First of all, we would like to thank all the reviewers for their comments. Below we answer to the concerns of each reviewer separately.

Reviewer 1**:** Thank you for the review and your comment. We will explain the motivation and the background in details and will give a clear description of the dynamic model and the difference between the streaming and the dynamic setting to improve the presentation of the paper. We will also write a complete related work section and the relation of our work to prior work specially a comparison between our work and the results from KZK18 and MKK17.

Reviewers 1 **and** 3**: a discussion for the update time and the memory constraint.** Here we briefly mention the difference between the streaming and the dynamic setting. In the streaming setting the main concern is the space complexity. We often compute a sketch of the input that is revealed in a streaming fashion. At the end of the stream we compute a solution/function using the sketch that we maintained in the course of stream. On the other hand, in the dynamic setting the main complexity is the time. The idea is given the input that is revealed in a streaming fashion, one is interested in seeing the solution and the changes in the solution after every insert or delete. The main motivation is for learning highly dynamic and sensitive data (such as time series) that we need to take an action as soon as we see a shift in the function of the underlying data that we observe. Since we need to react to changes in the solution fast, we need to (re)-compute the solution as fast as we can. Indeed, we cannot wait till the end of the stream and take the corresponding action afterwards. The underlying assumption for the dynamic setting is nowadays with machines that can easily have (SD)RAMs of GBs and soon TBs, the space constraint won't be a problem, but the time complexity is the main bottleneck. The results from KZK18 and MKK17 are streaming algorithms whose time complexities depend on the number of deletions (Theorem 1 of the second reference) which will be high if we want to (re)-compute a solution after each insertion or deletion.

Reviewer 2**:** Thank you for your comments. We find them useful and will incorporate them to improve the presentation of the paper. Yes, the assumption that the function f is monotone has been used in the related results from KZK18 and MKK17. Also, in Lemma 3 we used the fact that the function must be monotone, but indeed it is a good question to see if we can develop dynamic algorithms for non-monotone functions and we will think about it. As for the method of getting a worst case bound in Section 3, we did not know about "A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching". Definitely we will cite this paper.

Reviewer 3**:** Thank you for the review and the comments. In the appendix we mentioned a version of our algorithm that doesn't need to know the OPT value and has $O(\sqrt{n})$ time complexity. We recently found that a variant of our algorithm that does not need to know the OPT value and has poly-logarithmic dependency on n. The idea is to do our logarithmic rate of sampling and filtering besides maintaining a max-heap as we see new updates. We will add this new algorithm to the paper.

Reviewer 4**:** Thank you for your comments. We find them useful and will incorporate them to improve the presentation of the paper. Yes, the running time of our algorithms is measured on the number of oracle calls to the function f and we will explain it explicitly in the paper. We do not know if $1/2$ is the best that we can achieve or not. However, we think it will be very interesting to see if we can develop a dynamic algorithm with better than $1/2$-approximation guarantee. Many thanks for pointing out to the new arXiv submission "Fully Dynamic Algorithm for Constrained Submodular Optimization". We did not know about this paper. We should mention that this paper presents a dynamic algorithm whose expected update time is poly-logarithmic in n and k. Our algorithm works with high probability. We think we can use our worst case framework in Section 3 to improve their running time bound from expected to a high probability bound.

Reviewers 2 **and** 4**: the dependence on** k **in the running time.** we recently realized that a simple version of our algorithm has poly-logarithmic dependence on k in the running time. The idea is if at each step i of the recursion we sample $O(\epsilon^{-2}k\log n)$ elements and if we have many elements that are above the threshold, we collect more than one such element. We can then show that the number of elements that are survived after filtering at each step i drops by a factor of $\frac{\epsilon^{-2}}{2k}$. Thus, every insert or delete that changes a partial solution set G_i happens with probability $O(n/k)$, but the number of elements in the steps $> i$ are an order of $O(n/k)$ for which we have enough credit to re-run the following steps.