Thank you for your comments/feedback, it is much appreciated.

(1) Depth: Yes, these are important metrics you bring up. I can offer some insight about this already, I believe.

The size of the n-gram will have an effect on the speed of query answering for *some of the approximate search methods* I explored. The one I am primarily interested in now, however, is not effected. However, it may still have a large effect on **training time**. I am curious to see how that plays into training complexity. For instance, ignoring n-gram granularity, we could create a bloom filter for the entire text with n-grams from size 1 to size n. This means, simply, that a bloom filter will have a factor of n more members (the number of characters, or words, in an individual member is somewhat immaterial, I believe). Now, we can do exact (or approximate) search queries on this entire message in O(1) – with the exception that there will be a little bit of preprocessing on the actual query string the user submitted. While it is constant, it does allow for false positives.

So, we have more members, but we still have all those non-members to deal with that could potentially be matched as a false positive. We want to minimize the probability of false positives. So, we must learn parameters for the bloom filter to minimize the probability of a false positive while trying to keep the size of the bloom filter (how many indices + how many hash functions) reasonably small and the training time reasonably short. Some of my research, I think, may involve exploring different approaches to training them. Right now, I just use a random hash function generator; it seems to do a reasonably good job, though!

I'm not sure if having more members (more n-grams in the bloom filter) means a larger bloom filter. As a general rule, it is expected that every member requires 1.44 log2 (1/ε) bits, where ε is the probability of a false positive. However, we may be able to do this much better on the supposition that we do not actually need to reduce the false positive probability uniformly over the entire space of negative examples. So, how effective can we train it? How long will it take us to train it effectively—in terms of training time and bloom filter size? I think if we can get below 1.44 log2 (1/ε) for a given false probability ε , that would be pretty impressive. (We can get below that limit because of being selective on what we choose to have a low false probability on, i.e., P[fp | probable query]).

I like the idea of comparing our solution to inverted indices. I'm going to re-read that this evening (sorry I didn't get back to you earlier, btw; I will get back to you later this evening). I suspect there are marked differences (advantages and disadvantages) to our respective solutions, e.g., ours is more security conscious, but a comparison will still be illuminating, especially since, to provide locality metrics, we'll be using a block structure also (although it is not necessary unless we do want to provide locality metrics).

(2) Existing work: I appreciate your help here. I've been reading a lot too, although I haven't read 20 papers yet! I have my work cut out for me! ☺ During our meeting, maybe we can discuss this some more. I am curious to see what sources you have read, and I'll give you my sources also.