# DO WHAT YOU SAY: TOWARDS VERIFIABLE REASONING FOR SCIENTIFIC EDITING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Language models increasingly produce plans or rationales before their final outputs to guide generation and support supervision. When these traces are used for auditing or process supervision, a mismatch between their stated actions and the final output creates a *say–do gap*. Scientific editing provides a concrete testbed for this problem because domain tools can check both the requested outcome and the realized input–output transformation. Yet free-form traces make such comparison difficult because the claimed edit must first be interpreted from prose. We introduce **TRACE**, a structured reasoning-supervision framework that makes the declared edit explicit and directly checkable. TRACE represents each edit as a typed, replayable Goal–Local Edit–Safeguard contract constructed from task-computable facts. Through supervised fine-tuning, models learn to generate this contract before answering, while outcome-based reinforcement learning further improves task performance. Across three molecular editing benchmarks, TRACE consistently outperforms matched answer-only and free-form reasoning baselines, with gains that persist across model scales, backbones, and RL optimizers. On TOMG with Qwen3-1.7B, TRACE improves the average success–similarity score by $11.6\%$ over matched answer-only training. TRACE also makes agreement between stated and realized edits directly auditable. Controlled interventions provide evidence that the input-specific Local Edit carries most of the action-relevant signal, while the full contract performs best. The resulting verifier further improves say–do consistency through process supervision and enables reference-answer-free Best-of-$N$ selection that improves both editing performance and trace–answer consistency. Beyond molecular editing, TRACE extends to RNA codon optimization and reaction repair, demonstrating broader applicability to scientific editing.

## 1 INTRODUCTION

Language models are increasingly used to solve complex tasks by producing plans or rationales before their final outputs, both to guide the answer and to support inspection and supervision (Lightman et al., 2023; Guo et al., 2025). When these traces are used for auditing or process supervision, it matters whether their stated actions are reflected in the final output (Turpin et al., 2023). Scientific editing provides a particularly suitable setting for studying this question because models operate on structured objects, allowing the stated transformation to be compared directly with the change realized in the final output (Fang et al., 2024; Li et al., 2023; Ye et al., 2023). Recent LLM-based scientific editors broadly follow two approaches. Answer-only methods generate the edited object directly, with training and evaluation centered on the final output (Li et al., 2026a; Dey et al., 2025). They provide no explicit intermediate edit claim for process-level auditing or supervision. Methods with free-form reasoning instead generate a trace before the edited object (Li et al., 2026b). These traces can state a proposed edit, but the final object may not realize it (Yu et al., 2026).

We refer to a mismatch between the stated edit and the realized change as the *say–do gap*. Final-output evaluation alone cannot reveal this gap: it can determine whether the edited object succeeds, but not whether it realizes the edit stated in the trace. Process rewards and LLM graders can assess the reasoning text (Lightman et al., 2023; Gunjal et al., 2026), but free-form edit claims still require interpretation before they can be compared with the realized change. What is missing is a task-grounded edit representation whose claims can be checked directly against the generated object. Such



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.

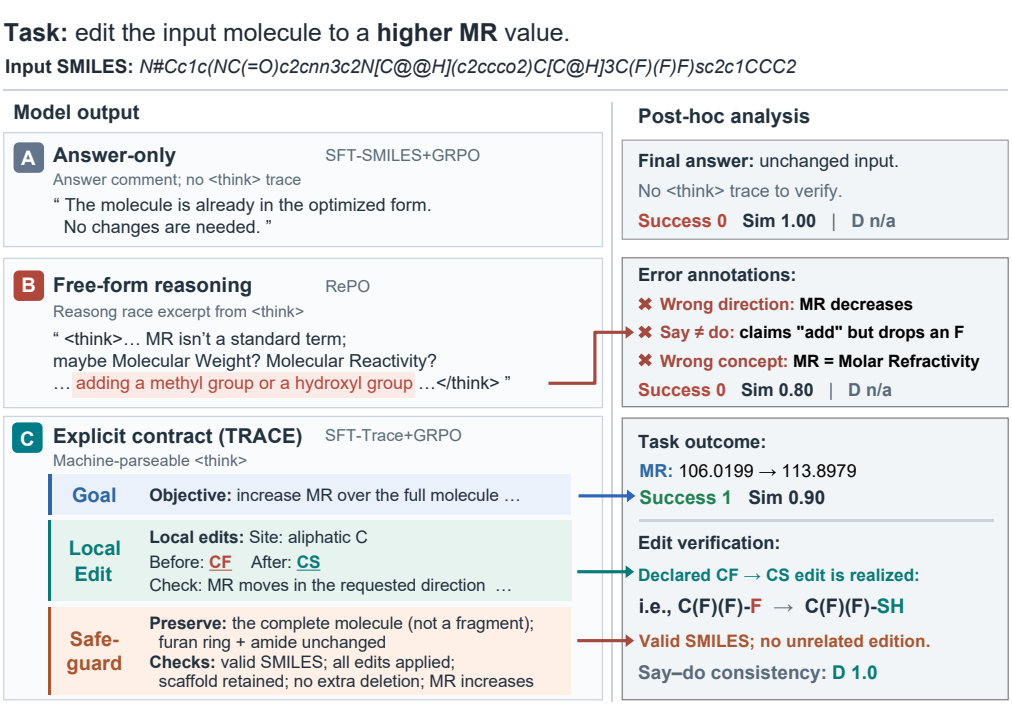



Figure 1: **Comparison of trace–answer consistency.** (A) Answer-only SFT-SMILES+GRPO provides no explicit trace and returns the input unchanged. (B) RePO produces a free-form trace, but the final molecule does not follow the stated edit and fails the task. (C) TRACE instead exposes a verifiable edit claim that is realized in the final answer and successfully completes the requested edit.

a representation would make say–do mismatches directly auditable while also supporting process supervision during training and candidate selection at test time.

Molecular editing provides a concrete setting in which such a representation can be constructed and checked. Models are asked to improve a molecular property or modify a functional group while preserving validity and similarity to the parent molecule (Li et al., 2026a; Dey et al., 2025; Li et al., 2026b), while task-specific tools can characterize both the resulting properties and the realized structural changes. Figure 1 illustrates the say–do gap: given the same editing query, the answer-only model makes no explicit edit claim, while RePO (Li et al., 2026b) describes an addition but produces a molecule that instead loses a fluorine. The question is therefore not simply whether the rationale is plausible, but whether the final molecule actually carries out the stated edit.

To address this gap, we introduce **TRACE**, which represents the stated edit as a structured contract that can be checked directly against the final output. For each training pair, task-specific tools compare the input with the reference answer and convert the resulting facts into a Goal–Local Edit–Safeguard contract (Figure 2). Goal specifies the requested objective, Local Edit records the input-specific transformation, and Safeguard specifies preservation requirements. Through supervised fine-tuning (SFT), the model learns to generate this contract before producing its answer, and reinforcement learning (RL) with outcome-only rewards further improves task performance. At evaluation, the same tools compare the generated contract with the actual input–output change, directly exposing say–do mismatches without a reference answer. The verifier supports auditing by default and can optionally provide process feedback during training or guide candidate selection at test time.

Across three molecular editing benchmarks, TRACE consistently outperforms matched answer-only and free-form reasoning baselines. On TOMG (Li et al., 2026a) with Qwen3-1.7B, TRACE improves the average success–similarity score by 11.6% over the answer-only baseline (0.5762 vs. 0.5165) and substantially outperforms RePO (0.3184). These gains persist across model scales, backbones, and RL optimizers, and also extend beyond molecular editing to codon optimization and reaction repair.

Beyond improving final outcomes, TRACE also makes the stated edits directly auditable. Across all six TOMG subtasks, every TRACE output exposes a complete contract, whereas RePO provides no

native contract and only $13.9\%$ of its outputs yield a correct coarse edit claim under permissive post-hoc parsing. Controlled ablations and forced-prefix interventions suggest that the input-specific Local Edit carries the main action signal, while the complete contract performs best. Separate experiments demonstrate the utility of the verifier beyond auditing: using it for process supervision improves say–do consistency, while verifier-guided Best-of-$N$ selection approaches oracle outcome performance with substantially higher trace–answer consistency. We summarize our main contributions as follows:

- We introduce **TRACE**, a structured reasoning-supervision framework for scientific editing that makes stated edits replayable and directly checkable against the generated output.
- TRACE improves performance across three molecular editing tasks, with consistent gains across model scales, backbones, and RL optimizers. Its extension to RNA codon optimization and reaction repair tasks further demonstrates the framework's broader applicability.
- Controlled interventions suggest that the input-specific Local Edit carries the main action signal, while the full contract performs best. The resulting verifier also supports direct auditing, process supervision, and verifier-guided candidate selection.

## 2 METHOD

### 2.1 PRELIMINARIES

We consider constrained editing over a task family $\tau$. An instance is $e_i = (x_i, \mathbf{s}_i, \mathbf{c}_i, \mathbf{s}_i^\star)$, where $x_i$ is the task context, $\mathbf{s}_i$ the input object, $\mathbf{c}_i$ the requested change, and $\mathbf{s}_i^\star$ a reference edit used only during data construction. Given $q_i = (x_i, \mathbf{s}_i, \mathbf{c}_i)$, the model generates a trace–answer pair $(\hat{\mathbf{t}}_i, \hat{\mathbf{s}}_i)$. The editing objective is to realize the requested change while preserving task-relevant input content:

$$\mathbf{s}_i^\dagger \in \arg\max_{\mathbf{s}\in\mathcal{S}_\tau} U_{\mathbf{c}_i}(\mathbf{s};\mathbf{s}_i) \quad \text{s.t.} \quad V_\tau(\mathbf{s}) = 1, \quad P_\tau(\mathbf{s},\mathbf{s}_i) \geq \delta_\tau. \tag{1}$$

where $V_\tau$ tests validity, $U_{\mathbf{c}_i}$ measures satisfaction of the requested change, $P_\tau$ measures preservation, and $\delta_\tau$ is the required preservation threshold. For molecular editing, $\mathbf{s}$ is a SMILES string, $V_\tau$ tests chemical validity, and $P_\tau$ is Tanimoto similarity to the input.

### 2.2 TRACE FRAMEWORK

Constrained-editing criteria such as Equation (1) evaluate the generated answer, but do not reveal whether its accompanying trace describes the edit actually performed. TRACE addresses this gap by representing the declared edit as task-grounded claims with corresponding verification rules, allowing stated and realized edits to be compared directly. The overall framework is summarized in Figure 2, with detailed trace construction workflow provided in Section B.1.

**Task-family interface.** For each task family, we define a fixed interface:

$$\mathcal{I}_\tau = (\Phi_\tau, \Sigma_\tau, \Lambda_\tau, \mathcal{R}_\tau, \mathcal{X}_\tau)\,, \tag{2}$$

where $\Phi_\tau$ extracts task facts from a query $q_i$ and an edited object; $\Sigma_\tau$ assigns each required field to the Goal ($G$), Local Edit ($L$), or Safeguard ($S$) role; $\Lambda_\tau$ defines the allowed claim types and their semantics; $\mathcal{R}_\tau$ renders claims from $q_i$ and the extracted facts; and $\mathcal{X}_\tau$ checks generated claims against recomputed facts. The schema and claim together define the trace contract. Every rendered claim must be licensed by $\Lambda_\tau$ and grounded in $q_i$ or the extracted facts, with a corresponding test in $\mathcal{X}_\tau$.

**Fact extraction.** We first describe the construction for a training example $i$. The extractor is applied to the reference edit $\mathbf{s}_i^\star$, so the resulting facts describe the transformation $\mathbf{s}_i \to \mathbf{s}_i^\star$:

$$\mathcal{F}_i = \Phi_\tau(q_i, \mathbf{s}_i^\star) = \left(\mathcal{F}_i^{\text{edit}}, \mathcal{F}_i^{\text{pres}}, \mathcal{F}_i^{\text{check}}\right). \tag{3}$$

The three components answer different questions. Edit facts $\mathcal{F}_i^{\text{edit}}$ specify *what changed* from $\mathbf{s}_i$ to $\mathbf{s}_i^\star$; preservation facts $\mathcal{F}_i^{\text{pres}}$ specify *what was kept*; and check facts $\mathcal{F}_i^{\text{check}}$ specify *what can be recomputed*, such as validity, property direction, or functional-group counts.Edit facts are site–before–after graph differences for molecular editing, indexed synonymous substitutions for codon optimization, and slot-indexed coefficient, drop, or move-side edits for reaction repair.

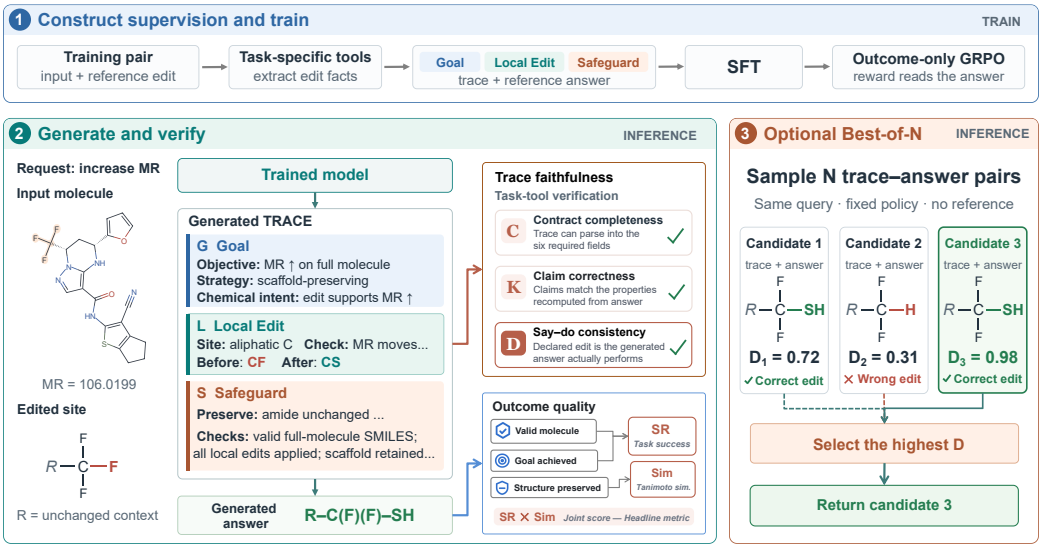



Figure 2: **Overview of TRACE.** (1) Task-specific tools construct traces from input–reference edits for SFT, followed by outcome-only GRPO. (2) The generated trace and answer are evaluated separately: whether the trace matches the realized edit, and whether the answer satisfies the task objective. (3) Optional Best-of-$N$ selects the answer that best follows its trace's stated edits (highest $D$ score).

**Trace construction.** The renderer then groups the extracted facts into three roles. For training sample $i$, let $G_i$, $L_i$, and $S_i$ denote the sample-specific Goal, Local Edit, and Safeguard contents. $G_i$ captures the requested objective and editing strategy, $L_i$ specifies the input-specific transformation, and $S_i$ records the content to preserve and the conditions to check. These roles form the structured trace:

$$\mathbf{r}_i = (G_i, L_i, S_i) = \mathcal{R}_\tau(q_i, \mathcal{F}_i; \Sigma_\tau, \Lambda_\tau)\,. \tag{4}$$

For molecular editing, the three roles are expressed through six literal fields: Goal comprises OBJECTIVE, derived from the request, and STRATEGY and CHEMICAL INTENT, grounded in the request and extracted facts. Local Edit contains LOCAL EDITS from $\mathcal{F}_i^{\text{edit}}$; Safeguard contains PRESERVE and CHECKS from $\mathcal{F}_i^{\text{pres}}$ and $\mathcal{F}_i^{\text{check}}$, respectively.

**Task-specific instantiations.** The same construction pattern applies across task families, while the underlying facts and claim types are task-specific. Table 3 summarizes the facts, claims, and checkers for each benchmark, with construction details in Section B.

**Training target.** The instantiated task-specific fields are then serialized into a structured trace and paired with the reference answer:

$$\mathbf{t}_i = \text{serialize}_\tau(\mathbf{r}_i), \quad d_i = \texttt{<think>}\,\mathbf{t}_i\,\texttt{</think><answer>}\,\mathbf{s}_i^\star\,\texttt{</answer>}. \tag{5}$$

Compared with answer-only supervision, TRACE adds only the structured trace prefix, yielding the SFT target $d_i$. Figure 6 illustrates the detailed construction, and Algorithm 1 summarizes the sample-level procedure. After SFT, we further optimize the policy with outcome-only GRPO (Shao et al., 2024). For a query $q = (x, \mathbf{s}, \mathbf{c})$ and generated answer $\hat{\mathbf{s}}$, the outcome reward is

$$r_{\text{out}}(q, \hat{\mathbf{s}}) = \mathbb{1}[F_\tau(\hat{\mathbf{s}}, \mathbf{s}, \mathbf{c})]\, h_\tau(\hat{\mathbf{s}}, \mathbf{s}, \mathbf{c}), \tag{6}$$

where $F_\tau$ checks validity and task-specific constraints, and $h_\tau$ ranks feasible candidates. GRPO updates the full trace–answer rollout, but its advantage is computed only from the generated answer. Task-specific rewards and optimization details are given in Section A.4.

### 2.3 TRACE VERIFICATION AND TASK OUTCOME

The construction above makes the generated trace directly checkable against the edit it accompanies. We therefore assess each output along two independent axes: whether the generated trace matches the realized change, and whether the final answer satisfies the task objective.

**Trace faithfulness.** Verification reuses the construction-time fact extractor. It is applied to the reference edit $\mathbf{s}_i^\star$ during construction and to the generated answer $\hat{\mathbf{s}}_i$ at evaluation:

$$\widehat{\mathcal{F}}_i = \Phi_\tau(q_i, \hat{\mathbf{s}}_i), \quad (C_i, K_i, D_i) = \mathcal{X}_\tau\left(q_i, \hat{\mathbf{t}}_i, \widehat{\mathcal{F}}_i\right). \tag{7}$$

The recomputed facts describe $\mathbf{s}_i \to \hat{\mathbf{s}}_i$ without consulting the reference. The checker returns three faithfulness scores: contract completeness $C_i$, claim correctness $K_i$, and say–do consistency $D_i$.

**Faithfulness scores.** Let $A_i$ denote the testable claims parsed from $\hat{\mathbf{t}}_i$ (with $K_i = 0$ when $A_i = \varnothing$), $\mathrm{supp}_\tau(a; \widehat{\mathcal{F}}_i)$ the support score for claim $a$, and $E_\tau(A_i; \widehat{\mathcal{F}}_i)$ the agreement between declared and executed edits. Let $g_{1,i}, \ldots, g_{m,i} \in [0,1]$ denote safeguard checks, and let $\mathrm{complete}_{\Sigma_\tau}(\hat{\mathbf{t}}_i)$ indicate whether the required schema and testable claims are present. We define

$$C_i = \mathbb{1}\left[\mathrm{complete}_{\Sigma_\tau}(\hat{\mathbf{t}}_i)\right], \quad K_i = \frac{1}{|A_i|}\sum_{a \in A_i} \mathrm{supp}_\tau(a; \widehat{\mathcal{F}}_i), \quad D_i = E_\tau(A_i; \widehat{\mathcal{F}}_i)\prod_{j=1}^{m} g_{j,i}. \tag{8}$$

Thus $C_i$ checks contract completeness, $K_i$ measures how well its claims are supported by the generated answer, and $D_i$ additionally requires the declared edits and safeguards to be realized. Together, these scores provide a direct audit of say-do consistency without a reference answer. Task-specific score definitions are given in Section C.3.1.

**Outcome quality.** The $C/K/D$ metrics measure whether the trace faithfully describes the realized edit, whereas the answer-level criteria in Equation (1) measure whether that edit solves the task. We report task success as SR and preservation as Sim when applicable, with SR $\times$ Sim as the headline composite for constrained molecular editing. Benchmark-specific definitions are given in Section 3.1.

**Optional extension.** Beyond auditing, we use the say–do score as a reference-answer-free criterion for Best-of-$N$ selection, returning the highest-scoring trace–answer pair among $N$ samples from a fixed policy. Experiments show that this improves both editing performance and trace faithfulness, indicating that stronger say–do consistency generally aligns with better outcomes. Note that this extension is evaluated separately in Section 3.3 and is not used in our main results.

## 3 Experiments

We first outline the experimental setup, followed by evaluations of TRACE's effectiveness and robustness, native checkability, mechanism, and broader utility across scientific editing tasks.

### 3.1 Setup

**Datasets.** We evaluate TRACE primarily on molecular editing. TOMG (Li et al., 2026a) is used as the main benchmark, covering property optimization and explicit structural edits. We further evaluate MUMO (Dey et al., 2025) and TDC-PairMolGen (Jin et al., 2019) for broader instruction and redesign generalization, and evaluate whether the same construction generalizes beyond molecular editing to codon optimization (Li et al., 2023) and Rhea reaction repair (Bansal et al., 2022).

**Models.** We use Qwen3 as the primary model family and fully fine-tune its 1.7B, 4B, and 8B variants to study scaling effects within a controlled architecture (Yang et al., 2025). To test whether TRACE generalizes beyond a single model family and tuning strategy, we further evaluate full-fine-tuned Llama-3.1-8B and LoRA-tuned Gemma-3-12B (Dubey et al., 2024; Team et al., 2025).

**Baselines.** We compare SFT-TRACE+GRPO against the matched answer-only baseline SFT-SMILES+GRPO. The two pipelines share the same reference answers, outcome reward, and training protocol, differing only in the SFT target: TRACE adds a structured trace prefix before the answer. We also reproduce RePO (Li et al., 2026b) as a free-form reasoning baseline. Training hyperparameters and task-specific rewards are given in Sections A.3 and A.4.

**Evaluation.** Following RePO (Li et al., 2026b), we evaluate molecular editing with success rate (SR), Tanimoto similarity (Sim), and their joint score $\mathrm{SR} \times \mathrm{Sim} = N^{-1}\sum_{i:\,\mathrm{success}_i=1} \mathrm{Sim}_i$. This composite assigns zero to failed edits while rewarding successful edits that better preserve the input structure. For other task families, we use the task-specific success criteria and, when applicable, preservation measures defined by their executors. Results are reported as mean$\pm$std over generation seeds 42/43/44 for a fixed checkpoint. Further evaluation details are given in Section A.2.

Table 1: **Main TOMG-Bench results with Qwen3-1.7B.** Rows report SR, Sim, and joint SR×Sim for six tasks; cells are mean±std over generation seeds 42/43/44. **Bold** and underline denote the best and second-best results. SFT-TRACE+GRPO achieves the highest SR and joint score on all tasks.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Optimization | LogP | SR | $0.5677_{\pm 0.0057}$ | $0.7434_{\pm 0.0053}$ | $0.4494_{\pm 0.0027}$ | $\underline{0.8283}_{\pm 0.0007}$ | $\mathbf{0.9425}_{\pm 0.0019}$ |
| | | Sim | $0.7070_{\pm 0.0046}$ | $0.6392_{\pm 0.0048}$ | $0.6856_{\pm 0.0005}$ | $\mathbf{0.7872}_{\pm 0.0001}$ | $\underline{0.7737}_{\pm 0.0019}$ |
| | | SR×Sim | $0.3966_{\pm 0.0059}$ | $0.4803_{\pm 0.0031}$ | $0.2948_{\pm 0.0020}$ | $\underline{0.6504}_{\pm 0.0006}$ | $\mathbf{0.7374}_{\pm 0.0007}$ |
| | MR | SR | $0.6117_{\pm 0.0565}$ | $\underline{0.7321}_{\pm 0.0027}$ | $0.5397_{\pm 0.0031}$ | $0.6695_{\pm 0.0006}$ | $\mathbf{0.7992}_{\pm 0.0020}$ |
| | | Sim | $0.7036_{\pm 0.0072}$ | $0.6342_{\pm 0.0034}$ | $0.6848_{\pm 0.0037}$ | $\mathbf{0.8009}_{\pm 0.0000}$ | $\underline{0.7638}_{\pm 0.0029}$ |
| | | SR×Sim | $0.4249_{\pm 0.0362}$ | $0.4669_{\pm 0.0040}$ | $0.3560_{\pm 0.0011}$ | $\underline{0.5347}_{\pm 0.0005}$ | $\mathbf{0.6183}_{\pm 0.0002}$ |
| | QED | SR | $0.4431_{\pm 0.0100}$ | $\underline{0.5387}_{\pm 0.0033}$ | $0.4651_{\pm 0.0059}$ | $0.4841_{\pm 0.0009}$ | $\mathbf{0.6003}_{\pm 0.0020}$ |
| | | Sim | $0.7045_{\pm 0.0061}$ | $0.6453_{\pm 0.0032}$ | $0.6785_{\pm 0.0029}$ | $\mathbf{0.7899}_{\pm 0.0001}$ | $\underline{0.7713}_{\pm 0.0005}$ |
| | | SR×Sim | $0.3098_{\pm 0.0059}$ | $0.3528_{\pm 0.0024}$ | $0.3034_{\pm 0.0029}$ | $\underline{0.3782}_{\pm 0.0008}$ | $\mathbf{0.4681}_{\pm 0.0019}$ |
| Structure Optimization | Add | SR | $0.5704_{\pm 0.0028}$ | $0.6348_{\pm 0.0018}$ | $0.5033_{\pm 0.0026}$ | $\underline{0.6942}_{\pm 0.0007}$ | $\mathbf{0.7085}_{\pm 0.0011}$ |
| | | Sim | $0.6888_{\pm 0.0012}$ | $0.6462_{\pm 0.0016}$ | $0.6460_{\pm 0.0006}$ | $\mathbf{0.7770}_{\pm 0.0002}$ | $\underline{0.7603}_{\pm 0.0021}$ |
| | | SR×Sim | $0.3979_{\pm 0.0022}$ | $0.4449_{\pm 0.0012}$ | $0.3494_{\pm 0.0018}$ | $\underline{0.5391}_{\pm 0.0005}$ | $\mathbf{0.5427}_{\pm 0.0008}$ |
| | Del | SR | $0.5740_{\pm 0.0187}$ | $0.5657_{\pm 0.0008}$ | $0.3989_{\pm 0.0033}$ | $\underline{0.6642}_{\pm 0.0004}$ | $\mathbf{0.7558}_{\pm 0.0007}$ |
| | | Sim | $0.6574_{\pm 0.0048}$ | $0.5788_{\pm 0.0023}$ | $0.6598_{\pm 0.0026}$ | $\mathbf{0.7212}_{\pm 0.0002}$ | $\underline{0.6947}_{\pm 0.0003}$ |
| | | SR×Sim | $0.3802_{\pm 0.0111}$ | $0.3674_{\pm 0.0005}$ | $0.2668_{\pm 0.0023}$ | $\underline{0.4736}_{\pm 0.0004}$ | $\mathbf{0.5368}_{\pm 0.0005}$ |
| | Sub | SR | $0.6210_{\pm 0.0041}$ | $0.5775_{\pm 0.0018}$ | $0.4473_{\pm 0.0021}$ | $\underline{0.6853}_{\pm 0.0003}$ | $\mathbf{0.7319}_{\pm 0.0004}$ |
| | | Sim | $0.7354_{\pm 0.0004}$ | $0.6958_{\pm 0.0003}$ | $0.6238_{\pm 0.0014}$ | $\mathbf{0.7600}_{\pm 0.0001}$ | $\underline{0.7528}_{\pm 0.0004}$ |
| | | SR×Sim | $0.4714_{\pm 0.0030}$ | $0.4386_{\pm 0.0012}$ | $0.3398_{\pm 0.0013}$ | $\underline{0.5232}_{\pm 0.0003}$ | $\mathbf{0.5540}_{\pm 0.0003}$ |
| All Tasks | Avg | SR | $0.5646_{\pm 0.0130}$ | $0.6320_{\pm 0.0011}$ | $0.4673_{\pm 0.0014}$ | $\underline{0.6709}_{\pm 0.0002}$ | $\mathbf{0.7564}_{\pm 0.0007}$ |
| | | Sim | $0.6994_{\pm 0.0035}$ | $0.6399_{\pm 0.0021}$ | $0.6631_{\pm 0.0009}$ | $\mathbf{0.7727}_{\pm 0.0000}$ | $\underline{0.7528}_{\pm 0.0009}$ |
| | | SR×Sim | $0.3968_{\pm 0.0077}$ | $0.4251_{\pm 0.0005}$ | $0.3184_{\pm 0.0007}$ | $\underline{0.5165}_{\pm 0.0001}$ | $\mathbf{0.5762}_{\pm 0.0004}$ |

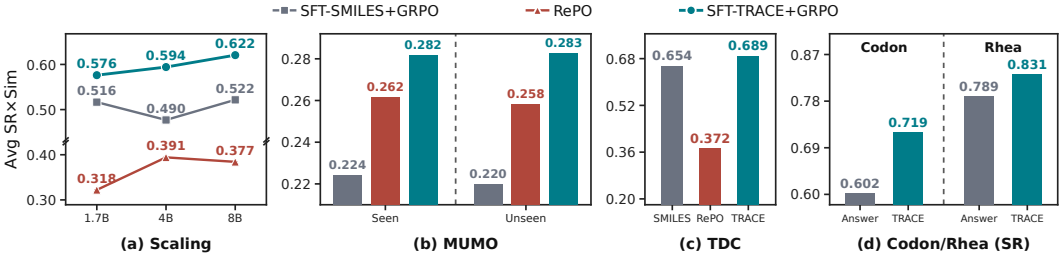



Figure 3: **TRACE's advantage extends across model scales, benchmarks, and editing domains.** (a) Average SR×Sim on TOMG across Qwen3 scales. (b,c) Average joint scores on two MUMO splits and TDC redesign tasks. (d) Success rate on codon optimization and Rhea reaction repair.

## 3.2 Quantitative Results

### 3.2.1 Performance and robustness on TOMG

**TRACE improves constrained editing.** Under the same setting, SFT-TRACE+GRPO achieves the best SR and SR×Sim across all six TOMG subtasks (Table 1). It improves the average joint score by 11.6% over the matched SFT-SMILES+GRPO baseline (0.5762 vs. 0.5165), and substantially outperforms the reproduced RePO baseline (0.3184). Interestingly, the answer-only baseline also improves under our task-specific outcome reward compared with the published RePO setting (Section C.4.6), while TRACE further improves over this stronger baseline.

**Scaling increases the benefit of trace supervision.** Across Qwen3-1.7B, 4B, and 8B, TRACE improves the average composite over SFT-SMILES+GRPO by 11.6%, 21.3%, and 19.2%, respectively (Figure 3 and tables 1, 24 and 25). The improvement is primarily driven by higher SR: although the answer-only baseline retains slightly higher Sim at 1.7B and 8B, TRACE achieves stronger overall editing success without a comparable loss of preservation.

**The gain generalizes across model families and tuning strategies.** Beyond Qwen3, TRACE improves the average composite over matched answer-only training by 22.5% on full-fine-tuned Llama-3.1-8B and by 6.0% on LoRA-tuned Gemma-3-12B. These results show that the benefit is not restricted to a specific backbone or adaptation method (Tables 26 and 27).

**The improvement is independent of the RL optimizer.** To test whether TRACE relies on a specific RL optimizer, we replace GRPO with GSPO (Zheng et al., 2025), RLOO (Ahmadian et al., 2024), and CISPO (MiniMax et al., 2025) at the Qwen3-1.7B scale while keeping the remaining training conditions fixed. TRACE consistently improves the average joint score by 8.5%–11.6% over the corresponding answer-only baselines across all optimizer settings (Table 4 and section C.1).

### 3.2.2 Generalization across tasks and modalities

Beyond TOMG, we evaluate whether TRACE generalizes to more complex editing scenarios, including multi-property objectives, diffuse redesigns, and other scientific editing domains. These settings test the construction under more diverse task structures.

**Strong generalization to unseen multi-property instructions.** As shown in Figure 3(b) and Table 28, TRACE improves the average joint score over the answer-only baseline by 25.8% on seen instructions and 28.7% on unseen instructions, indicating robust generalization to unseen instructions. Unlike on TOMG, RePO also outperforms the answer-only baseline under the multi-property objective. TRACE nevertheless achieves the highest joint score, surpassing RePO by 8.6%.

**Consistent gains on diffuse redesigns.** TDC-PairMolGen tests more diffuse property redesigns for which an exact local edit may not be available. As shown in Figure 3(c) and Table 29, TRACE leads all three tasks and improves the average joint score by 5.3% over matched answer-only training. The gain comes from higher SR (0.9027 vs. 0.8373) with only a modest similarity cost (0.7600 vs. 0.7793). RePO also reaches high SR (0.8884), but its much lower similarity (0.4450) leaves its joint score near SFT-only and far below both +GRPO arms.

**The construction extends to other scientific editing tasks.** Figure 3(d) shows that the same fact–claim–executor construction can be applied to other scientific editing tasks. For codon optimization, TRACE represents edits as indexed synonymous substitutions and uses an executor to verify preservation of the translated protein sequence. It improves SR by 19.4% and $\mathrm{SR} \times \mathrm{Sim}$ by 18.3% over the answer-only baseline, with more successful synonymous edits and fewer below-target CAI failures (Table 6 and section C.2.1). For Rhea reaction repair, TRACE increases SR by 5.3%, with the largest improvements on hard multi-edit reactions (Table 7 and section C.2.2). These results demonstrate that the construction principle extends beyond molecular editing to other scientific editing settings.

## 3.3 Native Checkability and Say–Do Consistency

Strong editing outcomes do not guarantee that the answer follows the stated process. We therefore evaluate native checkability through contract completeness $C$, claim correctness $K$, and say–do consistency $D$. On TOMG with Qwen3-1.7B, TRACE outputs expose executable contracts with $C = 1.000$, $K = 0.668$, and $D = 0.490$. In contrast, free-form RePO has no native contract or defined $K/D$; a permissive post-hoc parser recovers only limited claims, with a recovery score of 0.139 (Table 17). We next examine how this process signal relates to editing outcomes.

**Low $D$ localizes execution failures.** In the same Qwen3-1.7B setting, the lowest-$D$ group contains all invalid and no-op outputs and exhibits a 56.4% outcome-failure rate, compared with approximately 35% in the two higher groups (Figure 4a and Section C.3.4). Conditioning on valid, changed outputs shows that this association is largely driven by shared feasibility failures: beyond them, $D$ distinguishes faithful execution rather than whether the edit improves the requested objective (Table 11). We therefore examine two uses of the verifier: as a training-time process signal and as a test-time selector over generated candidates.

**Process rewards mainly improve faithfulness.** We augment outcome-only GRPO with deterministic process objectives to test whether verifier feedback can improve faithfulness during training. The additive variant combines a weighted verifier score with the outcome reward, while the token-split variant adds a group-centered process advantage only to `<think>` tokens. As shown in Figure 4b and Table 12, both variants improve faithfulness, but neither consistently improves the joint outcome.

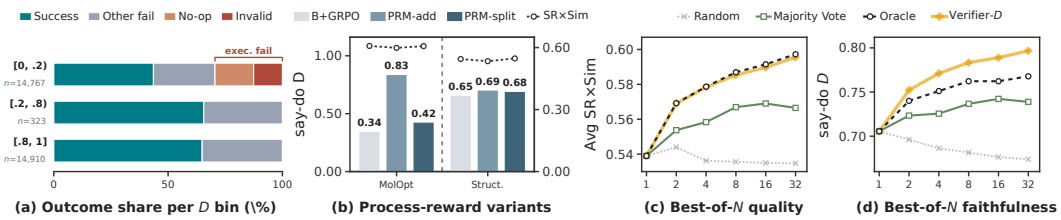



Figure 4: **Uses of executable say–do consistency.** (a) Low $D$ concentrates execution failures. (b) Process rewards improve $D$ without improving joint outcome. (c,d) $D$-based Best-of-$N$ improves outcome and say–do consistency without oracle access.

Training dynamics further explain why this diagnostic signal does not translate directly into better optimization (Figure 9). Process rewards encourage consistency with the declared edit, whereas the outcome reward encourages edits that achieve the requested objective. Improving faithfulness can therefore proceed without further improving editing performance, making the verifier better suited for candidate selection than direct policy optimization.

**$D$ supports verifier-guided test-time selection.** We therefore evaluate $D$ as a test-time selector over candidates from a fixed policy. We focus on the TOMG Structure subtasks, where $D$ directly checks whether the requested graph operation is realized while satisfying the declared preservation constraints. For each prompt, we select the highest-$D$ candidate among $N$ samples and compare with random selection, majority voting over canonical answers, and an oracle that maximizes $\mathrm{SR}\times\mathrm{Sim}$.

As $N$ increases from 1 to 32, $D$-based selection raises the selected candidate's $D$ from $0.706$ to $0.797$ and $\mathrm{SR}\times\mathrm{Sim}$ from $0.539$ to $0.596$, recovering $97\%$ of the oracle selector's gain (Figure 4c and d, Section C.3.6). Thus, $D$ provides a strong reference-answer-free signal, approaching oracle-level editing performance while also improving say–do consistency without changing the fixed policy.

**LLM-based construction extends TRACE beyond domain tools.** Finally, we test whether the same checkable interface can be instantiated when task-specific construction tools are unavailable. We replace the deterministic construction pipeline with Opus 4.8 (Anthropic, 2026), which generates traces from the instruction and input–target SMILES under the same schema. After SFT+GRPO, the resulting model reaches an average $\mathrm{SR}\times\mathrm{Sim}$ of $0.532$, improving over SMILES-only training ($0.5165$) but remaining below tool-built TRACE ($0.5762$) (Table 15).

The no-tool traces are also less aligned with the native verifier (Table 16). Although both constructors produce complete schemas ($C = 1.00$), no-tool TRACE has lower average $K/D$ across TOMG ($0.074/0.377$ vs. $0.668/0.490$). This suggests that a frontier LLM can preserve the TRACE interface without construction-time domain tools, while tool-derived facts provide stronger executable grounding. Full construction details and per-task analysis are provided in Section C.4.3.

### 3.4 How Does TRACE Improve Editing?

We next ask why TRACE improves constrained scientific editing. Unlike answer-only supervision, TRACE requires the model to state a structured reasoning trace before producing the final answer. Among its $G/L/S$ components, Local Edit ($L$) is the only field that specifies an input-dependent transformation: Goal states what should be achieved, Safeguard states what should be preserved, and $L$ identifies the concrete site–before–after change for the current input (Figure 1).

Although the outcome reward never reads the trace, its trajectory-level advantage is applied to the full generated sequence, so successful rollouts can reinforce trace tokens that precede the answer. This makes $L$ a plausible action-bearing component, but does not establish that the model actually depends on it. We therefore test this hypothesis with three interventions.

**Training requires an input-matched Local Edit.** The first question is whether TRACE helps because it names a molecule-specific edit, or merely because it gives the model a structured prefix. We compare two variants: *Remove* replaces $L$ with a fixed placeholder, while *Shuffled* substitutes an edit from another molecule under the same request. As shown in Figure 5a, *Remove* reduces the six-task composite from $0.5762$ to $0.5459$, but remains above the answer-only SMILES baseline ($0.5165$), showing that the remaining contract structure still provides useful supervision. *Shuffled* reaches $0.5537$ despite retaining a plausible edit-shaped field, but still falls below the *Full* contract,

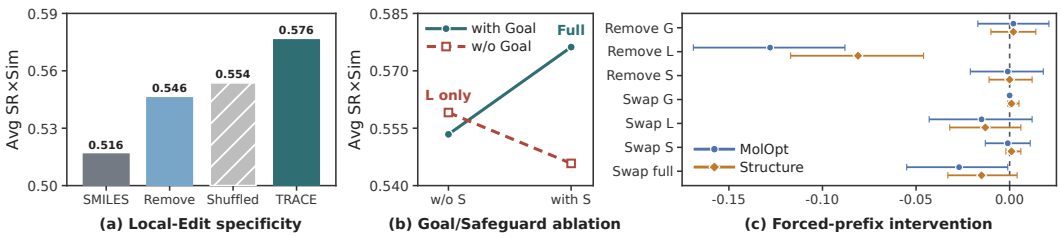



Figure 5: **Local Edit carries the direct action signal within the complete TRACE contract.** (a) Removing or shuffling $L$ lowers the six-task composite. (b) The complete $G/L/S$ contract performs best; component effects are non-additive. (c) Only removing $L$ clearly lowers forced-prefix SR; points show changes from the clean prefix with paired-bootstrap 95% confidence intervals.

suggesting that the benefit of $L$ comes from matching the edit to the current input rather than from the presence of an edit field alone (Section C.5.1).

**Contract components are complementary.** The first ablation identifies $L$ as the main candidate for the mechanism, but it does not imply that the rest of the contract is dispensable. We therefore test how $L$ interacts with the other contract components by holding the matched Local Edit fixed and varying the inclusion of $G$ and $S$. As shown in Figure 5b, $L$ alone already recovers most of the full-contract performance ($0.5591$ vs. $0.5762$), while adding either $G$ or $S$ alone provides no further gain ($0.5534$ and $0.5458$). The complete $G/L/S$ contract nevertheless performs best, indicating a non-additive interaction among the components: $L$ specifies the transformation, while $G$ and $S$ jointly provide the objective and preservation constraints needed to realize its full benefit (Table 20 and section C.5.2).

**Decoding relies most on Local Edit.** The training ablations establish which supervision is useful; we next test which contract component the trained decoder actually relies on. Starting from each model-generated TRACE, we remove or swap one $G/L/S$ component and re-decode only the answer against an unchanged forced-prefix control. As shown in Figure 5c, removing $L$ sharply reduces SR across both task families, whereas removing $G$ or $S$ changes SR by at most $0.010$. Swapping components produces the same ordering with smaller effects: perturbing $L$ hurts more than perturbing either $G$ or $S$ (Table 21 and section C.5.3). A descriptive attention probe provides consistent supporting evidence, with answer-token readout concentrating on Local Edit (Section C.5.4).

Together, these interventions suggest that TRACE improves editing primarily by making the model state an input-matched transformation before producing the answer, with the full contract providing the complementary context needed to use that transformation effectively.

## 4 Related Work

**Faithfulness and executable verification.** Recent LLM-based molecular editing methods generate structured outputs with direct supervision or free-form reasoning (Li et al., 2026a; Dey et al., 2025; Li et al., 2026b). More broadly, free-form chain-of-thought may not faithfully reflect how an answer is produced (Lanham et al., 2023; Yu et al., 2026). Executable reasoning methods instead validate intermediate artifacts with solvers, formal traces, or test harnesses (Zhong et al., 2026; Liu et al., 2025). TRACE addresses a complementary question: whether the generated structured output executes the edit declared in its trace. This trace–answer interface supports direct auditing, process feedback, and reference-answer-free candidate selection. See Section D for a fuller comparison.

## 5 Conclusion

TRACE addresses the say–do gap in scientific editing by turning stated edits into executable contracts that can be checked against the final output. Across molecular editing, codon optimization, and reaction repair, TRACE improves over matched answer-only training, with consistent gains across model scales, backbones, and optimization settings. Controlled interventions further provide evidence that the input-specific Local Edit carries most of the action-relevant signal. Beyond auditing, the same verifier also supports process supervision and verifier-guided candidate selection. **Limitations and future work.** TRACE relies on domain tools to construct executable supervision, although frontier LLMs can provide a partial fallback when such tools are unavailable. We leave further development of the tool-free setting to future work for broader applicability.

### AI Use Statement

Our main data-construction pipeline does not rely on generative AI; we use a frontier LLM only in two controlled comparison settings, either to instantiate the TRACE protocol without domain tools or to construct training data through a conventional LLM-based pipeline. Generative AI was not used to formulate hypotheses or mathematical claims, design the methodology or experiments, implement the methods, or interpret the experimental results. We also used generative AI for grammar checking and language polishing of the manuscript. All AI-assisted content and experimental artifacts were reviewed by the authors, who take responsibility for the final content of this work.

### Ethics Statement

Our experiments use public scientific datasets and involve no human subjects, personal data, or personally identifiable information. Molecular and biological editing can have dual-use applications; however, our experiments are limited to standard benchmark objectives and computational evaluation, without synthesis, experimental validation, or deployment. The generated outputs should therefore not be treated as experimentally validated chemical or biological designs.

### Reproducibility Statement

We provide the method and experimental protocol in Sections 2 and 3.1, with detailed data construction, contract definitions, checkers, reward functions, training settings, and evaluation protocols documented in the appendix. Code, configurations, and the constructed datasets will be released upon acceptance to support reproduction of the reported results.

## References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms, 2024. URL https://arxiv.org/abs/2402.14740.

Anthropic. Claude opus 4.8. https://www.anthropic.com/claude/opus, 2026. Accessed: 2026-07-18.

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful, 2026. URL https://arxiv.org/abs/2503.08679.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025. URL https://arxiv.org/abs/2503.11926.

Parit Bansal, Anne Morgat, Kristian B Axelsen, Venkatesh Muthukrishnan, Elisabeth Coudert, Lucila Aimo, Nevila Hyka-Nouspikel, Elisabeth Gasteiger, Arnaud Kerhornou, Teresa Batista Neto, Monica Pozzato, Marie-Claude Blatter, Alex Ignatchenko, Nicole Redaschi, and Alan Bridge. Rhea, the reaction knowledgebase in 2022. *Nucleic Acids Research*, 50(D1):D693–D700, 01 2022. ISSN 0305-1048. doi: 10.1093/nar/gkab1016. URL https://doi.org/10.1093/nar/gkab1016.

Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools, 2023. URL https://arxiv.org/abs/2304.05376.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. URL https://arxiv.org/abs/2211.12588.

Yanda Chen, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, Peter Hase, Misha Wagner, Fabien Roger, Vlad Mikulik, Samuel R. Bowman, Jan Leike, Jared Kaplan, and Ethan Perez. Reasoning models don't always say what they think, 2025. URL https://arxiv.org/abs/2505.05410.

Vishal Dey, Xiao Hu, and Xia Ning. Gellmo: Generalizing large language models for multi-property molecule optimization, 2025. URL https://arxiv.org/abs/2502.13398.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Carl Edwards, ChengXiang Zhai, and Heng Ji. Text2Mol: Cross-modal molecule retrieval with natural language queries. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 595–607, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.47. URL https://aclanthology.org/2021.emnlp-main.47/.

Carl Edwards, Tuan Lai, Kevin Ros, Garrett Honke, Kyunghyun Cho, and Heng Ji. Translation between molecules and natural language, 2022. URL https://arxiv.org/abs/2204.11817.

Yin Fang, Xiaozhuan Liang, Ningyu Zhang, Kangwei Liu, Rui Huang, Zhuo Chen, Xiaohui Fan, and Huajun Chen. Mol-instructions: A large-scale biomolecular instruction dataset for large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2306.08018.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023. URL https://arxiv.org/abs/2211.10435.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M. Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c1bTcrDmt4.

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, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin

Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Kevin Han, Renfei Zhang, Kathy Wei, Hamed Mahdavi, Niloofar Mireshghallah, and Amir Barati Farimani. SMDD-Bench: Can LLMs solve real-world small molecule drug design tasks?, 2026. URL https://arxiv.org/abs/2605.21740.

Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4198–4205, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.386. URL https://aclanthology.org/2020.acl-main.386/.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical graph-to-graph translation for molecules, 2019. URL https://arxiv.org/abs/1907.11223.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, Scott Emmons, Owain Evans, David Farhi, Ryan Greenblatt, Dan Hendrycks, Marius Hobbhahn, Evan Hubinger, Geoffrey Irving, Erik Jenner, Daniel Kokotajlo, Victoria Krakovna, Shane Legg, David Lindner, David Luan, Aleksander Mądry, Julian Michael, Neel Nanda, Dave Orr, Jakub Pachocki, Ethan Perez, Mary Phuong, Fabien Roger, Joshua Saxe, Buck Shlegeris, Martín Soto, Eric Steinberger, Jasmine Wang, Wojciech Zaremba, Bowen Baker, Rohin Shah, and Vlad Mikulik. Chain of thought monitorability: A new and fragile opportunity for ai safety, 2025. URL https://arxiv.org/abs/2507.11473.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, Kamilė Lukošiūtė, Karina Nguyen, Newton Cheng, Nicholas Joseph, Nicholas Schiefer, Oliver Rausch, Robin Larson, Sam McCandlish, Sandipan Kundu, Saurav Kadavath, Shannon Yang, Thomas Henighan, Timothy Maxwell, Timothy Telleen-Lawton, Tristan Hume, Zac Hatfield-Dodds, Jared Kaplan, Jan Brauner, Samuel R. Bowman, and Ethan Perez. Measuring faithfulness in chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2307.13702.

Jiatong Li, Yunqing Liu, Wenqi Fan, Xiao-Yong Wei, Hui Liu, Jiliang Tang, and Qing Li. Empowering molecule discovery for molecule-caption translation with large language models: A chatgpt perspective. *IEEE Transactions on Knowledge and Data Engineering*, 36 (11):6071–6083, November 2024. ISSN 2326-3865. doi: 10.1109/tkde.2024.3393356. URL http://dx.doi.org/10.1109/TKDE.2024.3393356.

Jiatong Li, Junxian Li, Weida Wang, Yunqing Liu, Changmeng Zheng, Yatao Bian, Dongzhan Zhou, Xiao yong Wei, and Qing Li. Speak-to-structure: Evaluating llms in open-domain natural language-driven molecule generation, 2026a. URL https://arxiv.org/abs/2412.14642.

Sizhen Li, Saeed Moayedpour, Ruijiang Li, Michael Bailey, Saleh Riahi, Milad Miladi, Jacob Miner, Dinghai Zheng, Jun Wang, Akshay Balsubramani, Khang Tran, Minnie Zacharia, Monica Wu, Xiaobo Gu, Ryan Clinton, Carla Asquith, Joseph Skalesk, Lianne Boeglin, Sudha Chivukula, Anusha Dias, Fernando Ulloa Montoya, Vikram Agarwal, Ziv Bar-Joseph, and Sven Jager. Codonbert: Large language models for mrna design and optimization. *bioRxiv*, 2023. doi: 10.1101/2023.09.09.556981. URL https://www.biorxiv.org/content/early/2023/09/12/2023.09.09.556981.

Xuan Li, Zhanke Zhou, Zongze Li, Jiangchao Yao, Yu Rong, Lu Zhang, and Bo Han. Reference-guided policy optimization for molecular optimization via llm reasoning, 2026b. URL https://arxiv.org/abs/2603.05900.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Yujian Liu, Jiabao Ji, Yang Zhang, Wenbo Guo, Tommi S. Jaakkola, and Shiyu Chang. HarnessLLM: Automatic testing harness generation via reinforcement learning, 2025. URL https://arxiv.org/abs/2511.01104.

Zequn Liu, Wei Zhang, Yingce Xia, Lijun Wu, Shufang Xie, Tao Qin, Ming Zhang, and Tie-Yan Liu. Molxpt: Wrapping molecules with text for generative pre-training, 2023. URL https://arxiv.org/abs/2305.10688.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning, 2023. URL https://arxiv.org/abs/2301.13379.

MiniMax, Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, Chengjun Xiao, Chengyu Du, Chi Zhang, Chu Qiao, Chunhao Zhang, Chunhui Du, Congchao Guo, Da Chen, Deming Ding, Dianjun Sun, Dong Li, Enwei Jiao, Haigang Zhou, Haimo Zhang, Han Ding, Haohai Sun, Haoyu Feng, Huaiguang Cai, Haichao Zhu, Jian Sun, Jiaqi Zhuang, Jiaren Cai, Jiayuan Song, Jin Zhu, Jingyang Li, Jinhao Tian, Jinli Liu, Junhao Xu, Junjie Yan, Junteng Liu, Junxian He, Kaiyi Feng, Ke Yang, Kecheng Xiao, Le Han, Leyang Wang, Lianfei Yu, Liheng Feng, Lin Li, Lin Zheng, Linge Du, Lingyu Yang, Lunbin Zeng, Minghui Yu, Mingliang Tao, Mingyuan Chi, Mozhi Zhang, Mujie Lin, Nan Hu, Nongyu Di, Peng Gao, Pengfei Li, Pengyu Zhao, Qibing Ren, Qidi Xu, Qile Li, Qin Wang, Rong Tian, Ruitao Leng, Shaoxiang Chen, Shaoyu Chen, Shengmin Shi, Shitong Weng, Shuchang Guan, Shuqi Yu, Sichen Li, Songquan Zhu, Tengfei Li, Tianchi Cai, Tianrun Liang, Weiyu Cheng, Weize Kong, Wenkai Li, Xiancai Chen, Xiangjun Song, Xiao Luo, Xiao Su, Xiaobo Li, Xiaodong Han, Xinzhu Hou, Xuan Lu, Xun Zou, Xuyang Shen, Yan Gong, Yan Ma, Yang Wang, Yiqi Shi, Yiran Zhong, Yonghong Duan, Yongxiang Fu, Yongyi Hu, Yu Gao, Yuanxiang Fan, Yufeng Yang, Yuhao Li, Yulin Hu, Yunan Huang, Yunji Li, Yunzhi Xu, Yuxin Mao, Yuxuan Shi, Yuze Wenren, Zehan Li, Zelin Li, Zhanxu Tian, Zhengmao Zhu, Zhenhua Fan, Zhenzhen Wu, Zhichao Xu, Zhihang Yu, Zhiheng Lyu, Zhuo Jiang, Zibo Gao, Zijia Wu, Zijian Song, and Zijun Sun. Minimax-m1: Scaling test-time compute efficiently with lightning attention, 2025. URL https://arxiv.org/abs/2506.13585.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023. URL https://arxiv.org/abs/2306.02707.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe

Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting, 2023. URL https://arxiv.org/abs/2305.04388.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

Weichu Xie, Haozhe Zhao, Wenpu Liu, Yongfu Zhu, Liang Chen, Minghao Ye, Zirong Chen, Yuqi Xu, Shuai Dong, Ziyue Wang, Xinbo Xu, Kean Shi, Ruoyu Wu, Xiaoying Zhang, Wenqi Shao, Baobao Chang, Nan Duan, and Jiaqi Wang. Step-wise rubric rewards for llm reasoning, 2026. URL https://arxiv.org/abs/2605.17291.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Geyan Ye, Xibao Cai, Houtim Lai, Xing Wang, Junhong Huang, Longyue Wang, Wei Liu, and Xiangxiang Zeng. Drugassist: A large language model for molecule optimization, 2023. URL https://arxiv.org/abs/2401.10334.

Qinan Yu, Alexa Tartaglini, Peter Hase, Carlos Guestrin, and Christopher Potts. Outcome rewards do not guarantee verifiable or causally important reasoning, 2026. URL https://arxiv.org/abs/2604.22074.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

Ninghan Zhong, Ahmet Ege Tanriverdi, Kaan Kale, and Sriram Vishwanath. VeryTrace: Verifying reasoning traces through compilable formalism and structured verification, 2026. URL https://arxiv.org/abs/2606.24124.

## APPENDIX CONTENTS

**A. Datasets and Experimental Setup**

- A.1 Benchmark and Dataset Details
- A.2 Evaluation Protocol
- A.3 Training and Reproducibility
- A.4 Outcome Rewards

**B. Data and Contract Construction**

- B.1 TRACE Construction Procedure
- B.2 TOMG Instantiation
- B.3 MUMO Instantiation
- B.4 TDC-PairMolGen Instantiation
- B.5 Codon and Rhea Instantiations
- B.6 Construction Audit Summary

**C. Additional Experiments and Analyses**

- C.1 Optimizer Robustness
- C.2 Cross-Modal Executable Tasks
  - C.2.1 Codon Optimization
  - C.2.2 Reaction Repair (Rhea)
- C.3 Verifier Diagnostics and Utilities
  - C.3.1 Per-Task $C/K/D$ Definitions
  - C.3.2 Why the Main Faithfulness Metric Is $D$
  - C.3.3 Say–Do Consistency Across Scale
  - C.3.4 Say–Do $D$ as a Failure/Audit Signal
  - C.3.5 Process-Reward Faithfulness Ablation
  - C.3.6 Verifier-Guided Selection
  - C.3.7 Successful Outcome with an Auditable Claim Error
- C.4 Trace Surface Controls
  - C.4.1 Prompting the Contract Without Training
  - C.4.2 Closed-Source Narrative Controls
  - C.4.3 Constructing TRACE Without Domain Tools
  - C.4.4 Outcome-Competitive Free-Form Traces Lack Native Verification
  - C.4.5 Audit Signal Tracks Contract Structure, Not Trace Length
  - C.4.6 Reproduction Fidelity Against RePO
- C.5 Where the Gains Come From: Local Edit in the TRACE Contract
  - C.5.1 The Supervised Edit Must Match the Input
  - C.5.2 The Contract Is Not Reducible to Local Edit
  - C.5.3 The Decoder Reads Local Edit at Inference Time
  - C.5.4 Attention Provides a Descriptive Readout
  - C.5.5 Where the Gains Land

**D. Expanded Related Work**

**E. Detailed Result Tables**

# A Datasets and Experimental Setup

This section provides the experimental details underlying the main results. We first describe the benchmarks, data splits, and task-specific success criteria (Section A.1), followed by the shared evaluation protocol (Section A.2). We then specify the training and checkpoint-selection settings used in the controlled comparisons (Section A.3) and the outcome rewards used by the different training arms (Section A.4).

## A.1 Benchmark and Dataset Details

All molecular benchmarks use SMILES inputs and RDKit-compatible molecular parsing.

**TOMG.** TOMG (Li et al., 2026a) is our primary constrained molecule-editing benchmark. We use all six instruction subtasks: three single-property optimization tasks (LogP, MR, and QED) and three explicit structural operations (AddComponent, DelComponent, and SubComponent).

The original supervised pool contains 3000 rows (500 per subtask). We exclude eight rows before training: two have no executable trace, four contain anomalous targets, one violates the operation ontology, and one is unstable under RDKit canonicalization. The resulting 2992 rows are shared by all TOMG training arms, with $499/500/498/500/496/499$ examples for LogP/MR/QED/AddComponent/ DelComponent/SubComponent, respectively. Evaluation uses 5000 prompts per subtask.

For MolOpt, success requires a valid, non-noop molecule whose requested property moves in the instructed direction. For Structure, success requires a valid, non-noop molecule that executes the requested component operation while satisfying the benchmark's scaffold and similarity constraints.

**MUMO.** MuMOInstruct (Dey et al., 2025) evaluates multi-property molecule optimization. We use the BDP, BDQ, and BPQ tasks on both seen and unseen instruction splits, where the letters denote combinations of BBBP, DRD2, penalized logP, and QED objectives defined by the benchmark. Each property combination contains 1500 supervised training examples, yielding 4500 SFT examples in total.

Evaluation uses 500 seen and 500 unseen prompts for each property combination, for 3000 prompts in total before generation seeds. A prediction is successful only when all requested properties improve under the benchmark evaluator; molecular similarity is measured against the input molecule.

**TDC-PairMolGen.** TDC-PairMolGen is derived from the molecule-pair generation setting introduced by Jin et al. (2019). We evaluate three single-property redesign objectives: LogP, QED, and DRD2. Each task contains 1500 training pairs and 500 reference-answer-free evaluation prompts.

The source–target pairs represent substantially more diffuse redesigns than TOMG: their mean training-pair Tanimoto similarity is approximately 0.43, and most involve scaffold changes. Evaluation does not use the curated target. Instead, the task oracle is recomputed on each generated molecule, and success requires a valid, non-noop output that improves the requested property.

**Codon optimization.** Codon optimization follows the CodonBERT setting (Li et al., 2023). Each input is a valid RNA coding sequence with an AUG start codon, a terminal stop codon, no internal stop codons, and at most 100 codons. The target is a synonymous coding sequence encoding the same protein and reaching human CAI of at least $\min(0.99, \mathrm{CAI}_{\mathrm{in}} + 0.08)$, with an optional GC-content constraint.

We use 350 training, 119 development, and 300 evaluation examples. The splits are protein-hash-disjoint, with no overlap between development and evaluation prompts. Success requires a valid synonymous edit that preserves the encoded protein and satisfies the row-specific CAI and, when present, GC constraint.

**Reaction repair.** Reaction repair uses Rhea release 141 (Bansal et al., 2022). From the release, we parse 14,002 atom- and charge-balanced reactions and generate corrupted instances by modifying coefficients, removing participants, or moving participants across reaction sides. We restrict corruption depth to 3 or 4 edits.

We use 350 training, 100 development, and 300 evaluation reactions with reaction-disjoint splits. Each prompt contains the corrupted reaction together with the molecular formula and charge of each

participant, and the model outputs a sparse slot-indexed repair program. Evaluation is reference-answer-free: success requires the repaired reaction to be parseable, non-degenerate, atom-balanced, and charge-balanced, rather than to exactly recover the original Rhea reaction. Further task and verifier details for Codon and Rhea are given in Section C.2.

### A.2 Evaluation Protocol

For molecular tasks, preservation is measured using Morgan-radius-2, 2048-bit Tanimoto similarity between the generated and input molecules unless stated otherwise. Task success is determined by the benchmark-specific criteria described in Section A.1; the reported molecular joint score is the per-sample $\mathrm{SR} \times \mathrm{Sim}$ metric defined in the main text.

Unless otherwise noted, generation uses temperature $0.10$, top-$p = 1.0$, and a maximum generation length of $1024$ tokens, with one sampled output per prompt. Each fixed checkpoint is evaluated with generation seeds $42$, $43$, and $44$, and results are reported as mean$\pm$std across these three decoding runs.

The reported variation therefore measures generation variability for a fixed trained checkpoint, not variability across independent training runs. All reported training arms are trained once; training-seed and checkpoint-selection details are given in Section A.3.

### A.3 Training and Reproducibility

Outcome RL starts from the corresponding SFT checkpoint and samples eight generations per prompt. Within each controlled comparison, all arms use the same training budget and task-family-specific outcome reward (Section A.4). All reported arms are trained once with training seed $42$.

For TOMG, MUMO, and TDC, we evaluate the last-epoch checkpoint. For Codon and Rhea, both arms use predeclared development-set checkpoint selection (Section C.2); this accounts for the different GRPO steps reported in Table 6.

**Answer-token weighting control.** TRACE introduces additional supervised trace tokens, reducing the relative weight of the answer span under a uniform token-level loss. We therefore use answer-token weight $\alpha = 2$ for the main TOMG TRACE SFT run. Answer-only SFT requires no corresponding adjustment because all supervised tokens belong to the answer.

We additionally retrain TRACE with uniform weighting ($\alpha = 1$), keeping the data, SFT and GRPO recipes, outcome reward, training seed, and evaluation protocol unchanged. After GRPO, this arm reaches $0.5812$ average $\mathrm{SR} \times \mathrm{Sim}$, compared with $0.5762$ for the main $\alpha = 2$ configuration and $0.5165$ for the matched answer-only baseline (Table 2). The TRACE gain therefore does not depend on answer-token reweighting.

Table 2: **Answer-token reweighting control on TOMG.** Qwen3-1.7B average $\mathrm{SR} \times \mathrm{Sim}$ under the main TOMG protocol. Removing the TRACE-only reweighting leaves the +GRPO advantage over the matched answer-only arm intact. Cells are mean$\pm$std over generation seeds 42/43/44.

| Arm | $\alpha$ | SFT | SFT+GRPO |
|---|---|---|---|
| SFT-SMILES | n/a | $0.3968_{\pm 0.0077}$ | $0.5165_{\pm 0.0001}$ |
| SFT-TRACE | 1 | $0.4225_{\pm 0.0007}$ | $\mathbf{0.5812}_{\pm 0.0005}$ |
| SFT-TRACE (main) | 2 | $0.4251_{\pm 0.0005}$ | $0.5762_{\pm 0.0004}$ |

Upon acceptance, we will release the construction, training, and evaluation code together with the per-benchmark extractors, contract schemas, renderers, checkers, and exact experimental configurations.

### A.4 Outcome Rewards

All main TRACE and matched answer-only arms use answer-only outcome rewards: the generated trace is not inspected by these rewards. We first describe the RePO objective used for the free-form baseline, followed by the task-specific rewards used in our matched comparisons.

**RePO reward and reproduction.** The RePO paper (Li et al., 2026b) describes a reference-guided GRPO reward consisting of a binary property-improvement term plus Tanimoto similarity. For candidate molecule $m$ and input molecule $m_0$,

$$r_{\text{RePO}}(m, m_0) = r_{\text{prop}}(m, m_0) + r_{\text{struct}}(m, m_0), \tag{9}$$

$$r_{\text{struct}}(m, m_0) = \frac{|FP(m) \cap FP(m_0)|}{|FP(m) \cup FP(m_0)|}, \tag{10}$$

$$r_{\text{prop}}(m, m_0) = \begin{cases} 1, & F(m) \geq F(m_0) \quad \text{for an increase instruction,} \\ 1, & F(m) \leq F(m_0) \quad \text{for a decrease instruction,} \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Its public implementation instead applies validity, non-noop, and similarity gates followed by the continuous shaped reward

$$r^{\text{impl}}_{\text{RePO}} = 0.3T + 0.7 \text{ clip}\left(\frac{\Delta_{\mathbf{c}} F}{\sigma_F}, -1, 1\right), \tag{12}$$

where $T$ is Morgan-radius-2, 2048-bit Tanimoto similarity, $\Delta_{\mathbf{c}} F$ is the property change signed in the requested direction, and $\sigma_F$ is the corresponding property scale. All RePO results in this paper optimize this implementation-level objective rather than our task-specific rewards. For MUMO and TDC properties not defined by RePO, we use the corresponding benchmark oracle while retaining the RePO reward shape and gates. Trainer, KL coefficient, epoch budget, and evaluation protocol match the corresponding +GRPO comparisons except for the method-specific objective. Section C.4.6 further discusses the fidelity of our TOMG reproduction.

**Matched task-specific rewards.** Let $T = \text{Tan}(\mathbf{s}_{\text{out}}, \mathbf{s}_{\text{in}})$. TOMG MOLOPT gives positive reward only to valid, non-noop outputs that improve the requested property while retaining a minimum similarity of 0.1:

$$R_{\text{MolOpt}} = \mathbb{1}[\text{valid} \wedge \text{non-noop} \wedge \text{dir.-correct} \wedge T \geq 0.1]\,(1 + 0.3T). \tag{13}$$

TOMG STRUCTURE instead uses the benchmark's operation-specific structural score $S_{\text{struct}} \in [0, 1]$:

$$R_{\text{Struct}} = \mathbb{1}[\text{valid} \wedge \text{non-noop} \wedge \text{scaffold} \wedge S_{\text{struct}} > 0]\,(0.3T + 0.7S_{\text{struct}}). \tag{14}$$

Here, scaffold requires $T \geq 0.4$, $H_{\text{out}}/H_{\text{in}} \geq 0.65$, and $H_{\text{out}} \geq 4$, where $H_{\text{in}}$ and $H_{\text{out}}$ are the input and output heavy-atom counts. For MUMO, let $K$ be the number of requested properties and $k$ the number satisfied by the output. We use the tiered reward

$$R_{\text{MUMO}} = \begin{cases} 0.4 + 0.6T, & k = K, \\ 0.3\,k/K, & 0 < k < K, \\ 0, & k = 0. \end{cases} \tag{15}$$

Every all-property success therefore outranks every partial success, while similarity ranks candidates within the successful tier.

TDC uses the corresponding benchmark oracle in place of the TOMG property-direction test:

$$R_{\text{TDC}} = \mathbb{1}[\text{valid} \wedge \text{non-noop} \wedge F_{\text{task}}(\mathbf{s}_{\text{out}}) > F_{\text{task}}(\mathbf{s}_{\text{in}})]\,(1 + 0.3T), \tag{16}$$

where $F_{\text{task}}$ is the TDC oracle for DRD2, QED, or LOGP.

**Molecular no-op handling.** For TOMG, MUMO, and TDC, no-ops are detected by equality of RDKit-canonicalized input and output SMILES. TOMG MOLOPT rejects canonical no-ops directly. For RePO-style continuous rewards, we additionally suppress numerical property noise by setting the reward to zero when the canonical molecules are identical and the normalized property change is below $\epsilon = 10^{-3}$. We use a similarity floor of 0.1. The TOMG property scales are $\sigma_{\text{LogP}} = 0.81$, $\sigma_{\text{MR}} = 9.20$, and $\sigma_{\text{QED}} = 0.089$, computed from the mean curated target delta of the corresponding training split. The TOMG STRUCTURE reward uses $(w_{\text{sim}}, w_{\text{struct}}) = (0.3, 0.7)$.

**Codon optimization.** Let $S_{\text{codon}}$ be the fraction of unchanged codons between the input and output coding sequence. The outcome reward is

$$G_{\text{Codon}} = \text{valid-CDS} \wedge \text{length/stop} \wedge \text{protein-preserved} \wedge \text{non-noop} \\ \wedge \text{CAI} \geq \text{target} \wedge \text{GC-band}, \tag{17}$$

$$R_{\text{Codon}} = \mathbb{1}[G_{\text{Codon}}]\,(1 + 0.3S_{\text{codon}}), \tag{18}$$

Table 3: **TRACE roles across task instantiations and TOMG controls.** The first five rows show how task-specific TRACE contracts instantiate the Goal ($G$), Local Edit/evidence ($L$), and Safeguard ($S$) roles. The final two rows show the TOMG narrative controls used to separate typed contracts from free-form reasoning and untyped operation cues.

| Instantiation | $G$: goal context | $L$: edit / evidence signal | $S$: safeguards and checks |
|---|---|---|---|
| **TOMG TRACE** *Local edit* | Requested objective, editing strategy, and property or structural intent. | Typed (SITE, BEFORE, AFTER) edits that specify the input-specific molecular change. | Parent/scaffold preservation, molecular validity, and replay checks for the declared edits and group changes. |
| **MUMO TRACE** *Local / partially diffuse* | Requested property directions and a whole-molecule editing strategy. | Local-difference evidence with signed group and heavy-atom changes; construction records whether the edit is *extractable*, *partial*, or *diffuse*. | Preservation of verifier-supported groups, together with validity, property-direction, delta, and unchanged-group checks. |
| **TDC TRACE** *Diffuse redesign* | Target property and its measurement mode for LogP, QED, or DRD2. | Structural and property evidence supporting the redesign, without a hard local-edit claim. | Evidence-consistency and property rechecks; no unsupported preserve claim is imposed for diffuse source–target changes. |
| **Codon TRACE** *Indexed sequence edit* | Increase human CAI while preserving the encoded protein and, when specified, the GC range. | Sparse indexed codon substitutions (`pos:before>after`) executed against the source sequence. | Position/source checks, synonymous substitutions, start/stop preservation, protein identity, and CAI/GC constraints. |
| **Rhea TRACE** *Indexed reaction repair* | Restore feasibility of a corrupted biochemical reaction. | Sparse slot-level coefficient, deletion, or side-move edits forming an executable repair program. | Parseability, nondegeneracy, and atom/charge-balance checks on the repaired reaction. |
| **narrative** *Control* | Free-form description of the requested property or structural change. | Free-form reasoning without a dedicated executable edit field. | No native replayable preserve or check fields; coarse claims are recoverable only through the retrofit parser. |
| **narrative + ops** *Control* | The same free-form goal description. | The narrative augmented with deterministic untyped cues such as *remove Cl* or *add S*, but without site–before–after binding. | No native preserve or replay-check fields, isolating explicit operation cues from the full TRACE contract. |

where the GC-band gate is active only for rows that specify one.

**Rhea reaction repair.** Rhea uses the feasibility reward

$$R_{\text{Rhea}} = \mathbb{1}[\text{parseable} \wedge \text{atom-balanced} \wedge \text{charge-balanced} \wedge \text{nondegenerate}] . \tag{19}$$

Similarity to the corrupted input is not part of the task.

# B Data and Contract Construction

This section specifies how TRACE training targets are constructed from task-specific facts. We first give the shared sample-level procedure (Section B.1), then instantiate its fact layer, contract fields, and executable checks for TOMG, MUMO, TDC-PairMolGen, Codon optimization, and Rhea reaction repair. Table 3 summarizes how these task-specific contracts map onto the common Goal/Local-Edit/Safeguard ($G/L/S$) roles. We conclude with the shared fail-closed construction rule (Section B.6). All main TRACE corpora are constructed deterministically from task requirements, reference edits, and tool-computed facts; no LLM authors the per-example trace. LLM-authored construction controls are introduced separately in Section C.4.

## B.1 TRACE Construction Procedure

For each task family $\tau$, we use the fixed interface $\mathcal{I}_\tau = (\Phi_\tau, \Sigma_\tau, \Lambda_\tau, \mathcal{R}_\tau, \mathcal{X}_\tau)$ defined in Section 2.2. The interface components, including the checker, are shared across examples, while the extracted facts and the Goal, Local Edit, and Safeguard contents are instantiated for each input–reference pair. Figure 6 illustrates this construction, and Algorithm 1 specifies the sample-level procedure.

At evaluation, the extractor $\Phi_\tau$ recomputes facts from the input–answer pair, and the checker $\mathcal{X}_\tau$ tests the generated trace against these facts (Section 2.3). The same verification procedure is reused in the process-reward and test-time-selection experiments.

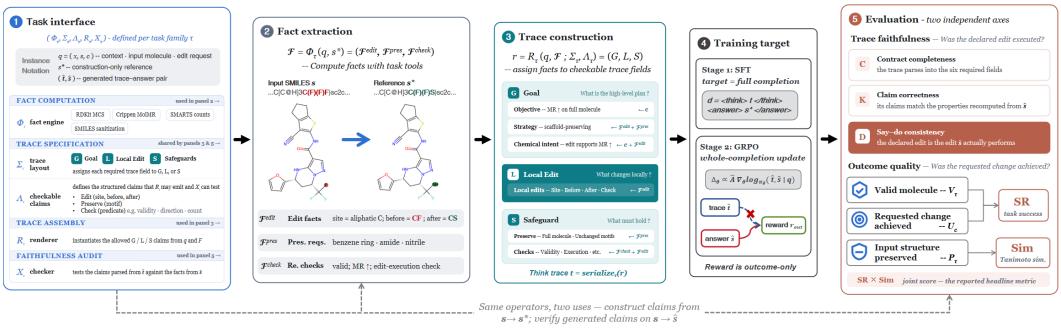



Figure 6: **TRACE data construction and verification.** Task-specific tools extract edit, preservation, and check facts from an input–reference pair. The renderer combines these facts with the query to construct a Goal–Local Edit–Safeguard trace, which is serialized and paired with the reference answer for SFT. Subsequent GRPO uses an outcome-only reward. Verification reuses the extractor and checker to compare the generated trace with facts recomputed from the input–answer pair, without a reference answer. The molecular pair is illustrative.

**Algorithm 1** TRACE data construction

**Require:** Query $q_i = (x_i, \mathbf{s}_i, \mathbf{c}_i)$ and construction-only reference $\mathbf{s}_i^\star$
**Require:** Construction components $(\Phi_\tau, \Sigma_\tau, \Lambda_\tau, \mathcal{R}_\tau)$ of the task-family interface $\mathcal{I}_\tau$
**Ensure:** Trace–answer training target $d_i$

$\mathcal{F}_i \leftarrow \Phi_\tau(q_i, \mathbf{s}_i^\star)$ ▷ extract facts from $\mathbf{s}_i \rightarrow \mathbf{s}_i^\star$
$\mathbf{r}_i \leftarrow \mathcal{R}_\tau(q_i, \mathcal{F}_i; \Sigma_\tau, \Lambda_\tau)$ ▷ render only licensed claims
$\mathbf{t}_i \leftarrow \text{serialize}_\tau(\mathbf{r}_i)$
$d_i \leftarrow$ <think>$\mathbf{t}_i$</think><answer>$\mathbf{s}_i^\star$</answer>
**return** $d_i$

### B.2 TOMG Instantiation

TOMG provides the most direct local-edit instantiation of TRACE because its source–target pairs are typically localized and can be represented by replayable molecular graph changes. The fact extractor identifies the edited site and its before/after components, together with requested property or structural operation, preserved-scaffold evidence, and relevant functional-group count changes.

The renderer maps these facts to the six model-facing fields introduced in Section 2.2: OBJECTIVE, STRATEGY, CHEMICAL INTENT, LOCAL EDITS, PRESERVE, and CHECKS. In particular, LOCAL EDITS records typed (SITE, BEFORE, AFTER) transformations, while PRESERVE and CHECKS expose scaffold, validity, and group-count constraints that can be recomputed from the answer.

A row is rendered only when these claims can be grounded in the extracted facts and replayed by the executor. The construction audit then replays the local edit, recomputes the preservation and count checks, and verifies that the <answer> span is the complete canonical reference molecule. The eight TOMG rows excluded by this fail-closed procedure are listed in Section A.1.

### B.3 MUMO Instantiation

MUMO extends TRACE to multi-property editing, where the source–target change is often less localized than in TOMG. The fact extractor therefore does not assume that every pair admits a single site-level transformation. Instead, it records the three requested property directions together with MCS-derived changed regions, functional-group and heavy-atom count changes, and dominant molecular motifs.

The renderer adapts the trace to the locality supported by these facts. When a clear local transformation is identified, it records the corresponding site-level before/after change. When the transformation is only partly localized, it reports the supported local changes together with the remaining net molecular changes. For diffuse transformations, it omits a unique local edit and records only verifiable aggregate

changes. Accordingly, each pair is classified as *extractable*, *partial*, or *diffuse*; depending on the MUMO task, 18–51% of source–target pairs are diffuse. This fail-closed treatment avoids attributing a specific (SITE, BEFORE, AFTER) edit when the source–target difference does not support one.

Preservation claims follow the same principle. A molecular group is included in PRESERVE only when its count is unchanged between the input and reference target. Property fields encode the requested directions rather than reference-specific oracle values, so the same claims can be evaluated on model outputs. The construction audit then recomputes every declared local or aggregate change and verifies all preservation claims.

### B.4 TDC-PAIRMOLGEN INSTANTIATION

TDC-PairMolGen represents the diffuse-redesign regime of TRACE. Its training pairs have a mean source–target Tanimoto similarity of approximately 0.43, and 95–99% involve scaffold hops. Unlike TOMG, these pairs therefore rarely admit either a unique (SITE, BEFORE, AFTER) transformation or a stable substructure that can be declared preserved.

TRACE adapts the trace to this weaker structural correspondence rather than forcing a local-edit interpretation. OBJECTIVE states the requested property change, while MODE DECISION records the type of evidence available for that property. STRUCTURAL EVIDENCE and PROPERTY EVIDENCE then report only recomputable source–target changes, without asserting a unique local edit. When no stable preservation claim can be supported, the trace instead constrains its own assertions: incomplete evidence is marked explicitly, and unsupported local or causal interpretations are omitted.

The form of property evidence depends on the oracle:

- **Additive (LOGP).** Penalized logP admits approximately atom-additive Crippen contributions. The trace can therefore report supported group-level directional contributions together with the recomputed property change.
- **Component (QED).** QED is not additive at the functional-group level. The trace reports only changes in QED components that can be recomputed directly and does not attribute the overall QED change to a particular molecular group.
- **Oracle (DRD2).** DRD2 activity is scored by a whole-molecule classifier and has no valid local decomposition into group-level contributions. The trace therefore reports the classifier-score change and any directly supported pharmacophore evidence, without assigning causal credit to a local motif.

All fields are rendered deterministically from RDKit and task-oracle outputs. At audit time, the same quantities are recomputed from the generated molecule rather than taken from the construction target. For TDC, the renderer further restricts claims to the available evidence: it avoids unsupported causal attributions, surfaces counter-evidence, and explicitly marks partial or missing mechanistic evidence.

### B.5 CODON AND RHEA INSTANTIATIONS

Codon optimization and Rhea reaction repair extend TRACE beyond molecular graph editing. In both tasks, the desired transformation can be represented as a sparse indexed edit program. This gives a direct executable correspondence between the rendered trace and the final answer: the constructor derives a legal program, the renderer exposes the supported edits and constraints, and a task-native executor applies the program and verifies the result.

**Codon optimization.** The fact layer records the indexed coding sequence, synonymous alternatives, human codon adaptiveness, the row-specific CAI target, any GC-content constraint, and the locked start and terminal-stop positions. During construction, synonymous substitutions are considered in descending $\Delta \log$-adaptiveness order until the CAI target is reached or the 16-edit budget is exhausted.

The renderer expresses these facts through OBJECTIVE, STRATEGY, CODON EDITS, PROPERTY INTENT, PRESERVE, and CHECKS. The answer is a sparse edit program with entries of the form `pos:before>after`, for example `5:CUG>CUC | 12:GCC>GCU`. The executor copies all unmentioned codons from the input and applies only the listed substitutions.

A program is rejected if it contains an invalid position, an incorrect `before` codon, duplicate or no-op edits, a nonsynonymous substitution, or a modification to the locked start or terminal stop. The construction audit then verifies protein preservation and attainment of the CAI target and, when specified, the GC-content constraint.

**Rhea reaction repair.** The fact layer records the corrupted reaction, indexed participant slots, molecular formulas and charges, and the sparse repair program needed to restore a balanced reaction. The corruption process perturbs coefficients, removes participants, or moves participants across reaction sides, as detailed in Section A.1.

The renderer exposes the supported slot-indexed repairs, including coefficient changes, participant drops, and side moves, and the answer is the corresponding executable program. The executor applies these edits and verifies parseability, nondegeneracy, atom balance, and charge balance.

The original Rhea reaction is used only to construct the supervised repair target. Evaluation does not require exact recovery of that reference reaction; any feasible output satisfying the balance constraints is accepted. Further task-level analyses are provided in Section C.2.

### B.6 Construction Audit Summary

TRACE construction follows a shared fail-closed rule: a supervised row is kept only when every rendered claim is supported by the task-specific fact layer and can be verified against the construction target by the corresponding executor or checker. Rows with unsupported or unverifiable claims are discarded rather than completed with inferred explanatory content.

For molecular tasks, the `<answer>` span contains the complete canonical reference molecule. For Codon and Rhea, it contains the sparse executable edit program applied to the input. In all cases, the reference is used only to construct the supervised target. At evaluation time, task success is recomputed from the generated answer and does not require recovery of the construction reference.

## C Additional Experiments and Analyses

This section extends the main results along five complementary directions. We first test whether TRACE's gain is robust to the choice of RL optimizer (Section C.1) and whether the same executable-contract supervision transfers beyond molecular graph editing (Section C.2). We then characterize the TRACE verifier and its uses for auditing, process feedback, and test-time selection (Section C.3). Trace-surface controls test whether the same benefit can be recovered through prompting, free-form reasoning supervision, or LLM-based construction (Section C.4). Finally, training and inference interventions examine why TRACE improves editing and provide evidence that Local Edit carries most of the action-relevant signal (Section C.5).

### C.1 Optimizer Robustness

We test whether TRACE's gains depend on GRPO by changing only the RL optimizer while keeping the training data, initialization, and outcome reward fixed. In addition to GRPO (Shao et al., 2024), we evaluate GSPO (Zheng et al., 2025), RLOO (Ahmadian et al., 2024), and CISPO (MiniMax et al., 2025) under the same comparison.

Table 4: **Optimizer robustness on TOMG-Bench with Qwen3-1.7B.** Results are averaged over all six tasks; cells are mean±std over generation seeds 42/43/44. Bold marks the better method between SFT-SMILES+GRPO and SFT-TRACE+GRPO under each optimizer.

| Metric | GRPO | | GSPO | | RLOO | | CISPO | |
|---|---|---|---|---|---|---|---|---|
| | SMILES | TRACE | SMILES | TRACE | SMILES | TRACE | SMILES | TRACE |
| SR | $0.6709_{\pm 0.0002}$ | $\mathbf{0.7564}_{\pm 0.0007}$ | $0.6711_{\pm 0.0007}$ | $\mathbf{0.7520}_{\pm 0.0023}$ | $0.6923_{\pm 0.0006}$ | $\mathbf{0.7694}_{\pm 0.0006}$ | $0.6706_{\pm 0.0002}$ | $\mathbf{0.7456}_{\pm 0.0008}$ |
| Sim | $\mathbf{0.7727}_{\pm 0.0000}$ | $0.7528_{\pm 0.0009}$ | $\mathbf{0.7655}_{\pm 0.0002}$ | $0.7463_{\pm 0.0008}$ | $\mathbf{0.7764}_{\pm 0.0001}$ | $0.7485_{\pm 0.0006}$ | $\mathbf{0.7728}_{\pm 0.0002}$ | $0.7260_{\pm 0.0001}$ |
| SR×Sim | $0.5165_{\pm 0.0001}$ | $\mathbf{0.5762}_{\pm 0.0004}$ | $0.5145_{\pm 0.0005}$ | $\mathbf{0.5734}_{\pm 0.0013}$ | $0.5384_{\pm 0.0004}$ | $\mathbf{0.5842}_{\pm 0.0008}$ | $0.5161_{\pm 0.0001}$ | $\mathbf{0.5603}_{\pm 0.0008}$ |

TRACE improves the final SR×Sim under all optimizers, with gains of about 0.044–0.060 (Table 4). The training curves in Figure 7 show that this advantage is not confined to the final checkpoint:

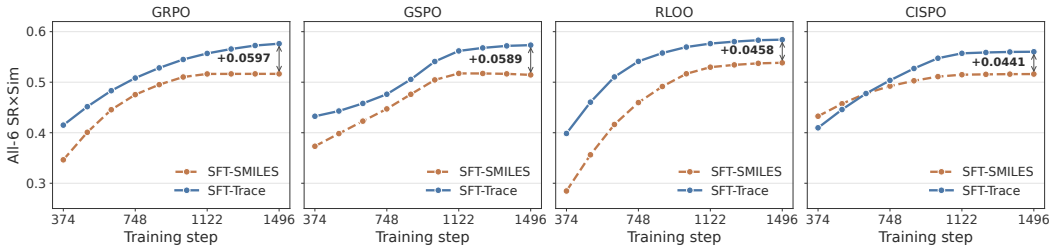



Figure 7: **Optimizer robustness on TOMG.** All-task per-sample SR $\times$ Sim under GRPO, GSPO, RLOO, and CISPO (Qwen3-1.7B, generation seeds 42/43/44). TRACE finishes above the matched answer-only arm for every optimizer.

TRACE stays ahead through most of training under GRPO, GSPO, and RLOO, while under CISPO it overtakes the answer-only arm after the early stage and remains ahead thereafter. The per-task results show the same pattern, with TRACE leading in 15 of 18 task–optimizer comparisons (Table 5). The gain is therefore robust to the choice of RL optimizer and is not driven by a single subtask.

Table 5: **Per-task optimizer robustness on TOMG-Bench.** Qwen3-1.7B results under GSPO, RLOO, and CISPO; cells are mean±std over generation seeds 42/43/44. Within each optimizer and metric, the better mean is **bold**.

| Task Type | Task | Metric | GSPO | | RLOO | | CISPO | |
|---|---|---|---|---|---|---|---|---|
| | | | SFT-SMILES | SFT-Trace | SFT-SMILES | SFT-Trace | SFT-SMILES | SFT-Trace |
| Property Optimization | LogP | SR | $0.7204_{\pm 0.0011}$ | $\mathbf{0.8767}_{\pm 0.0012}$ | $0.8546_{\pm 0.0009}$ | $\mathbf{0.9150}_{\pm 0.0036}$ | $0.7515_{\pm 0.0010}$ | $\mathbf{0.8768}_{\pm 0.0006}$ |
| | | Sim | $0.7752_{\pm 0.0000}$ | $\mathbf{0.7844}_{\pm 0.0002}$ | $\mathbf{0.8049}_{\pm 0.0002}$ | $0.7616_{\pm 0.0033}$ | $\mathbf{0.8037}_{\pm 0.0002}$ | $0.7283_{\pm 0.0008}$ |
| | | SR×Sim | $0.5555_{\pm 0.0008}$ | $\mathbf{0.6956}_{\pm 0.0012}$ | $0.6890_{\pm 0.0007}$ | $\mathbf{0.7100}_{\pm 0.0014}$ | $0.5924_{\pm 0.0009}$ | $\mathbf{0.6630}_{\pm 0.0008}$ |
| | MR | SR | $0.6789_{\pm 0.0004}$ | $\mathbf{0.7985}_{\pm 0.0057}$ | $0.7843_{\pm 0.0019}$ | $\mathbf{0.8267}_{\pm 0.0034}$ | $0.6795_{\pm 0.0014}$ | $\mathbf{0.7897}_{\pm 0.0022}$ |
| | | Sim | $\mathbf{0.7784}_{\pm 0.0002}$ | $0.7720_{\pm 0.0045}$ | $\mathbf{0.8107}_{\pm 0.0003}$ | $0.7672_{\pm 0.0005}$ | $\mathbf{0.8035}_{\pm 0.0001}$ | $0.7292_{\pm 0.0007}$ |
| | | SR×Sim | $0.5303_{\pm 0.0003}$ | $\mathbf{0.6193}_{\pm 0.0026}$ | $\mathbf{0.6394}_{\pm 0.0017}$ | $0.6391_{\pm 0.0027}$ | $0.5477_{\pm 0.0012}$ | $\mathbf{0.5956}_{\pm 0.0015}$ |
| | QED | SR | $0.4867_{\pm 0.0005}$ | $\mathbf{0.5843}_{\pm 0.0008}$ | $0.4643_{\pm 0.0005}$ | $\mathbf{0.6305}_{\pm 0.0028}$ | $0.4697_{\pm 0.0004}$ | $\mathbf{0.6215}_{\pm 0.0008}$ |
| | | Sim | $\mathbf{0.7855}_{\pm 0.0004}$ | $0.7657_{\pm 0.0010}$ | $\mathbf{0.8087}_{\pm 0.0001}$ | $0.7609_{\pm 0.0044}$ | $\mathbf{0.7921}_{\pm 0.0004}$ | $0.7152_{\pm 0.0021}$ |
| | | SR×Sim | $0.3830_{\pm 0.0005}$ | $\mathbf{0.4556}_{\pm 0.0014}$ | $0.3741_{\pm 0.0004}$ | $\mathbf{0.4833}_{\pm 0.0014}$ | $0.3709_{\pm 0.0004}$ | $\mathbf{0.4649}_{\pm 0.0010}$ |
| Structure Optimization | Add | SR | $0.7015_{\pm 0.0012}$ | $\mathbf{0.7764}_{\pm 0.0060}$ | $0.7008_{\pm 0.0009}$ | $\mathbf{0.7029}_{\pm 0.0008}$ | $\mathbf{0.7183}_{\pm 0.0012}$ | $0.7005_{\pm 0.0009}$ |
| | | Sim | $\mathbf{0.7607}_{\pm 0.0003}$ | $0.7276_{\pm 0.0008}$ | $\mathbf{0.7639}_{\pm 0.0000}$ | $0.7562_{\pm 0.0005}$ | $\mathbf{0.7715}_{\pm 0.0004}$ | $0.7434_{\pm 0.0015}$ |
| | | SR×Sim | $0.5350_{\pm 0.0010}$ | $\mathbf{0.5882}_{\pm 0.0042}$ | $0.5355_{\pm 0.0007}$ | $\mathbf{0.5407}_{\pm 0.0006}$ | $\mathbf{0.5528}_{\pm 0.0010}$ | $0.5441_{\pm 0.0008}$ |
| | Del | SR | $0.7269_{\pm 0.0009}$ | $\mathbf{0.7552}_{\pm 0.0031}$ | $0.6602_{\pm 0.0012}$ | $\mathbf{0.8061}_{\pm 0.0012}$ | $0.7301_{\pm 0.0003}$ | $\mathbf{0.7677}_{\pm 0.0025}$ |
| | | Sim | $\mathbf{0.7351}_{\pm 0.0004}$ | $0.6743_{\pm 0.0016}$ | $\mathbf{0.7133}_{\pm 0.0002}$ | $0.6910_{\pm 0.0006}$ | $\mathbf{0.6991}_{\pm 0.0006}$ | $0.6886_{\pm 0.0003}$ |
| | | SR×Sim | $\mathbf{0.5405}_{\pm 0.0008}$ | $0.5340_{\pm 0.0022}$ | $0.4680_{\pm 0.0007}$ | $\mathbf{0.5716}_{\pm 0.0009}$ | $0.5162_{\pm 0.0002}$ | $\mathbf{0.5463}_{\pm 0.0016}$ |
| | Sub | SR | $0.7124_{\pm 0.0009}$ | $\mathbf{0.7207}_{\pm 0.0002}$ | $0.6898_{\pm 0.0005}$ | $\mathbf{0.7353}_{\pm 0.0006}$ | $0.6747_{\pm 0.0003}$ | $\mathbf{0.7172}_{\pm 0.0016}$ |
| | | Sim | $\mathbf{0.7583}_{\pm 0.0003}$ | $0.7538_{\pm 0.0009}$ | $\mathbf{0.7567}_{\pm 0.0000}$ | $0.7544_{\pm 0.0014}$ | $\mathbf{0.7670}_{\pm 0.0001}$ | $0.7516_{\pm 0.0011}$ |
| | | SR×Sim | $0.5424_{\pm 0.0007}$ | $\mathbf{0.5474}_{\pm 0.0002}$ | $0.5247_{\pm 0.0004}$ | $\mathbf{0.5603}_{\pm 0.0004}$ | $0.5166_{\pm 0.0002}$ | $\mathbf{0.5477}_{\pm 0.0012}$ |
| All Tasks | Avg | SR | $0.6711_{\pm 0.0007}$ | $\mathbf{0.7520}_{\pm 0.0023}$ | $0.6923_{\pm 0.0006}$ | $\mathbf{0.7694}_{\pm 0.0006}$ | $0.6706_{\pm 0.0002}$ | $\mathbf{0.7456}_{\pm 0.0008}$ |
| | | Sim | $\mathbf{0.7655}_{\pm 0.0002}$ | $0.7463_{\pm 0.0008}$ | $\mathbf{0.7764}_{\pm 0.0001}$ | $0.7485_{\pm 0.0006}$ | $\mathbf{0.7728}_{\pm 0.0002}$ | $0.7260_{\pm 0.0001}$ |
| | | SR×Sim | $0.5145_{\pm 0.0005}$ | $\mathbf{0.5734}_{\pm 0.0013}$ | $0.5384_{\pm 0.0004}$ | $\mathbf{0.5842}_{\pm 0.0008}$ | $0.5161_{\pm 0.0001}$ | $\mathbf{0.5603}_{\pm 0.0008}$ |

### C.2 Cross-Modal Executable Tasks

The optimizer results above show that TRACE's gains are not tied to a particular RL objective. We next ask whether the same executable-contract supervision extends beyond molecular graph editing. Codon optimization and Rhea reaction repair provide two such settings: their answers are sparse executable programs rather than molecules, as illustrated in Figure 8.

In both tasks, we compare matched answer-only and TRACE training under the same Qwen3-1.7B backbone, SFT/GRPO recipe, outcome reward, and executable answer format; TRACE differs only by placing the task-specific contract before the answer. Construction and dataset details are given in Sections A.1 and B.5. Both experiments use outcome-only GRPO and 300 held-out prompts evaluated over generation seeds 42/43/44.

**Codon optimization / TRACE**

```
<think>
OBJECTIVE: Raise human CAI to >= 0.8864 while preserving the encoded protein.
STRATEGY: Replace low-weight codons with synonymous codons that satisfy all stated hard constraints;
  keep length, start codon, terminal stop, and every amino acid fixed.
CODON EDITS:
  - index 2:  GCG -> GCC; aa=A; synonymous=yes; dlogCAI=+0.037970
  - index 3:  CAA -> CAG; aa=Q; synonymous=yes; dlogCAI=+0.032312
  - index 7:  UCG -> AGC; aa=S; synonymous=yes; dlogCAI=+0.040773
PROPERTY INTENT: CAI 0.8064 -> 0.9011; GC 0.6762 -> 0.6857.
PRESERVE: protein_hash=0ebb4897f35e1f2c; protein_length=34; translate(output)==translate(input).
CHECKS: replay_edits=target_cds; answer_sparse_execute=target_cds; all_edits_synonymous=yes;
  alphabet=ACGU; len%3=0; start=AUG; terminal_stop_unchanged=yes; no_internal_stop=yes;
  protein_equal=yes; non_noop=yes; CAI>=target=yes.
</think><answer>2:GCG>GCC | 3:CAA>CAG | 7:UCG>AGC</answer>
```

**Reaction repair / TRACE**

```
<think>
ELEMENT DELTA (left - right, from formulas):  Fe:-8; H:+8; N:-8; O:-20
CHARGE DELTA (left - right):  -4
EDIT PLAN (slot coefficient changes that zero every delta):
  - slot 0 ([Fe+3]):  coeff +6 -> -6
  - slot 2 (O): coeff +2 -> -2
  - slot 3 ([Fe+2]):  coeff +2 -> +6
  - slot 4 (O=N[O-]):  coeff +9 -> +1
  - slot 5 ([H+]):  coeff -8 -> +8
PRESERVE (slots left unchanged):  slot 1
CHECKS: after edits every element delta = 0; charge delta = 0; >=1 reactant and >=1 product.
</think><answer>0:-6 | 2:-2 | 3:6 | 4:1 | 5:8</answer>
```

Figure 8: **Cross-modal executable trace surfaces.** Codon answers are sparse synonymous edit programs, while Rhea answers are slot-coefficient repair programs. In both settings, TRACE exposes fields that a deterministic executor can replay against the answer.

### C.2.1 Codon Optimization

**Setup.** Both arms emit the same sparse `pos:before>after` edit program, with TRACE differing only by placing the executable contract before the answer. The task-native executor applies the substitutions and evaluates the resulting coding sequence. SR measures the fraction of outputs that satisfy the codon-optimization target, Sim measures similarity to the input sequence, and $\mathrm{SR} \times \mathrm{Sim}$ is the per-sample joint metric. We also report the average number of successful synonymous substitutions and the fraction of outputs that remain below the target CAI.

Table 6: **Executable codon edits.** Held-out 300 results for fixed checkpoints; cells are mean$\pm$std over generation seeds 42/43/44. Mechanism columns report successful-edit count ($\uparrow$) and CAI-below-target failure rate ($\downarrow$).

| Arm | Stage | SR | Sim | SR$\times$Sim | Edit Count $\uparrow$ | CAI-Below $\downarrow$ |
|---|---|---|---|---|---|---|
| answer | SFT | $0.6911_{\pm 0.0042}$ | $\mathbf{0.8767}_{\pm 0.0017}$ | $0.6254_{\pm 0.0036}$ | $7.36_{\pm 0.01}$ | $0.2711_{\pm 0.0057}$ |
| trace | SFT | $0.6611_{\pm 0.0063}$ | $0.8352_{\pm 0.0074}$ | $0.5970_{\pm 0.0057}$ | $7.45_{\pm 0.03}$ | $0.2567_{\pm 0.0094}$ |
| answer | GRPO-348 | $0.6022_{\pm 0.0087}$ | $0.8683_{\pm 0.0043}$ | $0.5474_{\pm 0.0081}$ | $6.84_{\pm 0.02}$ | $0.3478_{\pm 0.0068}$ |
| trace | GRPO-261 | $\mathbf{0.7189}_{\pm 0.0031}$ | $0.7891_{\pm 0.0016}$ | $\mathbf{0.6476}_{\pm 0.0028}$ | $\mathbf{7.60}_{\pm 0.02}$ | $\mathbf{0.1533}_{\pm 0.0047}$ |

**TRACE improves codon optimization by sustaining enough legal edits to reach the target.** After GRPO, TRACE reaches $0.7189$ SR and $0.6476$ $\mathrm{SR} \times \mathrm{Sim}$, compared with $0.6022$ and $0.5474$ for the answer-only arm (Table 6). This gain comes despite lower sequence similarity, indicating that TRACE improves task completion rather than simply preserving the input more conservatively.

The edit statistics help explain the difference. Answer-only GRPO makes fewer successful substitutions and increases the CAI-below-target failure rate, whereas TRACE maintains roughly the same edit count while reducing below-target failures from $0.2567$ to $0.1533$. This suggests that TRACE helps preserve enough legal synonymous edits to satisfy the CAI objective more reliably.

### C.2.2 Reaction Repair (Rhea)

**Setup.** Both arms emit the same sparse slot-repair program, which the task-native executor applies and checks for a parseable, non-degenerate, atom- and charge-balanced reaction. *Arm* distinguishes answer-only from TRACE training, while *Stage* reports the SFT and GRPO checkpoints. SR is the

overall repair success rate; SR@depth-3 and SR@depth-4 report success on reactions whose minimal valid repair requires three or four slot edits, respectively. Similarity is not used because closeness to the corrupted input is not a preservation objective.

Table 7: **Reaction repair on Rhea.** Qwen3-1.7B results on 300 held-out examples, reported as mean±std over generation seeds 42/43/44.

| Arm | Stage | SR | SR@depth-3 | SR@depth-4 |
|---|---|---|---|---|
| answer | SFT | $0.8167_{\pm 0.0000}$ | $\mathbf{0.9118}_{\pm 0.0000}$ | $0.6923_{\pm 0.0000}$ |
| trace | SFT | $\mathbf{0.8244}_{\pm 0.0016}$ | $0.8882_{\pm 0.0000}$ | $\mathbf{0.7410}_{\pm 0.0036}$ |
| answer | GRPO | $0.7889_{\pm 0.0031}$ | $\mathbf{0.8824}_{\pm 0.0000}$ | $0.6667_{\pm 0.0073}$ |
| trace | GRPO | $\mathbf{0.8311}_{\pm 0.0031}$ | $0.8706_{\pm 0.0048}$ | $\mathbf{0.7795}_{\pm 0.0096}$ |

**TRACE improves reaction repair, with the largest gain on harder edits.** After GRPO, TRACE reaches an overall SR of $0.8311$, compared with $0.7889$ for the answer-only arm (Table 7). The difference is larger on depth-4 repairs, where TRACE achieves $0.7795$ SR versus $0.6667$ for answer-only, while the two methods are much closer on depth-3 cases.

The training trajectories also differ: answer-only GRPO falls below its SFT initialization, whereas TRACE improves slightly after GRPO. On the matched $300$ prompts, this yields a paired SR gain of $+0.042$ for TRACE (McNemar $p = 3.7 \times 10^{-4}$). The advantage therefore appears most clearly when successful repair requires a longer sequence of coordinated slot edits.

**Conclusion.** Together, the Codon and Rhea experiments show that TRACE extends beyond molecular graph editing and is particularly useful when success requires coordinating multiple constrained edits. In both tasks, answer-only GRPO struggles with these harder cases and can even degrade from its SFT initialization, whereas TRACE retains or improves the required multi-edit behavior. These results indicate that the executable trace is most useful when the model must coordinate several constrained edits that answer-only training does not reliably preserve.

### C.3 Verifier Diagnostics and Utilities

The TRACE verifier asks whether the generated answer carries out the claims stated in the trace. This is related to, but different from, task success: a model may faithfully execute an ineffective edit, or succeed despite an imperfect trace. We therefore first characterize what the verifier measures and then test how its reference-answer-free signal can be used during training and test-time selection.

#### C.3.1 Per-Task $C/K/D$ Definitions

Equation (8) defines a common verifier interface through the parsed claim set $A_i$, per-claim support $\mathrm{supp}_\tau$, edit agreement $E_\tau$, and safeguard gates $g_{j,i}$. Each task instantiates these quantities with its own claim types and executable checks, while contract completeness $C_i$ follows the corresponding task-specific schema. Table 8 summarizes the six instantiations.

#### C.3.2 Why the Main Faithfulness Metric Is $D$

The verifier exposes $C/K/D$, which capture increasingly stronger aspects of the trace–answer relationship. We use $D$ as the primary say–do metric because it most directly measures whether the edit described in the trace is carried out in the generated answer.

**$C$ establishes checkability.** It verifies that the required trace structure and checkable claims are present. This is necessary for subsequent verification, but once trained models reliably emit the schema, $C$ provides little distinction among valid generations.

**$K$ checks the stated claims.** It measures whether the claims made in the trace are supported by facts recomputed from the generated answer. This helps identify unsupported or incorrect statements, but does not by itself determine whether the trace fully accounts for the change realized in the answer.

Table 8: **Task-specific instantiations of the TRACE verifier.** Following Equation (8), each task defines its claims $A_i$, support score $\mathrm{supp}_\tau$, agreement term $E_\tau$, and safeguard gates $g_{j,i}$. Contract completeness $C_i$ follows the task-specific required schema.

| Contract | Claims $A_i$ | Support $\mathrm{supp}_\tau$ | Agreement $E_\tau$ | Safeguards $g_{j,i}$ |
|---|---|---|---|---|
| **TOMG MolOpt** | Declared (SITE, BEFORE, AFTER) edit triples. | Binary support for each claim based on whether its canonical before–after fragment change appears in the edit set recomputed from the input–answer pair. | Agreement between the declared and recomputed edit sets, accounting for both unsupported claims and omitted executed edits. | Replay of the declared PRESERVE and CHECKS conditions. |
| **TOMG Structure** | Declared structural operation and target groups. | Binary support requires the operation to match the request and all claimed target groups to be correctly identified. | The recomputed group-count changes must realize the declared structural operation. | Scaffold retention, target-group preservation, size, heavy-atom-delta, and other declared preservation checks. |
| **MUMO** | Requested property directions and net group-delta claims. | Agreement between the stated and recomputed property directions and group-count deltas. | Fraction of declared group-count changes realized by the input–answer pair. | Declared preserved groups must remain unchanged; invalid or degenerate outputs fail the safeguard checks. |
| **TDC** | Property measurements, deltas, and cited structural evidence. | Weighted agreement between stated and recomputed property values, deltas, charge, size, and cited evidence. | The property change recomputed from the answer must agree with the declared target direction. | Complete task schema, a valid single-molecule answer, and the task-specific evidence constraints. |
| **Codon** | Indexed (POS, BEFORE, AFTER) codon substitutions. | Agreement on the source codon, synonymous replacement, and encoded amino-acid identity for each declared edit. | Agreement between the declared substitutions and those realized by executing the sparse answer program, including undeclared changes. | CDS validity, protein identity, locked start/stop codons, non-noop output, and required CAI/GC constraints. |
| **Rhea** | Declared slot edits together with element- and charge-delta claims. | Agreement between declared and recomputed coefficient changes, element imbalances, and charge imbalance. | Executing the answer slot program must realize the declared repair and produce the corresponding balanced reaction. | The repaired reaction must be non-degenerate, and the declared CHECKS must replay successfully. |

**$D$ captures say–do consistency.** It compares the declared edit with the change actually realized by the answer, while enforcing the relevant preservation and validity conditions. This makes $D$ the closest match to the quantity of interest: whether the model does what its trace says. Because the comparison is made against the model's own output rather than a reference target, it remains distinct from task success. We therefore report $D$ as the main faithfulness metric and use $C$ and $K$ as complementary diagnostics.

### C.3.3 SAY–DO CONSISTENCY ACROSS SCALE

Table 9 compares say–do consistency with editing performance across Qwen3 scales. Under SFT, both $\mathrm{SR}\times\mathrm{Sim}$ and $D$ generally increase with model size. After outcome-only GRPO, the pattern is less regular: performance continues to improve with scale, while MolOpt $D$ drops at 1.7B and 8B, with 4B standing out as a notably high post-GRPO point; Structure $D$ remains comparatively stable.

Overall, higher editing performance tends to coincide with stronger say–do consistency. We next examine whether this association also holds at the generation level.

### C.3.4 SAY–DO $D$ AS A FAILURE/AUDIT SIGNAL

To examine the relationship at the generation level, we analyze 30,000 Qwen3-1.7B MolOpt generations from the SFT-TRACE and SFT-TRACE+GRPO checkpoints at seed 42. MolOpt provides a clean setting for this comparison: task success depends on property improvement, whereas $D$ checks whether the declared edit is reflected in the generated answer.

**Low $D$ is strongly associated with obvious generation failures.** The lowest-$D$ group contains every invalid and no-op output and has an outcome-failure rate of 0.564, compared with 0.344 and 0.351 in the higher-$D$ groups (Table 10). At the same time, high $D$ does not imply task success:

Table 9: **Say–do consistency and editing performance across scale.** Say–do $D$ and family-average per-sample $\mathrm{SR}\times\mathrm{Sim}$ for TRACE on TOMG-Bench. $D$ is evaluated on generation seed 42; outcome values are averaged over generation seeds 42/43/44.

| | | SFT-TRACE | | SFT-TRACE+GRPO | |
|---|---|---|---|---|---|
| Backbone | Family | $D$ | SR×Sim | $D$ | SR×Sim |
| Qwen3-1.7B | MolOpt | 0.586 | 0.4333 | 0.332 | 0.6079 |
| | Structure | 0.574 | 0.4170 | 0.648 | 0.5445 |
| Qwen3-4B | MolOpt | 0.622 | 0.4742 | 0.756 | 0.6174 |
| | Structure | 0.657 | 0.4702 | 0.665 | 0.5710 |
| Qwen3-8B | MolOpt | 0.626 | 0.4825 | 0.359 | 0.6508 |
| | Structure | 0.690 | 0.4927 | 0.698 | 0.5924 |

$35.1\%$ of the highest-$D$ group still fails the property criterion, and the per-generation association with outcome is weak (Spearman $\rho = -0.21$). Thus $D$ is useful for identifying poorly executed generations, but is not itself a proxy for the task oracle.

Table 10: **Low say–do $D$ concentrates execution failures.** Qwen3-1.7B MolOpt generations (30,000; seed 42) are grouped by $D$. The empty $[0.2, 0.5)$ bin is omitted.

| say-do $D$ bin | $n$ | outcome-fail | invalid | no-op | mean Sim |
|---|---|---|---|---|---|
| $[0, 0.2)$ | 14767 | 0.564 | 0.126 | 0.169 | 0.636 |
| $[0.5, 0.8)$ | 323 | 0.344 | 0.000 | 0.000 | 0.611 |
| $[0.8, 1]$ | 14910 | 0.351 | 0.000 | 0.000 | 0.734 |

**Conditioning on feasibility explains this pattern.** Let $E$ denote the event that the answer is valid and changed, $F$ whether it faithfully executes the declared edit, and $G$ whether that edit moves the requested property in the correct direction. Schematically,

$$D \simeq E\,F, \qquad r_{\text{out}} \simeq E\,G. \tag{20}$$

Both scores therefore fall when the answer is invalid or unchanged. Once feasibility is fixed, however, they measure different things: $D$ asks whether the stated edit was executed, while the task oracle asks whether that edit was useful.

This separation becomes clear among the 25,629 valid, changed outputs. Here, $D$ ranks a successful edit above a wrong-direction edit with pooled AUC $0.515$ (Table 11), and the fraction with $D \geq 0.8$ is similar for successful and wrong-direction edits ($59.3\%$ vs. $56.3\%$). The unconditional association between $D$ and task outcome is therefore driven largely by their shared sensitivity to infeasible outputs. Among feasible edits, $D$ primarily reflects execution consistency rather than whether the edit improves the requested property.

Table 11: **Say–do $D$ after conditioning on feasibility.** Qwen3-1.7B MolOpt outputs are restricted to valid, changed molecules, so the remaining failures are wrong-direction edits. AUC measures whether $D$ ranks a successful edit above a wrong-direction edit; $0.5$ is chance.

| Arm | $n$ | $\mathbb{E}[D \mid \text{success}]$ | $\mathbb{E}[D \mid \text{wrong dir.}]$ | AUC |
|---|---|---|---|---|
| SFT-TRACE | 11,618 | 0.566 | 0.578 | 0.494 |
| SFT-TRACE+GRPO | 14,011 | 0.623 | 0.560 | 0.532 |
| Pooled | 25,629 | 0.601 | 0.571 | 0.515 |

### C.3.5 Process-Reward Faithfulness Ablation

The previous analysis shows that, once feasibility is fixed, $D$ primarily tracks whether the declared edit is executed rather than whether that edit is useful. We next test this distinction interventionally

by optimizing the verifier signal during RL. Starting from the same TRACE-SFT initialization and outcome reward, we compare outcome-only GRPO with additive and token-split process rewards.

**Additive process reward.** The additive variant defines

$$s_P(\hat{\mathbf{t}}, \hat{\mathbf{s}}) = 0.2C + 0.3K + 0.5D$$

and adds it to the answer-level reward:

$$r_{\text{add}}(q, \hat{\mathbf{t}}, \hat{\mathbf{s}}) = r_{\text{out}}(q, \hat{\mathbf{s}}) + \lambda\, s_P(\hat{\mathbf{t}}, \hat{\mathbf{s}}), \qquad \lambda = 0.3. \tag{21}$$

The resulting group-normalized advantage is applied to the full rollout; setting $\lambda = 0$ recovers the main outcome-only objective.

**Token-split process reward.** The token-split variant keeps the group-normalized outcome advantage $\widehat{A}_g^{\text{out}}$ on all tokens and adds a group-centered process advantage only to the trace:

$$\widehat{A}_{g,k} = \begin{cases} \widehat{A}_g^{\text{out}}, & o_{g,k} \in \hat{\mathbf{s}}_g \quad \text{(answer)}, \\ \widehat{A}_g^{\text{out}} + \epsilon \widehat{A}_g^{P}, & o_{g,k} \in \hat{\mathbf{t}}_g \quad \text{(trace)}, \end{cases} \tag{22}$$

with $\epsilon = 0.1$. MolOpt uses the binary process signal $\mathbb{1}[CKD \geq 0.8]$ before group centering, whereas Structure uses $CK(0.6D_{\text{op}} + 0.4D_{\text{preserve}})$ before group normalization. The two families therefore share the same token-routing scheme but use task-specific verifier signals.

Table 12: **Symbolic process-reward ablation on TOMG-Bench.** Qwen3-1.7B arms share the same SFT initialization and outcome reward, differing only in the verifier term. Rows pair per-sample SR×Sim with $D_{\text{faith}}$; cells are mean±std over generation seeds 42/43/44.

| Task | Metric | B+GRPO | +PRM(add) | +PRM(split) |
|---|---|---|---|---|
| MolOpt | SR×Sim | $\mathbf{0.6079}_{\pm 0.0007}$ | $0.5986_{\pm 0.0007}$ | $0.6071_{\pm 0.0003}$ |
| | $D_{\text{faith}}$ | $0.3374_{\pm 0.0066}$ | $\mathbf{0.8277}_{\pm 0.0013}$ | $0.4168_{\pm 0.0089}$ |
| Struct | SR×Sim | $0.5445_{\pm 0.0005}$ | $0.5342_{\pm 0.0006}$ | $\mathbf{0.5475}_{\pm 0.0002}$ |
| | $D_{\text{faith}}$ | $0.6477_{\pm 0.0009}$ | $\mathbf{0.6942}_{\pm 0.0014}$ | $0.6821_{\pm 0.0008}$ |

The results show a clear separation between faithfulness and task performance. The additive process reward raises $D_{\text{faith}}$ from 0.3374 to 0.8277 on MolOpt and from 0.6477 to 0.6942 on Structure, while SR × Sim is slightly lower in both families (Table 12). The token-split variant produces a smaller increase in $D_{\text{faith}}$ and leaves task performance close to the outcome-only baseline. Thus the verifier signal can be optimized directly, but neither process-reward variant yields a consistent improvement in task outcome.

The training trajectories show the same separation. Outcome-only GRPO improves editing while $D$ declines, whereas the additive process reward drives $D$ upward without a matching gain in task performance; the token-split variant produces a milder change (Figure 9). Together with the generation-level analysis above, these results show that execution consistency can be improved directly without a corresponding improvement in edit quality.

### C.3.6 Verifier-Guided Selection

Since directly optimizing $D$ does not reliably improve task outcome, we next test whether it can instead help select among candidate generations. We focus on the Structure subtasks, where the verifier directly checks whether the requested graph operation is executed under the stated preservation constraints. Using the Qwen3-1.7B SFT-TRACE+GRPO checkpoint, we sample up to $N = 32$ candidates per prompt at temperature 0.9 and select the one with the highest $D$. We compare this rule with random selection, majority voting over canonical answer SMILES, and an oracle that maximizes SR × Sim. Bootstrap intervals measure prompt-level variation within the fixed candidate pools.

**Verifier-based selection improves steadily as the candidate pool grows.** At $N = 32$, SR × Sim increases from 0.539 to 0.596, nearly matching the oracle value of 0.597 (Figure 4c,d). This recovers 97% of the oracle Best-of-$N$ improvement, while validity rises from 0.932 to 0.988. These results show that $D$ provides a strong reference-answer-free signal for ranking candidate outputs.

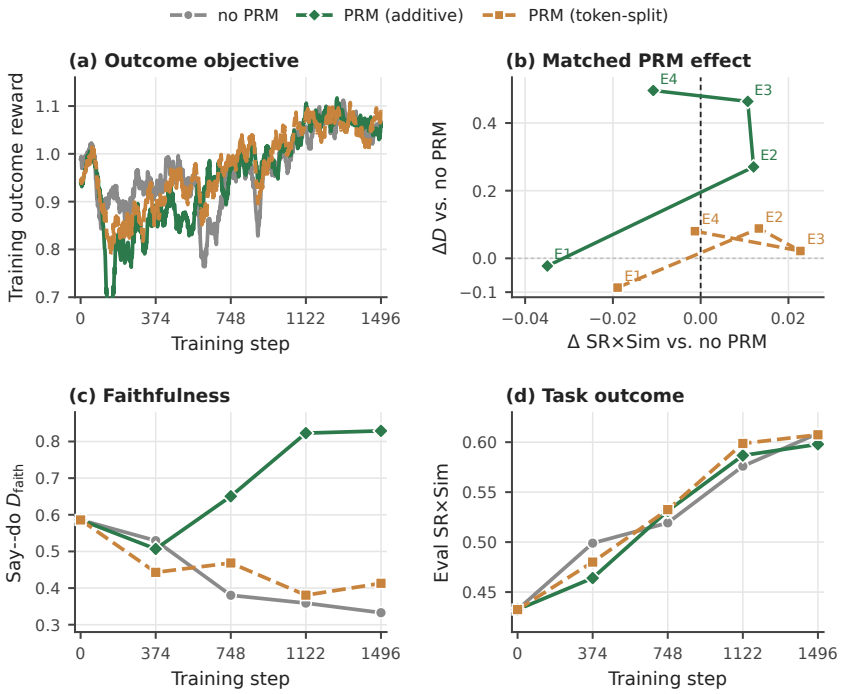



Figure 9: **Training dynamics with verifier-based process rewards on TOMG MolOpt.** Matched runs differ only in the process objective. The additive reward strongly raises $D$ with little outcome gain, while the token-split variant produces a smaller faithfulness change. E1–E4 denote checkpoints after epochs 1–4.

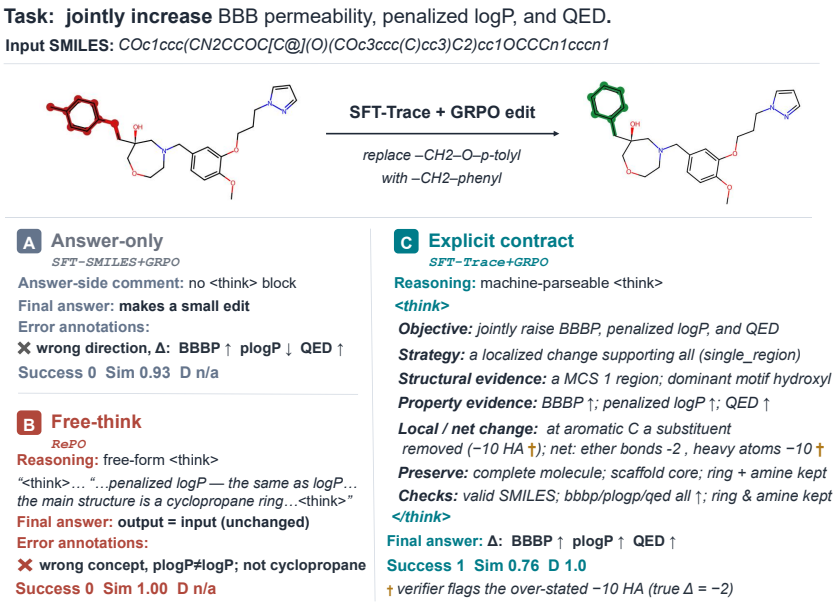



Figure 10: **Successful outcome with a faulty edit claim.** On a 35-heavy-atom multi-property case, SFT-SMILES+GRPO edits in the wrong direction, REPO returns the input, and SFT-TRACE+GRPO succeeds while the verifier flags an overstated heavy-atom delta.

Table 13: **Prompt-only baseline on TOMG with Qwen3-1.7B.** Prompt-only TRACE is compared with trained TRACE using the same evaluation protocol. Outcome metrics average seeds 42/43/44; audit statistics use seed 42. Here, n/a indicates that no contract is produced.

| | base Qwen3-1.7B (prompt-only) | | | trained TRACE | |
|---|---|---|---|---|---|
| Metric | answer-only | contract 0-shot | contract 2-shot | SFT-TRACE | +GRPO |
| SR | 0.2494 | 0.2610 | 0.2642 | 0.6320 | 0.7564 |
| Sim | 0.1311 | 0.2189 | 0.3615 | 0.6399 | 0.7528 |
| SR × Sim | 0.0339 | 0.0511 | 0.0759 | **0.4251** | **0.5762** |
| $C$ | n/a | 0.000 | 0.000 | 1.000 | 1.000 |
| $K$ | n/a | 0.000 | 0.000 | 0.797 | 0.668 |
| $D$ | n/a | 0.000 | 0.000 | 0.580 | 0.490 |
| valid <answer> (%) | 0.0 | 0.0 | 28.5 | 82.3 | 95.5 |

### C.3.7 Successful Outcome with an Auditable Claim Error

Complementing the TOMG example in the main text, Figure 10 shows a successful MUMO edit with an incorrect trace claim. The model satisfies the task objective but reports a heavy-atom change of $-10$ instead of the recomputed $-2$, which the verifier detects through $K$. The example illustrates that successful editing and accurate edit claims can be audited separately.

## C.4 Trace Surface Controls

We next examine whether TRACE's benefits depend on how the contract is exposed, represented, and constructed. Three controls address complementary questions:

- **Learning requirement.** Can the contract provide comparable benefits when supplied only through prompting, or does it need to be learned through training?
- **Supervision surface.** When training is used, how does the typed TRACE contract compare with the more conventional approach of using a stronger LLM to generate free-form reasoning supervision (Mukherjee et al., 2023), including explicit but untyped operation cues?
- **Construction backend.** If deterministic domain tools are unavailable, can a frontier LLM still instantiate the TRACE contract directly from input–target examples?

We then use additional controls to examine native verification, trace length, and reproduction fidelity against RePO.

### C.4.1 Prompting the Contract Without Training

Our first control asks whether TRACE must be learned through training, or whether exposing the same contract at inference time is already sufficient.

**Setup.** We evaluate the untrained Qwen3-1.7B base model under three conditions: a bare answer-only prompt, the TRACE system prompt used for SFT-TRACE, and the same prompt with two held-out, subtask-matched worked examples. The two-shot setting provides a stronger in-context learning baseline. All conditions use the same user prompts, evaluator, and generation seeds 42/43/44.

**Prompting recovers only a small fraction of TRACE's training gain.** The two-shot condition improves $\mathrm{SR}\times\mathrm{Sim}$ by $124\%$ over the bare base model, yet reaches only $17.9\%$ of SFT-TRACE performance. Most of this improvement comes from higher similarity, while SR changes little, suggesting that prompting mainly helps preserve the input without reliably carrying out the requested edit. We use SFT-TRACE as the direct training comparison, with SFT-TRACE+GRPO included only as a post-training reference.

**Prompting does not recover the native audit interface.** Zero-shot generations contain no well-formed <answer> blocks. Two-shot prompting raises answer-block compliance to $28.5\%$, but none of the outputs parse into valid typed claims, leaving native $C/K/D$ at zero.

**Conclusion.** Exposing the TRACE contract at inference time is helpful, but it recovers only a small fraction of the trained performance and does not reproduce the native audit interface. The benefit therefore cannot be explained by prompt-level exposure to the contract alone; learning the contract through training remains important.

### C.4.2 Closed-Source Narrative Controls

Our second control asks whether TRACE's typed supervision surface is necessary, or whether a more conventional frontier-LLM-generated rationale (Mukherjee et al., 2023) can provide the same benefit. We further add explicit operation cues to test whether any difference is simply due to TRACE making the declared edit easier to identify.

**Setup.** We compare three TOMG supervision surfaces while keeping prompts, answer molecules, SFT/GRPO recipes, outcome rewards, and generation protocols fixed. *Narrative* uses an Opus 4.8-authored free-form rationale generated from the same row-level facts. *Narrative+ops* retains the rationale but adds deterministic, untyped operation cues such as `add O` or `remove F`, testing whether explicit action information alone accounts for the gain. *TRACE* renders the same facts as typed Goal–Local Edit–Safeguard fields with site–before–after bindings and executable checks. Figure 11 shows the three surfaces for the same TOMG example.

**Shared TOMG row:** `sub_component_0499`

```
Prompt: Modify the molecule provided below (as SMILES) by substituting a halo with a thiol.
Input: CCOc1ccc(CC(=O)N2CCC(NC(=O)C(C)Cl)CC2)cc1
Answer: CCOc1ccc(CC(=O)N2CCC(NC(=O)C(C)S)CC2)cc1
```

**Natural-language narrative**

```
<think>
Step 1: Substitute the chlorine on the carbon
(atom 17) with a thiol group.
Reason: replaces the chlorine with the thiol at
the same edited site.
Summary: the edit removes one halo and
installs one thiol at the edited molecular site,
completing the requested group substitution.
</think>
```

**Natural-language narrative + atom ops**

```
<think>
Step 1: Substitute the chlorine on the carbon
(atom 17) with a thiol group.
Reason: replaces the chlorine with the thiol at
the same edited site.
Ops: remove atoms 19; add S (n19) onto atom 17
Summary: the edit removes one halo and
installs one thiol at the edited molecular site,
completing the requested group substitution.
</think>
```

**Explicit contract / TRACE**

```
<think>
Objective: Replace one halogen with thiol on the
full parent molecule.
Strategy: Swap the local motif at one anchor
without deleting unrelated scaffold.
Local edits:
  Site: aliphatic C
  Before: CCl
  After: CS
  Check: halogen is replaced by thiol at the same
parent molecule site
Chemical intent: The substitution changes halogen
to thiol at one local site while preserving
scaffold context.
Preserve:
- Output the complete edited molecule, not a
detached group or local fragment.
- Keep unchanged aromatic scaffold, unchanged ring
system, benzene ring, amide unchanged.
Checks:
- valid full-molecule SMILES
- all local edits applied
- parent scaffold retained
- no unrelated scaffold deletion
- halogen decreases by 1 and thiol increases by 1
</think>
```

Figure 11: **Same TOMG sample, three raw trace surfaces.** The answer is fixed; only the trace surface changes. TRACE expresses the edit through typed Goal/Local-Edit/Safeguard roles that can be parsed and replayed.

**Typed contracts remain strongest after GRPO.** The three surfaces perform similarly after SFT, but separate after GRPO (Table 14). TRACE achieves the highest average $\mathrm{SR} \times \mathrm{Sim}$ at $0.5762$, compared with $0.5686$ for narrative+ops and $0.5629$ for narrative. The narrative controls remain competitive and lead on some individual subtasks, showing that frontier-LLM rationales can provide useful supervision. Adding explicit operation cues narrows the gap but does not close it.

**Conclusion.** Frontier-LLM rationales provide useful intermediate supervision, but they do not match TRACE consistently across tasks. Adding explicit operation cues narrows the gap, indicating that action information matters, yet the remaining difference suggests that TRACE benefits from how those edits are organized and grounded in the typed contract rather than from exposing the declared operation alone.

Table 14: **Trace formulation by TOMG task family.** Natural-language narratives, operation-insert narratives, and the explicit contract are compared on Qwen3-1.7B. Cells are per-sample SR×Sim mean±std over generation seeds 42/43/44; bold/underline are within each stage block.

| Task Type | Task | Metric | SFT only | | | +GRPO | | |
|---|---|---|---|---|---|---|---|---|
| | | | Narrative | Narrative +ops | Contract | Narrative | Narrative +ops | Contract |
| Property Optimization | LogP | SR | $0.6337_{\pm 0.0288}$ | $\underline{0.6458}_{\pm 0.0229}$ | $\mathbf{0.7434}_{\pm 0.0053}$ | $\underline{0.8685}_{\pm 0.0057}$ | $0.8647_{\pm 0.0025}$ | $\mathbf{0.9425}_{\pm 0.0019}$ |
| | | Sim | $\underline{0.7027}_{\pm 0.0057}$ | $\mathbf{0.7028}_{\pm 0.0053}$ | $0.6392_{\pm 0.0048}$ | $\mathbf{0.7905}_{\pm 0.0003}$ | $0.7496_{\pm 0.0010}$ | $\underline{0.7737}_{\pm 0.0019}$ |
| | | SR×Sim | $0.4468_{\pm 0.0213}$ | $\underline{0.4535}_{\pm 0.0199}$ | $\mathbf{0.4803}_{\pm 0.0031}$ | $\underline{0.6872}_{\pm 0.0048}$ | $0.6719_{\pm 0.0019}$ | $\mathbf{0.7374}_{\pm 0.0007}$ |
| | MR | SR | $\underline{0.7119}_{\pm 0.0045}$ | $0.6839_{\pm 0.0104}$ | $\mathbf{0.7321}_{\pm 0.0027}$ | $0.7826_{\pm 0.0017}$ | $\mathbf{0.8511}_{\pm 0.0013}$ | $\underline{0.7992}_{\pm 0.0020}$ |
| | | Sim | $\underline{0.6768}_{\pm 0.0046}$ | $\mathbf{0.6862}_{\pm 0.0030}$ | $0.6342_{\pm 0.0034}$ | $\mathbf{0.7719}_{\pm 0.0004}$ | $0.7338_{\pm 0.0039}$ | $\underline{0.7638}_{\pm 0.0029}$ |
| | | SR×Sim | $\mathbf{0.4740}_{\pm 0.0018}$ | $0.4614_{\pm 0.0044}$ | $\underline{0.4669}_{\pm 0.0040}$ | $0.5971_{\pm 0.0014}$ | $\mathbf{0.6197}_{\pm 0.0037}$ | $\underline{0.6183}_{\pm 0.0002}$ |
| | QED | SR | $\mathbf{0.5625}_{\pm 0.0098}$ | $0.5361_{\pm 0.0143}$ | $\underline{0.5387}_{\pm 0.0033}$ | $\mathbf{0.6373}_{\pm 0.0031}$ | $0.5885_{\pm 0.0047}$ | $\underline{0.6003}_{\pm 0.0020}$ |
| | | Sim | $\underline{0.6719}_{\pm 0.0014}$ | $\mathbf{0.6797}_{\pm 0.0027}$ | $0.6453_{\pm 0.0032}$ | $\underline{0.7674}_{\pm 0.0004}$ | $0.7528_{\pm 0.0041}$ | $\mathbf{0.7713}_{\pm 0.0005}$ |
| | | SR×Sim | $\mathbf{0.3833}_{\pm 0.0061}$ | $\underline{0.3675}_{\pm 0.0120}$ | $0.3528_{\pm 0.0024}$ | $\mathbf{0.4932}_{\pm 0.0022}$ | $0.4515_{\pm 0.0034}$ | $\underline{0.4681}_{\pm 0.0019}$ |
| Structure Optimization | Add | SR | $\underline{0.6301}_{\pm 0.0019}$ | $0.6048_{\pm 0.0047}$ | $\mathbf{0.6348}_{\pm 0.0018}$ | $0.7053_{\pm 0.0014}$ | $\mathbf{0.7419}_{\pm 0.0013}$ | $\underline{0.7085}_{\pm 0.0011}$ |
| | | Sim | $\mathbf{0.6734}_{\pm 0.0014}$ | $\underline{0.6588}_{\pm 0.0005}$ | $0.6462_{\pm 0.0016}$ | $\mathbf{0.7678}_{\pm 0.0004}$ | $\underline{0.7673}_{\pm 0.0011}$ | $0.7603_{\pm 0.0021}$ |
| | | SR×Sim | $\underline{0.4408}_{\pm 0.0013}$ | $0.4243_{\pm 0.0029}$ | $\mathbf{0.4449}_{\pm 0.0012}$ | $\underline{0.5462}_{\pm 0.0012}$ | $\mathbf{0.5743}_{\pm 0.0009}$ | $0.5427_{\pm 0.0008}$ |
| | Del | SR | $\mathbf{0.5689}_{\pm 0.0070}$ | $0.5627_{\pm 0.0029}$ | $\underline{0.5657}_{\pm 0.0008}$ | $\mathbf{0.7673}_{\pm 0.0089}$ | $0.7519_{\pm 0.0046}$ | $\underline{0.7558}_{\pm 0.0007}$ |
| | | Sim | $\underline{0.6123}_{\pm 0.0063}$ | $\mathbf{0.6224}_{\pm 0.0062}$ | $0.5788_{\pm 0.0023}$ | $0.6885_{\pm 0.0028}$ | $\mathbf{0.7087}_{\pm 0.0016}$ | $\underline{0.6947}_{\pm 0.0003}$ |
| | | SR×Sim | $\mathbf{0.3713}_{\pm 0.0041}$ | $\underline{0.3709}_{\pm 0.0021}$ | $0.3674_{\pm 0.0005}$ | $\underline{0.5355}_{\pm 0.0060}$ | $0.5329_{\pm 0.0028}$ | $\mathbf{0.5368}_{\pm 0.0005}$ |
| | Sub | SR | $\underline{0.6217}_{\pm 0.0006}$ | $\mathbf{0.6743}_{\pm 0.0014}$ | $0.5775_{\pm 0.0018}$ | $0.6785_{\pm 0.0004}$ | $\mathbf{0.7335}_{\pm 0.0002}$ | $\underline{0.7319}_{\pm 0.0004}$ |
| | | Sim | $\underline{0.7329}_{\pm 0.0004}$ | $\mathbf{0.7483}_{\pm 0.0010}$ | $0.6958_{\pm 0.0003}$ | $\underline{0.7611}_{\pm 0.0002}$ | $\mathbf{0.7682}_{\pm 0.0006}$ | $0.7528_{\pm 0.0004}$ |
| | | SR×Sim | $\underline{0.4717}_{\pm 0.0003}$ | $\mathbf{0.5123}_{\pm 0.0013}$ | $0.4386_{\pm 0.0012}$ | $0.5181_{\pm 0.0003}$ | $\mathbf{0.5611}_{\pm 0.0002}$ | $\underline{0.5540}_{\pm 0.0003}$ |
| All Tasks | Avg | SR | $\underline{0.6215}_{\pm 0.0040}$ | $0.6179_{\pm 0.0025}$ | $\mathbf{0.6320}_{\pm 0.0011}$ | $0.7399_{\pm 0.0023}$ | $\underline{0.7553}_{\pm 0.0013}$ | $\mathbf{0.7564}_{\pm 0.0007}$ |
| | | Sim | $\underline{0.6783}_{\pm 0.0014}$ | $\mathbf{0.6830}_{\pm 0.0011}$ | $0.6399_{\pm 0.0021}$ | $\mathbf{0.7579}_{\pm 0.0004}$ | $0.7467_{\pm 0.0012}$ | $\underline{0.7528}_{\pm 0.0009}$ |
| | | SR×Sim | $\underline{0.4313}_{\pm 0.0024}$ | $\mathbf{0.4317}_{\pm 0.0019}$ | $0.4251_{\pm 0.0005}$ | $0.5629_{\pm 0.0017}$ | $\underline{0.5686}_{\pm 0.0016}$ | $\mathbf{0.5762}_{\pm 0.0004}$ |

### C.4.3 Constructing TRACE Without Domain Tools

Our final control changes the construction backend rather than the supervision surface. TRACE is most straightforward to instantiate when task-specific tools can extract and verify the relevant edit facts; we ask whether the same contract remains useful when those tools are unavailable during data construction.

**Setup.** Our main TOMG corpus uses deterministic chemistry tools to extract executable facts and render TRACE contracts. To isolate the role of construction-time domain tools, the no-tool control replaces this pipeline with Opus 4.8 (Anthropic, 2026), which instantiates the same Goal–Local Edit–Safeguard schema directly from the instruction, task metadata, input SMILES, and target SMILES. It produces schema-complete traces for all 2,992 rows without access to tool-derived graph differences, MCS, local fragments, atom operations, property values, or preservation facts. All downstream SFT, GRPO, reward, and evaluation settings remain unchanged.

Table 15: **Three-way construction control on TOMG-Bench (Qwen3-1.7B).** We compare SMILES-only supervision, no-tool TRACE generated by Opus 4.8, and tool-built TRACE. Entries report per-sample SR×Sim, mean±std over generation seeds 42/43/44. After SFT+GRPO, performance follows SMILES-only < no-tool TRACE < tool-built TRACE.

| Task | SFT only | | | SFT+GRPO | | |
|---|---|---|---|---|---|---|
| | SMILES-only | No-tool TRACE | Tool-built TRACE | SMILES-only | No-tool TRACE | Tool-built TRACE |
| LogP | $0.3966_{\pm 0.0059}$ | $0.387_{\pm 0.004}$ | $0.4803_{\pm 0.0031}$ | $0.6504_{\pm 0.0006}$ | $0.573_{\pm 0.002}$ | $0.7374_{\pm 0.0007}$ |
| MR | $0.4249_{\pm 0.0362}$ | $0.454_{\pm 0.005}$ | $0.4669_{\pm 0.0040}$ | $0.5347_{\pm 0.0005}$ | $0.575_{\pm 0.003}$ | $0.6183_{\pm 0.0002}$ |
| QED | $0.3098_{\pm 0.0059}$ | $0.326_{\pm 0.006}$ | $0.3528_{\pm 0.0024}$ | $0.3782_{\pm 0.0008}$ | $0.457_{\pm 0.002}$ | $0.4681_{\pm 0.0019}$ |
| Add | $0.3979_{\pm 0.0022}$ | $0.332_{\pm 0.004}$ | $0.4449_{\pm 0.0012}$ | $0.5391_{\pm 0.0005}$ | $0.544_{\pm 0.003}$ | $0.5427_{\pm 0.0008}$ |
| Del | $0.3802_{\pm 0.0111}$ | $0.363_{\pm 0.005}$ | $0.3674_{\pm 0.0005}$ | $0.4736_{\pm 0.0004}$ | $0.531_{\pm 0.002}$ | $0.5368_{\pm 0.0005}$ |
| Sub | $0.4714_{\pm 0.0030}$ | $0.483_{\pm 0.003}$ | $0.4386_{\pm 0.0012}$ | $0.5232_{\pm 0.0003}$ | $0.514_{\pm 0.002}$ | $0.5540_{\pm 0.0003}$ |
| Avg | 0.3968 | 0.391 | 0.4251 | 0.5165 | $\underline{0.532}$ | **0.5762** |

**Outcome and checkability.** At SFT, no-tool TRACE performs similarly to the SMILES-only baseline. After GRPO, it reaches 0.532 SR×Sim, improving over SMILES-only training (0.5165) while remaining below tool-built TRACE (0.5762) (Table 15). The remaining gap is largest on LogP, whereas several other subtasks are much closer.

Table 16: **Checkability of no-tool and tool-built TRACE.** Seed-42 SFT+GRPO generations (5,000 per subtask), audited against the model's own answer. $C/K/D$ denote contract completeness, claim correctness, and say–do consistency.

| Task family | Constructor | $C$ | $K$ | $D$ |
|---|---|---|---|---|
| MolOpt | No-tool Opus 4.8 | 1.000 | 0.148 | 0.142 |
| | Deterministic tools | 1.000 | 0.337 | 0.332 |
| Structure | No-tool Opus 4.8 | 1.000 | 0.000 | 0.612 |
| | Deterministic tools | 1.000 | 0.998 | 0.648 |
| TOMG Avg. | No-tool Opus 4.8 | 1.000 | 0.074 | 0.377 |
| | Deterministic tools | 1.000 | 0.668 | 0.490 |

Both constructors produce complete schemas ($C = 1.000$), but the no-tool traces align less closely with the deterministic checker (Table 16). Averaged across TOMG, no-tool TRACE obtains $K/D = 0.074/0.377$, compared with $0.668/0.490$ for tool-built TRACE. The gap is most pronounced in MolOpt, while Structure remains much closer in execution consistency ($D = 0.612$ vs. $0.648$). Part of the mismatch reflects representation: Opus 4.8 often produces larger human-readable fragments, whereas the checker expects minimal canonical fragments or exact operation keywords.

**Trade-off and scope.** The no-tool route avoids engineering a task-specific constructor, but requires frontier-model inference for every training row and produces less checker-aligned supervision. Deterministic extraction requires more upfront domain engineering, but yields reproducible labels once the pipeline is built. Note that "no-tool" applies only to training-label construction: the checker, task oracle, GRPO reward, and evaluation remain unchanged and still use the task-specific tools.

**Conclusion.** TRACE remains viable without construction-time domain tools, but deterministic construction provides stronger and more checker-aligned supervision when such tools are available.

### C.4.4 Outcome-Competitive Free-Form Traces Lack Native Verification

The narrative controls show that free-form reasoning can remain competitive on task outcome. We next ask whether this supervision also provides the native verification interface available to TRACE.

**Setup.** We pool all six TOMG subtasks at seed 42, and audit TRACE, SMILES-only, RePO, narrative, and narrative+ops. The native verifier requires typed Local edits, an answer-derived edit set, and replayable Preserve and Checks conditions (Equation (8)). TRACE exposes these fields directly, so native say–do $D$ is well defined. RePO, narrative, and narrative+ops do not expose this contract, making native $D$ undefined rather than low.

To characterize how much information can nevertheless be recovered from these free-form traces, we additionally apply a permissive post-hoc parser to coarse count or group claims:

$$\tilde{C} = \Pr[\text{claim extracted}],$$
$$\tilde{K} = \Pr[\text{claim correct} \mid \text{extracted}],$$
$$\tilde{U} = \Pr[\text{extracted and correct}] = \tilde{C}\tilde{K}.$$

This proxy measures recoverable factual content, but unlike $D$ does not test edit-set recall, site binding, or execution of the declared Preserve/Checks conditions.

**Free-form traces contain recoverable claims but no native audit interface.** The permissive parser extracts a coarse claim from $62.2\%/70.8\%/32.7\%$ of narrative, narrative+ops, and RePO outputs, respectively, yielding all-row $\tilde{U} = 0.429/0.443/0.139$ (Table 17). Thus free-form traces often contain factual information that can be recovered after generation, but this requires a separate parser and remains weaker than native contract verification.

Table 17: **Native $C/K/D$ and post-hoc retrofit verifiability.** Qwen3-1.7B, all six TOMG-Bench subtasks pooled, seed 42, $n$=30,000 per arm. Tilded metrics use the permissive parser.

| Trace type | $C$ | $K$ | $D$ | $\tilde{C}$ | $\tilde{K}$ | $\tilde{U}$ |
|---|---|---|---|---|---|---|
| **TRACE** | 1.000 | 0.668 | 0.490 | – | – | – |
| narrative | 0.00 | N/A | N/A | 0.622 | 0.690 | 0.429 |
| narrative+ops | 0.00 | N/A | N/A | 0.708 | 0.626 | 0.443 |
| **RePO** | 0.00 | N/A | N/A | 0.327 | 0.424 | 0.139 |
| SMILES | 0.00 | N/A | N/A | 0.000 | – | 0.000 |

**Conclusion.** Competitive task performance therefore does not imply native checkability. TRACE exposes its claims in a structured form that can be verified directly, whereas free-form traces require post-hoc extraction before even a weaker faithfulness proxy can be computed.

### C.4.5 Audit Signal Tracks Contract Structure, Not Trace Length

Trace length does not account for the audit differences in Figure 12. RePO is the longest surface, with a median of 824 think tokens; narrative and narrative+ops are much shorter at 63 and 95, while TRACE lies in between at 155. Neither audit signal nor task outcome follows this ordering.

Panel (b) reports native $D$ for TRACE, retrofit $\tilde{U}$ for non-contract reasoning, and zero for SMILES-only, which contains no trace. These quantities are not directly comparable: $\tilde{U}$ captures only post-hoc recovery of coarse claims, whereas $D$ measures native replayable say–do consistency. The figure therefore rules out trace length as a simple explanation for the observed audit differences.

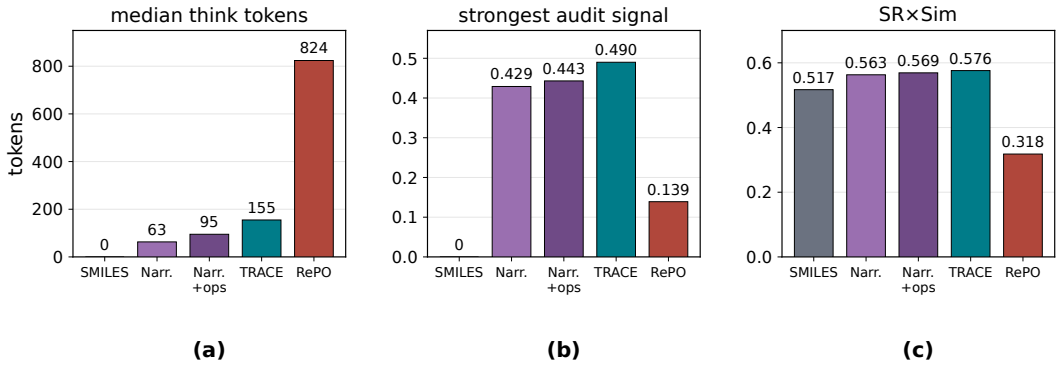



Figure 12: **Audit signal does not track trace length.** Median trace length, available audit signal, and $\mathrm{SR} \times \mathrm{Sim}$ across five supervision surfaces. Native $D$ is reported for TRACE and retrofit $\tilde{U}$ for non-contract traces; the two audit metrics are not equivalent.

### C.4.6 Reproduction Fidelity Against RePO

We finally verify that our RePO implementation provides a reasonable reference for the main comparisons. RePO Table 1 (Li et al., 2026b) reports $\mathrm{SR} \times \mathrm{Sim}$ for RePO and an SFT-initialized SFT+GRPO baseline on TOMG.

**Setup.** Our RePO runs follow the public implementation-level reward and training recipe, while matching the surrounding trainer and evaluation settings to our pipeline where applicable.

**Our RePO reproduction closely matches or exceeds the published results.** Table 18 compares the published RePO Table 1 values with our Qwen3-1.7B reproduction. Despite using the smaller backbone, our RePO results are higher on MR, QED, ADD, and DEL, and differ by at most 0.005 on LOGP and SUB. This supports our reproduction as a reasonable reference point for the main experiments. The published SFT+GRPO results are included only for context because our answer-only baseline uses the improved task-specific outcome reward adopted in this work; the main TRACE comparison remains internally matched under the same backbone, reward, and training setup.

Table 18: **Reproduction fidelity against RePO Table 1 on TOMG.** Published Qwen-2.5-3B results are compared with our Qwen3-1.7B reproduction. For SFT+GRPO, our column uses the task-specific outcome reward from the main pipeline.

| | RePO | | SFT+GRPO | |
|---|---|---|---|---|
| SR×Sim | Tab. 1 (3B) | Ours (1.7B) | Tab. 1 (3B) | Ours (1.7B) |
| LogP | 0.297 | 0.295 | 0.183 | 0.650 |
| MR | 0.294 | 0.356 | 0.225 | 0.535 |
| QED | 0.236 | 0.303 | 0.192 | 0.378 |
| Add | 0.239 | 0.349 | 0.156 | 0.539 |
| Del | 0.140 | 0.267 | 0.176 | 0.474 |
| Sub | 0.344 | 0.340 | 0.299 | 0.523 |

Table 19: **Wall-clock training cost with Qwen3-1.7B.** All runs use a single 4×A100 node. Times are reported in `hh:mm:ss`. Total time sums SFT and the two task-family GRPO runs; the final column reports average SR×Sim.

| Arm | SFT | GRPO MolOpt | GRPO Structure | Total | Avg SR×Sim |
|---|---|---|---|---|---|
| RePO | — | 12:32:05 | 12:11:02 | 24:43:07 | 0.3184 |
| SFT-SMILES+GRPO | 00:25:53 | 4:20:08 | 3:48:33 | 8:34:34 | 0.5165 |
| SFT-TRACE+GRPO | 00:28:49 | 5:47:55 | 5:56:22 | 12:13:06 | 0.5762 |

**RePO has a higher wall-clock training cost.** On the same 4×A100 setup, our RePO reproduction takes 24:43:07 in total, compared with 12:13:06 for SFT-TRACE+GRPO and 8:34:34 for SFT-SMILES+GRPO (Table 19). This corresponds to about 2.0× the wall-clock time of TRACE and 2.9× that of the answer-only pipeline. The higher cost is largely due to RePO's free-form generation, which can produce substantially longer intermediate reasoning before the final answer. TRACE also requires more training time than the answer-only arm, while achieving the highest average SR×Sim among the three.

### C.5 Where the Gains Come From: Local Edit in the TRACE Contract

We next ask why TRACE improves constrained editing. TRACE places a structured Goal–Local Edit–Safeguard trace before the final answer, but the three fields play different roles. Goal specifies the desired outcome, Safeguard specifies what should be preserved, and Local Edit is the only field that describes the input-specific transformation the model is supposed to carry out. If TRACE helps bridge the say–do gap by placing an executable edit before the answer, Local Edit should therefore carry the most direct action signal.

We test this hypothesis through matched training interventions, component ablations, and controlled forced-prefix interventions. These experiments ask, respectively, whether the supervised edit must match the current input, whether Local Edit is sufficient on its own, and whether the trained decoder uses it when producing the answer. We complement these interventions with a descriptive attention analysis and a failure taxonomy of the resulting outputs.

#### C.5.1 The Supervised Edit Must Match the Input

We first test whether TRACE benefits simply from having structured edit text, or from providing the correct edit for the current input.

**Setup.** On Qwen3-1.7B across all six TOMG subtasks, prompts and `<answer>` spans are byte-identical, and only the supervised `<think>` surface changes. *Remove* replaces the complete Local Edit block with a fixed placeholder. *Shuffled* replaces it with a no-self-donor edit from another example

in the same subtask and requested direction or canonical operation, preserving task compatibility while breaking its correspondence to the current input. Evaluation uses generation seeds 42/43/44.

Table 20: **Partial-factorial contract ablation.** All arms share prompt–answer pairs, training recipe, reward, and training seed. Scores are six-task per-sample $\mathrm{SR}\times\mathrm{Sim}$ means over generation seeds 42/43/44; unlisted combinations were not run.

| Slice | Arm | Goal | Local Edit | Safeguard | Avg. $\mathrm{SR}\times\mathrm{Sim}$ | $\Delta$ vs. Full |
|---|---|---|---|---|---|---|
| Shared | Full TRACE | yes | sample-specific | yes | **0.5762** | – |
| Local Edit | *Shuffled* | yes | stratified donor | yes | 0.5537 | −0.0225 |
| | *Remove* | yes | placeholder | yes | 0.5459 | −0.0303 |
| Contract | $L$ only | no | sample-specific | no | 0.5591 | −0.0171 |
| | −Safeguard | yes | sample-specific | no | 0.5534 | −0.0228 |
| | −Goal | no | sample-specific | yes | 0.5458 | −0.0304 |
| No trace | SMILES | no | none | no | 0.5165 | −0.0597 |

**Input-matched Local Edit provides the strongest supervision.** Full TRACE reaches 0.5762 average $\mathrm{SR}\times\mathrm{Sim}$, compared with 0.5459 for *Remove* and 0.5537 for *Shuffled* (Table 20). The shuffled edit retains some of the benefit of providing a task-compatible transformation, but does not recover the full TRACE result. In a single-seed MolOpt trajectory, both Full and Shuffled also show fewer late wrong-direction failures than Remove and SMILES-only supervision. Together, these results indicate that useful edit content matters, while matching that edit to the current input is needed for the full gain.

### C.5.2 The Contract Is Not Reducible to Local Edit

Given the benefit of an input-matched Local Edit, we next ask whether it is sufficient on its own.

**Setup.** We ablate Goal, Safeguard, or both under the same matched SFT→GRPO recipe. The partial-factorial design compares $L$ only, $L{+}S$ (−Goal), $G{+}L$ (−Safeguard), and the full $G{+}L{+}S$ contract. Because these component arms use one training seed, we treat the comparison as an interaction diagnostic rather than a general estimate of component effect sizes.

**Local Edit carries most of the signal, but the full contract performs best.** The $L$-only arm is the strongest incomplete variant at 0.5591, but remains 0.0171 below Full TRACE (Table 20). Adding Goal or Safeguard alone does not improve over $L$ only, whereas the complete $G{+}L{+}S$ contract reaches 0.5762. This is consistent with Local Edit specifying the transformation, while Goal and Safeguard provide complementary objective and preservation constraints around it.

The effect also varies by subtask. Removing Goal improves LogP and MR, where edit direction is already explicit in the prompt, but hurts SubComponent and DelComponent, where the requested group must be disambiguated. Local Edit is therefore the dominant component, but its benefit is strongest within the full contract rather than in isolation.

### C.5.3 The Decoder Reads Local Edit at Inference Time

The training ablations show what information is useful during learning. We next test whether the trained decoder also depends on Local Edit when generating the answer.

**Setup.** For each TOMG family, we sample 100 eligible rows per subtask from the Qwen3-1.7B TRACE checkpoint, requiring all TRACE fields to be parseable in the original generation. We then append either the original or an intervened trace to the prompt and re-decode only the answer at temperature 0.1. *Remove* deletes one field, while *Swap* replaces that field or the full trace with a task-compatible trace from another example. Confidence intervals use 5,000 paired bootstrap resamples over the 300 rows in each family.

Note that forced-prefix re-decoding differs from the standard evaluation setting. Under standard evaluation, the model freely generates TRACE and the answer, reaching SR 0.810 on MolOpt and

Table 21: **Paired TRACE-slot intervention.** Cells show the change from the clean own-TRACE forced-prefix control, with paired-bootstrap 95% CI (5,000 resamples, $n$=300 rows per family). Whole $G/L/S$ components are removed or swapped; only the answer is re-decoded.

| | MolOpt | | Structure | |
|---|---|---|---|---|
| Intervention | ΔSR | Δ(SR×Sim) | ΔSR | Δ(SR×Sim) |
| Remove Goal | $-0.010$ $[-0.037, +0.017]$ | $+0.002$ $[-0.017, +0.021]$ | $+0.007$ $[-0.013, +0.027]$ | $+0.002$ $[-0.010, +0.014]$ |
| Remove Local Edit | $-0.133$ $[-0.190, -0.077]$ | $-0.128$ $[-0.169, -0.088]$ | $-0.110$ $[-0.163, -0.057]$ | $-0.081$ $[-0.117, -0.046]$ |
| Remove Safeguard | $+0.010$ $[-0.020, +0.040]$ | $-0.001$ $[-0.021, +0.018]$ | $+0.003$ $[-0.017, +0.023]$ | $+0.000$ $[-0.011, +0.012]$ |
| Swap Goal | $+0.000$ $[+0.000, +0.000]$ | $+0.000$ $[+0.000, +0.000]$ | $+0.003$ $[+0.000, +0.010]$ | $+0.001$ $[-0.001, +0.005]$ |
| Swap Local Edit | $-0.030$ $[-0.070, +0.010]$ | $-0.015$ $[-0.043, +0.012]$ | $-0.020$ $[-0.053, +0.010]$ | $-0.013$ $[-0.032, +0.006]$ |
| Swap Safeguard | $-0.003$ $[-0.023, +0.017]$ | $-0.001$ $[-0.013, +0.011]$ | $+0.003$ $[+0.000, +0.010]$ | $+0.001$ $[-0.002, +0.006]$ |
| Swap full TRACE | $-0.047$ $[-0.083, -0.010]$ | $-0.027$ $[-0.055, -0.001]$ | $-0.023$ $[-0.053, +0.007]$ | $-0.015$ $[-0.033, +0.004]$ |

0.747 on Structure. When the original, unmodified TRACE is instead fixed as a prefix and only the answer is re-decoded, SR drops to 0.697 and 0.480, respectively. We therefore measure all intervention effects relative to this clean forced-prefix control.

**Removing Local Edit consistently degrades inference-time performance.** Within this controlled path, removing $L$ lowers SR by 0.133 on MolOpt and 0.110 on Structure, with corresponding $\mathrm{SR} \times \mathrm{Sim}$ drops of 0.128 and 0.081. All four confidence intervals exclude zero (Table 21). By contrast, removing Goal or Safeguard changes either metric only marginally. These results show that the decoder relies most directly on Local Edit when generating the answer.

**Swap interventions provide additional support.** Replacing $L$ with a task-compatible edit from another example produces the largest slot-level drop in both families, while the effect is smaller than removing $L$ entirely. This mirrors the training intervention: a task-compatible edit retains part of the useful action signal, but matching it to the current input provides additional benefit. Swapping the full TRACE also reduces performance on MolOpt, suggesting that consistency between the broader contract and the current input matters for this family. Limited within-task variation can yield similar or identical Goals, attenuating Goal-swap effects.

### C.5.4 Attention Provides a Descriptive Readout

The forced-prefix interventions show that Local Edit affects answer generation. We next complement these results with an attention analysis of where answer tokens focus within the generated contract.

**Setup.** On the main-table Qwen3-1.7B checkpoints, we run forced decoding on each policy's own evaluation generations and measure self-attention from answer tokens. Because TRACE fields differ in length, we compare per-token attention density within each trace rather than absolute attention mass across arms. The probe uses 100 test rows per subtask (300 per family).

**Answer tokens attend most densely to Local Edit.** Across the TRACE SFT and GRPO checkpoints, Local Edit receives about 1.4–1.8× the average per-token attention density of the full trace (Figure 13). This is consistent with the intervention results and further localizes the answer-generation signal to Local Edit, while the attention analysis itself remains descriptive.

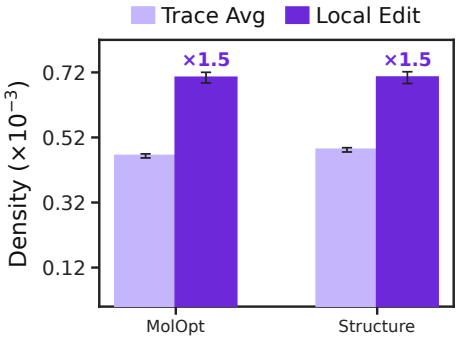



Figure 13: **Answer tokens attend most densely to Local Edit.** Per-token attention density within TRACE for Qwen3-1.7B; $n = 300$ per family. Local Edit receives about $1.5\times$ the full-trace average.

### C.5.5 Where the Gains Land

The preceding analyses support Local Edit as the main action-bearing field. We finally examine how this advantage appears in the model's observable failure modes.

**Setup.** We re-score the generated outputs from all six TOMG subtasks (5,000 per subtask) for SFT-SMILES+GRPO, REPO, and TRACE+GRPO across generation seeds 42/43/44.

Failure categories are mutually exclusive under a fixed precedence rule. *Invalid* denotes an unparseable or chemically invalid output; *No-op* denotes an unchanged molecule; and *Wrong direction* denotes a MolOpt edit that moves the requested property in the wrong direction. For Structure, *Wrong operation* means that the requested structural edit is not realized, *Preservation/fragment* captures violations of the required preservation constraints, and *Low similarity* denotes an otherwise valid edit that falls below the task similarity criterion. The SUCCESS matches the reported SR.

Table 22: **Mutually exclusive TOMG failure taxonomy.** Rates are mean$_{\pm}$std over generation seeds 42/43/44, and SUCCESS equals the main-table SR.

| Family | Outcome category | SMILES+GRPO | REPO | TRACE+GRPO |
|---|---|---|---|---|
| MOLOPT | Invalid | $0.060_{\pm 0.000}$ | $0.002_{\pm 0.000}$ | $0.028_{\pm 0.001}$ |
| | No-op | $0.027_{\pm 0.000}$ | $0.031_{\pm 0.000}$ | $0.001_{\pm 0.000}$ |
| | Wrong direction | $0.252_{\pm 0.000}$ | $0.483_{\pm 0.002}$ | $0.190_{\pm 0.001}$ |
| | Success | $0.661_{\pm 0.000}$ | $0.485_{\pm 0.003}$ | $0.781_{\pm 0.001}$ |
| STRUCTURE | Invalid | $0.059_{\pm 0.000}$ | $0.017_{\pm 0.001}$ | $0.026_{\pm 0.001}$ |
| | No-op | $0.020_{\pm 0.000}$ | $0.064_{\pm 0.001}$ | $0.013_{\pm 0.000}$ |
| | Preservation/fragment | $0.008_{\pm 0.000}$ | $0.127_{\pm 0.002}$ | $0.023_{\pm 0.001}$ |
| | Low similarity | $0.003_{\pm 0.000}$ | $0.033_{\pm 0.001}$ | $0.002_{\pm 0.000}$ |
| | Wrong operation | $0.229_{\pm 0.000}$ | $0.310_{\pm 0.001}$ | $0.204_{\pm 0.001}$ |
| | Success | $0.681_{\pm 0.000}$ | $0.450_{\pm 0.001}$ | $0.732_{\pm 0.001}$ |

Table 23: **Where successful edits occur.** Entries are percentages of all rows that both succeed and fall in each similarity bin, using the same checkpoints as Table 22.

| Family | Arm | $[0, 0.1)$ | $[0.1, 0.4)$ | $[0.4, 0.6)$ | $[0.6, 0.8)$ | $[0.8, 1]$ |
|---|---|---|---|---|---|---|
| MOLOPT | SMILES+GRPO | $0.00_{\pm 0.00}$ | $0.03_{\pm 0.00}$ | $1.77_{\pm 0.00}$ | $29.44_{\pm 0.04}$ | $34.82_{\pm 0.02}$ |
| | TRACE+GRPO | $1.25_{\pm 0.09}$ | $0.17_{\pm 0.03}$ | $1.21_{\pm 0.07}$ | $34.16_{\pm 0.14}$ | $41.28_{\pm 0.07}$ |
| STRUCTURE | SMILES+GRPO | $0.00_{\pm 0.00}$ | $0.00_{\pm 0.00}$ | $3.22_{\pm 0.03}$ | $44.11_{\pm 0.02}$ | $20.80_{\pm 0.01}$ |
| | TRACE+GRPO | $0.00_{\pm 0.00}$ | $0.00_{\pm 0.00}$ | $4.19_{\pm 0.04}$ | $49.62_{\pm 0.05}$ | $19.39_{\pm 0.00}$ |

**TRACE primarily reduces ineffective or misdirected edits.** On MolOpt, TRACE substantially reduces wrong-direction outputs and nearly eliminates no-op generations, while also lowering the invalid rate relative to SMILES+GRPO. On Structure, the largest reductions are in wrong-operation, invalid, and no-op failures, together with a small reduction in low-similarity outputs (Table 22). Preservation/fragment failures increase slightly, indicating that the gain is not uniform across all error types. The additional successes nevertheless occur mainly at moderate-to-high similarity (Table 23), suggesting that the improvement comes primarily from carrying out the requested transformation more reliably rather than from allowing larger departures from the input.

**Conclusion.** Taken together, these analyses support Local Edit as the main action-bearing component through which TRACE narrows the say–do gap. Its supervision is most effective when matched to the current input, the trained decoder relies on it when generating the answer, and answer tokens attend to it most densely within the contract. Local Edit is nevertheless not sufficient on its own: the full Goal–Local Edit–Safeguard contract performs best. Consistent with this mechanism, TRACE's gains appear primarily as fewer no-op, wrong-direction, and wrong-operation outputs, rather than as less constrained departures from the input.

## D Expanded Related Work

This section expands Section 4 along five lines: molecular editing and design, outcome and process supervision, reasoning-data construction, chain-of-thought faithfulness, and executable verification.

**Molecule editing and optimization with LLMs.** Text-conditioned models learn mappings between molecules and language (Edwards et al., 2022; 2021), generate molecules from descriptions (Li et al., 2024; Liu et al., 2023), and perform instruction-guided property editing (Ye et al., 2023; Dey et al., 2025). TOMG and MUMO provide controlled settings for constrained molecular editing with explicit optimization and preservation criteria (Li et al., 2026a; Dey et al., 2025). RePO further applies reference-guided RL to molecular editing while generating free-form reasoning (Li et al., 2026b), connecting this setting to broader work that distills reasoning from stronger LLMs or optimizes it with process rewards (Mukherjee et al., 2023; Lightman et al., 2023). SMDD-Bench places related operations, including lead optimization and scaffold hopping, inside longer-horizon workflows with protein targets and tool use (Han et al., 2026). TRACE differs in representing the declared transformation as a typed edit contract that can be checked directly against the generated object. We therefore use molecular editing as a controlled testbed for whether a model can carry out a requested change while preserving the relevant parts of the input, rather than as a complete simulation of hit-to-lead discovery.

**Outcome, rubric, and process rewards.** RLVR improves reasoning and generation by rewarding checkable final answers (Lambert et al., 2025; Guo et al., 2025). Common optimizers include GRPO (Shao et al., 2024), REINFORCE-style objectives (Ahmadian et al., 2024; Zheng et al., 2025; MiniMax et al., 2025), and PPO (Schulman et al., 2017; Ouyang et al., 2022). In molecular editing, RePO (Li et al., 2026b) couples GRPO to a similarity-aware property reward and generates a free-form `<think>` trace before the molecule. Its reward evaluates the final molecule rather than requiring the intermediate trace to expose claims that can be replayed against the answer.

Post-training can also supervise intermediate reasoning. Process reward models score intermediate steps (Lightman et al., 2023; Wang et al., 2024; Uesato et al., 2022). Rubrics as Rewards evaluates outputs against instance-specific criteria using an LLM grader (Gunjal et al., 2026), while step-wise rubrics localize this feedback to individual reasoning steps (Xie et al., 2026). TRACE differs in defining a typed claim space with task-specific execution semantics: the checker recomputes whether those claims are realized in the generated answer, rather than assigning a score directly to unrestricted reasoning text.

**Language-model-authored reasoning data.** Reasoning supervision can be distilled from explanations generated by stronger closed-source models, as in Orca's use of GPT-4 and ChatGPT traces (Mukherjee et al., 2023). Models can also bootstrap their own supervision: STaR iteratively trains on rationales associated with correct answers (Zelikman et al., 2022), while ReST alternates generation, filtering, and fine-tuning (Gulcehre et al., 2023). These approaches can provide useful intermediate supervision, but LLM-authored rationales are typically generated independently for each example and need not be grounded in a fixed set of task facts or edit semantics. Construction can also be

costly at scale because every training example requires additional model inference, especially when a frontier LLM is used as the teacher.

TRACE instead renders supervision from a fixed task-specific fact layer into a shared typed contract. When domain tools are available, this construction is deterministic and does not require frontier-model inference for every training example. The narrative controls in Section C.4.2 compare this fact-derived contract with frontier-LLM-authored free-form reasoning under matched downstream training. The no-tool control in Section C.4.3 holds the TRACE schema fixed while replacing deterministic fact extraction with frontier-LLM construction. These experiments separate the representation of the supervision from the method and cost of constructing it.

**Chain-of-thought faithfulness and the say–do gap.** A reasoning trace is useful for oversight only to the extent that its stated content corresponds to the behavior that produces the answer (Jacovi & Goldberg, 2020). Prior work studies this relationship through cue interventions, trace perturbations, and monitoring analyses. Free-form CoT can omit information that determines the answer (Turpin et al., 2023), remain stable when its stated reasoning is perturbed (Lanham et al., 2023), and remain unfaithful under natural prompts (Arcuschin et al., 2026). Related failures persist in RL-trained reasoning models (Chen et al., 2025), motivating methods that inspect or supervise intermediate reasoning rather than relying on final-answer accuracy alone (Korbak et al., 2025; Baker et al., 2025).

Yu et al. (2026) further show that improving answer accuracy under outcome-only RLVR does not reliably increase the causal importance or informational sufficiency of the generated trace. TRACE targets a narrower, directly testable relationship: whether the generated object carries out the edit declared in the trace. Typed edit claims give this relationship explicit execution semantics, allowing $D$ to measure say–do consistency directly against the generated answer. We do not equate this with complete causal faithfulness; it tests the more specific question of whether the model does what its trace says.

**Executable reasoning and validation artifacts.** Tool-augmented agents such as ChemCrow and ReAct ground generation through external actions (Bran et al., 2023; Yao et al., 2023); PoT and PAL execute model-produced programs (Chen et al., 2023; Gao et al., 2023). Faithful CoT executes a symbolic chain with a deterministic solver (Lyu et al., 2023). VeryTrace translates free-form traces into a compilable DSL and combines mechanical checks with targeted LLM audits (Zhong et al., 2026). HarnessLLM trains a model to generate executable test harnesses that validate program outputs and support test-time selection (Liu et al., 2025).

These methods expose different executable interfaces. Tool-using agents execute external actions, program-based reasoning executes intermediate computations, and test harnesses validate program behavior. TRACE instead connects a declarative edit contract to a separately generated structured answer and asks whether the latter realizes the former. This trace–answer interface allows the same task-specific checker to support construction auditing, optional process feedback, and reference-answer-free test-time selection.

# E Detailed Result Tables

The tables below give the complete per-task results for the remaining molecular settings.

Table 24: **Per-task TOMG-Bench results with Qwen3-4B.** Rows report SR, Sim, and per-sample SR×Sim; cells are mean±std over generation seeds 42/43/44.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Optimization | LogP | SR | $0.7195_{\pm 0.0092}$ | $\underline{0.8395}_{\pm 0.0021}$ | $0.5919_{\pm 0.0021}$ | $0.7129_{\pm 0.0010}$ | $\mathbf{0.9011}_{\pm 0.0001}$ |
| | | Sim | $0.7058_{\pm 0.0045}$ | $0.6367_{\pm 0.0028}$ | $0.6478_{\pm 0.0024}$ | $\underline{0.7089}_{\pm 0.0006}$ | $\mathbf{0.7704}_{\pm 0.0002}$ |
| | | SR×Sim | $0.5025_{\pm 0.0048}$ | $\underline{0.5479}_{\pm 0.0020}$ | $0.3834_{\pm 0.0019}$ | $0.4990_{\pm 0.0014}$ | $\mathbf{0.7024}_{\pm 0.0003}$ |
| | MR | SR | $0.7244_{\pm 0.0081}$ | $\underline{0.7870}_{\pm 0.0041}$ | $0.8044_{\pm 0.0035}$ | $0.7177_{\pm 0.0014}$ | $\mathbf{0.8421}_{\pm 0.0016}$ |
| | | Sim | $0.7165_{\pm 0.0103}$ | $0.6347_{\pm 0.0012}$ | $0.6307_{\pm 0.0009}$ | $\underline{0.7235}_{\pm 0.0004}$ | $\mathbf{0.7594}_{\pm 0.0017}$ |
| | | SR×Sim | $0.5026_{\pm 0.0051}$ | $\underline{0.5067}_{\pm 0.0031}$ | $0.5070_{\pm 0.0017}$ | $0.4980_{\pm 0.0013}$ | $\mathbf{0.6471}_{\pm 0.0000}$ |
| | QED | SR | $0.5061_{\pm 0.0045}$ | $\underline{0.5571}_{\pm 0.0060}$ | $0.5451_{\pm 0.0038}$ | $0.5030_{\pm 0.0029}$ | $\mathbf{0.6542}_{\pm 0.0007}$ |
| | | Sim | $0.7015_{\pm 0.0063}$ | $0.6434_{\pm 0.0029}$ | $0.6422_{\pm 0.0024}$ | $\underline{0.7059}_{\pm 0.0006}$ | $\mathbf{0.7669}_{\pm 0.0012}$ |
| | | SR×Sim | $0.3538_{\pm 0.0024}$ | $\underline{0.3681}_{\pm 0.0036}$ | $0.3520_{\pm 0.0023}$ | $0.3519_{\pm 0.0018}$ | $\mathbf{0.5028}_{\pm 0.0008}$ |
| Structure Optimization | Add | SR | $0.5982_{\pm 0.0166}$ | $0.6629_{\pm 0.0003}$ | $0.6305_{\pm 0.0046}$ | $\underline{0.6905}_{\pm 0.0003}$ | $\mathbf{0.7290}_{\pm 0.0003}$ |
| | | Sim | $0.6965_{\pm 0.0032}$ | $0.6531_{\pm 0.0010}$ | $0.6194_{\pm 0.0009}$ | $\underline{0.7608}_{\pm 0.0001}$ | $\mathbf{0.7645}_{\pm 0.0003}$ |
| | | SR×Sim | $0.4200_{\pm 0.0118}$ | $0.4649_{\pm 0.0002}$ | $0.4340_{\pm 0.0033}$ | $\underline{0.5230}_{\pm 0.0004}$ | $\mathbf{0.5667}_{\pm 0.0003}$ |
| | Del | SR | $0.6411_{\pm 0.0086}$ | $0.6659_{\pm 0.0015}$ | $0.5021_{\pm 0.0038}$ | $\underline{0.7595}_{\pm 0.0010}$ | $\mathbf{0.8138}_{\pm 0.0009}$ |
| | | Sim | $0.6692_{\pm 0.0079}$ | $0.5840_{\pm 0.0013}$ | $0.5563_{\pm 0.0019}$ | $\mathbf{0.6968}_{\pm 0.0002}$ | $\underline{0.6773}_{\pm 0.0007}$ |
| | | SR×Sim | $0.4225_{\pm 0.0046}$ | $0.4313_{\pm 0.0009}$ | $0.3262_{\pm 0.0023}$ | $\underline{0.5349}_{\pm 0.0007}$ | $\mathbf{0.5726}_{\pm 0.0007}$ |
| | Sub | SR | $0.6306_{\pm 0.0052}$ | $0.6849_{\pm 0.0002}$ | $0.4561_{\pm 0.0007}$ | $\underline{0.6985}_{\pm 0.0004}$ | $\mathbf{0.7590}_{\pm 0.0010}$ |
| | | Sim | $0.7310_{\pm 0.0012}$ | $0.7048_{\pm 0.0004}$ | $0.6021_{\pm 0.0004}$ | $\mathbf{0.7597}_{\pm 0.0002}$ | $\underline{0.7485}_{\pm 0.0007}$ |
| | | SR×Sim | $0.4767_{\pm 0.0036}$ | $0.5145_{\pm 0.0002}$ | $0.3416_{\pm 0.0004}$ | $\underline{0.5315}_{\pm 0.0003}$ | $\mathbf{0.5737}_{\pm 0.0007}$ |
| All Tasks | Avg | SR | $0.6366_{\pm 0.0085}$ | $\underline{0.6995}_{\pm 0.0005}$ | $0.5883_{\pm 0.0012}$ | $0.6804_{\pm 0.0007}$ | $\mathbf{0.7832}_{\pm 0.0006}$ |
| | | Sim | $0.7034_{\pm 0.0041}$ | $0.6428_{\pm 0.0007}$ | $0.6164_{\pm 0.0009}$ | $\underline{0.7259}_{\pm 0.0003}$ | $\mathbf{0.7478}_{\pm 0.0002}$ |
| | | SR×Sim | $0.4463_{\pm 0.0052}$ | $0.4722_{\pm 0.0002}$ | $0.3907_{\pm 0.0011}$ | $\underline{0.4897}_{\pm 0.0005}$ | $\mathbf{0.5942}_{\pm 0.0003}$ |

Table 25: **Per-task TOMG-Bench results with Qwen3-8B.** Rows report SR, Sim, and per-sample SR×Sim; cells are mean±std over generation seeds 42/43/44.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Optimization | LogP | SR | $0.7781_{\pm 0.0008}$ | $\underline{0.8370}_{\pm 0.0026}$ | $0.6295_{\pm 0.0025}$ | $0.8237_{\pm 0.0004}$ | $\mathbf{0.9279}_{\pm 0.0008}$ |
| | | Sim | $0.6932_{\pm 0.0002}$ | $0.6437_{\pm 0.0019}$ | $0.6451_{\pm 0.0018}$ | $\underline{0.7512}_{\pm 0.0004}$ | $\mathbf{0.7892}_{\pm 0.0007}$ |
| | | SR×Sim | $0.5364_{\pm 0.0004}$ | $0.5489_{\pm 0.0032}$ | $0.4063_{\pm 0.0015}$ | $\underline{0.6148}_{\pm 0.0002}$ | $\mathbf{0.7377}_{\pm 0.0007}$ |
| | MR | SR | $0.7463_{\pm 0.0010}$ | $\underline{0.8110}_{\pm 0.0017}$ | $0.6964_{\pm 0.0048}$ | $0.6995_{\pm 0.0007}$ | $\mathbf{0.9030}_{\pm 0.0026}$ |
| | | Sim | $0.7053_{\pm 0.0006}$ | $0.6469_{\pm 0.0024}$ | $0.6407_{\pm 0.0030}$ | $\mathbf{0.7688}_{\pm 0.0001}$ | $\underline{0.7536}_{\pm 0.0026}$ |
| | | SR×Sim | $0.5172_{\pm 0.0011}$ | $\underline{0.5292}_{\pm 0.0030}$ | $0.4442_{\pm 0.0023}$ | $0.5270_{\pm 0.0004}$ | $\mathbf{0.6833}_{\pm 0.0029}$ |
| | QED | SR | $0.4939_{\pm 0.0006}$ | $\underline{0.5565}_{\pm 0.0060}$ | $0.5163_{\pm 0.0012}$ | $0.5064_{\pm 0.0002}$ | $\mathbf{0.6791}_{\pm 0.0017}$ |
| | | Sim | $0.6930_{\pm 0.0001}$ | $0.6506_{\pm 0.0030}$ | $0.6508_{\pm 0.0008}$ | $\underline{0.7533}_{\pm 0.0002}$ | $\mathbf{0.7686}_{\pm 0.0015}$ |
| | | SR×Sim | $0.3410_{\pm 0.0002}$ | $0.3693_{\pm 0.0035}$ | $0.3320_{\pm 0.0007}$ | $\underline{0.3802}_{\pm 0.0001}$ | $\mathbf{0.5313}_{\pm 0.0009}$ |
| Structure Optimization | Add | SR | $0.6473_{\pm 0.0007}$ | $\underline{0.7779}_{\pm 0.0010}$ | $0.6069_{\pm 0.0004}$ | $0.6981_{\pm 0.0001}$ | $\mathbf{0.8249}_{\pm 0.0118}$ |
| | | Sim | $0.6979_{\pm 0.0002}$ | $0.6468_{\pm 0.0012}$ | $0.5815_{\pm 0.0018}$ | $\mathbf{0.7705}_{\pm 0.0001}$ | $\underline{0.7456}_{\pm 0.0010}$ |
| | | SR×Sim | $0.4514_{\pm 0.0005}$ | $\underline{0.5406}_{\pm 0.0008}$ | $0.4130_{\pm 0.0010}$ | $0.5372_{\pm 0.0001}$ | $\mathbf{0.6239}_{\pm 0.0089}$ |
| | Del | SR | $0.7046_{\pm 0.0010}$ | $0.6908_{\pm 0.0014}$ | $0.4953_{\pm 0.0044}$ | $\underline{0.7407}_{\pm 0.0010}$ | $\mathbf{0.8417}_{\pm 0.0023}$ |
| | | Sim | $0.6504_{\pm 0.0005}$ | $0.5903_{\pm 0.0009}$ | $0.5368_{\pm 0.0019}$ | $\mathbf{0.6951}_{\pm 0.0004}$ | $\underline{0.6869}_{\pm 0.0003}$ |
| | | SR×Sim | $0.4595_{\pm 0.0006}$ | $0.4460_{\pm 0.0007}$ | $0.3165_{\pm 0.0025}$ | $\underline{0.5235}_{\pm 0.0007}$ | $\mathbf{0.5937}_{\pm 0.0016}$ |
| | Sub | SR | $0.6761_{\pm 0.0003}$ | $0.6514_{\pm 0.0019}$ | $0.4731_{\pm 0.0017}$ | $\underline{0.7181}_{\pm 0.0002}$ | $\mathbf{0.7383}_{\pm 0.0007}$ |
| | | Sim | $0.7389_{\pm 0.0004}$ | $0.7197_{\pm 0.0004}$ | $0.5954_{\pm 0.0016}$ | $\mathbf{0.7590}_{\pm 0.0003}$ | $\underline{0.7420}_{\pm 0.0009}$ |
| | | SR×Sim | $0.5106_{\pm 0.0002}$ | $0.4915_{\pm 0.0012}$ | $0.3486_{\pm 0.0009}$ | $\underline{0.5468}_{\pm 0.0002}$ | $\mathbf{0.5597}_{\pm 0.0005}$ |
| All Tasks | Avg | SR | $0.6744_{\pm 0.0001}$ | $\underline{0.7208}_{\pm 0.0013}$ | $0.5696_{\pm 0.0008}$ | $0.6978_{\pm 0.0003}$ | $\mathbf{0.8191}_{\pm 0.0023}$ |
| | | Sim | $0.6965_{\pm 0.0002}$ | $0.6497_{\pm 0.0008}$ | $0.6084_{\pm 0.0004}$ | $\mathbf{0.7497}_{\pm 0.0002}$ | $\underline{0.7477}_{\pm 0.0002}$ |
| | | SR×Sim | $0.4693_{\pm 0.0002}$ | $0.4876_{\pm 0.0002}$ | $0.3767_{\pm 0.0006}$ | $\underline{0.5216}_{\pm 0.0001}$ | $\mathbf{0.6216}_{\pm 0.0016}$ |

Table 26: **TOMG-Bench results with Gemma-3-12B-LoRA ($r = 256$, $\alpha = 512$).** Rows report SR, Sim, and per-sample $\mathrm{SR}\times\mathrm{Sim}$; cells are mean±std over generation seeds 42/43/44.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Optimization | LogP | SR | $0.8011_{\pm 0.0023}$ | $0.8398_{\pm 0.0042}$ | $0.6535_{\pm 0.0052}$ | $\underline{0.8418}_{\pm 0.0016}$ | $\mathbf{0.9485}_{\pm 0.0008}$ |
| | | Sim | $0.7080_{\pm 0.0011}$ | $0.6554_{\pm 0.0032}$ | $0.6501_{\pm 0.0010}$ | $\mathbf{0.7718}_{\pm 0.0005}$ | $\underline{0.7204}_{\pm 0.0005}$ |
| | | SR×Sim | $0.5601_{\pm 0.0011}$ | $0.5542_{\pm 0.0043}$ | $0.4258_{\pm 0.0040}$ | $\underline{0.6472}_{\pm 0.0014}$ | $\mathbf{0.6916}_{\pm 0.0004}$ |
| | MR | SR | $0.7115_{\pm 0.0019}$ | $0.7737_{\pm 0.0039}$ | $0.6519_{\pm 0.0081}$ | $\underline{0.7929}_{\pm 0.0010}$ | $\mathbf{0.8368}_{\pm 0.0020}$ |
| | | Sim | $0.7203_{\pm 0.0004}$ | $0.6487_{\pm 0.0077}$ | $0.6530_{\pm 0.0017}$ | $\mathbf{0.7867}_{\pm 0.0004}$ | $\underline{0.7221}_{\pm 0.0018}$ |
| | | SR×Sim | $0.4999_{\pm 0.0011}$ | $0.5013_{\pm 0.0035}$ | $0.4261_{\pm 0.0024}$ | $\mathbf{0.6219}_{\pm 0.0009}$ | $\underline{0.6024}_{\pm 0.0013}$ |
| | QED | SR | $0.5275_{\pm 0.0006}$ | $\underline{0.5436}_{\pm 0.0104}$ | $0.5095_{\pm 0.0034}$ | $0.4968_{\pm 0.0023}$ | $\mathbf{0.6594}_{\pm 0.0027}$ |
| | | Sim | $0.7073_{\pm 0.0003}$ | $0.6498_{\pm 0.0009}$ | $0.6541_{\pm 0.0028}$ | $\mathbf{0.7940}_{\pm 0.0004}$ | $\underline{0.7156}_{\pm 0.0017}$ |
| | | SR×Sim | $0.3708_{\pm 0.0007}$ | $0.3565_{\pm 0.0086}$ | $0.3333_{\pm 0.0027}$ | $\underline{0.3931}_{\pm 0.0017}$ | $\mathbf{0.4869}_{\pm 0.0029}$ |
| Structure Optimization | Add | SR | $0.6343_{\pm 0.0017}$ | $\underline{0.6922}_{\pm 0.0019}$ | $0.6543_{\pm 0.0022}$ | $0.6741_{\pm 0.0010}$ | $\mathbf{0.7795}_{\pm 0.0017}$ |
| | | Sim | $0.7007_{\pm 0.0004}$ | $0.6763_{\pm 0.0004}$ | $0.6282_{\pm 0.0020}$ | $\mathbf{0.7548}_{\pm 0.0002}$ | $\underline{0.7216}_{\pm 0.0014}$ |
| | | SR×Sim | $0.4433_{\pm 0.0016}$ | $0.4876_{\pm 0.0016}$ | $0.4468_{\pm 0.0020}$ | $\underline{0.5073}_{\pm 0.0007}$ | $\mathbf{0.5939}_{\pm 0.0012}$ |
| | Del | SR | $0.6514_{\pm 0.0030}$ | $0.6530_{\pm 0.0024}$ | $0.5771_{\pm 0.0036}$ | $\underline{0.6910}_{\pm 0.0005}$ | $\mathbf{0.7488}_{\pm 0.0012}$ |
| | | Sim | $0.6530_{\pm 0.0004}$ | $0.5935_{\pm 0.0004}$ | $0.5417_{\pm 0.0037}$ | $\mathbf{0.7038}_{\pm 0.0006}$ | $\underline{0.6697}_{\pm 0.0003}$ |
| | | SR×Sim | $0.4256_{\pm 0.0020}$ | $0.4204_{\pm 0.0013}$ | $0.3652_{\pm 0.0024}$ | $\underline{0.4857}_{\pm 0.0005}$ | $\mathbf{0.5178}_{\pm 0.0010}$ |
| | Sub | SR | $0.6051_{\pm 0.0011}$ | $\underline{0.6863}_{\pm 0.0011}$ | $0.4097_{\pm 0.0030}$ | $\mathbf{0.7085}_{\pm 0.0008}$ | $0.6446_{\pm 0.0007}$ |
| | | Sim | $\underline{0.7421}_{\pm 0.0001}$ | $0.7278_{\pm 0.0007}$ | $0.5951_{\pm 0.0025}$ | $\mathbf{0.7515}_{\pm 0.0002}$ | $0.7334_{\pm 0.0014}$ |
| | | SR×Sim | $0.4574_{\pm 0.0008}$ | $\underline{0.5145}_{\pm 0.0007}$ | $0.3011_{\pm 0.0020}$ | $\mathbf{0.5345}_{\pm 0.0005}$ | $0.4899_{\pm 0.0006}$ |
| All Tasks | Avg | SR | $0.6552_{\pm 0.0011}$ | $0.6981_{\pm 0.0016}$ | $0.5760_{\pm 0.0016}$ | $\underline{0.7008}_{\pm 0.0002}$ | $\mathbf{0.7696}_{\pm 0.0002}$ |
| | | Sim | $0.7052_{\pm 0.0002}$ | $0.6586_{\pm 0.0015}$ | $0.6204_{\pm 0.0013}$ | $\mathbf{0.7605}_{\pm 0.0003}$ | $\underline{0.7138}_{\pm 0.0002}$ |
| | | SR×Sim | $0.4595_{\pm 0.0006}$ | $0.4724_{\pm 0.0021}$ | $0.3830_{\pm 0.0005}$ | $\underline{0.5316}_{\pm 0.0001}$ | $\mathbf{0.5637}_{\pm 0.0001}$ |

Table 27: **TOMG-Bench results with Llama-3.1-8B-Instruct.** This full-fine-tuning run is a cross-backbone robustness check, not a Qwen scaling point. Rows report SR, Sim, and per-sample $\mathrm{SR}\times\mathrm{Sim}$; cells are mean±std over generation seeds 42/43/44.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Optimization | LogP | SR | $0.7029_{\pm 0.0013}$ | $0.8313_{\pm 0.0058}$ | $0.6545_{\pm 0.0063}$ | $\underline{0.8640}_{\pm 0.0006}$ | $\mathbf{0.9105}_{\pm 0.0022}$ |
| | | Sim | $0.6899_{\pm 0.0001}$ | $0.6422_{\pm 0.0037}$ | $0.5477_{\pm 0.0014}$ | $\mathbf{0.7529}_{\pm 0.0003}$ | $\underline{0.7243}_{\pm 0.0032}$ |
| | | SR×Sim | $0.4771_{\pm 0.0010}$ | $0.5446_{\pm 0.0021}$ | $0.3647_{\pm 0.0039}$ | $\underline{0.6494}_{\pm 0.0004}$ | $\mathbf{0.6725}_{\pm 0.0011}$ |
| | MR | SR | $0.7004_{\pm 0.0012}$ | $\underline{0.7660}_{\pm 0.0029}$ | $0.7202_{\pm 0.0040}$ | $0.7075_{\pm 0.0019}$ | $\mathbf{0.8308}_{\pm 0.0013}$ |
| | | Sim | $\underline{0.7236}_{\pm 0.0009}$ | $0.6171_{\pm 0.0060}$ | $0.5399_{\pm 0.0028}$ | $\mathbf{0.7846}_{\pm 0.0003}$ | $0.7162_{\pm 0.0009}$ |
| | | SR×Sim | $0.4844_{\pm 0.0008}$ | $0.4800_{\pm 0.0031}$ | $0.3869_{\pm 0.0032}$ | $\underline{0.5352}_{\pm 0.0013}$ | $\mathbf{0.5954}_{\pm 0.0007}$ |
| | QED | SR | $0.4987_{\pm 0.0032}$ | $\underline{0.5553}_{\pm 0.0067}$ | $0.4975_{\pm 0.0038}$ | $0.5457_{\pm 0.0022}$ | $\mathbf{0.5658}_{\pm 0.0029}$ |
| | | Sim | $0.6931_{\pm 0.0002}$ | $0.6312_{\pm 0.0046}$ | $0.5295_{\pm 0.0039}$ | $\mathbf{0.7659}_{\pm 0.0001}$ | $\underline{0.7288}_{\pm 0.0019}$ |
| | | SR×Sim | $0.3430_{\pm 0.0022}$ | $0.3619_{\pm 0.0065}$ | $0.2776_{\pm 0.0010}$ | $\underline{0.4167}_{\pm 0.0015}$ | $\mathbf{0.4217}_{\pm 0.0017}$ |
| Structure Optimization | Add | SR | $0.6151_{\pm 0.0016}$ | $\underline{0.6465}_{\pm 0.0008}$ | $0.5223_{\pm 0.0049}$ | $0.5243_{\pm 0.0012}$ | $\mathbf{0.6984}_{\pm 0.0006}$ |
| | | Sim | $0.6907_{\pm 0.0008}$ | $0.6450_{\pm 0.0014}$ | $0.5467_{\pm 0.0012}$ | $\underline{0.7301}_{\pm 0.0003}$ | $\mathbf{0.7552}_{\pm 0.0004}$ |
| | | SR×Sim | $0.4301_{\pm 0.0011}$ | $\underline{0.4540}_{\pm 0.0002}$ | $0.3511_{\pm 0.0024}$ | $0.3940_{\pm 0.0009}$ | $\mathbf{0.5293}_{\pm 0.0006}$ |
| | Del | SR | $\underline{0.6346}_{\pm 0.0014}$ | $0.6217_{\pm 0.0014}$ | $0.4266_{\pm 0.0024}$ | $0.5565_{\pm 0.0014}$ | $\mathbf{0.7459}_{\pm 0.0015}$ |
| | | Sim | $0.6605_{\pm 0.0002}$ | $0.5607_{\pm 0.0010}$ | $0.4797_{\pm 0.0028}$ | $\mathbf{0.7213}_{\pm 0.0003}$ | $\underline{0.6761}_{\pm 0.0005}$ |
| | | SR×Sim | $\underline{0.4133}_{\pm 0.0010}$ | $0.4022_{\pm 0.0012}$ | $0.2702_{\pm 0.0017}$ | $0.3926_{\pm 0.0010}$ | $\mathbf{0.5194}_{\pm 0.0010}$ |
| | Sub | SR | $0.6563_{\pm 0.0001}$ | $\underline{0.6885}_{\pm 0.0008}$ | $0.2943_{\pm 0.0057}$ | $0.4201_{\pm 0.0006}$ | $\mathbf{0.7665}_{\pm 0.0006}$ |
| | | Sim | $\underline{0.7380}_{\pm 0.0002}$ | $0.7204_{\pm 0.0005}$ | $0.4868_{\pm 0.0009}$ | $\mathbf{0.7397}_{\pm 0.0002}$ | $0.7361_{\pm 0.0007}$ |
| | | SR×Sim | $0.4965_{\pm 0.0002}$ | $\underline{0.5157}_{\pm 0.0004}$ | $0.2128_{\pm 0.0036}$ | $0.3203_{\pm 0.0005}$ | $\mathbf{0.5787}_{\pm 0.0004}$ |
| All Tasks | Avg | SR | $0.6347_{\pm 0.0007}$ | $\underline{0.6849}_{\pm 0.0003}$ | $0.5192_{\pm 0.0018}$ | $0.6030_{\pm 0.0007}$ | $\mathbf{0.7530}_{\pm 0.0012}$ |
| | | Sim | $0.6993_{\pm 0.0003}$ | $0.6361_{\pm 0.0016}$ | $0.5217_{\pm 0.0007}$ | $\mathbf{0.7491}_{\pm 0.0001}$ | $\underline{0.7228}_{\pm 0.0010}$ |
| | | SR×Sim | $0.4407_{\pm 0.0005}$ | $\underline{0.4597}_{\pm 0.0008}$ | $0.3106_{\pm 0.0015}$ | $0.4514_{\pm 0.0005}$ | $\mathbf{0.5528}_{\pm 0.0005}$ |

Table 28: **MuMOInstruct multi-property optimization with Qwen3-1.7B.** BDP, BDQ, and BPQ jointly optimize three properties on seen and unseen instruction splits; AVG averages all six split–task cells. Cells are mean±std over generation seeds 42/43/44.

| Split | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| SEEN | BDP | SR | $0.4920_{\pm 0.0053}$ | $0.4387_{\pm 0.0595}$ | $0.5350_{\pm 0.0060}$ | $\mathbf{0.7667}_{\pm 0.0081}$ | $\underline{0.6240}_{\pm 0.0122}$ |
| | | Sim | $0.5750_{\pm 0.0095}$ | $\underline{0.5847}_{\pm 0.0823}$ | $\mathbf{0.6020}_{\pm 0.0060}$ | $0.3767_{\pm 0.0007}$ | $0.4667_{\pm 0.0108}$ |
| | | SR×Sim | $0.2514_{\pm 0.0014}$ | $0.2338_{\pm 0.0138}$ | $\underline{0.2680}_{\pm 0.0040}$ | $0.2642_{\pm 0.0036}$ | $\mathbf{0.2726}_{\pm 0.0027}$ |
| | BDQ | SR | $0.2560_{\pm 0.0060}$ | $0.3220_{\pm 0.0120}$ | $0.3840_{\pm 0.0060}$ | $\underline{0.4420}_{\pm 0.0144}$ | $\mathbf{0.5493}_{\pm 0.0117}$ |
| | | Sim | $\mathbf{0.6674}_{\pm 0.0107}$ | $0.5271_{\pm 0.0262}$ | $\underline{0.6423}_{\pm 0.0057}$ | $0.4910_{\pm 0.0053}$ | $0.4504_{\pm 0.0043}$ |
| | | SR×Sim | $0.1538_{\pm 0.0015}$ | $0.1476_{\pm 0.0040}$ | $\underline{0.2039}_{\pm 0.0039}$ | $0.1919_{\pm 0.0060}$ | $\mathbf{0.2311}_{\pm 0.0053}$ |
| | BPQ | SR | $0.5040_{\pm 0.0053}$ | $0.4887_{\pm 0.0064}$ | $0.5887_{\pm 0.0042}$ | $\mathbf{0.8873}_{\pm 0.0012}$ | $\underline{0.7547}_{\pm 0.0103}$ |
| | | Sim | $\underline{0.5990}_{\pm 0.0027}$ | $0.5539_{\pm 0.0529}$ | $\mathbf{0.6003}_{\pm 0.0043}$ | $0.2656_{\pm 0.0088}$ | $0.4679_{\pm 0.0042}$ |
| | | SR×Sim | $0.2838_{\pm 0.0016}$ | $0.2578_{\pm 0.0231}$ | $\underline{0.3134}_{\pm 0.0033}$ | $0.2164_{\pm 0.0078}$ | $\mathbf{0.3421}_{\pm 0.0052}$ |
| UNSEEN | BDP | SR | $0.5033_{\pm 0.0031}$ | $0.4707_{\pm 0.0081}$ | $0.5160_{\pm 0.0090}$ | $\mathbf{0.7727}_{\pm 0.0031}$ | $\underline{0.6153}_{\pm 0.0081}$ |
| | | Sim | $\underline{0.5685}_{\pm 0.0064}$ | $0.5539_{\pm 0.0480}$ | $\mathbf{0.6110}_{\pm 0.0080}$ | $0.3743_{\pm 0.0017}$ | $0.4743_{\pm 0.0115}$ |
| | | SR×Sim | $0.2545_{\pm 0.0044}$ | $0.2425_{\pm 0.0084}$ | $\underline{0.2630}_{\pm 0.0060}$ | $0.2615_{\pm 0.0038}$ | $\mathbf{0.2686}_{\pm 0.0043}$ |
| | BDQ | SR | $0.2547_{\pm 0.0042}$ | $0.3207_{\pm 0.0293}$ | $0.3907_{\pm 0.0110}$ | $\underline{0.4400}_{\pm 0.0122}$ | $\mathbf{0.5393}_{\pm 0.0042}$ |
| | | Sim | $\mathbf{0.6751}_{\pm 0.0063}$ | $0.5342_{\pm 0.0274}$ | $\underline{0.6403}_{\pm 0.0025}$ | $0.4880_{\pm 0.0027}$ | $0.4569_{\pm 0.0047}$ |
| | | SR×Sim | $0.1467_{\pm 0.0023}$ | $0.1580_{\pm 0.0175}$ | $\underline{0.2083}_{\pm 0.0043}$ | $0.1872_{\pm 0.0038}$ | $\mathbf{0.2323}_{\pm 0.0054}$ |
| | BPQ | SR | $0.4967_{\pm 0.0145}$ | $0.5127_{\pm 0.0231}$ | $0.5720_{\pm 0.0092}$ | $\mathbf{0.8993}_{\pm 0.0042}$ | $\underline{0.7560}_{\pm 0.0072}$ |
| | | Sim | $\underline{0.5906}_{\pm 0.0034}$ | $0.5496_{\pm 0.0666}$ | $\mathbf{0.6188}_{\pm 0.0054}$ | $0.2546_{\pm 0.0071}$ | $0.4733_{\pm 0.0035}$ |
| | | SR×Sim | $0.2734_{\pm 0.0080}$ | $0.2710_{\pm 0.0268}$ | $\underline{0.3039}_{\pm 0.0064}$ | $0.2109_{\pm 0.0047}$ | $\mathbf{0.3477}_{\pm 0.0046}$ |
| AVG | | SR | $0.4178_{\pm 0.0027}$ | $0.4256_{\pm 0.0175}$ | $0.4883_{\pm 0.0040}$ | $\mathbf{0.7013}_{\pm 0.0052}$ | $\underline{0.6398}_{\pm 0.0022}$ |
| | | Sim | $\underline{0.6126}_{\pm 0.0026}$ | $0.5505_{\pm 0.0328}$ | $\mathbf{0.6281}_{\pm 0.0018}$ | $0.3750_{\pm 0.0027}$ | $0.4649_{\pm 0.0037}$ |
| | | SR×Sim | $0.2273_{\pm 0.0019}$ | $0.2185_{\pm 0.0047}$ | $\underline{0.2601}_{\pm 0.0035}$ | $0.2220_{\pm 0.0033}$ | $\mathbf{0.2824}_{\pm 0.0030}$ |

Table 29: **Generalization to TDC-PairMolGen with Qwen3-1.7B.** Rows report SR, Sim, and per-sample SR×Sim for diffuse single-property redesigns; formatting follows Table 1.

| Task Type | Task | Metric | SFT-SMILES | SFT-Trace | RePO | SFT-SMILES +GRPO | SFT-Trace +GRPO |
|---|---|---|---|---|---|---|---|
| Property Redesign | QED | SR | $\mathbf{0.8627}_{\pm 0.0034}$ | $0.8013_{\pm 0.0159}$ | $\underline{0.8320}_{\pm 0.0071}$ | $0.7780_{\pm 0.0016}$ | $0.8313_{\pm 0.0057}$ |
| | | Sim | $0.4436_{\pm 0.0070}$ | $0.4240_{\pm 0.0030}$ | $0.5159_{\pm 0.0029}$ | $\mathbf{0.7006}_{\pm 0.0017}$ | $\underline{0.6728}_{\pm 0.0024}$ |
| | | SR×Sim | $0.3778_{\pm 0.0050}$ | $0.3326_{\pm 0.0063}$ | $0.4052_{\pm 0.0083}$ | $\underline{0.5475}_{\pm 0.0005}$ | $\mathbf{0.5552}_{\pm 0.0021}$ |
| | DRD2 | SR | $0.8947_{\pm 0.0052}$ | $0.8813_{\pm 0.0148}$ | $\mathbf{0.9453}_{\pm 0.0034}$ | $0.8407_{\pm 0.0034}$ | $\underline{0.9340}_{\pm 0.0043}$ |
| | | Sim | $0.3863_{\pm 0.0017}$ | $0.3496_{\pm 0.0012}$ | $0.3549_{\pm 0.0031}$ | $\mathbf{0.8364}_{\pm 0.0001}$ | $\underline{0.8247}_{\pm 0.0008}$ |
| | | SR×Sim | $0.3433_{\pm 0.0027}$ | $0.3036_{\pm 0.0048}$ | $0.3221_{\pm 0.0007}$ | $\underline{0.6981}_{\pm 0.0026}$ | $\mathbf{0.7728}_{\pm 0.0041}$ |
| | LogP | SR | $\underline{0.9173}_{\pm 0.0034}$ | $0.8620_{\pm 0.0147}$ | $0.8880_{\pm 0.0099}$ | $0.8933_{\pm 0.0019}$ | $\mathbf{0.9427}_{\pm 0.0009}$ |
| | | Sim | $0.4445_{\pm 0.0033}$ | $0.4162_{\pm 0.0053}$ | $0.4641_{\pm 0.0013}$ | $\mathbf{0.8010}_{\pm 0.0003}$ | $\underline{0.7825}_{\pm 0.0003}$ |
| | | SR×Sim | $0.4004_{\pm 0.0018}$ | $0.3562_{\pm 0.0055}$ | $0.3894_{\pm 0.0070}$ | $\underline{0.7165}_{\pm 0.0014}$ | $\mathbf{0.7379}_{\pm 0.0010}$ |
| All Tasks | Avg | SR | $\underline{0.8916}_{\pm 0.0037}$ | $0.8482_{\pm 0.0064}$ | $0.8884_{\pm 0.0031}$ | $0.8373_{\pm 0.0005}$ | $\mathbf{0.9027}_{\pm 0.0014}$ |
| | | Sim | $0.4248_{\pm 0.0025}$ | $0.3966_{\pm 0.0028}$ | $0.4450_{\pm 0.0002}$ | $\mathbf{0.7793}_{\pm 0.0005}$ | $\underline{0.7600}_{\pm 0.0009}$ |
| | | SR×Sim | $0.3738_{\pm 0.0007}$ | $0.3308_{\pm 0.0013}$ | $0.3722_{\pm 0.0014}$ | $\underline{0.6541}_{\pm 0.0005}$ | $\mathbf{0.6886}_{\pm 0.0004}$ |