- How would you evaluate your autocomplete server? If you made another version, how would you compare the two to decide which is better?

1. response time: when you send the request, for example 'what do you', what is the response time for this query. For my current version, the response time can be evaluated by curl -s -w "%{time_total}\n", which is about 0.004s on average on my laptop.
2. insert time: if you have a new sentence, what is the time for insert this sentence to the trie tree. In my algorithm, to insert a new sentence to the trie, I will first need to run ngram on this sentence to see if the ngram appears in my ngram dictionary before. If not, I will add the new ngram to my ngram dictionary and insert the new sentence to the trie. The time complexity for ngram check is O(1) and insertion will be O(N), N is the length of the sentence.
3. memory cost: how many information you need to store in the memory. I will need to store the ngram dictionary and the trie tree. The ngram dictionary is serializable, it can be read and used later once it is constructed.
4. scalable: the insertion of questions in my current version is not parallelized. It would be better to have a parallelizable version.

- One way to improve the autocomplete server is to give topic-specific suggestions. How would you design an auto-categorization server? It should take a list of messages and return a TopicId. (Assume that every conversation in the training set has a TopicId).

I would use Vectorizer to learn the vocabulary of the messages then use it to create a document-term matrix. Relate the text with TopicId. Then do logistic regression or SVM to determine the TopicId of a new conversation. The max features, minimum frequency and maximum frequency of features should be carefully tuned.

- How would you evaluate if your auto-categorization server is good?

I would check the precision and recall of the classification.

- Processing hundreds of millions of conversations for your autocomplete and auto-categorize models could take a very long time. How could you distribute the processing across multiple machines?

I will divide the conversations to several parts and each machine will deal with part of them. The resulting ngram dictionaries are then combined from different machines.