Personalized Knowledge Base Construction via Natural Language Instructions

Nghia T. Le

CMU-CS-20-123
AUGUST 2020

Computer Science Department
School of Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Matt Gormley (Advisor)
Tom Mitchell

Submitted to Carnegie Mellon University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Keywords: Personalized Knowledge Base, Learning from Instructions, Information Extraction, Transformer Encoder-Decoder

Abstract

We consider the problem of constructing personalized symbolic knowledge base (KB) through natural language instructions. This problem presents several challenges, including (1) integrating symbolic knowledge from the evolving KB with user utterances to produce the appropriate KB modification commands, and (2) handling open domain utterances that may, e.g., introduce new entities at test time. We design alternative neural network encoder-decoder models that combine the unstructured context from the utterance with the structured context from the KB. Empirical results and analysis show that our models are able to construct the knowledge bases from user utterances with high accuracy. We also contribute an evaluation dataset, and perform detailed analysis that reveals interesting properties when applying neural models on this task.

Contents

Chapter 1

Introduction

Users of current personal devices are able to interact with AI assistants (e.g., Siri, Alexa) in a limited capacity, often by querying pre-programmed commands (e.g. *"What is the weather like today?"*). These devices, however, do not allow users to teach their assistants personalized concepts. If AI assistants could be taught such knowledge, it would open the possibility of automated assistants that can cater to each user's unique needs.

While there is a growing research interest in learning from instructions [12, 13], there has been little effort in constructing personalized knowledge bases (KB) from user instructions. Personalized KBs would allow users to inspect their taught knowledge via visualization, as well as assist in downstream tasks such question answering, email classification, or commonsense reasoning [1]. At the same time, current automatic KB construction efforts mainly focus on extracting information from large text corpora like the Web [2, 5].

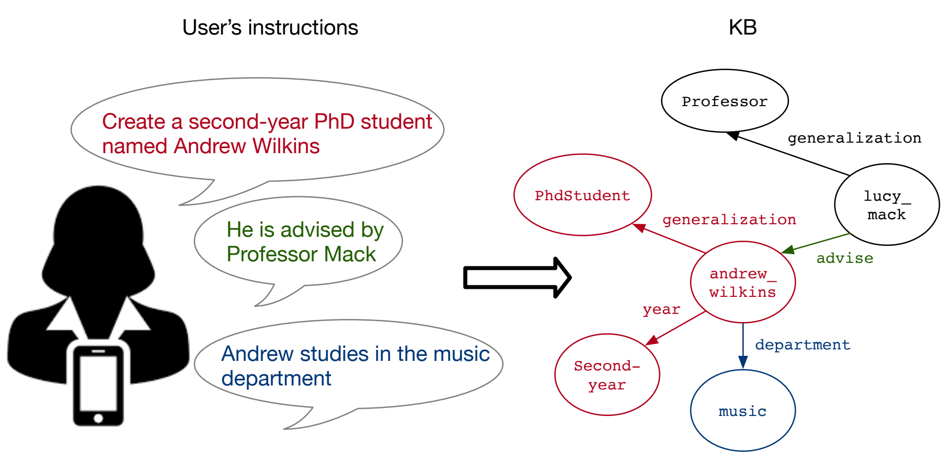


Figure 1.1: User's personal KB (represented as a Knowledge Graph) is constructed from user's instructions

This work lies between learning from instructions and automatic KB construction: we aim to construct a personalized KB from natural language instructions (Figure 1.1). We hypothesize that on-the-fly KBs constructed through interactions with the user will provide an additional source of structured context to aid in processing the unstructured natural language utterances from the user. This problem combines a unique set of challenges, including extracting relevant entities and relations from the user instructions, linking them to the symbolic KB, and executing the information to evolve the KB.

We propose several neural models that can effectively learn to map user instructions to a set of commands executable on the KB, therefore continuously populating the KB with personalized information as the user interacts with the system. Evaluations on a synthetic dataset of natural language instructions show that our models can construct the personalized KB with high accuracy, while our analysis reveals interesting properties when building neural models for this task.

In summary, our contributions include:

- Introducing the task of user-instructed KB construction, with a supporting dataset
- A neural system incorporating the symbolic, evolving KB that performs effectively on the proposed task and dataset
- Analysis that shows strengths, weaknesses, and interesting properties of the proposed task and system

Chapter 2

User-Instructed Knowledge Bases Construction

2.1 Task Definition

In the user-instructed KB construction setting, the system is given a sequence of natural language utterances and an initial personal KB from the user. The goal is to extract relevant information from these utterances in order to construct and modify the user's personal KB. For example, from the utterance *"Create a second-year PhD Student named Andrew Wilkins"*, we create a new entity called `andrew_wilkins`, set its category to `PhdStudent`, and set its year to `second_year` (Figure 1.1). Formally, given a sequence of T utterances $u_1, u_2, ..., u_T$ and an initial KB_0, we want to learn a function f that sequentially constructs KB_i from KB_{i-1} and utterance u_i:

$$\text{KB}_i = f(\text{KB}_{i-1}, u_i) \quad \forall 1 \leq i \leq T \tag{2.1}$$

KB Construction as Sequence Generation Instead of learning f (eq. 2.1) directly, we adopt the sequence generation approach for information extraction [3] by mapping utterance u_i to an executable target sequence called *command*, denoted c_i, via function f_{generate}. This command is then executed on KB_{i-1} using an execution engine f_{execute} to produce KB_i:

$$\begin{aligned} \text{KB}_i &= f(\text{KB}_{i-1}, u_i) \\ &= f_{\text{execute}}(\text{KB}_{i-1}, c_i) \\ &= f_{\text{execute}}(\text{KB}_{i-1}, f_{\text{generate}}(u_i)) \end{aligned}$$

The design of f_{execute} is tightly coupled with the format of c_i and the KB (e.g. if the KB is a SQL database and c_i is a SQL query, then f_{execute} will be a SQL executor). In this work, we treat the KB as a Knowledge Graph (KG), where each node in the graph represents an entity in the KB and edges between nodes represent their relations (Section 2.2). We also design a custom command language (Section 2.3) and execution engine f_{execute} (Section 2.4). The problem can then be reduced to learning the mapping f_{generate} from utterance u_i to command c_i in a supervised manner.

Treating this problem as a sequence-generation problem allows us (1) to experiment with rich encoder-decoder models from NLP literature and (2) the flexibility in designing the target language. While the target language in this work is designed for natural language instructions, we can extend the language to accommodate more complex natural language utterances, such as rules (*"PhD students are required to submit a thesis"*) and questions (*"Who are the PhD students?"*)

2.2 Directed Knowledge Graph

We represent our Knowledge Base as a Knowledge Graph (KG): each node in the graph represents an entity in the KB. Edges between nodes represent their relations. The node types correspond to pre-defined entity categories (e.g. `PhdStudent`, `Course`, etc). These entity categories are also themselves entities in our KB, and thus are also nodes in the graph. Representing categories as entities permits categories to participate in relations just like other entities do, including links to the utterances that mention them.

2.3 Target Command

Under the Knowledge Graph representation, knowledge fragments can be represented as triples (h, r, t), where h, r, t denote head entity, relation, and tail entity. To extract this information from the utterance u,[1] each entity in the triple must be matched with a *mention* from the utterance, and we need to know what *action* to take with this information. All of this information (mentions, entities, relations, and actions) must be encoded in c.

Notations	Example(s)
Utterance u	*Andrew studies in the music department*
Mention set M_u	{*Andrew*, *music*}
Entity set E_u	{`andrew_wilkins`, `new`}
Relations set R_u	{`department`}
Actions set A_u	{`set`}
Command c	*Andrew*:`andrew_wilkins` `department:set` `music:new`

Table 2.1: An example of an utterance u and corresponding command c, with the components that make up c

Mentions Mention set M_u contains noun phrases in the utterance that refer to entities in the KB. This mention set can consist of all possible spans in u or come from an oracle [14]. We only experiment with the mentions provided *a priori*, leaving the investigation of the all possible spans setting for future work.

[1] To simplify the notation, we omit the subscript i (e.g. u_i, c_i become u, c, respectively)

Entities Entity set E_u comprises all entities mentioned in utterance u. In particular, each $m \in M_u$ should have a corresponding $e \in E_u$. We also add special entity `new` if m refers to an entity not yet present in the KB, which can then be created during execution. This allows our model to link the mention m against an open vocabulary of entities during test time.

Relations and Actions Relation set R_u consists of pre-defined relations between the entities in u. Each relation $r \in R_u$ needs an action $a \in A_u$ to specify how to execute the extracted knowledge. a can take the following values:

- `add`: adding an edge
- `remove`: removing an edge
- `set`: replacing an old edge if one exists, otherwise adding an edge

We construct c by linearizing the components of E_u, M_u, R_u, A_u in roughly (h, r, t) order. Specifically, each token in c is either a mention-entity pair m:e or a relation-action pair r:a, separated by whitespace. The first token of c is the head mention-entity, and we constrain the utterance to have only one head. [2] Subsequent tokens of c are (r, t) pairs: a relation-action token r:a followed by its corresponding tail mention-entity token m:e. In other words, c is the linearization of triples $(m_h{:}e_h, r_1{:}a_1, m_1{:}e_1), \ldots, (m_h{:}e_h, r_p{:}a_p, m_p{:}e_p)$:

$$c = m_h{:}e_h \;\; r_1{:}a_1 \;\; m_1{:}e_1 \;\; \ldots \;\; r_p{:}a_p \;\; m_p{:}e_p$$

Table 2.1 shows a full example of c and its components. A limitation of this formulation for command c is that it requires explicit mentions to be linked to an entity, thus prohibiting us from detecting implicit entities not mentioned in the utterance.

Our formulation of command c requires explicit mentions to appear in the utterance, with the exception of commonly used Boolean entities `True` and `False`. We link these Boolean entities to special the mention `<BOOL>`.

2.4 Execution Engine

After generating the command c_i, we execute c_i on the KB using the f_{execute} algorithm 1

2.5 TextWorldsKB Dataset

2.5.1 Overall

We evaluate our approach using TextWorldsKB, a synthetic dataset generated ourselves using the TextWorlds framework [13]. This framework is designed to serve as the experiment testbed for the task of Question-Answering over user-instructed knowledge. We chose TextWorlds because of the available user-simulated settings, where each utterance in the dataset can introduce new knowledge or update existing knowledge, simulating a user's world. The flexibility of the

[2] We can relax the "one head mention-entity" constraint for more complex utterances by inserting a head token right before each (r, t) pair

Algorithm 1 KG construction algorithm f_{execute}

Input: initial KG_{i-1}, target command $c_i = m_h{:}e_h \;\; r_1{:}a_1 \;\; m_1{:}e_1 \;\; \ldots \;\; r_p{:}a_p \;\; m_p{:}e_p$

KG_i = KG_{i-1} m_h, e_h = first element of c_i $\mathcal{T}$ = pairs of $(r_j{:}a_j, m_j{:}e_j)$ from c_i **if** $e_h = \mathtt{new}$ **then**
    create head entity h with name `lowercase(`m_h`)` add h to KG_i
else
    retrieve head entity $h = e_h$ from KB_i
end
for $(r_j{:}a_j, m_j{:}e_j) \in \mathcal{T}$ **do**
    if $e_i = \mathtt{new}$ **then**
        create tail entity t with name `lowercase(`m_j`)`
    else
        retrieve head entity $t = e_j$ from KG_i
    end
    if a_j = `add` **then**
        Create an edge r_j connecting h and t in KG_i
    else if a_j = `remove` **then**
        Remove an edge r_j connecting h and t in KG_i
    else if a_j = `set` **then**
        Remove an edge r_j from t in KG_i Create an edge r_j connecting h and t in KG_i
end

framework also allows us to easily generate supporting KBs that change with each new utterance, as well as label the utterances with target commands. We generated four out of five *worlds* (i.e. domains),[3] where each world has a different set of entity categories and relations. Within a world are *stories*, with each story containing different entities but sharing the same relations. TextWorldsKB also has a natural flow of instructions, incorporating linguistic phenomena such as coreference (e.g. *"the meeting with Lucio to grade papers"*, *"the homework about linear algebra"*- Figure 2.1). We show the overall statistics in Table 2.2 and detailed statistic according to each world in Table 2.3.

Statistics	Single	Multiple
# of utterances	10000	12000
# stories	1	120
Avg. utterance length	7.1	7.3
Avg. mentions per story	19716	207.1
Avg. coreferences per story	3325	39.6
Avg. entities per story	2137	37.4
# relations	48	48
# entity category	57	57
# token size	1716	1787

Table 2.2: Overall dataset statistics, for single story and multiple stories

[3] Shopping world is not available on the TextWorlds framework

Statistics	ACADEMIC	MEETING	HOMEWORK	SOFTWARE	Total
# of utterances	3000	3000	3000	1000	10000
Avg. utterance length	6.5	6.3	8.1	8.5	7.1
# of mentions	5615	5578	6276	2247	19716
Avg. mention per utterance	1.9	1.86	2.1	2.2	2.0
# of coreferences	960	1026	821	518	3325
# of entities	520	545	854	218	2137
# of relation	25	10	12	12	48
# of entity category	9	7	9	8	57
Tokens vocabulary size	674	665	584	292	1716
Statistics	ACADEMIC	MEETING	HOMEWORK	SOFTWARE	Total
# of utterances	3000	3000	3000	3000	12000
Avg. utterance length	6.6	6.2	7.8	8.9	7.3
# of stories	30	30	30	30	120
Avg. # utterance per story	100	100	100	100	100
# of mentions	5848	5625	6255	7125	24853
Avg. mention per story	195.0	187.5	208.5	237.5	207.1
# of coreferences	971	992	872	1590	4425
Avg. coreference per story	32.4	33.1	29.1	53.0	36.9
# of entities	1047	1027	1230	1179	4483
Avg. # entity per story	35.0	34.2	41.0	39.3	37.4
# of relation	25	10	12	12	48
# of entity category	9	7	9	8	57
Tokens vocabulary size	903	731	856	373	1787

Table 2.3: Full TextWorldsKB dataset statistics

2.5.2 Single story vs. multiple stories

Each world in TextWorldsKB contains two types of datasets: *single story* and *multiple stories*. A single story dataset contains one story describing the perspective of a single user. This corresponds to the setting where the system only learns from a single user, where every entity in a single story dataset is unique. A multiple stories dataset contains multiple stories describing the perspectives from different users, corresponding to the setting where the system learns from multiple users. In multiple stories dataset, we can have cases where an entity refers to a category in one story, and another entity with the exact same lexical form refers to another category in another story (Figure 2.1). For each of these dataset types, we report the results and analysis separately.

Single Story

Create a second-year PhD student named **Andrew Wilkins**
He is advised by Professor Mack
Andrew studies in the music department
Andrew is now a third-year PhD student
U302 is TA-ed by **Andrew**
…

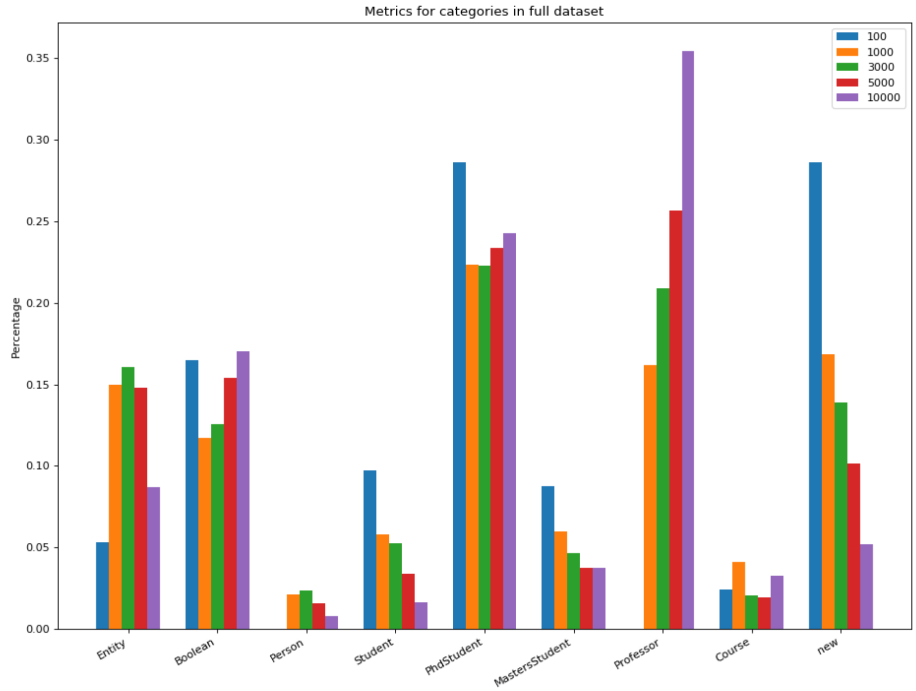
Multiple Stories

Story 1
Andrew Wilkins is a professor
He works in the computer science department
Professor Wilkins currently has funding
….
Story 2
Create a course in computer science called G900
Andrew Wilkins is a Masters student
That course is TA-ed by **Andrew**
…

Figure 2.1: Examples from the single story dataset (left) vs. the multiple stories dataset (right). In multiple stories, we can see that Andrew Wilkins is a professor in story 1, but a master student in story 2. These are two different Andrew Wilkins described by two different users, corresponding to two different stories. In single story, there is only one unique Andrew Wilkins, who is a PhD student.

2.5.3 Dataset Analysis

Since our dataset is synthetic and sequential in nature, it is not necessary the case that the more data the better the performance. In particular, since there are a finite amount of entities that can be created, subsequent utterances generated after we max out all the entities will be skewed towards certain relations. This is particularly true for the single story datasets: We can see from the Figure 2.2 that in dataset size with 10000 statements, the data is skewed towards relations `committee` and `sabbatical`. In this project, we want to evaluate the system on as many relations and entities as possible. Thus, we choose the dataset size with the most uniform distribution across the relations and entities. Note that this effect only appears in single story dataset. For multiple stories, since the number of utterances at each story is significantly smaller (100), and each story is independent, we don't have the issue of dataset skewing towards certain relations/entities (Figure 2.3). Thus, we chose the dataset size that is comparable to single story datasets (30 stories, 100 statements/stories, 3000 statements in total for each world)



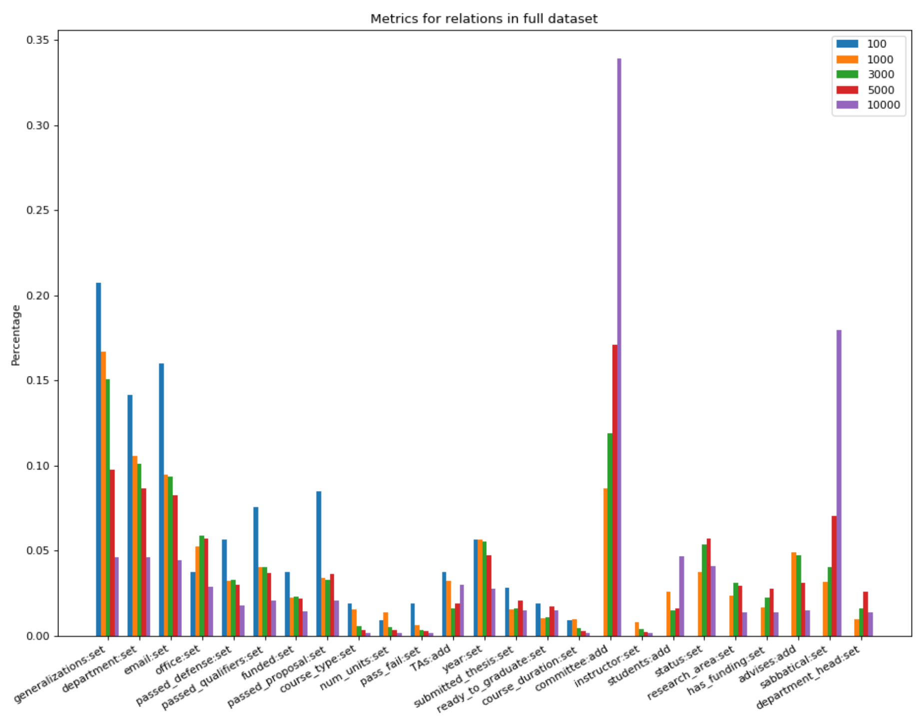
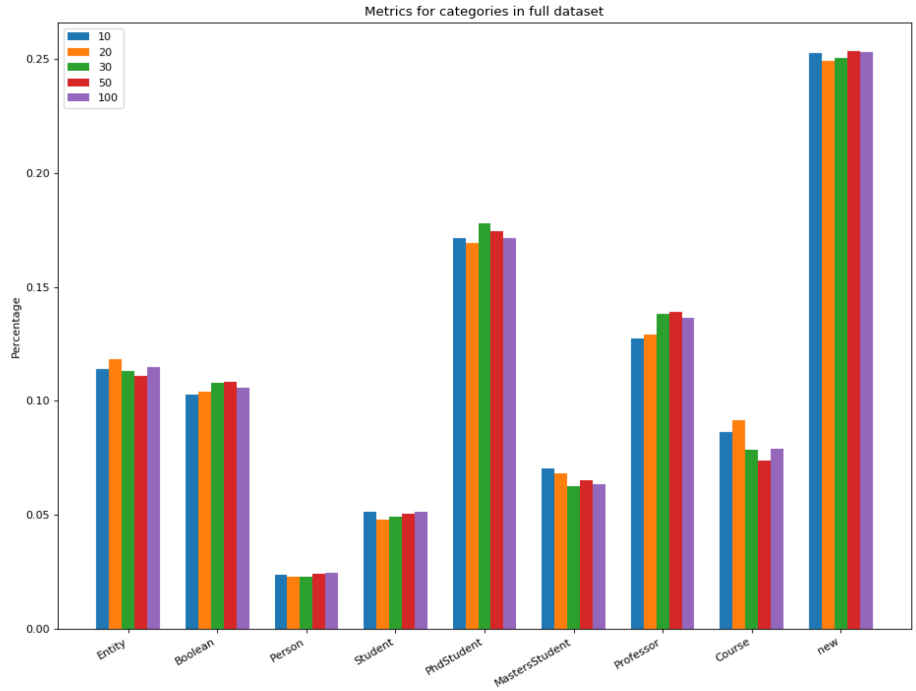


Figure 2.2: Distribution of categories (top) and relations (bottom) for Department world, single-story dataset with 100, 1000, 3000, 5000, 10000 statements



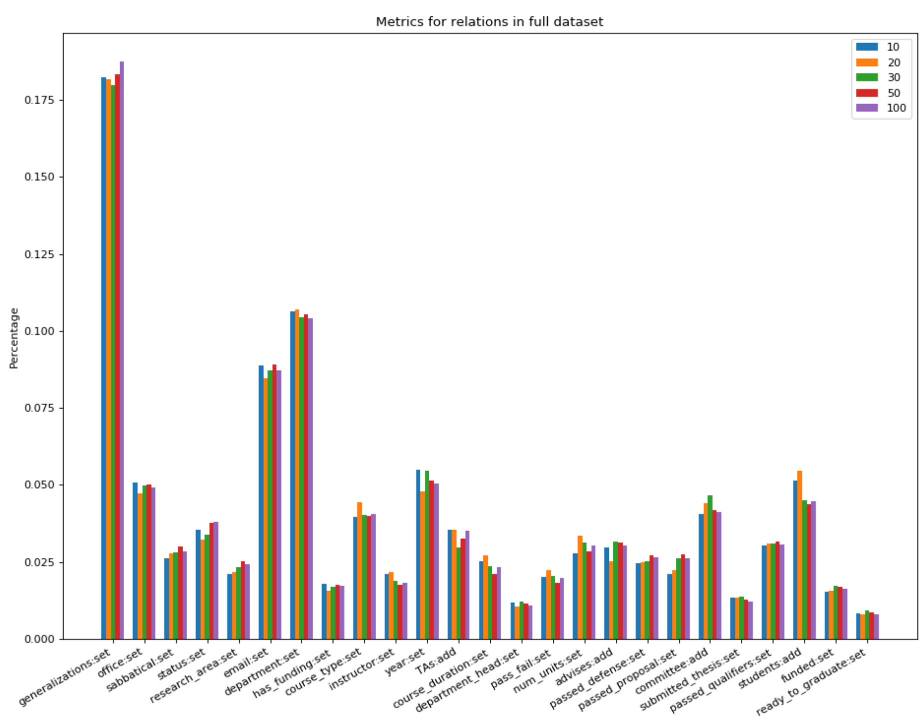


Figure 2.3: Distribution of categories (top) and relations (bottom) for Department world, multiple-stories dataset with 1000, 3000, 5000, 10000 statements

Chapter 3

Dynamic KG Transformer Networks

We learn the mapping from utterance u to command c with neural encoder-decoder models in a supervised manner.[1] Furthermore, having access to the KG motivates us to combine the structured information from the KG with the unstructured context from the utterances to produce the target command. Using the common Transformer encoder-decoder [26] as our base model (Section 3.1), we augment with (1) a mention embedding for each mention m (Section 3.2), (2) an entity embedding for each entity e that encodes both the structured context from the KB and unstructured context from the mention history (Section 3.3), and (3) an entity linker that learns to link m to e (Section 3.5).

3.1 Base Transformer Architecture

Our base architecture is the commonly used Transformer encoder-encoder model. Since our model architecture is almost identical to the original work, we omit the details and refer the reader to [26]. We refer to this architecture as BaseTransformer. While powerful, this architecture has a key shortcoming when applied to our task: it cannot split the mention from the entity in the mention-entity pair m:e, as it treats m:e as a single token in the output vocabulary. This prohibits the model from linking entities created during test time. We solve this by decoupling the mention-entity pair during the decoding process, allowing the model to learn how to link mentions to the correct entities (Section 3.4).

Input Representations We represent the input utterance u as the input vector $\mathbf{u}$. We pass $\mathbf{u}$ onto a pretrained embedding layer, which consists of context-independent GloVe embeddings [16] and Elmo embeddings [17]. We concatenate the these two pretrained token embeddings and pass the concatenated vector into the Transformer encoder, forming the contextualized encoding $\bar{\mathbf{u}}$ (Figure 3.1).

[1] Similar to section 2.3, we omit the subscript i denoting the order of utterance for most of this section, except in section 3.3 where we need information from previous utterance u_{i-1} and KG_{i-1} to construct entity embedding.

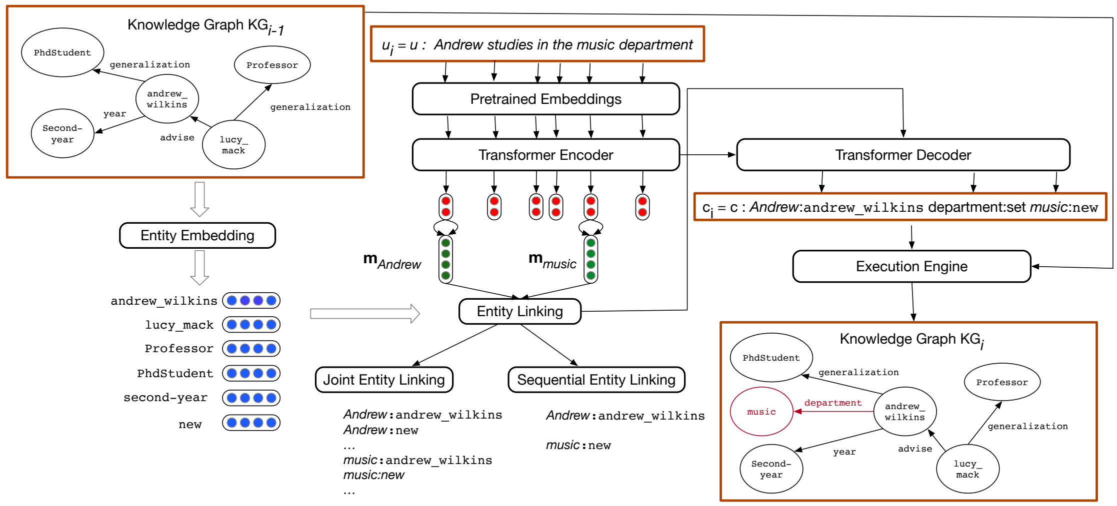


Figure 3.1: Overview of our approach. Given knowledge graph KG$_{i-1}$ and utterance u_i = *"Andrew studies in the music department"*, we first produce entity embeddings for each entity in KG$_{i-1}$ and mention embeddings $\mathbf{m}_{Andrew}$, $\mathbf{m}_{music}$. The entity linker (with two different mechanisms, joint and sequential, corresponding to two different models) then learns to linnk the mentions with the appropriate entities {andrew_wilkins, new}. These linking information is passed to the entity-aware decoder to help generate the target command c. The execution engine then combines c with KG$_{i-1}$ to produce KG$_i$

3.2 Mention Embedding

Given the contextual encoding $\bar{\mathbf{u}}_\mathbf{i}$, for each mention $m \in M_u$, we produce a mention embedding $\mathbf{m}$ concatenating the embeddings of the first and last words in the mention [10]. We then feed the concatenated vector into a standard feed forward network. Assuming the span of mention $m = [u_q, ..., u_r]$, we have

$$\mathbf{m} = \text{FFNN}_m([\bar{\mathbf{u}}_q; \bar{\mathbf{u}}_r)]$$

3.3 Entity Embedding

Inspired by [7], we leverage the KG constructed from the previous utterances to help with parsing the current utterance. For each entity node e in the KG, we compute the *mention history vector* $\mathbf{v_h}(e)$ representing the unstructured context of e, and the *graph embedding* $\mathbf{v_g}(e)$ that captures the structured context of e.

Mention history The mention history vector $\mathbf{v_h}(e)$ comprises the aggregated mentions of e up to, but not including, the current utterance u_i. [2] Concretely, suppose we are processing utterance

[2]Since we have yet to know if e is linked to mentions in u_i

u_i. Let $E_{u_{i-1}}$ be the extant entities [3] linked at utterance u_{i-1}. If $e \in E_{u_{i-1}}$, then we know that e is linked to a mention m_e (and its embedding, $\mathbf{m_e}$) in mention set $M_{u_{i-1}}$ of utterance u_{i-1}. We then incorporate this $\mathbf{m_e}$ into $\mathbf{v_h}(e)$, otherwise we inherit $\mathbf{v_h}(e)$:

$$\mathbf{v_h}(e) = \lambda \mathbf{v_h}(e) + (1 - \lambda)\mathbf{m_e}$$

where

$$\lambda = \begin{cases} \sigma(\mathbf{W_h} \cdot [\mathbf{v_h}(e); \mathbf{m_e}]) & \text{if } e \in E_{u_{i-1}} \\ 1 & \text{otherwise} \end{cases}$$

This formulation is similar to [7]. Our work differs in that (1) our KBs evolve as the interactions progress, whereas their KBs do not, and (2) we learn to link entities, and then integrate the mention embedding *after* decoding (since we only know the linking results after decoding), while they integrate their mention embedding after heuristically linking entities *before* the decoding process

Graph Embedding To learn the structured context of entity e from the KG, we utilize a Relational Graph Neural Network [21] to produce the graph embedding $\mathbf{v_g}(e)$. Specifically, for a graph network with L layers, we compute the output representation $\mathbf{h}_e^{(l+1)}$ at the l^{th} layer $(0 \leq l \leq L)$ for entity e as

$$\mathbf{h}_e^{(l+1)} = (\sum_{r \in \mathcal{R}} \sum_{e' \in \mathcal{N}_r(e)} \mathbf{W_r}^{(l)} \mathbf{h}_{e'}^{(l)})$$

where $\mathcal{R}$ is the relation set, and $\mathcal{N}_r(e)$ is the neighbors of e that are connected by relation r.

The initial input to the graph network is the mention history vector $\mathbf{h}_e^0 = \mathbf{v_h}(e)$, and the output of the graph network is graph embedding $\mathbf{v_g}(e) = \mathbf{h}_e^L$. This graph network allows information to propagate between nodes via message-passing.

3.4 Entity-aware Decoder

We decouple the mention-entity pair during the decoding process in order to learn how to link the mentions to the correct entities, allowing the model to link entities that are created at test time and thus access to a growing *open vocabulary* of entities. There are two ways we can link mention to entities: (1) link the mention before command generation in a *sequential* manner, or (2) link *jointly* with the decoding process and leverage the information from the previous decoder outputs.

Sequential Model This model attempts to first link the mention $m \in M_u$ of utterance u to entity e in entity set E_u, and only start the command generation process when we know which

[3]This is to exclude `new` from the mention history; a mention can be linked to `new` if it's not in the KB, but once it's created we take that new entity as an "extant" entity for this mention

entities are linked to which mentions. In particular, we want to maximize the likelihood

$$
\begin{aligned}
P(E_u, c|u, M_u) &= P(E_u|u, M_u)P(c|E_u; u, M_u) \\
&= \prod_{e \in E_u; m \in M_u}^{k} P(e|u, m)P(c|E_u; u, M_u) \\
&= \prod_{e \in E_u; m \in M_u}^{k} P(e|u, m) \prod_{t=1}^{|c|} P(c_t|c_{<t}; u, M_u; E_u)
\end{aligned}
$$

where e is the entity linked to mention m. At each decoding timestep t, we generate either relation-action pair $r{:}a$ [4] or a mention-entity pair $m{:}e$:

$$
\begin{aligned}
&P(e|u, m) \propto \exp(\text{score}(e, m)) \\
&P(c_t = r{:}a|c_{<t}; u, M_u, E_u) \propto \exp(\mathbf{W}^d_{ra}\mathbf{h}_t) \\
&P(c_t = m{:}e|c_{<t}; u, M_u, E_u) \propto \exp(\text{FFNN}_{me}([\mathbf{m}, \mathbf{v_g}(e)]) \cdot \mathbf{h}_t))
\end{aligned}
$$

where

- $\mathbf{W}^d_{ra}$ is the weight matrix for relation-action token $r{:}a$ during decoding
- $\mathbf{h}_t$ is the decoder hidden state at timestep t
- score(m, e) is the *sequential* entity scoring function (eq 3.1)

We can think of the score(m, e) as scoring the "matching goodness" between e and m. The "positional goodness" of pair $m{:}e$ at timestep t in the target command c must be scored at a later step in the generation process. We refer to this model as SequentialTransformer.

Joint model In this model (which we refer to as JointTransformer), we jointly link the entity with the decoding process, thus aiming to leverage the information from the previous decoder outputs to inform the linking process. Specifically, we want to maximize the likelihood

$$
P(E_u, c|u, M_u) = \prod_{t=1}^{|c|} P(E_u, c_t|c_{<t}; u, M_u)
$$

Similar to SequentialTransformer, at each time step t we generate either $r{:}a$ or $m{:}e$ as follows:

$$
\begin{aligned}
&P(c_t = r{:}a|c_{<t}; u, M_u) \propto \exp(\mathbf{W}^d_{ra}\mathbf{h}_t) \\
&P(c_t = m{:}e|c_{<t}; u, M_u) \propto \exp(\text{score}(m, e, \mathbf{h}_t))
\end{aligned}
$$

where score$(m, e, \mathbf{h}_t)$ is the *joint* entity scoring function (eq. 3.2), jointly scoring both the "matching goodness" between e and m, and the "positional goodness" of pair $m{:}e$ at timestep t of command c.

[4] Each r is associated with an action a, so we treat $r{:}a$ as a single token, which has the same cardinality as the vocabulary of relations

3.5 Entity Linking

The entity linking functions for both SequentialTransformer and JointTransformer share the same form, with the only difference being the integration of previous decoder hidden state $\mathbf{h}_t$ in the entity linking decision at timestep t for the joint model. Specifically, for SequentialTransformer, we have the entity linking function

$$\text{score}(m, e) = \mathbf{w}_l \cdot \text{FFNN}_l(\mathbf{v}(m, e)) \tag{3.1}$$

where $\mathbf{w}_l$ is the weight vector learned by the sequential entity linker, and $\mathbf{v}(m, e)$ is the feature vector defined below. For JointTransformer, we have

$$\text{score}(m, e, \mathbf{h}_t) = \mathbf{h}_t \cdot \text{FFNN}_l(\mathbf{v}(m, e)) \tag{3.2}$$

Feature vector $\mathbf{v}(m, e)$ is a key component of our scoring system, defined as:

$$\begin{aligned}\mathbf{v}(m, e) = [&\mathbf{m}, \phi(m, e), \mathbf{v_h}(e), \mathbf{v_g}(e), \\ &\mathbf{m} \circ \mathbf{v_h}(e), \mathbf{m} \circ \mathbf{v_g}(e)]\end{aligned} \tag{3.3}$$

with

- mention embedding $\mathbf{m}$ (Section 3.2)
- $\phi(m, e)$ is the distance (in number of entities linked) from the current mention m to the last time e was linked
- mention history $\mathbf{v_h}(e)$ and graph embedding $\mathbf{v_g}(e)$ of e (Section 3.3)

The feature vector and the entity linking function combine to learn the interactions between different components of m and e via element-wise product $\circ$ and feed-forward neural network FFNN_l.

3.6 Candidate Entities

For each mention, we consider a set of candidate entities to link to that mention. Since considering all the entities in the KB is prohibitively expensive, we develop a simple heuristic reduce the number of candidate entities: the candidate set includes pre-defined entities (eg `Entity`, `PhdStudent`, `True`, `False`, special entity `new` etc.), r number of most recently linked entities, and s number of entities in the KB with the most similar lexical form to the incoming mention, with similarity score computed by the modified Ratcliff and Obershelp algorithm, using Python's `SequenceMatcher` class of `difflib` library. [5]. During training, we include gold linked entities.

[5] https://docs.python.org/3/library/difflib.html

Chapter 4

Experiments and Analysis

4.1 Experimental Setup

Setup For both single story and multiple stories, we split the data sequentially into train, validation, and test sets (8:1:1). We use a 1-layer Transformer with a 256-unit feedforward network. The input to the encoder is the concatenation of 50-dimension GloVe embeddings, and 1024-dim ELMo embeddings. We use 200-dim embeddings for mention history $\mathbf{v_h}$, graph embedding $\mathbf{v_g}$, and mention embedding $\mathbf{m}$. For the Relational Graph Network, we employ a 2-layer graph convolution network with 50 hidden units. Hyperparameters are tuned with simple grid search on the validation set using the values in Table 4.1. We use the Adam optimizer [9] with a fixed learning rate of 10^{-4}. Each model is trained for 200 epochs, with 50 epochs of early stopping based on the validation exact match. We employ greedy decoding during inference.

Hyperparameters	Values
Learning rate	10^{-3}, 10^{-4}, 10^{-5}
Encoder-decoder hidden size	128, 256, 512
Encoder-decoder num layer	1, 2
Mention/Entity embedding dimension	100, 200, 300
Graph net hidden size	50, 100, 200

Table 4.1: Hyperparameter values

Single-utterance vs. multi-utterance There are two possible settings during inference: single-utterance and multi-utterances. In the single-utterance setting, we provide the gold KB before processing each utterance. In the multi-utterances setting, we process k utterances continuously, injecting gold KB only at the beginning of the sequence. Practically, single-utterance represents the case where the user corrects the system after every command, while in the multi-utterance setting, the user corrects the system after every k commands. We report the results on the single-utterance setting $k = 1$, and perform experiments on the effect of varying k in section 4.3.2

Evaluation metrics Our main evaluation criteria is *command exact match*, which measure if the predicted command exactly matches the gold command. This is a conservative metric, since a matching command will always yield a correct KB, but a nonmatching command may yield a correct KB under certain circumstances[1]. In addition, since the command contains extracted entities and relations, we further report the entity and relation F1 in order to gain insights into the effectiveness of our approach when extracting entities and relations.

4.2 Results

	Department		Meeting		Homework		Software		Average	
Single story	Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test
BaseTransformer	0.16	0.18	0.24	0.29	0.06	0.04	0.32	0.31	0.20	0.20
SequentialTransformer	**0.82**	0.77	**0.79**	**0.73**	**0.72**	**0.65**	**0.87**	0.73	**0.80**	**0.72**
JointTransformer	0.82	**0.79**	0.76	0.69	0.69	0.64	**0.87**	**0.78**	0.78	**0.72**
	Department		Meeting		Homework		Software		Average	
Multiple stories	Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test
BaseTransformer	0.07	0.06	0.12	0.07	0.08	0.02	0.11	0.16	0.09	0.08
SequentialTransformer	**0.63**	**0.62**	**0.60**	0.53	0.64	0.61	**0.56**	**0.62**	**0.61**	0.59
JointTransformer	0.61	0.61	**0.60**	**0.58**	**0.66**	**0.66**	0.54	0.57	0.60	**0.60**

Table 4.2: Exact match on validation and test data on single story (top) and multiple stories (bottom). We averaged the results over three separate runs with different seeds

	Single story		Multiple stories	
Metrics	Seq	Joint	Seq	Joint
Entity	89.8	89.1	76.2	76.5
Relation	99.2	99.5	97.3	93.4

Table 4.3: Average F1 scores on the val dataset of all the worlds, categorized by entity and relation metrics

Table 4.2 show our main results on both single story and multiple stories datasets. Both JointTransformer and SequentialTransformer expectedly outperform BaseTransformer. While SequentialTransformer has higher overall results, the difference between the results of the two models are not significant, showing that incorporating the previous outputs from the decoder during entity linking (for the JointTransformer) does not yield significant improvement.

[1]Consider commands with multiple relations. For some cases, if we switch the execution order of the relation-tail entity pairs, then we still get the correct KB (e.g. *"Both Andrew and Leo are TAs for U302"*). However, there are other cases where the order matters, such as *"Andrew was a second-year student, but is now a third-year student"*

We also report the average F1 scores of the entity and relation metrics on the validation datasets in Table 4.3. This results illustrates that our models perform well with generating the correct relation information, but still need improvement over entity linking. In addition, multiple stories datasets are harder to learn than single story datasets.

4.3 Analysis and Discussion

We analyze our results, both qualitatively and quantitatively. Our goal is to identify sources of error, as well as evaluate the strengths and weaknesses of our approach.

4.3.1 Error Analysis

	Single story		Multiple stories	
Error type	Seq	Joint	Seq	Joint
Entity-based	96%	97%	99%	99%
Relation-based	4%	2%	8%	6%
Command-based	19%	16%	10%	13%

Table 4.4: Percentage of incorrect commands based on error categories, average over all worlds. Note that a command can contain a combination of these errors

We report the percentage of errors, which are aggregated from all the experiments in section 4.2, in Table 4.4. We break the errors into three classes described below. Table 4.5 shows error examples.

Error type	Example utterances	Example gold and predicted commands
Entity-based	Alverta Mabee is now an assistant professor	*Gold*: Alverta_Mabee:alverta_mabee status:set assistant:assistant *Predicted*: Alverta_Mabee:alverta_depriest status:set assistant:assistant
Relation-based	Luanna currently has funding	*Gold*: Luanna:luanna_park has_funding:set ¡BOOL¿:True *Predicted*: Luanna:luanna_elvis funded:set ¡BOOL¿:True
Command-based	this student has Zenaida Pedraza and Alejandra Michael on the committee	*Gold*: this_student:eun_galbreath committee:add Zenaida_Pedraza:zenaida_pedraza committee:add Alejandra_Michael:alejandra_micha *Predicted*: this_student:eun_galbreath committee:add Alejandra_Michael:alejandra_michael

Table 4.5: Qualitative examples, based on error type. In this first utterance, the model incorrectly links *Alverta Mabee* to entity `alverta_depriest`. In the second utterance, the model incorrectly predicts relation `funded` instead of `has_funding`. It also made an entity-based error, erroneously mistaking entity `luanna_park` for `luanna_elvis`. In the last utterance, the model did not predict that *Zenaida Pedraza* is also on the committee.

Entity-based errors: These are the biggest sources of error, which happen either when a mention is linked to a wrong entity, or when the model predicts the incorrect position of the mention-entity pair within the command. Figure 4.1 shows the precision and recall for each category of the seperationTransformer model, for Department world, single story dataset. The worst-performing categories are `PhdStudent`, `Professor`, `MasterStudents`. As an example, a closer look at the distribution of predicted entities type `Professor` in Table 4.6 reveals that most of the errors come from mistaking entities within its own category (eg correct category linked but not correct entity itself, such as confusing a professor with another professor). This is further supported by looking at the t-SNE visualizations of the graph embeddings in Figure 4.2, where most of the entities are correctly clustered together. In summary, most of the entity-based errors come from mistaking entities within the same category (e.g. confusing a professor with another professor).

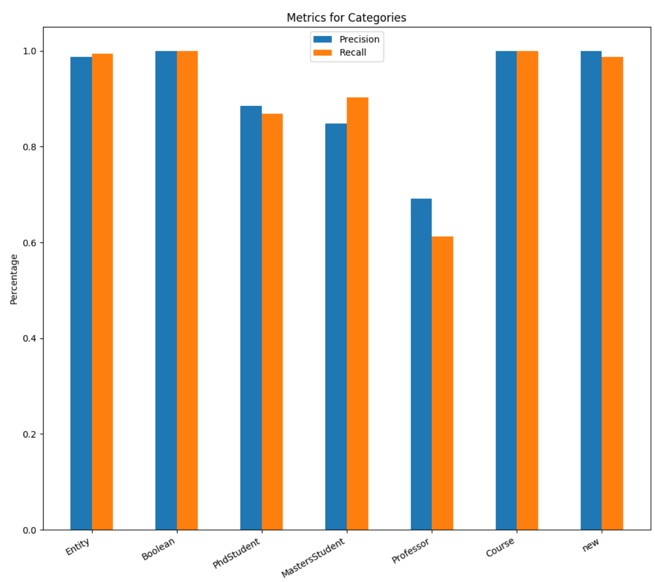


Figure 4.1: Precision and Recall for entities according to each category. Results from separationTransformer model on single story dataset, Department world

Statistics	Value
Correct entities linked	76 (58%)
Incorrect entities linked	53 (42%)
Wrong category (PhdStudent)	2 (4%)
Wrong category (MastersStudent)	2 (4%)
Wrong within category	49 (92%)

Table 4.6: Error statistics for entities of category `Professor`

We also categorize entity-based metrics into *entity-coref* F1, which reports the F1 over entities that are linked to an anaphoric mention, as opposed to *entity-nocoref* which reports the F1 over entities that are linked to their full lexical forms. Results in Table 4.7 shows that there is no

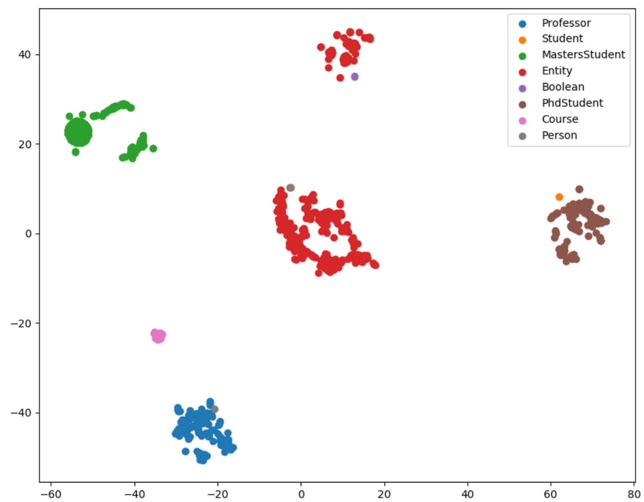


Figure 4.2: t-SNE visualization of graph embeddings for entities in the KB, clustered by category. Results after computed on the validation set, using best separationTransformer model on single story dataset, Department world

difference between linking anaphorical mentions (entity-coref) vs. linking entities' full lexical forms (entity-nocoref).

	single story		multiple stories	
Metrics	Sep	Joint	Sep	Joint
Entity-nocoref	90	89	81	80
Entity-coref	89	89	59	61
Relation	99	99	97	93

Table 4.7: Average F1 scores on the val dataset of all the worlds, categorized by entity and relation metrics

Relation-based error: These errors are extremely rare but do occur. For example, the model confuses the relations `funded` (for `PhdStudent` category) and `has_funding` (for `Professor` category). These relations have overlapping utterance semantics (e.g. *"that student currently has funding"* and *"this professor currently has funding"* are both valid utterances in our dataset), which confuses our neural models.

Command-based error: A closer look at the actual incorrect commands reveal that our models sometimes struggle with longer commands with multiple relations (Table 4.5).

4.3.2 Effect of k in Multi-utterances

We study the difference between the single-utterance and multi-utterances settings discussed in section 4.1 by varying the number of utterance k (Figure 4.3). We expect multi-utterances to be the harder setting, as errors will propagate from previous incorrect commands to affect subsequent generations (Table 4.8). We observe that for the single story dataset, the single-utterance case $k = 1$ yields the best exact match. This illustrates there is an advantage when the user corrects the system after every incorrect command. For $k > 1$ however, the value of k does not make a difference in performance. We hypothesize this is because similar errors are made across the dataset, regardless of where the corrections are in the sequence. For multiple stories datasets, there is essentially no difference when varying k, for similar reasons.

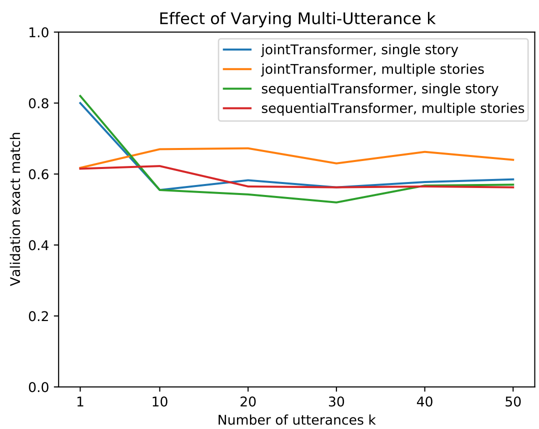


Figure 4.3: Effect of varying k in the multi-utterances setting

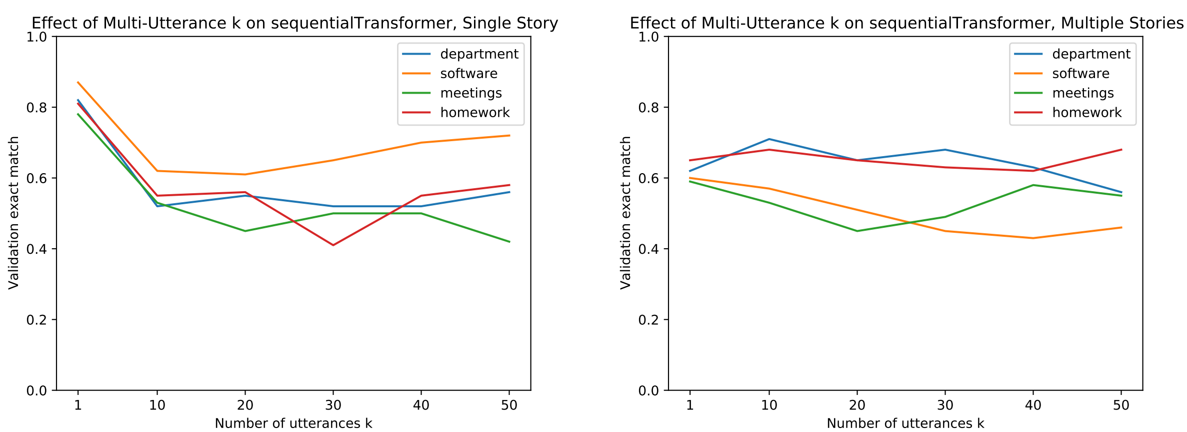


Figure 4.4: Effect of multi-utterance k vs val exact match for separationTransformer model, on single story (left) and multiple stories (right), breakdown down according to each world

Utterances	Commands for $k = 1$	Commands for $k = 10$
there is a new PhD student named Alejandra Galbreath	Alejandra_Galbreath:new generalizations:set PhD_student:PhdStudent	Alejandra_Galbreath:new generaliza PhD_student:luanna_elvis
Loyd Mabee's email is loyd_mabee@nh.edu	Loyd_Mabee's:loyd_mabee email:set loyd_mabee@nh.edu:new	Loyd_Mabee's:loyd_mabee email:se loyd_mabee@nh.edu:new
Alejandra Galbreath currently has funding	Alejandra_Galbreath:alejandra_galbreath funded:set ¡BOOL¿:True	Alejandra_Galbreath:hanna_galbreat ¡BOOL¿:True

Table 4.8: Three consecutive utterances that illustrate the error propagation for $k > 1$. For $k = 10$ (right), the first command is incorrectly generated, thus no entity `alejandra_galbreath` is created. This effects the third utterance, where $k = 10$ case incorrectly link the mention *Alejandra Galbreath* to `hanna_galbreath`, possibly due to `alejandra_galbreath` not existed in the KB. Contrast this to $k = 1$ case, where the gold KB is available at every utterance, and it was able to link *Alejandra Galbreath* to `alejandra_galbreath`

4.3.3 Ablation Studies

A key component of our model is the entity linker that produces the scores for linking a mention to entities in the KG. We study the importance of the linking feature vector in equation 3.3 by ablating its features and report the Exact Match (Ex. Mat.) performance in Table 4.9. We observe that all features are important in improving the performance of the model. Predictably, eliminating both the mention history vector $\mathbf{v_h}$ and the graph embedding $\mathbf{v_g}$ components reduces the performance the most. We also note that the ablation effects are more visible for the single story dataset, compared to the multiple stories (Tables 4.10 and 4.11).

Model	Ex. Mat.	$\Delta(\%)$
JointTransformer	0.80	
− distance feature ϕ	0.66	-0.14 (18%)
− mention history $\mathbf{v_h}$	0.68	-0.12 (15%)
− graph embedding $\mathbf{v_g}$	0.69	-0.11 (14%)
− both $\mathbf{v_h}$, $\mathbf{v_g}$	0.57	-0.23 (29%)

Table 4.9: Ablations on the entity linking features, done with the JointTransformer model, single story dataset, averaging on the validation set of all worlds

Model	Department	Meeting	Homework	Software	Average
jointTransformer	0.81	0.76	0.74	0.89	0.8
− distance feature ϕ	0.67	0.56	0.69	0.72	0.66 (-18%)
− mention history v_h	0.70	0.50	0.70	0.80	0.68 (-15%)
− graph embedding v_g	0.72	0.51	0.68	0.85	0.69 (-14%)
− entity embeddings (both v_h, v_g)	0.58	0.44	0.54	0.72	0.57 (-29%)

Table 4.10: Feature ablations, single story

Model	Department	Meeting	Homework	Software	Average
jointTransformer	0.62	0.61	0.67	0.59	0.62
− distance feature ϕ	0.59	0.60	0.65	0.57	0.60 (-3%)
− mention history v_h	0.58	0.54	0.65	0.54	0.58 (-6%)
− graph embedding v_g	0.57	0.57	0.65	0.54	0.58 (-6%)
− entity embeddings (both v_h, v_g)	0.57	0.5	0.49	0.51	0.52 (-16%)

Table 4.11: Feature ablations, multiple stories

Chapter 5

Related Work

Learning from Dialogs and User Interactions Recent years have seen studies on NLP systems that learn from the end-user via conversational dialog [6, 15, 27] and natural language interactions [12, 18, 23, 24]. While these works open the door to building conversational assistants that can learn from the user, most do not attempt to build a personalized, user-centric KB to be used for downstream personal tasks or the current task itself. [8] and [12] build a KB from extracted knowledge with a certain degree of success, but their information extraction systems did not take advantage of the representational power that neural models offer in modeling the extracted knowledge.

Knowledge Base Construction Most works in this area focus on KB construction over unstructured text extraction on the Web [2, 5] or commonsense reasoning [1, 20, 22]. Similar to our work is [4], where they build a Machine Reading Comprehension model that constructs dynamic knowledge graphs to track state changes in procedural text. However, their KG is limited to two node types with one edge (relation) type denoting the binary location change. In contrast, our KGs contains a diverse set of entity and relation types.

Neural Open Information Extraction Our work is formulated as a sequence-generation based Neural Open Information Extraction problem, similar to [3, 11]. Our work differs in that we (1) involve the KB in the extraction process by embedding it, and (2) perform entity linking on the constructed KB. The advantage of formulating this task as sequence generation instead of as sequence-labelling based information extraction [19, 25] is the possibility of extending the target language to more complex utterances. This formulation easily bridges the problem into that of semantic parsing, from which we can use the extensive results from the semantic parsing literature.

Chapter 6

Conclusion and Future Work

We study the task of user-instructed Knowledge Base construction by formulating it as a sequence-generation problem: learning how to map the natural language instructions to target command than can be executed to construct the KB. We build Transformer-based encoder-decoder models that integrate the structured context from the evolving KB with the unstructured context from the utterance to aid command generation. Our models, JointTransformer and SequentialTransformer, perform well when extracting relations, while also allow for linking of entities created during test time. However, they still struggle with linking entities within the same category and in more challenging settings (multiple stories data and multi-utterances).

Currently, this work has several simplifying assumptions, thus allowing for various future directions. While our models support open vocabulary of entities, it does not support out-of-vocabulary relations, restricting the ability to generalize to domains where the model is not trained on. In addition, our models assume the the entity mentions are provided *a priori*. Removing this assumption would add the problem of *Mention Detection* to the task, which is an interesting end-to-end setting to investigate. Finally, we hope to obtain and evaluate our models on more challenging datasets that include more variety of utterances as well as implicit knowledge that requires commonsense reasoning.

Bibliography

[1] Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. COMET: Commonsense transformers for automatic knowledge graph construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1470. URL https://www.aclweb.org/anthology/P19-1470. 1, 5

[2] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr., and T.M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pages 1306–1313. AAAI Press, 2010. 1, 5

[3] Lei Cui, Furu Wei, and Ming Zhou. Neural open information extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 407–413, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2065. URL https://www.aclweb.org/anthology/P18-2065. 2.1, 5

[4] Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. *CoRR*, abs/1810.05682, 2018. URL http://arxiv.org/abs/1810.05682. 5

[5] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014. URL https://cs.cmu.edu/~nlao/publication/2014.kdd.pdf. 1, 5

[6] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. Learning from dialogue after deployment: Feed yourself, chatbot! In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3667–3684, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1358. URL https://www.aclweb.org/anthology/P19-1358. 5

[7] He He, Anusha Balakrishnan, Mihail Eric, and Percy Liang. Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1766–1776, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1162. URL https://www.aclweb.org/anthology/P17-1162. 3.3, 3.3

[8] Ben Hixon, Peter Clark, and Hannaneh Hajishirzi. Learning knowledge graphs for question answering through conversational dialog. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 851–861, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1086. URL https://www.aclweb.org/anthology/N15-1086. 5

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 4.1

[10] Nikolaos Kolitsas, Octavian-Eugen Ganea, and Thomas Hofmann. End-to-end neural entity linking. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 519–529, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/K18-1050. URL https://www.aclweb.org/anthology/K18-1050. 3.2

[11] Keshav Kolluru, Samarth Aggarwal, Vipul Rathore, Mausam, and Soumen Chakrabarti. IMoJIE: Iterative memory-based joint open information extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5871–5886, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.521. URL https://www.aclweb.org/anthology/2020.acl-main.521. 5

[12] Igor Labutov, Shashank Srivastava, and Tom Mitchell. LIA: A natural language programmable personal assistant. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 145–150, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2025. URL https://www.aclweb.org/anthology/D18-2025. 1, 5

[13] Igor Labutov, Bishan Yang, Anusha Prakash, and Amos Azaria. Multi-relational question answering from narratives: Machine reading and reasoning in simulated worlds. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 833–844, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1077. URL https://www.aclweb.org/anthology/P18-1077. 1, 2.5.1

[14] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1018. URL https://www.aclweb.org/anthology/D17-1018. 2.3

[15] Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. Dialogue learning with human-in-the-loop. *CoRR*, abs/1611.09823, 2016. URL http://arxiv.org/abs/1611.09823. 5

[16] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors

for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://www.aclweb.org/anthology/D14-1162. 3.1

[17] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://www.aclweb.org/anthology/N18-1202. 3.1

[18] Sudha Rao and Hal Daumé III. Answer-based Adversarial Training for Generating Clarification Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 143–155, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1013. URL https://www.aclweb.org/anthology/N19-1013. 5

[19] Arpita Roy, Youngja Park, Taesung Lee, and Shimei Pan. Supervising unsupervised open information extraction models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 728–737, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1067. URL https://www.aclweb.org/anthology/D19-1067. 5

[20] Maarten Sap, Ronan LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. ATOMIC: an atlas of machine commonsense for if-then reasoning. *CoRR*, abs/1811.00146, 2018. URL http://arxiv.org/abs/1811.00146. 5

[21] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 593–607. Springer/Verlag, 2018. ISBN 9783319934167. doi: 10.1007/978-3-319-93417-4_38. 15th International Conference on Extended Semantic Web Conference, ESWC 2018 ; Conference date: 03-06-2018 Through 07-06-2018. 3.3

[22] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *CoRR*, abs/1612.03975, 2016. URL http://arxiv.org/abs/1612.03975. 5

[23] Shashank Srivastava, Igor Labutov, and Tom Mitchell. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1527–1536, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1161. URL https://www.aclweb.org/anthology/D17-1161. 5

[24] Shashank Srivastava, Igor Labutov, and Tom Mitchell. Learning to ask for conversational machine learning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4164–4174, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1426. URL https://www.aclweb.org/anthology/D19-1426. 5

[25] Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. Supervised open information extraction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 885–895, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1081. URL https://www.aclweb.org/anthology/N18-1081. 5

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. 3, 3.1

[27] Jason E Weston. Dialog-based language learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 829–837. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf. 5