We thank you (R1, R2, R3) for your meticulous feedback. We are delighted to find that you all regard the paper's conception and methodology as both novel and rigorous; R2 and R3 further commend the manuscript for its lucid exposition. We are particularly pleased that R3 recognizes that our method addresses an important problem for SPLs and proposes a solution that is applicable to real-world scenarios. We answer questions below and will incorporate all feedback in the final version.

\textbf{R1}

We sincerely thank you for your thorough review and incisive feedback. In response to your comments, we have revised every point line-by-line; all changes are highlighted in blue in the revised manuscript for immediate reference. The anonymized repository is available at: https://anonymous.4open.science/r/ASE-1. Your invaluable guidance has markedly elevated the quality of our work.

Q1-1: Did the competing baselines also use Coprocessor, and how does it affect algorithmic performance?

Thank you for flagging this critical omission. The original manuscript failed to state explicitly that every baseline was run with Coprocessor, which could indeed mislead. We have therefore inserted the following sentence in Sec.5 A of the revised paper: "For a strictly fair comparison, all algorithms were uniformly preprocessed with the identical Coprocessor configuration." This revision is purely editorial and does not require any experiments to be redone.

Coprocessor preserves the variable count while shrinking the clause set, thereby reducing SAT solving time. Because this acceleration is applied uniformly across all compared algorithms, the performance gaps we report originate solely from the algorithmic designs themselves, not from the preprocessor.

Q1-2 What are (1) the runtime share of the SAT solver within the framework, (2) the extent to which its performance determines overall performance, and (3) whether its invocation count can serve as a framework-level evaluation metric?

We have profiled the framework exhaustively on 124 benchmark instances. (1) The SAT solver dominates the total cost, accounting on average for 16 % ($\sigma = 2$ %) of the wall-clock time. (2) The solver's performance is decisive: Pearson's r between overall runtime and the number of SAT calls is 0.97 ($p < 0.001$). (3) Because every

experiment uses the identical solver version and the same encoding strategy, the count of SAT invocations is a direct, fair proxy for the entire framework's efficiency. The full data are available at https://anonymous.4open.science/r/ASE-2.

\textbf{R2}

Thank you for your careful reading and the invaluable comments on our paper. In direct response to your suggestions, we have completely rewritten the relevant sections to address every concern you raised. Every change is highlighted in blue and committed to the anonymized repository at https://anonymous.4open.science/r/ASE-1 for immediate inspection.

Q2-1 More granular details about the characteristics of the benchmark programs

The benchmark was originally assembled by Baranov et al. [1] and its instances vary widely in scale. They are encoded into CNF from a large collection of real-world feature models taken from operating systems, core utilities [2], and the KConfig configuration systems [3].

(1) The benchmark is both structurally representative and computationally challenging: the formula density (clause-to-variable ratio) clusters tightly between 2 and 3, situating the instances within the theoretical 3-SAT phase-transition region. (2) Clause lengths follow a bimodal distribution: binary clauses account for 62 % on average, while ternary and longer clauses constitute 38 %. (3) The solving-time distribution is tightly concentrated with no heavy-tail. Using minisat-2.2.0 as the solver, the median runtime is 0.0037 s (IQR = 0.00054 s). Detailed data are provided in the appendix: https://anonymous.4open.science/r/ASE-3.

\textbf{R3}

Thank you for your rigorous comments and constructive suggestions. To fully address your concerns, we provide a detailed point-by-point clarification below.

We acknowledge that our earlier wording was overly absolute. We have therefore revised "these methods merely focus on diversity and do not directly aim to enhance tuple coverage" to "existing methods improve tuple coverage only indirectly through diversity- or distance-based proxies," and we explicitly state that none of them set "directly maximizing the number of newly covered pairwise tuples" as their primary objective, whereas DivSampCA adopts this objective as its sole optimization target.

Since both Baital and LS-SamplingPlus are t-wise CovMax algorithms that improve tuple coverage through weighted sampling, and [4] has shown LS-SamplingPlus to outperform Baital significantly, we retain only the more representative one to avoid redundancy. Furthermore, our discussion focuses on methods whose technical innovations are closest to DivSampCA; CAmpactor, which follows a local-search paradigm, differs fundamentally in algorithmic structure and optimization objective and is therefore not discussed.

Finally, we have meticulously revised the manuscript to resolve all writing issues you identified. The updated version is now available at https://anonymous.4open.science/r/ASE-1 for your convenience.

Q3-1 What is the scenario of use for DivSampCA?

DivSampCA is particularly advantageous in scenarios demanding rapid initiation and incremental on-demand generation. Consider industrial smoke testing, where Chauhan [5] emphasizes: “\textit{Smoke testing is conducted to ensure whether the most crucial functions of a program are working, but not bothering with finer details.}” With DivSampCA, the first test cases can be executed immediately, whereas CAmpactor must generate the entire test suite in one go, precluding any early start of testing.

[1] https://dl.acm.org/doi/10.1145/3368089.3409744

[2] https://ieeexplore.ieee.org/abstract/document/8730148

[3] https://dl.acm.org/doi/abs/10.1145/3336294.3336322

[4] https://dl.acm.org/doi/10.1145/3688836

[5] https://www.ijsrp.org/research-paper-0214/ijsrp-p2663.pdf