EVALUATING MULTIPLE LLM GENERATIONS WITH VALIDATED TASK COVERAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

Many LLM applications are most useful when they provide several candidate outputs for comparison, validation, or combination. Predominant evaluation settings, however, still focus on individual outputs or reduce multiple samples to a single success or selected answer. This can miss whether the outputs include several genuinely different useful results. We introduce VTC-Bench, a five-domain benchmark for this setting, together with Validated Task Coverage (VTC) as its core evaluation quantity. The benchmark is built from carefully selected real-data tasks where both output quality and task-relevant distinctness can be checked automatically and reproducibly, without model-based judges. VTC measures how many distinct useful results are obtained within k attempts. Across multiple models and inference settings, the benchmark leads to different conclusions from conventional evaluation: configurations that look strongest from single-draw quality are not necessarily those with the best coverage, and simple measures of output variation do not reliably recover task-relevant coverage. These results show that finite candidate sets can be evaluated directly as objects of interest, revealing differences in model behavior that are not apparent from conventional per-output evaluation.

1 INTRODUCTION

Many LLM applications benefit from generating several candidate outputs rather than committing to a single response. A user may want alternative solutions, hypotheses, designs, or pieces of evidence to compare, validate, or combine before making a decision (Ippolito et al., 2019; Li et al., 2022; Shypula et al., 2025). In such settings, the relevant object is not an isolated generation but the set produced after repeated attempts. The value of that set depends not only on whether individual outputs are good, but also on whether later attempts add new useful possibilities rather than repeat what has already been found.

Most existing evaluation, however, is still organized around the quality of individual outputs. Benchmarks typically measure accuracy, success rate, or another per-attempt score, while multi-sample methods such as pass@k, self-consistency, and best-of-N ultimately reduce several generations to a single success event, aggregate answer, or selected output (Chen et al., 2021; Wang et al., 2023; Huang et al., 2025). These objectives are appropriate when only one final answer matters. They do not measure whether generated answers accumulate several distinct useful outcomes.

We introduce VTC-Bench, a benchmark specifically designed for this finite-candidate-set setting, together with Validated Task Coverage (VTC) as its core evaluation quantity. VTC asks a natural question: after k attempts, how many distinct useful outcomes have been validated? Making this quantity measurable in realistic tasks requires more than repeated generation: each task must define what counts as a valid output and when two valid outputs represent the same useful outcome.

The main challenge is making VTC operational in practical tasks. We therefore select settings where useful outputs can be validated automatically and mapped deterministically to task-specific useful outcomes. VTC-Bench comprises five real-data tasks built around these criteria, with reproducible scoring and no reliance on model-based judges.

With the benchmark in place, we can measure both output quality and task-relevant coverage directly on the same set of tasks. Across multiple models and inference settings, we find that these measurements lead to different conclusions: configurations that look strongest from single-draw quality are

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.

not necessarily those that achieve the best coverage, and simple measures of output variation do not reliably reflect task-specific coverage. Together, our contributions are:

- **VTC-Bench, a five-domain benchmark.** We construct practical tasks where both output validity and task-relevant distinctness can be checked automatically and reproducibly.
- **Validated Task Coverage.** We introduce a common framework for evaluating the distinct useful outcomes produced across repeated generations.
- **A comprehensive empirical study.** Across multiple models, inference settings, and attempt budgets, we show that validated coverage can lead to different model-selection conclusions from single-draw quality and surface-level diversity.

2 Related Work

Repeated-sampling metrics. Pass@k, self-consistency, and best-of-N all evaluate multiple generations, but reduce them to a single success event, aggregate answer, or selected output (Chen et al., 2021; Wang et al., 2023; Huang et al., 2025). These methods are appropriate when only one final answer matters. They do not measure how distinct useful outcomes accumulate across the retained set of generations, which is the object evaluated by VTC.

Distributional metrics. Prior work evaluates quality and diversity jointly by comparing generated and reference distributions, using textual similarity measures or learned representations (Pillutla et al., 2021; Le Bronnec et al., 2024; Alihosseini et al., 2019; Guo et al., 2025). These approaches estimate distributional agreement from samples and generally require a sufficiently representative reference set. Our setting asks a different question: given a fixed task instance and a finite number of attempts, how much useful task progress has actually been accumulated?

Task-specific coverage. Recent work shares our motivation that repeated generations should be evaluated by the distinct useful outcomes they recover. HypoSpace (Chen et al., 2026) is the closest conceptual precedent: it measures recovery of an enumerable valid solution space, which requires exhaustive ground truth and therefore favors deliberately controlled tasks. Shypula et al. (2025) and NoveltyBench (Zhang et al., 2025) study useful diversity in open-ended generation, using program semantics and learned functional-equivalence judgments, respectively. Other studies examine useful diversity for specific applications: PluralEval (Mundada et al., 2026) measures coverage of distinct human viewpoints for subjective questions, while Lee et al. (2025) measure algorithmic diversity among correct generated programs. Both rely on model-based semantic judgments to identify these distinctions. VTC instead formalizes finite-budget coverage: for a fixed k, it measures how many distinct useful outcomes are recovered across k attempts. We instantiate this across heterogeneous practical tasks using automatic validation and deterministic task-specific outcome mappings.

3 Validated Task Coverage

In this section, we define Validated Task Coverage (VTC), our core idea for evaluating finite candidate sets. VTC asks a simple question: after k attempts, how many distinct useful outcomes did a model produce? Each task defines what counts as useful and when two outputs count as the same. For example, in molecule design, the goal may be to generate several chemically distinct molecules satisfying the requested properties. Close variants of the same underlying structure add little new coverage, whereas a new valid structural family increases VTC.

Task attempts. For a given task, let $\mathcal{X}$ denote its set of instances. An instance $x \in \mathcal{X}$ specifies the problem presented to the model together with any task-specific constraints on a valid response. Evaluating a model–inference configuration for k attempts produces a sequence of submitted outputs $y_{1:k} = (y_1, \ldots, y_k)$, where y_i denotes the complete submission on attempt i.

VTC scoring. We score this $y_{1:k}$ sequence in two steps:

- **Validation and mapping** (τ_x). The function τ_x checks whether generated outputs are valid and satisfy the prompt constraints, then maps qualifying outputs to task-specific useful outcomes. Results that represent the same useful outcome map to the same outcome. Thus, $\tau_x(y_i)$ is the set of useful outcomes credited to attempt i, and is empty when nothing earns credit.

- **Coverage function** (u_x). We take the union of the useful outcomes across attempts, so encountering the same outcome again does not increase coverage. The function u_x converts this set into the task's coverage score. Depending on the task, this score counts distinct useful outcomes or reports the fraction of a known target set covered.

$$\textbf{Validated Task Coverage}: \quad \mathrm{VTC}_x(y_{1:k}) = u_x\left(\bigcup_{i=1}^{k} \tau_x(y_i)\right). \tag{1}$$

Expected coverage under independent sampling. Equation 1 scores any realized sequence of k attempts and does not require the attempts to be independent. We next consider the setting where attempts are sampled independently from a fixed model–inference configuration $\pi(y \mid x)$. In this setting, repeated VTC measurements estimate the expected useful coverage obtained after k attempts and connect directly to standard repeated-sampling evaluation such as pass@k. We therefore define

$$C_{\pi,\mathcal{X}}(k) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \mathbb{E}_{y_{1:k} \sim \pi(\cdot|x)^k} \left[\mathrm{VTC}_x(y_{1:k})\right]. \tag{2}$$

The quantity $C_{\pi,\mathcal{X}}(k)$ is the expected coverage at attempt budget k under independent sampling. More generally, evaluating VTC across values of k gives a coverage curve, whether attempts are independent or generated sequentially. The curve increases when additional attempts produce useful outcomes not seen earlier and flattens when they repeat previous outcomes. Configurations with similar coverage at one attempt can therefore have different coverage at larger k.

Coverage estimation under independent sampling. Independent sampling also gives a convenient estimator of expected coverage. Given a larger pool of independent attempts, we can average VTC over all size-k subsets rather than relying on a single set of k draws. This yields the subset-counting estimator described in Appendix A.2, analogous to the standard pass@k estimator. Pass@k is recovered when every successful attempt maps to the same single useful outcome and every unsuccessful attempt maps to the empty set.

4 VTC-Bench: Benchmark Suite

We build five candidate-generation tasks across different domains, focusing on settings where repeated outputs are genuinely useful while retaining fully automatic and reproducible evaluation. Each task is grounded in real data, provides an automatic quality check, and maps valid outputs deterministically to task-relevant useful outcomes. Together, these properties let us evaluate useful coverage directly with task-grounded scoring, without relying on model-based judges. Table 1 summarizes each task's attempt format, validation rule, one-draw quality measure, useful-outcome definition, and VTC calculation. Appendix C discusses further the considered design constraints.

Molecule design.

- *Why repeated candidates matter.* In molecule design, a model is asked to propose molecules that satisfy specified physicochemical constraints. Many structurally different molecules can satisfy the same constraints, so additional attempts are useful when they explore multiple on-spec structural cores rather than returning close variants of one.
- *Construction and attempt.* Following prior work on open molecule generation with LLMs, including TOMG-Bench (Li et al., 2024), we construct 164 prompts that specify one or two physicochemical property ranges. The ranges are derived from ZINC250k (Gómez-Bombarelli et al., 2018) so that each prompt defines a region of molecular space with many known feasible structures. On each attempt, the model returns one molecule as a SMILES string.
- *Validation and credit.* RDKit (RDKit contributors, 2026) parses and sanitizes each generated molecule, checks whether it satisfies the requested property ranges, and extracts its Bemis–Murcko scaffold, a representation of the molecule's core ring-and-linker structure. Only on-spec molecules contribute to coverage, and molecules with the same scaffold count as the same useful outcome. VTC at budget k counts the distinct on-spec scaffolds found across attempts.

Repository repair.

- *Why repeated candidates matter.* When repairing a software vulnerability, there may be several valid ways to modify the repository, targeting different parts of the code or using different im-

plementation strategies. Repeated generation can therefore provide several repair options that an engineer may want to compare before choosing one.

- *Construction and attempt.* We use all 230 runnable instances from PatchEval-Verified, spanning Python, JavaScript, and Go repositories (Wei et al., 2025). On each attempt, the model receives a fresh copy of the full repository, can inspect and edit the source but cannot run the project or tests, and submits one final Git diff.
- *Validation and credit.* Each patch is applied and tested for the target vulnerability in PatchEval-Verified's isolated environment. Passing patches are grouped by the set of named functions they modify, so patches that intervene in the same parts of the code count as the same useful outcome. VTC at budget k counts the distinct passing modified-function sets found across attempts.

Bug finding.

- *Why repeated candidates matter.* In bug finding, the model generates test inputs intended to expose failures in incorrect implementations of a programming problem. A single generated suite may reveal only some of these failures, while additional suites can uncover different faulty behaviors that were missed before.
- *Construction and attempt.* We construct 188 tasks from CodeContests C++ problems with verified accepted and incorrect submissions (Li et al., 2022). Incorrect submissions are grouped into behavioral bug classes according to which benchmark tests they fail, giving 6,422 classes in total. On each attempt, the model receives the problem statement and returns up to five complete standard-input tests.
- *Validation and credit.* A generated input is valid when the verified reference solutions agree on its output. Each valid input is executed against the incorrect submissions and credited with every behavioral bug class it exposes; an attempt contributes the union over its inputs. VTC at budget k is the fraction of the task's fixed bug classes exposed across attempts.

Differential diagnosis.

- *Why repeated candidates matter.* A differential diagnosis lists several plausible explanations for a case. Additional attempts should broaden this list with clinically distinct possibilities, not merely rephrase the same diagnoses.
- *Construction and attempt.* We use 302 NEJM clinicopathological-conference cases from the AMIE differential-diagnosis study (McDuff et al., 2025). Each case provides a reconstructed History of Present Illness and a human-written reference differential. We map the reference diagnoses to a fixed set of UMLS concepts. On each attempt, the model produces a ranked differential of at most seven diagnoses and maps them to UMLS Concept Unique Identifiers (CUIs) using frozen local search tools, without access to the reference concepts.
- *Validation and credit.* Submitted CUIs are checked for existence in the frozen index and provenance in the model's recorded search. Credit is restricted to submitted CUIs in the case's fixed reference concepts. Diagnosis-concept precision is the fraction of submitted concepts in that target; diagnosis-concept recall at budget k is the fraction of the target recovered across attempts.

BRIGHT-Pro evidence search.

- *Why repeated candidates matter.* Answering a complex question often requires several pieces of evidence, each supporting a different part of the answer. Additional search attempts should cover these different needs rather than return several passages containing the same information.
- *Construction and attempt.* We use BRIGHT-Pro (Zhao et al., 2026), where complex questions are annotated with distinct reasoning aspects and supporting gold passages for each aspect. These annotations let us distinguish evidence that supports different parts of an answer from redundant evidence for the same part. On each attempt, the model issues one to three search queries against a frozen domain-specific BM25 index, inspects up to fifteen retrieved passages, and selects one to five passages as evidence.
- *Validation and credit.* Selected passages receive credit when they are annotated gold passages, and each such passage contributes the reasoning aspect it supports. Multiple passages supporting the same aspect therefore do not add new coverage. VTC aggregates the expert-weighted reasoning aspects covered across attempts.

Table 1: **VTC-Bench task definitions.** n is the number of task instances. Each task defines the attempt format, one-draw quality measure, useful outcomes, and headline VTC budget H.

Task	Attempt	Validation and quality	Useful outcome(s)	Set utility (VTC)
Molecule design $n = 164$, $H = 20$	Generate one molecule as a SMILES string.	RDKit parses and sanitizes it; quality asks whether it satisfies every requested property range.	If the molecule is on spec, credit its Bemis–Murcko scaffold (structural core).	Distinct on-spec scaffolds accumulated.
Repository repair $n = 230$, $H = 10$	Edit a full repository and submit one Git diff.	The diff must apply and pass the isolated test for the target vulnerability.	Modified-function set of a passing patch.	Distinct passing modified-function sets accumulated.
Bug finding $n = 188$, $H = 5$	Propose up to five test inputs for a programming problem.	Reference solutions must agree on an input's output; useful-test rate is the fraction of generated inputs that are valid and expose a bug class.	Credit every behavioral bug class exposed by a valid input; take the union within the attempt.	Fraction of fixed bug classes exposed.
Diagnosis $n = 302$, $H = 5$	Produce up to seven ranked diagnoses and ground them to UMLS concepts.	Concept existence is checked; quality is diagnosis-concept precision against the fixed reference target.	Credit the submitted diagnosis concepts that occur in that reference target.	Diagnosis-concept recall.
Evidence search $n = 739$, $H = 10$	Issue one to three queries, inspect at most fifteen passages, and select one to five.	Search and selection calls must be well formed; quality is precision against released gold passages.	Credit the reasoning aspects supported by selected gold passages.	Reasoning-aspect recall.

5 Experimental Protocol

Generation configuration. We evaluate four models on all five tasks: Qwen3.6-27B, Qwen3.6-35B-A3B, GLM-4.7, and DeepSeek-V4-Flash. For each model, we cross temperatures $T \in \{0.6, 1.0, 1.2\}$ with thinking disabled or enabled, yielding 24 model–inference configurations. Appendix A provides checkpoint names and generation details.

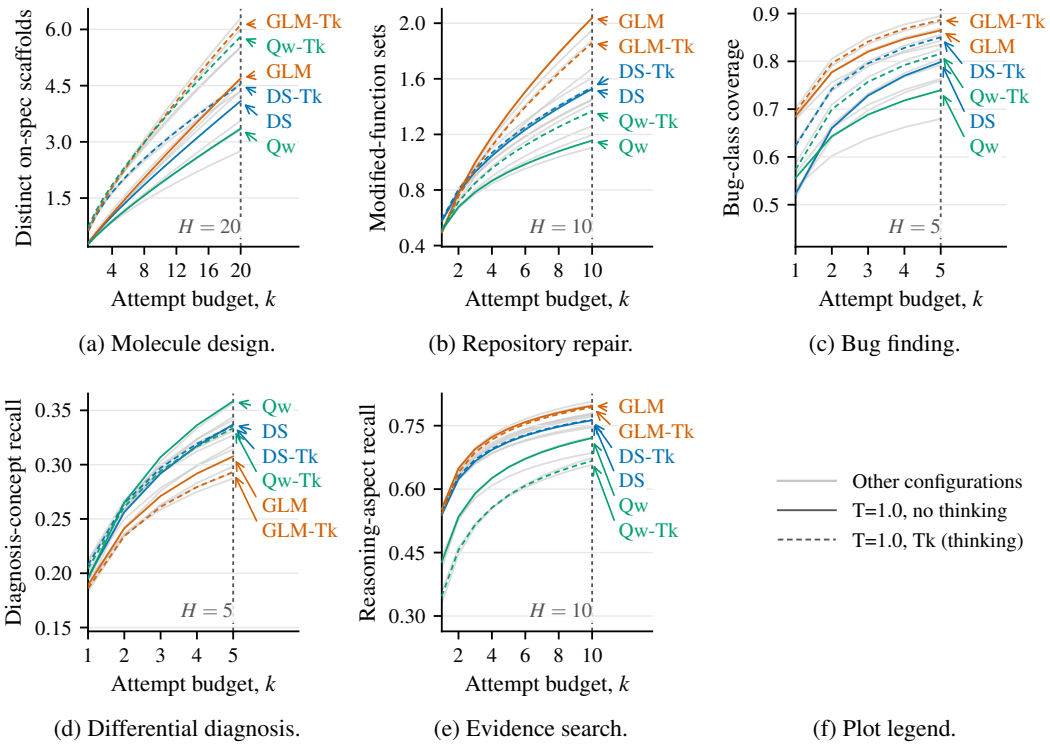
Repeated sampling and budgets. For the main 24-configuration study, we use independent repeated sampling from each model–inference configuration, without conditioning later attempts on earlier outputs. This provides a simple parallel reference setting with a fixed generation distribution. The task-specific H values in Table 1 were chosen to keep candidate sets practically inspectable while controlling generation and validation cost. In Section 6.3, we compare this reference setting with chained generation, conditioning later attempts on earlier outputs while explicitly prompting for complementary candidates. Further experimental details are presented in Appendix A.

6 Results

We first characterize validated coverage across configurations and attempt budgets, then ask whether conventional quality and surface-diversity evaluation identifies high-VTC systems. Finally, we examine how inference choices affect quality and coverage, including temperature, thinking, and chained generation, which conditions later attempts on earlier outputs. Results are reported for the same 24 configurations on every task, with chained generation evaluated as a separate matched comparison. We use a task-specific headline budget H for the primary comparison; Table 1 lists the value for each task.

6.1 Coverage across attempt budgets

Coverage accumulates at different rates. Figure 1 shows how VTC exposes distinct coverage profiles across models and tasks. GLM-4.7 configurations tend to achieve high VTC on four of the five tasks, while DeepSeek-V4-Flash is generally competitive without dominating and Qwen3.6-27B shows more task-dependent performance. These patterns are themselves budget-dependent: some configurations gain most of their coverage at small k, while others continue to add useful outcomes, causing performance gaps to widen or narrow as the budget increases.



(a) Molecule design. (b) Repository repair. (c) Bug finding.

(d) Differential diagnosis. (e) Evidence search. (f) Plot legend.

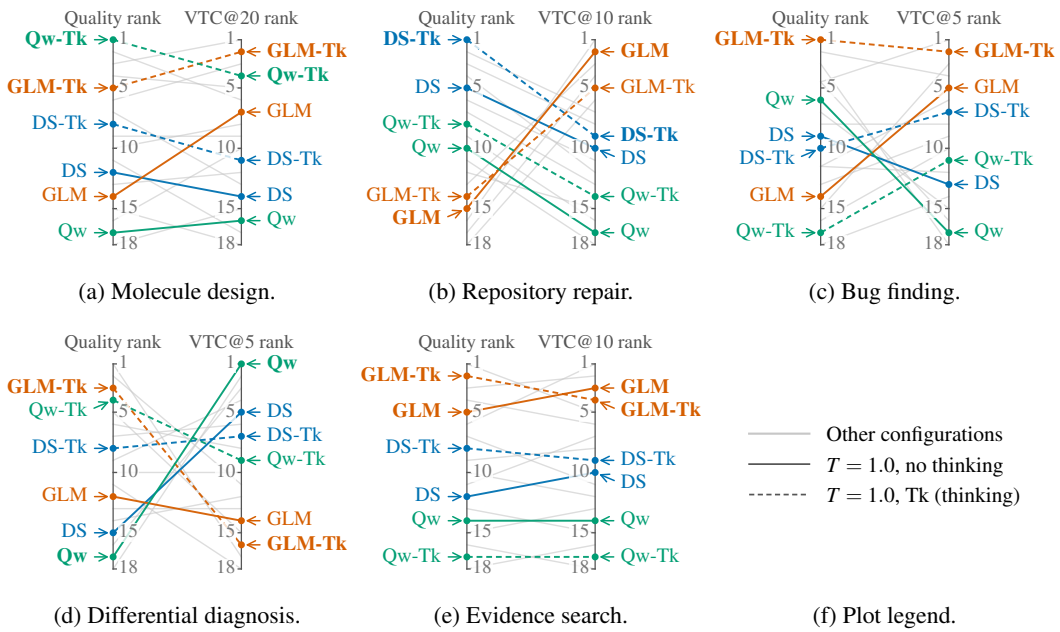
Figure 1: Each panel shows VTC evolution over the headline budget range. Configurations follow different coverage trajectories, producing both rank reversals and changing performance gaps as k increases. Qwen3.6-35B-A3B is omitted for legibility.

Coverage rankings change with the attempt budget. In four of the five tasks, the configuration with the highest VTC at $k = 1$ is not the one with the highest VTC at the headline budget H. The preferred configuration can therefore change as the attempt budget increases. In molecule design, Qwen3.6-27B leads at $k = 1$, but by $H = 20$ GLM-4.7 leads by 0.502 distinct on-spec scaffolds per instance. Repository repair shows a similar pattern: DeepSeek-V4-Flash leads at $k = 1$, whereas by $H = 10$, GLM-4.7 leads by 0.504 distinct passing modified-function sets per instance. Differential diagnosis and evidence search also change leaders, while bug finding retains the same overall leader despite crossings among other configurations. Thus, the preferred configuration depends on the intended attempt budget, not only on performance at a single generation. Appendix D.1 summarizes the consequences of selection and the VTC changes across attempt budgets. All four leader changes remain under paired instance resampling; details in Appendix D.5.

6.2 Quality, Diversity, and Coverage

Models and inference configurations are commonly compared using individual-output quality or generic measures of variation across multiple outputs. We ask whether these signals identify the same high-VTC configurations at a headline budget H. To quantify this, we rank configurations by a selection metric and report the relative VTC regret of its top-ranked configuration, as summarized in Table 2. The regret of configuration π is $\frac{C^*(H)-C_\pi(H)}{C^*(H)}$, with $C^*(H) = \max_\pi C_\pi(H)$.

One-draw quality does not reliably identify high-VTC configurations. Models are predominantly compared and selected using quality-focused benchmarks, so a model judged "stronger" is typically one that performs better on individual attempts. Figure 2 shows that these rankings can differ substantially from rankings by VTC at the headline budget H. The disagreement extends beyond the top-ranked configuration: configurations move throughout the ranking in every task. The configuration ranked highest by one-draw quality incurs VTC regret ranging from 0.9% in bug finding to 24.7% in repository repair, with an 11.1% task-macro average. Quality Top-5 sets share only 10 of 25 positions with the VTC Top-5 sets. Thus, high one-draw quality does not reliably identify the configurations with the highest finite-budget VTC.



(a) Molecule design. (b) Repository repair. (c) Bug finding.

(d) Differential diagnosis. (e) Evidence search. (f) Plot legend.

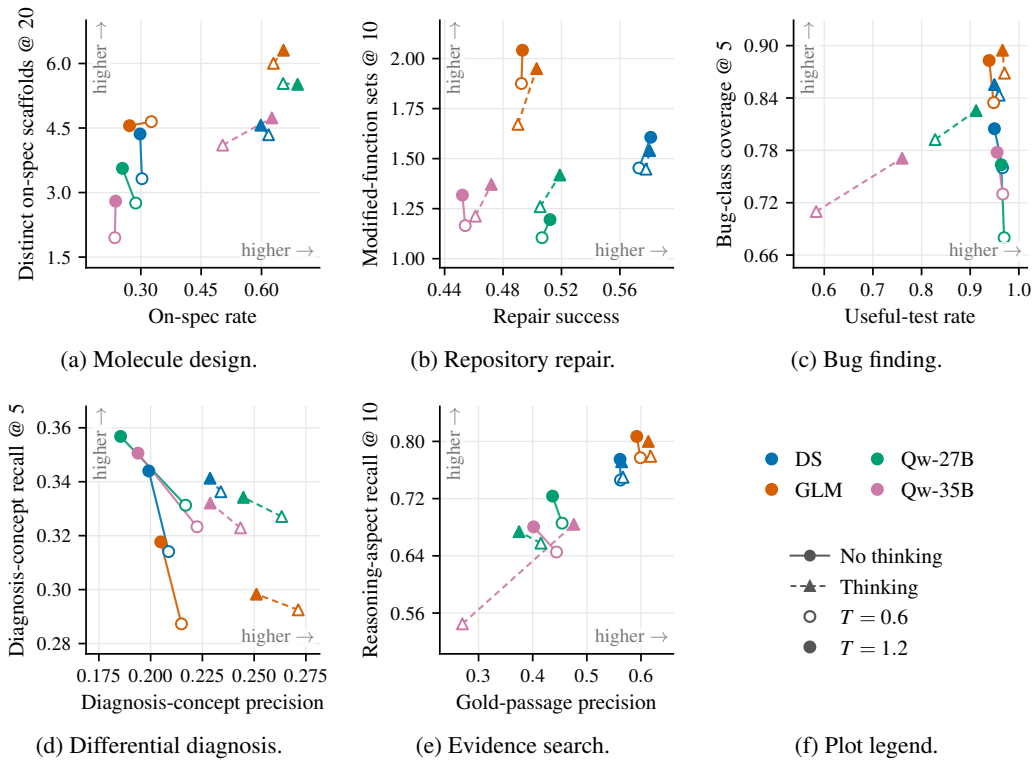
Figure 2: **Quality and VTC rank differently.** Each panel compares the displayed configurations ranked by one-draw quality (left) and VTC at the headline budget H (right). The best $T = 1.0$ configuration under either metric is bolded. Lines show how configurations move between the two rankings. Qwen3.6-35B-A3B is omitted for legibility.

More distinct outputs do not necessarily mean more useful coverage. A common way to evaluate repeated generation is to measure variation among the generated outputs themselves, using lightweight metrics such as exact match or textual similarity. Such measures capture surface-level variation, but differences in form need not correspond to distinct useful outcomes. VTC instead maps validated outputs to task-specific useful outcomes. We therefore compare VTC with a budget-matched surface-diversity measure for each task, defined at a finer output level than the corresponding useful-outcome mapping (e.g., molecules rather than scaffolds, or test inputs rather than bug classes). Appendix D.4 defines the task-specific measures and reports the complete results.

Surface diversity does not consistently identify the configurations with the highest VTC either. As shown in Table 2, across the five tasks, surface-diversity selection has 9.7% task-macro Top-1 VTC regret and 11/25 total Top-5 agreement with VTC. The mismatch is task-dependent: Top-1 regret ranges from zero in bug finding to 21.3% in repository repair. Thus, directly measuring non-repetition is not sufficient to determine whether repeated outputs contribute new useful outcomes. The selection mismatch is stable under paired instance resampling (Appendix D.5).

Table 2: **Quality and surface diversity can misselect for VTC.** Top-1 regret is the relative VTC@H shortfall of each metric's top-ranked configuration from the VTC leader; agreement is the overlap of their Top-5 sets. Surface diversity is defined at a finer output level than VTC.

	One-draw quality		Surface diversity	
Task	**Top-1 regret**	**Top-5 agreement**	**Top-1 regret**	**Top-5 agreement**
Molecule design	8.0%	4/5	8.0%	4/5
Repository repair	24.7%	0/5	21.3%	0/5
Bug finding	0.9%	2/5	0.0%	3/5
Differential diagnosis	18.4%	0/5	4.0%	4/5
Evidence search	3.5%	4/5	15.2%	0/5
Task macro-average / total	11.1%	10/25	9.7%	11/25



(a) Molecule design. (b) Repository repair. (c) Bug finding.

(d) Differential diagnosis. (e) Evidence search. (f) Plot legend.

Figure 3: **Temperature and thinking affect quality and VTC differently.** Each panel places one-draw quality on the horizontal axis and VTC@H on the vertical axis, with segments showing the effect of increasing temperature from $T = 0.6$ to $T = 1.2$ for matched model–thinking configurations. Higher temperature increases VTC@H in 38 of 40 comparisons, often despite lower one-draw quality. Thinking introduces additional variation in both quality and coverage, with effects that differ across tasks.

6.3 Effects of inference choices on VTC

We next examine inference choices that may change how useful outcomes accumulate across attempts. We consider temperature and thinking within the independent-sampling setting, then ask whether explicitly conditioning later attempts on earlier outputs can improve coverage.

Temperature can improve coverage while reducing one-draw quality. Figure 3 shows that temperature can affect one-draw quality and validated coverage in opposite directions. Increasing temperature from 0.6 to 1.2 raises the point-estimate VTC@H in 38 of 40 matched comparisons; in 25 of these, one-draw quality decreases. Paired instance-resampling intervals are entirely above zero for 36 of the 40 comparisons (Appendix D.5). Thus, an inference change that looks worse under conventional quality-focused evaluation can still produce a more useful candidate set with higher validated coverage.

Thinking has heterogeneous effects on quality and coverage. Thinking is generally intended to improve the quality of an individual response, but this need not translate into higher finite-budget coverage. Across the 60 matched comparisons over all three temperatures, thinking improves one-draw quality in 44 cases but VTC@H in 36. Its benefits are therefore less consistent for coverage than for individual-output quality, and the two measures move in opposite directions in 23 comparisons. The pattern is also heterogeneous across tasks: molecule design generally benefits from thinking, whereas evidence search often does not.

Explicit diversity prompting does not reliably increase useful coverage. We also evaluate a chained generation strategy explicitly designed to promote diversity: each attempt is conditioned

on earlier outputs and asked to avoid repeating previous candidates. VTC shows that this does not reliably translate into more distinct useful outcomes. Chaining consistently improves coverage on evidence search, consistently reduces it on repository repair, reduces it in most differential-diagnosis configurations, has little effect on bug finding, and shows larger configuration-dependent effects on molecule design. Thus, even when diversity is explicitly encouraged at generation time, the resulting candidate set need not cover more validated task-relevant outcomes. We evaluate this comparison at $H_{\text{chain}} = 20$ for molecule design and $H_{\text{chain}} = 5$ for the other four tasks, with matched independent controls at the same budgets. Further details and analysis appear in Appendix B.

Table 3: **Effect of chained generation on VTC.** Relative ΔVTC compares chained generation with independent generation at the chained-experiment budget H_{chain} (20 for molecule design and 5 otherwise). Mean, minimum, and maximum changes are computed across eight matched model–inference configurations; "Positive" counts configurations for which chaining increases VTC.

Task	Mean relative ΔVTC	Min	Max	Positive
Molecule design	+16.0%	-22.0%	+46.5%	6/8
Repository repair	-11.5%	-30.2%	-0.9%	0/8
Bug finding	+0.7%	-2.9%	+5.4%	4/8
Differential diagnosis	-7.9%	-15.4%	+0.6%	1/8
Evidence search	+8.3%	+3.9%	+13.6%	8/8

7 Discussion and Limitations

VTC depends on task-specific outcome mappings. VTC depends on task-specific choices about which outputs are valid and which valid outputs count as the same useful outcome. We deliberately use lightweight, deterministic mappings so that coverage remains automatic and reproducible, but these mappings abstract away some distinctions that could matter in practice. For example, modified-function sets capture where a patch intervenes rather than whether two patches implement genuinely different repair strategies; diagnosis is evaluated against a fixed UMLS target; and evidence coverage depends on the available passage and aspect annotations. Differential diagnosis and BRIGHT-Pro additionally rely on fixed, human-curated reference targets that may omit clinically plausible diagnoses or relevant evidence. Freezing these targets across configurations ensures consistent comparison, but makes absolute coverage reference-relative: valid outcomes outside the annotated target receive no credit.

Beyond fixed attempt budgets. We compare configurations at a fixed number of attempts within each task, providing a simple and controlled budget axis for repeated generation. Attempt counts are not directly comparable across tasks in terms of token use, compute, or human review effort. The same framework could instead measure coverage against these resource budgets when they can be tracked consistently, addressing different practical questions from the attempt-budget analysis studied here.

Inference choices for coverage. Our experiments show that inference choices can affect validated coverage in different ways: higher temperature often helps, while thinking and chained diversity prompting have more task-dependent effects. This leaves open how to choose an inference strategy that makes the best use of a fixed generation budget for a given task, a question that VTC-Bench is designed to make directly measurable.

8 Conclusion

VTC-Bench makes finite sets of LLM generations directly evaluable across five tasks. By mapping validated outputs to task-relevant useful outcomes, VTC tracks how these outcomes accumulate as the attempt budget grows. The results show different coverage trajectories, budget-dependent rankings, and task-dependent effects of inference choices. More broadly, our results show that finite candidate sets exhibit measurable behavior of their own, making them a meaningful object of model evaluation beyond individual generations.

AI USE STATEMENT

We used generative AI tools throughout the research workflow, including to assist with software development, refine and assess the feasibility of research ideas, provide feedback on experimental and benchmark design, and support literature search and review, particularly during the construction of the benchmark tasks. We also used generative AI tools to assist with drafting and editing parts of the manuscript for clarity and presentation. All AI-assisted outputs were reviewed by the authors, and research decisions, experimental results, citations, and claims were independently checked against the underlying code, data, and source literature. The authors take full responsibility for the final content of the paper and all associated artifacts.

ETHICS STATEMENT

This work uses existing benchmark and published data and does not involve new data collection from human participants. The differential-diagnosis benchmark is constructed from previously published clinical case reports and existing AMIE-derived annotations; restricted clinical inputs and diagnosis targets are not redistributed. The repository-repair benchmark uses real software vulnerability instances, but evaluation is performed in isolated environments without network access. We use these resources solely for controlled evaluation of model behavior and do not release new sensitive clinical information or exploit code beyond the benchmark artifacts provided by the original sources.

REPRODUCIBILITY STATEMENT

Appendix A documents the evaluated model checkpoints, inference settings, attempt budgets, software environment, and statistical aggregation procedure. Appendix B documents the matched chained-generation protocol and task-specific chaining procedures, while Appendix C describes the construction and deterministic scoring of each benchmark task. The later appendices report complete configuration-level results and robustness analyses. The supplementary material provides the frozen redistributable benchmark inputs, prompts, data provenance, and code for running the experiments and reproducing the analyses.

REFERENCES

Danial Alihosseini, Ehsan Montahaei, and Mahdieh Soleymani Baghshah. Jointly measuring diversity and quality in text generation models. In *Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation*, pp. 90–98, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-2311. URL https://aclanthology.org/W19-2311/.

G. W. Bemis and M. A. Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of Medicinal Chemistry*, 39(15):2887–2893, 1996. doi: 10.1021/jm9602928. URL https://doi.org/10.1021/jm9602928.

Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 362–367. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-3346. URL https://aclanthology.org/W14-3346/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Tingting Chen, Beibei Lin, Zifeng Yuan, Qiran Zou, Hongyu He, Anirudh Goyal, Yew-Soon Ong, and Dianbo Liu. Hypospace: A diagnostic benchmark for set-valued hypothesis generation under underdetermination and sublinear coverage bounds. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QpjtK65JHO.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. doi: 10.1021/acscentsci.7b00572. URL https://doi.org/10.1021/acscentsci.7b00572.

Yanzhu Guo, Guokan Shang, and Chloé Clavel. Benchmarking linguistic diversity of large language models. *Transactions of the Association for Computational Linguistics*, 13:1507–1526, 2025. doi: 10.1162/tacl.a.47. URL https://aclanthology.org/2025.tacl-1.69/.

Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J. Foster. Is best-of-N the best of them? coverage, scaling, and optimality in inference-time alignment. In *International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=QnjfkhrbYK.

Daphne Ippolito, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3752–3762, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1365. URL https://aclanthology.org/P19-1365/.

Florian Le Bronnec, Alexandre Verine, Benjamin Negrevergne, Yann Chevaleyre, and Alexandre Allauzen. Exploring precision and recall to assess the quality and diversity of LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11418–11441, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.616. URL https://aclanthology.org/2024.acl-long.616/.

Seonghyeon Lee, HeeJae Chon, Joonwon Jang, Dongha Lee, and Hwanjo Yu. How diversely can language models solve problems? exploring the algorithmic diversity of model-generated code. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 152–167, 2025. doi: 10.18653/v1/2025.findings-emnlp.10. URL https://aclanthology.org/2025.findings-emnlp.10/.

Jiatong Li, Junxian Li, Yunqing Liu, Dongzhan Zhou, and Qing Li. TOMG-Bench: Evaluating LLMs on text-based open molecule generation. *arXiv preprint arXiv:2412.14642*, 2024. doi: 10.48550/arXiv.2412.14642. URL https://arxiv.org/abs/2412.14642.

Yujia Li et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://doi.org/10.1126/science.abq1158.

Daniel McDuff, Mike Schaekermann, Tao Tu, et al. Towards accurate differential diagnosis with large language models. *Nature*, 642:451–457, 2025. doi: 10.1038/s41586-025-08869-4. URL https://doi.org/10.1038/s41586-025-08869-4.

Gagan Mundada, Rohan Surana, Nandhini Swaminathan, Bodhisattwa Prasad Majumder, Junda Wu, Julian McAuley, and Zhouhang Xie. Evaluating language model pluralism through in-the-wild crowd discussions. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 42273–42296, 2026. doi: 10.18653/v1/2026.acl-long.1957. URL https://aclanthology.org/2026.acl-long.1957/.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems*, volume 34, pp. 4816–4828, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/260c2432a0eecc28ce03c10dadc078a4-Abstract.html.

RDKit contributors. RDKit: Open-source cheminformatics. Version 2026.03.3, 2026. URL https://www.rdkit.org/.

Alexander Shypula, Shuo Li, Botong Zhang, Vishakh Padmakumar, Kayo Yin, and Osbert Bastani. Evaluating the diversity and quality of LLM generated content. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=O7bF6nlSOD.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Zichao Wei, Jun Zeng, Ming Wen, Zeliang Yu, Kai Cheng, Yiding Zhu, Jingyi Guo, Shiqi Zhou, Le Yin, Xiaodong Su, and Zhechao Ma. PATCHEVAL: A new benchmark for evaluating LLMs on patching real-world vulnerabilities, 2025. URL https://arxiv.org/abs/2511.11019.

Yiming Zhang, Harshita Diddee, Susan Holm, Hanchen Liu, Xinyue Liu, Vinay Samuel, Barry Wang, and Daphne Ippolito. NoveltyBench: Evaluating language models for humanlike diversity. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=XZm1ekzERf.

Yilun Zhao, Jinbiao Wei, Tingyu Song, Siyue Zhang, Chen Zhao, and Arman Cohan. Rethinking reasoning-intensive retrieval: Evaluating and advancing retrievers in agentic search systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36776–36806, 2026. doi: 10.18653/v1/2026.acl-long.1705. URL https://aclanthology.org/2026.acl-long.1705/.

A Experimental and Reproducibility Details

A.1 Evaluated generation settings

Table 4 records the settings used for the five tasks. Every draw uses $p = 0.95$ and one of the three temperatures in the main study. The evaluated checkpoints are Qwen3.6-27B-FP8, Qwen3.6-35B-A3B-FP8, GLM-4.7-FP8, and DeepSeek-V4-Flash. Thinking is controlled through each model family's chat template, with the checkpoint and task prompt held fixed. The output cap includes hidden reasoning when thinking is enabled. PatchEval uses the same cap in both modes because its agent trajectory contains multiple model calls. In differential diagnosis, the generation and concept-coding stages use the same temperature, thinking setting, and output cap. In BRIGHT-Pro, the search and selection turns use the same sampled configuration and output cap.

Table 4: **Task-level generation settings.** H is the headline comparison budget; complete curves are retained through $K_{\max}$.

Task	Instances	$K_{\max}$	H	No-think / think cap	Additional limit
Molecule	164	50	20	256 / 32,768	one SMILES
Repository repair	230	20	10	16,384 / 16,384	120 agent messages
Bug finding	188	20	5	8,192 / 32,768	at most 5 inputs
Differential diagnosis	302	20	5	2,048 / 32,768	7 diagnoses
BRIGHT-Pro	739	20	10	2,048 / 32,768	3 queries; 5 passages

A.2 Independent-sampling estimator and statistical aggregation

For each task result, all unreduced draw-level scores are grouped by stable task-instance identifiers before computing coverage curves, single-draw diagnostics, and diversity statistics. Every draw enters aggregation regardless of its outcome. The analysis restricts each task to the same set of 24 configurations and applies the task-specific budget H in Table 4. The same aggregation procedure is applied independently within each task.

We use the subset-counting construction of the unbiased pass@k estimator introduced in the Codex evaluation of Chen et al. (2021), applying it separately to each useful outcome and summing the resulting inclusion probabilities with task-specific weights. Under IID sampling from a fixed configuration $\pi(\cdot \mid x)$, this gives an unbiased estimator of the expected VTC obtained from k fresh draws for instance x. For an instance x with K_x scored attempts, let $n_{x,t}$ be the number of attempts containing useful outcome t, and let $w_{x,t}$ be that outcome's task-specific utility weight. For every $k \le K_x$, we compute

$$\widehat{C}_x(k) = \sum_{t \in \mathcal{T}_x} w_{x,t} \left[1 - \frac{\binom{K_x - n_{x,t}}{k}}{\binom{K_x}{k}} \right]. \tag{3}$$

The bracketed term is the fraction of size-k subsets of the observed attempts that contain useful outcome t, so Equation 3 is the exact average coverage over all such subsets. We average $\widehat{C}_x(k)$ equally across the task instances. As with pass@k estimation, collecting $K_{\max} > H$ reduces the sampling variance of the estimated coverage for $k \le H$.

A.3 Software, environment, and data provenance

Software and environment. The runs use Inspect AI 0.3.241, with models served locally using vLLM 0.21.0. The evaluated checkpoints were obtained from the public Hugging Face repositories `Qwen/Qwen3.6-27B-FP8`, `Qwen/Qwen3.6-35B-A3B-FP8`, `zai-org/GLM-4.7-FP8`, and `deepseek-ai/DeepSeek-V4-Flash`. Generation is stochastic, so independent reruns draw new completions from the same sampling distribution. Surface SelfBLEU is computed with NLTK 3.10.0.

Data availability. Redistributable frozen benchmark inputs and prompt templates accompany the paper. Upstream release or revision information is specified when available. The restricted AMIE HPI input, diagnosis-concept target, and term-level coding records are not redistributed.

B Chained Generation Experiments

B.1 Overall setting and analysis

Chained generation setting. We compare chained generation against a matched IID baseline with the same total number of generated candidates per instance. The chained experiment uses $H_{\text{chain}} = 20$ for molecule design and $H_{\text{chain}} = 5$ for the other four tasks. Repository repair and evidence search therefore use a smaller budget than in the main study to keep the additional chained generation and evaluation tractable. For each instance, we generate $R = 3$ independent chains and average VTC over their length-k prefixes:

$$C_{\text{chain},\mathcal{X}}(k) = \frac{1}{|\mathcal{X}|R} \sum_{x \in \mathcal{X}} \sum_{r=1}^{R} \text{VTC}_x\left(y_{1:k}^{(r)}\right), \qquad R = 3. \tag{4}$$

Because later attempts condition on earlier outputs, arbitrary subsets of a chain are not valid length-k chained runs; we therefore score the actual prefix of each chain. The matched IID baseline consists of $H_{\text{chain}} \times R$ independent attempts per instance and is evaluated with the IID subset estimator from Appendix A.2. Because it uses a separately generated matched control pool, its estimates may differ slightly from the corresponding main-study results.

Chained generation analysis. We further examine whether the effect of chaining on VTC is reflected by conventional per-attempt quality or surface-level diversity. For each model–thinking configuration at $T = 1.0$, we use the same task-specific quality and surface-diversity measures as in the main analysis. BRIGHT-Pro is included in the VTC and quality comparisons but omitted from the surface-diversity comparison because its paper surface proxy requires a separate retrieval-conditioned answer-generation stage, which was not run for the chained campaign. Tables 5–7 report VTC, per-attempt quality, and surface diversity, respectively.

Table 5: **VTC under chained and independent generation.** Chain reports VTC from the actual length-H_{chain} chain prefixes; IID reports the subset-counting expectation at the same budget from the matched independent controls. Darker shading indicates higher values within each column; bold marks the largest value.

Configuration			Molecule design		Repository repair		Bug finding		Diagnosis		Evidence search	
Model	**Thinking**	T	Chain	IID	Chain	IID	Chain	IID	Chain	IID	Chain	IID
DS-V4	No	1.0	4.715	4.130	1.019	1.183	0.775	0.794	**0.321**	0.334	0.747	0.713
DS-V4	Yes	1.0	6.435	4.394	1.165	1.176	0.842	0.845	0.308	0.340	**0.787**	0.712
GLM-4.7	No	1.0	4.929	4.753	**1.309**	**1.367**	0.839	0.864	0.307	0.305	0.766	0.738
GLM-4.7	Yes	1.0	**8.827**	**6.047**	1.116	1.205	**0.901**	**0.884**	0.286	0.298	0.773	**0.740**
Qw3.6-27B	No	1.0	2.667	3.419	0.874	0.928	0.752	0.736	0.308	**0.354**	0.724	0.652
Qw3.6-27B	Yes	1.0	7.785	5.863	1.007	1.035	0.842	0.812	0.291	0.333	0.678	0.597
Qw3.6-35B	No	1.0	2.148	2.568	0.690	0.948	0.747	0.762	0.291	0.344	0.648	0.588
Qw3.6-35B	Yes	1.0	5.746	4.657	0.670	0.959	0.806	0.765	0.311	0.331	0.642	0.592

The three quantities often move differently under chaining. On evidence search, chaining increases VTC in all 8 configurations despite reducing per-attempt quality in 6/8, indicating a tradeoff between passage precision and reasoning-aspect coverage. On bug finding, surface diversity decreases in all 8 configurations, by 32.6% on average, while VTC changes little overall, showing that substantially fewer distinct concrete tests need not imply lower behavioral bug-class coverage. Repository repair shows a different pattern: VTC decreases in all 8 configurations even though quality and surface diversity change comparatively little. Differential diagnosis is the clearest case where the metrics agree, with chaining generally reducing quality, surface diversity, and VTC together.

Molecule design is more configuration-dependent. Chaining often increases both surface diversity and VTC, particularly in several thinking configurations, but some Qwen configurations show the opposite behavior. This variation reinforces that neither the intended purpose of the chaining intervention nor its effect on surface-level output variation is sufficient to predict the resulting useful coverage.

Overall, chained generation reproduces the main paper's quality–diversity–coverage mismatch under an explicit diversity-oriented intervention: changes in per-attempt quality and output-level variation do not reliably determine changes in validated task coverage.

Table 6: **Quality under chained and independent generation.** Chain is task-native quality averaged over all positions in the three chains; IID is quality over the attempt-count-matched independent control. Darker shading indicates higher values within each column; bold marks the largest value.

Configuration			Molecule design		Repository repair		Bug finding		Diagnosis		Evidence search	
Model	**Thinking**	T	Chain	IID	Chain	IID	Chain	IID	Chain	IID	Chain	IID
DS-V4	No	1.0	0.331	0.302	0.554	0.573	0.943	0.957	0.090	0.204	0.422	0.560
DS-V4	Yes	1.0	0.606	0.606	**0.581**	**0.596**	0.935	0.955	0.104	0.233	0.513	0.571
GLM-4.7	No	1.0	0.282	0.294	0.451	0.488	0.920	0.943	0.133	0.209	0.519	0.599
GLM-4.7	Yes	1.0	0.582	0.635	0.455	0.497	0.912	0.940	0.127	**0.264**	0.478	**0.622**
Qw3.6-27B	No	1.0	0.255	0.270	0.502	0.504	**0.949**	**0.967**	0.135	0.195	**0.536**	0.448
Qw3.6-27B	Yes	1.0	0.657	**0.726**	0.529	0.518	0.904	0.891	0.116	0.252	0.437	0.399
Qw3.6-35B	No	1.0	0.239	0.242	0.482	0.448	0.947	0.964	**0.148**	0.203	0.344	0.421
Qw3.6-35B	Yes	1.0	**0.696**	0.608	0.452	0.451	0.867	0.732	0.131	0.235	0.461	0.476

Table 7: **Surface diversity under chained and independent generation.** Chain reports direct-prefix coverage of validated canonical molecules, accepted exact diffs, exact valid test inputs, and gold-mapped raw diagnosis terms; IID reports the corresponding subset-counting expectation at H_{chain}. BRIGHT-Pro is omitted because its separate answer-generation surface experiment was not run for chained generation. Bold marks the largest value in each column.

Configuration			Molecule design		Repository repair		Bug finding		Diagnosis	
Model	**Thinking**	T	Chain	IID	Chain	IID	Chain	IID	Chain	IID
DS-V4	No	1.0	6.470	5.442	2.512	2.611	7.172	12.566	**3.139**	**4.804**
DS-V4	Yes	1.0	12.061	9.351	**2.655**	**2.654**	12.379	17.342	2.932	4.248
GLM-4.7	No	1.0	5.602	5.831	2.246	2.347	9.819	16.637	3.024	3.491
GLM-4.7	Yes	1.0	11.624	10.519	2.251	2.318	**16.238**	**19.067**	2.402	3.170
Qw3.6-27B	No	1.0	4.459	4.522	2.184	2.099	9.417	13.627	3.043	4.258
Qw3.6-27B	Yes	1.0	13.067	**11.393**	2.365	2.247	11.387	17.267	2.512	3.302
Qw3.6-35B	No	1.0	4.425	4.371	1.920	2.035	7.851	13.199	2.882	4.242
Qw3.6-35B	Yes	1.0	**13.654**	9.015	1.746	2.037	10.512	14.557	3.011	3.353

B.2 Task-specific chaining procedures

Across tasks, the first attempt uses the original task prompt. Each later attempt receives a bounded record of the model's earlier submissions and a task-specific request for one additional candidate. The history contains neither validation results nor correctness feedback, and each of the three chain replicates starts without history from the others.

Molecule design. The model sees the SMILES strings submitted on earlier attempts and is asked for one additional molecule that satisfies the original constraints while being materially different from its prior submissions. Each attempt retains the original one-SMILES output format.

Repository repair. The model sees the patches submitted earlier in the chain, but not their validation results. The repository is reset to the original vulnerable snapshot before every attempt, and the model is asked to produce one materially different patch using the same editing interface as in the main study.

Bug finding. Earlier test suites are shown in the same input format used for submission. The continuation asks for another suite based on a materially different testing idea, subject to the original limit of at most five inputs. Outcomes from executing earlier tests are not shown to the model.

Differential diagnosis. The model sees the diagnosis terms from its earlier ranked differentials and is asked for another substantively different but clinically plausible differential in the original list format. The UMLS mapping process and matches against the reference differential are not included in the history.

Evidence search. The model sees the queries and selected evidence excerpts from earlier attempts and is asked to use complementary queries and evidence for the same question. Each attempt retains the original limits of up to three queries and five selected passages; unselected retrieval results and coverage scores are not carried into later attempts.

C Benchmark Construction and Scoring

C.1 Design criteria for useful outcomes

For each task, we choose a useful-outcome representation that can be checked automatically, is reproducible across systems, and captures distinctions that matter for the task. Repeatedly producing the same outcome should add little or no new coverage.

Exact output identity is often too fine-grained, while semantic grouping with embeddings or judges would make scoring less deterministic. We therefore use task-specific intermediate representations grounded in the structure of each task. Table 8 summarizes these choices.

Table 8: **Task-specific useful outcomes.** Each representation abstracts away surface variation while remaining deterministic, automatically scoreable, and grounded in task structure.

Task	Too-fine representation	Requires semantic judgment	Useful outcome	Rationale
Molecule design	Exact molecule	Functional or semantic chemical similarity	Bemis–Murcko scaffold	Collapses peripheral substituent and encoding variation while retaining the molecular structural core.
Repository repair	Exact Git diff	Semantic repair strategy	Modified-function set	Collapses incidental diff-level variation while distinguishing repairs that intervene in different parts of the implementation.
Bug finding	Exact test input	Underlying bug mechanism	Behavioral bug class	Different concrete inputs may expose the same faulty behavior; execution against the fixed submission pool gives a deterministic behavioral grouping.
Diagnosis	Diagnosis string	Unrestricted clinical equivalence	UMLS concept	Collapses synonyms, abbreviations, and paraphrases that refer to the same clinical concept while retaining concept-level distinctions.
Evidence search	Passage ID	Free-form semantic contribution	Reasoning aspect	Multiple passages may support the same information need; aspect annotations capture complementary pieces of evidence without requiring semantic judgment of full answers.

C.2 Molecule design

Choice of useful-outcome representation. We represent useful outcomes by Bemis–Murcko scaffolds, extracted automatically with RDKit. Exact molecule identity is too fine-grained for our purpose: molecules can differ in peripheral substituents while sharing the same underlying structural core, so counting them separately would reward variation that contributes little new structural coverage. Bemis–Murcko scaffolds provide a deterministic intermediate abstraction that retains ring systems and the linkers between them while removing peripheral substituents (Bemis & Murcko, 1996). Molecules with the same scaffold therefore map to the same useful outcome even when their SMILES encodings or peripheral substituents differ. We use the number of distinct on-spec scaffolds accumulated across attempts as a reproducible proxy for structural exploration under explicit physicochemical constraints.

Instance construction. We construct the benchmark from ZINC250k, a standard molecular-generation dataset of approximately 250,000 drug-like, commercially available molecules sampled from the ZINC database (Gómez-Bombarelli et al., 2018). We parse all entries with RDKit

2026.03.3, retaining 249,455 valid molecules. For each molecule, we record its canonical SMILES, Bemis–Murcko scaffold, and six conditioning descriptors: molecular weight, RDKit-calculated logP, topological polar surface area (TPSA), hydrogen-bond donors, hydrogen-bond acceptors, and rotatable bonds. QED is excluded because it is a composite desirability score rather than a primitive property constraint.

For each descriptor, we form the five 30-percentile-wide windows $[Q_{.05}, Q_{.35}]$, $[Q_{.20}, Q_{.50}]$, $[Q_{.35}, Q_{.65}]$, $[Q_{.50}, Q_{.80}]$, and $[Q_{.65}, Q_{.95}]$. Adjacent windows overlap by half their width, and their union covers the central 90% of the marginal distribution. We round endpoints to multiples of 10 Da for molecular weight, 0.5 for logP, 5 Å^2 for TPSA, and one count for the integer descriptors. For the integer-valued descriptors, rounding can cause two quantile windows to produce the same interval. When this occurs, we keep only one copy of that interval. We enumerate every single-descriptor band and every combination of bands over descriptor pairs. This procedure produces 379 candidate specifications.

To avoid instances whose feasible region is too narrow in the reference pool, we require each specification to be satisfied by at least 5,000 ZINC250k molecules and at least 1,000 distinct Bemis–Murcko scaffolds. All 379 candidates pass this prespecified, model-independent filter, yielding 29 one-constraint and 350 two-constraint prompts.

For the paper evaluation, we retain all 29 one-constraint specifications. For each two-descriptor pair, we retain only combinations formed from the first, middle, and last window of each descriptor, giving $3 \times 3 = 9$ prompts per descriptor pair. This yields 135 two-constraint prompts and 164 prompts overall.

The released prompt artifact records the exact resolved bands, constraints, support counts, ZINC250k source identity, and RDKit version.

Scoring. At scoring time, we recompute the same RDKit descriptors used during construction and apply all interval bounds inclusively. Valid on-spec molecules are then mapped to their Bemis–Murcko scaffold.

C.3 Repository repair

Benchmark setup. We use all 230 runnable instances from PatchEval-Verified, spanning Python, JavaScript, and Go repositories (Wei et al., 2025). Each instance provides a vulnerable repository snapshot and an isolated environment for validating candidate patches. The task description includes the CVE identifier and description together with its CWE category. PatchEval-Verified also provides annotated vulnerable files and line ranges; we withhold these annotations so that the agent must identify the relevant implementation locations from the full repository.

Each attempt begins from a fresh repository snapshot. The agent may search and read files and edit the source, but cannot execute the project, run tests, or access the network. It returns one complete Git diff, used as the final submission. It does not receive any validator feedback.

Choice of useful-outcome representation. We represent each successful patch by the set of named functions it modifies. Exact Git-diff identity is too fine-grained for our purpose: two patches can differ in formatting, local edits, or other incidental details while intervening in the same parts of the implementation. Modified-function sets provide a deterministic intermediate representation that collapses such diff-level variation while distinguishing successful repairs that act at different implementation locations. Patches with the same modified-function set therefore map to the same useful outcome.

Evaluation details. A submitted patch must be a parseable Git diff that applies cleanly to the original repository. The corresponding PatchEval-Verified environment applies the diff and runs the revised vulnerability test with a 600-second limit. A patch earns credit only if this validation succeeds; successful patches contribute their modified-function set, while all other attempts contribute no useful outcome.

To construct the modified-function set, we parse each changed Python, JavaScript/TypeScript, or Go source file and map changed lines to their innermost enclosing named function in the original

repository. Removed lines retain their original coordinates, while inserted lines are anchored to adjacent original lines. Anonymous functions map to their nearest named enclosing function, edits outside named functions map to a module-level site, and unparseable files remain explicit file-level sites. The resulting sorted set of sites defines the patch's useful outcome.

C.4 BUG FINDING

Benchmark setup. We construct the task from C++ submissions in the CodeContests test and validation splits (Li et al., 2022). We remove problems that require file I/O or images and retain problems with at least three accepted submissions, five incorrect submissions, and one stored test. We verify up to five accepted submissions on a fixed probe of at most 80 stored tests and require at least three of them to reproduce every expected output.

We retain incorrect submissions that compile and fail at least one probe test. Incorrect submissions with the same pattern of failures on the probe tests are grouped into one behavioral bug class. This produces 188 problems and 6,422 fixed bug classes, with a median of 30 classes per problem.

On each attempt, the model receives the problem statement and generates up to five complete standard-input test cases. The prompt does not ask the model to rank the tests or make them distinct.

Choice of useful-outcome representation. We measure useful outcomes at the level of behavioral bug classes rather than generated test inputs. Exact input identity is too fine-grained: different concrete inputs can expose the same faulty behavior, so counting them separately would overstate useful coverage. Instead, each valid input is credited with the behavioral bug classes it exposes when executed against the fixed pool of incorrect submissions. Different inputs that expose the same bug class therefore contribute the same useful outcome, while an input may contribute multiple outcomes if it exposes multiple classes. Credits are unioned across the inputs in an attempt and across repeated attempts.

Evaluation details. At evaluation time, the accepted reference programs and incorrect programs are compiled under contest-judge conditions and executed without network access, using each problem's memory limit and a wall-time limit of $\min(t_{\text{problem}} + 1, \text{s}, 6, \text{s})$.

A generated input is valid when at least two reference programs finish and all finishing references agree on the output. Outputs are compared as whitespace-separated tokens, with a numeric tolerance of 10^{-6}. A valid input exposes an incorrect implementation if that implementation produces a different output, exits unsuccessfully, or exceeds a CPU or wall-time limit. The corresponding behavioral bug class is then credited to the attempt, with each class credited at most once within an attempt.

The scorer evaluates at most the first five parsed inputs in response order. Invalid inputs contribute no bug-class credit but remain part of the submitted attempt.

C.5 DIFFERENTIAL DIAGNOSIS

Benchmark setup. We follow the case selection of the AMIE differential-diagnosis study (McDuff et al., 2025), using its 302 NEJM clinicopathological-conference cases. For each case, we reconstruct the History of Present Illness from the corresponding NEJM report, beginning at "Presentation of Case" and ending at the next clinical section, while removing tables, captions, headers, footers, and affiliations. The reconstructed History of Present Illness is the sole clinical case information provided to the evaluated model.

Reference target construction. We map the physician-written differential diagnoses and final diagnosis to concepts in UMLS 2025AB. Our frozen local index contains English, non-suppressed UMLS names, synonyms, semantic types, and definitions. Queries and indexed strings are normalized for case, punctuation, and whitespace. Retrieval prioritizes exact matches, followed by BM25 matches over query tokens, and returns the ten highest-ranked distinct Concept Unique Identifiers (CUIs).

The reference diagnoses are mapped once to CUIs using OpenAI Codex with the frozen local search tools. These mappings are then frozen as part of the benchmark target; Codex is not used to judge

evaluated outputs, whose scores are computed deterministically by exact CUI matching against this fixed target. Codex may reformulate searches and inspect retrieved names, synonyms, semantic types, and definitions before selecting one or more retrieved CUIs, or no CUI when none is appropriate. The mapping preserves the specificity of the written diagnosis, and multiple CUIs may be retained when the diagnosis is ambiguous. Final diagnoses are mapped independently from the published differential. Every selected reference CUI must exist in the frozen index and have appeared among the retrieved candidates.

The published differentials contain 2,190 terms, of which 2,179 map to at least one CUI and 148 have multiple retained mappings. Final diagnoses are mapped separately, with 296 of 302 receiving a CUI. The target for each case is the union of the CUIs assigned to its published differential and final diagnosis, producing target sets of size 2–19 (median 8, mean 8.32). The same fixed target is used for every evaluated configuration.

Choice of useful-outcome representation. We represent diagnoses by UMLS CUIs rather than their generated strings. Exact diagnosis strings are too fine-grained: synonyms, abbreviations, and paraphrases can refer to the same clinical concept and should not count as distinct coverage. UMLS concepts provide a fixed intermediate representation that collapses such surface variation while retaining clinically distinct concepts. Different generated diagnoses mapped to the same CUI therefore contribute the same useful outcome.

Prediction coding and evaluation. On each attempt, the evaluated model generates a ranked differential of at most seven diagnoses. The same model then maps each generated diagnosis to UMLS concepts using the frozen local search tools using the same temperature, thinking setting, and output-token allowance as differential generation. It retains the case and its own generated differential as context but does not receive the reference CUIs. Concept coding is therefore part of the evaluated attempt rather than a post-hoc evaluator-side transformation.

The scorer retains at most the first seven parsed diagnoses and compares the submitted CUIs with the fixed reference target by exact CUI intersection. Diagnosis-concept precision is the fraction of distinct submitted CUIs contained in the target, with zero assigned when no CUI is submitted. VTC at budget k is the fraction of target CUIs recovered across the k attempts.

C.6 BRIGHT-PRO EVIDENCE SEARCH

Benchmark setup. We use a pinned public release of the `yale-nlp/Bright-Pro` dataset (Zhao et al., 2026). It contains 739 questions from seven StackExchange domains, 2,763 expert-annotated reasoning aspects, 5,272 released gold passages, and 526,319 corpus documents. Each question has between one and five reasoning aspects. Raw aspect weights are in $1, 2, 3$ and are normalized to sum to one within each question.

We build a separate frozen BM25 index for each domain, so each question searches only documents from its corresponding domain. The indices use BM25S 0.3.9 with Lucene BM25 IDF, $k_1 = 0.9$, $b = 0.4$, lowercase regex tokenization, Porter stemming, and the Lucene English stopword set. These settings match the original BRIGHT BM25 parameters, although our tokenizer is a close Python reproduction rather than the original Lucene analyzer.

Choice of useful-outcome representation. We represent useful evidence at the level of annotated reasoning aspects rather than exact passage identity. Passage identity is too fine-grained for coverage because several passages may support the same part of the information need. Counting those passages separately would therefore reward redundant evidence. At the other extreme, deciding whether complete generated answers contribute different information would require semantic judgment. BRIGHT-Pro's expert reasoning-aspect annotations provide a fixed intermediate representation of the distinct pieces of evidence needed to answer a question. Multiple selected passages supporting the same aspect therefore contribute the same useful outcome.

Retrieval and selection procedure. On each attempt, the model submits one batch of one to three nonempty search queries. Each query retrieves the top 15 passages from the corresponding domain index. Results are interleaved by within-query rank, deduplicated by document ID, and truncated

to the first 15 passages. The visible candidate bundle is further truncated under the fixed character budget specified in the released task artifact.

The model then selects between one and five unique passage labels from the visible pool. It sees the passage text and query/rank provenance, but not document IDs, retrieval scores, released-gold labels, reasoning aspects, aspect weights, qrels, or the reference answer. Missing or malformed search or selection calls are retained as zero-scoring attempts.

Evaluation details. Exact released document IDs determine whether a selected passage is a released gold passage. Each selected gold passage contributes its annotated reasoning aspect, and an aspect receives credit at most once within an attempt even when multiple selected passages support it. VTC at budget k is the weighted fraction of reasoning aspects recovered across the k attempts, using the normalized aspect weights.

Because the released qrels are not exhaustive relevance judgments, unjudged passages receive no credit.

D Additional Results and Robustness

D.1 VTC rankings across attempt budgets

Table 9 compares configuration rankings by VTC@1 and VTC@H across all 24 configurations. We select the configuration with the highest VTC@1 and evaluate its Top-1 VTC@H regret using the same definition as Table 2. Agreement counts how many configurations appear in both Top-5 sets. The final column reports the mean VTC gain from one attempt to the headline budget, averaged across all 24 configurations in each task. These gains remain in each task's native VTC units and are not averaged across tasks.

For molecule design and repository repair, VTC@1 equals the one-draw quality metric because each attempt contributes at most one useful outcome. In the other three tasks, one attempt can cover a set of useful outcomes, so VTC@1 is already a set-coverage quantity and is not identical to the quality metric used in Table 2.

Table 9: **VTC rankings can depend on the attempt budget.** Top-1 regret is the relative VTC@H shortfall of the configuration selected by VTC@1 from the VTC@H leader. Top-5 agreement is the overlap between the VTC@1 and VTC@H Top-5 sets. Mean VTC gain is the difference between VTC@H and VTC@1, averaged across all 24 configurations and reported on each task's native scale.

Task	Top-1 regret	Top-5 agreement	Mean VTC gain
Molecule design	8.0%	4/5	3.919
Repository repair	24.7%	0/5	0.974
Bug finding	0.0%	5/5	0.231
Differential diagnosis	6.2%	1/5	0.131
Evidence search	3.7%	1/5	0.277
Task macro-average / total	8.5%	11/25	—

D.2 FULL CONFIGURATION RESULTS

Table 10 reports both requested selection quantities for every combination of model, thinking setting, and temperature. The task-specific definitions of one-draw quality and VTC are given in Table 1; values are kept on their native task scales rather than normalized across tasks. The complete tables abbreviate DeepSeek-V4-Flash, Qwen3.6-27B, and Qwen3.6-35B-A3B as DS-V4, Qw3.6-27B, and Qw3.6-35B.

Table 10: **Quality and VTC for all 24 configurations.** Each task reports its one-draw quality measure and VTC at headline budget H. Quality is, from left to right, on-spec rate, repair success, useful-test rate, diagnosis-concept precision, and gold-passage precision. Darker shading indicates higher values within each task–metric column; bold marks the largest value.

Configuration			Molecule design		Repository repair		Bug finding		Diagnosis		Evidence search	
Model	**Thinking**	T	Quality	VTC@20	Quality	VTC@10	Quality	VTC@5	Quality	VTC@5	Quality	VTC@10
DS-V4	No	0.6	0.303	3.322	0.573	1.453	0.967	0.760	0.209	0.314	0.562	0.746
DS-V4	No	1.0	0.299	4.089	0.576	1.527	0.957	0.798	0.205	0.337	0.557	0.763
DS-V4	No	1.2	0.298	4.362	0.581	1.607	0.950	0.805	0.199	0.344	0.561	0.775
DS-V4	Yes	0.6	0.618	4.344	0.578	1.448	0.960	0.843	0.234	0.336	0.567	0.750
DS-V4	Yes	1.0	0.616	4.529	**0.588**	1.537	0.950	0.850	0.229	0.336	0.567	0.764
DS-V4	Yes	1.2	0.598	4.566	0.580	1.540	0.950	0.855	0.229	0.341	0.564	0.772
GLM-4.7	No	0.6	0.326	4.648	0.493	1.875	0.948	0.835	0.215	0.287	0.599	0.777
GLM-4.7	No	1.0	0.291	4.697	0.494	2.038	0.945	0.865	0.210	0.308	0.598	0.797
GLM-4.7	No	1.2	0.272	4.554	0.493	**2.042**	0.939	0.883	0.205	0.318	0.592	**0.807**
GLM-4.7	Yes	0.6	0.630	6.005	0.490	1.671	0.971	0.869	**0.271**	0.292	**0.618**	0.779
GLM-4.7	Yes	1.0	0.640	6.122	0.496	1.859	**0.971**	0.886	0.263	0.293	0.615	0.794
GLM-4.7	Yes	1.2	0.655	**6.307**	0.503	1.950	0.967	**0.895**	0.251	0.298	0.614	0.800
Qw3.6-27B	No	0.6	0.287	2.756	0.507	1.105	0.970	0.680	0.217	0.331	0.454	0.686
Qw3.6-27B	No	1.0	0.265	3.353	0.508	1.155	0.966	0.740	0.196	**0.358**	0.448	0.721
Qw3.6-27B	No	1.2	0.254	3.563	0.512	1.195	0.964	0.763	0.185	0.357	0.437	0.723
Qw3.6-27B	Yes	0.6	0.654	5.535	0.505	1.260	0.828	0.792	0.263	0.327	0.415	0.658
Qw3.6-27B	Yes	1.0	**0.720**	5.806	0.516	1.367	0.900	0.816	0.253	0.334	0.400	0.667
Qw3.6-27B	Yes	1.2	0.690	5.515	0.519	1.418	0.912	0.826	0.245	0.334	0.375	0.674
Qw3.6-35B	No	0.6	0.235	1.950	0.454	1.165	0.967	0.730	0.222	0.323	0.444	0.645
Qw3.6-35B	No	1.0	0.249	2.644	0.452	1.222	0.963	0.763	0.202	0.338	0.417	0.670
Qw3.6-35B	No	1.2	0.237	2.800	0.452	1.318	0.955	0.778	0.194	0.351	0.402	0.681
Qw3.6-35B	Yes	0.6	0.503	4.103	0.461	1.213	0.583	0.710	0.243	0.323	0.270	0.545
Qw3.6-35B	Yes	1.0	0.612	4.625	0.473	1.307	0.736	0.769	0.233	0.329	0.479	0.676
Qw3.6-35B	Yes	1.2	0.626	4.736	0.472	1.371	0.760	0.771	0.229	0.332	0.476	0.684

D.3 MOLECULE STRUCTURAL-GRANULARITY SENSITIVITY

Bemis–Murcko scaffolds are an intermediate abstraction: canonical molecule identity also preserves sidechains, whereas a ring-system identity removes both sidechains and the linkers between separate ring assemblies. We test whether the MolGen result depends on this intermediate choice by applying all three mappings to the same on-spec molecules. The ring-system key is the sorted collection of fused or bridged ring assemblies in a molecule; it preserves atom and bond types and repeated assemblies but ignores how separate assemblies are linked. Acyclic molecules share one empty key. All columns in Table 11 use the same 164 prompts, quality gate, and subset-counting estimator.

At one draw, all three mappings necessarily give the same coverage and select Qwen3.6-27B at $T = 1.0$ with thinking. At $H = 20$, exact molecule identity retains that leader, but both Bemis–Murcko scaffold and ring-system coverage select GLM-4.7 at $T = 1.2$ with thinking. The one-draw leader reaches 3.895 ring systems, versus 4.559 for the ring-system leader, so the budget-dependent leader change is not specific to the scaffold mapping. The complete ordering is less stable: scaffold and ring-system coverage have Spearman correlation 0.840 across the 24 configurations and share two of their Top-5 configurations. For comparison, exact-molecule and scaffold coverage have correlation 0.909 and Top-5 overlap four of five. Thus the endpoint winner and leader reversal survive a coarser structural definition, while finer configuration-rank claims remain granularity-dependent.

Table 11: **MolGen sensitivity to structural granularity.** Entries are the mean expected number of distinct on-spec identities over the 164-prompt paper scope. VTC@1 is common to all three mappings because one accepted molecule contributes one identity at any granularity. Bold marks the largest value in each column.

Configuration			**One draw**	**Coverage at $H = 20$**		
Model	**Thinking**	T	VTC@1	Molecule	Scaffold	Ring system
DS-V4	No	0.6	0.303	4.242	3.322	2.837
DS-V4	No	1.0	0.299	5.380	4.089	3.690
DS-V4	No	1.2	0.298	5.664	4.362	3.996
DS-V4	Yes	0.6	0.618	9.338	4.344	3.123
DS-V4	Yes	1.0	0.616	9.582	4.529	3.203
DS-V4	Yes	1.2	0.598	9.387	4.566	3.282
GLM-4.7	No	0.6	0.326	6.016	4.648	4.268
GLM-4.7	No	1.0	0.291	5.780	4.697	4.376
GLM-4.7	No	1.2	0.272	5.419	4.554	4.280
GLM-4.7	Yes	0.6	0.630	10.230	6.005	4.235
GLM-4.7	Yes	1.0	0.640	10.692	6.122	4.275
GLM-4.7	Yes	1.2	0.655	11.050	**6.307**	**4.559**
Qw3.6-27B	No	0.6	0.287	3.993	2.756	2.198
Qw3.6-27B	No	1.0	0.265	4.441	3.353	2.837
Qw3.6-27B	No	1.2	0.254	4.520	3.563	3.061
Qw3.6-27B	Yes	0.6	0.654	10.608	5.535	3.731
Qw3.6-27B	Yes	1.0	**0.720**	**11.397**	5.806	3.895
Qw3.6-27B	Yes	1.2	0.690	10.703	5.515	3.684
Qw3.6-35B	No	0.6	0.235	3.395	1.950	1.811
Qw3.6-35B	No	1.0	0.249	4.483	2.644	2.472
Qw3.6-35B	No	1.2	0.237	4.447	2.800	2.644
Qw3.6-35B	Yes	0.6	0.503	7.602	4.103	3.359
Qw3.6-35B	Yes	1.0	0.612	9.161	4.625	3.877
Qw3.6-35B	Yes	1.2	0.626	9.458	4.736	3.859

D.4 SURFACE-DIVERSITY METHODOLOGY AND COMPLETE RESULTS

Surface diversity measures variation that is directly observable in the generated outputs, before the coarser task-specific mapping used by VTC. For molecule design, we count distinct canonical molecules that are valid and satisfy the requested property constraints. For repository repair, we count distinct exact textual diffs among patches that pass PatchEval-Verified. For bug finding, we count distinct normalized test-input strings that agree with the reference implementation and kill at least one bug class. For differential diagnosis, we count distinct canonicalized raw diagnosis terms whose mapped concept set intersects the case's gold differential. These choices deliberately preserve differences that VTC may merge: for example, distinct molecules can share a scaffold, distinct patches can modify the same functions, and distinct diagnosis phrases can map to the same clinical concept.

For these four tasks, we apply the subset-counting estimator in Equation 3 to the surface units and average the resulting expected count equally over task instances. Invalid or off-target attempts contribute no unit but remain in the attempt budget. Evidence search has no natural exact surface unit. For each saved evidence-selection attempt, we run a separate answer-generation call using the question and the passages selected in that attempt. We measure variation among the answers from the first $H = 10$ attempts using $1 - \mathrm{SelfBLEU}$. Each answer is compared with the other answers for the same question, and scores are averaged across answers and questions. We use NLTK 3.10.0 sentence-level BLEU-4 with method-2 smoothing (Chen & Cherry, 2014). Questions with fewer than two nonempty answers receive surface diversity zero. Higher values indicate greater surface variation, but values are not comparable across tasks.

Table 12 pairs each surface measure with VTC at the same headline budget H for all 24 configurations and uses the same row order as Table 10.

Table 12: **Surface diversity and VTC for all 24 configurations.** Each task pairs its surface-diversity measure with VTC at the same headline budget H. Surface diversity counts canonical molecules, accepted exact diffs, exact valid test inputs, and gold-mapped raw diagnosis terms; evidence search uses $1 - \mathrm{SelfBLEU}$ over retrieval-conditioned answers. Bold marks the largest value in each column.

Configuration			Molecule design		Repository repair		Bug finding		Differential diagnosis		Evidence search	
Model	**Thinking**	T	Surf@20	VTC@20	Surf@10	VTC@10	Surf@5	VTC@5	Surf@5	VTC@5	Surf@10	VTC@10
DS-V4	No	0.6	4.24	3.32	4.82	1.45	11.79	0.76	4.29	0.31	0.36	0.75
DS-V4	No	1.0	5.38	4.09	5.00	1.53	12.36	0.80	4.83	0.34	0.43	0.76
DS-V4	No	1.2	5.66	4.36	**5.21**	1.61	12.41	0.80	**5.02**	0.34	0.47	0.77
DS-V4	Yes	0.6	9.34	4.34	4.83	1.45	17.01	0.84	4.10	0.34	0.38	0.75
DS-V4	Yes	1.0	9.58	4.53	5.04	1.54	17.48	0.85	4.16	0.34	0.43	0.76
DS-V4	Yes	1.2	9.39	4.57	5.02	1.54	17.69	0.86	4.31	0.34	0.47	0.77
GLM-4.7	No	0.6	6.02	4.65	4.53	1.88	15.06	0.83	3.10	0.29	0.37	0.78
GLM-4.7	No	1.0	5.78	4.70	4.65	2.04	16.77	0.86	3.59	0.31	0.43	0.80
GLM-4.7	No	1.2	5.42	4.55	4.69	**2.04**	17.17	0.88	3.81	0.32	0.47	**0.81**
GLM-4.7	Yes	0.6	10.23	6.01	4.37	1.67	18.72	0.87	3.03	0.29	0.36	0.78
GLM-4.7	Yes	1.0	10.69	6.12	4.54	1.86	19.65	0.89	3.11	0.29	0.40	0.79
GLM-4.7	Yes	1.2	11.05	**6.31**	4.66	1.95	**20.08**	**0.89**	3.25	0.30	0.43	0.80
Qw3.6-27B	No	0.6	3.99	2.76	3.84	1.10	11.22	0.68	3.67	0.33	0.37	0.69
Qw3.6-27B	No	1.0	4.44	3.35	4.04	1.16	13.70	0.74	4.25	**0.36**	0.44	0.72
Qw3.6-27B	No	1.2	4.52	3.56	4.13	1.20	14.71	0.76	4.46	0.36	0.48	0.72
Qw3.6-27B	Yes	0.6	10.61	5.54	4.01	1.26	15.81	0.79	3.13	0.33	0.44	0.66
Qw3.6-27B	Yes	1.0	**11.40**	5.81	4.25	1.37	17.34	0.82	3.30	0.33	0.50	0.67
Qw3.6-27B	Yes	1.2	10.70	5.52	4.38	1.42	18.06	0.83	3.39	0.33	0.54	0.67
Qw3.6-35B	No	0.6	3.39	1.95	3.66	1.17	12.31	0.73	3.71	0.32	0.38	0.65
Qw3.6-35B	No	1.0	4.48	2.64	3.84	1.22	13.13	0.76	4.16	0.34	0.46	0.67
Qw3.6-35B	No	1.2	4.45	2.80	3.96	1.32	13.02	0.78	4.40	0.35	0.50	0.68
Qw3.6-35B	Yes	0.6	7.60	4.10	3.83	1.21	11.71	0.71	3.22	0.32	0.46	0.54
Qw3.6-35B	Yes	1.0	9.16	4.62	4.12	1.31	14.73	0.77	3.37	0.33	0.51	0.68
Qw3.6-35B	Yes	1.2	9.46	4.74	4.22	1.37	15.17	0.77	3.53	0.33	**0.55**	0.68

D.5 INSTANCE-RESAMPLING ROBUSTNESS OF HEADLINE COMPARISONS

We assess the three headline comparisons that depend most directly on the finite instance set using 10,000 paired bootstrap resamples within each benchmark. Within a task, each replicate applies the same resampled instance weights to every configuration and metric, then recomputes the coverage values, rankings, Top-5 sets, Top-1 regret, and agreement. MolGen is resampled uniformly over its 164-prompt evaluation set. We use percentile 95% intervals throughout; Table 13 summarizes the resulting robustness checks.

Table 13: **Instance-resampling robustness of the headline comparisons.** For leader reversals, Δ_k is VTC@k of the headline-budget leader minus that of the one-attempt leader. Proxy-selection results rerank configurations within each resample. Temperature results report matched $T = 0.6$ versus $T = 1.2$ comparisons whose VTC difference remains positive under resampling.

Claim	Scope	Observed result	Instance-resampling robustness
Leader reversal	Molecule design Qw-27B T=1.0 Tk → GLM T=1.2 Tk	$\Delta_1 = -0.066$; $\Delta_H = +0.502$	Δ_1: [-0.089, -0.044]; Δ_H: [0.150, 0.855]
Leader reversal	Repository repair DS T=1.0 Tk → GLM T=1.2	$\Delta_1 = -0.095$; $\Delta_H = +0.504$	Δ_1: [-0.129, -0.062]; Δ_H: [0.319, 0.693]
Leader reversal	Differential diagnosis DS T=0.6 Tk → Qw-27B T=1.0	$\Delta_1 = -0.0173$; $\Delta_H = +0.0222$	Δ_1: [-0.0263, -0.0082]; Δ_H: [0.0092, 0.0352]
Leader reversal	Evidence search GLM T=0.6 → GLM T=1.2	$\Delta_1 = -0.0131$; $\Delta_H = +0.0298$	Δ_1: [-0.0180, -0.0084]; Δ_H: [0.0222, 0.0376]
Proxy selection	One-draw quality five-task macro	Top-1 regret 11.1%; Top-5 agreement 10/25	regret [8.7%, 16.9%]; agreement [8, 12]/25
Proxy selection	Surface diversity five-task macro	Top-1 regret 9.7%; Top-5 agreement 11/25	regret [7.6%, 11.6%]; agreement [9, 13]/25
Temperature	$T = 0.6 \rightarrow 1.2$ 40 matched pairs	38/40 positive	36/40 intervals above zero MolGen 6/8; PatchEval 7/8; BugFind 8/8; AMIE 7/8; BRIGHT-PRO 8/8

The intervals for all four fixed point-estimate leader pairs remain below zero at $k = 1$ and above zero at H. The task-macro Top-1 regret intervals are 8.7–16.9% for one-draw quality and 7.6–11.6% for surface diversity, while their total Top-5 agreement intervals remain 8–12 and 9–13 out of 25, respectively. Of the 38 positive matched temperature comparisons, 36 have individual intervals entirely above zero. These are *instance-resampling robustness intervals*: they measure sensitivity to reweighting the frozen benchmark instances, not uncertainty over newly generated attempts or population-level certainty over real-world tasks.