5-2019

SYNTHNOTES: TOWARDS SYNTHETIC CLINICAL TEXT GENERATION

Kris Allen Brown
University of Tennessee, kbrown42@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Kris Allen Brown entitled "SYNTHNOTES: TOWARDS SYNTHETIC CLINICAL TEXT GENERATION." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Gregory Peterson, Major Professor

We have read this thesis and recommend its acceptance:

Michael Berry, Edmon Begoli

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

SYNTHNOTES: TOWARDS SYNTHETIC CLINICAL TEXT GENERATION

A Thesis Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Kris Allen Brown
May 2019

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Gregory Peterson, for his time, patience, and guidance through every stage of creating this thesis. Without his aid and mentorship this thesis would not have been possible. When Dr. Peterson first agreed to take me on as a graduate student I only had vague ideas of wanting to conduct research in natural language processing. Through his efforts I was allowed to join the research team at Oak Ridge National Lab where I was able to work on natural language processing and generation as part of the Open Source Lab there. So thank you, Dr. Peterson, for your faith in me over the many months.

I would also like to show my appreciation for Dr. Edmon Begoli, another member of this thesis committee. Through his guidance and collaboration this project was conceived and he played a central role in driving its evolution over the last year.

Thank you to Dr. Michael Berry for his generosity in agreeing to dedicate his time to serving as a committee member for this thesis.

A significant amount of work presented in this paper was only possible through the resources and expertise provided by Oak Ridge National Lab and specifically the Open Source Lab research team. In particular, I would like to thank Josh Arnold for his knowledge and guidance in setting up much of the infrastructure and automation involved in creating SynthNotes. I would also like to thank Ben Mayer for his mentorship, assistance in planning the software architecture, and his time in editing drafts of this work.

ABSTRACT

SynthNotes is a statistical natural language generation tool for the creation of realistic medical text notes for use by researchers in clinical language processing. Currently, advancements in medical analytics research face barriers due to patient privacy concerns which limits the numbers of researchers who have access to valuable data. Furthermore, privacy protections restrict the computing environments where data can be processed. This often adds prohibitive costs to researchers. The generation method described here provides domain-independent statistical methods for learning to generate text by extracting and ranking templates from a training corpus. The primary contribution in this work is automating the process of template selection and generation of text through classic machine learning methods. SynthNotes removes the need for human domain experts to construct templates, which can be time intensive and expensive. Furthermore, by using machine learning methods, this approach leads to greater realism and variability in the generated notes than could be achieved through classical language generation methods.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AMR	Abstract meaning representation
BLEU	Bilingual Evaluation Understudy Score
cTAKES	clinical Text Analysis and Knowledge Extraction System
CID	Cluster Identifier
CUI	Concept Unique Identifier
HDFS	Hadoop File System
HTML	Hypertext Markup Language
ICD-10	International Classification of Diseases, 10th Revision
METEOR	Method for Evaluation of Translation with Explicit ORdering
MIMIC	Medical Information Mart for Intensive Care
LSTM	Long short term memory
NER	Named Entity Recognition
NLG	Natural language generation
POS	Part-of-Speech
RNN	Recurrent neural network
TF-IDF	Term frequency-inverse document frequency
UMLS	Unified Medical Language System
VCPU	Virtual central processing unit
XML	Extensible Markup Language

CHAPTER 1
INTRODUCTION

The software tool presented here, SynthNotes, aims to provide a generation tool for medical text corpora which can be used by researchers studying clinical natural language processing (NLP). Currently, computational methods for studying healthcare data are largely restricted to structured electronic healthcare records which leaves a wealth of untapped analytical potential locked in the clinical text of records. Progress in clinical NLP is severely limited by the inherent risks of sharing sensitive medical data containing personally identifiable information. In particular, the present crisis of suicide rates [1] among veterans in the United States presents a pressing need for greater research and understanding of clinical text in the mental health domain. The work here seeks to directly address this barrier to progress by providing a tool for developing research datasets free from privacy concerns.

Historically, institutions have made health data available after de-identification, anonymization, and other techniques. However, these tasks suffer from several limitations relevant to researchers attempting to build high-precision analytical models. Moreover, although significant effort has gone towards releasing structured health information, the availability of high-quality clinical text is sparse at best. This is even more true of text in the mental health domain due to the increased risk of exposing private information.

The lack of available textual mental health data and the surrounding privacy concerns present an opportunity for advancing this area of research through synthetically generated clinical notes. Realistic clinical notes offer a wealth of benefits to both researchers and medical institutions.

Classic natural language generation (NLG) took the form of a three-stage pipeline: document planning, microplanning, and sentence writing [2]. These systems usually relied on a large set of complicated rules, developed by domain experts, who would create document plans based on a knowledge base. These systems are commonly seen today in financial and sports reporting publications. While useful in a highly

constrained domain, they are costly to create and do not generalize well to other domains. Instead, new systems and sets of rules must be created for each new domain or writing task. This is equally true for sub-domains within the medical field which often have their own text formats, styles, and sub-languages.

Recent developments in NLG have taken advantage of statistical and machine learning approaches to reduce the fragility of classic NLG systems. Statistical systems are cheaper to develop, robust in their adaptability to new domains, and given enough data can provide a high degree of coverage of the language features in the corpus. However, statistical systems tend to suffer from high error rates and low coherence in the synthesized text. Thus, a combination of templates and statistical methods provides benefits from both paradigms of NLG.

SynthNotes takes advantage of past research [3, 4] in such hybrid NLG systems. Rather than solely relying on human experts or an automated learning agent, a hybrid system utilizes an existing corpus of human generated documents and learns to write modified templates. Hybrid systems like this offer the realism and precision of many domain experts while also utilizing the scalability and variation of statistical systems.

Kondadadi (2013) demonstrated the success of a hybrid system as applied to the weather and biography domain. Their work split documents into sentences, extracted named entities, and replaced the entities with fillable slots to be later filled by a document generator. Clustering is performed to group conceptually similar sentences together. Once clustered, the system learns to choose or rank templates to write to new documents. SynthNotes takes this approach and applies it to the medical domain.

The primary contribution of this thesis is a full hybrid NLG system capable of processing a corpus of clinical text, extracting templates, and learning to write medical notes without the involvement of human clinical practitioners. Language processing and generation of clinical text presents new issues not typically encountered in other more common domains. Many of those challenges are addressed in this thesis and a path towards future progress is discussed.

1.1 Related Work

Natural language generation has been a key goal in computer science and artificial intelligence research for decades. Among the most famous examples is the ELIZA [5] chatbot developed at the MIT Artificial Intelligence lab in the 1960's. Using only the

simplest language processing tools, pattern matching, rules, and scripts, the ELIZA program was able to interact with humans effectively enough that end users often ascribed human-like feelings and emotions to the software agent. Over the following decades, NLG applications have moved beyond simple chatbot programs and have taken over human writing tasks such as sports, weather, and financial reporting, translation, document summarization, and patient healthcare summaries.

In the late 1990's Reiter and Dale [2] laid out a conceptual architecture for language generation systems that could be used as an agreed upon starting point for building NLG systems. In their work, the describe what has become known as a classic or consensus architecture for NLG software. They frame the NLG process as a pipeline of three fundamental tasks: document planning, micro-planning, and surface realization. Each stage of the pipeline contains tasks that can be split as content tasks and structural tasks [6]. These are further described below. An overview diagram of the pipeline system is presented in Figure 1-1.

The classic NLG pipeline described by Reiter and Dale [2] relies heavily on research in software data structures, knowledge bases, expert systems, and areas of linguistic study such as grammars, discourse theory, and rhetorical structure theory.

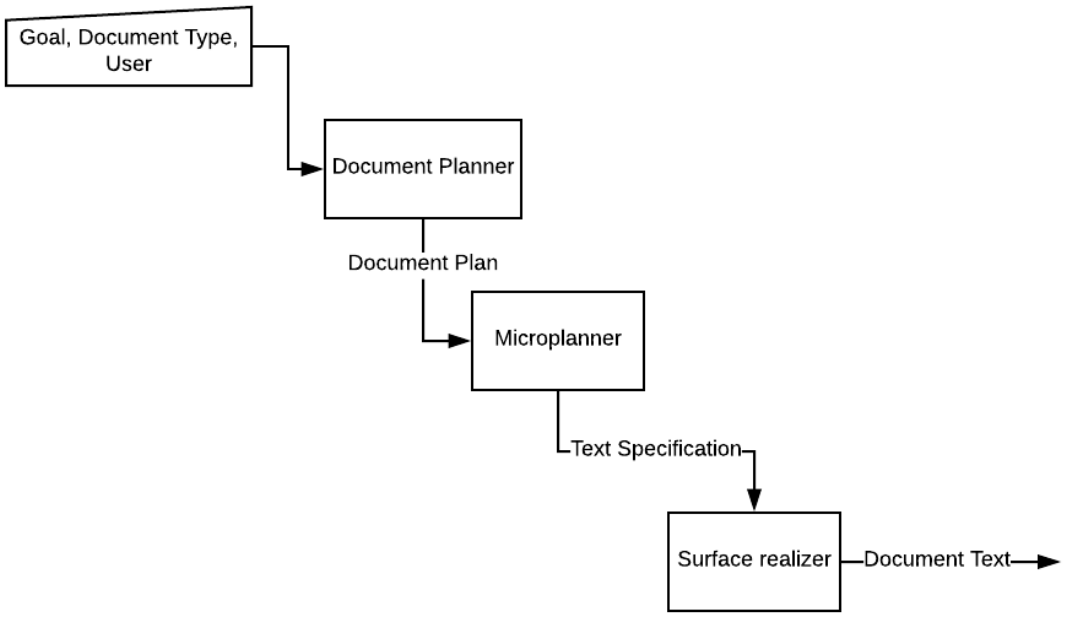

Figure 1-1. Classic NLG pipeline architecture.

The first stage of the pipeline, document planning, is responsible for determining what information is most significant and should be conveyed to the user based on the communication goal, document type, and a user model. The document planner should make different choices based on the previous factors. For example, different content determination paths would be taken for a document aiming to inform a domain expert, such as a hospital physician reading patient summaries, versus a software agent generating a pamphlet which seeks to persuade a patient to stop smoking. The document planner is also responsible for organizing the structure of a generated discourse.

The microplanner, the second pipeline stage, moves down one layer of abstraction from the document planner and determines the words and syntactic structures, lexicalization, that should be used for writing the content chosen by the previous stage. The microplanner is often where operations are performed that contribute to the naturalness and fluency of generated text. Specifically, the microplanner will additionally perform aggregation and referring expression generation. Aggregation is a structural task in which the program determines how the concepts and structures from the document plan should be realized as linguistic elements [2]. That is, sentences and paragraphs. It is often more fluent to a reader if multiple concepts or information elements are grouped together into a sentence joined by a conjunction rather than in multiple shorter sentences. Referring expression generation decides how previously mentioned entities are referred to in the text. It is more fluent to refer to named entities as 'she' or 'it' rather than always reiterating a specific name. Referring expressions constitute a content task of the microplanner.

The final stage is the surface realizer which is responsible for generating the final text form that is written to the generated document. The content task of the realizer, linguistic realization, converts the abstract sentence representations from the microplanner into real text [2]. This often involves grammar rules created for the application domain or derived from other work. Structurally, the realizer converts the abstract structural information from the previous stage and creates symbols which can be parsed and understood by document presentation tools [2]. If we consider a document written for a website in HTML rather than plain text, there is a clear syntax for indicating different document structures such as section headers, paragraphs, and tables. These would all be handled by the surface realizer.

The architecture as described by Reiter and Dale defines passing plans between each stage of the pipeline as a tree data structure. For the document plan, internal leaf nodes represent structural information and leaf nodes are *messages*. The idea underlying messages in this paradigm is that any language generation domain can define entities and relations which when taken together encode the domain's particular concepts [2].

The encoded concepts are then mapped into linguistic forms in later pipeline stages. So, messages are a way to map abstract domain concepts into final textual forms. In some instances the messages are immediately apparent, for example in the medical domain, where defined ontologies and knowledge bases exist.

The document planner passes the above described tree structure to the microplanner which then passes another tree to the surface realizer. The microplanner output represents internal nodes as structural information and leaf nodes as sentence representations [2].

Several notable systems have been produced using the fundamental ideas laid out by Reiter and Dale which are worth mentioning here. Goldberg, Driedger, Kittredge, and Richards (1994) developed the FOG [7] system as a tool for automatically generating multilingual weather reports based on a specific geographic location. In FOG, numerical data from weather simulations is input into the system and a text summary of the data is produced. Although FOG is not a tremendously exciting system, it demonstrates useful content decisions based on regions and forecast type. For example, when FOG is fed data in a marine area it will emphasize some data, such as wind speed and precipitation, over other data that may be less useful. It is also able to make choices on aggregating forecasts for related geographic regions when appropriate. It is also able to take abstract representations from a microplanning stage and generate reports in either French or English.

PIGLIT [8] is another system developed under the classic pipeline architecture by Cawsey, Binstead, and Jones (1995) for the purpose of providing personalized explanations of a patient's medical record. The goal was to create a new type of generated document that was more understandable and useful for patients than a collection of lab notes and physician reports. Having a generation system such as PIGLIT to perform such a task is highly beneficial as the medical community is notoriously overwhelmed already when it comes to writing reports. A key feature of PIGLIT's content was hyperlinks embedded in the text which allowed patients to further research their conditions, medications, and treatments. Underlying PIGLIT is a

knowledge base curated by the application developers. This is similar to the UMLS [9] or ICD10 [10] coding systems which are well known in the United States' healthcare environment.

STOP [11], developed by Reiter in 2000, generates personalized pamphlets to persuade readers to stop smoking. Users of the system voluntarily take a questionnaire on topics such as health problems, previous smoking cessation attempts, and their likes and dislikes about smoking. Using this information, STOP creates text tailored to the users background and user profile. For instance, in one example pamphlet, a user is prompted to try nicotine patches again despite them not working the last time they attempted to quit [2]. The unique aspect of the STOP system is the way it integrates user knowledge, domain experts, effective writing techniques, and research on the psychology of behavior change to create a persuasive and ideally useful document for the reader.

A more recent example system with high relevance to SynthNotes is BT-Nurse [12] developed by Hunter et al. (2011). BT-Nurse was developed to create shift summaries in a hospital's neonatal intensive care unit (NICU) based on electronic medical records. BT-Nurse is an example of a system that takes in heterogeneous data and makes choices as to which information is most relevant. It is even able to take physiological measurements such as heart rate and express significant events in natural language. As with previously mentioned systems, BT-Nurse relies heavily on a medical ontology and rules developed in collaboration with neonatal experts.

It is clear from the above discussion of production systems using the consensus NLG architecture that the approach and design is useful and has many benefits. Its modular approach allows for a separation of concerns between each stage of the pipeline that can be used to great effect for any given application. Developers, in collaboration with domain experts, can exert fine-grained control in generating high quality texts. It is also a well studied and understood design and each component performs individual tasks with decades of existing research behind them. However, for clinical note generation of the type SynthNotes is designed for, the pipeline approach is not a viable option.

There are serious concerns with the pipelined approach to NLG. First, it is not a general system that can be applied across domains. While the architecture is domain independent, the actual work that each module performs much be customized to match the application. This presents a problem in constantly having to rewrite major sections of the software for each new application despite each one performing similar functions.

The messages abstraction presented in the discussion on document planners is a clear example of this problem at work. Even if every other component of the system was able to be reused, messages are inherently domain dependent as they capture the concepts and relations contained within it. In a large domain, significant work is involved just in constructing a set of reasonable messages. In a weather system like FOG the list of possible messages is tightly constrained. However, for a system in the medical field the count of messages could easily explode and completely overwhelm the available human resources available to create them. Thus, despite being effective text generating agents, pipeline systems are inherently rigid and constrained in their coverage of language both inside and outside the domain area.

Second, as discussed in the example systems, pipeline applications rely heavily on domain experts. For example, the document planner in any system relies heavily on input from domain experts who are familiar with common practices and document structuring in the targeted field. PIGLIT, FOG, BT-Nurse, and STOP all relied on close collaboration between developers and domain experts. It is also clear that the messages paradigm is both time consuming and heavily reliant on expert input in the same way. In the case of FOG, numerical data fed into the program had to be annotated by weather experts [2] and STOP made use of various experts in psychology and behavior change [11] for planning and writing effective documents. PIGLIT also required customizing a medical knowledge base [8] in order to provide information to patients. Knowledge bases also present a unique challenge in that they are not available for most domains. Furthermore, finding and hiring domain experts adds additional costs in money and time to developing a production language generation system.

The previous two problems with the pipeline approach create challenges for using that architecture in the SynthNotes application. SynthNotes had to be developed with cost and time constraints and with minimal expert involvement. The four example systems also have distinct and specific communication goals for their audience. SynthNotes sits in a unique position as its goal is to reduce the barriers to research for medical NLP practitioners. In a sense, SynthNotes attempts to provide a suitable discourse for processing by other programs, parsers, information extractors, etc., rather than information consumed by other human readers. Finally, the example systems are often constrained even within their domains. The STOP system applies to quitting smoking but does not attempt to create a general method for other areas that could benefit from behavior change literature such as diet or exercise. BT-Nurse suffers from the same

specialization as it was developed for a specific ward in a single hospital. SynthNotes needs to cover the range of medical discourse encompassing numerous sub-domains which would each require their own expert input which would further increase time and cost.

These are known problems with the classic pipeline approach to NLG. As a result, more recent research has moved to take advantage of machine learning and deep learning methods as a way to generalize the task. We will first turn to past work in machine learning before transitioning to the most recent work in deep learning approaches.

The addition of machine learning methods to the task of natural language generation offers many options in how developers and researchers approach the task. Some work focuses on using statistical methods as a module in the previously discussed pipeline architecture, while others have combined multiple stages into a single step or created and entire end-to-end system.

Barzilay and Lee (2004) demonstrated how to create a domain-independent statistical model for the task of content planning. Their approach modeled text documents as a sequence of text spans or sentences, which are defined as "topics", and learns the likelihood of topic transitions throughout the document [13]. This assumed a singular content model for each document which they define as a Hidden Markov Model. In their HMM, the hidden states correspond to topics which are able to produce topic-specific sentences. In order to define a topic, Barzilay and Lee cluster corpus sentences according to the cosine similarity of the sentence bigrams [13]. Thus, rather than relying on domain experts or input data, they create a document planner which is adaptable to the training corpus.

Machine Learning in NLG often takes the form of data-to-text where structured database records are paired with their corresponding output text. The most popular examples in the NLG literature are those of weather forecasts and sports summaries. These types of copora can be incredibly useful for training statistical models that learn to generate text from data. For an example of aligning data and text see Table 1-1.

Liang et al. (2009) took advantage of aligned data-text corpora to create a system of grounded language learning [14]. Grounded language learning refers to the process of learning the meaning of a sentence based on an observed world state [14]. In the case of an aligned corpus the world state would be the structured database records. Given

Table 1-1. Example of database records with their aligned text. Adapted from Gatt (2018)

Record	**SkyCover**	**Temperature**
Field	percent=0-25	time=6am-9am min=9 max=21
Text	cloudy,	with temperatures between 10 20 degrees [...]

the aligned data and text, Liang et al. created a generative model, $p(w|s)$, which learns the correspondence between a text, w, and a world state (set of database records), s. Their generative model takes three steps. First, the model selects a sequence of records from s. Then, it selects a sequence of fields from each chosen record. Finally, for each chosen field a number $c > 0$ is selected and c words written to describe the field. The final text w is the concatenation of all field descriptions generated in sequence. The record and field choices are made by Markov models on the record and field types, respectively. The word choice model is a probabilistic model conditioned on the field type and field value. By using their generative model, Liang et al. created a single probabilistic system that handles all three stages of the classic pipeline.

Extending and modifying the work of Liang et al. (2009), Konstas and Lapata (2013) use the data-text alignment model from Liang, but rely on a Probabilistic Context-Free Grammar (PCFG) for the text generation rather than a Markov model [15]. By using a set of probabilistic rewrite rules, which when followed by the generation algorithm creates a parse tree which can be traversed for generating text.

The probabilistic rules defined by Konstas and Lapata are meant to model transitions from non-terminals (records and fields), to terminals (words) [15]. In their framework, the rules are not domain dependent and should be able to describe any database structure.

Kondadadi (2013) took a different approach than what has been discussed previously, instead choosing to use templates, classification, and ranking for combining content selection and text realization [3]. Kondadadi (2013) extends the work from Howald et al. (2013) which presented domain specific clustering as a way to identify concepts [4], or topics as was the case in Barzilay and Lee [13]. Howald's approach to clustering went a step beyond Barzilay and Lee who only used sentence bigrams and cosine similarity for clustering. Howald (2013) and Kondadadi (2013) redefined sentences as a combination of domain specific named entities and predicates. The redefined sentences are then clustered to yield groups of semantically related sentences. Once the corpus

sentences have been clustered into their concepts or topics, sentence templates are fed into a ranking SVM which selects the best template given the current sentence position in the generated document. In Kondadadi and Howald, sentence templates are original sentences with the named entities removed. See Example 2.0.1 for example sentences and their related templates.

Machine learning offers many advantages over the classic architecture to natural language generation. It can consolidate multiple stages into a single model and offers more generalization over that of the modular pipeline approach. Deep learning offers methods that can expand further on machine learning research.

Deep learning and neural networks have risen and fallen from popularity multiple times over the past decades. With respect to NLG, neural networks have been used as far back as Kukich in 1987 where she lays out advantages of connectionist architectures [16, 17], neural networks, over that of the messaging paradigm found in Reiter and Dale's architecture and demonstrates those advantages through two natural language generation tasks. Many of the ideas laid out by Kukich are still relevant today in NLG. In particular, Kukich notes the ability of neural networks to find semantic representations of texts.

Over the intervening years interest in deep learning has fallen and risen again in large part due to advances in hardware and processing capabilities of modern computing systems [17, 18]. When it comes to natural language processing, deep learning has made hugely influential advances in research, especially with regards to representation. Mikilov et al. (2013) demonstrated the ability of a neural network to learn dense, low-dimensional, and distributed representations for phrases [19]. In this process, words are converted into a vector representation which places similar words close together in a vector space [19]. This work laid the ground work for deep learning researchers to make use of abstracted knowledge and meaning representations which also captured semantic and grammatical information of text spans in a training corpus. Mikolov's algorithm, commonly known as word2vec, is far more valuable than the standard language models of the past which relied primarily on n-grams.

Directly applicable to NLG is Sutskever et al. (2011) who presented work on using a more advanced neural network architecture, LSTMs, to create a character-level model for generating sentences based on training corpora from Wikipedia, New York Times, and academic machine learning papers [20]. LSTMs, Long Short Term Memory, are a variant of RNNs, Recurrent Neural Networks. Each model is able to

learn representations of long-range sequential information. In this case, sequences of text tokens. As implied by the name, LSTMs add 'memory' units to an RNN which are able to store information over long sequences of data [20]. Sutskever et al. is an example of using deep learning for surface realization but it can also be used to generate text based on semantic representations of inputs [17].

One problem with standard feed-forward neural networks is that they are unable to efficiently handle input modeled as a sequential data of unknown length. Machine translation is an excellent example where the input and output can vary depending on the sentence. Sutskever et al. (2014) proposed a solution using a pair of RNNs whose functionality is split [21]. One RNN serves as an encoder which transforms a sequence of input data into a vector representation. The resulting encoding can then be fed to the decoder RNN which predicts an output sequence based on the encoded input data. Sutskever et al. (2014) demonstrated the effectiveness of this architecture by applying it to an English to French translation task.

Using a similar encoder-decoder architecture as Sutskever et al. (2014), Castro Ferreira et al. (2017) apply the architecture to language generation using abstract meaning representations (AMRs) [22] mapped to English output text [23]. Abstract meaning representations were designed to provide structured semantic meanings of sentences in order to aid the progress of research in natural language understanding and generation [22]. They are valuable for researchers in that they abstract out the syntax of a sentence so that sentences which have similar meaning should have the same AMR. AMRs are also traversable as a graph which makes them reasonably easy to read by software programs. For an example of an AMR and its possible text representations see Example 1.0.1.

An attention mechanism was also added to the network by Castro Ferreira et al. (2017). Attention mechanisms in neural networks such as RNNs and LSTMs allow the network to give weighted importance to specific input fields when predicting certain segments of the output [17, 24]. As an example, consider Example 1.0.1. When the network is predicting the verb in the sentence "The boy doesn't have to go", it should assign a high weight to the verb "obligate" as well as the polarity field in order to negate the obligation of the boy in the sentence. Such an attention mechanism removes the need for explicit alignment that was discussed in previous work by Liang et al. (2009).

(p / obligate-01
:arg2 (g / go-01
:arg0 (b / boy))
:polarity -)

The boy doesn't have to go.
The boy isn't obligated to go.
The boy need not go.

Example 1.0.1. An AMR example and its related text realizations. Notice that from a single AMR multiple realizations are possible that capture the same semantic meaning. Example adapted from Banarescu et al. (2013).

Machine learning and deep learning approaches offer significant advantages over the pipeline architecture. The methods are more general and can be created with greater efficiency in both time and cost. However, in general they both trade accuracy for efficiency. Whereas pipelined architectures are heavily controlled through rules and grammars created with domain experts, machine learning (ML) and deep learning (DL) methods attempt to learn from data. In order to do so, these algorithms tend to be extremely data hungry. This is especially the case with deep learning algorithms. This presents a challenge in NLG systems for medicine. Training data is hard to acquire due to concerns surrounding patient privacy, and when it is available the datasets are generally quite small.

CHAPTER 2
APPROACH

2.1 Overview

The general strategy for note generation in this thesis seeks to strike a balance between classic fully human created systems and newer fully automatic deep learning approaches to language generation. NLG systems created by human domain experts are expensive to create in terms of cost and time. Moreover, it is difficult or impossible for such systems to capture the full breadth of variability found in some domains, the medical field being one of them. On the other hand, fully automated learning agents tend to have a high level of error and struggle to capture long term language dependencies, logic, and the requisite knowledge base necessary for coherent and realistic text generation. The method described here fits between the two extremes. This research uses statistical techniques to learn what to say and when, but relies on the wealth of high quality human generated notes to figure out *how*.

At the highest level, SynthNotes extracts *semantic templates* [3, 4] using existing medical NLP software and learns to rank the best template to use based on available information and sentence position at generation time. Below are some useful definitions for understanding SynthNotes:

- **Semantic Templates:** A simplified version of a sentence from the original note text. All tokens are stripped away except for: medical concepts identified by cTAKES and extracted predicates.
- **Templates:** Original sentence from a note with fillable slots in place of the original extracted medical concept text. Slots are filled at generation time.
- **Conceptual cluster:** Clusters of semantic templates. Each cluster should have semantic templates assigned to it which all convey a similar semantic concept.

The first step in SynthNotes for learning to generate clinical notes begins with processing notes from the MIMIC-III database [25] through Apache cTAKES [26].

cTAKES is an NLP framework commonly used by medical researchers for standard language processing tasks in addition to medical information extraction. cTAKES is primarily used to extract sentences, tokens, predicates, and medical concepts, which are used to replace the named entities in Kondadadi (2013). The information extracted from cTAKES is the cornerstone for creating templates, used in generating notes, and semantic templates, used for assigning an abstract concept to a sentence.

Using the information extracted from the clinical notes, semantic templates are formed. A semantic template is simply the predicates and named entities from the original note sentence with all other tokens removed. It is then necessary to group semantic templates together that convey similar conceptual meanings. Here, the k-means [27] clustering algorithm is used. In addition to creating semantic templates, standard sentence templates are formed for use during note generation. A regular template is a sentence with the medical concepts, or named entities, removed and replaced with a slot to be filled with other medical entities by the generator. Examples of original sentences with their corresponding template types can be found in Example 2.0.1.

Sentence
a. RLL atelectasis and labs were drawn.
b. MRI spine showed a mass in the T10 region.
Conceptual Meaning
c. ANATOMICALSITE | SIGNSYMPTOM | DRAW
d. PROCEDURE | SHOW
Templates
e. [**anatomicalsite**] [**signsymptom**] and labs were drawn.
f. [**procedure**] spine showed a mass in the T10 region.

Example 2.0.1. Example sentences with their templates. Notice that predicates are used when creating conceptual templates in (c) and (d). These are used for clustering sentences. However, predicates are not reserved slots in templates for generation time as shown in (e) and (f). Diagram adapted from Kondadadi et al. (2013).

The core work of the generator in SynthNotes is the learned ranking model. Ranking is a well studied task in other areas of machine learning, especially those related to the relevance of documents or web pages in search engines. In this case, rather than ranking documents based on relevance, the task is to find the most relevant template at a given sentence position during note generation. Following template extraction and clustering, numerous features are created based on corpus statistics and fed into a ranking algorithm. SynthNotes uses the LambdaMart [28] algorithm developed by Microsoft researchers. Following training, all templates in all clusters are ranked and note generation can proceed.

Documents, or clinical notes, are generated sentence by sentence. At each step, or sentence position, a template is chosen from the above model, its empty slots filled with medical concepts, and it is written to a new document. In addition to the ranking model, two other methods are implemented to serve as a comparison. First is a random template selection model which chooses a template at random from the entire pool of extracted templates. This is done without regard for sentence position or clustering and should be seen as a baseline. Second is a frequency selection model. Frequency selection is a simple two step process. It first performs a random weighted selection of a cluster label from which to choose a template based on the frequency with which cluster labels are observed at each sentence position. Next, a template is chosen at random and it is written to the document. This is a small step above the random model

in that it is performing a rough selection based on conceptual meaning. A high level view of this software system can be found in Figure 2-1.

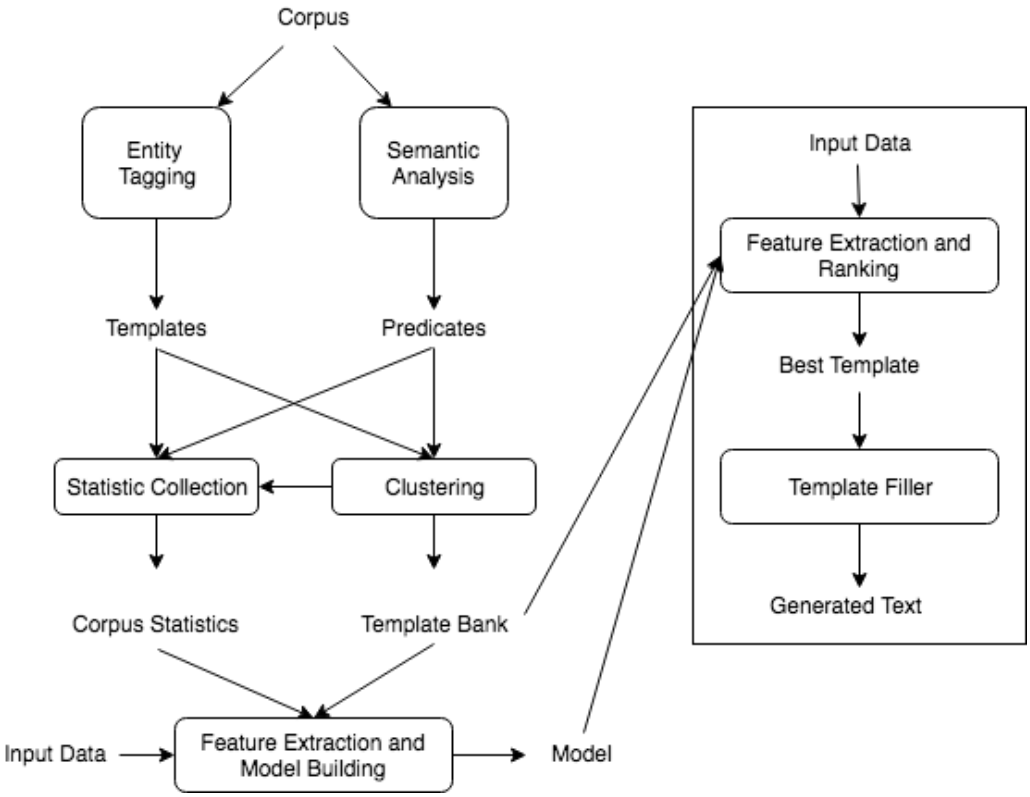

Figure 2-1. Natural language generation system architecture overview. Diagram adapted from Kondadadi et al. (2013)

2.2 Preprocessing

In order to perform the core work of generating clinical text, linguistic features must be extracted from the training corpus and processed so that they are usable for later parts of the system. The tool used for feature extraction is Apache cTAKES. The key features extracted using cTAKES are sentences, tokens, predicates, and medical concepts taken from the Unified Medical Language System (UMLS) [9].

The most common and easiest output format to use from cTAKES without additional development work on the cTAKES framework is XML. As such, a cTAKES XML parser was developed as part of this work which is able to process XML files, extract the necessary features, and store them in parquet files on a distributed file system. The cTAKES and XML components are discussed below.

2.2.1 cTAKES Processing

Apache cTAKES [26] is a natural language processing and information extraction framework developed by the Mayo clinic and commonly used by researchers in the clinical domain. cTAKES uses discrete modules linked together in a processing pipeline to accomplish its varied tasks. It is capable of many common NLP tasks available in other language processing software, such as sentence detection, tokenizing, dependency parsing, and part-of-speech tagging, in addition to tasks specific to processing medical free text. Medically specific tasks include medical concept extraction, smoker status detection, and drug information extraction.

This work uses cTAKES for sentence detection, tokenization, semantic role labeling, and medical concept extraction. Of particular interest is concept extraction which is used instead of standard NLP named entity recognition (NER).

While cTAKES is a useful tool for processing free text, it is constrained by its performance. So, in order to process MIMIC-III notes in a reasonable time, it was necessary to spread the computation across multiple nodes.

The automated deployment tool, Ansible [29], was used for installing and configuring multiple virtual machines in parallel. Virtual machines were hosted in an OpenStack [30] deployment in the ORNL CADES environment. OpenStack is an open-source software platform providing infrastructure-as-a-service (IaaS) tools for cloud computing. CADES is the shared cloud computing resources hosted at ORNL.

Once cTAKES nodes were configured, MIMIC notes were deployed to a centralized storage node. A Python script was used to split the files equally across multiple directories named by the destination machine IP address where they would be processed. An rsync module in Ansible was used to copy directories from the storage machine to a data directory on each of the cTAKES nodes. A final Ansible playbook was used to run cTAKES on each machine.

cTAKES outputs a single XML file for each processed note which contains all the parsed and extracted information to be used for training a note generator. Upon finishing the cTAKES processing, the output files in the data directories were copied to an HDFS [31] cluster so that the XML files could have the relevant features extracted and stored. For a reference of several XML element types output by cTAKES see Example 2.0.2.

```
<textspan:Sentence xmi:id="25" sofa="1" begin="0" end="6"
        sentenceNumber="0"/>

<syntax:WordToken xmi:id="970" sofa="1" begin="150" end="158"
    tokenNumber="35" partOfSpeech="NN" capitalization="0"
    numPosition="0" canonicalForm="facility"/>

<textsem:MedicationMention xmi:id="7820" sofa="1" begin="850"
    end="853" id="0" ontologyConceptArr="7796␣7806" typeID="1"
    discoveryTechnique="1" confidence="0.0" polarity="1"
    uncertainty="0" conditional="false" generic="false"
    subject="patient" historyOf="0"/>

<textsem:ProcedureMention xmi:id="10704" sofa="1" begin="221"
    end="231" id="0" ontologyConceptArr="10691" typeID="5"
    discoveryTechnique="1" confidence="0.0" polarity="1"
    uncertainty="0" conditional="false" generic="false"
    subject="patient" historyOf="0"/>

<refsem:UmlsConcept xmi:id="10866" codingScheme="SNOMEDCT_US"
    code="89187006" score="0.0" disambiguated="false"
    cui="C0458827" tui="T023" preferredText="Airway␣structure"/>
```

Example 2.0.2. Sample of cTAKES XML output parsed by the XML parser. Generally, each element type contains unique attributes that must be extracted. There is some overlap across elements such as *xmi:id*. Others can be grouped together as types. In this example, Medication and Procedure mentions are both of type *textsem*. Several irrelevant fields were excluded in this work such as confidence, polarity, uncertainty, and generic.

2.2.2 Unified Medical Language System

cTAKES identifies five broad categories of named entities: diseases/disorders, signs/symptoms, anatomical sites, procedures, and medications. In addition to the broad category classification, entities contain a wealth of information about the specific medical concept they are identified as. This information comes from the Unified Medical Language System (UMLS) [9]. UMLS is a set of knowledge sources and dictionaries containing a Metathesaurus, semantic network, and specialist lexicon. The largest component is the Metathesaurus which is a large knowledge source of medical terms and concepts. The Metethesaurus also links concepts and terms between dozens

of other medical terminology dictionaries. Each term is identified by its Concept Unique Identifier (CUI). The CUI allows for greater specificity beyond just the broad entities classes that cTAKES identifies. In addition to greater specificity, the Metathesaurus provides recommended text for each term which allows for text normalization if required.

The medical concepts from UMLS are used in SynthNotes to stand in for the standard named entity categories found in other language processing software frameworks. Those frameworks are trained on corpora such as news articles and web pages and perform poorly on clinical text.

2.2.3 XML Processing

XML is not an ideal data storage format. It is unnecessarily large and difficult and time consuming to parse if a parser does not already exist. However, it is the default cTAKES output format. There exist community developed modules for cTAKES to store directly to a distributed file system like HDFS, but these projects are not actively developed and have not been updated in approximately five years. Developing and maintaining cTAKES modules was beyond the scope of this work. Instead, a simple XML processing library was developed for extracting and storing the parsed information from cTAKES in HDFS.

Extracting information from XML files first requires creating a parse tree from the XML text. SynthNotes generates a parse tree using lxml [32]. lxml is a python wrapper around the C library libxml2. It is fast, easy to use, and thread-safe as it frees Python's GIL. This makes it an attractive library for future use when conducting performance optimizations.

In order to distribute the work of parsing files across multiple nodes a messaging queue, RabbitMQ [33], was used. RabbitMQ was chosen for its low latency, documentation, and an easy to use Python library which kept development fast and simple.

The scale of notes being processed only required a single RabbitMQ process to feed tasks to worker processes. There are multiple producer and consumers in this system, each performing a relatively simple and discrete task. To begin, a filename producer submits filenames from the XML data directory stored in HDFS. A parser consumer reads a filename from the filename queue, extracts the file's contents, and parses the

XML into a standard Python dictionary. The dictionary is serialized and sent back to another queue to be processed by an HDFS writer. The final consumer in the system takes the serialized dictionary, deserializes it, and buffers it for future storage. Once a configured number of messages have been buffered, the writer performs a bulk write into parquet files stored on HDFS. Storing data in parquet files offers two main benefits. It is efficiently compressed which reduces storage significantly. Compression rates of approximately 50% were observed during this work. Parquet is also a widely used storage format for various data analysis libraries in multiple languages. SynthNotes uses Pandas, but it could have been another processing framework such as Apache Spark.

Although using RabbitMQ and task queues adds some complexity to the system, it has other benefits. First of all, XML files and the parsed output resides in distributed and replicated storage: HDFS. As a result, producers and consumers can read and write to a central location without coordinating with each other. In order to do so with cTAKES parsing, extra steps were required for partitioning the data and running with Ansible playbooks. Also, using a task queue like RabbitMQ allows the system to scale to much larger datasets as needed with relatively simple software. For a graphic depiction of the preprocessing system see Figure 2-2.

2.3 Template Clustering

Before training the ranking model for template selection, semantic templates are clustered in order to create groups of conceptually related sentences and templates. From a high level this means extracting the entity types and predicate tokens from the cTAKES output and joining the tokens together into a single space-separated string. The idea of taking semantic or conceptual representations is based off the work done by Barzilay and Lee (2004) which was further extended by Howald et al. (2013) and Kondadadi et. al (2013). Barzilay and Lee viewed documents as a sequence of topic shifts whose transition probabilities could be learned. They model sentence concepts by the bi-grams contained in the sentence. Howald and Kondadadi created conceptual templates of sentences based off the predicates and domain specific entities present. Barzilay, Howald, and Kondadadi created domain specific clusters of sentences based on their conceptual meanings which could then be used during generation as a form of content selection. Although the clusters themselves are domain-specific, the method is independent of any domain, provided entities can be extracted from a training corpus.

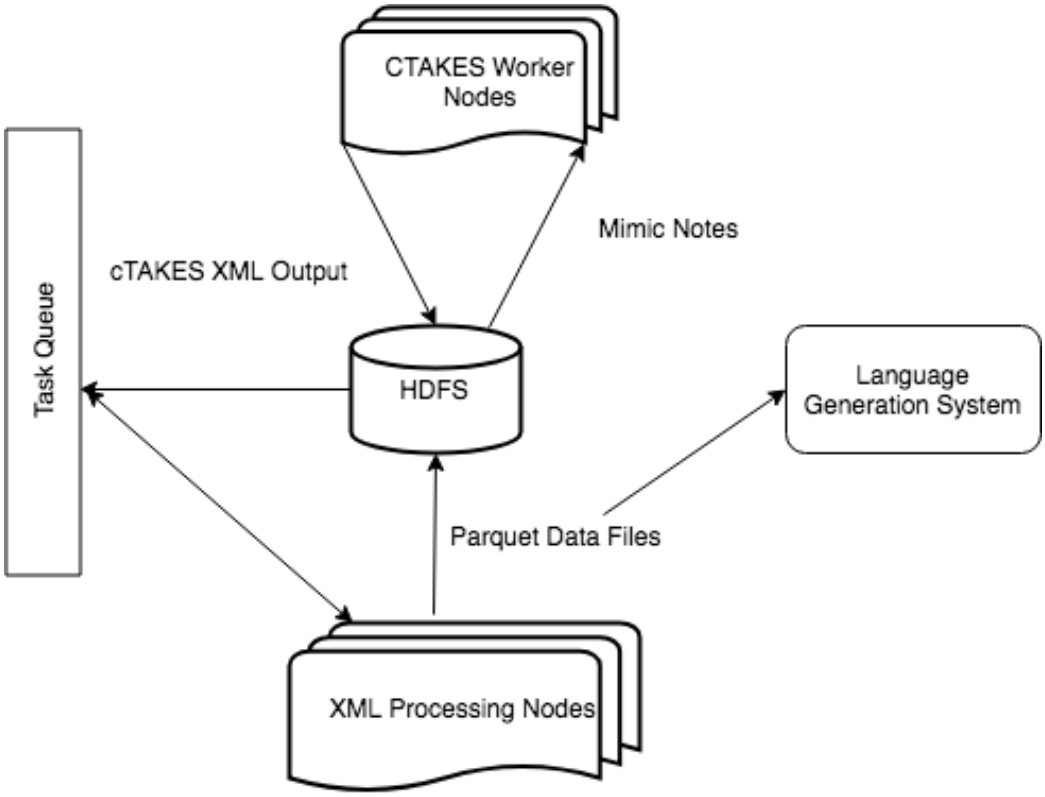

Figure 2-2. High level overview of the preprocessing system which extracts information from the XML files output by cTAKES. MIMIC notes are distributed among the cTAKES worker nodes which process them and send the XML output to an HDFS cluster. XML processing nodes read all the XML files and send the contents to other worker processes which extract the information and store the extracted data back in HDFS as a parquet dataset. The parquet dataset is later used by the language generation system.

SynthNotes follows the same course as these past researchers and applies the method to the clinical domain.

When the set of semantic templates is extracted from the corpus, they are processed using a term frequency-inverse document frequency (TF-IDF) vectorizer in which each template string is treated as a document. The vectors are then clustered using the k-means clustering algorithm [27]. Once clustered, templates are labeled with their cluster ID (CID) and are taken to represent a group of sentences that convey similar conceptual meanings. Sentences that do not contain an entity are removed from the set of templates to be clustered as there are no entities to be replaced in the template at generation time. Sentences without entities contain no medically relevant semantic information that could be parsed.

In the original work [3] on which SynthNotes is based, the authors chose a number of clusters equal to the number of entities in the domain. However, because SynthNotes uses cTAKES for entity recognition, the number of entities is limited to the five broad semantic types discussed previously. On the other hand, UMLS Concept Unique Identifiers (CUIs) are also output by cTAKES, but the number of CUIs is in the thousands, which is far too many clusters. Moreover, CUIs represent a hierarchy of concepts, so the degree of generalization or specificity is highly varied.

There are several methods commonly used for analyzing clusters. For this work the Silhouette Coefficient [34] was chosen. The Silhouette Coefficient is a metric that can be used to identify how distinct, or well separated, clusters are. Values of the coefficient are between -1 and +1. A coefficient closer to +1 would indicate that the proper number of clusters has been chosen and that each group is a distinct semantic concept. Values towards -1 indicate improperly grouped elements, and values near 0 mean clusters are overlapping. A cluster count of approximate 120-125 gave the best Silhouette Coefficient. See Table 2-1 for these results.

Table 2-1. Silhouette coefficients computed from clustering templates. Based on these numbers a cluster count of 120 was selected.

Number of clusters	Silhouette coefficient
70	0.520
80	0.520
90	0.520
100	0.541
110	0.541
120	0.551
125	0.553
130	0.550

2.4 Note Generation

Note generation begins by selecting a document and its accompanying entities from the original corpus. The entities from the document stand in for a database and provide the set of medical concepts to be included in a generated note.

Template selection can be performed in three ways. The first and simplest is a simple random template selector which takes a random sample from the remaining template

bank. The second method is a weighted sampling based on the frequency of cluster labels at each sentence position. Finally, templates could be selected based on a ranking model. Multiple methods for template selection were defined in order to make a more thorough evaluation of the ranking model's performance.

Templates from the template bank that contain entities not present in the document are removed so that templates are not written with blank spaces at generation time. That is, this approach avoids a final generated document with unfilled slots. Templates with entities not present in the document are removed from the candidate pool of templates regardless of the template selection method used.

The random and frequency generators both continue with selecting and writing templates until there are either no more entities left to write about or no valid templates that could be used. The previous two stopping conditions are present in the ranking generator as well. However, the ranking model will not generate more sentences than the max number observed in the corpus. This is done because beyond the maximum sentence number there are no features or training examples.

The next section discusses the various selection methods more in depth and is followed by a brief discussion of how template slots are filled during generation.

2.5 Template Selection

2.5.1 Frequency Selection

Following the clustering phase, a frequency table is created of cluster labels by sentence position across all documents. A weighted sampling is performed on the frequency table to choose a cluster for each position. A template is chosen at random from the selected cluster. Finally, templates have their empty slots filled with entities from the document. When an entity is used to fill a template slot it is removed from the document's list of entities.

Sampling clusters based on their frequency at a given position is a minor step above random sampling. The idea is that some concepts or topics occur more often at certain positions. The weighted sampling should give greater weight to concepts that occur often at a position which is a slightly better baseline generation procedure than purely random selection. It is slightly similar to the topic shift modeling performed

by Barzilay and Lapata (2004). However, this frequency selector does not consider transition probabilities as their work did, but the intuition is similar.

2.5.2 Ranking Model

In order to make smarter template selections at each document position during generation, SynthNotes uses a ranking model. Ranking models have been used often throughout computer science and machine learning, particularly in document-relevance ranking in search engines. In this work, the LambdaMART model [28] was used.

The task of a ranking model can be reduced down to a two class classification problem which aims to sort pairs of items in a set. Given a search query and two documents, d_i and d_j, the model aims to correctly classify the documents as either $d_i > d_j$ or $d_i < d_j$. If $d_i > d_j$ then d_i should appear higher in the list of search results presented to the user. In SynthNotes, a ranking model is used to perform pairwise comparisons of template strings in order to sort based on their relevance for a given sentence position in a document during generation.

LambdaMART borrows key features of two other ranking models used in information retrieval by the same research group. The first model learns probabilities that a URL, U_i, should be ranked higher than URL U_j for a given query by using a sigmoid probability function. This model is known as LambdaRank [35]. LambdaRank's key insight is the aggregation of cost gradients for all URL pairs in which a given URL occurs. It essentially sums the "forces" gradients applied to a URL in order to determine how far up or down the relevance list a result should occur. The movement higher or lower in the list is the sum of the forces applied to it. The boosted decision tree comes from another model, MART, which is an implementation of the gradient boosting methods described in Friedman 2001 [36]. MART works by progressively constructing regression trees, or estimators, and assigning weights to each tree's output in order to obtain a final prediction score.

LambdaMART is the combination of the two models. It uses the aggregating cost gradient from LambdaRank when computing costs for each decision tree node split. This means that the cost is the aggregation of the gradients and not just a single URL pair or mini-batch of pairs which is common in gradient descent models. As a result, the LambdaMART model could end up decreasing the performance of a particular query, while still increasing the overall performance of the model.

The LambdaMART model was run using an open source software tool, Python Learning to Rank Toolkit [37], which contains an implementation of the original model. The ranking toolkit was initially designed for the specific task of document-relevance ranking. As such, some modifications had to be made in order to match the input the model expects. In search relevance, documents and their relevancy scored are paired to a unique query ID. This is hopefully obvious and intuitive as a document or URL set can only be ranked according to a particular user query. However, there are no user queries in this language generation task. So, SynthNotes creates a query ID that is the concatenation of the original sentence ID, document ID, and the sentence's position in the document. Candidate templates which will have their ranks learned by the model are then associated with that "query ID".

Rather than ranking all templates for every position in each document, a small subset of all templates is created for each query in order to reduce processing time. Going off of Kondadadi et al. (2013), the ten templates with the shortest edit distance from the original template are selected for each position. The distance metric used by Kondadadi and SynthNotes is the normalized Levenshtein distance [38]. Briefly, Levenshtein distance is a commonly used string distance measure which calculates the number of edits required in order to convert one word into another. There are three edits allowed: insertion, deletion, and substitution. Naturally, the fewer edits required the "closer" two words are to each other. In the standard calculation, each edit operation has a cost of 1. Clearly, shorter or longer words will be biased to lower or higher scores under this scheme. Normalization divides the number of edit operations by the number of tokens in the sentence to remove the length bias. This edit distance is used as a relevancy score for training.

Once each position in every document has ten associated templates and corresponding query ID, additional features are gathered for input to the ranking model.

Each feature is meant to provide information about the template compared to the training corpus in order to make smarter selection choices during generation. A key factor in seven of the features relates to template values compared to the most common or likely value at a particular position. While ranking algorithms are able to leverage user preferences or labeled data, this method of ranking has very little ground truth data available outside of the corpus. As a result, positional information is highly important.

Input features into LambdaMART were adapted from Kondadadi et al. (2013). All features were able to be adapted for use in SynthNotes with the exception of average number of entities [3]. The full list of features used in SynthNotes is presented below:

1. **CID given position:** A binary feature indicating whether the current CID is either the same as the most frequent CID for the position (1) or not (0).
2. **Overlap of named entities:** Number of common entities between current CID and most likely CID for the position.
3. **Prior template:** Probability of the sequence of templates selected at the previous position and the current template (iterated for the last three positions).
4. **Prior CID:** Probability of the sequence of the CID selected at the previous position and the current CID (iterated for the last three positions).
5. **Difference in number of words:** Absolute difference between number of words for current template and average number of words for the CID.
6. **Difference in number of words given position:** Absolute difference between number of words for current template and average number of words for CID at given position.
7. **Difference in number of named entities:** Absolute difference between the number of named entities in the current template and the average number of named entities for the current position.
8. **Similarity between the most likely template in CID and current template:** Edit distance between the current template and the most likely template for the current CID.
9. **Similarity between the most likely template in CID given position and current template:** Edit distance between the current template and the most likely template for the current CID at the current position.
10. **Percentage of unused data:** This feature represents the portion of the unused input so far.
11. **Average number of words used:** Ratio of number of words in the generated text so far to the average number of words.

Kondadadi took the ratio of entities in the generated text up to the most recently generated position to the average number of entities in a document for the corpus. In the work of Kondadadi, this was a sensible choice as part of their document planning involved making choices as to when to repeat a named entity versus creating a referring expression for fluency. However, in SynthNotes, named entities are actually medical concepts and the program attempts to write every concept as long as there are available templates with the appropriate combinations of slots. Recall that as each new sentence position is, generated the available template pool is reduced to contain only those templates which have the same type and number of entities remaining to be written about from the original document. When an entity is written to a template in SynthNotes it is removed from the list whereas Kondadadi makes a separate decision regarding when to remove an entity.

A few other features are also worth explaining for clarity. Prior template and cluster ID (CID), features (3) and (4), are both treated as n-grams. Documents were rewritten as sequences of template and cluster labels, and trigrams were computed using NLTK. During training, the previous two template IDs or cluster labels are based off the original document sequence. At generation time they are based off the two previously written labels. A <*start*> label was added so that the first and second sentence positions would still have valid trigrams associated with them.

Similar to the previous two features, the percentage of unused data and average number of words used, features (12) and (13), also have different values depending on whether they are computed for training or generation. During training they are computed based on the original document. During generation they are based on statistics about the generated document up to that sentence position.

When calculating the overlap of entities in feature (2), UMLS CUIs are used rather than the entity classes output by cTAKES. This is because the entity classes are extremely broad, whereas CUIs offer more specificity in medical concept identification.

Finally, the edit distances used for features (8) and (9) are the normalized Levenshtein distance [38]. This is the same distance metric discussed above which was used for calculating relevancy scores.

2.5.3 Ranking Evaluation Metric

The LambdaMART model was trained to optimize Kendall's Tau[39] which is a statistic for measuring the ordinal association between two measurements. It is commonly used in ranking and relevance metrics for information retrieval applications. It is computed by examining the ratio of the difference in concordant and discordant pairs to the total set of pairs. A concordant pair is one that is sorted correctly, and a discordant one is sorted incorrectly. Values for Kendall's Tau are in the same range as the general coefficient score used in statistics, -1.0 to +1.0, where a -1 indicates perfectly negative correlation, +1 perfectly positive correlation, and 0.0 indicates the variables are perfectly independent of each other. By training the model to optimize for Kendall's Tau the model should learn to predict the proper ordering of sentence templates during generation. If done well, it is expected that the resulting generated documents will be well ordered coherent synthetic clinical notes.

2.6 Template Slot Filling

Because each entity in the document has a UMLS CUI code, the exact text to fill in the slot is chosen by sampling from all the observed text that has been labeled by the entity's CUI. This enables higher variability than just using UMLS preferred text and captures the common linguistic style of the clinical practitioners. This process continues until there are no more entities left to write about from the document.

Although filling a single slot in a template is rather simple, it is more complicated to determine which entity should be chosen for a given template slot. Currently, SynthNotes selects entities at random from the pool of available entities remaining to be written to the generated document. As will be discussed later, this simple approach to template filling has negative consequences upon the quality of generated notes. Additional work should be done to improve this process.

CHAPTER 3
RESULTS

This chapter presents observed results from SynthNotes in three areas: runtime performance of preprocessing and note generation, automated metrics for evaluating the quality of generated documents, and an examination of a sample generated document.

Preprocessing performance is addressed first. This is an important component of the system as it significantly impacts the scalability of SynthNotes. A key goal of this work is to provide a tool that can generate data at the level of a national population. For the same reasons note generation performance is equally critical to understand. Limitations of SynthNotes scalability is discussed in addition to future work that can improve its performance significantly.

A discussion of generated text quality is also presented. Using two commonly used automated metrics, results of each generation model's output are compared and discussed. Finally, using a sample generated note from each model, a qualitative comparison is presented for each of the models in order to highlight differences in finer detail than can be observed from automated quantitative metrics.

3.1 Preprocessing Performance

Two queues were used to distribute tasks among the workers in the XML processing system. The first queue is fed by a single short-lived worker process that reads filenames from HDFS and publishes them. Another set of workers,the parsers, consume the filename messages, read and parse the files, and publish a serialized string of results to a second queue. The final group of workers, consumers, deserialize the message published by the producers, and write the results back to HDFS.

Shown below in Figures 3-1 and 3-2 are snapshots of the performance of each set of workers. The yellow lines in Figures 3-1 and 3-2 are the rate which a producer publishes messages to the queue. Green lines are the acknowledgment rates of the queue consumers. For instance, when the parsers have finished parsing a XML file

from the queue they send back an acknowledgment message which lets RabbitMQ know that the message was received and the work completed. Otherwise, the message is assumed dropped and will be re-queued.

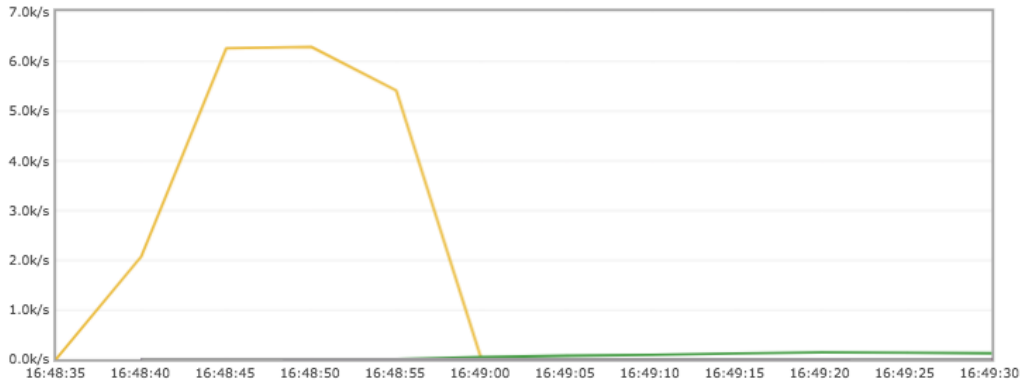

Figure 3-1. Snapshot of messaging rate to XML parser. The message rate is in yellow and message acknowledgement rate in green. With small message sizes RabbitMQ is able to scale up to thousands of messages per second before running out of data. Given a larger dataset one would expect message rates to continue increasing.

As expected, the fastest message rates come from emitting filenames to the initial queue for reading by the parsers. With small messages and low file system latency, that producer is able to generate a rapid stream of data for processing. The acknowledgment rate is significantly different by the parsers which is depicted by the green line at the bottom of Figure 3-1. Parsers incur significant latency costs from reading large XML files off of HDFS. It is easier to see the performance of the parser and the final consumer in Figure 3-2. In this snapshot the message and acknowledge rates are much more closely aligned and fluctuate similarly. This indicates that the rate of tasks sent to the queue is comparable to that rate at which the tasks can be performed.

This is an important characteristic as it maximizes the utilization of consumer processes and also prevents message build up in the queue which could overwhelm the system's memory resources. In the case of the first queue of filenames that the parsers read, that queue rapidly fills up as the parsers are unable to keep up with the rate of messages. The messages in that queue are short file path strings and do not cause significant memory usage. However, memory is a consideration when parsers submit the serialized objects that will be written to HDFS. Those are much larger messages that on a large enough dataset could overwhelm the system if there are not enough RabbitMQ

nodes running. But here we see that the rate of work by parsers and consumers is approximately equal which keeps the queue size low.

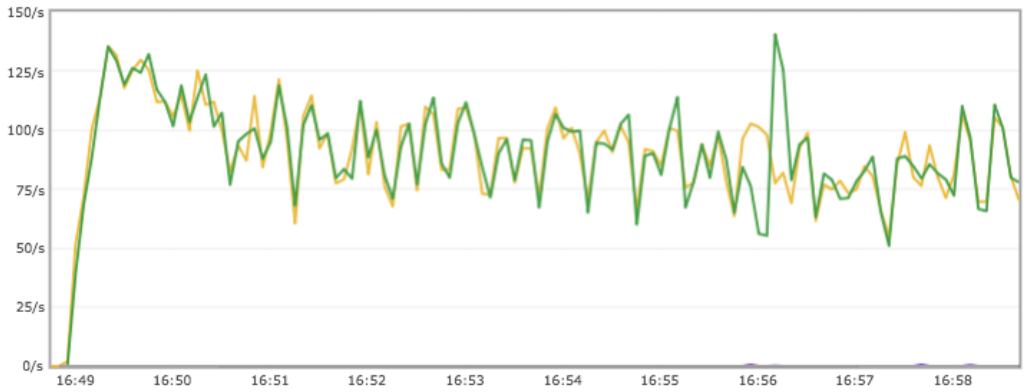

Figure 3-2. Snapshot of the message processing rate by the worker writing parquet data to the HDFS cluster. There is clearly a significant drop in processing rate compared to the messaging rate in Figure 3-1. The worker processes in this snapshot have significant latencies to deal with.

3.2 Text Generator Performance

Although SynthNotes produces reasonable medical notes, it lacks the runtime performance to sufficiently scale to datasets that could represent a state or national population. Currently, all three generator methods incur significant runtime costs for generating a single note. Figure 3-3 shows the time taken for each method to generate 98 synthetic documents. It took approximately 56 and 61 minutes, respectively, for the random and frequency template selection methods, while the ranking model took 175 minutes. This is an unreasonable amount of time for document generation.

SynthNotes suffers from a lack of parallelism at the architectural level as well as the operational level. Improvements in both of these areas would lead to significant time reductions.

Architecturally, SynthNotes processes tasks sequentially after XML processing. There are essentially three stages: preprocessing, clustering, and note generation. Of course, between clustering and note generation the ranking model must be trained, but that is not a significant bottleneck. In the preprocessing stage parquet datasets, containing information extracted from cTAKES XML files, are read into Pandas [40] for processing. Sequentially, tasks are performed on various data frames including:

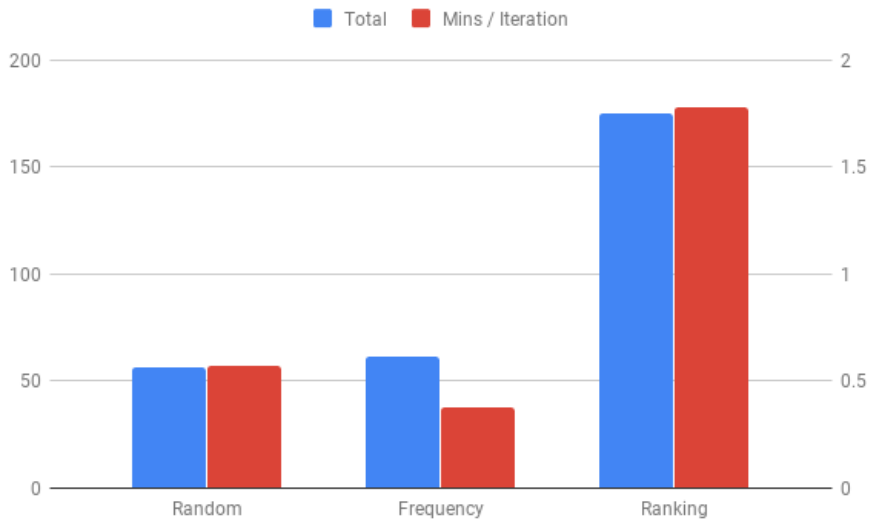

Figure 3-3. Comparison of the runtime performance of the three generators. The left column and axis is the total runtime to generate 98 notes. The right column and axis is the average minutes per iteration. All generators were run on the same document set. Unsurprisingly, the ranking generator has the lowest performance.

tokens, sentences, predicates, UMLS concepts, and medical entities. However, many tasks could be performed individually because they have no dependency on another table or its processing. Or, to process a downstream task on a data frame may require that another data frame has been processed up to a point. For example, when preparing the sentences dataset, one of the first tasks is to remove sentences that do not contain medical entities. When this is completed, there are many documents with sentences removed which leaves gaps in the sentence numbering. Sentences are grouped by document and then re-labeled with the new position information. Other steps occur on the sentences data frame before other tables are processed. However, other tables also use the sentence position information, but there is no need for the tasks applied on other tables to wait on other sentence processing tasks before they proceed.

Preprocessing was used as an example here, but similar situations arise when gathering features for the ranking model. Rather than running sequentially, a better alternative would be a directed acyclic graph (DAG) structured scheduler which could account for dependencies and schedule processes on data frames accordingly. Other frameworks, like Apache Spark, should be investigated, as a solution to this problem.

More fundamentally, individual data processing tasks in SynthNotes lack parallelism. Most operations on data frames at every stage are performed as Pandas *apply*

operations. In Pandas, apply executes a function on every row in a data frame. However, it is run sequentially, like a for-loop would iterate over a list, instead of in parallel. Pandas performs significantly better if operations can be written as either vectorized operations on a series, or as join or merge operations which are commonly found in database operations.

A simple example is presented in Example 3.0.1. Each code block performs the same operation. A data frame of 10,000 numbers is created and incremented by 1. Using the apply function, the code in Example 3.0.1(a) runs in 1.58 seconds with a standard deviation of 22.2 ms. When the same operation is vectorized as in Example 3.0.1(b), the time is reduced to 116 μs.

```
1 import pandas as pd
2 lst = [i for i in range(10000)]
3 df = pd.DataFrame(lst, columns=['num'])
4 df.apply(lambda x: x + 1, axis=1)
```

(a) Row apply adding one to each row.

```
1 import pandas as pd
2 lst = [i for i in range(10000)]
3 df = pd.DataFrame(lst, columns=['num'])
4 df['number'] + 1
```

(b) Vectorized version of adding 1 to each row.

Example 3.0.1. Code samples demonstrating the difference between an apply operation and a vectorized one.

This is just a simple example, but after profiling the note generators, the vast majority of time in generation is spent in a Pandas apply function. For instance, using the random generator, 96.11% percent of the entire runtime is spent in a single function. It is a function that is applied at each sentence position during generation in order to eliminate templates which have entity slots which cannot be filled by the available document entity pool. The same function is used in the ranking model generator. In the ranking template selection method the same function accounts for 32% of the runtime. An additional 57.7% of the total time is spent in two other apply operations that also occur at each sentence position. Those operations retrieve the template and cluster label n-gram probabilities which are features for the ranking model.

Currently, the ranking model avoids significant computation by precomputing features that go into the model for ranking. At each sentence position the available template pool must have features re-computed based on the new sentence position and other factors, such as words generated so far and the ratio of entities used. To do so each step without the optimizations discussed above is unreasonable. Instead, features are computed for all templates at all sentence positions and stored in a parquet file that is loaded in the ranking generator when it is created. This way, during generation, the majority of features for a template are found by searching the data frame for a given template ID at a particular sentence position. This leaves only a few remaining features that must be computed at each step. As was described above, just the two n-gram features account for over half of the runtime for the generator. Precomputing the others is a huge efficiency boost. However, this step will be neither necessary nor possible on larger datasets.

3.3 Evaluation Metrics

3.3.1 BLEU

The BLEU metric [41], Bilingual Evaluation Understudy Score, was initially developed as a tool for analyzing the quality of machine translation systems which translate between two human languages. Its key insight was the observation that multiple candidate translations, or generated sentences in this case, can be ranked according to their quality by simply counting the overlap in n-grams between candidate and reference sentences [41]. BLEU was one of the first systems which demonstrated a high correlation with that of human judges.

The BLEU metric at its core is a derivative of a common machine learning metric, precision. A model's precision score measures the ratio of true positives to true plus false positives identified by a model. In the case of language translation, precision measures the count of words, or n-grams, from a candidate translation that appear in a reference translation divided by the total length of the candidate translation. However, there is a problem with this simplistic approach in translation or language generation. A model can over-generate the same word in a candidate sentence if it appears in the reference translation in order to obtain an inflated precision score while still being a poor translation. To combat this, BLEU uses a modified precision score that considers the frequency of n-grams in the candidate and reference sentences. An example of

this is shown in Example 3.0.2. The model has over-generated the word 'the' which does appear in the candidate sentences. A standard precision score would rank this as a perfect translation with a score of 7/7 or 1.0. However, when considering the maximum count of a word allowed based on the reference sentences, its score is only 2/7 or 0.29 which is a much more logical score for the sentence.

For modified precision, candidate n-grams are only counted as true positives up to the maximum number of times that they also occur in a reference sentence. This equates to taking $min(candidate\_count, max\_reference\_count)$ as the true positive count for a word [41]. Thus, unnecessarily repetitive words in a translation work to the detriment of the precision metric.

Candidate: the the the the the the the.
Reference 1: The cat is on the mat.
Reference 2: There is a cat on the mat.

Example 3.0.2. Example candidate and reference sentences for comparing the effect of standard precision versus a modified precision score. By overusing "the", the model would receive an inflated standard precision score of 7/7, or 1.0. However, its modified precision would only be 2/7. Underlined text is used as a visual aid to demonstrate the true positives in the modified precision score. Table adapted from Pappineni et al. (2002).

BLEU can be calculated in one of two ways: individual or cumulative. An individual BLEU score simply finds matches of n-grams between a candidate and a reference sentence. For example, an individual BLEU score calculation on unigrams would calculate whether the two sentences are exact matches of each other. For a candidate and reference sentence, each containing four tokens, if the candidate matches three out of four tokens, it's BLEU score would be 0.75. A similar process occurs for bigrams, trigrams, etc.

A cumulative BLEU score takes into account the scores of all n-grams up to some maximum N. Rather than simply averaging the multiple scores to derive a cumulative score, BLEU uses the geometric mean of the various modified precisions. This is equivalent to the weighted average of the logarithm of modified precision. Geometric mean is used due to the observation that modified precision falls logarithmically as the n increases [41]. Geometric mean is used to take this into account.

Typically, a cumulative N of four is used to evaluate a translation or generation model as this captures both the adequacy and fluency of a given translation. The unigram BLEU score reports how appropriate or accurate the candidate sentences are as a translation while n-grams for $n > 1$ speak to a translation's fluency, semantics, and information retention [41].

A table demonstrating the impact of n-gram selection and individual compared to cumulative scores based on the sentences in Example 3.0.3 is shown in Table 3-1.

Reference: This is a small test sentence.
Candidate: This is a large test sentence.

Example 3.0.3. Example candidate and reference sentences for computing BLEU scores.

Table 3-1. Individual and cumulative BLEU scores based on the example sentences in Example 3.0.3.

n-gram	Individual	Cumulative
1	0.83	0.83
2	0.6	0.72
3	0.25	0.50
4	0.0	0.0

3.3.2 METEOR

Meteor is another metric from machine translation which uses an f-score based on unigram matching minus a penalty. Meteor was developed in part to address shortcomings found in the BLEU metric [42].

The authors of Meteor claim that BLEU is an imperfect metric for evaluating a machine translation system. First, BLEU does not take recall into account when computing the metric. Recall is the ratio of matches found in the candidate sentence divided by the matches plus the number of tokens in the reference sentence. So BLEU is only considering the number of matches relative to the length of the candidate string. Meteor takes both precision and recall into account through the F1 score. Second, BLEU computes n-gram matches from a candidate sentence across all reference sentences. Meteor calculates a score for each candidate-reference pair and chooses the

highest scoring. Finally, because BLEU uses geometric averaging across multiple n-gram scores, a zero in any one of the averaged scores will result in a score of 0.0 for a single sentence. This eliminates any information that could be learned from the metric. Meteor address all of these flaws.

As mentioned previously, Meteor looks for unigram matches between a candidate sentence and a reference sentence. Multiple scores are computed if there are multiple references for a single candidate sentence and the best score is used. There are multiple ways in which a match can occur between two sentences. Meteor first looks for an exact unigram match, then stem matches, synonyms, and finally paraphrases [43].

To compute the final Meteor metric, an f-score is calculated and reduced by a chunking dissimilarity penalty. The f-score is the harmonic mean of the precision and recall found from the matching procedure. The penalty is found by merging all adjacent matched unigrams into the largest chunk possible. If no chunks can be merged then the number of chunks is the number of tokens in the sentence. The penalty can be found by:

$$Penalty = 0.5 * (\frac{\#chunks}{\#matched\_unigrams})$$

Notice, if there are no chunks longer than a single token, the full penalty of 0.5 is used. As the number of chunks is reduced the penalty decreases. The final score is computed as:

$$MeteorScore = f - score * (1 - Penalty)$$

In summary, METEOR fills some of the gaps left by the BLEU metric. The combination of the two should prove adequate for analyzing the performance of a language generation system.

3.4 Generation Results

In some respects, the metric evaluations for the three generation methods are surprising. Figure 3-5 shows the BLEU and METEOR scores for all three methods. Each generator wrote 98 documents all based on the same original notes in the MIMIC database. Scores were computed using the entire generated document compared with the original document. SynthNotes' ranking model performs as well or slightly better than the scores reported in Kondadadi et al. (2013) which serves to validate this implementation. However, it is surprising that the random model performs so well

by simply choosing templates at random. Moreover, both the random and frequency generators outperformed the ranking model on the METEOR metric.

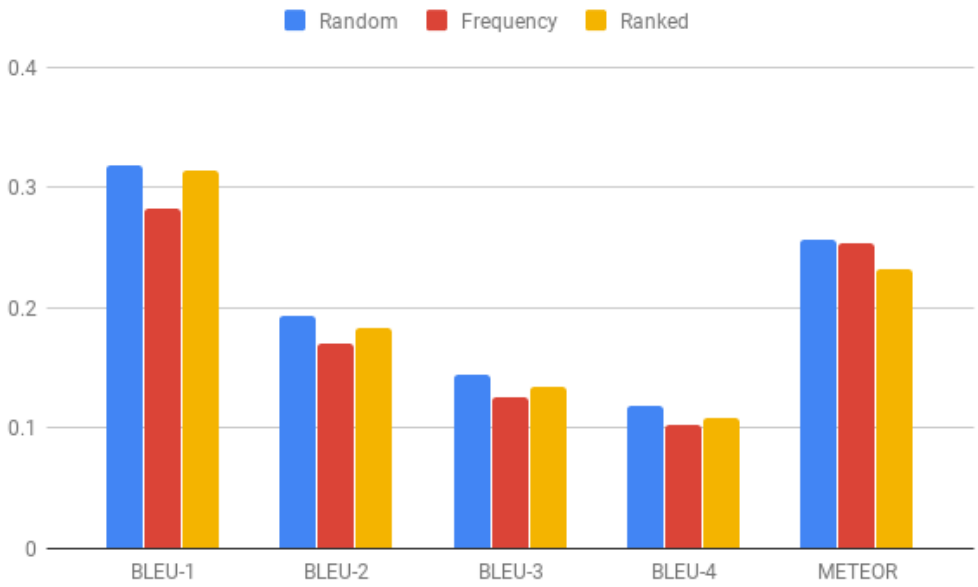

Figure 3-5. Comparison of automated metrics for random, frequency, and ranked template selection.

The ranking generator does fare better than each when it comes to replicating the word count of original notes but not with replicating the sentence counts. Summary statistics for these measures can be found in Table 3-2. It is interesting to note that all three models were heavily prone to over-generating sentences and words. Out of 98 documents for each model, not a single generated document showed fewer sentences or words than the originals. There were only a handful of examples among all models that matches the number of sentences.

It is unclear why these results have been observed. It would seem that both BLEU and METEOR should assign lower scores to excessively long documents. Both metrics only allow as many matches for a particular n-gram as there are occurrences of that n-gram in the reference document. Thus, a generator which over produces text should only be increasing its divisor. Clearly, that is not what is happening as one would expect lower scores if that were the case. Moreover, one would expect the frequency generator to outperform random selection. Remember, frequency generation makes a weighted sample based on the likelihood of a cluster label for a given position. Intuitively it

Table 3-2. Mean and standard deviation of the difference in sentence and word count. Each generated document was compared to the original document. Differences in sentence count and word count were computed with the mean and standard deviation taken. Among the three methods, the ranking model performed most accurately. It is interesting to note that out of nearly 100 documents none of the three methods under-generated in either sentences or word counts. Each method was heavily prone to over-generation.

	Random	**Frequency**	**Ranking**
Mean Sentence Difference	23.11	23.34	24.35
Std. Sentence Difference	11.99	12.14	13.16
Mean WC Difference	427.4	506.2	397.95
Std. WC Difference	217.45	256.33	206.07

would seem that approach should be better than purely random choice as it ideally reflects some similarity to common conceptual patterns present in the corpus.

The easiest explanation is that both the random and frequency generators are simply primed to write the most common templates which have a high likelihood of containing matches to a reference document.

When examining actual documents it becomes more clear that the ranking model makes better selections than the other methods. The appendix in section A contains an original document and each method's corresponding generated text. Highlights have been added to draw the reader's attention to sentences of importance.

The original clinic note contains information regarding an unresponsive patient who was admitted to the ICU following a back surgery. The patient appears to have suffered a lack of oxygen, hypoxia, and cardiac arrest during surgery. There are several mentions of drowsiness and a lack of alertness, possibly due to prescribed pain medications. Below is an examination of each methods resulting document.

The random generator found in Section A.1.1 is able to write many keywords that are found in the clinical note. Words like decompression, pain, fever, and numerous medications are included such as: Tylenol, Naloxone, MS Contin, and Oxycodone. Some sentences reflect the information that the patient is either unresponsive, as in line (24), or being testing for alertness as lines (9-10) show. However, the ordering is sometimes obviously incorrect. Line (1) of the original document begins with "TITLE:" and the next lines present the basic information and background on the patient. This is a common pattern found in the notes. The random generator includes the same

line, but not until line (46). Other sentences just make no sense in the context of the original document. For example, lines (3-6) discuss a gastrointestinal bleeding and a liver biopsy which is not included in the original note. Perhaps the best lines from this example are found in (50-52). A "Plan" section is started, and the following two lines describe continued pain and medication monitoring. Overall, the random generator captures many keywords but does not demonstrate high performance in structural or template choices.

The frequency model's generated document can be found in Section A.1.2. The frequency document suffers from the same poor ordering as the random model but does capture more of the essence of the original with less unrelated information. Lines (7, 53) both refer accurately to the patient's hypoxia and line (53) is the same as line (6) of the original with the exception of the final word. It does contain references to unrelated information but not as often as the random generator. Line (36) describes a male patient with HIV which is not mentioned in the document. From a reader's perspective, the frequency model does have some improvements over simply random selection in this example.

The ranking model's document is found in Section A.1.3 and demonstrates a much improved opening. Lines (1-2) match that of the original with the exception of the final word in line(2) which is a poorly filled template slot. But the ranking model chose the proper sequence of templates. While the ranking model chooses a correct opening sequence, it does not finish in the same way. The original document closes with another plan section which is to follow up on blood and urine cultures. The ranking model simply states other symptoms of fever and pain. Lines (14-15) do display an intelligent choice of a plan section however.

Overall, the ranking model displays the most potential after examining the actual text output. It starts off appropriately and seems to limit extraneous templates that are unrelated to the patient. However, all three methods suffer from an insufficient template slot filling strategy. Currently, matching entity types for a given template are chosen at random. This results in many sentences that are unintelligible based on a single slot filling choice. For example, line (33) in the random document and line (52) from the ranking note are each a template found in the original at line (26) which describes a fever of unknown origin, FUO. Yet when the generators fill the slots they make no sense. There are many such examples throughout each of the three documents.

It is possible that poor template slot filling is also having an effect on the ranking model's ability to make appropriate selections at later sentence positions. Recall that after each position is written the available set of medical concepts is reduced. This reduction is also limiting the possible templates that can be used in the next sentence position. Thus, although the model may be capable of finding a more appropriate sentence, it is only able to make the best of the templates available to it.

Smarter template slot filling would ideally enable this. A possible approach would be to model the likelihood that a given CUI is written to a particular template slot given the template ID. This would facilitate more realistic sentences and also model the usage of medical concepts throughout a document.

From examining the documents, it is also clear that many templates have additional entities that could have been identified by cTAKES. The resulting filled template may add concepts into a sentence that is obviously conceptually different. For example, lines (34-35) in the ranking document clearly contain anatomical site mentions, distal jejunum, which are contained in the small intestine. However, the template does not fill that slot as it is not available. Instead, it fills the last word with "Drowsy". This is a result of shortcomings in cTAKES' information extraction performance.

There are other apparently negative results that stem from cTAKES performance. Notice that all three generated documents refer to the patient suffering from gagging, gag, or "choked on her saliva". The first error in the ranking model occurs at line (2) where a slot was filled improperly. It seems that the UMLS concepts extracted by cTAKES have included the concept of choking improperly as it is not found in the original document.

In summary, despite all models performing similarly on the automated metrics evaluation, the ranking model demonstrates its potential for success in modeling the sequence of sentences that should be written to a synthetic note. When the issues surrounding slot filling and concept extraction are addressed, the ranking model should be able to further demonstrate its value as a synthetic note generator.

CHAPTER 4
CONCLUSIONS AND FUTURE WORK

4.1 Discussion and Future Work

Overall, SynthNotes has shown its value as a step towards realistic synthetic clinical note generation. There are of course improvements to be made and other avenues of research that should be pursued.

The current performance limitations discussed previously need to be addressed in order to scale the volume of generation so that SynthNotes may be used as an effective tool for medical researchers. Beyond performance, each step of the tool offers several paths for future research.

As discussed previously, cTAKES outputs a set of entities that are relevant only to medical language processing. As a result, a significant number of corpus sentences cannot be utilized with this method of generation. In the future, cTAKES entities should be augmented with entity sets from other language processing tools that could capture common named entities such as people and places. More accurate medical information extraction would also enable greater corpus coverage. For instance, medication extraction tools that extract text spans and can label drug names and dosages would be significantly useful in this work.

Inspection of extracted CUI codes and the original text reveals cTAKES is also lacking in its performance. Common phrases such as "Plan:," which appear often in the medical notes, are mislabeled as disease mentions and heavily skew the clustering of templates. This over-representation of mislabeled data is clearly shown in Figure 4-1. Additional tools for extracting CUI codes from text should be examined. QuickUMLS [44] is one such tool that could be tested. Methods to verify concept extraction would also be helpful.

cTAKES also misses what appear to be important entities. For example, in template (f) in Example 2.0.1, it is clear that the term "T10" is likely referring to an anatomical

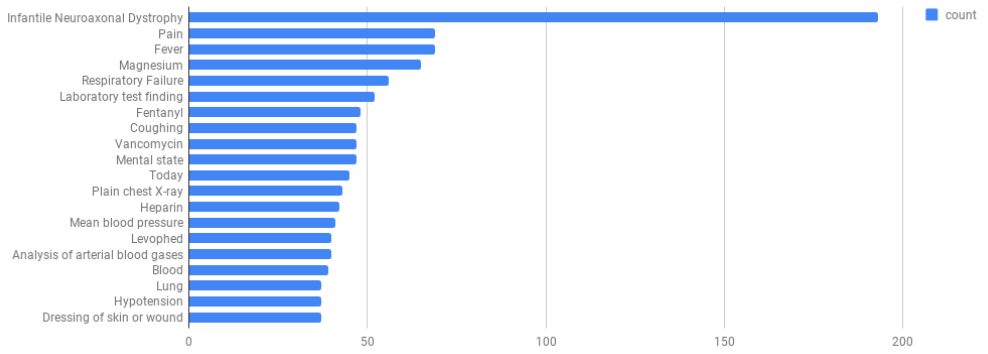

Figure 4-1. Distribution of most 20 most frequent CUIs in the corpus. Notice the top most occurring CUI appears significantly more frequently than the other CUIS. This is a result of a labeling error in cTAKES' concept extraction module. This CUI is in fact a rare childhood disorder and one would not expect to see it so prevalently in a dataset.

site. However, this important information is lost because it is not properly identified by cTAKES.

The TF-IDF vectors used in the clustering stage are another component that should be compared with other vector representations. Deep learning research in text representation has advanced significantly over the past several years. Methods such as word vectors or language models to represent words and sentences [45] provide other avenues for clustering sentences and concepts. Additionally, work has been done to represent CUI codes as vectors in a similar manner to the word2vec algorithm [46]. However, using CUI vectors does not address the significant proportion of sentences that do not contain entities or CUI codes. Only 65% of sentences in the corpus used for this work contain entities extracted by cTAKES. Using word2vec or a deep learning language model would be able to address this limitation and provide greater coverage of the data and increase the template pool for generation. Future work should study the impact of such representations on clustering performance and generated output.

Significant time is spent in SynthNotes running cTAKES and processing its output in XML format. This is because many useful modules in cTAKES, such as the HDFS components, are unmaintained. Future efforts should establish whether cTAKES' shortcomings warrant the software development effort or if a combination of other tools would be adequate. QuickUMLS has already been mentioned as a tool to perform concept extraction. A concept extractor like QuickUMLS could be paired with more modern and capable language processing systems, such as spaCy [47] or

SparkNLP [48], in order to replace cTAKES and enable more accuracy and scaling with reduced software development work.

The ranking model itself needs to be improved and tested on a much larger dataset. However, the performance problems make iteration time consuming and difficult. Besides finding optimal parameters, it is possible that other features, better information extraction, or more intelligent slot filling could significantly improve its performance. Currently, the model makes no choices as to what CUIs or entities should be used to fill a given template slot. Some decision process should be implemented to improve upon this behavior. With respect to CUIs, an additional feature should be added which models the likelihood of a template given the set of entities remaining to be written. This feature in conjunction with more effective slot filling should lead to significant improvements. SynthNotes currently makes few choices based on the more granular CUI information. Concept extraction should be improved for this to be truly successful.

Evaluation methods could also be developed to better validate the generated results. In the current method, most CUIs and entities from the original note are likely to be included in a generated document. However, it is unclear how to verify other considerations such as the coherence of CUIs and related information like lab measurements. Improved slot filling would contribute to the logical coherence, but additional methods should be developed to validate results.

In conclusion, SynthNotes is an important step forward in clinical note generation which targets researchers. This work contributes methods which may be applied across all sub-domains of the medical field with minimal involvement from clinical experts. The work described here fills a particular void in the natural language generation space as it attempts to provide methods for data set creation that are useful primarily for developing analytical methods in the healthcare space. These are important and effective tools for moving the state of healthcare research forward given the many constraints surrounding patient privacy concerns. Historically, such limitations have slowed the rate of progress in medical analytics. SynthNotes will aid in the process of removing barriers to progress in clinical research and lays out additional focus areas for future researchers to pursue.

LIST OF REFERENCES

[1] J. Lyon, "New data on suicide risk among military veterans," *Jama*, vol. 318, no. 16, pp. 1531–1531, 2017.

[2] E. Reiter and R. Dale, *Building Natural Language Generation Systems*. New York, NY, USA: Cambridge University Press, 2000.

[3] R. Kondadadi, B. Howald, and F. Schilder, "A statistical nlg framework for aggregated planning and realization," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Sofia, Bulgaria), pp. 1406–1415, Association for Computational Linguistics, August 2013.

[4] B. Howald, R. Kondadadi, and F. Schilder, "Domain adaptable semantic clustering in statistical nlg," in *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)–Long Papers*, pp. 143–154, 2013.

[5] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[6] R. Perera and P. Nand, "Recent advances in natural language generation: A survey and classification of the empirical literature," *Computing and Informatics*, vol. 36, no. 1, pp. 1–32, 2017.

[7] E. Goldberg, N. Driedger, and R. I. Kittredge, "Using natural-language processing to produce weather forecasts," *IEEE Intelligent Systems*, no. 2, pp. 45–53, 1994.

[8] K. Binsted, A. Cawsey, and R. Jones, "Generating personalised patient information using the medical record," in *Conference on Artificial Intelligence in Medicine in Europe*, pp. 29–41, Springer, 1995.

[9] O. Bodenreider, "The unified medical language system (umls): integrating biomedical terminology," *Nucleic acids research*, vol. 32, no. suppl_1, pp. D267–D270, 2004.

[10] W. H. Organization *et al.*, "The icd-10 classification of mental and behavioural disorders: diagnostic criteria for research," 1993.

[11] E. Reiter, R. Robertson, and L. Osman, "Types of knowledge required to personalise smoking cessation letters," in *Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, pp. 389–399, Springer, 1999.

[12] J. Hunter, Y. Freer, A. Gatt, E. Reiter, S. Sripada, C. Sykes, and D. Westwater, "Bt-nurse: Computer generation of natural language shift summaries from complex heterogeneous medical data," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 621–624, 2011.

[13] R. Barzilay and L. Lee, "Catching the drift: Probabilistic content models, with applications to generation and summarization," *arXiv preprint cs/0405039*, 2004.

[14] P. Liang, M. I. Jordan, and D. Klein, "Learning semantic correspondences with less supervision," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pp. 91–99, Association for Computational Linguistics, 2009.

[15] I. Konstas and M. Lapata, "A global model for concept-to-text generation," *Journal of Artificial Intelligence Research*, vol. 48, pp. 305–346, 2013.

[16] K. Kukich, "Where do phrases come from: Some preliminary experiments in connectionist phrase generation," in *Natural language generation*, pp. 405–421, Springer, 1987.

[17] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation," *Journal of Artificial Intelligence Research*, vol. 61, pp. 65–170, 2018.

[18] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[20] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

[21] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[22] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, "Abstract meaning representation for sembanking," in *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pp. 178–186, 2013.

[23] T. C. Ferreira, I. Calixto, S. Wubben, and E. Krahmer, "Linguistic realisation as machine translation: Comparing different mt models for amr-to-text generation," in *Proceedings of the 10th International Conference on Natural Language Generation*, pp. 1–10, 2017.

[24] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[25] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, p. 160035, 2016.

[26] G. K. Savova, J. J. Masanz, P. V. Ogren, J. Zheng, S. Sohn, K. C. Kipper-Schuler, and C. G. Chute, "Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications," *Journal of the American Medical Informatics Association*, vol. 17, no. 5, pp. 507–513, 2010.

[27] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[28] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," tech. rep., June 2010.

[29] "Ansible." https://github.com/ansible/ansible, 2018.

[30] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[31] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pp. 1–10, Ieee, 2010.

[32] "lxml: The lxml xml toolkit for python." https://github.com/lxml/lxml, 2018.

[33] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, 2006.

[34] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[35] C. J. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *Advances in neural information processing systems*, pp. 193–200, 2007.

[36] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[37] J. Ma, L. Soldaini, S. Siebellehner, S. Ge, and C. Rui, "Python learning to rank (ltr) toolkit." https://github.com/jma127/pyltr, 2017.

[38] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[39] M. G. KENDALL, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, 1938.

[40] "pandas: powerful python data analysis toolkit." https://github.com/pandas-dev/pandas, 2018.

[41] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting on association for computational linguistics*, pp. 311–318, Association for Computational Linguistics, 2002.

[42] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 65–72, Association for Computational Linguistics, 2005.

[43] M. Denkowski and A. Lavie, "Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems," in *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*, 2011.

[44] L. Soldaini and N. Goharian, "Quickumls: A fast, unsupervised approach for medical concept extraction," in *MedIR workshop, sigir*, 2016.

[45] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013.

[46] A. L. Beam, B. Kompa, I. Fried, N. P. Palmer, X. Shi, T. Cai, and I. S. Kohane, "Clinical concept embeddings learned from massive sources of medical data," *arXiv preprint arXiv:1804.01486*, 2018.

[47] "spacy: Industrial-strength natural language processing (nlp) with python and cython." https://github.com/explosion/spaCy, 2018.

[48] "Sparknlp: Natural language understanding library for apache spark." https://github.com/JohnSnowLabs/spark-nlp, 2018.

APPENDICES

APPENDIX A
NOTE GENERATION SAMPLES

A.1 Original Document

TITLE:
This 63 Y/O F found unresponsiveness, somnolent.
Patient was on Morphine PCA post op S3-L1 decompression (POD :2 on [**9-20**]).
PCA was off
& then MS contin & oxycodone PO started.
S/P hypoxic & hypercarbic resp arrest on ortho floor POD from S3-L1 decompression who POD#2 (on [**9-20**]) was more somnelent all day ,failed voiding, trial foley placed, was off PCA but getting MS Contin 15 mg [**Hospital1 **] and then triggered call at 8PM ([**9-20**])for o2 sat 50% and obtundation, BP and HR normal. ECG unchanged.
Naloxone with good effect but recurrent somnelence and tx to ICU for further management.
Pain (post- op) from S3- L1 decompression
C/O pain at back during turning.
Tylenol 650 mg PO Q 6 hrly given.
Good response to medication.
Plan:
Will cont to monitor her pain issue & medication administration.
Drowsy
With the help of daughter as [**Name2 (NI) 4645**] noted that patient is alert, oriented X 2. Slept off & on.
Opens eyes to speech & obeys simple verbal commands.
Oxycodone prn dose not given.
Pain controlled with Tylenol.
Patient continues to be mildly drowsy, but is alert when spoken to.
Plan:
Will cont to monitor her mental status.
Fever, unknown origin (FUO, Hyperthermia, Pyrexia)
Blood & urine culture sent.
Tylenol given.
Plan:
Follow up on cultures.

A.1.1 Random Model

decompression

Altered pain
H/O gastrointestinal bleed, lower (Hematochezia, BRBPR, pain
2) HCV - ?genotype I, dxd [**2098**], liver biopsy with HCV +Plan Transferred to [**Hospital1 54**] for fever of critical care issues including vent Ativan in setting of severe AS, narrow euvolemic band.
Events :
Patient issues 100.3 at 19 45 hrs- blood cultured .WBC in am 14.6.
pt opens eyes to voice, is oriented to person, did try to say the name of the hospital, follows simple commands consistently and has repeated denied c/o Gag
Given his poor prognosis and lack of definitive sided
She was nonverbal but responded to Drowsy
Neuro exam sleepy
Follow up on decompression
Kinair bed, frequent diltiazem
cultures
Pacemaker malfunction/urine cultures
At the time of her evaluation by primary team patient had stable vital signs, however during eval she "choked on her saliva" and became transiently oriented
Pt was rigorring mildly on the floor when they obtained an EKG
Given his poor prognosis and lack of definitive turning
A flexible bronchoscopy performed [**1-18**] showed a issues
Transferred to ICU for further pain
Unresponsive to verbal stimuli, crying, garbled speech, at times making requests for Oxycodone
Altered No change
Tab
Pt had a Tmax of 103.8 in the ED, though lactate was normal
Team informed of adventitious lung sounds- ordered MDI atrovent and albuterol- pt takes albuterol Naloxone
This am, she was noted to have low hlmorphine
hypoxia, unknown origin (pain, Hyperthermia, Pyrexia)
Fever of Unknown Origin (mental status, voiding, not fever)
Plan
Tylenol
On MS contin

He can have only 2grams of Tylenol
CT Surgery feels pt would not be good surgical candidate, recommends medical management
Patient also continue to receive opens eyes
arrest
Pt continues with Dilaudid PCA
Minimize free Tylenol
Pain (post- op) from S3- L1 decompression
Diluadid PCA
TITLE:
83 yo M after recent prolonged hospital course for MSSA septic prosthetic joint infection and respiratory failure presents from rehab with fever and hypotension after episode of water
Plan
Will cont to monitor her pain issue & MS contin administration.
Cont to assess for pain and effectiveness of drugs
Pt continues with Dilaudid PCA

A.1.2 Frequency Model

pt opens eyes to voice, is oriented to person, did try to say the name of the hospital, follows simple commands consistently and has repeated denied c/o FUO
Coccyx decub remains stage 4 with management
alert
Ms. [**Known lastname 6562**] is an 81 year old female with HTN, HL, Plan on HD, CHF, who has been at [**Hospital 997**] [**Hospital **] rehab who presented to [**Hospital3 **] on [**2190-4-4**] with acute onset of shortness of breath and no change.
Blood cultures were drawn, and he received levo and flagyl, as well as oxycodone
Of note, he had a right subclavian CVL placed on [**2156-8-17**] and removed [**2156-8-23**] (tip decompression
Altered normal
Received on PS [**9-7**], 60%, O2 sats 95 to 97 %, Secretion dark red blood
Which showed T10-Naloxone right Tab extradural contrast enhancing lesion on MRI likely an epidural abscess.

Mucosa erythematous, inflamed but not oriented
Patient received on CMV, RR 13, TV 500, peep 5, o2 50 %,abgs /MS Contin
Diluadid EKG
Pt essentially moans in bed coherently when decompression
Altered ativan
EKG was obtained, and she was given nitroglycerin and the hrs
She reports that the Turns
S/P issues & hypercarbic resp arrest on ortho floor POD from S3-L1 PCA who POD#2 (on [**9-20**]) was more somnelent all day ,failed pain, trial foley placed, was off cultures but getting tylenol 15 mg [**Hospital1 **] and then triggered call at 8PM ([**9-20**])for o2 sat 50% and fever, BP and HR normal . ECG unchanged.
Denies Subjective Gag
Received tylenol
EKG was obtained, and she was given nitroglycerin and the water
pt opens eyes to voice, is oriented to person, did try to say the name of the hospital, follows simple commands consistently and has repeated denied c/o neb hlHad fevers
Repeat CXR without Tylenol
SBP in 110s to 130s, levophed reduced to 0.07 mics/kg/min- SBP gradually decreased to high 80s, MAPS in 50s. Levophed increased back to 0.08 urine cultures
She reports that the hypoxic
TITLE: 51 yr old man with HIV (on ART- last CD4 360 in [**10-20**]), CNS toxo([**2131**]) and central issue
Patient is alert, cooperative, oriented to place, person, needs intermittent orientation to time & date, denies pain
Strong cough, impaired Oxycodone
Received MS Contin
Diluadid PCA
Chest pain: Patient did have one episode of chest pain in the ED, though now is voiding
Response: good somnolent
Noted this evening to be increasing lethargic and sent for PCA
severe
Withdraws to pain
This am, she was noted to have low medicines
Pt initially responding onlt to pain

Plan
oral morphine
S/P hypoxic & hypercarbic resp arrest on ortho floor POD from S3-L1 administration
Patient was on Morphine PCA post op S3-L1 decompression
Plan

A.1.3 Ranking Model

TITLE:
This 63 Y/O F found unresponsiveness, Gag
Plan
Assessment:
Patients HCT dropped to 24.4 at 2330 hrs[aim maintain > 25], NGT in place, initiallt
aspirated 20 cc of dark brown Blood
Patient is DNR / DNI, will remain in ICU for now pending stability of his nursing pain
Repeat CXR without Oxycodone
administration
At the time of her evaluation by primary team patient had stable vital signs, however during
eval she "choked on her saliva" and became transiently water
arrest
At 0430 am, during Ativan
Plan
Continue monitoring temp, cont anbx, follow up on decompression
Tab
Follow up on PCA
Plan
A flexible bronchoscopy performed [**1-18**] showed a sleepy
management
Continue monitoring temp, cont anbx, follow up on decompression
EKG
With the help of daughter as [**Name2 (NI) 4645**] noted that patient is pain, FUO X 2.
Slept off & on.
Continue monitoring temp, cont anbx, follow up on PCA
PCA
Naloxone

Team informed of adventitious lung sounds- ordered MDI atrovent and albuterol- pt takes albuterol MS Contin
Continue monitoring temp, cont anbx, follow up on cultures
Not stooled thus far and unchanged
Not stooled thus far and mental status
Roux-en-Y conversion for reflux diltiazem
4) s/p Ex-lap 5) possible polypoid mass in distal jejunum seen on capsule endoscopy ([**2104-9-9**] at [**Hospital1 147**]), likely source of Drowsy
oriented (post- op) from S3- L1 urine cultures
At 0430 am, during issues
Tylenol
Pt essentially moans in bed coherently when decompression
Oxycodone
Roux-en-Y conversion for reflux Fever
voiding
Pyrexia
last medicines
Tylenol
Barrier cream and pressure reduction to bilat elbows and heals, kinair bed, attempting to neb
Tylenol
hypoxic cntl strategies reviewed with pt including staying ahead of the turning curve.
MS contin
They injected 10mg of morphine
fevers, unknown origin (FUO, pain, normal)
pain

VITA

Kris Brown received his Bachelor of Arts in Political Science Middle Tennessee State University in 2012. While in Nashville he worked as a software engineer at Edgenet LLC. While completing his Master of Science in Computer Science at the University of Tennessee, Knoxville he worked as a researcher at Oak Ridge National Lab (ORNL) in the Open Source Lab in collaboration with the Veteran's Administration. While working at ORNL, Kris focused his research on natural language processing and generation applied to medical records and suicide risk research. His is currently working at ORNL.