

Coder as Editor: Code-driven Interpretable Molecular Editing

Wenyu Zhu^{1,3}, Chengzhu Li⁴, Xiaohe Tian⁵, Yifan Wang⁶,
Yinjun Jia¹, Jianhui Wang¹, Bowen Gao^{1,3,7}, Haichuan Tan^{1,3,7},
Ya-Qin Zhang¹, Wei-Ying Ma¹, Yanyan Lan^{1,2,3,*}

¹Institute for AI Industry Research, Tsinghua University, Beijing, China

²Beijing Frontier Research Center for Biological Structure, Tsinghua University, Beijing, China

³AIPher Technology Co., Ltd., Beijing, China

⁴Zhili College, Tsinghua University, Beijing, China

⁵School of Pharmaceutical Sciences, Peking University, Beijing, China

⁶College of AI, Tsinghua University, Beijing, China

⁷Department of Computer Science and Technology, Tsinghua University, Beijing, China

*Correspondence: lanyanyan@air.tsinghua.edu.cn

Abstract

Molecular design is a central task in drug discovery that requires precise structural reasoning and domain knowledge. While large language models (LLMs) have shown promise in generating high-level editing intentions in natural language, they often struggle to faithfully execute these modifications, particularly when operating on non-intuitive representations like SMILES. We introduce MECo, a framework that bridges reasoning and execution by translating editing actions into executable code. MECo reformulates molecular modification for LLMs as a cascaded framework: generating human-interpretable editing intentions from a molecule and design objective, followed by translating those intentions into executable structural edits via code generation. Our approach achieves over 98% accuracy in reproducing held-out realistic edits derived from chemical reactions and target-specific compound pairs. On downstream benchmarks spanning physicochemical properties and target activities, MECo substantially improves consistency by 38-86 percentage points to 90%+ and achieves higher success rates over SMILES-based baselines while preserving structural similarity. By aligning intention with execution, MECo enables consistent, controllable and interpretable molecular design, laying the foundation for high-fidelity feedback loops and collaborative human-AI workflows in drug discovery.

1 Introduction

Recent reasoning-centric large language models (LLMs) have demonstrated notable progress in scientific problem-solving, particularly in programming (Wang et al., 2023), mathematics (Lewkowycz et al., 2022), and chemistry (Zhang et al., 2024). In chemistry, their success on question answering tasks highlights an abil-

ity to encode domain knowledge (Hou et al., 2025), but applying this knowledge to tasks like molecular modification, which require both chemical reasoning and precise structural control, remains a challenge.

A growing number of studies have applied LLMs to molecular modification by prompting them to generate optimized molecules in the form of SMILES strings, conditioned on an input structure and a desired property goal (e.g., increased solubility or binding affinity) (Liu et al., 2024a; Ye et al., 2025). While intuitive, this generation paradigm faces major limitations. First, generated SMILES may not always be chemically plausible. Second, even valid outputs may diverge from the design rationale articulated by reasoning LLMs, undermining interpretability and expert trust. Such misalignments hinder human-in-the-loop workflows where reproducibility and verifiability are critical, and break the feedback loop required for iterative model refinement. Moreover, extensive uncontrolled edits may severely compromise synthetic feasibility by disrupting established synthetic routes and limiting the reuse of existing intermediates, thereby increasing the practical cost and difficulty of experimental validation.

At the root of these limitations is a modality mismatch: SMILES is a linearized encoding of molecular graphs, originally designed for compact storage in cheminformatics systems. A single molecule may correspond to many valid SMILES depending on the traversal path, and even small structural edits can cause large, unintuitive changes in the string. In contrast, chemists typically reason over molecular graphs and use graphical tools instead of directly modifying SMILES, making precise control difficult for LLMs.

To bridge this gap, we propose using code as

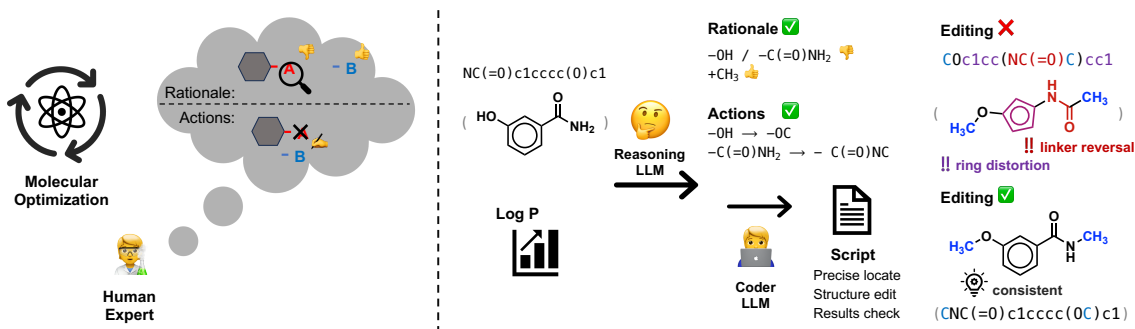


Figure 1: Motivation for MECO: bridging reasoning and execution in molecular optimization through code generation. Left: Human chemists reason over molecular graphs to design targeted edits, such as modifying a functional group to adjust polarity or introducing new interactions, and annotate them directly on the structure. Top right: Reasoning LLMs can generate similar high-level rationales and editing actions, but often struggle to execute them faithfully due to the limitations of sequential molecular representations like SMILES. Bottom right: MECO addresses this gap by introducing a coder LLM that translates editing actions into executable code, enabling precise, interpretable, and reproducible structural edits on the molecular graph.

an intermediate representation, a domain where LLMs have already demonstrated strong proficiency. Rather than generating molecular structures directly as SMILES, we reformulate molecular editing as a code generation task: the LLM produces executable scripts (e.g., using RDKit (Landrum, 2013)) that specify verifiable structural modifications. This approach leverages the strengths of LLMs and enables faithful execution of editing intentions to improve controllability, reproducibility, and transparency in molecular design.

We introduce **MECO** (Molecular Editing via Code generation), a framework that translates high-level design rationales into structured editing programs. We fine-tune a compact code-oriented language model solely on synthetic data generated by limited moiety substitutions on random molecules. The model achieves 98% accuracy on realistic edits derived from reaction and bioactivity datasets, significantly outperforming a same-size general-purpose language model fine-tuned to directly generate SMILES. When using a reasoning LLM such as DeepSeek-R1 (Guo et al., 2025) as the upstream component, MECO substantially improves the consistency between editing intentions and resulting structures by 38–86 percentage points to 90%+, leading to higher success rates while preserving structural similarity across multiple property and activity optimization benchmarks.

By bridging natural language reasoning and structural molecular modification through code, MECO enables consistent, controllable and interpretable molecular modification, bringing LLM-guided molecule design closer to real-world scien-

tific application. Our main contributions are:

- **Code-based formulation for molecular modification.** We introduce MECO, a novel framework that recasts molecular editing as a code generation task, enabling LLMs to translate natural language intentions into precise, verifiable, and executable modifications.
- **Scalable data construction for training and evaluation.** We develop a scalable pipeline for constructing both synthetic and realistic editing samples, combining programmatic moiety replacement with edit extraction from chemical reactions and bioactive molecules.
- **Generalization to realistic molecular transformations.** We show that a code LLM trained solely on synthetic edits generalizes effectively to real-world modifications, achieving over 98% accuracy on both reaction- and activity-derived edits.
- **Superior molecule optimization performance.** MECO outperforms direct generation baselines across property and activity benchmarks, with notably *double the structure–intention consistency*, enhancing interpretability and reliability.

2 Related work

Despite the widespread adoption of SMILES for sequence-based molecular representation (Weininger, 1988; Gómez-Bombarelli et al., 2018; Segler et al., 2018; Schwaller et al., 2019), it poorly aligns with natural language because small structural modifications can cause

disproportionately large string differences, and it suffers from syntactic non-uniqueness for a single molecule (Merz Jr et al., 2020). Conversely, while graph-based optimization approaches (Jin et al., 2018; Shi et al., 2020; Jin et al., 2020; Vignac et al., 2023) bypass string syntax constraints, they fundamentally rely on oracle-driven feedback (Gao et al., 2022) to explore the property landscape.

LLMs have been investigated as general-purpose optimizers (Yang et al., 2024; Meyerson et al., 2024; Liu et al., 2024b) and extended to this domain through prompt-based methods that propose modifications using external databases (Wang et al., 2025; Liu et al., 2024a). To enhance generation quality, other frameworks leverage representation learning (Ranković and Schwaller, 2023; Bedrosian et al., 2024; Fang et al., 2024b; Kristiadi et al., 2024) or instruction tuning on large-scale datasets (Ye et al., 2025; Nguyen and Grover, 2025). Despite these advances, most existing systems still directly generate final molecular representations from high-level objectives or reasoning, while the interface between high-level molecular reasoning and low-level structural manipulation remains underdeveloped. MECo bridges this gap by introducing code as an execution interface between chemical reasoning and molecular graphs, enabling structured editing intentions to be translated into consistent, verifiable, and controllable molecular modifications.

3 Methods

3.1 Problem formulation

Molecular optimization is a central task in drug discovery, where the objective is to generate a modified molecule M_o from an initial compound M_i to improve one or more target properties T (e.g., permeability, target binding affinity), while preserving essential structural features such as the core scaffold or pharmacophores, and enabling reuse of steps in a common synthetic route.

In many prior approaches, particularly those based on RNNs or early LLMs, this task is formulated as:

$$M_o = \mathcal{F}(M_i, T) \quad (1)$$

where $\mathcal{F}$ is a black-box model that directly maps the input molecule and target to a SMILES string.

While this formulation has shown some empirical success, it suffers from two key limitations. First, the editing process is entirely implicit: the model does not explain what was changed or why,

making the transformation uninterpretable. Second, the output often diverges from well-established medicinal chemistry principles. These models tend to make broad, unconstrained modifications, rather than the minimal, targeted edits that chemists use, such as modifying a functional group to adjust polarity for better permeability, or introducing an H-bond donor to improve affinity and selectivity. Without such structure–property reasoning, the results are hard to interpret or trust, limiting their practical utility in real-world design workflows.

In contrast, recent reasoning-centric LLMs offer a fundamentally different capability: rather than directly generating M_o , they can first articulate a structured editing intention based on M_i and T . These intentions often consist of: **a set of editing actions** $A = \{a_1, a_2, \dots\}$ (e.g., replace the para-methyl with a hydroxyl group), and **a corresponding rationale** (e.g., to increase polarity and improve solubility).

This shift from black-box generation to interpretable, rationale-driven editing actions more closely reflects how medicinal chemists reason about molecular design based on structural and physicochemical considerations. The corresponding formulation becomes:

$$(A, M_o) \leftarrow \mathcal{F}(M_i, T) \quad (2)$$

However, in current systems, this promising capability remains underutilized. While reasoning LLMs can propose chemically meaningful edits, they often fail to execute them faithfully. This is largely due to the difficulty of performing precise and constrained modifications on SMILES representations, which are sensitive to minor syntax errors and lack structural locality. As a result, the generated molecule M_o may deviate from the proposed intention or violate chemical constraints.

To bridge this gap between reasoning and execution, we reformulate molecular editing as a code generation task:

$$\mathcal{C} = \mathcal{G}(M_i, A), \quad M_o = \mathcal{C}(M_i) \quad (3)$$

where $\mathcal{G}$ denotes the code generation model (e.g., a coder LLM), and $\mathcal{C}$ is an executable script (e.g., using RDKit) that implements the specified editing actions A .

Why code as an interface? LLMs have demonstrated strong performance in translating natural language into structured domain-specific languages

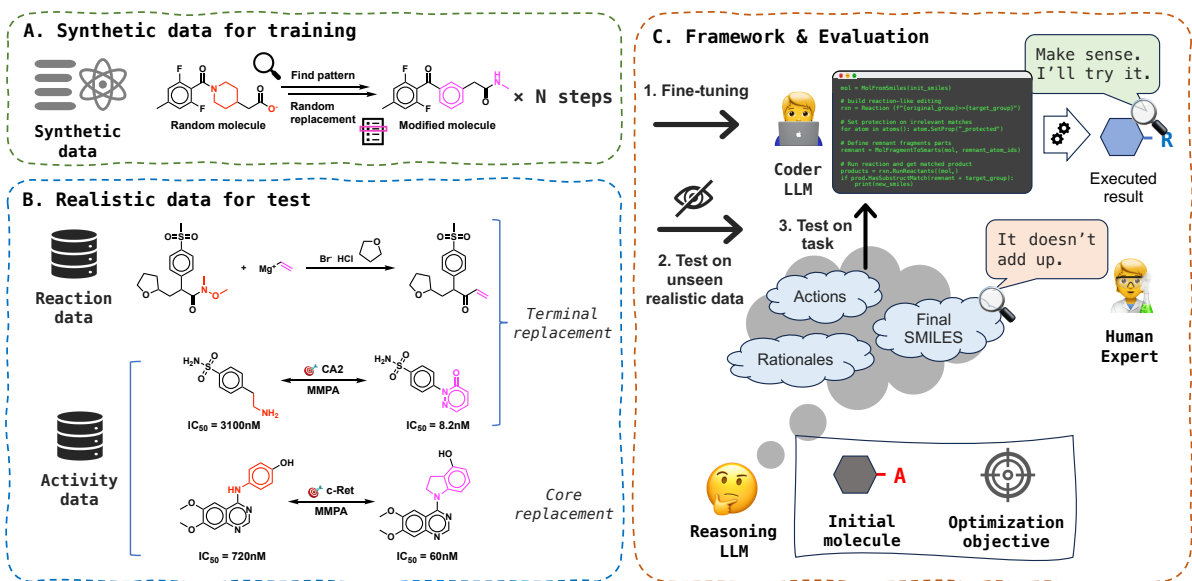


Figure 2: Overview of the MECo framework. (A) Synthetic data construction for model training. (B) Realistic data for testing on unseen molecular modifications derived from both reaction and activity data, reflecting chemical transformations observed in practice and in human reasoning. Modifications are categorized into terminal or core replacements. (C) End-to-end workflow and evaluation of the framework. The coder LLM is evaluated both independently and within the integrated pipeline. A reasoning LLM first generates rationales and editing actions, which are then executed by the coder model to produce optimized molecules. Outputs are subsequently reviewed by human experts to assess the alignment between generated structures and intended editing actions.

such as programming code (Fang et al., 2024a). Models like CodeT5+ (Wang et al., 2023) and StarCoder (Li et al., 2023) can reliably generate and edit code given natural language instructions. This success is attributed to the syntactic and semantic regularity of code, and structure-aware strategies such as abstract syntax trees (Wang et al., 2021), fill-in-the-middle training (Li et al., 2023), and execution-based feedback (Li et al., 2022).

This motivates us to treat molecular editing as a code generation task. Rather than generating fragile SMILES strings, the LLM produces executable scripts that specify structural edits. Notably, cheminformatics libraries like RDKit (Landrum, 2013) provide robust APIs for manipulating molecular graphs, allowing SMILES strings to be parsed into graph-based data structures and modified programmatically. This enables verifiable and reproducible molecular transformations grounded in chemical structure, not string syntax.

3.2 Framework overview

We propose **MECo** (Molecular Editing via Code generation), a framework that decouples high-level chemical reasoning from low-level molecular editing, as illustrated in Figure 2C. Given an initial molecule M_i and a target property goal T , a rea-

soning LLM first generates an editing intention (A , rationale), where A denotes a set of editing actions and the rationale explains their purpose. The focus shifts to executing A through code generation, producing the optimized molecule M_o .

MECo operates in a cascading manner:

Intention generation (reasoning LLM): Generates editing actions A and associated rationales based on (M_i, T) .

Action execution (code LLM): Translates A into code C , which is applied to M_i to yield M_o .

$$\underbrace{(M_i, T) \longrightarrow (A, \text{rationale})}_{\text{reasoning LLM}} \longrightarrow \underbrace{C = \text{CodeGen}(A)}_{\text{code LLM}} \longrightarrow M_o = C(M_i) \quad (4)$$

This decoupled formulation offers several advantages: (1) It separates reasoning (*why* to edit) from execution (*how* to edit), mirroring expert design workflows. (2) It provides interpretable, verifiable, and debuggable records of molecular transformation. (3) It ensures structural validity and fine-grained control by grounding edits in explicitly defined procedures.

By translating natural language intentions into code-level structural edits, MECo enables LLM-driven molecular design that is more reliable, trans-

parent, and amenable to practical drug-discovery workflows.

3.3 Data construction

To train the code LLM to translate editing actions into executable molecular transformations, we construct a large-scale dataset of molecule–edit–code triples in the form:

$$(M_i, A) \longrightarrow \mathcal{C} \quad (5)$$

where M_i is the input molecule, A is a set of editing actions expressed in natural language, and $\mathcal{C}$ is the corresponding Python code that applies the edit using cheminformatics toolkits such as RDKit.

We adopt a hybrid data construction strategy combining synthetic and realistic samples:

Synthetic edits. We constructed synthetic editing examples by applying programmatically defined moiety replacements to molecules randomly sampled from the ZINC database (Sterling and Irwin, 2015), as shown in Figure 2A. Each initial molecule M_i was iteratively edited by substituting one of its substructures with a moiety from a predefined pattern pool $\mathcal{P} = \bigcup_i \mathcal{P}_i$, where each subset $\mathcal{P}_i$ contains fragments with i attachment points.

This procedure yields chemically valid and structurally controlled synthetic edits, providing precise edit-code supervision that enables the model to learn robust mappings from actions to code without requiring exposure to real-world editing data. The full method and list of predefined patterns are provided in Appendix A.1.

Realistic edits. To evaluate generalization beyond synthetic edits, we construct a set of realistic editing samples derived from two sources (Figure 2B, detailed in Appendix A.2): (1) Reaction-derived edits, where we retained samples with a single modification site while preserving the molecular core. (2) Target-based edits, where we extracted target-specific compounds from the ChEMBL35 database and processed with matched molecular pairs analysis. The data were categorized into two types: **terminal replacements** (single-point attachment, same as those in **Reaction-derived edits**) and **core replacements** (multi-point attachment).

Together, these two sources reflect the two most common strategies in real-world molecular design. The first captures reaction-derived transformations, grounded in feasible chemical synthesis steps. The second comprises target-specific structural modifications observed across bioactive compounds,

which represent plausible edits made during lead optimization. While the directionality of property change is not explicitly defined in our setup, these samples provide chemically meaningful, human-curated edits. Although our model is trained solely on synthetic data, evaluation on these realistic edits allows us to assess its generalization to authentic molecular transformation scenarios.

3.4 Model Training

To train the coder LLM for accurate molecular editing, we used a curated dataset of 50,000 synthetic editing samples (Section 3.3). Each sample includes a molecule M_i , an editing action A described using a combination of natural language and cheminformatics notation (e.g., SMARTS), and a Python code snippet $\mathcal{C}$ that transforms M_i accordingly. For training the direct SMILES generation baseline, the corresponding output molecules $M_o = \mathcal{C}(M_i)$ were used as targets. All models were fine-tuned using the official Qwen2.5-Coder finetuning framework. Additional details regarding prompt design, formatting choices, and training settings are provided in Appendix B.

3.5 Benchmark and evaluation metrics

To evaluate both the core editing capability and downstream utility of our framework, we introduced two complementary benchmarks aligned with our problem formulation:

1. Molecular editing benchmark for problem diagnosis. We used the realistic editing samples constructed in Section 3.3 to identify the limitations of existing approaches and to quantitatively evaluate the molecular editing capabilities of MECo. For each sample, we assess whether the model can faithfully execute the specified editing action A on the input molecule M_i , either directly or via code generation, to produce the expected output molecule M_o .

2. Molecular modification benchmark for application endpoint. To assess the practical utility of our framework, we evaluate end-to-end performance on ChemCoTBench (Hao et al., 2025). This benchmark comprises six molecular modification tasks: three involving physicochemical properties and three involving target-specific bioactivities. We compare direct molecule generation from a reasoning LLM with the MECo framework in terms of (1) chemical validity, (2) optimization success rate, (3) mean similarity to source molecules, and (4) structure–action consistency.

Action	Model	Terminal replacement		Core replacement
		Reaction-derived	Target-specific	Target-specific
Direct	Qwen2.5-7B-Instruct	0.5	1.3	0.0
	Qwen2.5-7B-Instruct-FT	38.7	7.4	2.7
CodeGen	Qwen2.5-7B-Instruct	35.4	3.7	4.1
	Qwen2.5-Coder-7B-Instruct	50.3	9.9	12.2
	Qwen2.5-Coder-7B-Instruct-FT (ours)	99.9	98.3	98.3

Table 1: Execution accuracy (%) across realistic edit benchmarks. Rows are shaded to distinguish model variants: Qwen2.5, Qwen2.5-Coder, and finetuned versions (darker, denoted by -FT).

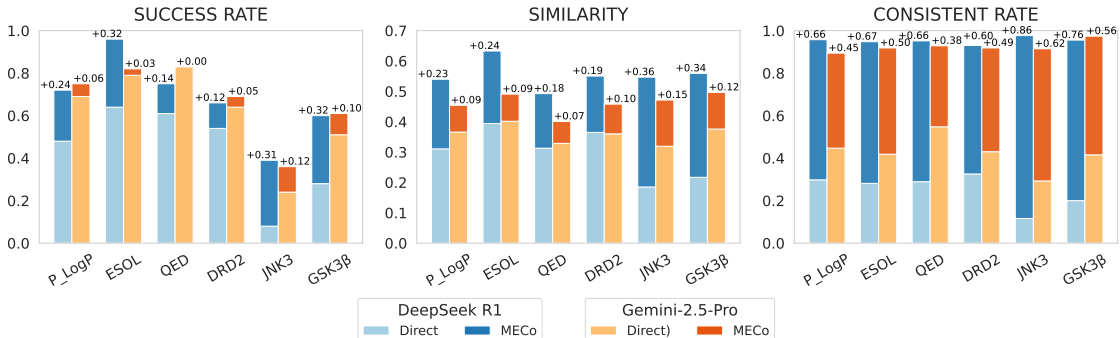


Figure 3: Incremental bar plots comparing direct generation (light bars) and MECo (dark stacked bars) across tasks and metrics under two different reasoning LLMs. Numbers above the bars indicate the absolute improvements of MECo over direct.

4 Experiments

4.1 Experimental setup

In the molecular modification task under the MECo framework, we applied a unified wrapper to the original prompt to constrain the output format and facilitate reliable extraction of editing actions. The full wrapper template is provided in Appendix C.2. To obtain ground-truth labels, we manually applied the actions generated by the reasoning LLMs to the initial molecules, following the evaluation criteria described in Appendix C.3.

We selected DeepSeek-R1 (Guo et al., 2025) as the reasoning LLM used in MECo by default, due to its strong performance in structured scientific reasoning and its fully open-source availability, which facilitates reproducibility and downstream integration. Gemini-2.5-Pro (Comanici et al., 2025) was also evaluated for comparison, given its top performance on ChemCoTBench. We employed Qwen2.5-7B (Yang et al., 2025) to showcase an open-source implementation of the edit execution model, fine-tuning its Coder variant (Hui et al., 2024) for code-based editing, and its general-purpose variant for direct SMILES generation.

4.2 Edit execution on realistic benchmark

To ensure a fair evaluation of editing generalization, we removed all samples from each realistic dataset whose original or modified fragments had a Tanimoto similarity ≥ 0.6 (computed using ECFP4 (Rogers and Hahn, 2010)) with any moiety used in the synthetic training set. From the remaining pool, we randomly sampled 1,000 examples per category: reaction-derived, target-specific terminal replacement, and target-specific core replacement, as the test sets. This setup ensures that the model is evaluated on structurally diverse and unseen edits, providing a robust assessment of its generalization ability. Results are summarized in Table 1. Appendix Figure 6 further visualizes the fragment distributions via t-SNE, showing the broad and largely non-overlapping chemical space of realistic edits.

This comparison demonstrates the advantage of code-based molecular editing over direct SMILES generation, independent of both string and structural similarities (see Appendix Figure 7). The direct generation model Qwen2.5-7B-Instruct exhibits extremely low execution accuracy across all categories, suggesting a limited ability to faithfully apply structural modifications. Notably, even without specialized training, prompting the same gen-

Property optimization tasks												
Model	Penalized logP				Solubility				QED			
	VR%	SR%	Sim	CR%	VR%	SR%	Sim	CR%	VR%	SR%	Sim	CR%
<i>W/o thinking</i>												
GPT-4o	22	8	0.11	–	41	35	0.21	–	31	17	0.11	–
Gemini-2.0-flash	66	44	0.27	–	32	28	0.13	–	64	54	0.26	–
DeepSeek-V3	53	21	0.20	–	52	44	0.21	–	48	33	0.18	–
<i>W/ thinking</i>												
GPT-5	58	32	0.21	–	67	63	0.25	–	68	66	0.21	–
Claude-Sonnet-4.6	71	67	0.28	–	66	66	0.28	–	74	70	0.32	–
Gemini-2.5-Pro	78	69	0.37	45	79	79	0.40	42	83	83	0.33	55
DeepSeek-R1	62	48	0.31	30	70	64	0.39	28	68	61	0.31	29
MECo	93	72	0.54	96	96	96	0.63	95	87	75	0.49	95
Activity optimization tasks												
Model	DRD2				JNK3				GSK3 β			
	VR%	SR%	Sim	CR%	VR%	SR%	Sim	CR%	VR%	SR%	Sim	CR%
<i>W/o thinking</i>												
GPT-4o	27	12	0.12	–	20	4	0.08	–	19	10	0.08	–
Gemini-2.0-flash	60	32	0.29	–	56	17	0.25	–	71	39	0.36	–
DeepSeek-V3	46	22	0.20	–	40	9	0.14	–	43	18	0.15	–
<i>W/ thinking</i>												
GPT-5	64	50	0.20	–	57	15	0.17	–	46	28	0.14	–
Claude-Sonnet-4.6	67	53	0.28	–	39	9	0.13	–	56	40	0.20	–
Gemini-2.5-Pro	85	64	0.36	43	74	24	0.32	42	74	51	0.38	29
DeepSeek-R1	73	54	0.36	33	43	8	0.19	12	45	28	0.22	20
MECo	89	66	0.55	93	92	39	0.55	98	91	60	0.56	96

Table 2: Performance on molecular optimization tasks. Each task reports validity rate (VR%), success rate (SR%), mean similarity to the source molecule (Sim), and consistency rate (CR%). **Bold** numbers highlight the best value in each column. Note: GPT-5 refused to answer a small number of samples.

eral LLM to generate code leads to substantially higher accuracy than its direct generation counterpart, underscoring the inherent advantages of structured code over direct SMILES generation. Interestingly, we find that the discrepancy between reaction-derived and target-specific terminal replacements can be largely attributed to differences in SMILES syntax patterns, suggesting that LLMs struggle to generalize over SMILES syntax in both direct generation and code generation. A detailed discussion is provided in Appendix C.4.

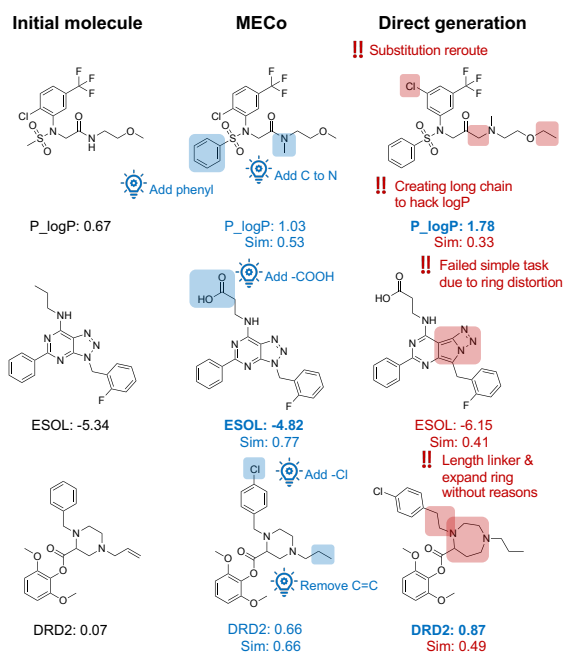
With supervised fine-tuning, the code-specialized model Qwen2.5-Coder-7B-Instruct-FT achieves over 98% execution accuracy across all benchmarks, while direct generation shows only marginal improvement. These results affirm that MECo’s code generation formulation dramatically improves execution fidelity, even on realistic and diverse editing tasks.

4.3 Improvement on molecular modification

Table 2 summarizes performance across six molecular optimization tasks, covering both physicochemical properties and target activities. We compare several non-reasoning / reasoning LLMs with our proposed MECo framework, which augments reasoning-centric molecular modification with code generation for edit execution. We omit mean property improvement from the main results, since models can achieve large gains by sacrificing structural preservation and producing molecules that differ substantially from the initial ones. Such cases inflate the average improvement without reflecting meaningful optimization. We therefore adopt success rate, and provide the mean improvement results in Appendix C.7.

MECo achieves markedly higher consistency rates (CR%), indicating that the resulting structures more faithfully realize the editing intentions proposed by the reasoning LLM. This closer intention–

(A) Failed Direct Generation



(B) Successful MECO Optimization

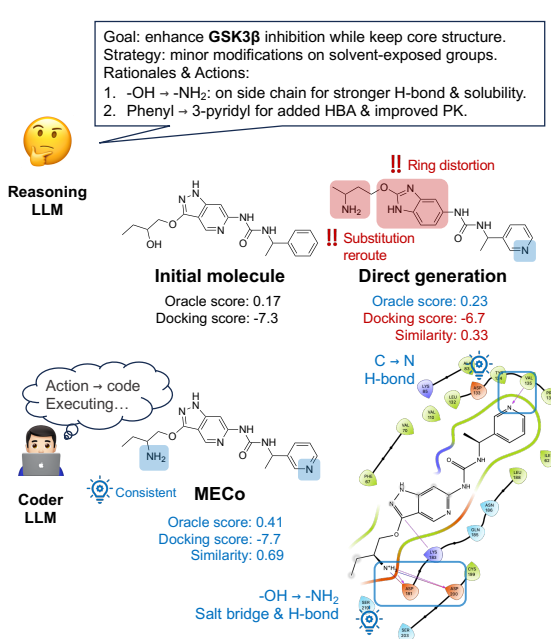


Figure 4: Representative case study of molecular modification. (A) Direct generation produces uncontrolled changes and low similarity, often leading to unrealistic score inflation or failed optimization; MECo achieves improved scores and higher similarity through interpretable, chemistry-preserving edits. (B) MECo demonstrates coherent consistency across the reasoning–execution–analysis pipeline.

structure alignment improves the transparency and interpretability of the editing process by making the intended modifications explicit and their structural realization directly verifiable. Together with the gains in validity, success rate, and structural similarity, these results suggest that more faithful execution helps translate editing strategies proposed by the reasoning LLMs into valid, successful, and structurally conservative molecular modifications.

In contrast, direct SMILES generation often produces unintended or overly disruptive modifications, with poor alignment to the specified editing instructions (see cases in Section 4.4). Even strong reasoning LLMs fall short when lacking explicit action execution, underscoring the need to bridge high-level reasoning and low-level structural manipulation through code-based editing.

Figure 3 further shows that MECo’s improvements generalize across reasoning LLMs (DeepSeek-R1 and Gemini-2.5-Pro), demonstrating its broad applicability and utility in controllable, interpretable molecule design.

4.4 Case study

To illustrate the difference between direct generation and our MECo framework, we analyze molec-

ular modification tasks and present representative cases in Figure 4A. Direct generation often introduces uncontrolled structural changes with low similarity to the initial molecule, leading to unrealistic score inflation or failed optimization. In contrast, MECo achieves improved scores while maintaining high similarity through interpretable, chemistry-preserving edits.

The detailed example on the GSK3 β activity optimization task in Figure 4B further demonstrates how MECo maintains consistency across the reasoning–execution–analysis pipeline. The reasoning LLM proposes rational modifications, such as replacing a hydroxy with an amino to strengthen hydrogen bonding and improve solubility, and substituting a phenyl ring with a pyridin-3-yl ring to introduce an additional hydrogen bond acceptor and enhance pharmacokinetic potential. The coder LLM then faithfully translates these actions into executable edits, ensuring that the intended modifications are precisely applied. This yields improved oracle and docking scores, while complementary binding analysis further supports the rationales. The coherent consistency underpins the interpretability and reliability of MECo compared to direct generation.

5 Conclusion

We introduce MECo, a framework that reformulates molecular modification as a code generation task, bridging high-level reasoning with low-level structural execution. By translating interpretable editing intentions into executable scripts, MECo achieves near-perfect edit fidelity on realistic benchmarks and consistently outperforms direct-SMILES-generation baselines across both physicochemical property and bioactivity optimization tasks. This formulation enables consistent, controllable and interpretable molecular modification workflows, paving the way for reliable human-AI collaboration and future high-fidelity feedback loops in iterative molecular design workflows, such as agent-style extensions and multi-round optimization. Beyond this application, our results suggest a general paradigm for bridging reasoning and execution in scientific domains, highlighting the potential of LLMs as reliable assistants for structured, verifiable discovery workflows.

Limitations

Our work decouples molecular reasoning from structural execution, but its end-to-end performance remains bounded by the quality of the upstream reasoning LLM: if the reasoning model proposes chemically plausible but unreasonable editing actions, the coder LLM will execute them faithfully rather than correct them. This propagation of upstream errors is an inherent consequence of the cascaded design.

While multi-step replacements are sufficient to express virtually any structural modification at the molecular level, certain medicinal chemistry design strategies—such as ring opening/closure and bond rerouting—can be somewhat unintuitive when represented purely as replacements. Extending the framework to more naturally accommodate these transformations could better align model representations with expert medicinal chemistry priors, potentially improving the utilization of domain knowledge and the quality of generated modifications, which is important for practical deployment and system optimization.

The consistency rate (CR%) is a metric specifically designed to evaluate intention–execution fidelity and is not intended as an independent measure of molecular or pharmacological optimization quality. Details on its manual evaluation, subjectivity, and scalability are provided in Appendix C.3.

Acknowledgments

This work is supported by Innovative Drug Research and Development-National Science and Technology Major Project (No.2025ZD1802501) and Beijing Frontier Research Center for Biological Structure Fundings.

References

- Menua Bedrosian, Philipp Guevorguian, Tigran Fahradyan, Gayane Chilingaryan, Hrant Khachatrian, and Armen Aghajanyan. 2024. Small molecule optimization with large language models. In *Neurips 2024 Workshop Foundation Models for Science: Progress, Opportunities, and Challenges*.
- G. Richard Bickerton, Gaia V. Paolini, J  r  my Besnard, Sorel Muresan, and Andrew L. Hopkins. 2012. [Quantifying the chemical beauty of drugs](#). *Nature Chemistry*, 4(2):90–98.
- Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. [Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities](#). *arXiv preprint*.
- John S Delaney. 2004. ESOL: estimating aqueous solubility directly from molecular structure. *Journal of chemical information and computer sciences*, 44(3):1000–1005.
- Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun. 2024a. Large language models for code analysis: do LLMs really do their job? In *Proceedings of the 33rd USENIX Conference on Security Symposium, SEC ’24*, pages 829–846, USA. USENIX Association.
- Yin Fang, Ningyu Zhang, Zhuo Chen, Lingbing Guo, Xiaohui Fan, and Huajun Chen. 2024b. [Domain-agnostic molecular generation with chemical feedback](#). In *The Twelfth International Conference on Learning Representations*.
- Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor Coley. 2022. Sample efficiency matters: a benchmark for practical molecular optimization. *Advances in neural information processing systems*, 35:21342–21357.
- Rafael G  mez-Bombarelli, Jennifer N Wei, David Duvenaud, Jos   Miguel Hern  ndez-Lobato, Benjam  n S  nchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Al  n Aspuru-Guzik. 2018. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276.

- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. [DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning](#). *Nature*, 645(8081):633–638.
- Li Hao, He Cao, Bin Feng, Daniel Shao, Robert Tang, Zhiyuan Yan, Yonghong Tian, Li Yuan, and Yu Li. 2025. Beyond Chemical QA: Evaluating LLM’s chemical reasoning with modular chemical operations. In *Advances in Neural Information Processing Systems*, volume 38.
- Yu Hou, Zaifu Zhan, and Rui Zhang. 2025. [Benchmarking GPT-5 for biomedical natural language processing](#). *arXiv preprint*.
- Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. 2021. Therapeutics Data Commons: Machine Learning Datasets and Tasks for Drug Discovery and Development. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 1.
- Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.
- Jameed Hussain and Ceara Rea. 2010. [Computationally Efficient Algorithm to Identify Matched Molecular Pairs \(MMPs\) in Large Data Sets](#). *Journal of Chemical Information and Modeling*, 50(3):339–348.
- Wengong Jin, Dr Regina Barzilay, and Tommi Jaakkola. 2020. [Multi-Objective Molecule Generation using Interpretable Substructures](#). In *Proceedings of the 37th International Conference on Machine Learning*, pages 4849–4859. PMLR.
- Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR.
- Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. 2017. Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in neural information processing systems*, volume 30.
- Agustinus Kristiadi, Felix Strieth-Kalthoff, Marta Skreta, Pascal Poupart, Alán Aspuru-Guzik, and Geoff Pleiss. 2024. A sober look at llms for material discovery: are they actually good for bayesian optimization over molecules? In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24. JMLR.org.
- Greg Landrum. 2013. [RDKit: Open-source cheminformatics software](#).
- Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. [Solving quantitative reasoning problems with language models](#). In *Advances in Neural Information Processing Systems*.
- Mingxuan Li, Hanchen Li, and Chenhao Tan. 2026. HypoEval: Hypothesis-guided evaluation for natural language generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 42424–42443.
- R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shli-azhko, and 49 others. 2023. [StarCoder: May the source be with you!](#) *Transactions on machine learning research*.
- Yibo Li, Liangren Zhang, and Zhenming Liu. 2018. [Multi-objective de novo drug design with conditional graph generative model](#). *Journal of Cheminformatics*, 10(1):33.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.
- Yen-Ting Lin and Yun-Nung Chen. 2023. [LLM-Eval: Unified Multi-Dimensional Automatic Evaluation for Open-Domain Conversations with Large Language Models](#). In *Proceedings of the 5th Workshop on NLP for Conversational AI (NLP4ConvAI 2023)*, pages 47–58, Toronto, Canada.
- Shengchao Liu, Jiong Xiao Wang, Yijin Yang, Chengpeng Wang, Ling Liu, Hongyu Guo, and Chaowei Xiao. 2024a. Conversational drug editing using retrieval and domain feedback. In *The twelfth international conference on learning representations*.
- Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. 2024b. Large language models to enhance bayesian optimization. In *International Conference on Learning Representations*, volume 2024, pages 31252–31284.
- David Mendez, Anna Gaulton, A. Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F. Mosquera, Prudence Mutowo, Michał Nowotka, María Gordillo-Marañón, Fiona Hunter, Laura Junco, Grace Mugumbate, Milagros Rodriguez-Lopez, Francis Atkinson, Nicolas Bosc, Chris J. Radoux, Aldo Segura-Cabrera, and 2 others. 2019. [ChEMBL: towards direct deposition of bioassay data](#). *Nucleic Acids Research*, 47(D1):D930–D940.

- Kenneth M Merz Jr, Gianni De Fabritiis, and Guo-Wei Wei. 2020. Generative models for molecular design. *Journal of Chemical Information and Modeling*, 60(12):5635–5636.
- Elliot Meyerson, Mark J Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K Hoover, and Joel Lehman. 2024. Language model crossover: Variation through few-shot prompting. *ACM Transactions on Evolutionary Learning*, 4(4):1–40.
- Tung Nguyen and Aditya Grover. 2025. Lico: Large language models for in-context molecular optimization. In *International Conference on Learning Representations*, volume 2025, pages 53576–53597.
- Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. 2017. [Molecular de-novo design through deep reinforcement learning](#). *Journal of Cheminformatics*, 9(1):48.
- Bojana Ranković and Philippe Schwaller. 2023. Bochemian: Large language model embeddings for bayesian optimization of chemical reactions. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*.
- David Rogers and Mathew Hahn. 2010. [Extended-Connectivity Fingerprints](#). *Journal of Chemical Information and Modeling*, 50(5):742–754.
- Philippe Schwaller, Teodoro Laino, Théophile Gaudin, Peter Bolgar, Christopher A Hunter, Costas Bekas, and Alpha A Lee. 2019. Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, 5(9):1572–1583.
- Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. 2018. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131.
- Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. 2020. [GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation](#). In *the 8th International Conference on Learning Representations*.
- Teague Sterling and John J. Irwin. 2015. [ZINC 15 – Ligand Discovery for Everyone](#). *Journal of Chemical Information and Modeling*, 55(11):2324–2337.
- Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. 2023. [DiGress: Discrete Denoising diffusion for graph generation](#). In *The Eleventh International Conference on Learning Representations*.
- Haorui Wang, Marta Skreta, Cher-Tian Ser, Wenhao Gao, Ling kai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, and 1 others. 2025. Efficient evolutionary search over chemical space with large language models. In *International Conference on Learning Representations*, volume 2025, pages 51694–51727.
- Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023. CodeT5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 1069–1088.
- Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 8696–8708.
- David Weininger. 1988. SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2025. [Qwen2.5 Technical Report](#). *arXiv preprint*.
- Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.
- Geyan Ye, Xibao Cai, Houtim Lai, Xing Wang, Junhong Huang, Longyue Wang, Wei Liu, and Xiangxiang Zeng. 2025. Drugassist: A large language model for molecule optimization. *Briefings in Bioinformatics*, 26(1):bbae693.
- Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. 2024. [FLASK: Fine-grained Language Model Evaluation based on Alignment Skill Sets](#). In *The Twelfth International Conference on Learning Representations*.
- Di Zhang, Wei Liu, Qian Tan, Jingdan Chen, Hang Yan, Yuliang Yan, Jiatong Li, Weiran Huang, Xianguyu Yue, Wanli Ouyang, Dongzhan Zhou, Shufei Zhang, Mao Su, Han-Sen Zhong, and Yuqiang Li. 2024. ChemLLM: A chemical large language model. *arXiv preprint arXiv:2402.06852*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in neural information processing systems*, volume 36, pages 46595–46623.
- Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. [Instruction-Following Evaluation for Large Language Models](#). *arXiv preprint*.

A Data construction details

A.1 Synthetic edit generation

We provide here the full algorithm for generating synthetic editing examples, as described in Section 3.3. The procedure applies pre-defined moiety replacements to randomly sampled molecules from the ZINC database, using structured pattern pools grouped by attachment point count.

Algorithm 1 Iterative Moiety Replacement

Require: Molecule M_i , pattern pool $\mathcal{P} = \bigcup_i \mathcal{P}_i$, iterations N

```
1:  $M \leftarrow M_i$  {Initialize molecule}
2: for  $n = 1$  to  $N$  do
3:    $\mathcal{P}_{\text{avail}} \leftarrow \mathcal{P}$  {Reset available patterns}
4:    $\text{replaced} \leftarrow \text{False}$ 
5:   while  $\mathcal{P}_{\text{avail}} \neq \emptyset$  and  $\text{replaced} = \text{False}$  do
6:     Sample  $p \sim \mathcal{P}_{\text{avail}}$ ;  $\mathcal{P}_{\text{avail}} \leftarrow \mathcal{P}_{\text{avail}} \setminus \{p\}$ 
       {Draw and remove pattern}
7:      $\mathcal{S} \leftarrow \text{GETSUBSTRUCTMATCHES}(M, p)$ 
       {Identify all matching substructures}
8:     if  $\mathcal{S} \neq \emptyset$  then
9:        $s \sim \mathcal{S}$  {Randomly select a match site}
10:       $i \leftarrow \text{attachment\_count}(p)$  {Determine
        connection type}
11:       $r \sim \mathcal{P}_i \setminus \{p\}$  {Sample replacement
        with same connectivity}
12:      for  $a$  in  $M$  do
13:        if  $a \notin s$  then
14:          Protect( $a$ ) {Protect atoms not in
            the selected substructure match  $s$ }
15:        end if
16:      end for
17:       $M \leftarrow \text{replace}(M, p, r)$  {Apply re-
        placement using REACTION}
18:      Record  $(p, s, r)$  { $s$  uniquely specifies
        the matched substructure instance}
19:       $\text{replaced} \leftarrow \text{True}$ 
20:    end if
21:  end while
22:  if  $\text{replaced} = \text{False}$  then
23:    break {Stop further iterations if replace-
        ment failed}
24:  end if
25: end for
26: return  $M_o, \{(p_j, s_j, r_j)\}_{j=1}^n$  {Final molecule
    and list of edits}
```

At each iteration, a pattern p was randomly drawn from the pool, and a matching site in M_i was identified. If a match was found, it was

replaced with another randomly selected moiety $r \in \mathcal{P}_i \setminus \{p\}$ of the same connectivity. If no match was found, the process continued with a different pattern until the pool was exhausted. Replacement was performed iteratively, with all successful edits recorded. For each edit, the corresponding executable code snippet $\mathcal{C}$ was automatically generated, and the edited molecule M_o was obtained by applying $\mathcal{C}$ to the input molecule M_i , i.e., $M_o = \mathcal{C}(M_i)$.

The lists of predefined substituents (Table 3) and molecular linkers (Table 4) are provided for constructing synthetic editing samples.

A.2 Realistic edit generation

We construct a set of realistic editing samples derived from two sources:

(1) Reaction-derived edits: We extracted matched reactant-product pairs from the USPTO-MIT reaction dataset (Jin et al., 2017). Atom mapping was used to identify structural changes, and we retained samples with a single modification site while preserving the molecular core.

(2) Target-based edits: We extracted compounds from the ChEMBL35 database (Mendez et al., 2019) and grouped them by target ID after applying basic structure-based filters (details provided in Appendix A.3). Matched molecular pairs (MMPs) were identified using the algorithm by Hussain and Rea (2010), and further filtered following the criteria described in Appendix A.4. Attachment points were explicitly identified and recorded to enable precise modification tracking. Based on the number of attachment points in each transformation, the data were categorized into two types: **terminal replacements** (single-point attachment, same as those in **Reaction-derived edits**) and **core replacements** (multi-point attachment).

A.3 Compound preprocessing and filtering

We extracted standardized SMILES strings and associated bioactivity data from the ChEMBL35 database. To ensure molecular quality and downstream compatibility, we applied the following filtering criteria:

- Salt removal: only the largest fragment was retained;
- Molecular weight between 100 and 800 Da;
- Allowed atom types: H, C, N, O, F, Cl, Br, I, S, P, B, Se;

Category	Name	SMILES
Halogens	Fluoro	[*:1]F
	Chloro	[*:1]Cl
	Bromo	[*:1]Br
	Iodo	[*:1]I
Alkyl	Methyl	[*:1]C
	Ethyl	[*:1]CC
	Isopropyl	[*:1]C(C)C
	tert-Butyl	[*:1]C(C)(C)C
Aryl	Phenyl	[*:1]c1ccccc1
	p-Tolyl	[*:1]c1ccc(cc1)C
	p-Chlorophenyl	[*:1]c1ccc(cc1)Cl
Oxygen-containing	Hydroxyl	[*:1]O
	Methoxy	[*:1]OC
	Ethoxy	[*:1]OCC
	Carboxyl	[*:1]C(=O)O
	Aldehyde	[*:1]C=O
	Ketone	[*:1]C(=O)C
Nitrogen-containing	Amino	[*:1]N
	Methylamino	[*:1]NC
	Dimethylamino	[*:1]N(C)C
	Cyano	[*:1]C#N
	Nitro	[*:1][N+](=O)[O-]
Sulfur-containing	Thiol	[*:1]S
	Methylthio	[*:1]SC
	Sulfonyl	[*:1]S(=O)(=O)C

Table 3: Common substituents for synthetic data.

- No linear unbranched chains longer than six heavy atoms.

These filters resulted in a curated set of drug-like molecules, which served as the input for MMP generation.

A.4 Target grouping and MMP Analysis

After filtering, compounds were grouped by target ID, and each group was exported as a separate .smi file. For each group, matched molecular pairs (MMPs) were generated using the RDKit Contrib MMPA pipeline (rfrag.py and indexing.py). This procedure applies systematic bond fragmentation and indexing to identify molecular pairs that differ by a single localized transformation. Each transformation was encoded as a SMIRKS pattern, representing the minimal structural change. All MMP results were merged and formed a high-quality pool of target-specific matched molecular

pairs.

A.5 Transformation type classification

To better characterize the nature of molecular edits, the resulting MMPs were further categorized into two types based on structural properties:

1. Core replacement. MMPs were classified as core replacements if they satisfied the following criteria:

- **Attachment point constraint:** both fragments have the same number of attachment points, with count ≥ 2 .
- **Scaffold difference:** the Murcko scaffolds of the prior and latter fragments in replacement are different;
- **Ring requirement:** both fragments contain at least one ring;

Category	Name	SMILES
Aromatic linkers	Meta-phenylene	<chem>[*:1]c1cc([*:2])ccc1</chem>
	Para-phenylene	<chem>[*:1]c1ccc([*:2])cc1</chem>
	Ortho-phenylene	<chem>[*:1]c1c([*:2])cccc1</chem>
Carbonyl-based linkers	Amide	<chem>[*:1][C;!R](=O)[N;!R][*:2]</chem>
	Reverse amide	<chem>[*:1][N;!R][C;!R](=O)[*:2]</chem>
	Ester	<chem>[*:1][C;!R](=O)[O;!R][*:2]</chem>
	Ketone bridge	<chem>[*:1][C;!R](=O)[*:2]</chem>
	Urea	<chem>[*:1][N;!R][C;!R](=O)[N;!R][*:2]</chem>
	Carbamate	<chem>[*:1][O;!R][C;!R](=O)[N;!R][*:2]</chem>
	Sulfonamide	<chem>[*:1]S(=O)(=O)[N;!R][*:2]</chem>
Alkyl / heteroatom linkers	Methylene	<chem>[*:1][C;!R][*:2]</chem>
	Ethylene	<chem>[*:1][C;!R][C;!R][*:2]</chem>
	Ether	<chem>[*:1][O;!R][*:2]</chem>
	Thioether	<chem>[*:1][S;!R][*:2]</chem>
	Secondary amine	<chem>[*:1][N;!R][*:2]</chem>
Extended / heterocyclic linkers	1,2,3-Triazole	<chem>[*:1]c1nnn([*:2])c1</chem>
	Imidazole-type	<chem>[*:1]c1[nH]cc([*:2])n1</chem>
	Piperazine	<chem>[*:1]N1CCN([*:2])CC1</chem>
	Piperidine	<chem>[*:1]N1CCC([*:2])CC1</chem>
Polar chain linker	PEG unit (ethylene glycol)	<chem>[*:1][O;!R][C;!R][C;!R][O;!R][*:2]</chem>

Table 4: Common molecular linkers for synthetic data.

- **Non-ring atom constraint:** each fragment contains no more than 5 non-ring heavy atoms, including the attachment points, to avoid heavily decorated rings;
- **Component ratio:** the core fragment in the initial molecule accounts for less than 50% of the heavy atoms.

In cases where the original fragment exhibits symmetry and the replacement fragment does not (as illustrated in the last line of Figure 5), multiple distinct products may arise due to different permutations of attachment point mappings, accounting for 7.5% of our test set. Since the original attachment sites are chemically interchangeable, all such permutations are treated as valid ground truths for evaluation.

2. terminal replacement. MMPs were classified as single-point replacements (or terminal replacements) if the number of heavy atoms in the modified fragment accounts for no more than 30% of the initial molecule and only one attachment point.

A.6 t-SNE visualization of fragment chemical space

To assess the chemical diversity of the Realistic Edits test set relative to the synthetic moiety pool used in training, we computed ECFP fingerprints (radius = 2, 2048 bits) for all fragments and projected them into two dimensions using t-SNE. As shown in Figure 6, fragments from the synthetic training pool cover only a small portion of the chemical space and form several compact clusters, reflecting the limited structural motifs used to construct the synthetic edit pairs. In contrast, fragments from the Realistic Edits set are widely dispersed across the projection space, occupying diverse regions that are largely non-overlapping with the synthetic clusters.

This spread confirms that Realistic Edits cover a substantially broader and more diverse chemical space. Despite this distribution shift, the finetuned code LLM in MECo maintains strong performance, supporting our claim that the model learns generalizable code transformation patterns rather than relying on fragment-level memorization.

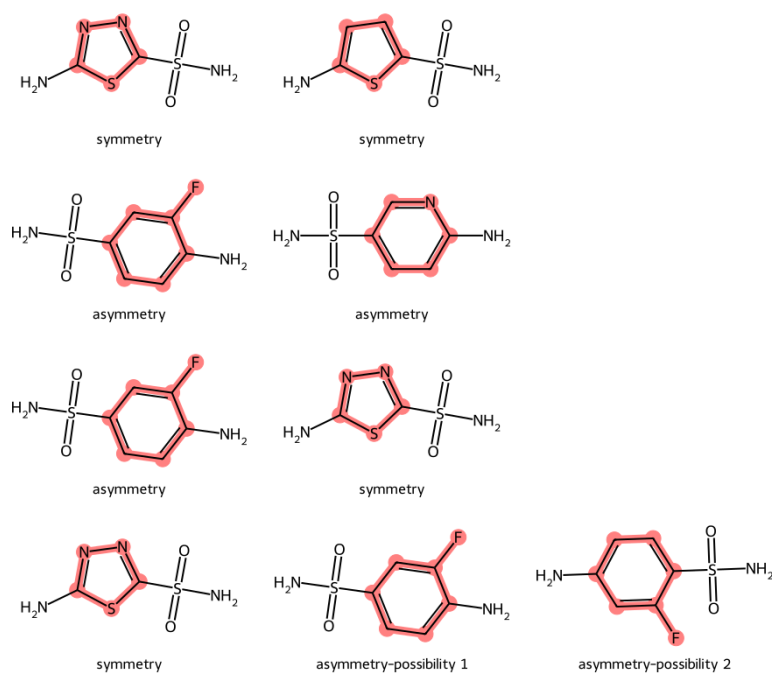


Figure 5: Multiple possible permutations in symmetry-to-asymmetry core replacement.

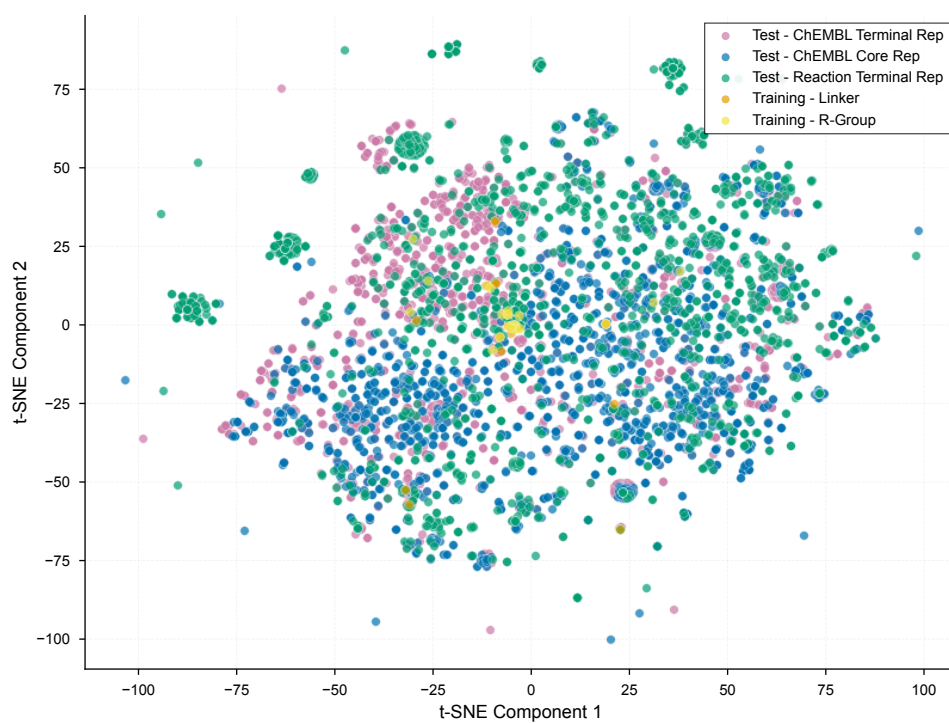


Figure 6: t-SNE for fragments in synthetic training data construction and realistic edit test set.

B Model training details

B.1 Prompt format and design rationale

To determine an optimal prompting strategy, we conducted preliminary experiments examining the effects of various formatting choices. These preliminary experiments were conducted under conditions that differ from our final evaluation tasks; they serve only as pilot studies to guide the choice of attachment-point style.

For numbering attachment points in the SMILES, we tested:

- Mark attachment points only in source SMILES with atom map numbers;
- Numerically number all atoms in SMILES and give the corresponding numbers of attachment points.

For marking attachment points in the fragment, we wrote original and target fragments with:

- atom map number, e.g., [C:1]N
- rooted at attachment point with a short dash, e.g., -CN
- an asterisk mark, or a so-called dummy atom, e.g., *CN

Preliminary results are shown in Table 5. While the asterisk symbol (*) was not always the most accurate in pilot trials, it was retained because of its conventional role in representing dummy atoms in cheminformatics toolkits (e.g., RDKit, ChemDraw) and its compatibility with code generation. We also observed that passing numbered SMILES directly to the reasoning LLM leads to more stable responses, likely due to the limited ability of LLMs to handle SMILES during reasoning. In contrast, the rooted attachment-point style is not compatible with fragments requiring multiple attachment points, such as in core-replacement tasks. Our final prompt for code-based molecular editing is listed in Table 6.

Generating both natural language rationales and SMARTS annotations introduces an additional structured representation requirement compared with using only natural-language editing intentions. However, this design does not simply transfer the difficulty of full-molecule SMILES generation to the intention stage. Instead, MECo decomposes molecular modification into short, local,

and indexed edit specifications followed by explicit graph-level execution.

Pure natural-language intentions may be easier to express, but they are not always directly executable without ambiguity. For example, "replace the hydroxyl group" is underspecified when multiple similar groups are present, while symmetric and multi-attachment transformations, such as replacing a heteroaromatic core, require explicit correspondence between attachment sites. SMARTS-style fragments and atom indices are therefore introduced to reduce execution ambiguity and provide direct compatibility with RDKit. A natural-language-only formulation would require an additional grounding step from free-text descriptions or chemical names to molecular substructures before execution, potentially introducing errors in name resolution, functional-group localization, and attachment mapping.

B.2 Training setup

We used the official fine-tuning framework from the Qwen2.5-Coder repository¹ and conducted full-parameter supervised fine-tuning (SFT) using DeepSpeed on both Qwen2.5-Coder-7B-Instruct and Qwen2.5-7B-Instruct models. The training was performed on 4 NVIDIA A100 80GB GPUs, with each epoch taking approximately 4 hours.

C Experiment details

C.1 Execution accuracy across fragment similarity bins

To further assess whether MECo generalizes beyond the structural motifs present in its synthetic training set, we evaluate execution accuracy across fragment similarity bins computed using both structural similarity (ECFP-Tanimoto) and string similarity (SequenceMatcher). Figure 7 reports accuracy curves over a wide range of similarity scores, covering multiple realistic edit test sources, including ChEMBL for activity-derived edits, USPTO for reaction-derived edits, and both terminal and core replacement types.

The results show that MECo maintains stable performance across the entire similarity spectrum, including bins where fragments are substantially dissimilar from those in the synthetic moiety pool.

¹<https://github.com/QwenLM/Qwen2.5-Coder/tree/main/finetuning/sft>

Action/Model	Molecule Number Style		Attachment Point	Whole SMILES
	Fragment Attachment Style			
CodeGen / Coder	Numbered	[C:1]N	0%	0%
	Rooted	-CN	20%	1%
	Asterisked	*CN	6%	6%
Direct / General	Numbered	[C:1]N	0%	3%
	Rooted	-CN	12%	6%
	Asterisked	*CN	4%	4%

Table 5: Accuracy across different SMILES attachment styles and numbering strategies.

This provides further evidence that MECo’s execution ability is tied to neither structural coverage nor string pattern overlap of fragments in the training data, but instead generalizes robustly to structurally diverse and previously unseen edits.

C.2 Implementation details for the molecular modification task

ChemCoTBench (Hao et al., 2025) comprises six molecular modification tasks: three involving physicochemical properties (penalized logP (Gómez-Bombarelli et al., 2018), solubility (Delaney, 2004), QED (Bickerton et al., 2012), implemented in RDKit (Landrum, 2013)) and three involving target-specific bioactivities (DRD2 (Olivecrona et al., 2017), JNK3, and GSK3 β (Li et al., 2018), implemented in TDC Oracles (Huang et al., 2021)). Each task provides a set of input molecules M_i paired with a shared optimization goal T . This setting reflects real-world design scenarios, where high-level property goals must be translated into concrete molecular modifications. For each (M_i, T) pair, we compare direct molecule generation from a reasoning LLM with the MECo framework, which first generates an editing intention and then executes it via code. This evaluation allows us to assess whether MECo improves over direct generation approaches in terms of (1) chemical validity; (2) optimization success rate; (3) mean similarity to source molecules; and (4) structure–action consistency, which is computed only for human-interpretable cases.

A prompt wrapper is used to instruct LLMs to generate modification schemes together with the underlying rationales, rather than merely producing SMILES strings, as shown in Table 7.

All dataset sampling, model training, and local inference used a fixed random seed of 42. For API-based models, we used a fixed seed of 42 and

greedy decoding where supported. GPT-5 and GPT-4o were accessed through Microsoft Azure OpenAI using API versions 2025-01-01-preview and 2024-08-01-preview, respectively. DeepSeek-R1-250528 and deepseek-v3-241226 were accessed through Volcengine Ark, and Gemini-2.5-Pro and Gemini-2.0-flash through Google Vertex AI. All unspecified API parameters were left at their service defaults.

To quantify evaluation uncertainty, we performed paired bootstrap resampling over the existing instance-level outputs. Macro-average improvements of MECo over direct generation from DeepSeek-R1 are 31.2 points in validity with 95% CI [26.7, 35.5], 24.2 in success [19.8, 28.5], 0.256 in similarity [0.231, 0.281], and 70.2 points in CR [66.2, 74.1]; all remain significant after Holm correction.

C.3 Manual evaluation of consistency

Criteria. We considered syntax errors in actions that are still discernible to human experts. Examples include malformed atom map annotations such as [$\ast$:1:] or [$\ast$:1], which should be [$\ast$:1], and missing explicit hydrogens on aromatic nitrogens, e.g., [$\ast$:1]c1nnnn1, which should be [$\ast$:1]c1[nH]nnn1.

For attachment point checks, we adopted relatively loose criteria. For instance, in [cH:1]1[cH:2][cH:3][cH:4][cH:5][c:6]1[O:7][CH2:8][CH:9]1[CH2:10][CH2:11]1, assigning the attachment of [$\ast$:1]OCC1CC1 to either atom 6 or 7 is considered discernible. Similarly, in [$\ast$:1]O[$\ast$:2], providing only one side of the atom index (e.g., 6 or 8) or the atom index alone (7) is also regarded as discernible. By contrast, edits that invoke groups not present in the structure are considered invalid, even if similar patterns can be found.

You are given a molecule in SMILES format:

"{numerically_numbered_source_smiles}".

For reference, atoms where new groups will be attached are marked with "[*:n]", where "n" is the atom mapping number.

Future connected atoms in groups are labeled using the same numbers, ensuring one-to-one attachment correspondence. You will then be given multiple instructions on how to edit the molecule.

Replace the substructure corresponding to
"{original_fragment_smarts_1}"
connected at atom {numbers_indicating_attachment_points}
with "{target_fragment_smarts_1}".

...

Replace the substructure corresponding to
"{original_fragment_smarts_n}"
connected at atom {numbers_indicating_attachment_points}
with "{target_fragment_smarts_n}".

Generate a Python code snippet that performs these replacements using RDKit via ChemicalReaction.

Ensure the code is executable and returns the modified molecule as a new SMILES string.

You must return a Python code snippet wrapped in triple backticks.

The code should import modules from RDKit, perform the operation, and print only the modified molecule in SMILES format.

Table 6: Final prompt for code-based molecular editing

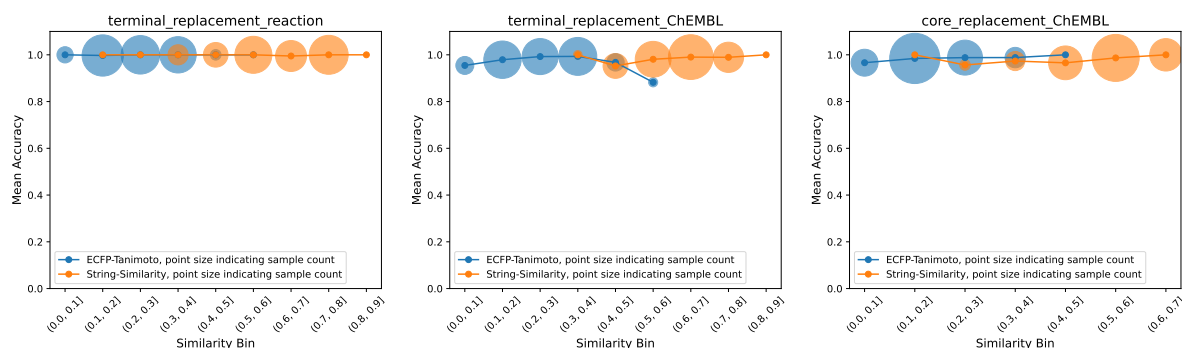


Figure 7: Execution accuracy of the finetuned code LLM (Qwen2.5-Coder-7B-Instruct-FT) across bins of structural similarity (ECFP-Tanimoto) and string similarity (SequenceMatcher), evaluated on data from different sources and for both terminal and core replacements. The size of the semi-transparent halo around each point reflects the number of test samples in that bin.

{Objective description}

Your response must be actions to perform the transformation, i.e.
"replace original_group_smiles connected at atom_number with
target_group_smiles".

Each action must include two SMILES strings indicating the original
group and the target group.

Use dummy atoms to mark the connection point, e.g.

"replace [*:1]Cl connected at atom_number with [*:1]OC"

or

"replace [*:2]c1ccc([*:3])cc1 connected at atom_number2, atom_number3
with [*:2]c1cnc([*:3])cc1".

Your response must be in directly parsable JSON format:

```
{
  "Action Description": [
    "replace original_group_smiles connected
      at atom_number with target_group_smiles",
    ...
  ],
  "Final Target Molecule": "SMILES"
}
```

Given the source molecule with atoms numbered:

{number_smi(source_smiles)}.

You should ignore Hydrogen (H) and numbers in the group SMILES.

Table 7: Prompt wrapper in the molecular modification task.

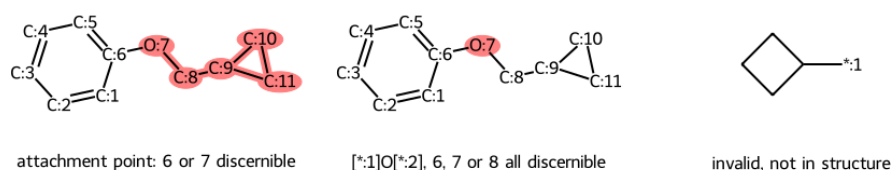


Figure 8: Visualization of manual check criteria.

Nevertheless, manual checking is inherently subjective. Different researchers may interpret borderline cases differently, and there remains a possibility of human error.

Subjectivity. To further assess potential subjectivity, we adopt a blinded multi-annotator protocol: three annotators independently evaluate 120 samples (evenly drawn from DeepSeek-R1), without access to model identities, task provenance, or one another’s labels. Labels were likewise highly consistent: three-way agreement = 95.83%, two-way = 3.33%, and complete disagreement = 0.83%. Attachment-point partiality judgments achieved Fleiss’ κ = 0.9342 for the 104 jointly valid cases, indicating near-perfect agreement.

Scalability. This verification pipeline naturally supports scalable dataset growth: for each natural-language edit instruction, annotators provide one gold-standard structure (e.g., SMILES) that correctly implements the edit. Each labeled (instruction, targeted structure) pair becomes a reusable test case and can be automatically applied to any future code-LLM output via structure matching. Thus, each round of human labeling incrementally expands a standardized, reproducible, and low-cost evaluation suite. In addition, the annotation process can be substantially accelerated through formalized validation code. For example, substructure errors (e.g., fragments not present in the source molecule) can be automatically excluded before human inspection. This further reduces subjective load and ensures that annotators only review chemically valid candidate edits.

Regarding scalability toward larger and more diverse test sets, we draw on recent progress in LLM evaluation and outline several feasible extensions:

1. **LLM-as-a-Judge with minimal supervision.** Lin and Chen (2023); Li et al. (2026) show that structured prompting enables LLMs to reliably approximate human judgments at scale, suggesting that MECo’s intention-consistency checks could be extended through supervised automatic judges.

2. **Decomposed, verifiable evaluation dimensions.** Zhou et al. (2023); Ye et al. (2024) show that breaking complex behaviors into verifiable atomic criteria yields more stable and interpretable assessments. Future MECo versions may evaluate edits along dimensions such as (i) substructure localization, (ii) bond/valence correctness, (iii) atom-mapping agreement, and (iv) chemically valid topology, paired with automatic graph/SMILES

validators. This reduces subjectivity and aligns evaluation with chemically interpretable components.

3. **Standardized benchmark construction.** Successful evaluation suites (Zhou et al., 2023; Zheng et al., 2023) emphasize fixed task sets, unified templates, and executable scoring pipelines. Inspired by these designs, we could build a standardized benchmark for molecular edit-intention consistency with public task suites and unified evaluation scripts, ensuring fully reproducible cross-model comparison.

These extensions provide a clear and technically grounded path toward scaling MECo’s intention-consistency evaluation beyond the manually assessed subset while enhancing reproducibility, transparency, and objectivity.

C.4 Observation on terminal replacement test samples

In Section 4.2, we noted a discrepancy between reaction-derived and target-specific terminal replacements. To investigate this further, we conducted a closer inspection and found that the discrepancy may stem from differences in SMILES syntax patterns: in target-specific edits, dummy atoms (e.g., [$\ast$:1]) often appear in the middle or at the end of SMILES strings, whereas in reaction-derived edits, they frequently occur at the beginning. This difference arises from the implementation of the extraction program rather than the underlying data distribution, meaning that the two syntax types can be transformed into each other by choosing whether the dummy atom or another atom is used as the SMILES root. Such pattern sensitivity suggests that LLMs struggle to generalize over SMILES syntax, further motivating the use of structured code as a more robust and interpretable intermediate representation.

C.5 Qualitative Error Analysis

Qualitative inspection of failed examples reveals representative failure modes on both the code-generation and reasoning sides. On the code-generation side, as discussed in Section C.4, performance is sensitive to dummy-atom placement, particularly when attachment markers appear at the end of complex SMARTS expressions.

On the reasoning side, failures in the downstream end-to-end setting can arise from structured editing intentions that are difficult to execute reliably. Occasional atom-index offsets may either

cause editing to fail or mislocalize an otherwise valid editing intention, with the error subsequently propagating to execution. Other cases lie at the boundary between reasoning and execution. For example, the reasoner may propose atom-level substitutions within aromatic systems, where lower-case SMARTS atoms encode aromaticity. Such intentions can be readily understood by a human expert, but their realization through reaction-based editing may require reformulating the edit into a reaction-compatible operation. Since the current lightweight coder is designed for faithful execution rather than semantic reinterpretation and reimplementation, these cases remain challenging.

C.6 Claude-Sonnet-5 on Realistic Edit Benchmarks

To assess whether the code-based editing formulation can be effectively executed by a leading frontier code LLM without task-specific fine-tuning, we additionally evaluated Claude-Sonnet-5 on 100 randomly sampled examples from each realistic edit category. It achieves accuracies of 98.0%, 62.0%, and 73.0% on reaction-derived terminal replacement, target-specific terminal replacement, and target-specific core replacement, respectively.

C.7 Property improvement vs. similarity

To further analyze the trade-off between property improvement and structural preservation, we compare the results of direct generation and MECo separately on all six tasks (Figure 9). The top row shows the average property improvement (relative to the initial molecule), where MECo generally achieves higher improvements than direct generation. The bottom row shows the average structural similarity (ECFP4-based) to the initial molecule. While Gemini-2.5-Pro exhibits larger improvements in the upper row, it tends to reduce similarity more than DeepSeek-R1, indicating that different LLMs show different levels of willingness to sacrifice structural similarity in exchange for property gains. This observation motivates our use of success rate as the primary evaluation metric in the main paper, as it balances both optimization effectiveness and structural plausibility.

C.8 Docking setups for case study

For the case study, molecular docking was performed using the Glide SP (standard precision) protocol implemented in the Schrödinger suite. The high-resolution X-ray co-crystallized target protein

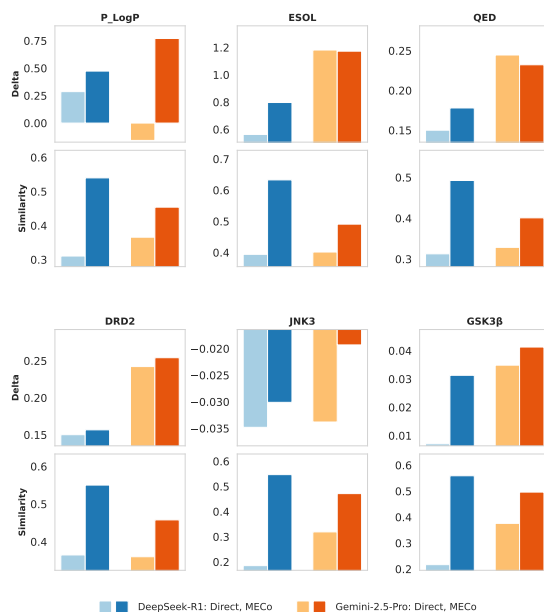


Figure 9: Bar plots of molecular optimization performance across six tasks, showing property improvement (top row) and structural similarity (bottom row) for Direct generation and MECo for DeepSeek-R1 and Gemini-2.5-Pro.

structure was obtained from the Protein Data Bank (PDB ID: 4AFJ).

Protein preparation was carried out using the Protein Preparation Wizard, including addition of hydrogen atoms, assignment of bond orders, optimization of the hydrogen-bonding network, and restrained minimization of heavy atoms with the OPLS_2005 force field. Ligands were prepared using LigPrep to generate at most 32 stereoisomers.

Docking was performed with default Glide SP parameters unless otherwise specified. The receptor grid was centered on the co-crystallized ligand SJJ in 4AFJ. The top-ranked docking poses were retained for analysis and visualization.