Short Answer

Question 1: Semantics

Part 1: I think it should be the third sense, which is “examination”.
The Lesk Algorithm’s assumption is that we can identify the sense of words based on the content in the neighbors. The word “question” is both in the definition and the sentence while we don’t have any intersection of words for other senses. In addition, the word “failed” and “studied” in the neighborhood also implies that there is an evaluation of skill or knowledge. Therefore, I think “test” means “examination” here based on the Lesk Algorithm and neighborhoods.

Part 2: For the fifth definition, “the act of testing something”, I would like to add more words to the definition such as “measurement” or “measure” or “quantity”, because they’re usually present with “test” in this sense. The previous definition is not clear enough for Lesk Algorithm because it doesn’t have symbolic words.

Question 2: Evaluation

Part 1: We can set up a confusion matrix at first. Each column is for predictions while each row is for a true label.

	Negative	Neutral	Positive	NaN
Negative	2	3		
Neutral	2			
Positive	1	1	2	1
NaN		3	2	1

In my opinion, we can treat all NaN labels as neutral because it’s essentially not negative or positive, while I’m going to throw out all NaN labels since they can’t be used to evaluate the model. Therefore, we have the following updated table.

	Negative	Neutral	Positive
Negative	2	3	
Neutral	2	3	2
Positive	1	1	2

For Negative, we have precision = 2 / (2 + 2 + 1) = 0.4, recall = 2 / (2 + 3)=0.4
For Neutral, we have precision = 3 / (3 + 3 + 1) = 0.43, recall = 3 / (3 + 2 + 2) = 0.43
For Positive, we have precision = 2 / (2 + 2) = 0.5, recal = 2 / (2 + 1 + 1) = 0.5
Since each category has the same precision and recall, the F1 score just equals the precision. Then we average F1 over all categories, and get (0.4 + 0.43 + 0.5) / 3 = 0.443

Part 2: In this case, we're treating the original neutral and positive categories as one called "negative", and treat the original negative one called "positive". Therefore, we can update the second table.

	Positive	Negative
Positive	2	3
Negative	3	8

We have precision = 2 / (2 + 3) = 0.4, recall = 2 / (2 + 3) = 0.4, F1 = (2 * 0.4 * 0.4) / (0.4 + 0.4) = 0.4

Question 3: Sequence Tagging

Part 1: P(J) = 0.5, P(N) = 0.25, P(V) = 0.25

Part 2: P(J to N) = 1, P(V to D) = 0.5, P(V to STOP) = 0.5, P(D to N) = 1, P(N to STOP) = 0.75, P(N to V) =0.25

Part 3: "What/W a/D cute/J cat/N!" Since we have zero initial probability for "What/W" and any transition, the HMM probability should be 0 too.

Question 4: Attention

(i): First, we can define multiple important word vectors, such as "meeting", "traveling" and calculate the inner product between each input word and important word vectors. Then for each word, we get a new product vector whose length is equal to the number of important words. The new product vector can be treated as a new embedding of each word, and then we can add another attention layer on the new embedding to check whether the word focuses more on "meeting" or "traveling". In the end, we use a softmax function taking the output of the second attention layer and giving us the probability for multi-classification. Since we may need multiple actions for one message, we need to set a threshold for the probability to trigger the action. The threshold may include absolute value as well as a relative value. For example, if the probability for "meeting" is 0.5, "traveling" is 0.25, and the absolute threshold is 0.2, then the user should have two links for booking a meeting or traveling. We may also set a special category representing "nothing needs to be done".

(ii): The time complexity for the model output should be linear to the number of words, but I think 100-1000 words should be enough, because we only need to take those related to specific action into consideration. For each action, such as "book a meeting", we can use the words which always present these actions together as important words.

Long Answer

Question 1: Bringing history to life

In this project, we'll first develop an annotation guideline to label the text from letters manually as training, validation and test data of a model. The input format of the data should be every letter, while the output of our model should be a vector with elements representing how each sentence is related to the civil war in the context of the letter. The length of the vector is a constant, so it may not work well for some exception cases where the letter is extremely long. For a letter somewhat relevant to the war, the output for each sentence can be a number between 0 and 1. For irrelevant sentences or padding sentences, we will only have 0.

For the annotations guideline, it'll be much easier to mark something as related instead of not related because the search space is much smaller for related text. To mark any text as related, there should be some words related to the Civil War presented in the paragraph, such as "soldier", "troop" or some entity name such as a general's name or some location. In addition, to find high-quality letters, we may need to analyze the emotion behind the text. A straightfoward record of the war won't be very reflective although it has a lot of related words while a paragraph describing how afraid the soldier is in the warzone can be very reflective. Therefore, we also need to consider the relationship between the words and the content of the letter. Therefore, we need to consider if the content of the letter shows the emotional feeling or the inner activity of the writer. Since typically we have the same emotion for the same paragraph, we will rate the emotional feeling of each paragraph by one rating. Therefore, our final rating for each sentence in the letter is (relevance rating of the sentence * emotional rating of the paragraph)

The relevance between the guideline and the war can be predicted by bag of words algorithms such as logistic regression, while the emotional feeling of each paragraph can be predicted by a sequence model such as LSTM. The hidden state of LSTM can somehow represent the emotional state of the writer.

However, we may have other sentences with important meaning but don't have these words presented. Hence, we need some ways to find out the relationship between sentences. Therefore, we can use dependence parsing, discourse analysis and attention to find out the relationship between sentences. If sentences have strong attention or dependence on one which is quite related to the war, we can cluster them together or use an edge to connect the sentence together like a graph. The value of attention or dependence showing how strong they're connected.

After the connection, sentences closely connected together can be regarded as clusters. Then, we can use pos tagging to extract the important verb, entities from the text, and get the main idea of the reflections or anecdotes. We can rate the clusters or entire letters by ranking the average rating of each sentence as well so that we can rank them and choose the top one for exhibition.

Question 2: Better, more private email auto-complete

Since we only have limited data for training, we can build a template for the suggestion and fill it by the context. Therefore, our model can be composed into two parts:

1. Generate the suggestion template based on the emails and users' input. For example, if we received an email with the words "Congratulations", we should generate a template such as "Thank you, I'm honored for xxx".

2. Identify the context information which will be used to fulfill the template. In the last example, there should be some entity or noun presented in the email, therefore, we need to extract the key information from the email and replace "xxx" with the word.

For the template generation, we don't require a lot of data from the users since typically we share a common way of writing email. We may also use other email records as training data. In this model, the input should be any email that the user received, and the output or label should be a template with entities masked. To generate a suggested template, we may use a LSTM model for a multi-classification. The output of the model should be the probability of choosing each template candidate as the reply template.

For the context information extraction, we can use pos tagging, attention mechanism and dependency parsing to find the important entities and relationship between them. Typically the noun attracts a lot of attention and the verb that connects them will be frequently used to fulfill the template. Following the previous example, for an email "Congratulations, you're admitted by UMSI", the model should be able to extract the verb admitted and UMSI. Then, the model will consider the relationship between admit and UMSI by dependency parsing to determine the prepositions, so that it can be put into the template so that we have "admission from UMSI" for "xxx". For the input, we need to take two parts into consideration, first is some content from which we need to extract entities, second is the template generated from the first model. The model needs to know how to fulfill the template properly based on the relationship between the words. In this way, we can use any context containing some entities and predefined template as the trading data, while the output can be a sequence of vectors, which is the probability of each word being put into the slot.

For the evaluation of the template generation model, it's essentially a classification problem, hence we can use F1 score to measure the performance. For the entity extraction model, we can use perplexity to check how surprised the model is of the entity presenting in the specific slot of the template.

Question 3: Comment your code

To determine the quality of comment, we should firstly need to know which part of the codes the comments are talking about. Since we know the liner number of the comment as well as the codes, we can apply an attention mechanism or similarity between the comments and the codes. The lines which catch the highest attention from the comments can be the target.

Not every part of the code needs comments. Some comments may be very helpful for understanding while others may be redundant. Therefore, based on the structure of Python code, we will decompose the codes into blocks or functions, and check if there are comments targeted for the blocks or functions. If we have the comment targeted to the block, we'll evaluate the quality of the comment based on the codes.

If we have some sub blocks in bigger blocks, such as if statements in a loop, we may need to aggregate the rating of subblocks to generate the output of outer blocks. What's more, there may be some dependency between blocks or functions. A block that only calls other functions may still require comments. Therefore, the quality of the comment for a block can be dependent on the block itself, the sunblock in it, and the block that its code depends on.

When we know the relation between code and comments, we can evaluate the quality. A good comment should be able to explain the necessary part of the code and be succinct enough. Therefore, we may train two models aiming at understanding the comments and the code and express them by throwing out a state vector. If the vectors are similar enough, meaning that both models are understanding the context well.

Therefore, our system should have 3 parts. The first one should find out the dependency graph between different codes and identify the scope of each comment. The second and third one should take the code and comment of each block as an input and output a state vector to describe the content of them. Then we can evaluate the quality of the comments by comparing the cosine similarity between the two state vectors. The whole quality rating can be the aggregated score based on the vectors.

For the first model, we need to manually label the scope of each comment, and we can use total variation to calculate the overlapping of the label and the output of the model. If there is a lot of overlap, then our model is good at finding the target of the comments. For the second two models, we're going to use LSTM to predict the input code or comments so that we can check if the latent state can grasp what the codes or comments are about. To evaluate the performance, we can use F1 score to see if our LSTM model can make a good prediction based on the input. If all three parts of the system can work well, then we can generate the final score of the comments by aggregating them based on the dependency graph.

Question 4: Tinker Tailor Soldier Spy

The key of this project is to capture the meaning of the spy's message and decorate it with another writing style. Since we already know the estimated length of the input, we can use a seq2seq model with attention to complete the task. The input should be the distributed representation of the word with the position embedding, and the encoder should be able to turn the paragraph into a state vector. To evaluate the performance of the encoder, we can train another decoder and see if it can restore the original input.

If the encoder can capture the meaning well, we need to train another model which is good at decorating the state vector into text in another style. For the training data, we may need to manually create some text in another style with the same meaning as our label. What's more, we may want to decorate the text in different ways, hence we can add a random input to decide what kind of decoration that we want.

In short, we'll use neuron networks with attention to make predictions on output words based on the input text. Since the prediction is essentially a classification, we can use Softmax as our loss function and F1 score to evaluate the prediction. We necessarily need spy's text as training data but any data with different writing style but same meaning.

The risk and benefit actually depends on how the firm is going to apply this technology. For example, if our spy is acting as an undercover agent in some terrorists, then my system may save people's life by help the spy to transmit important messages. However, if the spy is trying to steal information for some malicious purpose, then it may be harmful for people. On the other hand, the system may be a recreational tool which can change people's style. It'll be fun to see how people's words are represented in other styles.

Therefore, we need to set up an evaluation procedure for NLP models to see their potential. Not for their performance but their potential to harm or benefit the society. In this example, we need to evaluate how many people will be affected or how much money will be lost if such a model is used for stealing secret data. Although it's hard to estimate because we don't know how many spies are there, setting regulations for the application of NLP and evaluating its impact can always be helpful. Otherwise, as long as the system is powerful enough, people will try to exploit it for their own purpose at all costs.

Personally, it's impossible for me to take this role without doing a careful background search on the company. What's more, we need to make sure the system is used in the right way. Because the ultimate goal for NLP and other technology is to make the world better.