We thank the reviewers for their positive and detailed comments. We split our rebuttal letter into two parts; the first part identifies some factual errors in reviews 3 and 4, while the second part responds to some specific queries raised in the reviews.

**Part I: factual errors we identified in the reviews.**

**(R3)"…One potential reason to accept this at CAV would be if the technique involved some novelty at the level of the optimization algorithm..."**

Our paper was submitted to "Industrial Experience Reports and Case Studies".

The CfP explicitly states that: "These papers are expected to describe the use of formal methods techniques in industrial settings or in new application domains. Papers in this category do not necessarily need to present original research results but are expected to contain novel applications of formal methods techniques as well as an evaluation of these techniques in the chosen application domain". Review 1 points out “I appreciate the application of formal methods in industrial applications…, and believe that the paper... could be a nice addition to CAV” and review 2 describes "The paper is one of the first to apply formal methods to PV system design... I think it's fantastic to see researchers applying formal methods in areas related to renewable energy… and the experimental results seem good".

**(R4)"…the synthesis algorithm does not model the optimality objective explicitly, instead relying on standard CEGIS."**

In Section 2.1, we describe that we do not rely on standard CEGIS: "In our CEGIS variant, there exist four differences related to the traditional one...". In this section, we provide details of the main differences in our approach to standard CEGIS. Besides, we model the optimality objective explicitly, as stated in Section 3: "The optimal sizing of PV systems is made by the best compromise between two objectives: power reliability and system cost...". As described in Section 2.1, the VERIFY phase is responsible for obtaining the lowest cost (minimum cost is the optimization objective function); in particular, the cost of that solution is calculated in F\_{obj} (Line 21of Algorithm 1).

**(R4)"The paper claims… more accurate results than existing commercial tools…** **this claim is not validated..."**

In Section 4, we report that HOMER Pro does not use costs related to charge controllers, which were introduced into our synthesis model. This charge controller cost makes the synthesis approach more precise and real. We observed that there exists a divergence in case study 2, where the costs presented by HOMER Pro were 54% higher than our synthesis tool (cf. EG3).

**(R3)"CBMC is able to generate an optimal configuration for all but one of the settings"**

In Section 4.4, we describe: "CPAchecker was able to synthesize the optimal sizing in six out of seven case studies (cases 1 to 6)". In particular, CBMC is unable to produce any conclusive results, as stated in Section 4.4: "CBMC and ESBMC are unable to produce any conclusive results since time-outs or memory-outs occurred."

**Part II: questions posed by the reviews.**

**(R1)"Isn't CBMC a model checker for C/C++? Can't you use directly an SMT solver like Z3?"**

Yes, CBMC is a model checker for C/C++, which we used to implement our synthesis algorithm (similar to https://doi.org/10.1007/978-3-319-96145-3\_15). We tried to use SMT solvers directly using the SMT-lib format. However, they have all struggled with the discharged verification conditions (VCs) resulting from Algorithm 1 (cf. Section 3); in particular, we tried the SMT solvers Z3 and Boolector. State-of-the-art software model checkers are doing an excellent job of simplifying the VCs before invoking the underlying SMT solvers. Since we also wrote Algorithm 1 in the C programming language, we were able to find one software model checker (CPAchecker), which was able to deal with the VCs produced from Algorithm 1.

**(R1)"Is this optimization problem beyond the scope (or ability) of probabilistic programming?"**

We have not investigated the application of probabilistic programming yet within our research. We are aware that using this method, we can work even with incomplete models. However, the answer or solution presented by the technique could be a local optimal and not a global optimal; we will address this suggestion for future work in comparison with other heuristics that can produce fast response (e.g., genetic algorithm), but with limitation w.r.t. precision. In this future work, we can analyze the trade-off between performance and precision.

**(R2)"Currently, the CEGIS loop results in a \*single input\* being added at each iteration... Is there a more \*general\* constraint that the verifier can produce…?"** –

We realized this observation just after the paper submission. In an updated version of the algorithm, we have included a minimum cost for the beginning of the iterative loop (see Chapter 4, page 75, Algorithm 2 in https://ssvlab.github.io/lucasccordeiro/supervisions/phd\_thesis\_alessandro.pdf). This cost is the lowest among the list of equipment used, considering a configuration of just one solar PV panel, one charge controller, one battery, and one inverter. Indeed, if we use an extensive list of equipment, then the time to obtain the result tends to be more significant. We will address this comment in the camera-ready version of our manuscript.

**(R2)"Where does the verification step occur in Algorithm 1?... the VERIFY and SYNTHESIZE phases don't show up..."**

The SYNTHESIZE phase is obtained by a possible combination of equipment that meets the properties collected from the electrical requirements of the equipment itself, e.g., power, current, or voltage (cf. Section 3). This phase produces one feasible solution in line 20 of Algorithm 1 in terms of equipment and configuration (among 40 equipment items). The algorithm reaches line 21 with the cost related to this feasible solution. The VERIFY phase in line 22 ensures that only the lowest cost for some specific equipment combination will lead to a FAIL output (counterexample with equipment list and cost). The process is iterative since the "assert" statement in line 22 will perform a check, and the step of the iteration will be adjusted to a higher value if there exists no “FAIL”. The VERIFY phase can lead to "SUCCESS" if there exists no feasible solution, and the for-loop started in line 6 reaches the maximum cost. We will address this comment in the camera-ready version of our manuscript.

**(R2)"My understanding is that the verification step uses a model checker. What exactly does the model look like?"**

We use symbolic model checking for our VERIFY phase. As described in Section 3, the model (incl. constraints and properties) is obtained from the mathematical model used for the sizing, from the electrical characteristic of the equipment (to combine electrical compatibility among equipment, as current, voltage, or power), from the assume statements included explicitly in Algorithm 1, and from the objective function (minimum cost of the feasible PV sizing solution). We created a specific page for specifying the resulting VCs (constraints and properties): see https://drive.google.com/file/d/1ByNJJTd9Xhb1cYijGKpCxLq-fm7jprSG/view?usp=sharing.

**(R2)"In regards to the mismatched Windows/Ubuntu machine specs in section 4.3, why couldn't you just use separate Windows/Ubuntu VMs on the same host machine?"**

The ideal scenario would be to use the same hardware configuration for the experiments. However, we faced restrictions concerning the license for the HOMER Pro tool; besides, we did not have the autonomy to change the Linux VM machine installed in the servers of our university due to the internal policy.

**(R4)“the application demands reasoning about constraints in nonlinear real arithmetic, which is a hard problem. It is not clear how this problem will be solved.”**

At the end of Section 2.2, we describe that all the provided equations model the continuous-time behavior of the PV system; they produce real numbers except for the batteries and panels, which consist of integer numbers. Note that the underlying symbolic verifiers perform bit-precise verification based on the Floating-Point theory using the command-line provided in the footnotes of page 8. We could use a real arithmetic strategy to tackle these equations; however, we have exploited the floating-point arithmetic, which is an approximation of the real one.