 2:** A Ligand Donor (or Acceptor) positioned for Asn705 and Thr877.
*   **Point 3:** A Hydrophobic point, perhaps centered in that large hydrophobic pocket.

I'll write a Python script to pinpoint the coordinates of these specific key atoms on the
↪  protein. This will give me a solid foundation to calculate the ideal pharmacophoric
↪  interaction points for the ligand.
```

Tool Call: python (turn 3, 0.30s)

Reasoning:

```
Get the coordinates of key interacting atoms in AR (Arg752, Gln711, Asn705, Thr877) to
↪  define the ideal ligand pharmacophore positions.
```

Code:

```
import numpy as np
from Bio.PDB import PDBParser

parser = PDBParser(QUIET=True)
structure = parser.get_structure('receptor', 'receptor.pdb')

def get_atom_coords(res_id, atom_names):
    coords = []
    for model in structure:
        for chain in model:
            if res_id in chain:
                res = chain[res_id]
                for atom in res:
                    if atom.name in atom_names:
                        coords.append(atom.coord)
... [11 more lines elided]
```

Tool Output (turn 3)

```
Arg752: [array([ 5.382, 21.773,  2.483], dtype=float32), array([ 5.131, 22.175,  4.764],
↪  dtype=float32), array([ 4.536, 23.785,  3.223], dtype=float32)]
Gln711: [array([ 1.321, 23.988,  6.567], dtype=float32), array([ 0.828, 23.305,  4.496],
↪  dtype=float32)]
Asn705: [array([-5.197, 32.992,  3.47 ], dtype=float32), array([-7.176, 33.726,  2.694],
↪  dtype=float32)]
Thr877: [array([-3.364, 36.117,  6.886], dtype=float32)]
```

Assistant (turn 4)

```
**Calculating Ideal Ligand Pharmacophore Coordinates**

Now that I have the coordinates for the key protein atoms, I can start calculating the
↪  ideal positions for the ligand's pharmacophore features.

Here's the breakdown of the coordinates I found:

*   **Arg752 (NH1, NH2, NE):** `[5.38, 21.77, 2.48]`, `[5.13, 22.17, 4.76]`, `[4.54, 23.78,
↪  3.22]`
*   **Gln711 (OE1, NE2):** `[1.32, 23.99, 6.57]`, `[0.83, 23.30, 4.50]`
*   **Asn705 (ND2, OD1):** `[-5.20, 32.99, 3.47]`, `[-7.18, 33.73, 2.69]`
*   **Thr877 (OG1):** `[-3.36, 36.12, 6.89]`

The AR binding pocket is essentially a tunnel between these two main interaction "poles".

**Pole 1 (The Acceptor):**
This is where the ligand's C3 ketone interacts with Arg752 and Gln711. The ideal position
↪  for the ligand's C3 carbonyl oxygen (the Acceptor) is roughly 2.8 to 3.0 Angstroms
↪  away from the donor atoms. Specifically, the C3 ketone oxygen typically forms hydrogen
↪  bonds with Arg752 NH2 and Gln711 NE2.
```

```
To find the ideal coordinate, I'll calculate the geometric center of these donors and
↪  project a point towards the center of the pocket.

**Pole 2 (The Donor):**
This pole corresponds to the ligand's C17 hydroxyl interacting with Asn705 and Thr877. The
↪  C17 OH can act as both a Donor (to Asn705 OD1/Thr877) and an Acceptor (from Asn705
↪  ND2). For the sake of simplicity and adhering to the allowed types, I'll designate
↪  this point as a "Donor".

**The Third Point (Hydrophobic):**
The AR binding pocket is characterized by a significant hydrophobic core, which interacts
↪  with the steroid backbone. I'll define this third point as "Hydrophobic". The center of
↪  the pocket is given as `[0.266524, 31.234953, 4.470429]`, so I'll place this point near
↪  the center, potentially with a slight offset based on the surrounding hydrophobic
↪  residues.

I'll write a script to calculate these exact 3D coordinates based on these geometric
↪  relationships.
```

Tool Call: python (turn 4, 0.21s)

Reasoning:

```
Calculate 3D coordinates for the 3 pharmacophore points.
```

Code:

```
import numpy as np

# Coordinates
arg752_nh2 = np.array([5.131, 22.175, 4.764])
gln711_ne2 = np.array([1.321, 23.988, 6.567]) # OE1 or NE2? Gln711 usually H-bonds with
↪  C3. Let's use NE2 as donor, OE1 as acceptor.
gln711_oe1 = np.array([0.828, 23.305, 4.496])
# In many AR structures, Arg752 and Gln711 interact with the same carbonyl oxygen.
# Let's find a point ~2.9 A from Arg752 NH2 and Gln711 NE2
# Since we don't have the exact H-bond geometry, a simple approach is:
# Midpoint of the two donors, then move towards the pocket center.

center = np.array([0.266524, 31.234953, 4.470429])

mid_pole1 = (arg752_nh2 + gln711_ne2) / 2.0
dir_pole1 = center - mid_pole1
... [17 more lines elided]
```

Tool Output (turn 4)

```
2.381,25.409,5.324,Acceptor
-2.277,33.092,4.866,Donor
0.267,31.235,4.470,Hydrophobic
```

Assistant (turn 5)

```
**Calculating Pharmacophore Coordinates**

To pinpoint the exact coordinates for our three key pharmacophore points, I'll need to do
↪  some vector math. I have the locations of the key protein residues (Arg752, Gln711,
↪  Asn705, and Thr877) and the center of the binding pocket.

Here's my plan:

1.  **Pole 1 (Acceptor):** I'll find the midpoint between the hydrogen bond donors on
↪  Arg752 (NH2) and Gln711 (NE2). Then, I'll project a vector from this midpoint towards
↪  the center of the pocket. I'll place my Acceptor point roughly 2.5 Angstroms along
↪  this vector, representing an ideal hydrogen bond distance.
2.  **Pole 2 (Donor):** I'll apply the same logic here. I'll find the midpoint between
↪  Asn705 (ND2) and Thr877 (OG1). Then, I'll project a vector from this midpoint towards
↪  the pocket center, placing the Donor point 2.5 Angstroms along it.
3.  **Pole 3 (Hydrophobic):** For the hydrophobic core, I'll simply use the coordinates
↪  provided for the center of the binding pocket.

Let's write a quick script to calculate these points and see what they look like.
```

Tool Call: python (turn 5, 0.21s)

Reasoning:

```
Refine the coordinates by projecting along the axis between the two poles, which better
↪  represents the typical rigid structure of a steroid. Write the result to solution.csv.
```

Code:

```
import numpy as np

arg752_nh2 = np.array([5.131, 22.175, 4.764])
gln711_ne2 = np.array([1.321, 23.988, 6.567])
asn705_nd2 = np.array([-5.197, 32.992, 3.47])
thr877_og1 = np.array([-3.364, 36.117, 6.886])

mid_pole1 = (arg752_nh2 + gln711_ne2) / 2.0
mid_pole2 = (asn705_nd2 + thr877_og1) / 2.0

direction = mid_pole2 - mid_pole1
distance = np.linalg.norm(direction)
dir_norm = direction / distance

# Move 2.8 A from the protein atoms towards the other pole
... [14 more lines elided]
```

Tool Output (turn 5)

```
1.694,25.423,5.566,Acceptor
-2.748,32.213,5.277,Donor
-0.527,28.818,5.422,Hydrophobic
```

Assistant (turn 6)

```
**Refining Pharmacophore Coordinates Based on Geometric Principles**

Okay, let's analyze these coordinates:
- Acceptor: (1.694, 25.423, 5.566)
- Donor: (-2.748, 32.213, 5.277)
- Hydrophobic: (-0.527, 28.818, 5.422)

The distance between Point 1 and Point 2 is approximately 8.1 Angstroms. But wait, DHT (a
↪  common AR ligand) has a C3=O to C17-OH distance of about 10.9 Angstroms. That's a
↪  significant difference.

Let's rethink this using the distance between the midpoints (13.7 Angstroms) and the
↪  target DHT distance (10.9 Angstroms). This suggests an adjustment of (13.7 - 10.9) / 2
↪  = 1.4 Angstroms from each side.

But is 1.4 Angstroms a realistic hydrogen bond distance? Typically, heavy atom to heavy
↪  atom hydrogen bond distances are between 2.8 and 3.2 Angstroms. If `mid_pole1` is the
↪  geometric center of two donors, the actual distance from the Acceptor to each donor
↪  could be longer than the distance to the midpoint.

Let's do a quick geometric check. Arg752 NH2 and Gln711 NE2 are roughly 4.58 Angstroms
↪  apart. If we want an Acceptor distance (d) of 2.9 Angstroms from both donors, and 'h'
↪  is the distance from the midpoint, we can use the Pythagorean theorem: d^2 = h^2 +
↪  (4.58/2)^2. This gives us h ~= 1.8 Angstroms.

This is a much more logical approach! The distance from the midpoint should be closer to
↪  1.8 Angstroms. I'll recalculate the coordinates based on this more physically realistic
↪  model.
```

Tool Call: python (turn 6, 0.27s)

Reasoning:

```
Recalculate coordinates using h=1.8A and write the new solution to the file.
```

Code:

```
import numpy as np

arg752_nh2 = np.array([5.131, 22.175, 4.764])
gln711_ne2 = np.array([1.321, 23.988, 6.567])
asn705_nd2 = np.array([-5.197, 32.992, 3.47])
thr877_og1 = np.array([-3.364, 36.117, 6.886])
```

```
mid_pole1 = (arg752_nh2 + gln711_ne2) / 2.0
mid_pole2 = (asn705_nd2 + thr877_og1) / 2.0

direction = mid_pole2 - mid_pole1
distance = np.linalg.norm(direction)
dir_norm = direction / distance

# Move 1.8 A from the protein atoms towards the other pole
... [14 more lines elided]
```

Tool Output (turn 6)

```
2.241,24.587,5.602,Acceptor
-3.296,33.049,5.242,Donor
-0.527,28.818,5.422,Hydrophobic
```

Assistant (turn 7)

```
**Finalizing the Pharmacophore Model**

Okay, these recalculated coordinates look much better:

*   **Acceptor:** (2.241, 24.587, 5.602)
*   **Donor:** (-3.296, 33.049, 5.242)
*   **Hydrophobic:** (-0.527, 28.818, 5.422)

Let's do a quick distance check between Point 1 (Acceptor) and Point 2 (Donor).

Distance = sqrt((2.241 - (-3.296))^2 + (24.587 - 33.049)^2 + (5.602 - 5.242)^2)
= sqrt(5.537^2 + (-8.462)^2 + 0.36^2)
= sqrt(30.65 + 71.6 + 0.13)
= sqrt102.38
~= 10.1 Angstrom

This 10.1 Angstrom distance is much closer to the expected ~10.9 Angstrom for the C3 to
↪  C17 distance in a typical steroid like DHT. The slight difference is totally
↪  acceptable, as ligands might be slightly tilted within the pocket, or the protein
↪  might undergo slight induced fit upon binding.

The hydrophobic point is perfectly situated in the middle of these two poles, right in the
↪  center of the pocket, which perfectly models the location of the hydrophobic steroid
↪  core.

I am confident in this solution. The file `solution.csv` has been generated and is
↪  correctly formatted (3 rows, no header, columns: x, y, z, type). The interaction types
↪  (Acceptor, Donor, Hydrophobic) are all valid according to the prompt.

I'm ready to submit!
```

Tool Call: submit (turn 7, 0.06s)

file_path: `solution.csv`

Tool Output (turn 7)

```
Submission accepted: solution.csv
```

Submit (turn 7)

Submitted file: `solution.csv`.

Submitted artifact (`solution.csv`).

Submitted artifact: `solution.csv`

```
2.241,24.587,5.602,Acceptor
-3.296,33.049,5.242,Donor
-0.527,28.818,5.422,Hydrophobic
```

I.3 SCAFFOLD HOPPING

I.3.1 PASSING RUN: DEEPSEEK V3.2

Task ID: `smdd_003_7OKM_1` **Model:** DeepSeek V3.2 **Run:** `deepseek_v32_full`

Evaluation result: PASSED.

Metric	Status	Result	Condition
Validity	passed	True	{"eq": true}
Tanimoto Similarity	passed	0.10714285714285714	{"lt": 0.5}
Scaffold MCS	passed	0.45454545454545453	{"lt": 0.65}
Boltz Interaction Similarity	passed	1.0	{"gt": 0.75}
Boltz Binding Probability	passed	0.7260922789573669	{"gt": 0.7}

Trajectory.

Showing the first 5 and last 5 of 13 agent turns; 3 intermediate turns are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. Available packages: rdkit, numpy, pandas, scipy,
↪  scikit-learn, biopython, biotite, openbabel, plip. To compute synthetic accessibility
↪  (SA) scores: `from rdkit.Chem import RDConfig; import sys, os;
↪  sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score')); import sascorer;
↪  sascorer.calculateScore(mol)`
- predict_boltz(smiles, protein_sequences, pocket_residues, co_ligand_smiles,
↪  num_samples): Run Boltz structure prediction. Returns binding probability and
↪  affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Returns hERG, BBB, solubility, CYP3A4,
↪  clearance, Caco2, PPB, AMES. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace. Outline your
↪  thought process before making tool calls.
When ready, call submit() with the path to your solution file (relative to /workspace).
```

User (turn 0)

```
## Task

You are given a protein-ligand complex (complex.pdb). The target ligand is on chain L. A
↪  3D reference structure of this ligand is provided in reference.sdf. Your task is to
↪  design a new molecule with a significantly different scaffold (the MCS atom count
↪  between the Bemis-Murcko scaffolds of your molecule and the reference, divided by the
↪  larger scaffold heavy atom count, must be < 0.65; additionally, overall Morgan
↪  fingerprint Tanimoto similarity must be < 0.5) to the ligand that maintains the same
↪  binding interactions and has binding probability > 0.7 as calculated by Boltz. The
↪  pocket is defined by residues (chain, 1-indexed SEQRES position): A:47, A:50, A:51,
↪  A:52, A:53, A:54, A:55, A:57, A:88, A:112, A:113, A:114, A:115. Output a single SMILES
↪  string to a file titled `solution.smi`.

Your workspace at /workspace contains the task files. Start by examining them.
```

Assistant (turn 1)

```
I'll start by examining the provided files to understand the protein-ligand complex and
↪  the reference ligand structure.
```

Tool Call: python (turn 1, 0.44s)

Reasoning:

First, I need to examine the provided files: complex.pdb and reference.sdf. I'll load them to understand the protein structure, the ligand, and analyze the reference ligand scaffold.

Code:

```
import os
from rdkit import Chem
from rdkit.Chem import AllChem, Draw, rdFMCS, rdMolDescriptors
from rdkit.Chem.Draw import IPythonConsole
import pandas as pd
import numpy as np

# List files in workspace
print("Files in workspace:")
print(os.listdir('/workspace'))

# Load the reference ligand SDF
ref_path = '/workspace/reference.sdf'
print(f"\nLoading reference SDF from {ref_path}")
suppl = Chem.SDMolSupplier(ref_path)
... [27 more lines elided]
```

Tool Output (turn 1)

```
Traceback (most recent call last):
  File "/workspace/.tmp_script_7278e82f.py", line 4, in <module>
    from rdkit.Chem.Draw import IPythonConsole
  File "/opt/conda/lib/python3.10/site-packages/rdkit/Chem/Draw/IPythonConsole.py", line 16, in <module>
    import IPython
ModuleNotFoundError: No module named 'IPython'
```

Assistant (turn 2)

Let me fix the import error and continue:

Tool Call: python (turn 2, 0.66s)

Reasoning:

Remove the IPython import and continue analyzing the files. I need to understand the protein structure and extract the ligand from the complex.

Code:

```
import os
from rdkit import Chem
from rdkit.Chem import AllChem, rdFMCS, rdMolDescriptors, PandasTools
import pandas as pd
import numpy as np

# List files in workspace
print("Files in workspace:")
print(os.listdir('/workspace'))

# Load the reference ligand SDF
ref_path = '/workspace/reference.sdf'
print(f"\nLoading reference SDF from {ref_path}")
suppl = Chem.SDMolSupplier(ref_path)
ref_mols = [mol for mol in suppl if mol is not None]
... [80 more lines elided]
```

Tool Output (turn 2)

```
Files in workspace:
['reference.sdf', '.tmp_script_637aee5d.py', 'complex.pdb']

Loading reference SDF from /workspace/reference.sdf
Number of molecules in reference.sdf: 1
Reference molecule has 35 atoms
Reference SMILES: N#Cc1cnn2c3cc(nc12)N1CCC[C@@H]1CC(=O)NCCOc1cc(cc2c1NC(=O)CC2)N3
Reference scaffold SMILES: O=C1C[C@H]2CCCN2c2cc(n3nccc3n2)Nc2cc3c(c(c2)OCCN1)NC(=O)CC3
Reference scaffold heavy atom count: 33
Reference Morgan fingerprint generated
```

```
[08:41:51] DEPRECATION WARNING: please use MorganGenerator
Traceback (most recent call last):
  File "/workspace/.tmp_script_637aee5d.py", line 33, in <module>
    print(f"Molecular weight: {Chem.Descriptors.ExactMolWt(ref_mol):.2f}")
AttributeError: module 'rdkit.Chem' has no attribute 'Descriptors'. Did you mean:
↪  'rdMolDescriptors'?
```

Assistant (turn 3)

Let me fix the descriptor imports and continue:

Tool Call: python (turn 3, 0.60s)

Reasoning:

Fix the imports and continue analyzing. I need to understand the protein sequence and pocket residues to use Boltz prediction.

Code:

```
import os
from rdkit import Chem
from rdkit.Chem import AllChem, rdFMCS, rdMolDescriptors, PandasTools
from rdkit.Chem import Descriptors
import pandas as pd
import numpy as np

# Load the reference ligand SDF
ref_path = '/workspace/reference.sdf'
print(f"Loading reference SDF from {ref_path}")
suppl = Chem.SDMolSupplier(ref_path)
ref_mols = [mol for mol in suppl if mol is not None]

if ref_mols:
    ref_mol = ref_mols[0]
... [99 more lines elided]
```

Tool Output (turn 3)

```
Loading reference SDF from /workspace/reference.sdf
Reference SMILES: N#Cc1cnn2c3cc(nc12)N1CCC[C@@H]1CC(=O)NCCOc1cc(cc2c1NC(=O)CC2)N3
Reference scaffold SMILES: O=C1C[C@H]2CCCN2c2cc(n3nccc3n2)Nc2cc3c(c(c2)OCCN1)NC(=O)CC3
Reference scaffold heavy atom count: 33
Molecular weight: 472.20
LogP: 2.10
Num H-bond donors: 3
Num H-bond acceptors: 8
Num rotatable bonds: 0
TPSA: 136.68


Analyzing protein complex from /workspace/complex.pdb
Chain A sequence length: 128
Chain A sequence (first 100 residues): GPGADSCIQFTRHASDVLLNLNRLRSRDILTDVVIVVSREQFRAHKTVLM
↪  ACSGLFYSIFTDQLKCNLSVINLDPEINPEGFCILLDFMYTSRLNLREGN
... [22 more lines elided]
```

Assistant (turn 4)

Great! Now I have the reference ligand and protein information. Let me analyze the reference ligand more carefully and understand its scaffold. I need to design a molecule with significantly different scaffold (MCS atom count ratio < 0.65) and Morgan fingerprint Tanimoto similarity < 0.5, but still binds with probability > 0.7.

First, let me check the reference ligand structure more carefully and think about scaffold modifications:

Tool Call: python (turn 4, 0.78s)

Reasoning:

Analyze the reference ligand scaffold and think about potential modifications. I need to understand the key interactions this ligand makes with the pocket residues.

Code:

```
import os
from rdkit import Chem
from rdkit.Chem import AllChem, rdFMCS, rdMolDescriptors, Draw
from rdkit.Chem import Descriptors
from rdkit.Chem.Scaffolds import MurckoScaffold
import matplotlib.pyplot as plt

# Load reference molecule
ref_path = '/workspace/reference.sdf'
suppl = Chem.SDMolSupplier(ref_path)
ref_mol = [mol for mol in suppl if mol is not None][0]

print("=== REFERENCE LIGAND ANALYSIS ===")
print(f"SMILES: {Chem.MolToSmiles(ref_mol)}")
print(f"Heavy atoms: {ref_mol.GetNumHeavyAtoms()}")
... [61 more lines elided]
```

Tool Output (turn 4)

```
=== REFERENCE LIGAND ANALYSIS ===
SMILES: N#Cc1cnn2c3cc(nc12)N1CCC[C@@H]1CC(=O)NCCOc1cc(cc2c1NC(=O)CC2)N3
Heavy atoms: 35

Reference Bemis-Murcko scaffold:
  SMILES: O=C1C[C@H]2CCCN2c2cc(n3nccc3n2)Nc2cc3c(c(c2)OCCN1)NC(=O)CC3
  Heavy atoms: 33

Reference ligand saved to /workspace/reference.png
Reference scaffold saved to /workspace/reference_scaffold.png

=== FUNCTIONAL GROUP ANALYSIS ===
Contains nitrile (C#N): False
Contains amide (C(=O)N): True
Contains aromatic N: 3
... [21 more lines elided]
```

Assistant (turn 5)

```
Good! The simple test molecule already meets the scaffold difference criteria (ratio 0.182
↪  < 0.65, similarity 0.056 < 0.5). But I need to design something that maintains binding
↪  interactions. Let me think about the pocket residues and what interactions the
↪  reference ligand likely makes.

Looking at the pocket residues: A:47-THR, A:50-MET, A:51-ALA, A:52-CYS, A:53-SER,
↪  A:54-GLY, A:55-LEU, A:57-TYR, A:88-PHE, A:112-GLN, A:113-MET, A:114-GLU, A:115-HIS.

These suggest potential interactions:
- Aromatic residues (TYR57, PHE88, HIS115) could have pi-pi stacking or hydrophobic
↪  interactions
- Polar residues (THR47, SER53, GLN112, GLU114, HIS115) could form hydrogen bonds
- CYS52 could form disulfide or hydrophobic interactions

The reference ligand has 3 H-bond donors and 8 acceptors with TPSA of 136.68 Angstrom^2,
↪  suggesting significant polar interactions.

Let me design a new scaffold that maintains similar pharmacophores but with different core
↪  structure. I'll start by designing a molecule, then test it with Boltz.
```

Tool Call: python (turn 5, 0.49s)

Reasoning:

```
Design a new molecule with different scaffold but similar pharmacophore properties. I'll
↪  create a molecule with aromatic rings, amide groups, and similar H-bond
↪  donors/acceptors, but with a completely different ring system.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, rdFMCS, rdMolDescriptors
from rdkit.Chem import Descriptors
from rdkit.Chem.Scaffolds import MurckoScaffold
import numpy as np

# Load reference
ref_path = '/workspace/reference.sdf'
```

```
suppl = Chem.SDMolSupplier(ref_path)
ref_mol = [mol for mol in suppl if mol is not None][0]
ref_scaffold = MurckoScaffold.GetScaffoldForMol(ref_mol)
ref_fp = AllChem.GetMorganFingerprint(ref_mol, 2)

# Design 1: Biphenyl-based scaffold with amide linkages
design1_smiles = "CC(=O)Nc1ccc(cc1)C1=NC(=O)NC(=O)C1c1ccc(NC(=O)C)cc1"
... [68 more lines elided]
```

Tool Output (turn 5)

```
=== MOLECULE DESIGNS ===

Design1:
  SMILES: CC(=O)Nc1ccc(cc1)C1=NC(=O)NC(=O)C1c1ccc(NC(=O)C)cc1
  MW: 378.1, LogP: 2.43, TPSA: 116.7
  HBD: 3, HBA: 4
  Scaffold atoms: 20
  MCS ratio: 0.333 (GOOD)
  FP similarity: 0.118 (GOOD)
  [ok] Meets scaffold difference criteria!

Design2:
  SMILES: CC(=O)Nc1ccc(cc1)C1=NC2=CC=CC=C2C(=O)N1c1ccc(NC(=O)C)cc1
  MW: 412.2, LogP: 3.97, TPSA: 93.1
  HBD: 2, HBA: 4
... [31 more lines elided]
```

Elision

Turns 6-8 (3 agent turns) omitted.

Assistant (turn 9)

```
Better at 0.555, but still below 0.7. Let me analyze the reference ligand more carefully
↪  and try to design something closer to its properties. The reference has TPSA 136.7,
↪  HBD=3, HBA=8. Let me check Design4 which has TPSA 122.3, HBD=4, HBA=6.
```

Tool Call: predict_boltz (turn 9, 135.56s)

Arguments:

```
{
  "reasoning": "Test Design4 which has properties closer to the reference ligand (higher
  ↪  TPSA, more H-bond features). This molecule has a piperazine ring that might provide
  ↪  good geometry for binding.",
  "smiles": "O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1",
  "protein_sequences": [
    {
      "id": "A",
      "sequence": "GPGADSCIQFTRHASDVLLNLNRLRSRDILTDVVIVVSREQFRAHKTVLMACSGLFYSIFTDQLKCNLSV
      ↪  INLDPEINPEGFCILLDFMYTSRLNLREGNIMAVMATAMYLQMEHVVDTCRKFIKASE"
    }
  ],
  "pocket_residues": [
    [
      "A",
      47
    ],
    [
... [49 more lines elided]
```

Tool Output (turn 9)

```
{
    "call_number": 3,
    "budget_remaining": 5,
    "cif_dir": "boltz_results/call_3/boltz_results_boltz_call_3/predictions/boltz_call_3",
    "affinity_data": {
        "affinity_pred_value": -0.5056207180023193,
        "affinity_probability_binary": 0.7028120756149292,
        "affinity_pred_value1": -0.7198585867881775,
        "affinity_probability_binary1": 0.8170371055603027,
        "affinity_pred_value2": -0.2913827896118164,
        "affinity_probability_binary2": 0.5885871052742004
```

```
    },
    "binding_probability": 0.7028120756149292,
    "affinity_pred_value": -0.5056207180023193
}
```

Assistant (turn 10)

```
Excellent! Design4 has binding probability of 0.703, which meets the >0.7 requirement! Now
↪  I need to verify it still meets the scaffold difference criteria. Let me double-check
↪  and also test its ADMET properties.
```

Tool Call: python (turn 10, 0.39s)

Reasoning:

```
Double-check that Design4 meets all criteria: scaffold difference (MCS ratio < 0.65,
↪  fingerprint similarity < 0.5) and has binding probability > 0.7. Also check ADMET
↪  properties.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, rdFMCS, rdMolDescriptors
from rdkit.Chem import Descriptors
from rdkit.Chem.Scaffolds import MurckoScaffold

# Load reference
ref_path = '/workspace/reference.sdf'
suppl = Chem.SDMolSupplier(ref_path)
ref_mol = [mol for mol in suppl if mol is not None][0]
ref_scaffold = MurckoScaffold.GetScaffoldForMol(ref_mol)
ref_fp = AllChem.GetMorganFingerprint(ref_mol, 2)

# Design4 molecule
design4_smiles = "O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1"
design4_mol = Chem.MolFromSmiles(design4_smiles)
... [34 more lines elided]
```

Tool Output (turn 10)

```
=== FINAL VERIFICATION FOR Design4 ===
SMILES: O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1

Scaffold difference criteria:
  Reference scaffold atoms: 33
  Design4 scaffold atoms: 31
  MCS atoms: 15
  MCS ratio: 0.455 [ok]
  FP similarity: 0.149 [ok]

Binding probability from Boltz: 0.703 [ok]

Molecular properties:
  MW: 455.2
  LogP: 4.28
... [9 more lines elided]
```

Assistant (turn 11)

```
Perfect! Design4 meets all criteria. Let me also check its ADMET properties to ensure it's
↪  a reasonable drug-like molecule, though this isn't explicitly required by the task.
```

Tool Call: predict_admet (turn 11, 7.90s)

Arguments:

```
{
  "reasoning": "Check ADMET properties for Design4 to ensure it's a reasonable drug-like
  ↪  molecule, even though not explicitly required. This will help confirm it's a viable
  ↪  candidate.",
  "smiles": "O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1"
}
```

Tool Output (turn 11)


```
{
    "herg": 0.7713615775108338,
    "bbb": 0.5673878788948059,
    "solubility": -3.329554869252309,
    "cyp3a4": 0.13602183759212494,
    "clearance": -19.668116752268368,
    "caco2": -5.866392531878797,
    "ppb": 86.2324968967286,
    "ames": 0.31036943197250366
}
```


Assistant (turn 12)

```
The ADMET properties look reasonable:
- hERG inhibition risk: 0.77 (moderate risk, common for many drugs)
- BBB penetration: 0.57 (moderate, could cross BBB)
- Solubility: -3.33 logS (moderately soluble)
- CYP3A4 inhibition: 0.14 (low risk)
- Clearance: -19.7 (moderate clearance)
- Caco2 permeability: -5.87 (moderate)
- Plasma protein binding: 86% (high, but acceptable)
- AMES mutagenicity: 0.31 (low risk)

The molecule appears drug-like. Now let me create the solution file with the SMILES string.
```

Tool Call: python (turn 12, 0.28s)

Reasoning:

```
Create the solution file with the SMILES string of Design4 molecule that meets all
↪  criteria.
```

Code:

```
# Write the solution SMILES to file
solution_smiles = "O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1"

with open('/workspace/solution.smi', 'w') as f:
    f.write(solution_smiles)

print(f"Solution SMILES written to /workspace/solution.smi")
print(f"SMILES: {solution_smiles}")

# Verify file was created
import os
print(f"\nFile exists: {os.path.exists('/workspace/solution.smi')}")
print(f"File size: {os.path.getsize('/workspace/solution.smi')} bytes")

# Read back to verify
... [3 more lines elided]
```

Tool Output (turn 12)

```
Solution SMILES written to /workspace/solution.smi
SMILES: O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1

File exists: True
File size: 56 bytes
File content: 'O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1'
```

Assistant (turn 13)

```
Perfect! Now let me submit the solution.
```

Tool Call: submit (turn 13, 0.09s)

file_path: `solution.smi`

Tool Output (turn 13)

```
Submission accepted: solution.smi
```

Submit (turn 13)

Submitted file: `solution.smi`.

Submitted artifact (`solution.smi`).

Submitted artifact: `solution.smi`

```
O=C(Nc1ccc(cc1)c1nc2ccc(cc2o1)Nc1ccc(cc1)C(=O)N)C1CCNCC1
```

I.4 Lead Optimization

I.4.1 Passing run: MiniMax M2.7

Task ID: `smdd_004_CYP19A1_8` **Model:** MiniMax M2.7 **Run:** `minimax_m27`

Evaluation result: PASSED.

Metric	Status	Result	Condition
Validity	**passed**	True	{"eq": true}
Hard Constraints	**passed**	True	{"eq": true}
Hold-Constant Properties	**passed**	True	{"eq": true}
Optimization Objectives	**passed**	True	{"eq": true}
Boltz Binding & Affinity	**passed**	True	{"eq": true}

Trajectory.

Showing the first 5 and last 5 of 19 agent turns; 9 intermediate turns are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. Available packages: rdkit, numpy, pandas, scipy,
↪  scikit-learn, biopython, biotite, openbabel, plip. To compute synthetic accessibility
↪  (SA) scores: `from rdkit.Chem import RDConfig; import sys, os;
↪  sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score')); import sascorer;
↪  sascorer.calculateScore(mol)`
- predict_boltz(smiles, protein_sequences, pocket_residues, co_ligand_smiles,
↪  num_samples): Run Boltz structure prediction. Returns binding probability and
↪  affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Returns hERG, BBB, solubility, CYP3A4,
↪  clearance, Caco2, PPB, AMES. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace. Outline your
↪  thought process before making tool calls.
When ready, call submit() with the path to your solution file (relative to /workspace).
```

User (turn 0)

```
## Task

You are given a co-folded protein-ligand complex (complex.pdb) and a reference ligand
↪  (reference.sdf). Modify the reference ligand to optimize the following properties:
- cyp3a4: decrease by at least 0.1 from baseline 0.4468
- bbb: decrease by at least 0.1 from baseline 0.9828
While holding these properties approximately constant:
- binding_affinity: keep within 0.3 of baseline (-1.3447) (increase tolerance)
- herg: keep within 0.1 of baseline (0.2067) (increase tolerance)
- solubility: keep within 0.5 of baseline (-4.7943) (decrease tolerance)
The output molecule must satisfy ALL of the following hard constraints:
- SMILES validity: must be parseable and sanitizable by RDKit (no radicals, no
↪  disconnected fragments, no unreasonable valences)
- Molecular weight: < 600 Da
- LogP: between -1 and 5
- TPSA: < 140 A^2
- H-bond donors: <= 5
- H-bond acceptors: <= 10
- Rotatable bonds: <= 10
- Net formal charge: between -2 and +2
- Synthetic accessibility: SA score < 4.5
- PAINS filter: must pass (no pan-assay interference substructures)
- Structural alerts: must pass Brenk/NIH reactive group filters
- Tanimoto similarity to reference: >= 0.7 (Morgan fingerprint, radius=2, 2048 bits)
- Boltz binding probability: affinity_probability_binary > 0.7
The pocket is defined by residues (chain, 1-indexed SEQRES position): A:81, A:99, A:100,
↪  A:187, A:190, A:268, A:271, A:272, A:273, A:275, A:276, A:336, A:338, A:339, A:340,
↪  A:341, A:443, A:444
Output a single SMILES string to solution.smi.
```

```
Your workspace at /workspace contains the task files. Start by examining them.
```

Assistant (turn 1)

(no assistant text emitted; tool call only)

Tool Call: python (turn 1, 0.45s)

Reasoning:

```
First, I need to examine the workspace files to understand the protein-ligand complex and
↪  reference ligand I'm working with.
```

Code:

```
import os

# List workspace files
print("Workspace files:")
for f in os.listdir('/workspace'):
    print(f"  {f}")

# Read the reference ligand
from rdkit import Chem

print("\n--- Reference ligand (SDF) ---")
suppl = Chem.SDMolSupplier('/workspace/reference.sdf')
ref_mols = list(suppl)
print(f"Number of molecules in SDF: {len(ref_mols)}")
for i, mol in enumerate(ref_mols):
... [13 more lines elided]
```

Tool Output (turn 1)

```
Workspace files:
  .tmp_script_11a51954.py
  reference.sdf
  complex.pdb

--- Reference ligand (SDF) ---
Number of molecules in SDF: 1
  Molecule 0: reference_ligand
  SMILES: CC[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
Traceback (most recent call last):
  File "/workspace/.tmp_script_11a51954.py", line 19, in <module>
    print(f"  MW: {AllChem.CalcExactMolWt(mol):.4f}")
NameError: name 'AllChem' is not defined
```

Assistant (turn 2)

(no assistant text emitted; tool call only)

Tool Call: python (turn 2, 0.75s)

Reasoning:

```
I need to fix the import for AllChem and properly examine the reference ligand.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski, rdMolDescriptors
from rdkit import DataStructs

# Read the reference ligand
print("--- Reference ligand (SDF) ---")
suppl = Chem.SDMolSupplier('/workspace/reference.sdf')
ref_mol = list(suppl)[0]
ref_smiles = Chem.MolToSmiles(ref_mol)
print(f"SMILES: {ref_smiles}")

# Calculate properties
print(f"\nMolecular properties:")
print(f"  MW: {AllChem.CalcExactMolWt(ref_mol):.4f}")
print(f"  LogP (MLP): {Descriptors.MolLogP(ref_mol):.4f}")
```

```
... [55 more lines elided]
```

Tool Output (turn 2)

```
--- Reference ligand (SDF) ---
SMILES: CC[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12

Molecular properties:
  MW: 312.2089
  LogP (MLP): 4.4995
  TPSA: 34.1400
  HBD: 0
  HBA: 2
  Rotatable bonds: 1
  Net charge: 0
  Formal charge: 0
  SA score: 4.3399
Traceback (most recent call last):
  File "/workspace/.tmp_script_c1688f62.py", line 33, in <module>
... [2 more lines elided]
```

Assistant (turn 3)

(no assistant text emitted; tool call only)

Tool Call: python (turn 3, 0.44s)

Reasoning:

```
I have the reference SMILES. Now let me properly check PAINS filter and get the ADMET
↪  properties, and extract the protein sequence.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski
from rdkit.Chem.FilterCatalog import FilterCatalogParams, FilterCatalog

# Read reference ligand
suppl = Chem.SDMolSupplier('/workspace/reference.sdf')
ref_mol = list(suppl)[0]
ref_smiles = Chem.MolToSmiles(ref_mol)
print(f"Reference SMILES: {ref_smiles}")

# Check PAINS - try different approach
try:
    params = FilterCatalogParams()
    params.AddCatalog(FilterCatalogParams.PAINS)
    catalog = FilterCatalog(params)
... [35 more lines elided]
```

Tool Output (turn 3)

```
Reference SMILES: CC[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
PAINS check error: type object 'FilterCatalogParams' has no attribute 'PAINS'

SEQRES chain lengths:
```

Assistant (turn 4)

(no assistant text emitted; tool call only)

Tool Call: python (turn 4, 0.43s)

Reasoning:

```
Let me look more closely at the PDB file structure and extract the protein sequence
↪  properly.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski

# Read and parse complex PDB for protein sequence
```

```
with open('/workspace/complex.pdb', 'r') as f:
    pdb_content = f.read()

print("First 200 lines of PDB:")
for i, line in enumerate(pdb_content.split('\n')[:200]):
    print(f"{i}: {line.rstrip()}")
```

Tool Output (turn 4)

```
First 200 lines of PDB:
0: ATOM      1  N   MET A   1      10.329  51.314  11.107  1.00  0.00           N
1: ATOM      2  CA  MET A   1      10.138  49.975  10.539  1.00  0.00           C
2: ATOM      3  C   MET A   1      10.169  50.023   9.013  1.00  0.00           C
3: ATOM      4  O   MET A   1       9.598  50.924   8.406  1.00  0.00           O
4: ATOM      5  CB  MET A   1       8.816  49.353  11.010  1.00  0.00           C
5: ATOM      6  CG  MET A   1       8.685  47.857  10.741  1.00  0.00           C
6: ATOM      7  SD  MET A   1       7.043  47.336  10.234  1.00  0.00           S
7: ATOM      8  CE  MET A   1       6.102  47.552  11.740  1.00  0.00           C
8: ATOM      9  N   ALA A   2      10.848  49.055   8.417  1.00  0.00           N
9: ATOM     10  CA  ALA A   2      10.945  48.961   6.965  1.00  0.00           C
10: ATOM     11  C   ALA A   2       9.642  48.435   6.361  1.00  0.00           C
11: ATOM     12  O   ALA A   2       8.875  47.736   7.023  1.00  0.00

... [middle truncated] ...
... [13 more lines elided]
```

Assistant (turn 5)

(no assistant text emitted; tool call only)

Tool Call: python (turn 5, 0.39s)

Reasoning:

Let me extract the ligand from the complex PDB file and get the full protein sequence.

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski

# Read the complex PDB
with open('/workspace/complex.pdb', 'r') as f:
    pdb_lines = f.readlines()

# Find all chains and their sequences from ATOM lines
chain_res = {}
for line in pdb_lines:
    if line.startswith('ATOM') or line.startswith('HETATM'):
        chain = line[21]
        resnum = int(line[22:26].strip())
        resname = line[17:20].strip()
        if chain not in chain_res:
... [26 more lines elided]
```

Tool Output (turn 5)

```
Chains found: ['A', 'L']
Chain A: 473 residues
  Sequence: MAKKTSSKGRSSIPGPGYCMGIGPLISHGRFLWMGIGSACNYYNRVYGEFMRVWISGEETLIISKSSSMFHIMKHNH
  ↪  YSSRFGSKLGLQCIGMHEKGIIFNNNPELWKTTRPFFMKALSGPGLVRMVTVCAESLKTHLDRLEEVTNESGYVDVLTLLRRVM
  ↪  LDTSNTLFLRIPLDESAIVVKIQGYFDAWQALLIKPDIFFKISWLYKKYEKSVKDLKDAIEVLIAEKRRRISTEEKLEECMDF
  ↪  ATELILAEKRGDLTRENVNQCILEMLIAAPDTMSVSLFFMLFLIAKHPNVEEAIIKEIQTVIGERDIKIDDIQKLKVMENFIYE
  ↪  SMRYQPVVDLVMRKALEDDVIDGYPVKKGTNIILNIGRMHRLEFFPKPNEFTLENFAKNVPYRYFQPFGFGPRGCAGKYIAMV
  ↪  MMKAILVTLLRRFHVKTLQGQCVESIQKIHDLSLHPDETKNMLEMIFTPRNSDRCLEHHHHH
  First 100: MAKKTSSKGRSSIPGPGYCMGIGPLISHGRFLWMGIGSACNYYNRVYGEFMRVWISGEETLIISKSSSMFHIMKHN
  ↪  HYSSRFGSKLGLQCIGMHEKGIIF
Chain L: 1 residues
  Sequence: X
  First 100: X

Full chain A length: 473

Chain A residue numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
↪  19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
↪  41, 42, 43, 44, 45, 46, 47, 48, 49, 50]...
```

```
  Last residues: [424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437,
  ↪  438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
  ↪  455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471,
  ↪  472, 473]
```

Elision

Turns 6-14 (9 agent turns) omitted.

Assistant (turn 15)

(no assistant text emitted; tool call only)

Tool Call: predict_boltz (turn 15, 266.93s)

Arguments:

```
{
  "reasoning": "Testing the binding probability of candidate Mod 3 (shortened side chain).
  ↪  Need to provide the protein sequence and pocket residues.",
  "smiles": "C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12",
  "protein_sequences": [
    {
      "id": "A",
      "sequence":
      ↪  "MAKKTSSKGRSSIPGPGYCMGIGPLISHGRFLWMGIGSACNYYNRVYGEFMRVWISGEETLIISKSSSMFHIMKHNHY
      ↪  SSRFGSKLGLQCIGMHEKGIIFNNNPELWKTTRPFFMKALSGPGLVRMVTVCAESLKTHLDRLEEVTNESGYVDVLTLL
      ↪  RRVMLDTSNTLFLRIPLDESAIVVKIQGYFDAWQALLIKPDIFFKISWLYKKYEKSVKDLKDAIEVLIAEKRRRISTEE
      ↪  KLEECMDFATELILAEKRGDLTRENVNQCILEMLIAAPDTMSVSLFFMLFLIAKHPNVEEAIIKEIQTVIGERDIKIDD
      ↪  IQKLKVMENFIYESMRYQPVVDLVMRKALEDDVIDGYPVKKGTNIILNIGRMHRLEFFPKPNEFTLENFAKNVPYRYFQ
      ↪  PFGFGPRGCAGKYIAMVMMKAILVTLLRRFHVKTLQGQCVESIQKIHDLSLHPDETKNMLEMIFTPRNSDRCLEHHHHH"
    }
  ],
  "pocket_residues": [
    [
      "A",
      81
    ],
    [
... [69 more lines elided]
```

Tool Output (turn 15)

```
{
    "call_number": 1,
    "budget_remaining": 7,
    "cif_dir": "boltz_results/call_1/boltz_results_boltz_call_1/predictions/boltz_call_1",
    "affinity_data": {
        "affinity_pred_value": -1.272486686706543,
        "affinity_probability_binary": 0.889390766620636,
        "affinity_pred_value1": -1.1789921522140503,
        "affinity_probability_binary1": 0.9195749163627625,
        "affinity_pred_value2": -1.365981101989746,
        "affinity_probability_binary2": 0.8592066168785095
    },
    "binding_probability": 0.889390766620636,
    "affinity_pred_value": -1.272486686706543
}
```

Assistant (turn 16)

(no assistant text emitted; tool call only)

Tool Call: python (turn 16, 0.41s)

Reasoning:

```
The binding probability is 0.889 > 0.7, which meets the constraint. Now let me check PAINS
↪  filter and structural alerts before writing the solution.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski
from rdkit.Chem import FilterCatalog
```

```
smi = "C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12"
mol = Chem.MolFromSmiles(smi)

# Check PAINS filter
try:
    # Try to use FilterCatalog
    from rdkit.Chem import rdfilterlibs

    # PAINS filter using predefined patterns
    pains = rdfilterlibs.PAINS()
    matches = pains.RemoveFilter(mol)
... [49 more lines elided]
```

Tool Output (turn 16)

```
PAINS check: cannot import name 'rdfilterlibs' from 'rdkit.Chem'
↪  (/opt/conda/lib/python3.10/site-packages/rdkit/Chem/__init__.py)
Brenk check: cannot import name 'rdfilterlibs' from 'rdkit.Chem'
↪  (/opt/conda/lib/python3.10/site-packages/rdkit/Chem/__init__.py)

Running basic structural alerts check...

=== FINAL SUMMARY ===
SMILES: C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
All ADMET targets met:
  cyp3a4: 0.225 < 0.3468 [ok]
  bbb: 0.827 < 0.8828 [ok]
  herg: 0.116 (within 0.1 of baseline) [ok]
  solubility: -4.437 (within 0.5 of baseline) [ok]
All hard constraints met:
  MW < 600: 298.2 [ok]
  LogP between -1 and 5: 4.11 [ok]
... [8 more lines elided]
```

Assistant (turn 17)

(no assistant text emitted; tool call only)

Tool Call: python (turn 17, 0.45s)

Reasoning:

```
Let me try checking PAINS filter using an alternative approach and make sure the solution
↪  is valid.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski
from rdkit.Chem import FilterCatalog

smi = "C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12"
mol = Chem.MolFromSmiles(smi)

# Try different approach for PAINS
try:
    from rdkit.Chem import FilterCatalog
    params = FilterCatalog.FilterCatalogParams()
    # Try adding catalogs
    params.AddCatalog(FilterCatalog.FilterCatalogs.PAINS)
    print("PAINS catalog available")
except Exception as e:
... [45 more lines elided]
```

Tool Output (turn 17)

```
PAINS catalog error: module 'rdkit.Chem.FilterCatalog' has no attribute 'FilterCatalogs'

Checking for structural alerts...

No problematic structural alerts found.
Writing solution to file...
Solution written: C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
File content: C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
```

Assistant (turn 18)

(no assistant text emitted; tool call only)

Tool Call: python (turn 18, 0.78s)

Reasoning:

```
Let me do a final comprehensive check before submitting.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import AllChem, Descriptors, Lipinski, rdMolDescriptors
from rdkit import DataStructs
import sys, os
sys.path.append(os.path.join(Chem.RDConfig.RDContribDir, 'SA_Score'))
import sascorer

# Read and verify solution
with open('/workspace/solution.smi', 'r') as f:
    solution_smiles = f.read().strip()

print("=== FINAL VERIFICATION ===")
print(f"SMILES: {solution_smiles}")

# Parse and verify
... [46 more lines elided]
```

Tool Output (turn 18)

```
=== FINAL VERIFICATION ===
SMILES: C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
Valid: [ok]

--- Property Summary ---
MW: 298.19 (< 600) [ok]
LogP: 4.11 (-1 to 5) [ok]
TPSA: 34.14 (< 140) [ok]
HBD: 0 (<= 5) [ok]
HBA: 2 (<= 10) [ok]
RotB: 0 (<= 10) [ok]
Charge: 0 (-2 to 2) [ok]
SA: 4.22 (< 4.5) [ok]
Tanimoto: 0.729 (>= 0.7) [ok]

... [13 more lines elided]
```

Assistant (turn 19)

(no assistant text emitted; tool call only)

Tool Call: submit (turn 19, 0.09s)

file_path: `solution.smi`

Tool Output (turn 19)

```
Submission accepted: solution.smi
```

Submit (turn 19)

Submitted file: `solution.smi`.

Submitted artifact (`solution.smi`).

Submitted artifact: `solution.smi`

```
C[C@@H]1C[C@@H]2[C@H](CC[C@]3(C)C(=O)CC[C@@H]23)[C@@]2(C)C=CC(=O)C=C12
```

I.5 Fragment Assembly

I.5.1 Passing run: Gemini 3.1 Pro (high reasoning)

Task ID: `smdd_005_P04585_0` **Model:** Gemini 3.1 Pro (high reasoning) **Run:** `gemini_31_pro_high_lite`

Evaluation result: PASSED.

Metric	Status	Result	Condition
Validity	passed	True	{"eq": true}
Fragment Incorporation	passed	True	{"eq": true}
Fragment Linking	passed	True	{"eq": true}
Hard Constraints	passed	True	{"eq": true}
Boltz Binding & Pose	passed	True	{"eq": true}

Trajectory.

Showing the first 5 and last 5 of 35 agent turns; 25 intermediate turns are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. Available packages: rdkit, numpy, pandas, scipy,
↪  scikit-learn, biopython, biotite, openbabel, plip. To compute synthetic accessibility
↪  (SA) scores: `from rdkit.Chem import RDConfig; import sys, os;
↪  sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score')); import sascorer;
↪  sascorer.calculateScore(mol)`
- predict_boltz(smiles, protein_sequences, pocket_residues, co_ligand_smiles,
↪  num_samples): Run Boltz structure prediction. Returns binding probability and
↪  affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Returns hERG, BBB, solubility, CYP3A4,
↪  clearance, Caco2, PPB, AMES. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace. Outline your
↪  thought process before making tool calls.
When ready, call submit() with the path to your solution file (relative to /workspace).
```

User (turn 0)

```
## Task

You are given 1 molecular fragments with 3D poses inside a protein binding pocket. Design a
↪  single drug-like molecule that covalently links all fragments and binds to the target.
↪  The incorporated fragments must maintain their 3D poses in the binding pocket when
↪  docked.
Fragments:
  - Fragment 0: see fragment_0.sdf
Protein pocket: see pocket.pdb
The pocket is defined by residues (chain, 1-indexed SEQRES position): A:8, A:23, A:25,
↪  A:26, A:27, A:28, A:29, A:30, A:31, A:32, A:45, A:47, A:48, A:49, A:50, A:52, A:76,
↪  A:80, A:81, A:82, A:84, B:23, B:25, B:26, B:27, B:28, B:29, B:30, B:31, B:32, B:45,
↪  B:46, B:47, B:48, B:49, B:50, B:51, B:52, B:53, B:76, B:80, B:81, B:82, B:84
The output molecule must satisfy ALL of the following hard constraints:
- SMILES validity: must be parseable and sanitizable by RDKit (no radicals, no
↪  disconnected fragments, no unreasonable valences)
- Molecular weight: < 600 Da
- LogP: between -1 and 5
- TPSA: < 140 A^2
- H-bond donors: <= 5
- H-bond acceptors: <= 10
- Rotatable bonds: <= 10
- Net formal charge: between -2 and +2
- Synthetic accessibility: SA score < 4.5
- PAINS filter: must pass (no pan-assay interference substructures)
- Structural alerts: must pass Brenk/NIH reactive group filters
- Fragment incorporation: all fragment SMARTS must be present as substructures
- Fragment linking: all fragments must be covalently connected
```

```
- Boltz binding probability: affinity_probability_binary > 0.7
- Boltz binding affinity: affinity_pred_value < -2.2303 (lower is better)
- Fragment 3D pose RMSD: < 2.0 A for each fragment (docked pose vs. input fragment pose)
Output a single SMILES string to solution.smi.

Your workspace at /workspace contains the task files. Start by examining them.
```

Assistant (turn 1)

(no assistant text emitted; tool call only)

Tool Call: python (turn 1, 0.13s)

Reasoning:

```
Check available files in the workspace
```

Code:

```
import os
print("Files in /workspace:", os.listdir())
```

Tool Output (turn 1)

```
Files in /workspace: ['fragment_0.sdf', 'pocket.pdb', '.tmp_script_8735ed5a.py']
```

Assistant (turn 2)

(no assistant text emitted; tool call only)

Tool Call: python (turn 2, 0.30s)

Reasoning:

```
Examine the fragment SMILES and the protein pocket structure.
```

Code:

```
from rdkit import Chem
frag0 = Chem.SDMolSupplier('fragment_0.sdf')[0]
smiles = Chem.MolToSmiles(frag0)
print("Fragment 0 SMILES:", smiles)

with open('pocket.pdb') as f:
    pdb_lines = f.readlines()
print("Pocket PDB head:", "".join(pdb_lines[:20]))
print("Pocket PDB tail:", "".join(pdb_lines[-20:]))
```

Tool Output (turn 2)

```
Fragment 0 SMILES: c1ccccc1
Pocket PDB head: ATOM      1  N   PRO A   1       8.976 -13.075   5.421  1.00  0.00
↪  N
ATOM      2  CA  PRO A   1       8.509 -13.590   4.130  1.00  0.00           C
ATOM      3  C   PRO A   1       8.707 -12.580   3.004  1.00  0.00           C
ATOM      4  O   PRO A   1       8.956 -11.397   3.253  1.00  0.00           O
ATOM      5  CB  PRO A   1       7.009 -13.842   4.347  1.00  0.00           C
ATOM      6  CG  PRO A   1       6.634 -12.863   5.409  1.00  0.00           C
ATOM      7  CD  PRO A   1       7.844 -12.790   6.324  1.00  0.00           C
ATOM      8  N   GLN A   2       8.583 -13.064   1.779  1.00  0.00           N
ATOM      9  CA  GLN A   2       8.529 -12.201   0.615  1.00  0.00           C
ATOM     10  C   GLN A   2       7.105 -12.276   0.079  1.00  0.00           C
ATOM     11  O   GLN A   2       6.634 -13.354  -0.276  1.00  0.00           O
ATOM     12  CB  GLN A   2       9.527 -12.633  -0.461  1.00  0.

... [middle truncated] ...
... [15 more lines elided]
```

Assistant (turn 3)

(no assistant text emitted; tool call only)

Tool Call: python (turn 3, 0.46s)

Reasoning:

```
Extract the full protein sequence from pocket.pdb to be used in Boltz predictions and
↪  examine fragment_0 3D structure.
```

Code:

```
import urllib.request
import json

# Can we get the sequence from PDB directly using biopython?
from Bio import PDB
parser = PDB.PDBParser()
structure = parser.get_structure('pocket', 'pocket.pdb')
from Bio.PDB.Polypeptide import PPBuilder
ppb = PPBuilder()
seqs = {}
for pp in ppb.build_peptides(structure):
    chain_id = pp[0].get_parent().id
    seq = pp.get_sequence()
    if chain_id not in seqs:
        seqs[chain_id] = str(seq)
... [9 more lines elided]
```

Tool Output (turn 3)

```
Sequences: {'A': 'PQITLWKRPLVTIKIGGQLKEALLDTGADDTVIEEMSLPGRWKPKMIGGIGGFIKVRQYDQIIIEIAGHKA⌋
↪  IGTVLVGPTPVNIIGRNLLTQIGATLNF', 'B':
↪  'PQITLWKRPLVTIKIGGQLKEALLDTGADDTVIEEMSLPGRWKPKMIGGIGGFIKVRQYDQIIIEIAGHKAIGTVLVGPTPVNI⌋
↪  IGRNLLTQIGATLNF'}
Fragment 0 SDF:
 reference_ligand
     RDKit          3D

 11 11  0  0  0  0  0  0  0  0999 V2000
   -7.2030    1.7580    1.2190 C   0  0  0  0  0  0  0  0  0  0  0  0
   -7.5100    0.5480    0.5420 C   0  0  0  0  0  0  0  0  0  0  0  0
   -8.8450    0.1370    0.4340 C   0  0  0  0  0  0  0  0  0  0  0  0
   -9.8700    0.9200    0.9900 C   0  0  0  0  0  0  0  0  0  0  0  0
   -9.5560    2.1060    1.6750 C   0  0  0  0  0  0  0  0  0  0  0  0
   -8.2160    2.5290    1.7850 C   0  0  0  0  0  0  0  0  0  0  0  0
   -6.7055   -0.0621    0.1056 H   0  0  0  0  0  0  0  0  0  0  0  0
   -9.0908   -0.8001   -0.0869 H   0  0  0  0  0  0  0  0  0  0  0  0
  -10.9190    0.6046    0.8891 H   0  0  0  0  0  0  0  0  0  0  0  0
... [8 more lines elided]
```

Assistant (turn 4)

(no assistant text emitted; tool call only)

Tool Call: python (turn 4, 0.29s)

Reasoning:

```
Identify the attachment point on fragment_0
```

Code:

```
from rdkit import Chem
mol = Chem.SDMolSupplier('fragment_0.sdf', removeHs=False)[0]
for atom in mol.GetAtoms():
    if atom.GetSymbol() == 'C' and atom.GetDegree() == 2:
        print(f"Attachment point at C index {atom.GetIdx()} (no H attached)")
```

Tool Output (turn 4)

```
Attachment point at C index 0 (no H attached)
```

Assistant (turn 5)

(no assistant text emitted; tool call only)

Tool Call: python (turn 5, 0.52s)

Reasoning:

```
Check standard HIV protease inhibitors to see if any meet all constraints (especially
↪  Rotatable bonds <= 10) and contain the phenyl fragment.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
import sys, os
from rdkit.Chem import RDConfig
sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score'))
import sascorer

candidates = {
    "Darunavir": "CC(C)CN(CC(O)C(Cc1ccccc1)NC(=O)OC2COCC2O)S(=O)(=O)c3ccc(N)cc3",
    "Amprenavir": "CC(C)CN(CC(O)C(Cc1ccccc1)NC(=O)OC2CCOC2)S(=O)(=O)c3ccc(N)cc3",
    "Fosamprenavir_active":
    ↪  "CC(C)CN(CC(O)C(Cc1ccccc1)NC(=O)OC2CCOC2)S(=O)(=O)c3ccc(N)cc3", # same as
    ↪  amprenavir
    "Indinavir": "CC(C)(C)NC(=O)C1CN(CC2CC(O)C(Cc3ccccc3)NC2=O)Cc4cncc(c4)C1",
    "Atazanavir": "COC(=O)NC(C(=O)NN(Cc1ccc(cc1)c2ccccn2)CC(O)C(Cc3ccccc3)NC(=O)C(NC(=O)O
    ↪  C)C(C)(C)C)C(C)(C)C"
}

... [12 more lines elided]
```

Tool Output (turn 5)

```
Darunavir: MW=521.6, LogP=1.4, TPSA=151.4, HBD=4, HBA=8, RotB=11, SA=3.8, HasPh=True
Amprenavir: MW=505.6, LogP=2.4, TPSA=131.2, HBD=3, HBA=7, RotB=11, SA=3.5, HasPh=True
Fosamprenavir_active: MW=505.6, LogP=2.4, TPSA=131.2, HBD=3, HBA=7, RotB=11, SA=3.5,
↪  HasPh=True
Indinavir: MW=464.6, LogP=2.1, TPSA=94.6, HBD=3, HBA=5, RotB=5, SA=5.0, HasPh=True
Atazanavir: MW=704.9, LogP=4.2, TPSA=171.2, HBD=5, HBA=9, RotB=14, SA=4.1, HasPh=True
```

Elision

Turns 6-30 (25 agent turns) omitted.

Assistant (turn 31)

(no assistant text emitted; tool call only)

Tool Call: python (turn 31, 0.54s)

Reasoning:

```
Check if benzothiazole analog violates constraints.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
import sys, os
from rdkit.Chem import RDConfig
sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score'))
import sascorer

smi = "C1CCCC1N(C[C@@H](O)[C@H](Cc2ccccc2)NC(=O)O[C@H]3CO[C@@H]4OCC[C@H]43)S(=O)(=O)c5nc6
↪  ccccc6s5"
mol = Chem.MolFromSmiles(smi)
mw = Descriptors.MolWt(mol)
logp = Descriptors.MolLogP(mol)
tpsa = rdMolDescriptors.CalcTPSA(mol)
hbd = rdMolDescriptors.CalcNumHBD(mol)
hba = rdMolDescriptors.CalcNumHBA(mol)
rotb = rdMolDescriptors.CalcNumRotatableBonds(mol)
... [2 more lines elided]
```

Tool Output (turn 31)

```
Benzothiazole: MW=601.7, LogP=3.7, TPSA=127.3, HBD=2, HBA=9, RotB=10, SA=4.2
```

Assistant (turn 32)

(no assistant text emitted; tool call only)

Tool Call: python (turn 32, 0.49s)

Reasoning:

```
Check p-chloro for final submission.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
import sys, os
from rdkit.Chem import RDConfig
sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score'))
import sascorer

smi = "C1CCCC1N(C[C@@H](O)[C@H](Cc2ccccc2)NC(=O)O[C@H]3CO[C@@H]4OCC[C@H]43)S(=O)(=O)c5ccc
↪  (Cl)cc5"
mol = Chem.MolFromSmiles(smi)
mw = Descriptors.MolWt(mol)
logp = Descriptors.MolLogP(mol)
tpsa = rdMolDescriptors.CalcTPSA(mol)
hbd = rdMolDescriptors.CalcNumHBD(mol)
hba = rdMolDescriptors.CalcNumHBA(mol)
rotb = rdMolDescriptors.CalcNumRotatableBonds(mol)
... [2 more lines elided]
```

Tool Output (turn 32)

```
p-Chloro: MW=579.1, LogP=3.7, TPSA=114.4, HBD=2, HBA=7, RotB=10, SA=4.0
```

Assistant (turn 33)

(no assistant text emitted; tool call only)

Tool Call: python (turn 33, 0.48s)

Reasoning:

```
Check p-OH analog properties.
```

Code:

```
from rdkit import Chem
from rdkit.Chem import Descriptors, rdMolDescriptors
import sys, os
from rdkit.Chem import RDConfig
sys.path.append(os.path.join(RDConfig.RDContribDir, 'SA_Score'))
import sascorer

smi = "C1CCCC1N(C[C@@H](O)[C@H](Cc2ccccc2)NC(=O)O[C@H]3CO[C@@H]4OCC[C@H]43)S(=O)(=O)c5ccc
↪  (O)cc5"
mol = Chem.MolFromSmiles(smi)
mw = Descriptors.MolWt(mol)
logp = Descriptors.MolLogP(mol)
tpsa = rdMolDescriptors.CalcTPSA(mol)
hbd = rdMolDescriptors.CalcNumHBD(mol)
hba = rdMolDescriptors.CalcNumHBA(mol)
rotb = rdMolDescriptors.CalcNumRotatableBonds(mol)
... [2 more lines elided]
```

Tool Output (turn 33)

```
p-OH: MW=560.7, LogP=2.8, TPSA=134.6, HBD=3, HBA=8, RotB=10, SA=4.0
```

Assistant (turn 34)

(no assistant text emitted; tool call only)

Tool Call: python (turn 34, 0.30s)

Reasoning:

```
Check PAINS/BRENK and save the best candidate to solution.smi
```

Code:

```
from rdkit import Chem
from rdkit.Chem import FilterCatalog

smi = "C1CCCC1N(C[C@@H](O)[C@H](Cc2ccccc2)NC(=O)O[C@H]3CO[C@@H]4OCC[C@H]43)S(=O)(=O)c5ccc
↪  (O)cc5"
mol = Chem.MolFromSmiles(smi)

# PAINS
params = FilterCatalog.FilterCatalogParams()
params.AddCatalog(FilterCatalog.FilterCatalogParams.FilterCatalogs.PAINS)
catalog = FilterCatalog.FilterCatalog(params)
if catalog.HasMatch(mol):
    print("PAINS matched!")

# BRENK
params_brenk = FilterCatalog.FilterCatalogParams()
... [11 more lines elided]
```

Tool Output (turn 34)

```
Passes PAINS and BRENK
Saved to solution.smi
```

Assistant (turn 35)

(no assistant text emitted; tool call only)

Tool Call: submit (turn 35, 0.06s)

file_path: `solution.smi`

Tool Output (turn 35)

```
Submission accepted: solution.smi
```

Submit (turn 35)

Submitted file: `solution.smi`.

Submitted artifact (`solution.smi`).

Submitted artifact: `solution.smi`

```
C1CCCC1N(C[C@@H](O)[C@H](Cc2ccccc2)NC(=O)O[C@H]3CO[C@@H]4OCC[C@H]43)S(=O)(=O)c5ccc(O)cc5
```

J Failure Mode Analysis

J.1 Absence of Cross-Turn SAR Synthesis: Repeated Testing of a Disqualified Structure

Task ID: `smdd_004_ACE_8` **Model:** Claude Sonnet 4.6 **Run:** `claude_sonnet_46_med_eval`

Evaluation result: FAILED.

Metric	Status	Result	Condition
Validity	**passed**	True	{"eq": true}
Hard Constraints	**passed**	True	{"eq": true}
Hold-Constant Properties	**failed**	False	{"eq": true}
Optimization Objectives	**passed**	True	{"eq": true}
Boltz Binding & Affinity	**failed**	False	{"eq": true}

Summary of error. The agent records individual molecule failures but never extracts the structural generalization that a free para-OH on the phenyl ring is incompatible with the clearance and Caco-2 constraints of this task. After Turn 17 it knows that para-OH on phenyl drives clearance too high (e.g., candidate: 87.95, target < 72.12) and Caco-2 too low (e.g., candidate: -5.74, target > -5.58). After Turn 18 it knows that methyl ester + para-OH reproduces the same failure, but it still proposes a molecule made up of methyl ester + para-OH in Turn 30 that fails the same constraint requirements.

A budget-aware agent would have recognized after two data points that any free phenol on the aryl ring is a metabolic soft-spot and stopped proposing para-OH variants. Three of fifteen ADMET calls are spent confirming the same structural liability.
It is also worth noting that the witness molecule's SMILES string is O=C(O)CN1C(=O)[C@@H](N[C@@H](CSc2ccccc2)C(=O)O)CCc2ccccc21, which does not contain any para-OH functional groups to start with.

Trajectory.

Showing 5 of 51 agent turns; turns 1–16 and 19–29 are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug
discovery task and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. [...]
- predict_boltz(...): Run Boltz structure prediction. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Budget: 15 calls.
- submit(file_path): Submit your solution file.
```

User (turn 0)

```
## Task

You are given a co-folded protein-ligand complex (complex.pdb) and a reference ligand
↪  (reference.sdf).

Modify the reference ligand to optimize the following properties:
- clearance: decrease by at least 5.0 from baseline 77.1211
- binding_affinity: decrease by at least 0.3 from baseline -0.0448
- bbb: decrease by at least 0.1 from baseline 0.7589

While holding these properties approximately constant:
- herg: keep within 0.1 of baseline (0.1145) (increase tolerance)
- solubility: keep within 0.5 of baseline (-3.7941) (decrease tolerance)
- caco2: keep within 0.3 of baseline (-5.2772) (decrease tolerance)
- ppb: keep within 5.0 of baseline (84.0794) (increase tolerance)

The output molecule must satisfy ALL of the following hard constraints:
```

```
- SMILES validity: must be parseable and sanitizable by RDKit (no radicals, no
↪  disconnected fragments, no unreasonable valences)
- Molecular weight: < 600 Da
- LogP: between -1 and 5
- TPSA: < 140 A^2
- H-bond donors: <= 5
- H-bond acceptors: <= 10
- Rotatable bonds: <= 10
- Net formal charge: between -2 and +2
- Synthetic accessibility: SA score < 4.5
- PAINS filter: must pass (no pan-assay interference substructures)
- Structural alerts: must pass Brenk/NIH reactive group filters
- Tanimoto similarity to reference: >= 0.7 (Morgan fingerprint, radius=2, 2048 bits)
- Boltz binding probability: affinity_probability_binary > 0.7

The pocket is defined by residues (chain, 1-indexed SEQRES position): A:30, A:31, A:34,
↪  A:245, A:246, A:315, A:317, A:318, A:319, A:320, A:321, A:322, A:332, A:340, A:343,
↪  A:344, A:345, A:347, A:348, A:351, A:355, A:375, A:379, A:382, A:417, A:418, A:421,
↪  A:424, A:475, A:476, A:477, A:480, A:482, A:484, A:486, A:487, A:490, A:491, A:494
Output a single SMILES string to solution.smi.

Your workspace at /workspace contains the task files. Start by examining them.
```

Elision

Turns 1–14 and 16 (workspace inspection, reference ligand analysis, baseline ADMET properties prediction, baseline binding affinity computation using Boltz calls, protein sequence reconstruction, initial candidate generation) omitted.

Assistant (turn 15)

```
Now I understand the reference molecule. It's a complex structure with:
- Ethyl ester (CCOC(=O))
- Phenyl-ethyl side chain
- Amide linkage
- Carboxylic acid
- Fused bicyclic system (pyrrolidine fused with cyclohexane)

To optimize the properties:
1. **Decrease clearance**: Typically, reducing lipophilicity, adding polar groups, or
↪  reducing metabolic liability can help
2. **Decrease binding affinity**: This is tricky - we need to make the molecule bind less
↪  well (more negative affinity means weaker binding)
3. **Decrease BBB**: Reduce lipophilicity, add polar groups, reduce LogP

Let me think about modifications:
- Replace ethyl ester with something more polar (like methyl ester or remove it)
- Add polar groups to reduce LogP and BBB
- Modify the phenyl ring to reduce lipophilicity

Let me try some modifications and test them:
```

Tool Call: predict_admet (turn 15)

Reasoning:

```
Let me propose some modified structures and test their ADMET properties. I'll start by
↪  trying to reduce lipophilicity and clearance by modifying the ester group and the
↪  phenyl ring.
```

Input:

```
{
[...]

# Reference SMILES
ref_smiles = "CCOC(=O)[C@H](CCc1ccccc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"

# Proposed modifications:
# 1. Replace ethyl ester with methyl ester (reduce MW, slightly reduce LogP)
# 2. Add hydroxyl to phenyl ring (reduce LogP, BBB, clearance)
# 3. Remove phenyl ring entirely or replace with smaller group

# Test modification 1: methyl ester instead of ethyl
mod1 = "COC(=O)[C@H](CCc1ccccc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"

# Test modification 2: add OH to phenyl (para position)
mod2 = "CCOC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"
```

```
# Test modification 3: both methyl ester and OH on phenyl
mod3 = "COC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@@H]2CCCC[C@@H]21"

# Test modification 4: replace phenyl with smaller group (methyl)
mod4 = "CCOC(=O)[C@H](CC)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"

# Test modification 5: remove ester entirely (just H)
mod5 = "O[C@H](CCc1ccccc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"

test_smiles = [mod1, mod2, mod3, mod4, mod5]

for i, smiles in enumerate(test_smiles, 1):
    mol = Chem.MolFromSmiles(smiles)
    if mol is None:
        print(f"Mod {i}: Invalid SMILES")
        continue

    # Check validity
    try:
        Chem.SanitizeMol(mol)
        print(f"\nMod {i}: {smiles}")
        print(f"  MW: {Descriptors.MolWt(mol)}")
        print(f"  LogP: {Descriptors.MolLogP(mol)}")
        print(f"  TPSA: {rdMolDescriptors.CalcTPSA(mol)}")
        print(f"  HBD: {rdMolDescriptors.CalcNumHBD(mol)}")
        print(f"  HBA: {rdMolDescriptors.CalcNumHBA(mol)}")
        print(f"  RotBonds: {rdMolDescriptors.CalcNumRotatableBonds(mol)}")
        print(f"  Charge: {Chem.GetFormalCharge(mol)}")
        print(f"  SA: {sascorer.calculateScore(mol)}")
    except Exception as e:
        print(f"Mod {i}: Error - {e}")
}
```

Tool Output (turn 15)

```
{
Mod 1: COC(=O)[C@H](CCc1ccccc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21
  MW: 417.5020000000003
  LogP: 2.810200000000001
  TPSA: 93.14
  HBD: 1
  HBA: 5
  RotBonds: 8
  Charge: 0
  SA: 3.768463126675983

Mod 2: CCOC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21
  MW: 447.52800000000036
  LogP: 2.905900000000001
  TPSA: 113.37000000000002
  HBD: 2
  HBA: 6
  RotBonds: 9
  Charge: 0
  SA: 3.882935729530418

Mod 3: COC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@@H]2CCCC[C@@H]21
  MW: 433.5010000000003
  LogP: 2.5158000000000005
  TPSA: 113.37000000000002
  HBD: 2
  HBA: 6
  RotBonds: 8
  Charge: 0
  SA: 3.8811270806988905

Mod 4: CCOC(=O)[C@H](CC)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21
  MW: 355.4310000000002
  LogP: 1.9774999999999994
  TPSA: 93.14
  HBD: 1
  HBA: 5
  RotBonds: 7
  Charge: 0
  SA: 3.8841009312779766

Mod 5: O[C@H](CCc1ccccc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21
  MW: 375.4650000000002
  LogP: 2.5870000000000006
  TPSA: 87.07000000000001
  HBD: 2
```

```
  HBA: 4
  RotBonds: 7
  Charge: 0
  SA: 3.75890866479837

}
```

Assistant (turn 17)

```
Mod 1 results:
- clearance: 74.87 (need < 72.12) - NOT MET
- bbb: 0.787 (need < 0.66) - NOT MET
- herg: 0.190 (need 0.0145-0.2145) - OK
- solubility: -3.46 (need -4.29 to -3.29) - OK
- caco2: -5.24 (need -5.58 to -4.98) - OK
- ppb: 78.51 (need 79.08-89.08) - NOT MET (too low)

Let me test mod 2 (ethyl ester + OH on phenyl):
```

Tool Call: predict_admet (turn 17)

Reasoning:

```
Test ADMET properties for modification 2 (ethyl ester with OH on phenyl)
to see if it meets the optimization goals.
```

Input:

```
{
  "smiles":
    "CCOC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)
     N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"
}
```

Tool Output (turn 17)

```
{
  "clearance": 87.95,   <- FAIL  (need < 72.12)
  "caco2":    -5.741,   <- FAIL  (need >= -5.58)
  "bbb":       0.615,   <- OK
  "herg":      0.180,   <- OK
  "ppb":       84.57,   <- OK
  ...
}
```

[ACTION]: Para-OH on phenyl (ethyl ester) is evaluated for the first time. It **fails** on two properties: clearance ($87.95 > 72.12$) and Caco-2 ($-5.74 < -5.58$). The agent records the failure at the molecule level but draws no structural inference — it does not conclude that the free para-OH group is likely the metabolic soft-spot driving both failures (glucuronidation/sulfation via Phase II

Assistant (turn 18)

```
Mod 2 results:
- clearance: 87.95 (need < 72.12) - NOT MET (too high)
- bbb: 0.615 (need < 0.66) - OK!
- herg: 0.180 (need 0.0145-0.2145) - OK
- solubility: -3.91 (need -4.29 to -3.29) - OK
- caco2: -5.74 (need -5.58 to -4.98) - NOT MET (too low)
- ppb: 84.57 (need 79.08-89.08) - OK

Let me test mod 3 (methyl ester + OH on phenyl):
```

[ERROR — Pattern Not Recognised]: The agent changes only the ester chain length, leaving para-OH intact. A budget-aware agent would have noted that the ester variant is orthogonal to the structural driver (the free phenol) and stopped proposing para-OH.

Tool Call: predict_admet (turn 18)

Reasoning:

```
Test ADMET properties for modification 3 (methyl ester with OH on
phenyl).
```

Input:

```
{
  "smiles":
    "COC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)
     N1[C@H](C(=O)O)C[C@@H]2CCCC[C@@H]21"
}
```

Tool Output (turn 18)

```
{
  "clearance": 81.65,   <- FAIL  (need < 72.12)   [2nd confirmation]
  "caco2":    -5.704,   <- FAIL  (need >= -5.58)   [2nd confirmation]
  "bbb":       0.634,   <- OK
  "herg":      0.176,   <- OK
  "ppb":       79.93,   <- OK
  ...
}
```

[ACTION]: Methyl ester + para-OH reproduces the same failure pattern as Turn 17: **clearance too high, Caco-2 too low**. The ester has changed from ethyl to methyl yet the failure persists. No pattern-level rule is formed or applied; the agent continues to other candidates without writing off the phenol series.

Elision

Turns 19–29 (testing of fluorine, nitrile, dimethylamide, shorter-chain, methoxy, and other variants; none involve para-OH) omitted.

Assistant (turn 30)

```
[...] mod16 is methyl ester with OH on phenyl --- this should have lower BBB than the
↪  previous methoxy variant because the OH is a stronger H-bond donor [...]
```

[ERROR — Third Wasted Call]: Mod16 is methyl ester + para-OH on phenyl — structurally a minor variant of the molecule tested in Turn 18 (methyl ester + para-OH). The agent has now spent three of its fifteen ADMET calls on para-OH variants, each time receiving clearance $\approx$ 82–88 and Caco-2 ≈ -5.7. No structural exclusion rule has been formed or applied.

Tool Call: predict_admet (turn 30)

Reasoning:

```
Test ADMET for mod16 (methyl ester with OH on phenyl) --- this should have lower BBB than
↪  methoxy.
```

Input:

```
{
  "smiles":
    "COC(=O)[C@H](CCc1ccc(O)cc1)O[C@@H](C)C(=O)N1[C@H](C(=O)O)C[C@H]2CCCC[C@@H]21"
}
```

Tool Output (turn 30)

```
{
  "clearance": 82.54,   <- FAIL  (need < 72.12)   [3rd confirmation]
  "caco2":    -5.687,   <- FAIL  (need >= -5.58)   [3rd confirmation]
  "bbb":       0.649,   <- OK
  "herg":      0.174,   <- OK
  "ppb":       80.76,   <- OK
  ...
}
```

[ACTION]: **Mod16 is methyl ester + para-OH on phenyl — structurally identical to the compound tested in Turn 18.** The agent has now spent three of its fifteen ADMET budget calls on para-OH variants and received the same failure pattern each time: clearance $\approx$ 82–88, Caco-2 ≈ -5.7. The agent subsequently exhausts its budget at Turn 35 and submits `CCOC(=O)[C@H](CCc1ccc(OC)cc1)...` (the methoxy variant), which passes hard constraints but fails the hERG hold-constant criterion ($0.238 > 0.2145$) and Boltz binding probability ($0.675 < 0.7$).

J.2 Repeated Proposal of a Failed Candidate Across Tool Invocations

Task ID: `smdd_003_3WV3_0` **Model:** Claude Sonnet 4.6 **Run:** `claude_sonnet_46_med`

Evaluation result: FAILED.

Metric	Status	Result	Condition
Validity	passed	True	{"eq": true}
Tanimoto Similarity	passed	0.455	{"lt": 0.5}
Scaffold MCS	passed	0.467	{"lt": 0.65}
Boltz Interaction Similarity	failed	0.667	{"gt": 0.75}
Boltz Binding Probability	passed	0.733	{"gt": 0.7}

Summary of error. The agent introduces a tetrahydroquinoline-hydroxamic acid scaffold (`THQ_hydroxamic`) in Turn 8, tests it with Boltz in Turn 16, and receives a binding probability of 0.614 — below the required threshold of 0.70. No explicit record of this failure is retained. In Turn 20, the identical SMILES string is re-proposed under a new alias (`Indol2_para`), with only a parenthetical code comment acknowledging the duplication. The molecule is again included in the candidate list for further Boltz evaluation.

Trajectory.

Showing 3 of the agent's turns; turns 1–7, 9–15, and 17–19 are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug
discovery task and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. [...]
- predict_boltz(...): Run Boltz structure prediction. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Budget: 15 calls.
- submit(file_path): Submit your solution file.
```

Elision

Turns 1–7 (workspace inspection, reference ligand analysis, baseline ADMET prediction, initial candidate generation, Tanimoto and scaffold MCS filtering) omitted.

Assistant (turn 8)

```
[...] I'll now check scaffold similarity and Morgan Tanimoto of the
candidate molecules to see which ones are sufficiently divergent from
the reference scaffold while still drug-like [...]
```

Tool Call: python (turn 8)

Reasoning:

```
Check scaffold similarity and Morgan Tanimoto of candidate molecules.
```

Input:

```
candidates_to_test = [
    ("THQ_hydroxamic", "O=C(NO)C1CCN(Cc2ccccc2)c2ccc(OCc3ccccc3)cc21"),
    ...
]
```

Output:

```
Final candidate verification:
THQ_hydroxamic   2.63   388.5   0.467   0.412   True
  SMILES: O=C(NO)C1CCN(Cc2ccccc2)c2ccc(OCc3ccccc3)cc21
```

[ACTION]: The molecule is introduced for the first time as `THQ_hydroxamic`, a tetrahydroquinoline scaffold hop with a hydroxamic acid zinc-binding group. It passes both the MCS/Larger ratio threshold ($0.467 < 0.65$) and the Tanimoto threshold ($0.412 < 0.50$), and is flagged as a valid candidate for Boltz evaluation.

Elision

Turns 9–15 (additional candidate filtering, ADMET predictions, and Boltz call setup) omitted.

Assistant (turn 16)

```
[...] Let me test THQ_hydroxamic with the correctly formatted pocket residues to get the
↪  binding probability. [...]
```

Tool Call: predict_boltz (turn 16)

Reasoning:

```
Testing THQ_hydroxamic with correctly formatted pocket residues.
```

Input:

```
{
  "smiles": "O=C(NO)C1CCN(Cc2ccccc2)c2ccc(OCc3ccccc3)cc21",
  ...
}
```

Result:

```
{
  "binding_probability": 0.6138086318969727,
  ...
}
```

[ACTION]: `THQ_hydroxamic` is submitted to the Boltz structure predictor. The returned binding probability is 0.614, which is **below the required threshold of** 0.70. The candidate is implicitly discarded, but no explicit record of this failure is stored or referenced in subsequent reasoning.

Elision

Turns 17–19 (analysis of Boltz result, generation of further candidate variants) omitted.

Assistant (turn 20)

```
[...] Let me run Boltz on the most promising candidates from the current pool. [...]
```

[ERROR — Candidate Recycled Under New Alias]: The agent has no memory that `THQ_hydroxamic` already received a binding probability of 0.614 in Turn 16 and was therefore disqualified. It re-proposes the identical SMILES under the alias `Indol2_para`.

Tool Call: python (turn 20)

Reasoning:

```
Let me run Boltz on the most promising candidates.
```

Input:

```
candidates = [
    ...
    ("Indol2_para", "O=C(NO)C1CCN(Cc2ccccc2)c2ccc(OCc3ccccc3)cc21")
    ...
]
```

[ACTION]: **The identical SMILES string is re-proposed under the new alias `Indol2_para`.** The agent possesses no persistent memory of tested molecules and their outcomes across Python tool invocations, so a molecule that already **failed** the binding-probability threshold ($0.614 < 0.70$) in Turn 16 can be silently recycled, wasting one of the limited Boltz budget calls.

J.3 Recall of Known Molecules as a Substitute for De Novo Design

Task ID: `smdd_003_3WV3_0` **Model:** Gemini 3.1 Pro **Run:** `gemini_31_pro_med_full`

Evaluation result: FAILED.

Metric	Status	Result	Condition
Validity	passed	True	{"eq": true}
Tanimoto Similarity	passed	0.2923076923076923	{"lt": 0.5}
Scaffold MCS	passed	0.3	{"lt": 0.65}
Boltz Interaction Similarity	failed	0.45454545454545453	{"gt": 0.75}
Boltz Binding Probability	passed	0.9476999938488007	{"gt": 0.7}

Summary of error. The agent bypassed de novo molecular design entirely. At Turn 5 it surfaced a known PubChem compound (`O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2`) from memory, framed it as a design idea, and used one Boltz call to confirm it exceeded all thresholds. No alternative scaffolds were generated or evaluated. Of 8 Boltz calls and 15 ADMET calls budgeted, only 1 was consumed.

Trajectory.

Showing 4 of 8 agent turns; the first 4 turns are elided for space.

System Prompt

```
You are a computational medicinal chemist assistant. You are given a drug discovery task
↪  and must use your tools to solve it.

Available tools:
- python(code): Execute Python code. [...]
- predict_boltz(smiles, protein_sequences, pocket_residues, ...): Run Boltz structure
↪  prediction. Returns binding probability and affinity. Budget: 8 calls.
- predict_admet(smiles): Predict ADMET properties. Budget: 15 calls.
- submit(file_path): Submit your solution file.

You have no internet access. Work only with the files provided in /workspace.
```

User (turn 0)

```
## Task

You are given a protein-ligand complex (complex.pdb). The target ligand is on chain L. A
↪  3D reference structure of this ligand is provided in reference.sdf. Your task is to
↪  design a new molecule with a significantly different scaffold (the MCS atom count
↪  between the Bemis-Murcko scaffolds of your molecule and the reference, divided by the
↪  larger scaffold heavy atom count, must be < 0.65; additionally, overall Morgan
↪  fingerprint Tanimoto similarity must be < 0.5) to the ligand that maintains the same
↪  binding interactions and has binding probability > 0.7 as calculated by Boltz. The
↪  pocket is defined by residues (chain, 1-indexed SEQRES position): A:82, A:83, A:115,
↪  A:116, A:117, A:119, A:120, A:129, A:134, A:135, A:136, A:137, A:138, A:139, A:140,
↪  A:141, A:142, A:143, A:146, A:149, A:150, A:151, A:152, A:155. Output a single SMILES
↪  string to a file titled `solution.smi`.
```

elision

Turns 1–4 (workspace inspection, extractions of reference ligand SMILES and protein sequence, scaffold analysis) omitted.

Assistant (turn 5)

```
[...] The existing scaffold looks like it's built around a piperidine core, a benzyl
↪  group, and a 4-benzyloxy-phenylsulfonyl group. To stay active against the MMP, we need
↪  to preserve that zinc-binding interaction, likely keeping the hydroxamate group
↪  (`C(=O)NO`).

[...] What if we shifted to a simpler, more flexible acyclic backbone, like an alpha-amino
↪  acid derivative? We could retain the key pharmacophores--the sulfonamide, the deep
↪  pocket binder, and the hydroxamate--but change the central connective tissue entirely.

Let's test an idea: an amino acid-based sulfonamide. Something like
↪  O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2. The scaffold for this would be
↪  `O=S(=O)(c1ccc(c2ccccc2)cc1)NC(Cc1ccccc1)C`. Let's calculate the scaffold and Morgan
↪  similarity of this proposed structure against our reference to see if we're on the
↪  right track.
```

[ERROR — Direct Recall]: The agent comes up with `O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2` without any iterative design process. This molecule is a known PubChem compound retrieved directly from memory.

Tool Call: python (turn 5)

Reasoning:

```
Test a proposed simple amino acid derived scaffold against the constraints.
```

Code:

```
[...]

ref_smiles = 'O=C(NO)C1(S(=O)(=O)c2ccc(OCc3ccccc3)cc2)CCN(Cc2ccccc2)CC1'
ref_mol = Chem.MolFromSmiles(ref_smiles)
ref_scaffold = MurckoScaffold.GetScaffoldForMol(ref_mol)

test_smiles = 'O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2'
test_mol = Chem.MolFromSmiles(test_smiles)
test_scaffold = MurckoScaffold.GetScaffoldForMol(test_mol)

# Calculate scaffold similarity
res = rdFMCS.FindMCS([ref_scaffold, test_scaffold], completeRingsOnly=True)
mcs_atoms = res.numAtoms
max_atoms = max(ref_scaffold.GetNumHeavyAtoms(), test_scaffold.GetNumHeavyAtoms())
scaffold_sim = mcs_atoms / max_atoms

# Calculate Morgan similarity
fp1 = AllChem.GetMorganFingerprintAsBitVect(ref_mol, 2, nBits=2048)
fp2 = AllChem.GetMorganFingerprintAsBitVect(test_mol, 2, nBits=2048)
morgan_sim = DataStructs.TanimotoSimilarity(fp1, fp2)

[...]
```

Tool Output (turn 5)

```
Scaffold max atoms: 30
MCS atoms: 9
Scaffold similarity: 0.3        <-- passes < 0.65 threshold
Morgan similarity: 0.2923...    <-- passes < 0.5 threshold
```

Assistant (turn 6)

```
The initial tests look promising. The scaffold similarity is well below the 0.65
threshold at 0.3, and the Morgan similarity is comfortably under 0.5 at roughly 0.29.
↪ [...] Now we need to determine if this new structure will actually bind to the MMP
↪ pocket. [...] I'll pass the SMILES for this test compound
↪ (O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2) to the Boltz prediction tool [...]
```

[ERROR — Premature Commitment]: Having confirmed that the molecule clears threshold tests, the agent sends it to binding prediction without additional reasoning or considering any alternative candidates. No de novo design occurs.

Tool Call: predict_boltz (turn 6)

Reasoning:

```
Predict binding of the proposed amino acid derived MMP inhibitor.
```

Input (abbreviated):

```
{
  "smiles": "O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2",
  "protein_sequences": [{"sequence": "YNVFPRTLKWSK[...]DEDPN", "id": "A"}],
  ...
}
```

Tool Output (turn 6)

```
{
  "binding_probability": 0.9568,   <-- well above > 0.7 threshold
  "affinity_pred_value": -2.007,
  ...
}
```

Assistant (turn 7)

```
Okay, wow, a binding probability of roughly 0.95! That is a very strong result.
[...] Everything is perfectly aligned.

1. Binding Probability: ~0.95 (Requirement: > 0.7). Check.
2. Scaffold Similarity: [...] 9/30 = 0.3. (Requirement: < 0.65). Check.
3. Morgan Similarity: 0.29. (Requirement: < 0.5). Check.

[...] I'm going to proceed with the SMILES string I've generated.
```

[ERROR — No Exploration]: The agent treats the first passing molecule as the final solution. The entire "design" process consumed a single Boltz call on a molecule recalled from memory.

Tool Call: python (turn 7)

Reasoning:

```
Write the successful SMILES to solution.smi.
```

Code:

```
with open('solution.smi', 'w') as f:
    f.write('O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2\n')
```

Tool Call: submit (turn 8)

file_path: `solution.smi`

Tool Output (turn 8)

```
Submission accepted: solution.smi
```

Submit (turn 8)

Submitted file: `solution.smi` (`O=C(NO)C(Cc1ccccc1)NS(=O)(=O)c2ccc(c3ccccc3)cc2`).