We thank the reviewers for the not-so-common careful reading.

Following your suggestions:
- The write-up was significantly improved and was sent to a professional English editor.
- Pseudo code was replaced by more intuition and illustrations.
- Python's code was extended with comments and examples on "extreme" synthetic data to give more intuition. As promised, It will be published upon acceptance, and can be sent to the reviewers already now if necessary.

k-Means for lines is much harder than the k-means for points which has numerous approximations and coresets. We try to intuitively explain why below. This is also the reason why it took us few years to write this paper that suggests the first solution to such a fundamental problem in machine learning. Extension of this explanation was added to the new version.

Reviewer 1:

Q1: Do you assume that the input space is discrete and bounded?
A1: Certainly not. As stated in the main theorems, the guarantees of the algorithms hold for any set of n lines in $\mathbb{R}^d$. No hidden assumptions.

Q2: The objective function of the k-means line clustering problem is confusing. In Line 1 it is minimizing sum of squared distances, which is exactly same as k-means. Later on it changes to minimizing sum of distance. I am not sure this comparison is fair because EM k-means is to minimize the sum of squares.
A2: This is indeed confusing. The reason is that for simplicity we focused on the classic k-means. However, the results easily generalized to any Lipschitz function of distance, including sum to the power of $z > 0$ or m-estimators. Following this comment, we focus only on squared distances (including experimental results) and moved the generalization to the last section. We thank the reviewer for this useful comment.

Reviewer 2:

Q1: The paper is extremely difficult to read for a non-expert. Give intuition and move the pseudo-code.
A1: Indeed, to obtain such strong provable guarantees we had to use deep mathematical proofs. To help the non-expert: (1) We accepted the reviewer's suggestion and replaced the pseudo code by illustrations that were added to the overview. (2) We added detailed comments to our open Python's scripts, as well as example data sets of extreme cases that give intuition about how and why the algorithms work.

Reviewer 3:

We appreciate the careful reading of the reviewer.

Q1: You are saying that you have a deterministic algorithm but the theorem says the opposite.
A1: This is indeed a mistake in the introduction that was fixed. Algorithm 1 is deterministic as claimed in Lemma 6.3, but our coreset construction holds with high probability as stated in the main theorem.

Q2: Why using EM + k-means++ and not simply k-means++?
A2: k-means++ was never used in this paper. Unlike EM, we could not generalized k-means++ for lines. The reason is that both its input and output sets are points. In k-line means, the input is a set of lines and the output is a set of points. Moreover, the correctness of k-means++ heavily based on metric spaces, while the triangle inequality does not apply for a set of lines in $\mathbb{R}^d$. For example, constructing a coreset for the case that the optimal cost is 0 is trivial in k-means but not for k-line-means.

Q3: How did you compute the optimal solution?
A3: We use exhaustive search (few days of computation on Amazon's cloud). This is part of our open source code.

Q4: It seems to me that for getting an offline $(1+\epsilon)$-approximation, one may be able to combined your lemmas on sampling (say Lemma 6.3) together with a Simple D^2-Sampling.
A4: We aware of this result but unfortunately could not apply it due to the reasons in **A2** above. It would be awesome if the reviewer can give us a hint in case we missed something.