We are addressing only the major comments in this document. Most certainly, we will incorporate all minor comments relating to presentations, citations, clarifications, etc. In this document, RXCY refers to Comment Y by Reviewer X.

R1C1 [Heuristic method]: We agree GCOMB is a heuristic. This holds true for [4],[15] as well and, in general, for most deep-learning techniques. Our main objective is to scale to billion-sized graphs, where [4],[15], and approximation algorithms, such as Greedy, IMM and CELF fail. We will ensure to make this crystal clear.

R1C2 [Novelty]: GCOMB focuses on the *budget-constrained* scenario and this leads to several design choices that are different from [4]. Examples include a novel *probabilistic greedy* algorithm to construct budget-focused training data, and noise predictor for pruning the search space. In addition GCOMB uses a mixture of supervised learning (GCN) and reinforcement learning. In contrast, [4] is an end-to-end reinforcement learning architecture and thus time-consuming.

R2C1 [Relevance of CELF]: As reported in Table 1, GCOMB is consistently faster than CELF in both MCP and IM. Since CELF did not finish in IM even after 2 days, the reported results for IM are based on a random sample containing 5% of the edges in YouTube. The slowness of CELF in IM is also reported in [2].

R2C2 [Noise pruning + GREEDY/CELF]: NOISEPRUNER+CELF, i.e., running CELF only on non-noisy nodes, is orders of magnitude slower than GCOMB in IM (See Table 1). Pruning noisy nodes does not reduce the graph size; it only reduces the number of candidate nodes. To compute expected spread in IM, or coverage in MCP, we still require the entire graph, resulting in non-scalability. For ex., at budget $b \geq 100$, CELF did not finish in 2 days as computing marginal gain is #P-hard. Note that NOISEPRUNER+CELF is much faster than CELF. However, speed-up over CELF in Table 1 is smaller since the dataset there is 5% of YouTube. If our accepted, we will use a common dataset for both CELF and NOISEPRUNER+CELF, so that the speed-up is better depicted (We could not due to 5-day rebuttal deadline).

R2C3 [Budget constrained set combinatorial problems]: This is an excellent point. We propose to restate our scope as follows: *We aim to learn algorithms that can be approximated well by greedy algorithm (or where greedy is nearly optimal) and that have cardinality constraints.* Although GCOMB may generalize to other combinatorial problems, it remains to be investigated. Learning with a small budget and generalizing it to a larger one in test phase is quite non-trivial and we tackle it via probabilistic greedy and noise predictor.

R2C4 [Approx. guarantees]: See R1C1. **R2C5 [MVC in MCP formulation]** Will do. We lack space in rebuttal.

R2C6 [Is NOISEPRUNER part of GCN]: The noise prediction is a pre-processing step for GCN. As highlighted in Fig. 1, the hidden layers of GCN are trained only with nodes that pass the noise filter. The noise prediction component has r^b_{max} as a trainable parameter, which is trained using the first layer information of GCN through linear interpolation.

R3C1 [Compare with LazyGreedy (CELF), STOCHASTICGREEDY(SG), and OPIM]: GCOMB is faster than all three techniques (Table 1). For CELF, see R2C1 and R2C2. GCOMB is faster than OPIM and SG, and on average, better in quality. Like CELF, SG did not finish in IM even after 10 hours as it only reduces the number of marginal cost computations, but not the cost of computing the marginal gains itself, which is #P-hard. In contrast, GCOMB reduces *both* number of marginal cost computations (Noise pruning) and cost of computing marginal gains (GCN+Q-learning). In MCP, GCOMB is up to 20% better in quality at $\epsilon = 0.2$ and yet 2 times faster. SG fails to match quality even at $\epsilon = 0.05$, where it is even slower. Furthermore, SG is not drastically faster than CELF in MCP due to two reasons: (1) cost of computing marginal gain is $O(Avg.degree)$, which is fast. The additional data structure maintenance in SG to access sampled nodes in sorted order does not substantially offset the savings in reduced marginal gain computations. (2) Due to being scale-free networks, the inner loop of CELF terminates early in the sorted order.

R3C2 [Comparison to SODA 14]: IMM is an optimized version of the SODA-14 paper. SODA-14 does not scale due to the ϵ^{-3} factor and other large hidden constants in its time complexity of $O(bl^2(|E| + |V|)\log^2|V|/\epsilon^3)$.

R4C1 [Citations]: We will replace/add the citations as suggested.

R4C2 [Explanation of distribution D] and R4C4 [Choice of real graphs]: We do not assume any known distribution. We only assume that the training and test graphs come from the same (unknown) distribution D. For example, if one trains on scale-free social networks, the test graphs are expected to be scale-free social networks as well. We hope this also clarifies our choice of real graph datasets, all of which are social networks. The metric $|X|/|B|$ is distribution agnostic. We have reported the mean. We can also report the standard deviation through error bars.

R4C3 [Noise prediction and node quality prediction]: The key differentiating factor between noise predictor and node quality prediction is their computation complexities. This design choice is motivated by the observation that computationally expensive predictive tasks should be performed only for the promising nodes. Thus, the noise predictor prunes out nodes that will *not* be in the solution. For the remaining nodes, node quality prediction is performed.

Experiment	Speedup over CELF		NOISEPRUNER+CELF	Speedup over OPIM		Spread Difference with OPIM		SG (MCP in YT)		
Budget	IM (CO) in YT 5%	MCP in YT	Speedup in IM (CO) in YT	CO in YT	TV in YT	CO in YT	TV in YT	Speedup $\epsilon = 0.2$	Coverage Difference $\epsilon = 0.2$	Coverage Difference $\epsilon = 0.05$
20	9	4	12840	35	15	1×10^{-5}	$\mathbf{-1 \times 10^{-4}}$	2	$\mathbf{-0.09}$	$\mathbf{-0.001}$
50	11	4	46119	30	10	$\mathbf{-3 \times 10^{-5}}$	$\mathbf{-2 \times 10^{-4}}$	2	$\mathbf{-0.13}$	$\mathbf{-0.003}$
100	28	3	–	18	7	2×10^{-5}	$\mathbf{-3 \times 10^{-4}}$	2	$\mathbf{-0.16}$	$\mathbf{-0.005}$
150	–	2	–	15	6	$\mathbf{-2 \times 10^{-5}}$	$\mathbf{-3 \times 10^{-4}}$	2	$\mathbf{-0.18}$	$\mathbf{-0.005}$
200	–	2	–	13	4	$\mathbf{-7 \times 10^{-5}}$	$\mathbf{-4 \times 10^{-4}}$	2	$\mathbf{-0.20}$	$\mathbf{-0.006}$

Table 1: **[R2C1, R2C2, R3C1]** OPIM (version OPIM-C^+, which is the fastest. Code provided by authors) is run with the default parameters recommended by the authors in the paper. ϵ is recommended to be kept in range $[0.01, 0.1]$. Thus, we set it to $\epsilon = 0.05$. In Spread Difference and Coverage Difference, we compute (Baseline Performance-GCOMB performance), and thus, a negative difference indicates *better* performance by GCOMB. CO and TV denote the edge weight models in IM. YT denotes the YouTube dataset. MCP denotes Max Coverage Problem and IM denotes Influence Maximization.